

**DEVELOPMENT OF AN UNDERGRADUATE
MICROPROCESSOR TEACHING FACILITY**

VOLUME 2

by
JAMES GRAHAM JACK
B.Sc. (Electrical) Engineering (Cape Town)

**Submitted to the University of Cape Town in fulfillment of
the requirements for the degree of Master of Science in
Engineering.**

April 1988

The University of Cape Town has been given
the right to reproduce this thesis in whole
or in part. Copyright is held by the author.

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

The following is a table of contents for the last six appendices
that are contained in this second volume.

<u>CONTENTS</u>	<u>Page</u>
F. EDT Software listing	
F.1 Symbol index	F-1
F.2 Program listing	F-8
G. ASM Software listing	
G.1 Symbol index	G-1
G.2 Program listing	G-9
H. DEB Software listing	
H.1 Symbol index	H-1
H.2 Program listing	H-14
I. DIAG Software listing	
I.1 Symbol index	I-1
I.2 Build command file	I-8
I.3 Program listing	I-11
J. LINK Software listing	
J.1 Symbol index	J-1
J.2 Build command file	J-5
J.2 Program listing	J-9
K. KIT Software listing	
K.1 Build command file	K-1
K.2 Program listing	K-2

APPENDIX F

EDT LISTING

The following pages contain an index of the symbols used in the editor source listing. In order to keep the index reasonably short, the local symbols, those defined and referenced within a subroutine, are not listed.

As well as the symbol name and page reference, a symbol type is also given. The possible symbol types are:

- a) CONSTANT (in this case a value is also given)
- b) SUBROUTINE
- c) BYTE ARRAY (in this case the length is also given)
- d) WORD ARRAY (in this case the length, in bytes, is also given)
- e) STRUCTURE
- f) TABLE
- g) MESSAGE

Some of the symbol types may be preceded by the word ABSOLUTE. This simply means that they are stored at fixed addresses in the memory map and storage is not dynamically allocated in the editor data area.

NOTE: The page reference given in the index below is the page on which the symbol is defined.

<u>SYMBOL</u>	<u>DESCRIPTION</u>	<u>PAGE REFERENCE</u>
ACK	CONSTANT, VALUE = '@'.....	F-11
ASCBIN	CONSTANT, VALUE = 0F031H.....	F-13
ASCII	CONSTANT, VALUE = 01111111B.....	F-9
BBUF	SUBROUTINE.....	F-21
BELL	CONSTANT, VALUE = 07H.....	F-10
BLINE	SUBROUTINE.....	F-131
BLROU	SUBROUTINE.....	F-111
BOTLIN	CONSTANT, VALUE = 23.....	F-11
BREAK	CONSTANT, VALUE = 0F025H.....	F-13
BRROU	SUBROUTINE.....	F-111
BSPACE	CONSTANT, VALUE = 08H.....	F-10

BUF	BYTE ARRAY, LENGTH = STRLEN.....	F-158
BYTE	CONSTANT, VALUE = 1.....	F-9
CCMOVE	SUBROUTINE.....	F-37
CCMOVT	TABLE.....	F-143
CHAR	WORD VARIABLE.....	F-157
CHKFN	SUBROUTINE.....	F-106
CI	CONSTANT, VALUE = 0F013H.....	F-13
CITEST	CONSTANT, VALUE = 0F00AH.....	F-13
CLRL	SUBROUTINE.....	F-131
CLRLC	CONSTANT, VALUE = 'K'.....	F-9
CLRS	SUBROUTINE.....	F-131
CLRSC	CONSTANT, VALUE = 'J'.....	F-9
CMTAB	TABLE.....	F-143
CO	CONSTANT, VALUE = 0F01CH.....	F-13
COL	BYTE VARIABLE.....	F-157
COM	BYTE VARIABLE.....	F-158
COMTAB	TABLE.....	F-144
CONCAP	SUBROUTINE.....	F-22
CONFRM	MESSAGE.....	F-151
CONIN	CONSTANT, VALUE = CI.....	F-10
CONOUT	CONSTANT, VALUE = CO.....	F-10
CONDO	CONSTANT, VALUE = CONTROL AND 'C'.....	F-10
CONTEST	CONSTANT, VALUE = CITEST.....	F-10
CONTLE	CONSTANT, VALUE = CONTROL AND 'D'.....	F-10
CONTRI	CONSTANT, VALUE = CONTROL AND 'F'.....	F-10
CONTROL	CONSTANT, VALUE = 00011111B.....	F-9
CONTUP	CONSTANT, VALUE = CONTROL AND 'R'.....	F-10
COPY	SUBROUTINE.....	F-21
CR	CONSTANT, VALUE = 0DH.....	F-11
CTRLC	CONSTANT, VALUE = CONTROL AND 'C'.....	F-11
CTRLK	CONSTANT, VALUE = CONTROL AND 'K'.....	F-11
CTRLQ	CONSTANT, VALUE = CONTROL AND 'Q'.....	F-11
CTRLR	CONSTANT, VALUE = CONTROL AND 'R'.....	F-11
CTRLS	CONSTANT, VALUE = CONTROL AND 'S'.....	F-11
CTRLU	CONSTANT, VALUE = CONTROL AND 'U'.....	F-11
CTRLZ	CONSTANT, VALUE = CONTROL AND 'Z'.....	F-11
CUTBL	WORD VARIABLE.....	F-158
CUTBUF	BYTE VARIABLE.....	F-158
CUTEM1	MESSAGE.....	F-153
CUTEM2	MESSAGE.....	F-153
CUTLIB	WORD VARIABLE.....	F-158
CUTLM1	MESSAGE.....	F-154
CUTLM2	MESSAGE.....	F-154
CUTLM3	MESSAGE.....	F-154
CUTLRO	SUBROUTINE.....	F-91
CUTROU	SUBROUTINE.....	F-88
DATA	CONSTANT, VALUE = 0E800H.....	F-11
DBLEN	CONSTANT, VALUE = 4.....	F-11
DCHARS	SUBROUTINE.....	F-72
DCMP	CONSTANT, VALUE = 0F043H.....	F-13
DCMPI	CONSTANT, VALUE = 0F046H.....	F-13
DECMUL	SUBROUTINE.....	F-127
DEFEN	CONSTANT, VALUE = 2.....	F-145
DEFERR	MESSAGE.....	F-154
DEFINED	SUBROUTINE.....	F-136

DEFMES	MESSAGE.....	F-154
DEFROU	SUBROUTINE.....	F-137
DEFTAB	STRUCTURE (1).....	F-158
DEL	CONSTANT, VALUE = 7FH.....	F-11
DELAY	SUBROUTINE.....	F-106
DELCRO	SUBROUTINE.....	F-73
DELETE	SUBROUTINE.....	F-73
DELROU	SUBROUTINE.....	F-75
DERRM1	MESSAGE.....	F-152
DERRM2	MESSAGE.....	F-152
DIGBUF	BYTE ARRAY, LENGTH = DBLEN.....	F-158
DLINES	SUBROUTINE.....	F-76
DOSUB	SUBROUTINE.....	F-61
DOWNC	CONSTANT, VALUE = 'B'.....	F-9
DOWNRO	SUBROUTINE.....	F-84
DTEOLR	SUBROUTINE.....	F-78
ECHO	CONSTANT, VALUE = 0F034H.....	F-13
EDTINI	SUBROUTINE.....	F-23
ELROU	SUBROUTINE.....	F-112
EOBLIN	BYTE VARIABLE.....	F-157
EOBMES	MESSAGE.....	F-151
EOFMARK	CONSTANT, VALUE = CTRLZ.....	F-11
EOT	CONSTANT, VALUE = 04H.....	F-10
EPOINT	ABSOLUTE WORD VARIABLE.....	F-159
ERR	CONSTANT, VALUE = 0FFH.....	F-9
ERRCCS	MESSAGE.....	F-151
ERROU	SUBROUTINE.....	F-112
ERRSTR	SUBROUTINE.....	F-115
ESC	CONSTANT, VALUE = 1BH.....	F-11
ESCFLG	BYTE VARIABLE.....	F-157
ETABLE	CONSTANT, VALUE = 0FFH.....	F-11
EXITRO	SUBROUTINE.....	F-48
FALSE	CONSTANT, VALUE = 0.....	F-9
FDCMP	SUBROUTINE.....	F-24
FDNOTE	MESSAGE.....	F-155
FETCHD	SUBROUTINE.....	F-137
FILEIO	SUBROUTINE.....	F-99
FINDRO	SUBROUTINE.....	F-54
FLMESS	MESSAGE.....	F-155
FNCRC	ABSOLUTE BYTE VARIABLE.....	F-159
FSMESS	MESSAGE.....	F-155
GET1NM	CONSTANT, VALUE = 0F055H.....	F-13
GET2NM	CONSTANT, VALUE = 0F058H.....	F-13
GET3NM	CONSTANT, VALUE = 0F05BH.....	F-13
GETCM	CONSTANT, VALUE = 0F04FH.....	F-13
GETECHO	CONSTANT, VALUE = 0F02BH.....	F-13
GETHX	CONSTANT, VALUE = 0F052H.....	F-13
GETKEY	SUBROUTINE.....	F-115
GETPDP	CONSTANT, VALUE = UI0.....	F-13
GETUC	CONSTANT, VALUE = 0F028H.....	F-13
GETWD	CONSTANT, VALUE = 0F02EH.....	F-13
GOBACK	SUBROUTINE.....	F-24
GOTO	SUBROUTINE.....	F-132
GOTOC	CONSTANT, VALUE = 'Y'.....	F-9
HELPMES	TABLE.....	F-148

HELPRO	SUBROUTINE.....	F-147
HOME	SUBROUTINE.....	F-132
HOMECL	CONSTANT, VALUE = 'H'.....	F-9
INCLIN	SUBROUTINE.....	F-25
INCNU	SUBROUTINE.....	F-78
INCPTR	SUBROUTINE.....	F-116
INITDT	SUBROUTINE.....	F-25
INSERT	SUBROUTINE.....	F-95
INSFLG	BYTE VARIABLE.....	F-157
INVFN	MESSAGE.....	F-155
IPAD	SUBROUTINE.....	F-42
IPADLEN	CONSTANT, VALUE = 20.....	F-11
IPROMPT	CONSTANT, VALUE = '<'.....	F-11
IROU	SUBROUTINE.....	F-95
IVCMES	MESSAGE.....	F-151
K	CONSTANT, VALUE = 1024.....	F-9
KBLN	CONSTANT, VALUE = 120.....	F-11
KDTAB	STRUCTURE (2).....	F-158
KEYBUF	BYTE ARRAY, LENGTH = KBLN.....	F-158
KEYIN	SUBROUTINE.....	F-116
KEYNUM	BYTE VARIABLE.....	F-158
KEYSTK	SUBROUTINE.....	F-117
KILL1C	SUBROUTINE.....	F-138
KILLC	SUBROUTINE.....	F-138
LCLINE	BYTE VARIABLE.....	F-157
LEFTC	CONSTANT, VALUE = 'D'.....	F-9
LEFTRO	SUBROUTINE.....	F-86
LENROU	SUBROUTINE.....	F-50
LF	CONSTANT, VALUE = 0AH.....	F-11
LFORLF	SUBROUTINE.....	F-63
LINE	BYTE VARIABLE.....	F-157
LINROU	SUBROUTINE.....	F-81
LM1	MESSAGE.....	F-151
LM2	MESSAGE.....	F-151
LM3	MESSAGE.....	F-151
LM4	MESSAGE.....	F-151
LNKEM	MESSAGE.....	F-155
LO	BYTE VARIABLE.....	F-158
LOADEN	CONSTANT, VALUE = 0.....	F-145
LOADF	SUBROUTINE.....	F-101
LOADRO	SUBROUTINE.....	F-99
LOOKFS	SUBROUTINE.....	F-55
LPRINT	SUBROUTINE.....	F-118
LSTLIN	CONSTANT, VALUE = 22.....	F-11
MAKECOM	SUBROUTINE.....	F-102
MASK	CONSTANT, VALUE = 00011111B.....	F-12
MAXCL	CONSTANT, VALUE = 10.....	F-11
MAXCOL	CONSTANT, VALUE = 79.....	F-11
MAXLEN	CONSTANT, VALUE = 44*K.....	F-11
MAXMOV	CONSTANT, VALUE = 40.....	F-12
MAXTXT	WORD VARIABLE.....	F-158
MEMFUL	MESSAGE.....	F-151
MINCOL	CONSTANT, VALUE = 0.....	F-11
MINIT	SUBROUTINE.....	F-25
MODCCH	CONSTANT, VALUE = CONTROL AND 'Z'.....	F-10

MODCOL	SUBROUTINE.....	F-40
MONDATA	CONSTANT, VALUE = 0EF00H.....	F-11
MOVCUR	SUBROUTINE.....	F-36
MOVEBL	SUBROUTINE.....	F-82
MOVENL	SUBROUTINE.....	F-81
MOVL	SUBROUTINE.....	F-132
MOVR	SUBROUTINE.....	F-133
MOVSTR	SUBROUTINE.....	F-56
NBPACK	CONSTANT, VALUE = 10.....	F-12
NEWROU	SUBROUTINE.....	F-52
NEWSCN	SUBROUTINE.....	F-26
NEWSF	BYTE VARIABLE.....	F-157
NEWSP	STRUCTURE (3).....	F-158
NFINDS	WORD VARIABLE.....	F-158
NFNAMM	MESSAGE.....	F-155
NLINE	SUBROUTINE.....	F-133
NMOUT	CONSTANT, VALUE = 0F03AH.....	F-13
NOSEND	BYTE VARIABLE.....	F-157
NOSTRM	MESSAGE.....	F-151
NOSUCH	MESSAGE.....	F-152
NULL	CONSTANT, VALUE = 00H.....	F-10
NULLS	BYTE VARIABLE.....	F-157
NULLUP	SUBROUTINE.....	F-79
OCOL	BYTE VARIABLE.....	F-157
OFFSET	CONSTANT, VALUE = 32.....	F-12
OK	CONSTANT, VALUE = 000H.....	F-9
OLINE	BYTE VARIABLE.....	F-157
ONSCN	SUBROUTINE.....	F-26
OUTMSG	CONSTANT, VALUE = 0F03DH.....	F-13
PACK	SUBROUTINE.....	F-45
PAD	SUBROUTINE.....	F-42
PASEM1	MESSAGE.....	F-154
PASEM2	MESSAGE.....	F-154
PASTER	SUBROUTINE.....	F-92
PBELL	SUBROUTINE.....	F-119
PCUR	SUBROUTINE.....	F-40
PDPCOM	MESSAGE.....	F-155
PDPEM	MESSAGE.....	F-155
PLINE	SUBROUTINE.....	F-119
POINT	WORD VARIABLE.....	F-157
PRINT	SUBROUTINE.....	F-120
PRINTD	SUBROUTINE.....	F-127
PROG	CONSTANT, VALUE = 01000H.....	F-11
PRVAL	CONSTANT, VALUE = 0F037H.....	F-13
PTOEOL	SUBROUTINE.....	F-121
PUTD	SUBROUTINE.....	F-139
PUTKD	SUBROUTINE.....	F-139
PWR10	TABLE.....	F-128
PZRFLG	BYTE VARIABLE.....	F-157
QCHAR	SUBROUTINE.....	F-38
QCHAROK	SUBROUTINE.....	F-109
QCLINE	SUBROUTINE.....	F-27
QCURM	SUBROUTINE.....	F-36
QEOB	SUBROUTINE.....	F-27
QFILN1	MESSAGE.....	F-155

QFILN2	MESSAGE.....	F-155
QFINDR	SUBROUTINE.....	F-56
QINIT	SUBROUTINE.....	F-28
QKILLC	CONSTANT, VALUE = CTRLK.....	F-11
QLET	SUBROUTINE.....	F-28
QNULLF	SUBROUTINE.....	F-113
QNUM	SUBROUTINE.....	F-29
QSOB	SUBROUTINE.....	F-29
QTAB	SUBROUTINE.....	F-30
QUITRO	SUBROUTINE.....	F-48
QUOTE	CONSTANT, VALUE = ' '.....	F-11
RBULF	SUBROUTINE.....	F-31
READKD	SUBROUTINE.....	F-140
READNU	SUBROUTINE.....	F-128
READS	SUBROUTINE.....	F-121
RELCH	BYTE VARIABLE.....	F-157
RELEASE	CONSTANT, VALUE = 0.....	F-9
REPBUF	BYTE ARRAY, LENGTH = 30.....	F-158
REPSL	BYTE VARIABLE.....	F-158
RESCUR	SUBROUTINE.....	F-123
RESFLG	BYTE VARIABLE.....	F-157
REVISION	CONSTANT, VALUE = 1.....	F-9
REVLf	SUBROUTINE.....	F-133
REVLFC	CONSTANT, VALUE = 'I'.....	F-9
RFILE	SUBROUTINE.....	F-104
RFILMF	MESSAGE.....	F-155
RIGHTC	CONSTANT, VALUE = 'C'.....	F-9
RIGHTR	SUBROUTINE.....	F-85
RUBOUT	CONSTANT, VALUE = 08H.....	F-10
RULF	SUBROUTINE.....	F-31
SAVEEN	CONSTANT, VALUE = 1.....	F-145
SAVEF	SUBROUTINE.....	F-105
SAVERO	SUBROUTINE.....	F-99
SCPRESS	SUBROUTINE.....	F-124
SCTPDP	SUBROUTINE.....	F-103
SEABUF	BYTE ARRAY, LENGTH = 30.....	F-158
SEAFS	SUBROUTINE.....	F-57
SEARCH	SUBROUTINE.....	F-37
SEASL	BYTE VARIABLE.....	F-158
SETMODE	SUBROUTINE.....	F-32
SETNSF	SUBROUTINE.....	F-63
SMMSG	MESSAGE.....	F-153
SNDCOM	SUBROUTINE.....	F-134
SNDPDP	CONSTANT, VALUE = U00.....	F-13
SOFN	ABSOLUTE BYTE ARRAY, LENGTH = 30.....	F-159
SPACE	CONSTANT, VALUE = ' '.....	F-11
SPDPWE	SUBROUTINE.....	F-108
SPOINT	ABSOLUTE WORD VARIABLE.....	F-159
STACKC	SUBROUTINE.....	F-124
START	MAIN PROGRAM.....	F-14
STKLEN	CONSTANT, VALUE = 100 * WORD.....	F-11
STRCMP	SUBROUTINE.....	F-59
STRCOM	TABLE.....	F-145
STRER1	MESSAGE.....	F-151
STRING	SUBROUTINE.....	F-125

STRLEN	CONSTANT, VALUE = 80.....	F-11
SUB16	SUBROUTINE.....	F-93
SUBBIT	BYTE VARIABLE.....	F-157
SUBEM1	MESSAGE.....	F-152
SUBEM2	MESSAGE.....	F-152
SUBEM4	MESSAGE.....	F-152
SUBEM5	MESSAGE.....	F-152
SUBEM6	MESSAGE.....	F-152
SUBEM7	MESSAGE.....	F-153
SUBM8A	MESSAGE.....	F-153
SUBM8B	MESSAGE.....	F-153
SUBREP	BYTE VARIABLE.....	F-157
SUBROU	SUBROUTINE.....	F-65
TAB	CONSTANT, VALUE = 09H.....	F-11
TCOMBU	BYTE ARRAY, LENGTH = MAXCL.....	F-158
TIMOEM	MESSAGE.....	F-155
TOCONST	CONSTANT, VALUE = 0FFFFH.....	F-11
TOPLIN	CONSTANT, VALUE = 0.....	F-11
TPADL	WORD VARIABLE.....	F-157
TRANSP	CONSTANT, VALUE = 0F049H.....	F-13
TRUE	CONSTANT, VALUE = NOT FALSE.....	F-9
U0TEST	CONSTANT, VALUE = 0F00DH.....	F-13
U1TEST	CONSTANT, VALUE = 0F010H.....	F-13
UI0	CONSTANT, VALUE = 0F016H.....	F-13
UI1	CONSTANT, VALUE = 0F019H.....	F-13
UNDEFK	MESSAGE.....	F-154
U00	CONSTANT, VALUE = 0F01FH.....	F-13
U01	CONSTANT, VALUE = 0F022H.....	F-13
UP	BYTE VARIABLE.....	F-158
UPC	CONSTANT, VALUE = 'A'.....	F-9
UPDATE	SUBROUTINE.....	F-33
UPROU	SUBROUTINE.....	F-83
VALDL	CONSTANT, VALUE = 0F04CH.....	F-13
VERMSG	MESSAGE.....	F-154
VERROU	SUBROUTINE.....	F-51
VERSION	CONSTANT, VALUE = 1.....	F-9
VERTP	SUBROUTINE.....	F-39
WAITFA	SUBROUTINE.....	F-107
WAITFU	SUBROUTINE.....	F-108
WORD	CONSTANT, VALUE = 2.....	F-9
XCHAR	SUBROUTINE.....	F-34
XFLG	BYTE VARIABLE.....	F-157
XPROMPT	CONSTANT, VALUE = '^'.....	F-11

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$TITLE('UCT SABus editor')
		2	\$PAGELENGTH(75)
		3	;
		4	-----
		5	;
		6	UCT SABus editor
		7	-----
		8	;
		9	Language : 8085 Assembler.
		10	Author : J.G.Jack
		11	Date : 1983.
		12	;
		13	Main file : EDT.A85
		14	;
		15	The following files are included at assembly time, they could not be
		16	assembled and linked independently as we did not have a linker.
		17	;
		18	;
		19	1) : EDTEQU.A85 - Editor equates: ASCII characters, monitor
		20	routines, table sizes etc.
		21	2) : EDTCURSOR.A85 - Routines used for cursor positioning.
		22	3) : EDTPAD.A85 - The editor PAD routine, used to extend
		23	the file when inserting.
		24	4) : EDTPACK.A85 - The editor PACK routine, used to remove
		25	all NULLS from the file.
		26	5) : EDTEXIT.A85 - Editor exit point.
		27	6) : EDTLEN.A85 - Displays length of file in memory and
		28	the amount of space left.
		29	7) : EDTNEW.A85 - Clears the file in memory.
		30	8) : EDTSUB.A85 - The editor substitute and find routines.
		31	9) : EDTDELOU.A85 - Delete routines, used to delete single
		32	characters, a number of characters,
		33	single lines and a number of lines.
		34	10) : EDTMOVROU.A85 - Routines used to move the cursor around
		35	called when cursor keys typed.
		36	11) : EDTCUTPAS.A85 - Editor CUT and PASTE routines.
		37	12) : EDTINSERT.A85 - Editor insert routine.
		38	13) : EDTFILEIO.A85 - Editor file I/O routines, used to LOAD/
		39	SAVE files from/to the PDP-11/23.
		40	14) : EDTRANGE.A85 - Routines to handle the following 4
		41	commands: BL,EL,BR and ER.
		42	15) : EDTIO.A85 - Editor I/O routines.
		43	16) : EDTDEC.A85 - Editor decimal routines.
		44	17) : EDTESC.A85 - Editor ESC sequence routines.
		45	18) : EDTDEFINE.A85 - Editor key definition and execution
		46	routines.
		47	19) : EDTTABLE.A85 - Editor table search routines.
		48	20) : EDTMES.A85 - Editor messages.
		49	21) : EDTHELP.A85 - Editor help routine and message.
		50	22) : EDTDATA.A85 - Editor data area.
		51	23) : EDTGEN.A85 - General routines.
		52	24) : EDTFIND.A85 - Editor find routines.
		53	;
		54	-----
		55	;
		56	NAME EDT
		57	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 2

LOC	OBJ	LINE	SOURCE STATEMENT
		58 ;	
		59 ;	Include all equates used by editor.
		60 ;	
		61 \$INCLUDE(EDTEQU.A85)	
		= 62 ;=====;	
		= 63 ;	
		= 64 ;	Equate module for the SABus kit editor.
		= 65 ;	
		= 66 ;=====;	
		= 67 ;	
		= 68 ;	a : Version information:
		= 69 ;	
0001		= 70 VERSION EQU 1	; Version number
0001		= 71 REVISION EQU 1	; Revision number
0000		= 72 RELEASE EQU 0	; Release number
		= 73 ;	
		= 74 ;-----	
		= 75 ;	
		= 76 ;	b : Logic values:
		= 77 ;	
0000		= 78 FALSE EQU 000H	; Zero = Logic false.
FFFF		= 79 TRUE EQU NOT FALSE	; \$FF = Logic true.
		= 80 ;	
		= 81 ;-----	
		= 82 ;	
		= 83 ;	c : Variable sizes:
		= 84 ;	
0002		= 85 WORD EQU 2	; Length of a word (in bytes).
0001		= 86 BYTE EQU 1	; Length of a byte variable.
0400		= 87 K EQU 1024	; 1K = 1024 Bytes.
		= 88 ;	
		= 89 ;-----	
		= 90 ;	
		= 91 ;	d : Code returned by subroutines:
		= 92 ;	
0000		= 93 OK EQU 000H	; Zero = Status OK!
00FF		= 94 ERR EQU OFFH	; Non zero = Failed.
		= 95 ;	
		= 96 ;-----	
		= 97 ;	
		= 98 ;	e : Conversion masks:
		= 99 ;	
001F		= 100 CONTROL EQU 00011111B	; Mask to give control codes.
007F		= 101 ASCII EQU 01111111B	; 7 bit ASCII mask.
		= 102 ;	
		= 103 ;-----	
		= 104 ;	
		= 105 ;	f : VT52 cursor control commands: (ESC commands)
		= 106 ;	
0048		= 107 HOMEC EQU 'H'	; Home cursor command.
004A		= 108 CLRSC EQU 'J'	; Clear screen command.
004B		= 109 CLRLC EQU 'K'	; Clear line command.
0049		= 110 REVLFC EQU 'I'	; Reverse line feed.
0059		= 111 GOTOC EQU 'Y'	; Direct cursor placement.
		= 112 ;	
		= 113 ;-----	
		= 114 ;	
		= 115 ;	g : VT52 cursor movement codes: (ESC + Code send)
		= 116 ;	
0041		= 117 UPC EQU 'A'	; Move cursor up 1 Line.
0042		= 118 DOWNC EQU 'B'	; Move cursor down 1 Line.
0043		= 119 RIGHTC EQU 'C'	; Move cursor right 1 Line.
0044		= 120 LEFTC EQU 'D'	; Move cursor left 1 Line.
		= 121 ;	
		= 122 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 3

LOC	OBJ	LINE	SOURCE STATEMENT
		= 123 ;	-----
		= 124 ;	
		= 125 ;	h : Target system definitions:
		= 126 ;	
0000		= 127 NEWBRAIN EQU FALSE	; Assmeble for SABus Kit.
0000		= 128 USART EQU FALSE	; Assemble for VDU and Keyboard.
		= 129 ;	
		= 130 ;	-----
		= 131 ;	
		= 132 ;	i : Conditional assembly for different target systems:
		= 133 ;	
		= 134 ;	-----
		= 135 ;	
		= 136 ;	i-1 (Cursor movement keys)
		= 137 ;	
		= 138 IF NEWBRAIN	
		= 139 ;	
		= 140 ;	Codes for Newbrain.
		= 141 ;	
		= 142 CONTUP EQU 11	; Cursor Up.
		= 143 CONTRI EQU 26	; Cursor Right.
		= 144 CONTDO EQU 10	; Cursor Down.
		= 145 CONTLE EQU 8	; Cursor Left.
		= 146 ;	
		= 147 MODCCH EQU 17	; Mode change character.
		= 148 ;	
		= 149 ELSE	
		= 150 ;	
		= 151 ;	Codes for Kit.
		= 152 ;	
0012		= 153 CONTUP EQU CONTROL AND 'R'	; Cursor Up.
0006		= 154 CONTRI EQU CONTROL AND 'F'	; Cursor Right.
0003		= 155 CONTDO EQU CONTROL AND 'C'	; Cursor Down.
0004		= 156 CONTLE EQU CONTROL AND 'D'	; Cursor Left.
		= 157 ;	
001A		= 158 MODCCH EQU CONTROL AND 'Z'	; Mode change character.
		= 159 ;	
		= 160 ENDIF	
		= 161 ;	
		= 162 ;	-----
		= 163 ;	
		= 164 ;	i-2 (Character output routines)
		= 165 ;	
		= 166 IF USART	
		= 167 ;	
		= 168 ;	Output routines for USART:
		= 169 ;	
		= 170 CONOUT EQU UO1	; Character output = USART 1 output.
		= 171 CONIN EQU UI1	; Character input = USART 1 input.
		= 172 CONTEST EQU U1TEST	; Character test = USART 1 test.
		= 173 ;	
		= 174 ELSE	
		= 175 ;	
		= 176 ;	Output routines for VDU and keyboard:
		= 177 ;	
F01C		= 178 CONOUT EQU CO	; Character output = Console output.
F013		= 179 CONIN EQU CI	; Character input = Console input.
F00A		= 180 CONTEST EQU CITEST	; Character test = Console test.
		= 181 ;	
		= 182 ENDIF	
		= 183 ;	
		= 184 ;	-----
		= 185 ;	
		= 186 ;	j : ASCII Constants:
		= 187 ;	
0000		= 188 NULL EQU 00H	; Null character.
0004		= 189 EOT EQU 04H	; End of text marker.
0007		= 190 BELL EQU 07H	; ASCII Bell character.
0008		= 191 BSPACE EQU 08H	; Back space.
0008		= 192 RUBOUT EQU 08H	; Rubout / Back_space

LOC	OBJ	LINE	SOURCE STATEMENT
0009		= 193 TAB EQU 09H	; Horiz tab character.
000A		= 194 LF EQU 0AH	; Line feed.
000D		= 195 CR EQU 0DH	; Carriage_return.
001B		= 196 ESC EQU 1BH	; Escape code.
0020		= 197 SPACE EQU ' '	; ASCII Space character.
003C		= 198 IPROMPT EQU '<'	; Insert prompt
005E		= 199 XPROMPT EQU '^'	; Exchange prompt
0027		= 200 QUOTE EQU ''''	; Quote character.(Used as a find command)
007F		= 201 DEL EQU 7FH	; Delete character.
		= 202 ;	
		= 203 ;	
		= 204 ;	
		= 205 ;	
		= 206 ;	k : Control characters:
		= 207 ;	
0003		= 208 CTRLC EQU CONTROL AND 'C'	; Control C
000B		= 209 CTRLK EQU CONTROL AND 'K'	; Control K
0011		= 210 CTRLQ EQU CONTROL AND 'Q'	; Control Q
0012		= 211 CTRLR EQU CONTROL AND 'R'	; Control R
0013		= 212 CTRLS EQU CONTROL AND 'S'	; Control S
0015		= 213 CTRLU EQU CONTROL AND 'U'	; Control U
001A		= 214 CTRLZ EQU CONTROL AND 'Z'	; Control Z
		= 215 ;	
		= 216 ;	
		= 217 ;	
		= 218 ;	l : Editor text length and memory equates:
		= 219 ;	
1000		= 220 PROG EQU 01000H	; Start adr of program
E800		= 221 DATA EQU 0E800H	; Start adr of the programs data block.
EF00		= 222 MONDATA EQU 0EF00H	; Monitor data area.
		= 223 ;	
B000		= 224 MAXLEN EQU 44*K	; Set max file size = 44K.
		= 225 ;	
		= 226 ;	
		= 227 ;	
		= 228 ;	m : Equates used by Editor file_I/O module.
		= 229 ;	
001A		= 230 EOFMARK EQU CTRLZ	; End_of_file marker = ^Z.
0040		= 231 ACK EQU '@'	; KIT ack character.
000B		= 232 QKILLC EQU CTRLK	; Queue killing character.
00FF		= 233 ETABLE EQU OFFH	; End of table marker.
FFFF		= 234 TOCONST EQU OFFFFH	; Filer time out constant (+- 1 sec).
		= 235 ;	
		= 236 ;	
		= 237 ;	
		= 238 ;	n : Screen limits.
		= 239 ;	
0000		= 240 TOPLIN EQU 0	; Line number for the TOP line of the screen.
0017		= 241 BOTLIN EQU 23	; Line number for the bottom line.
0000		= 242 MINCOL EQU 0	; Minimum column value (Col = 0..79)
004F		= 243 MAXCOL EQU 79	; Max column value
		= 244 ;	
0016		= 245 LSTLIN EQU 22	; Line number of the last line of text.
		= 246 ;	
		= 247 ;	
		= 248 ;	
		= 249 ;	o : Table and buffer lengths.
		= 250 ;	
0014		= 251 IPADLEN EQU 20	; Pad 20 bytes when inserting.
0078		= 252 KBLN EQU 120	; Keyboard buffer length.
0050		= 253 STRLEN EQU 80	; Max command string lenght.
00C8		= 254 STKLEN EQU 100 * WORD	; Length of editor stack.
000A		= 255 MAXCL EQU 10	; Maximum length of a string that is a legal command.
		= 256 ;	
0004		= 257 DBLEN EQU 4	; Maximum number of digits in a number.
		= 258 ;	
		= 259 ;	
		= 260 ;	
		= 261 ;	p : Text pointers:
		= 262 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 5

LOC	OBJ	LINE	SOURCE STATEMENT
		= 263	;SPOINT EQU MONDATA+100H-4 ; Start_of_text pointer.
		= 264	;EPOINT EQU , MONDATA+100H-2 ; End_of_text pointer.
		= 265	;
		= 266	-----
		= 267	;
		= 268	;
		= 269	q : Limits and offsets:
0020		= 270	OFFSET EQU 32 ; Offset used for direct cursor positioning.
000A		= 271	NBPACK EQU 10 ; Number of characters before you pack.
0028		= 272	MAXMOV EQU 40 ; Maximum number of lines that are displayed
		= 273	; when move down lines.
		= 274	;
		= 275	-----
		= 276	;
		= 277	;
		= 278	r : General equates:
001F		= 279	MASK EQU 00011111B ; Mask used for keyboard check period
		= 280	;
		= 281	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 6

LOC	OBJ	LINE	SOURCE STATEMENT
		= 282 ;	=====
		= 283 ;	
		= 284 ;	s : Monitor jump table.
		= 285 ;	
		= 286 ;	=====
		= 287 ;	
F00A		= 288 CTEST EQU	OF00AH ; Test if char in USART 0
F00D		= 289 UOTEST EQU	OF00DH ; Test if char in USART 1
F010		= 290 U1TEST EQU	OF010H ; Test if char in USART 2
F013		= 291 CI EQU	OF013H ; Input char from USART 0
F016		= 292 UI0 EQU	OF016H ; Input char from USART 1
F019		= 293 UI1 EQU	OF019H ; Input char from USART 2
F01C		= 294 CO EQU	OF01CH ; Output char to USART 0
F01F		= 295 UOO EQU	OF01FH ; Output char to USART 1
F022		= 296 UO1 EQU	OF022H ; Output char to USART 2
F025		= 297 BREAK EQU	OF025H ; Implement XON/XOFF protocol and sense ESC
F028		= 298 GETUC EQU	OF028H ; Input char from USART 0 - convert to UC
F02B		= 299 GETECHO EQU	OF02BH ; Input from console and echo
F02E		= 300 GETWD EQU	OF02EH ; Input up to four hex digits from console
F031		= 301 ASCBIN EQU	OF031H ; Convert ASCII to binary checking validity
F034		= 302 ECHO EQU	OF034H ; Output char to console checking for ESC, CR
F037		= 303 PRVAL EQU	OF037H ; Print nibble as hex digit
F03A		= 304 NMOUT EQU	OF03AH ; Print byte as two hex digits
F03D		= 305 OUTMSG EQU	OF03DH ; Print a string on the console
F043		= 306 DCMPI EQU	OF043H ; Compare HL to DE
F046		= 307 DCMPI EQU	OF046H ; Compare (HL) to DE
F049		= 308 TRANSP EQU	OF049H ; Enter transparent communication mode
F04C		= 309 VALDL EQU	OF04CH ; Test if valid delimiter
F04F		= 310 GETCM EQU	OF04FH ; Return to monitor command fetch routine
F052		= 311 GETHX EQU	OF052H ; Fetch 0/1 hex number, error return to monitor
F055		= 312 GET1NM EQU	OF055H ; Fetch one hex number, error return to monitor
F058		= 313 GET2NM EQU	OF058H ; Fetch 2 hex numbers, error return to monitor
F05B		= 314 GET3NM EQU	OF05BH ; Fetch 3 hex numbers, error return to monitor
		= 315 ;	
		= 316 ;	Other names used for monitor routines.
		= 317 ;	
F016		= 318 GETPDP EQU	UI0 ; Get_char_from_PDP = UI0
F01F		= 319 SNDPDP EQU	UOO ; Send a character to the PDP.
		= 320 ;	
		= 321 ;	End of Editor equates module.
		322 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 7

LOC	OBJ	LINE	SOURCE STATEMENT
		323 ;	
		324 ;	
		325 ;	
		326 ;	S T A R T O F M A I N P R O G R A M .
		327 ;	
1000		328	ORG PROG ; Start of program.
		329	;
		330	START: ; Editor entry point.
		331	;
1000	313EED	332	LXI SP,NEWSP ; Setup new stack pointer.
1003	CDC511	333	CALL EDTINI ; Init Editor.
		334 ;	
		335 ;	
		336 ;	
		337 ;	Start of main command loop.
		338 ;	
		339 ;	
		340 ;	
		341	COMLOP: ; Start of Main editor loop.
		342	;
1006	CDD513	343	CALL UPDATE ; Update the screen.
1009	CD0615	344	CALL PCUR ; Position cursor on screen.
100C	CD9323	345	CALL GETKEY ; Get a valid key from the keyboard.
100F	47	346	MOV B,A ; Save char in B register.
		347 ;	
		348 ;	
		349 ;	
		350 ;	Check for queue killing character.
		351 ;	
		352 ;	
		353 ;	
1010	FE0B	354	CPI QKILLC ;
1012	C21B10	355	JNZ NOTQKC ; No - Then try again
		356	;
		357	;
1015	CDEA12	357	CALL QINIT ; Initialize input queue
1018	C30610	358	JMP COMLOP ; Loop for next command
		359 ;	
		360 ;	
		361 ;	
		362 ;	Check for start of ESC seq.
		363 ;	
		364 ;	
		365 ;	
		366	NOTQKC: ;
101B	FE1B	367	CPI ESC ; ESC ?
101D	C22810	368	JNZ NOTESC ; No - Then not start of ESC seq.
		369	;
		370	;
1020	3EFF	370	MVI A,TRUE ;
1022	320FE8	371	STA ESCFLG ; ESC_Flag = TRUE
1025	C30610	372	JMP COMLOP ; Loop.
		373 ;	
		374 ;	
		375 ;	
		376 ;	Check for Delete key.
		377 ;	
		378 ;	
		379 ;	
		380	NOTESC: ;
1028	FE7F	381	CPI DEL ; Delete key ?
102A	C23310	382	JNZ NDEL ; No - Then try next option.
		383	;
		384	;
102D	CD811A	384	CALL DELCRO ; Delete character.
1030	C30610	385	JMP COMLOP ; Loop.
		386 ;	
		387 ;	
		388 ;	
		389 ;	Check to see if 2nd char in ESC seq (i.e. ESC_Flag set)
		390 ;	
		391 ;	
		392 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 8

LOC	OBJ	LINE	SOURCE STATEMENT
		393	NDEL: ;
1033	3A0FE8	394	LDA ESCFLG ;
1036	FE00	395	CPI FALSE ;
1038	CA6610	396	JZ NOESCS ; IF Esc_Flag # False the move cursor.
		397	;
103B	78	398	MOV A,B ; Restore character
103C	CD5614	399	CALL QCURM ; Character part of VT52 ESC seq ?
103F	CA6010	400	JZ ESCSEQ ; No - Then try for ESCn (n=0..9)
		401	;
		402	-----
		403	;
		404	Not VT52 <ESC> seq, check for <ESC>Digit
		405	;
		406	-----
		407	;
1042	78	408	MOV A,B ; Restore character
1043	FE30	409	CPI '0' ;
1045	FA5310	410	JM NOTDK ; If Character < '0' then not defined key
1048	FE3A	411	CPI '9'+1 ;
104A	F25310	412	JP NOTDK ; If Character > '9' then not defined key
		413	;
104D	CDFE26	414	CALL DEFINED ; Apply key def
1050	C30610	415	JMP COMLOP ;
		416	;
		417	-----
		418	;
		419	Character seq not <ESC>['A','B','C','D'] or <ESC>n (Error)
		420	;
		421	-----
		422	;
		423	NOTDK: ; Not defined key
1053	3E07	424	MVI A,BELL ;
1055	CD9D24	425	CALL PRINT ; Send a bell to the terminal
1058	3E00	426	MVI A,FALSE ;
105A	320FE8	427	STA ESCFLG ;
105D	C30610	428	JMP COMLOP ;
		429	;
		430	-----
		431	;
		432	Character seq = <ESC>['A','B','C','D'], to move the cursor.
		433	;
		434	-----
		435	;
		436	ESCSEQ: ;
1060	CD2E14	437	CALL MOVCUR ; Move the cursor
1063	C30610	438	JMP COMLOP ; Loop and get the next command.
		439	;
		440	-----
		441	;
		442	Check for Mode_Change_character.
		443	;
		444	-----
		445	;
		446	NOESCS: ;
1066	78	447	MOV A,B ; Restore character.
1067	FE1A	448	CPI MODCCH ; Char = Mode_change_char ?
1069	C27210	449	JNZ NOTCZ ; No - Then do not change mode.
		450	Else
106C	CDA213	451	CALL SETMODE ; Goto next mode.
106F	C30610	452	JMP COMLOP ; loop.
		453	;
		454	-----
		455	;
		456	Check for insert mode (i.e. Insert_flag = True).
		457	;
		458	-----
		459	;
		460	NOTCZ: ;
1072	3A15E8	461	LDA INSFLG ;
1075	FEFF	462	CPI TRUE ; Insert flag = true ?

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 9

LOC	OBJ	LINE	SOURCE STATEMENT
1077	C28110	463	JNZ NOTIM ; No - Then try next option.
		464	; Else
107A	78	465	MOV A,B ; Restore character.
107B	CDB41F	466	CALL INSERT ; Insert_Characters.
107E	C30610	467	JMP COMLOP ; Loop.
		468 ;	
		469 ;	
		470 ;	
		471 ;	Check for exchange mode (i.e. Exchange_flag = True).
		472 ;	
		473 ;	
		474 ;	
		475	NOTIM: ;
1081	3A16E8	476	LDA XFLG ;
1084	FEFF	477	CPI TRUE ; Exchange_flag = True ?
1086	C28F10	478	JNZ COMM ; No - Then try next Option.
		479	; Else
1089	CD0914	480	CALL XCHAR ; Exchange_Character.
108C	C30610	481	JMP COMLOP ; Loop.
		482 ;	
		483 ;	
		484 ;	
		485 ;	Read command string from keyboard.
		486 ;	
		487 ;	
		488 ;	
		489	COMM: ;
108F	3EFF	490	MVI A,TRUE ;
1091	3211E8	491	STA RESFLG ; Set restore cursor flag.
		492	;
1094	78	493	MOV A,B ; Restore character
1095	CDF524	494	CALL READS ; Read command line.
1098	CD9F25	495	CALL SCPRESS ; Space compress command line
		496	;
109B	3A17E8	497	LDA BUF ; Load 1st char of command.
109E	FE04	498	CPI EOT ; End of text ?
10A0	CA7411	499	JZ ENDCOM ; If no command entered then exit command
		500	; line routine.
		501 ;	
		502 ;	
		503 ;	
		504 ;	Command now read (1st check for nnnnL).
		505 ;	
		506 ;	
		507 ;	
10A3	CD0613	508	CALL QNUM ; Is the 1st character on the line a number? ,
10A6	C2C810	509	JNZ NONUM ; No - Then try next option.
		510	;
10A9	2117E8	511	LXI H,BUF ; Get start adr of command buffer.
10AC	CD4A26	512	CALL READNU ; If it is a number then read it (Into DE).
10AF	C25811	513	JNZ NERROR ; If read number returns error then number to
		514	; big error.
		515	;
10B2	7E	516	MOV A,M ; Load char after the number.
10B3	CDB811	517	CALL CONCAP ; Convert to upper case.
10B6	FE4C	518	CPI 'L' ; Char = 'L' ? (i.e Command = nnnnL ?).
10B8	C26E11	519	JNZ CER ; No - Then command error.
		520	;
10BB	23	521	INX H ;
10BC	7E	522	MOV A,M ; Fetch next character.
10BD	FE04	523	CPI EOT ;
10BF	C26E11	524	JNZ CER ; If next character not EQT then command error.
		525	;
10C2	CD3F1C	526	CALL MOVENL ; Move cursor nnnn lines down (DE = nnnn).
10C5	C37411	527	JMP ENDCOM ; Exit command line routine.
		528 ;	
		529 ;	
		530 ;	
		531 ;	(1st char # Num). Check for "-nnnnL"
		532 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 10

LOC	OBJ	LINE	SOURCE STATEMENT
		533 ;	-----
		534 ;	
		535 NONUM:	;
10C8	3A17E8	536	LDA BUF ; Load 1st character of the command line.
10CB	FE2D	537	CPI '-' ; Is the 1st character a minus sign?
10CD	C21411	538	JNZ NOTMIN ; Jump if NOT_MINUS.
		539	;
10D0	2117E8	540	LXI H,BUF ;
10D3	23	541	INX H ; Inc pointer to second character in
		542	; the command.
10D4	7E	543	MOV A,M ; Load character after the MINUS sign.
10D5	CDB811	544	CALL CONCAP ; Convert character to upper case.
10D8	FE04	545	CPI EOT ; EOT ?
10DA	CA6E11	546	JZ CER ; Just a minus => command error.
		547	;
10DD	FE4C	548	CPI 'L' ; 'L' ?
10DF	C2F210	549	JNZ NOTB1L ; Command # '-L...'
		550	; If BUF(2)='L' then command could be '-L'
10E2	23	551	INX H ; Inc next character.
10E3	7E	552	MOV A,M ; Fetch character after the 'L'.
10E4	FE04	553	CPI EOT ; EOT ?
10E6	C26E11	554	JNZ CER ; If Not EOT the command error.
		555	;
10E9	110100	556	LXI D,1 ; Set up line counter.
10EC	CD901C	557	CALL MOVEBL ; Move back N lines.
10EF	C37411	558	JMP ENDCOM ; End command.
		559 ;	
		560 ;	-----
		561 ;	
		562 ;	Command = "-???....", Check from "-nnnnL".
		563 ;	
		564 ;	-----
		565 ;	
		566 NOTB1L:	;
10F2	CD0613	567	CALL QNUM ; Is the character after the '-' a number ?
10F5	C26E11	568	JNZ CER ; If not "-n???" then command error.
		569	;
10F8	CD4A26	570	CALL READNU ; Read the number from the command line.
10FB	C25811	571	JNZ NERROR ; If read number returns error then number to
		572	; big error.
		573	;
10FE	7E	574	MOV A,M ; Get number terminator.
10FF	CDB811	575	CALL CONCAP ; Convert to upper case.
1102	FE4C	576	CPI 'L' ; char = 'L' ? (I.e. command = "-nnnnL" ?
1104	C26E11	577	JNZ CER ; No - then command error.
		578	;
1107	23	579	INX H ;
1108	7E	580	MOV A,M ; Fetch character after the 'L'
1109	FE04	581	CPI EOT ;
110B	C26E11	582	JNZ CER ; If next char not EOT then command error.
		583	;
110E	CD901C	584	CALL MOVEBL ; Else move back N lines DE = number of lines.
		585	;
1111	C37411	586	JMP ENDCOM ; Jump to end of command.
		587 ;	
		588 ;	-----
		589 ;	
		590 ;	Check for Word commands (e.g. "SAVE", "LOAD" etc).
		591 ;	
		592 ;	-----
		593 ;	
		594 NOTMIN:	;
1114	CD8B11	595	CALL COPY ; Copy the string From the input line to the
		596	; command buffer.
1117	E5	597	PUSH H ; Save pointer to next char on stack.
1118	CD2C13	598	CALL QTAB ; Command in command_table ?
111B	C1	599	POP B ; Put pointer to next character in BC.
111C	FA4211	600	JM NOTSC1 ; If command not in string table then the
		601	; command is not a string command.
111F	67	602	MOV H,A ; Save entry number.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 11

LOC	OBJ	LINE	SOURCE STATEMENT
1120	OA	603	LDAX B ; Get Copy terminating character.
1121	FE04	604	CPI EOT ; EOT ?
1123	CA5211	605	JZ EXCOM ;
		606	;
1126	FE20	607	CPI SPACE ; Space ?
1128	C23911	608	JNZ QDEF ;
		609 ;	
		610 ;	-----
		611 ;	
		612 ;	Command terminating character was a space, check to see if it
		613 ;	was a 'LOAD' or a 'SAVE' command.
		614 ;	
		615 ;	-----
		616 ;	
112B	7C	617	MOV A,H ; Restore table entry number.
112C	FE00	618	CPI LOADEN ; Load entry number ?
112E	CA5211	619	JZ EXCOM ; Yes - Then execute command.
		620	;
1131	FE01	621	CPI SAVEEN ; Save entry number ?
1133	CA5211	622	JZ EXCOM ;
1136	C36E11	623	JMP CER ;
		624 ;	
		625 ;	-----
		626 ;	
		627 ;	If terminating character not space or EOT check for DEFn (n=0-3).
		628 ;	
		629 ;	-----
		630 ;	
		631 QDEF:	;
1139	7C	632	MOV A,H ; Restore table entry number.
113A	FE02	633	CPI DEFEN ;
113C	C26E11	634	JNZ CER ; If not DEF then command error.
113F	C35211	635	JMP EXCOM ; Execute then command
		636 ;	
		637 ;	-----
		638 ;	
		639 ;	Not Word commands. Check for 1st letter in command table.
		640 ;	
		641 ;	-----
		642 ;	
		643 NOTSC1:	;
1142	3A17E8	644	LDA BUF ; Get the 1st character of the command.
1145	CDB811	645	CALL CONCAP ; Convert to UPPER case.
1148	47	646	MOV B,A ;
1149	213928	647	LXI H,COMTAB ; Load pointer to start of command table.
114C	CD5D14	648	CALL SEARCH ; Search command table for command.
114F	C26E11	649	JNZ CER ;
		650 ;	
		651 ;	-----
		652 ;	
		653 ;	Execute command (Adr of routine = DE, Command_ptr = BC).
		654 ;	
		655 ;	-----
		656 ;	
		657 EXCOM:	;
1152	217411	658	LXI H,ENDCOM ;
1155	E5	659	PUSH H ; Setup return adr.
1156	EB	660	XCHG ;
1157	E9	661	PCHL ; Indirect jump via HL register.
		662 ;	
		663 ;	=====
		664 ;	
		665 ;	Error exit points from the main command checker.
		666 ;	
		667 ;	=====
		668 ;	
		669 ;	NERROR (Error in number specification)
		670 ;	
		671 NERROR:	;
1158	21382C	672	LXI H,DERRM2 ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 12

LOC	OBJ	LINE	SOURCE STATEMENT
115B	CD8723	673	CALL ERRSTR ; Print error message
115E	C37411	674	JMP ENDCOM ; End of command loop
		675 ;	
		676 ;	
		677 ;	
		678 ;	Invalid ESC sequence.
		679 ;	
		680 COMERR:	
1161	3E00	681	MVI A,FALSE ;
1163	320FE8	682	STA ESCFLG ; ESC_Flag = False.
		683 ;	
1166	3E07	684	MVI A,BELL ;
1168	CD9D24	685	CALL PRINT ; Print (BELL)
		686 ;	
116B	C30610	687	JMP COMLOP ; Editor main loop.
		688 ;	
		689 ;	
		690 ;	
		691 ;	CER (Command error - Print "<Invalid Command>")
		692 ;	
		693 CER:	
116E	213E2B	694	LXI H,IVCMES ;
1171	CD8723	695	CALL ERRSTR ; Print error message
		696 ;	
		697 ;	
		698 ;	
		699 ;	End of command line routine.
		700 ;	
		701 ;	
		702 ;	
		703 ENDCOM:	
1174	3A11E8	704	LDA RESFLG ;
1177	FEFF	705	CPI TRUE ; If Restore_Flag = True then
1179	CC8F25	706	CZ RESCUR ; Restore cursor to old position.
		707 ;	
117C	C30610	708	JMP COMLOP ; Editor main loop.
		709 ;	
		710 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 13

LOC	OBJ	LINE	SOURCE STATEMENT
711			;
712			;
713			;
714			;
715			;
716			;
717			=====;
718			;
719			;
720			INCLUDE EXTERNAL MODULES.
721			;
722			;
723			=====;
724			;
725			Include general routines (Table searches etc)
726			;
727			\$INCLUDE(EDTGEN.A85)
728			-----;
729			;
730			EDTGEN.A85 module for the UCT SABus editor.
731			;
732			This module contains (in order) the following subroutines:
733			;
734			a) BBUF - Blank command buffer.
735			b) COPY - This routine is used to copy a command from the
736			input line buffer to the command buffer.
737			c) CONCAP - Convert characters to upper case.
738			d) EDTINI - Initialise all editor variables (Cold start).
739			e) FDCMP - Test for HL = DE.
740			f) GOBACK - Go backwards and search for the start of the
741			current line.
742			g) INCLIN - Increment line number
743			h) INITDT - Initialise definition table.
744			i) MINIT - Initialise mode (To command mode).
745			j) NEWSN - Redraws the text screen.
746			k) ONSCN - Checks to see if a line is on the screen or not.
747			l) QCLINE - Tests to see if the line is correct
748			m) QEOB - Test for end of buffer (EOB).
749			n) QINIT - Initialise input queue.
750			o) QLET - Tests to see if a character is a letter ('A'..'Z')
751			p) QNUM - Tests character for valid number (0..9).
752			q) QSOB - Test for start of buffer (SOB).
753			r) QTAB - Searches a table for the string in the command
754			buffer.
755			s) RULF - Read until line feed.
756			t) RBULF - Read backwards until line feed.
757			u) SETMODE - Changes the editor mode to the next in the list.
758			v) UPDATE - Update the screen.
759			w) XHAR - Exchange character.
760			;
761			-----;
762			;
763			\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 14

LOC	OBJ	LINE	SOURCE STATEMENT
		= 764 ;	=====
		= 765 ;*	
		= 766 ;*	Function : BBUF (Blank command buffer)
		= 767 ;*	Inputs : HL : Start address of command buffer.
		= 768 ;*	Outputs : None.
		= 769 ;*	Destroys : B and Flags.
		= 770 ;*	Calls : None.
		= 771 ;*	
		= 772 ;*	Description : This routine is used to blank the command
		= 773 ;*	buffer before the command string is copied into it by the
		= 774 ;*	<COPY> routine.
		= 775 ;*	
		= 776 ;	=====
		= 777 ;	
		= 778 BBUF:	; Start of blank buffer routine
117F	E5	= 779	PUSH H ;
1180	060A	= 780	MVI B,MAXCL ; B := Max_command_length;
		= 781	; Repeat
1182	3620	= 782 BBUFPL:	; Repeat
		= 783	MVI M,' ' ; Buffer(HL) = space
1184	23	= 784	INX H ; HL := HL + 1;
1185	05	= 785	DCR B ; B := B - 1;
1186	C28211	= 786	JNZ BBUFPL ; Until B=0
		= 787	;
1189	E1	= 788	POP H ;
118A	C9	= 789	RET ; Return from <BBUF> routine.
		= 790 ;	
		= 791 ;	=====
		= 792 ;*	
		= 793 ;*	Function : COPY (Copies a string from the input line to
		= 794 ;*	the command buffer)
		= 795 ;*	Inputs : None.
		= 796 ;*	Outputs : HL : Points to the next character in the
		= 797 ;*	input line.
		= 798 ;*	Destroys : A,B,C,D,E,H,L and Flags.
		= 799 ;*	Calls : CONCAP and QLET.
		= 800 ;*	
		= 801 ;*	Description : This routine is used to copy a command string
		= 802 ;*	from the input line into the command buffer. A command
		= 803 ;*	string is simply a series of letters ('A'..'Z'). If the
		= 804 ;*	command buffer is filled before the command string has
		= 805 ;*	been copied from the input line to the command buffer then
		= 806 ;*	the routine will skip character in the input line until a
		= 807 ;*	non valid command string character is found.
		= 808 ;*	
		= 809 ;	=====
		= 810 ;	
		= 811 COPY:	; Start of <COPY> routine
		= 812	;
118B	1117E8	= 813	LXI D,BUF ; DE = Start address of the input line.
118E	2167E8	= 814	LXI H,TCOMBU ; HL = Start address of the command buffer
		= 815	;
1191	CD7F11	= 816	CALL BBUF ; Blank the command buffer.
1194	060A	= 817	MVI B,MAXCL ; B = Maximum length of the command buffer
		= 818 MORTC:	;
1196	1A	= 819	LDAX D ; Fetch a character from the input line.
1197	CDB811	= 820	CALL CONCAP ; Convert it to upper case.
119A	4F	= 821	MOV C,A ; Save upper case character in C register
119B	CDF312	= 822	CALL QLET ; Is it a letter ('A'..'Z') ?
119E	C2B611	= 823	JNZ CPYEX ; No - Then the end of the command has been
		= 824	; reached (so exit)
		= 825	;
11A1	71	= 826	MOV M,C ; Put the uppercase character in the command
		= 827	; buffer.
		= 828	;
11A2	23	= 829	INX H ; Increment the command buffer pointer
11A3	13	= 830	INX D ; Increment the input line pointer
11A4	05	= 831	DCR B ; Decrement the character counter
11A5	C29611	= 832	JNZ MORTC ; If Character counter <> zero then loop
		= 833 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 15

LOC	OBJ	LINE	SOURCE STATEMENT
		= 834 ;	-----
		= 835 ;	
		= 836 ;	If the command buffer is full but the end of the command has not yet
		= 837 ;	been reached then continue reading characters until the character
		= 838 ;	read is not a valid command string character (i.e. Not a letter).
		= 839 ;	
		= 840 SKIPNC:	; Start of skip character loop.
11A8	1A	= 841	LDAX D ; Fetch the next character from the input line
11A9	CDB811	= 842	CALL CONCAP ; Convert to upper case
11AC	CDF312	= 843	CALL QLET ; Is it a letter ?
11AF	C2B611	= 844	JNZ CPYEX ; If it is not a letter then the end of the
		= 845	; command string has been reached so exit
		= 846	
11B2	13	= 847	INX D ; Else - Increment the input line ptr and
11B3	C3A811	= 848	JMP SKIPNC ; loop.
		= 849 CPYEX:	
11B6	EB	= 850	XCHG ; Put the pointer to the terminating character
		= 851	; into the HL register pair.
11B7	C9	= 852	RET ; Return from the <COPY> routine.
		= 853 ;	
		= 854 ;	=====
		= 855 ;*	*
		= 856 ;*	Function : CONCAP (Convert from lower to upper case) *
		= 857 ;*	Inputs : A : Character to be converted to upper case) *
		= 858 ;*	Outputs : A : Upper case character or original *
		= 859 ;*	character if it was not a letter. *
		= 860 ;*	Destroys : A and Flags. *
		= 861 ;*	Calls : None. *
		= 862 ;*	*
		= 863 ;*	Description : This routine checks to see if the character *
		= 864 ;*	passed in the A register is a lower case character and if *
		= 865 ;*	it is, it is converted to upper case. *
		= 866 ;*	*
		= 867 ;	=====
		= 868 ;	
		= 869 CONCAP:	; Start of <CONCAP> routine
11B8	FE61	= 870	CPI 'a' ; If Character >= 'a' OR
11BA	FAC411	= 871	JM CCEXIT ;
		= 872	
11BD	FE7B	= 873	CPI 'z' + 1 ; Character < 'z'+1 THEN
11BF	F2C411	= 874	JP CCEXIT ;
		= 875	
11C2	C6E0	= 876	ADI 'A'-'a' ; Character := Character - 'a' + 'A'
		= 877 CCEXIT:	
11C4	C9	= 878	RET ; Return from <CONCAP> routine
		= 879 ;	
		= 880 ;	=====
		= 881 ;*	*
		= 882 ;*	Function : EDTINI (Editor initialise routine) *
		= 883 ;*	Inputs : None. *
		= 884 ;*	Outputs : None. *
		= 885 ;*	Destroys : A,D,E,H,L and Flags. *
		= 886 ;*	Calls : UOTEST, UIO, INITDT, QINIT, MINIT, HOME, CLRS *
		= 887 ;*	FDCMP, BLINE and PLINE. *
		= 888 ;*	*
		= 889 ;*	Description : This routine initialises all the editor *
		= 890 ;*	variables at the start of an editing session. If there is *
		= 891 ;*	no file in memory an end of line character (A line feed) *
		= 892 ;*	is stored at the start of the file. The following is a *
		= 893 ;*	summary of the state of the editor after initialisation: *
		= 894 ;*	*
		= 895 ;*	Start_of_line_ptr and Character_ptr = Start_of_text *
		= 896 ;*	There is no CUT buffer active. *
		= 897 ;*	All key definitions have been cancelled. *
		= 898 ;*	There are no characters in the input queue. *
		= 899 ;*	The editor mode has be reset to command mode. *
		= 900 ;*	There are no NULLS in the file. *
		= 901 ;*	*
		= 902 ;	=====
		= 903 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 16

LOC	OBJ	LINE	SOURCE STATEMENT
		= 904	EDTINI: ; Editor Init Routine.
11C5	CD0DF0	= 905	CALL UOTEST ; Check for a character in the USART.
11C8	C416F0	= 906	CNZ UIO ; If there Is a character fetch it.
		= 907	;
11CB	3E00	= 908	MVI A,00H ;
11CD	320EE8	= 909	STA LCLINE ; Set Lowest_Correct_Line = 0.
11D0	320AE8	= 910	STA RELCH ; Set Relative character pos = 0.
11D3	3254EC	= 911	STA SEASL ; Init Search string length = 0.
11D6	320BE8	= 912	STA NULLS ; Init Null counter = 0.
		= 913	;
11D9	2AFCEF	= 914	LHLD SPOINT ; Get start of text pointer.
11DC	2200E8	= 915	SHLD POINT ; Init Line pointer = start of text.
11DF	2202E8	= 916	SHLD CHAR ; Init Char pointer = start of text.
		= 917	;
11E2	11FFAF	= 918	LXI D,MAXLEN-1 ;
11E5	19	= 919	DAD D ;
11E6	22F0E8	= 920	SHLD MAXTXT ; Save max text pointer.
		= 921	;
11E9	3E00	= 922	MVI A,FALSE ;
11EB	3213E8	= 923	STA NOSEND ; Init flow control flag = FALSE.
11EE	3212E8	= 924	STA SUBBIT ; Sub buffers not in tack.
11F1	320FE8	= 925	STA ESCFLG ; Reset ESC flag.
11F4	32EFE8	= 926	STA CUTBUF ; Reset CUT buffer flag = Empty.
		= 927	;
11F7	CD6712	= 928	CALL INITDT ; Init key def table
		= 929	;
11FA	CD7612	= 930	CALL MINIT ; Init Mode (Command mode).
11FD	CDEA12	= 931	CALL QINIT ; Init input Queue.
		= 932	;
1200	3E17	= 933	MVI A,LSTLIN+1 ; 1st line off the screen.
1202	3208E8	= 934	STA EOBLIN ; Set EOB off screen.
		= 935	;
1205	CDCA26	= 936	CALL HOME ; Home cursor.
1208	CDAE26	= 937	CALL CLRS ; Clear screen.
		= 938	;
120B	2AFCEF	= 939	LHLD SPOINT ;
120E	EB	= 940	XCHG ;
120F	2AFEEF	= 941	LHLD EPOINT ;
1212	CD3012	= 942	CALL FDCMP ;
1215	CA2D12	= 943	JZ NOFILE ; If Start pointer # End pointer then
		= 944	Print the line.
1218	7E	= 945	MOV A,M ; Fetch last character in the file
1219	FE0A	= 946	CPI LF ; Last character = LF ?
121B	CA2412	= 947	JZ LASTLF ;
		= 948	;
121E	23	= 949	INX H ; No then add one
121F	360A	= 950	MVI M,LF ; Add a line feed to the end of the file.
1221	22FEEF	= 951	SHLD EPOINT ; Save new end pointer
		= 952	LASTLF: ;
1224	CD9D26	= 953	CALL BLINE ; Move cursor to beginning of line.
1227	CD7224	= 954	CALL PLINE ; Then print the top line.
122A	C32F12	= 955	JMP EDTIE ; Goto main Editor Loop.
		= 956	NOFILE: ;
122D	360A	= 957	MVI M,LF ; Make the first char=LF if no file.
		= 958	EDTIE: ; Edt init Exit.
122F	C9	= 959	RET ; Return from <EDTINI>
		= 960 ;	
		= 961 ;	=====
		= 962 ;*	
		= 963 ;*	Function : FDCMP (Fast DE/HL compare routine) *
		= 964 ;*	Inputs : DE : 1st 16 bit number. *
		= 965 ;*	HL : 2nd 16 bit number. *
		= 966 ;*	Outputs : Z = 1: If DE=HL *
		= 967 ;*	Z = 0: If DE#HL *
		= 968 ;*	Destroys : A and Flags. *
		= 969 ;*	Calls : None. *
		= 970 ;*	
		= 971 ;*	Description : This routine performs part of the function of *
		= 972 ;*	the <DCMP> routine, but all that has to be tested is *
		= 973 ;*	equality then this routine is faster. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 17

LOC	OBJ	LINE	SOURCE STATEMENT
		= 974 ;*	
		= 975 ;=====	
		= 976 ;	
		= 977 FDCMP:	; Start of fast D Compare routine (DE=HL ?).
1230	7A	= 978 MOV A,D	
1231	BC	= 979 CMP H	
1232	C0	= 980 RNZ	; If D_reg <> H_reg Then return with zero
		= 981	; Flag Clear
1233	7B	= 982 MOV A,E	
1234	BD	= 983 CMP L	
1235	C9	= 984 RET	; If E_reg <> L_reg then Zero flag clear.
		= 985 ;	
		= 986 ;=====	
		= 987 ;*	
		= 988 ;* Function : GOBACK (Go backwards in a line, to find the	*
		= 989 ;* beginning)	*
		= 990 ;* Inputs : HL=Character pointer (Somewhere in a line)	*
		= 991 ;* Outputs : HL=Start of line pointer.	*
		= 992 ;* Destroys : H,L,D,E,A and Flags.	*
		= 993 ;* Calls : FDCMP.	*
		= 994 ;*	*
		= 995 ;* Description : This routine is used to search backwards in a	*
		= 996 ;* line to try and find the start of line pointer. It does	*
		= 997 ;* this by searching backwards until it finds a line feed and	*
		= 998 ;* assumes that the character following the line feed is the	*
		= 999 ;* start of line pointer from the current line.	*
		=1000 ;*	*
		=1001 ;* The routine also checks that it does not go passed the	*
		=1002 ;* start of the file.	*
		=1003 ;*	*
		=1004 ;=====	
		=1005 ;	
		=1006 GOBACK:	; Start of <GOBACK> routine.
1236	2202E8	=1007 SHLD CHAR	; Update character pointer to character given.
1239	EB	=1008 XCHG	; hold in DE.
123A	2AFCEF	=1009 LHLD SPOINT	; Load start of text pointer.
123D	EB	=1010 XCHG	; Restore character pointer from DE.
123E	7E	=1011 MOV A,M	; Load current character.
123F	FE0A	=1012 CPI LF	; Pointing to the end_of_line ?
1241	C24512	=1013 JNZ' GOB05	; If not end of line then look for line feed.
		=1014	
1244	2B	=1015 DCX H	
		=1016 GOB05:	
1245	7E	=1017 MOV A,M	; Load character from text.
1246	FE0A	=1018 CPI LF	
1248	CA5512	=1019 JZ SOLFO	; If char=LF then start of line found.
		=1020	
124B	CD3012	=1021 CALL FDCMP	
124E	CA5612	=1022 JZ SOBFO	; If Pointer = Start_of_text then start of
		=1023	; Buffer found.
1251	2B	=1024 DCX H	; Decrement the line pointer and loop
1252	C34512	=1025 JMP GOB05	
		=1026 ;	
		=1027 ;-----	
		=1028 ;	
		=1029 ; Exit here is a line feed was found (i.e. Inc pointer to start of line)	
		=1030 ;	
		=1031 SOLFO:	
1255	23	=1032 INX H	; Inc to character after LF
		=1033 ;	
		=1034 ;-----	
		=1035 ;	
		=1036 ; Exit here is the start of the file was detected.	
		=1037 ;	
		=1038 SOBFO:	
1256	2200E8	=1039 SHLD POINT	; Update the start of line pointer
1259	C9	=1040 RET	; Return from the <GOBACK> routine
		=1041 ;	
		=1042 ;=====	
		=1043 ;*	*

LOC	OBJ	LINE	SOURCE STATEMENT
		=1044 ;*	Function : INCLIN (Increment line number) *
		=1045 ;*	Inputs : None. *
		=1046 ;*	Outputs : None. *
		=1047 ;*	Destroys : A and Flags. *
		=1048 ;*	Calls : None. *
		=1049 ;*	;
		=1050 ;*	Description : This routine simply increments the current *
		=1051 ;*	line number, but checks that it does not go above the *
		=1052 ;*	maximum number of lines on the screen. *
		=1053 ;*	;
		=1054 ;	=====
		=1055 ;	;
		=1056 INCLIN:	; Start of the <INCLIN> routine
125A	3A05E8	=1057 LDA LINE	; A = Current_line
125D	FE16	=1058 CPI LSTLIN	; If Current_line_ <> Last_line then
125F	CA6612	=1059 JZ INCLEX	;
		=1060	;
1262	3C	=1061 INR A	; Current_line := Current_line + 1;
1263	3205E8	=1062 STA LINE	;
		=1063 INCLEX:	;
1266	C9	=1064 RET	; Return from <INCLIN> routine.
		=1065 ;	;
		=1066 ;	=====
		=1067 ;*	;
		=1068 ;*	Function : INITDT (Initialise key definition table) *
		=1069 ;*	Inputs : None. *
		=1070 ;*	Outputs : None. *
		=1071 ;*	Destroys : A, B, H, L and Flags. *
		=1072 ;*	Calls : None. *
		=1073 ;*	;
		=1074 ;*	Description : This routine sets all keys as undefined by *
		=1075 ;*	setting the length of all definitions to zero. *
		=1076 ;*	;
		=1077 ;	=====
		=1078 ;	;
		=1079 INITDT:	; Start of <INITDT> routine
1267	21F6E8	=1080 LXI H,DEFTAB-1	; HL = Start address of key definition
		=1081	; pointer table.
126A	060A	=1082 MVI B,10	; B = Number of entries in the key defintion
		=1083	; pointer table.
		=1084 INITDL:	;
126C	23	=1085 INX H	;
126D	23	=1086 INX H	;
126E	23	=1087 INX H	; Pointer = Pointer + 3 (next def flag).
126F	3600	=1088 MVI M,0	; Set length = zero (i.e. undefined)
1271	05	=1089 DCR B	;
1272	C26C12	=1090 JNZ INITDL	; Init all 10 entries
		=1091	;
1275	C9	=1092 RET	; Return from <INITDT>
		=1093 ;	;
		=1094 ;	=====
		=1095 ;*	;
		=1096 ;*	Function : MINIT (mode initialise) *
		=1097 ;*	Inputs : None. *
		=1098 ;*	Outputs : None. *
		=1099 ;*	Destroys : A. *
		=1100 ;*	Calls : None. *
		=1101 ;*	;
		=1102 ;*	Description : This routine resets the editor mode to *
		=1103 ;*	command mode by clearing both the insert and exchange *
		=1104 ;*	flags. *
		=1105 ;*	;
		=1106 ;	=====
		=1107 ;	;
		=1108 MINIT:	; Init editor mode.
1276	3E00	=1109 MVI A,FALSE	;
1278	3215E8	=1110 STA INSFLG	; Init Mode: Insert mode OFF.
127B	3216E8	=1111 STA XFLG	; Exchange mode OFF.
		=1112	; (i.e. Powerup in command mode).
127E	C9	=1113 RET	; Return from <MINIT>.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 19

LOC	OBJ	LINE	SOURCE STATEMENT
		=1114 ;	
		=1115 ;=====	
		=1116 ;*	
		=1117 ;*	Function : NEWSN (New screen routine)
		=1118 ;*	Inputs : None.
		=1119 ;*	Outputs : None.
		=1120 ;*	Destroys : A and Flags.
		=1121 ;*	Calls : HOME and PLINE.
		=1122 ;*	
		=1123 ;*	Description : This routine is used to redraw the entire
		=1124 ;*	screen when it has been changed (e.g. after a 'BR' or 'ER'
		=1125 ;*	command).
		=1126 ;*	
		=1127 ;=====	
		=1128 ;	
		=1129 NEWSN:	; Start of <NEWSN> routine.
127F	CDCA26	=1130	CALL HOME ;
1282	3E17	=1131	MVI A,LSTLIN+1 ; Set EOB pos off the screen.
1284	3208E8	=1132	STA EOBLIN ;
1287	3E00	=1133	MVI A,00 ; Reset Lowest_correct_line pointer.
1289	320EE8	=1134	STA LCLINE ;
128C	3E00	=1135	MVI A,FALSE ; Clear Restore cursor flag.
128E	3211E8	=1136	STA RESFLG ;
1291	CD7224	=1137	CALL PLINE ;
1294	C9	=1138	RET ; Return from <NEWSN>
		=1139 ;	
		=1140 ;=====	
		=1141 ;*	
		=1142 ;*	Function : ONSN (Is the character pointed to by the
		=1143 ;*	character pointer on the screen ?)
		=1144 ;*	Inputs : HL : Character pointer
		=1145 ;*	Outputs : Sign bit = 1: Line off screen
		=1146 ;*	Sign bit = 0, A = Line number of line
		=1147 ;*	containing the character pointer.
		=1148 ;*	Destroys : A,B,C,D,E,H,L and Flags.
		=1149 ;*	Calls : FDCMP
		=1150 ;*	
		=1151 ;*	Description : This routine, knowing the current line
		=1152 ;*	number and the start address of the current line works out
		=1153 ;*	whether or not the character pointed to by the character
		=1154 ;*	pointer (Passed in HL) is pointing to a character of the
		=1155 ;*	screen. If it is then the line number is returned to the
		=1156 ;*	calling routine in the A register.
		=1157 ;*	
		=1158 ;=====	
		=1159 ;	
		=1160 ONSN:	; Start of <ONSN> routine.
1295	EB	=1161	XCHG ; Move pointer to DE register pair
1296	2A02E8	=1162	LHLD CHAR ; load pointer to sub string.
1299	EB	=1163	XCHG ;
129A	3A07E8	=1164	LDA OLINE ; Load line number that corresponds to the
		=1165	; start of line pointer in the HL Reg Pair.
129D	47	=1166	MOV B,A ; Save line number in the B register
129E	3A0EE8	=1167	LDA LCLINE ; Load line number of the lowest correct
		=1168	; line on the screen.
12A1	4F	=1169	MOV C,A ; Save lowest_correct_line # in C register.
		=1170 ONS05:	
12A2	79	=1171	MOV A,C ; LOAD Lowest correct line
12A3	B8	=1172	CMP B ; LINE # > LCLINE ?
12A4	F8	=1173	RM ; If minus then yes.
		=1174 ONS10:	
12A5	CD3012	=1175	CALL FDCMP ; Text pointer = character pointer ?
12A8	CABA12	=1176	JZ ONS20 ; If it is then substitute on screen.
		=1177	
12AB	7E	=1178	MOV A,M ; Load character from the text.
12AC	FE0A	=1179	CPI LF ;
12AE	CAB512	=1180	JZ ONS15 ; End of the line.
		=1181	
12B1	23	=1182	INX H ;
12B2	C3A512	=1183	JMP ONS10 ; Loop until eoln=true.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 20

LOC	OBJ	LINE	SOURCE STATEMENT
		=1184	ONS15: ;
12B5	23	=1185	INX H ; Inc pointer past the line feed.
12B6	04	=1186	INR B ; Inc line number.
12B7	C3A212	=1187	JMP ONS05 ; Loop until below Lowest_correct_line
		=1188	ONS20: ;
12BA	78	=1189	MOV A,B ; Get current line number.
12BB	B7	=1190	ORA A ; Set flags.
12BC	C9	=1191	RET ; Return from <ONSCN> routine
		=1192 ;	
		=1193 ;=====	
		=1194 ;*	
		=1195 ;*	Function : QCLINE (? Correct line) *
		=1196 ;*	Inputs : None. *
		=1197 ;*	Outputs : Z=1 : If the current line is correct. *
		=1198 ;*	Z=0 : If the current line is not correct. *
		=1199 ;*	Destroys : A,B and Flags. *
		=1200 ;*	Calls : None. *
		=1201 ;*	
		=1202 ;*	Description : This routine is used to check that the *
		=1203 ;*	current line is correct when the cursor is being moved up *
		=1204 ;*	and down within a page, e.g. If the cursor is on the last *
		=1205 ;*	line of the screen and was moved up 4 lines and then moved *
		=1206 ;*	down 5 lines only the last move down would return current *
		=1207 ;*	line not correct. *
		=1208 ;*	
		=1209 ;=====	
		=1210 ;	
		=1211	QCLINE: ; Start of <QCLINE> routine
12BD	3A05E8	=1212	LDA LINE ;
12C0	FE16	=1213	CPI LSTLIN ; Current line = last line on screen ?
12C2	CACD12	=1214	JZ LNOTC ; If last line then not correct
		=1215	;
12C5	47	=1216	MOV B,A ; Save current line in the B register.
12C6	3A0EE8	=1217	LDA LCLINE ; A = Lowest correct line.
12C9	B8	=1218	CMP B ;
12CA	F2D212	=1219	JP LC ; If Current_line > Lowest_correct_line then
		=1220	;
		=1221	LNOTC: ;
12CD	3EFF	=1222	MVI A,OFFH ; A := OFFH (* Clears zero flag *)
12CF	C3D412	=1223	JMP QCLRET ;
		=1224	;
		=1225	LC: ;
12D2	3E00	=1226	MVI A,00H ; A := 0; (* Sets zero flag *)
		=1227	QCLRET: ;
12D4	B7	=1228	ORA A ; Set flags before returning
12D5	C9	=1229	RET ; End of <QCLINE> routine
		=1230 ;	
		=1231 ;=====	
		=1232 ;*	
		=1233 ;*	Function : QEOB (End of buffer ?) *
		=1234 ;*	Inputs : HL : Character pointer. *
		=1235 ;*	Outputs : Z=1 : If the end of the buffer has been *
		=1236 ;*	reached. *
		=1237 ;*	Z=0 : If the end of the buffer has not been *
		=1238 ;*	reached. *
		=1239 ;*	Destroys : D,E,H,L,A and Flags. *
		=1240 ;*	Calls : DCMP. *
		=1241 ;*	
		=1242 ;*	Description : This routine checks to see if the character *
		=1243 ;*	pointer passed in the HL register pair is at or passed the *
		=1244 ;*	end of buffer pointer. *
		=1245 ;*	
		=1246 ;=====	
		=1247 ;	
		=1248	QEOB: ; Start of <QEOB> routine
12D6	EB	=1249	XCHG ;
12D7	2AFEEF	=1250	LHLD EPOINT ;
12DA	EB	=1251	XCHG ;
12DB	CD43F0	=1252	CALL DCMP ;
12DE	3E00	=1253	MVI A,OK ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 21

LOC	OBJ	LINE	SOURCE STATEMENT
12E0	DAE812	=1254	JC EOB05 ; If Character_pointer <= End_pointer then
		=1255	;
12E3	3EFF	=1256	MVI A,ERR
12E5	C3E812	=1257	JMP EOB05
		=1258	EOB05: ;
12E8	B7	=1259	ORA A ; Set flags before returning
12E9	C9	=1260	RET ;
		=1261 ;	
		=1262 ;=====	
		=1263 ;*	
		=1264 ;*	Function : QINIT (Input queue initialise routine)
		=1265 ;*	Inputs : None.
		=1266 ;*	Outputs : None.
		=1267 ;*	Destroys : A.
		=1268 ;*	Calls : None.
		=1269 ;*	
		=1270 ;*	Description : This routine initialises the input queue by
		=1271 ;*	setting both pointer to zero - the fact that the pointers
		=1272 ;*	are equal means that the queue is empty.
		=1273 ;*	
		=1274 ;=====	
		=1275 ;	
		=1276	QINIT: ; Start of <QINIT> routine
12EA	3E00	=1277	MVI A,00H ;
12EC	32EDE8	=1278	STA UP ; Upper_pointer := 0;
12EF	32EEE8	=1279	STA LO ; Lower_pointer := 0;
		=1280	;
12F2	C9	=1281	RET ; Return from <QINIT> routine.
		=1282 ;	
		=1283 ;=====	
		=1284 ;*	
		=1285 ;*	Function : QLET (? letter, 'A'..'Z').
		=1286 ;*	Inputs : A : Character to be checked.
		=1287 ;*	Outputs : Z=1 : If the character is a letter.
		=1288 ;*	Z=0 : If the character is not a letter.
		=1289 ;*	Destroys : A and Flags.
		=1290 ;*	Calls : None.
		=1291 ;*	
		=1292 ;*	Description : This routine is used to check to see if a
		=1293 ;*	character is a letter ('A'..'Z'), i.e. a valid command
		=1294 ;*	string character.
		=1295 ;*	
		=1296 ;=====	
		=1297 ;	
		=1298	QLET: ; Start of the <QLET> routine.
12F3	FE41	=1299	CPI 'A' ;
12F5	FAFD12	=1300	JM NOTLET ; If Character < 'A' AND
		=1301	;
12F8	FE5B	=1302	CPI 'Z'+1 ;
12FA	FA0213	=1303	JM OKLET ; Character > 'Z' THEN
		=1304	;
		=1305	NOTLET: ;
12FD	3EFF	=1306	MVI A,OFFH ; A := OFFH;
12FF	C30413	=1307	JMP QLETEX ; Else
		=1308	OKLET: ;
1302	3E00	=1309	MVI A,000H ; A := 0H;
		=1310	QLETEX: ;
1304	B7	=1311	ORA A ; Set flags before returning
		=1312	;
1305	C9	=1313	RET ; Return from <QLET> routine
		=1314 ;	
		=1315 ;=====	
		=1316 ;*	
		=1317 ;*	Function : QNUM (Number ?)
		=1318 ;*	Inputs : A: Character to be checked.
		=1319 ;*	Outputs : Z=1: If character is a valid number (0..9)
		=1320 ;*	Z=0: If Character is not a valid number.
		=1321 ;*	Destroys : A and Flags.
		=1322 ;*	Calls : None.
		=1323 ;*	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 23

LOC	OBJ	LINE	SOURCE STATEMENT
		=1394 ;*	The format of the table is as follows: *
		=1395 ;*	*
		=1396 ;*	TABLE: DB '<First_string_command>',EOT *
		=1397 ;*	DB '<Second_string_command>',EOT *
		=1398 ;*	" " *
		=1399 ;*	" " *
		=1400 ;*	DB '<Last_string_command>',EOT *
		=1401 ;*	DB ETABLE *
		=1402 ;*	*
		=1403 ;*	The string command typed in is stored in a space filled *
		=1404 ;*	buffer starting at <TCOMBU> and has a length of 10 *
		=1405 ;*	characters. *
		=1406 ;*	*
		=1407 ;*	Note1: All the letters in the command table are in upper *
		=1408 ;*	case and the command string is converted to upper *
		=1409 ;*	case when it is copied to the command string *
		=1410 ;*	buffer. The command line is therefore not case *
		=1411 ;*	sensitive. *
		=1412 ;*	*
		=1413 ;	=====
		=1414 ;	*
		=1415	QTABLE: ; Start of QTABLE.
132C	0600	=1416	MVI B,00H ; Init counter = 0.
132E	114F28	=1417	LXI D,STRCOM ; Load DE with a pointer to the start
		=1418	; of the commands that are strings.
		=1419	QTABLE: ;
1331	2167E8	=1420	LXI H,TCOMBU ; HL = Start_of_word_to_check.
		=1421	QTABLENB: ;
1334	1A	=1422	LDAX D ; Get byte from Table.
1335	FE04	=1423	CPI EOT ; Table_byte = EOT ?
1337	CA5B13	=1424	JZ SFOUND ; Yes - then Entry found.
		=1425	;
133A	FEFF	=1426	CPI ETABLE ; Table_byte = End_Table ?
133C	CA5613	=1427	JZ ENDTAB ; Yes - Then Error entry not in table.
		=1428	;
133F	BE	=1429	CMP M ; Table_byte = command_line_byte ?
1340	C24813	=1430	JNZ NOTE ; No - Then not this entry.
		=1431	;
1343	13	=1432	INX D ; Inc table pointer.
1344	23	=1433	INX H ; Inc command pointer.
1345	C33413	=1434	JMP QTABNB ; Loop
		=1435	;
		=1436	NOTE: ; Not this entry (Skip to end of it).
1348	13	=1437	INX D ; Inc Table_ptr
1349	1A	=1438	LDAX D ; Get Table_byte.
134A	FE04	=1439	CPI EOT ; Table_byte = EOT ?
134C	C24813	=1440	JNZ NOTE ; No - then loop until end_of_entry.
		=1441	;
134F	13	=1442	INX D ; Inc past EOT.
1350	13	=1443	INX D ; Inc pointer past low byte of Vector.
1351	13	=1444	INX D ; Inc pointer past high byte of vector.
		=1445	;
1352	04	=1446	INR B ; Inc entry counter.
1353	C33113	=1447	JMP QTABL ; Try next entry.
		=1448	;
		=1449	ENDTAB: ; If entable found then error return
		=1450	; (i.e. String not found).
1356	3EFF	=1451	MVI A,OFFH ; Acc = Error return.
1358	C36713	=1452	JMP QTABX ;
		=1453	;
		=1454	SFOUND: ;
135B	7E	=1455	MOV A,M ; Fetch next character in the command buffer.
135C	FE20	=1456	CPI SPACE ; Space ?
135E	C24813	=1457	JNZ NOTE ; If not space then this is not the command.
		=1458	;
1361	EB	=1459	XCHG ; Move table pointer to HL.
1362	23	=1460	INX H ; Inc pointer to the LByte of the Vector.
1363	5E	=1461	MOV E,M ; Load LSByte.
1364	23	=1462	INX H ;
1365	56	=1463	MOV D,M ; Load MSByte.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 24

LOC	OBJ	LINE	SOURCE STATEMENT
		=1464	;
		=1465	QTABEX: ;
1366	78	=1466	MOV A,B ; Put the entry number in the acc.
		=1467	;
		=1468	QTABX: ;
1367	B7	=1469	ORA A ; Set flags.
1368	C9	=1470	RET ; Return from <QTAB>
		=1471	;
		=1472	=====
		=1473	;
		=1474	Function : RBULF (Read backwards until line feed) *
		=1475	Inputs : None. *
		=1476	Outputs : HL : Pointer to the character following the *
		=1477	line feed. *
		=1478	Destroys : A,D,E,H,L and Flags *
		=1479	Calls : FDCMP, KEYSTK and DCMF. *
		=1480	;
		=1481	Description : This routine is used to find the new start of *
		=1482	line pointer when the cursor is moved up a line. The *
		=1483	routine starts searching backwards, from the current start *
		=1484	of line pointer, until it finds either a line feed or the *
		=1485	start of text pointer is reached. *
		=1486	;
		=1487	=====
		=1488	;
		=1489	RBULF: ; Start of <RBULF>
1369	2AFCEF	=1490	LHLD SPOINT ; Get start of text pointer.
136C	EB	=1491	XCHG ; Save pointer in DE register.
136D	2A00E8	=1492	LHLD POINT ; Get start of current line pointer.
		=1493	;
1370	2B	=1494	DCX H ; Dec pointer to the <LF> at the end of the
		=1495	Previous line.
1371	CD3012	=1496	CALL FDCMP ; Start of buffer ?
1374	2B	=1497	DCX H ; Dec pointer (NOTE: Does not effect the flags)
1375	CA8813	=1498	JZ RBULFE ; Exit now if start of buffer.
		=1499	;
		=1500	RBULFL: ;
1378	7E	=1501	MOV A,M ; Get character.
1379	FE0A	=1502	CPI LF ; <LF> ?
137B	CA8813	=1503	JZ RBULFE ; Inc pointer and exit.
		=1504	;
137E	2B	=1505	DCX H ; Dec pointer.
137F	CDF223	=1506	CALL KEYSTK ; Check the keyboard
1382	CD43F0	=1507	CALL DCMF ; Check for start of buffer.
1385	D27813	=1508	JNC RBULFL ; Loop until start_of_buffer or <LF>.
		=1509	;
		=1510	RBULFE: ;
1388	23	=1511	INX H ; Inc pointer to after the <LF> and return.
1389	C9	=1512	RET ; End of <RBULF>.
		=1513	;
		=1514	=====
		=1515	;
		=1516	Function : RULF (Read until line feed) *
		=1517	Inputs : None. *
		=1518	Outputs : HL : Pointer to the character following the *
		=1519	line feed. *
		=1520	Destroys : A,D,E,H,L and Flags. *
		=1521	Calls : DCMF and KEYSTK. *
		=1522	;
		=1523	Description : The routine is used to find the new start of *
		=1524	line pointer when the cursor is moved down a line. The *
		=1525	routine, starting from the current start of line pointer, *
		=1526	searches forward until either a line feed is found or the *
		=1527	pointer goes passed the end of the file. *
		=1528	;
		=1529	=====
		=1530	;
		=1531	RULF: ; Start of <RULF> routine.
138A	2AFEEF	=1532	LHLD EPOINT ; Fetch end_of_text pointer
138D	EB	=1533	XCHG ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 25

LOC	OBJ	LINE	SOURCE STATEMENT
138E	2A00E8	=1534	LHLD POINT ; Fetch start_of_line pointer
		=1535 ;	
		=1536 ;	HL = Start of current line pointer.
		=1537 ;	DE = End of text pointer.
		=1538 ;	
		=1539	RULFL: ;
1391	CD43F0	=1540	CALL DCMF ; While Ptr <> End_of_text_ptr do
1394	CAA113	=1541	JZ RULFEX ; begin
		=1542 ;	
1397	7E	=1543	MOV A,M ; Acc := Text(Ptr);
1398	23	=1544	INX H ;
1399	CDF223	=1545	CALL KEYSTK ; Check_keyboard;
139C	FE0A	=1546	CPI LF ;
139E	C29113	=1547	JNZ RULFL ; If Acc = LF then exitwhile
		=1548 ;	end;
		=1549	RULFEX: ;
13A1	C9	=1550	RET ; Return from <RULF> routine.
		=1551 ;	
		=1552 ;	
		=1553 ;	=====
		=1554 ;*	
		=1555 ;*	Function : SETMODE (Set editor mode to next mode) *
		=1556 ;*	Inputs : None. *
		=1557 ;*	Outputs : None. *
		=1558 ;*	Destroys : A and Flags. *
		=1559 ;*	Calls : None, *
		=1560 ;*	
		=1561 ;*	Description : This routine simply changes the states of the *
		=1562 ;*	two mode flags, the 3 possible modes are: *
		=1563 ;*	
		=1564 ;*	Command mode : INSFLG = FALSE, XFLG = FALSE *
		=1565 ;*	Exchange mode : INSFLG = FALSE, XFLG = TRUE *
		=1566 ;*	Insert mode : INSFLG = TRUE, XFLG = FALSE *
		=1567 ;*	
		=1568 ;*	This routine changes from the current mode to the next in *
		=1569 ;*	the sequence. The order of the modes is as follows: *
		=1570 ;*	
		=1571 ;*	Command_mode -> Exchange_mode -> Insert_mode -> Command_mode *
		=1572 ;*	
		=1573 ;	=====
		=1574 ;	
		=1575	SETMODE: ; Start of <SETMODE> routine
13A2	3A15E8	=1576	LDA INSFLG ; Get Insert flag.
13A5	FEFF	=1577	CPI TRUE ;
13A7	CABF13	=1578	JZ GOTOXM ; IF Insert_mode = True then Set command_mode.
		=1579 ;	
13AA	3A16E8	=1580	LDA XFLG ; Get exchange flag.
13AD	FEFF	=1581	CPI TRUE ;
13AF	CACA13	=1582	JZ GOTOIM ; IF Exchange_mode = True then set Insert_mode.
		=1583 ;	
		=1584	GOTOXM: ; Else Mode = Command mode.
		=1585 ;	
		=1586 ;	Therefore Set Exchange_mode.
13B2	3E00	=1587	MVI A,FALSE ;
13B4	3215E8	=1588	STA INSFLG ; Insert_mode_flag = False.
13B7	3EFF	=1589	MVI A,TRUE ;
13B9	3216E8	=1590	STA XFLG ; Exchange_mode_flag = True.
		=1591 ;	
13BC	C3D413	=1592	JMP SMEXIT ; Exit set mode routine.
		=1593 ;	
		=1594	GOTOXM: ; Set command_mode.
13BF	3E00	=1595	MVI A,FALSE ;
13C1	3215E8	=1596	STA INSFLG ; Insert_mode_flag = False.
13C4	3216E8	=1597	STA XFLG ; Exchange_mode_flag = True.
		=1598 ;	
13C7	C3D413	=1599	JMP SMEXIT ; Exit set mode routine.
		=1600 ;	
		=1601	GOTOIM: ; Set insert_mode.
13CA	3EFF	=1602	MVI A,TRUE ;
13CC	3215E8	=1603	STA INSFLG ; Insert_mode_flag = True.

LOC	OBJ	LINE	SOURCE STATEMENT
13CF	3E00	=1604	MVI A,FALSE ;
13D1	3216E8	=1605	STA XFLG ; Exchange_mode_flag = False.
		=1606	;
		=1607	SMEXIT: ; Set mode exit point.
13D4	C9	=1608	RET ; Return from set mode routine.
		=1609 ;	
		=1610 ;	=====
		=1611 ;*	*
		=1612 ;*	Function : UPDATE (Update screen) *
		=1613 ;*	Inputs : None. *
		=1614 ;*	Outputs : None. *
		=1615 ;*	Destroys : A,B,H,L and Flags. *
		=1616 ;*	Calls : DOWNRO *
		=1617 ;*	*
		=1618 ;*	Description : This routine updates the screen by saving the *
		=1619 ;*	the current cursor,line and character positions on the *
		=1620 ;*	stack, moving the cursor down until either the bottom of *
		=1621 ;*	the screen or the end of the file is reached. The routine *
		=1622 ;*	the restores the old cursor, line and character positions *
		=1623 ;*	and sets LCLINE to the last updated line on the screen. *
		=1624 ;*	*
		=1625 ;	=====
		=1626 ;	
		=1627	UPDATE: ; Start of UPDATE Screen Routine.
13D5	2A00E8	=1628	LHLD POINT ;
13D8	E5	=1629	PUSH H ; Save Line pointer.
		=1630	
13D9	2A02E8	=1631	LHLD CHAR ;
13DC	E5	=1632	PUSH H ; Save Character pointer.
		=1633	
13DD	3A05E8	=1634	LDA LINE ;
13E0	F5	=1635	PUSH PSW ; Save Line Position.
		=1636	;
		=1637	UP05: ; Start of Update line loop.
13E1	3A08E8	=1638	LDA EOBLIN ;
13E4	FE16	=1639	CPI LSTLIN ; If EOBLIN < LSTLIN THEN EXIT(Update)
13E6	FAFC13	=1640	JM UP10 ;
		=1641	;
13E9	47	=1642	MOV B,A ; Save EOBLIN in B register.
		=1643	;
13EA	3A0EE8	=1644	LDA LCLINE ; Load lowest correct line number.
13ED	FE16	=1645	CPI LSTLIN ;
13EF	CAFC13	=1646	JZ UP10 ; If Lowest_correct_line = Last Line then
		=1647	Exit (UPDATE).
13F2	B8	=1648	CMP B ;
13F3	FAF613	=1649	JM UP07 ; IF LCLINE < EOBLIN then Update.
		=1650	;
		=1651	UP07: ; Update the current line.
13F6	CD361D	=1652	CALL DOWNRO ; Move cursor down.
13F9	CAE113	=1653	JZ UP05 ; Loop until LCLINE=LSTLIN or EOB=TRUE.
		=1654	UP10: ; End Of Update line loop.
		=1655	;
13FC	F1	=1656	POP PSW ;
13FD	3205E8	=1657	STA LINE ; Restore Line Position.
		=1658	;
1400	E1	=1659	POP H ;
1401	2202E8	=1660	SHLD CHAR ; Restore Character Pointer.
		=1661	;
1404	E1	=1662	POP H ;
1405	2200E8	=1663	SHLD POINT ; Restore Line pointer.
		=1664	;
1408	C9	=1665	RET ; Return from <UPDATE>
		=1666 ;	
		=1667 ;	=====
		=1668 ;*	*
		=1669 ;*	Function : XCHAR (exchange character) *
		=1670 ;*	Inputs : B: Character typed (While in exchange mode) *
		=1671 ;*	Outputs : None *
		=1672 ;*	Destroys : Acc, B, H, L and flags. *
		=1673 ;*	Calls : CLRL, LPRINT and PTOEOL. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 27

LOC	OBJ	LINE	SOURCE STATEMENT
		=1674 ;*	
		=1675 ;*	Description: This routine is called whenever a character
		=1676 ;*	other than a cursor key or the DEL key is typed while in
		=1677 ;*	exchange mode. All that needs to be done is to replace
		=1678 ;*	current character (the character pointed to by the
		=1679 ;*	current character pointer <CHAR>) with the character
		=1680 ;*	typed.
		=1681 ;*	
		=1682 ;*	Note: Carriage returns cannot be inserted into the text
		=1683 ;*	while in exchange mode and lines cannot be extended
		=1684 ;*	in exchange mode.
		=1685 ;*	
		=1686 ;*	=====
		=1687 ;	
		=1688 XCHAR:	
		=1689	MOV A,B ; Xchange CHARACTER.
1409	78	=1690	CPI CR ; Restore character from B register.
140A	FE0D	=1691	JZ XCHR10 ; <CR> ?
140C	CA2D14	=1692	
		=1693	LHLD CHAR ; Get character pointer.
140F	2A02E8	=1694	MOV A,M ; Get character to be replaced.
1412	7E	=1695	CPI LF ; <LF> ? (i.e. End of the line).
1413	FE0A	=1696	JZ XCHR10 ; If Text(Ptr)=LF then exit (cannot extend)
1415	CA2D14	=1697	
		=1698	MOV M,B ; Replace character in text.
1418	70	=1699 XCHR05:	
		=1700	INX H ; Inc pointer to next non null character.
1419	23	=1701	MOV A,M ;
141A	7E	=1702	CPI NULL ;
141B	FE00	=1703	JZ XCHR05 ;
141D	CA1914	=1704	
		=1705	SHLD CHAR ; Save as new character pointer.
1420	2202E8	=1706	CALL CLRL ; Clear to end of line.
1423	CDA826	=1707	MOV A,B ; Get replacement character.
1426	78	=1708	CALL LPRINT ; Print it.
1427	CD2F24	=1709	CALL PTOEOL ; Print_to_End_of_Line.
142A	CDD124	=1710	
		=1711 XCHR10:	
		=1712	RET ; exchange character exit point.
142D	C9	=1713 ;	
		=1714 ;	End of <EDTGEN.A85>
		1715 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 29

LOC	OBJ	LINE	SOURCE STATEMENT
		=1740 ;	=====
		=1741 ;*	
		=1742 ;*	Function : MOVCUR (Move Cursor) *
		=1743 ;*	Inputs : B : Second character in <ESC> sequence *
		=1744 ;*	Outputs : None. *
		=1745 ;*	Destroys : D,E,H,L,A,B and Flags *
		=1746 ;*	Calls : LPRINT, QCURM and one of the following: *
		=1747 ;*	UPROU, DOWNRO, RIGHTR or LEFTRO. *
		=1748 ;*	
		=1749 ;*	Description : This routine overwrites the current cursor *
		=1750 ;*	and then calls one of the four cursor movement routines. *
		=1751 ;*	
		=1752 ;	=====
		=1753 ;	
		=1754	MOVCUR: ; Move cursor routine.
		=1755	;
142E	2A02E8	=1756	LHLD CHAR ; Get Character pointer.
1431	7E	=1757	MOV A,M ; Get current character.
1432	FE09	=1758	CPI TAB ;
1434	CA4114	=1759	JZ PSPACE ; IF char = TAB then Char = Space.
		=1760	;
1437	FE0A	=1761	CPI LF ;
1439	CA4114	=1762	JZ PSPACE ; IF char = Line_Feed then Char = Space.
		=1763	;
143C	FE00	=1764	CPI NULL ; IF char = Null then Char = Space.
143E	C24314	=1765	JNZ DELPRO ;
		=1766	;
		=1767	PSPACE: ;
1441	3E20	=1768	MVI A,SPACE ;
		=1769	DELPRO: ;
1443	CD2F24	=1770	CALL LPRINT ; Overwrite cursor with current character.
		=1771	;
		=1772	;
		=1773	;
1446	78	=1774	MOV A,B ; Restore character typed. e.g. <ESC> 'A'
1447	CD5614	=1775	CALL QCURM ; Recalculate routine address from table
		=1776	;
		=1777	MOVCU5: ;
144A	215014	=1778	LXI H,ENDCM ;
144D	E5	=1779	PUSH H ; Setup Return adr on stack.
144E	EB	=1780	XCHG ;
144F	E9	=1781	PCHL ; Indirect Jump Via HL .
		=1782	;
		=1783	ENDCM: ; Return adr for Cursor move routines.
1450	3E00	=1784	MVI A,FALSE ;
1452	320FE8	=1785	STA ESCFLG ; Reset ESC flag.
1455	C9	=1786	RET ; Return from <MOVCUR>
		=1787 ;	
		=1788 ;	=====
		=1789 ;*	
		=1790 ;*	Function : CURM (? Cursor Movement ?) *
		=1791 ;*	Inputs : A : Second character in <ESC> sequence. *
		=1792 ;*	Outputs : DE : Address of cursor movement routine. *
		=1793 ;*	Destroys : D,E,H,L,A and flags. *
		=1794 ;*	Calls : SEARCH. *
		=1795 ;*	
		=1796 ;*	Description : This routine searches for the address of the *
		=1797 ;*	cursor movement routine. *
		=1798 ;*	
		=1799 ;	=====
		=1800 ;	
		=1801	QCURM: ; Start of ? Cursor move.
1456	212328	=1802	LXI H,CMTAB ; Load pointer with start adr of cursor
		=1803	;
1459	CD5D14	=1804	CALL SEARCH ; movement commands.
145C	C9	=1805	RET ;
		=1806 ;	
		=1807 ;	=====
		=1808 ;*	
		=1809 ;*	Function : SEARCH (Table search) *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 31

LOC	OBJ	LINE	SOURCE STATEMENT
147A	E5	=1880	PUSH H ;
147B	47	=1881	MOV B,A ; Save character in B reg.
147C	213028	=1882	LXI H,CCMOVT ; Get start adr of control code trans table.
		=1883	;
		=1884	CCMOV1: ;
147F	7E	=1885	MOV A,M ; Get control character from table.
1480	FEFF	=1886	CPI ETABLE ; End_of_table ?
1482	CA8E14	=1887	JZ CCMOV2 ; Yes - Then not control code move.
		=1888	;
1485	B8	=1889	CMP B ; Table_char = Input_char ?
1486	CA9314	=1890	JZ CCMOV4 ; Yes - Then it is a control code move.
		=1891	;
1489	23	=1892	INX H ; Table_ptr = Table_ptr + 2
148A	23	=1893	INX H ;
148B	C37F14	=1894	JMP CCMOV1 ; Loop until found or end_of_table.
		=1895	;
		=1896	CCMOV2: ; End_of_table found, => error.
148E	3EFF	=1897	MVI A,ERR ;
1490	C39714	=1898	JMP CCMEX ; Setup Acc = Error and exit.
		=1899	;
		=1900	CCMOV4: ; Table_char = Input_char, => Control_code_move
1493	23	=1901	INX H ; Inc ESC seq character.
1494	46	=1902	MOV B,M ; Move to B reg as returned character.
1495	3E00	=1903	MVI A,OK ; Setup Acc = No error and exit.
		=1904	;
		=1905	CCMEX: ; ? Control code move exit point.
1497	E1	=1906	POP H ;
1498	C9	=1907	RET ;
		=1908	;
		=1909	=====
		=1910	;* ;
		=1911	;* Function : QCHAR (Sets <RELCH> to current col pos * ;
		=1912	;* Inputs : None * ;
		=1913	;* Outputs : None * ;
		=1914	;* Destroys : All * ;
		=1915	;* Calls : FDCMP, MODCOL * ;
		=1916	;* ;
		=1917	;* Description : This routine calculates the current column * ;
		=1918	;* position and stores in <RELCH>, so that if the cursor is * ;
		=1919	;* moved down it can be moved to the same horiz position. * ;
		=1920	;* ;
		=1921	=====
		=1922	;
		=1923	QCHAR: ; Start of <QCHAR>
1499	2A00E8	=1924	LHLD POINT ;
149C	EB	=1925	XCHG ;
149D	2A02E8	=1926	LHLD CHAR ;
		=1927	;
		=1928	HL = Current character pointer.
		=1929	DE = Start of line pointer.
		=1930	;
14A0	0600	=1931	MVI B,00 ; Init column count = zero.
		=1932	QCHAR5: ;
14A2	CD3012	=1933	CALL FDCMP ; Pointer = Char_pointer ?
14A5	CAB814	=1934	JZ QCH10 ; Yes - then Column position found.
		=1935	;
14A8	1A	=1936	LDAX D ; Get current character.
14A9	13	=1937	INX D ; Inc character pointer.
14AA	FE00	=1938	CPI NULL ; Char = NULL ?
14AC	CAA214	=1939	JZ QCHAR5 ; Yes - then ignore.
		=1940	;
14AF	CDF014	=1941	CALL MODCOL ; Else mod col position by character.
14B2	CDF223	=1942	CALL KEYSTK ; Stack incomming characters.
14B5	C3A214	=1943	JMP QCHAR5 ; loop until pointer = Char_pointer.
		=1944	;
		=1945	QCH10: ; Exit point for QCHAR (with B = Col pos)
14B8	78	=1946	MOV A,B ; Store Col position in the Relch byte.
14B9	320AE8	=1947	STA RELCH ;
14BC	C9	=1948	RET ; Return from <QCHAR>
		=1949	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 33

LOC	OBJ	LINE	SOURCE STATEMENT
		=2020 ;*	If the character is a control char then *
		=2021 ;*	COL = COL + 2 *
		=2022 ;*	Else *
		=2023 ;*	COL = COL + 1 +
		=2024 ;*	*
		=2025 ;	=====
		=2026 ;	
		=2027 MODCOL:	; Start of <MODCOL>
		=2028 ;	
		=2029 ;	Check for a TAB character.
		=2030 ;	
14F0	FE09	=2031	CPI TAB ; Char = Tab ?
14F2	CAFF14	=2032	JZ MCTAB ; Yes - the Mod col tab .
		=2033 ;	
		=2034 ;	Check for a control character.
		=2035 ;	
14F5	FE20	=2036	CPI ' ' ;
14F7	F2FB14	=2037	JP GTSPC ; If char < SPACE then
		=2038	; Col = Col + 1
14FA	04	=2039	INR B ;
		=2040 GTSPC:	; Endif
14FB	04	=2041	INR B ; Col = Col + 1
		=2042	;
14FC	C30515	=2043	JMP MCEX ; Exit MODCOL.
		=2044 ;	
		=2045 ;	MODCOL tab routine.
		=2046 ;	
		=2047 MCTAB:	; Jump here if a Char = TAB.
14FF	78	=2048	MOV A,B ; Get col counter.
1500	E6F8	=2049	ANI 11111000B ; Col = Col div 8
1502	C608	=2050	ADI 8 ; Col = Col + 8
1504	47	=2051	MOV B,A ; Return col to B reg
		=2052 MCEX:	;
1505	C9	=2053	RET ; Return from <MODCOL>
		=2054 ;	
		=2055 ;	=====
		=2056 ;*	*
		=2057 ;*	Function : PCUR (Position Cursor) *
		=2058 ;*	Inputs : (CHAR) : Current cursor. *
		=2059 ;*	(POINT) : Current start of line pointer. *
		=2060 ;*	Outputs : None. *
		=2061 ;*	Destroys : A,B,C,D,E,H,L and flags. *
		=2062 ;*	Calls : KEYSTK, EDCMP, MODCOL, GOTO, PRINT, LPRINT. *
		=2063 ;*	*
		=2064 ;*	Description : This routine positions the cursor at the *
		=2065 ;*	correct position on the screen. *
		=2066 ;*	*
		=2067 ;	=====
		=2068 ;	
		=2069 PCUR:	; Position cursor routine.
1506	2A02E8	=2070	LHLD CHAR ; Get current character pointer.
1509	EB	=2071	XCHG ;
150A	2A00E8	=2072	LHLD POINT ; Get start of line pointer.
		=2073	;
150D	0600	=2074	MVI B,00H ; Init col = 0.
		=2075	;
		=2076 NXTCH:	;
150F	CDF223	=2077	CALL KEYSTK ; Stack all incoming keys.
1512	CD3012	=2078	CALL FDCMP ; Compare pointer with char.
1515	CA2A15	=2079	JZ COLSET ;
		=2080	;
1518	7E	=2081	MOV A,M ;
1519	23	=2082	INX H ; Inc to next character.
151A	FE00	=2083	CPI NULL ;
151C	CA0F15	=2084	JZ NXTCH ;
		=2085	;
151F	CDF014	=2086	CALL MODCOL ; Modify COL value with current character.
		=2087	;
1522	78	=2088	MOV A,B ; Fetch col position.
1523	FE4F	=2089	CPI MAXCOL ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 34

LOC	OBJ	LINE	SOURCE STATEMENT
1525	FAOF15	=2090	JM NXTCH ;
		=2091	;
1528	064F	=2092	MVI B,MAXCOL ; If COL > MAXCOL then COL = MAXCOL
		=2093	;
		=2094	COLSET: ;
152A	78	=2095	MOV A,B ; Fetch COL position from the B register.
152B	3204E8	=2096	STA COL ; Set Col position in memory.
		=2097	;
152E	3A15E8	=2098	LDA INSFLG ;
1531	FE00	=2099	CPI FALSE ;
1533	CA4115	=2100	JZ NOTINS ; If Insert_mode then
		=2101	begin ;
1536	CDB426	=2102	CALL GOTO ; Goto(Col,Line);
1539	3E3C	=2103	MVI A,IPROMPT ;
153B	CD9D24	=2104	CALL PRINT ; Write(Chr(Insert_prompt));
153E	C37A15	=2105	JMP PCUREX ; end
		=2106	;
		=2107	NOTINS: ;
1541	3A16E8	=2108	LDA XFLG ;
1544	FE00	=2109	CPI FALSE ;
1546	CA5415	=2110	JZ NOTX ; Elseif Exchange_mode then
		=2111	begin ;
1549	CDB426	=2112	CALL GOTO ; Goto(Col,Line);
154C	3E5E	=2113	MVI A,XPROMPT ;
154E	CD9D24	=2114	CALL PRINT ; Write(Chr(Exchange_prompt))
1551	C37A15	=2115	JMP PCUREX ; end;
		=2116	;
		=2117	NOTX: ; Else
1554	CDB426	=2118	CALL GOTO ; begin
1557	2A02E8	=2119	LHLD CHAR ; Goto(Col,Line);
155A	7E	=2120	MOV A,M ;
155B	FE0A	=2121	CPI LF ;
155D	CA7515	=2122	JZ BLKWS ; If Text(ptr) Not IN [LF , TAB] then
		=2123	;
1560	FE09	=2124	CPI TAB ;
1562	CA7515	=2125	JZ BLKWS ;
		=2126	;
1565	47	=2127	MOV B,A ;
1566	3A04E8	=2128	LDA COL ;
1569	F5	=2129	PUSH PSW ; Save col position
156A	78	=2130	MOV A,B ;
156B	CD2F24	=2131	CALL LPRINT ; Write(chr(text(ptr)))
156E	F1	=2132	POP PSW ;
156F	3204E8	=2133	STA COL ;
1572	C37A15	=2134	JMP PCUREX ; else
		=2135	;
		=2136	BLKWS: ;
1575	3E20	=2137	MVI A,SPACE ; Write(' ');
1577	CD9D24	=2138	CALL PRINT ; end;
		=2139	PCUREX: ;
157A	CDB426	=2140	CALL GOTO ; Goto(Col,Line)
		=2141	;
157D	C9	=2142	RET ; return from <PCUR>
		=2143	;
		=2144	End of <EDTCURSOR.A85> module
		2145	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 35

LOC	OBJ	LINE	SOURCE STATEMENT
		2146 ;	
		2147 ;	Include PAD routine (Inserts nulls in file).
		2148 ;	
		2149 \$INCLUDE(EDTPAD.A85)	
		=2150 ;=====	
		=2151 ;*	
		=2152 ;*	Function : PAD (Pad file with nulls) *
		=2153 ;*	Inputs : Number of nulls to insert into the file. *
		=2154 ;*	Outputs : None *
		=2155 ;*	Destroys : Acc and_flags. *
		=2156 ;*	Calls : KEYSTK and DCMF *
		=2157 ;*	
		=2158 ;*	Description : This routine is used whenever anything has to *
		=2159 ;*	be inserted into the file in memory. *
		=2160 ;*	
		=2161 ;=====	
		=2162 ;	
		=2163 ;	Entry point number 1, Insert PAD routine - inserts a fixed number
		=2164 ;	of nulls into the file.
		=2165 ;	
		=2166 IPAD:	; Insert PAD Entry point.
157E	E5	=2167	PUSH H ;
157F	211400	=2168	LXI H,IPADLEN ; Setup pad length before calling jumping
1582	C39515	=2169	JMP PADLS ; to 'PAD'.
		=2170 ;	
		=2171 ;-----	
		=2172 ;	
		=2173 ;	Entry point number 2, Normal PAD entry point - inserts a variable
		=2174 ;	number of nulls into the file, number passed in A. (byte value)
		=2175 ;	
		=2176 PAD:	
1585	FE00	=2177	CPI 00H ; Pad length = 0 ?
1587	CAE515	=2178	JZ PADRET ; Exit if pad length = 0
		=2179	
158A	E5	=2180	PUSH H ; Save HL on stack
158B	6F	=2181	MOV L,A ;
158C	2600	=2182	MVI H,0 ; HL = Number of nulls to insert
158E	C39515	=2183	JMP PADLS ; Continue with PAD, HL=Pad size.
		=2184 ;	
		=2185 ;-----	
		=2186 ;	
		=2187 ;	Entry point number 3, Paste PAD - inserts a variable number of
		=2188 ;	nulls into the file, number passed in the Cut_buffer_length
		=2189 ;	variable in memory (Word value).
		=2190 ;	
		=2191 CUTPAD:	
1591	E5	=2192	PUSH H ; Save HL on the stack
1592	2AF4E8	=2193	LHLD CUTBL ; Fetch cut buffer length
		=2194 ;	
		=2195 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 37

LOC	OBJ	LINE	SOURCE STATEMENT
		=2266 ;	End of editor PAD routine (EDTPAD.A85)
		2267 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 38

LOC	OBJ	LINE	SOURCE STATEMENT
		2268 ;	
		2269 ;	Include File Pack routine.
		2270 ;	
		2271 \$INCLUDE(EDTPACK.A85)	
		=2272 ;=====	
		=2273 ;*	
		=2274 ;*	Function : Pack.
		=2275 ;*	Inputs : Unpacked file in memory.
		=2276 ;*	Outputs : Packed file in memory.
		=2277 ;*	Destroys : None.
		=2278 ;*	Calls: : KEYSTK and FDCMP.
		=2279 ;*	
		=2280 ;*	Description: This routine is used to pack the text in
		=2281 ;*	memory, (i.e. Remove all the nulls in the file.). It must
		=2282 ;*	do this without destroying the 4 pointers, "SPOINT",
		=2283 ;*	"POINT", "CHAR", "EPOINT" and must therefore check to see
		=2284 ;*	when the forward pointer in the packing routine is equal to
		=2285 ;*	each of the pointers and when it is the value of the lower
		=2286 ;*	pointer should be stored as the new value of that pointer.
		=2287 ;*	
		=2288 ;*	Note(1): The pointer will always occur in the file in the
		=2289 ;*	following order and therefore we can expect one before the
		=2290 ;*	other and speed up the routine.
		=2291 ;*	
		=2292 ;*	1st : SPOINT, Start_of_text_pointer.
		=2293 ;*	2nd : POINT , Start_of_current_line_pointer.
		=2294 ;*	3rd : CHAR , Current_character_pointer.
		=2295 ;*	4th : EPOINT, End_of_text_pointer.
		=2296 ;*	
		=2297 ;*	Note(2): The keyboard buffering routine <KEYSTK> is called
		=2298 ;*	regularly to check that no characters are lost.
		=2299 ;*	
		=2300 ;=====	
		=2301 ;	
		=2302 PACK:	; Pack the File (remove Nulls)
		=2303 ;	
15E6	F5	=2304	PUSH PSW ;
15E7	C5	=2305	PUSH B ; Save all registers.
15E8	D5	=2306	PUSH D ;
15E9	E5	=2307	PUSH H ;
		=2308 ;	
		=2309 ;	-----
		=2310 ;	
		=2311 ;	a) Look for Start_of_line_pointer "POINT".
		=2312 ;	
		=2313 ;	
15EA	2AFCE	=2314	LHLD SPOINT ;
15ED	EB	=2315	XCHG ;
15EE	2A00E8	=2316	LHLD POINT ;
15F1	42	=2317	MOV B,D ;
15F2	4B	=2318	MOV C,E ; Move start_of_text_pointer to BC.
		=2319 ;	
		=2320 ;	Pointers setup as follows.
		=2321 ;	
		=2322 ;	BC = Start_of_text_pointer. (Used as packed text output pointer)
		=2323 ;	DE = Start_of_text_pointer. (Used as text pointer)
		=2324 ;	HL = Start_of_current_line_pointer (1st pointer in file).
		=2325 ;	
		=2326 PACK05:	
15F3	CD3012	=2327	CALL FDCMP ; Compare DE and HL.(POINT = SPOINT ?).
15F6	CA0B16	=2328	JZ POINTF ; If equal then POINT found.
		=2329 ;	
15F9	7B	=2330	MOV A,E ;
15FA	E61F	=2331	ANI MASK ; If DE MOD 32 = 0 then
15FC	CCF223	=2332	CZ KEYSTK ; Pack_the_file.
		=2333 ;	
15FF	1A	=2334	LDAX D ; Load char from memory.
1600	13	=2335	INX D ; Inc text pointer.
1601	FE00	=2336	CPI NULL ; Char = Null ?
1603	CAF315	=2337	JZ PACK05 ; Read until char <> Null.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 39

LOC	OBJ	LINE	SOURCE STATEMENT		
		=2338			;
1606	02	=2339	STAX	B	; Save in lower pointer.
1607	03	=2340	INX	B	; Inc lower pointer.
1608	C3F315	=2341	JMP	PACK05	; Loop and move the next character.
		=2342			;
		=2343	POINTF:		; Point found.
		=2344			;
160B	60	=2345	MOV	H,B	;
160C	69	=2346	MOV	L,C	; Move lower pointer to HL.
160D	2200E8	=2347	SHLD	POINT	; Save as start_of_line_pointer.
1610	2A02E8	=2348	LHLD	CHAR	; Get value of next pointer to look for.
		=2349			;
		=2350	-----		
		=2351			;
		=2352			b) Look for Character_pointer "CHAR".
		=2353			;
		=2354			;
		=2355	PACK10:		;
1613	CD3012	=2356	CALL	FDCMP	; Text_pointer = CHAR ?
1616	CA2B16	=2357	JZ	CHARF	; Yes - then char_ptr found.
		=2358			;
1619	7B	=2359	MOV	A,E	;
161A	E61F	=2360	ANI	MASK	; If DE MOD 32 = 0 then
161C	CCF223	=2361	CZ	KEYSTK	; Pack_the_file.
		=2362			;
161F	1A	=2363	LDAX	D	; Get Text(text_ptr)
1620	13	=2364	INX	D	; Inc text_ptr
1621	FE00	=2365	CPI	NULL	; Char = Null ?
1623	CA1316	=2366	JZ	PACK10	; Yes then do not save in packed file.
		=2367			;
1626	02	=2368	STAX	B	; Store char in packed file.
1627	03	=2369	INX	B	; inc Lower_ptr
1628	C31316	=2370	JMP	PACK10	; Loop until CHAR found.
		=2371			;
		=2372	CHARF:		; Character_ptr found.
		=2373			;
162B	60	=2374	MOV	H,B	;
162C	69	=2375	MOV	L,C	; Move lower_ptr to HL.
162D	2202E8	=2376	SHLD	CHAR	; Save lower_ptr as new CHAR.
1630	2AFEEF	=2377	LHLD	EPOINT	; Get End_of_text_ptr.
		=2378			;
		=2379			;
		=2380	-----		
		=2381			;
		=2382			c) Look for End_of_text_pointer "EPOINT".
		=2383			;
		=2384	PACK15:		;
1633	CD3012	=2385	CALL	FDCMP	; Text_ptr = End_ptr ?
1636	CA4B16	=2386	JZ	EPTRFO	; Yes - Then End_pointer found.
		=2387			;
1639	7B	=2388	MOV	A,E	;
163A	E61F	=2389	ANI	MASK	; If DE MOD 32 = 0 then
163C	CCF223	=2390	CZ	KEYSTK	; Pack_the_file.
		=2391			;
163F	1A	=2392	LDAX	D	; Get Text(Text_ptr)
1640	13	=2393	INX	D	; Inc text_ptr
1641	FE00	=2394	CPI	NULL	; Char = Null ?
1643	CA3316	=2395	JZ	PACK15	; Yes - then do not save in packed file.
		=2396			;
1646	02	=2397	STAX	B	; Save in packed file.
1647	03	=2398	INX	B	; Inc lower_ptr
1648	C33316	=2399	JMP	PACK15	; Loop until Lower_ptr = EPOINT.
		=2400			;
		=2401	EPTRFO:		; Text_ptr = lower_ptr
164B	1A	=2402	LDAX	D	; Get last character from unpacked text.
164C	02	=2403	STAX	B	; Save as last character in packed text.
		=2404			;
164D	60	=2405	MOV	H,B	;
164E	69	=2406	MOV	L,C	; Move lower_ptr to HL.
164F	22FEEF	=2407	SHLD	EPOINT	; Save lower_ptr as new End_of_text_ptr.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 40

LOC	OBJ	LINE	SOURCE STATEMENT
		=2408	
1652	3E00	=2409	MVI A,00H
1654	320BE8	=2410	STA NULLS
		=2411	
1657	E1	=2412	POP H
1658	D1	=2413	POP D
1659	C1	=2414	POP B
165A	F1	=2415	POP PSW
		=2416	
165B	C9	=2417	RET
		=2418	
		=2419	End of file "EDTPACK.A85".
		2420	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 41

LOC	OBJ	LINE	SOURCE STATEMENT
		2421 ;	
		2422 ;	Include Exit/Quit Routine.
		2423 ;	
		2424 \$INCLUDE(EDTEXT.A85)	
		=2425 ;=====	
		=2426 ;*	*
		=2427 ;*	Function : QUIT and SAVE Routine. *
		=2428 ;*	Inputs : None. *
		=2429 ;*	Outputs : None. *
		=2430 ;*	Destroys : None. *
		=2431 ;*	Calls : PACK, HOME and CLRS. *
		=2432 ;*	*
		=2433 ;*	Description: This routine is the Exit point for the editor *
		=2434 ;*	and returns the user to the SABus Monitor via and HALT. *
		=2435 ;*	*
		=2436 ;=====	
		=2437 ;	
		=2438 QUITRO:	
		=2439 EXITRO:	; Quit and Exit Do the same thing.
165C	CDE615	=2440 CALL PACK	; Pack the file before returning to the
		=2441	; Monitor.
165F	CDCA26	=2442 CALL HOME	;
1662	CDAE26	=2443 CALL CLRS	; Home and clear the screen.
1665	76	=2444 HLT	; Return to SABus monitor V3.1 via
		=2445	; the HLT trap.
		=2446 ;	
		=2447 ;	End of QUIT and EXIT routines.
		2448 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 42

LOC	OBJ	LINE	SOURCE STATEMENT
		2449 ;	
		2450 ;	Include Length routine.
		2451 ;	
		2452	\$INCLUDE(EDTLEN.A85)
		=2453 ;	-----;
		=2454 ;	;
		=2455 ;	Length module for the UCT SABus editor. ;
		=2456 ;	----- ;
		=2457 ;	;
		=2458 ;	This module contains (in order) the following subroutines: ;
		=2459 ;	;
		=2460 ;	;
		=2461 ;	a) LENROU - Prints the length of the text file in memory. ;
		=2462 ;	b) VERROU - Prints the editors version information. ;
		=2463 ;	;
		=2464 ;	-----;
		=2465 ;	
		=2466	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=2467 ;	=====
		=2468 ;*	*
		=2469 ;*	Function : LENROU (Length routine) *
		=2470 ;*	Inputs : None *
		=2471 ;*	Outputs : None *
		=2472 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=2473 ;*	Calls : PACK, CLRL, STRING, DPRINT and DCMF. *
		=2474 ;*	*
		=2475 ;*	Description: This routine prints the length of the file in *
		=2476 ;*	memory in bytes and lines. It calculates the length in *
		=2477 ;*	bytes by simply subtracting the Start_of_text pointer from *
		=2478 ;*	the End_of_text Pointer. *
		=2479 ;*	*
		=2480 ;*	The number of lines is calculated by counting the number of *
		=2481 ;*	<LF>'s between the above limits. *
		=2482 ;*	*
		=2483 ;	=====
		=2484 ;	
		=2485	LENROU: ;
1666	CDE615	=2486	CALL PACK ; Pack the file before printing its
		=2487	; length. (Otherwise you will count nulls)
1669	CDA826	=2488	CALL CLRL ; Clear current line. (The command line)
166C	21892B	=2489	LXI H,LM1 ;
166F	CDE125	=2490	CALL STRING ; Print the 1st part of the Length of file
		=2491	; message.
1672	2AFCEF	=2492	LHLD SPOINT ;
1675	EB	=2493	XCHG ;
1676	2AFEEF	=2494	LHLD EPOINT ; Setup DE and HL as pointers
		=2495 ;	
		=2496 ;	DE = Start of text pointer.
		=2497 ;	HL = End of text pointer.
		=2498 ;	
1679	CD911F	=2499	CALL SUB16 ; HL = HL - DE ; (* 16 bit subtract *)
		=2500 ;	
		=2501 ;	HL = End_of_text_pointer - Start_of_text_pointer.
		=2502 ;	
167C	E5	=2503	PUSH H ; Save length on stack.
167D	CDFA25	=2504	CALL PRINTD ; Print the number of bytes in the file
		=2505	; as a decimal number.
		=2506	;
1680	21942B	=2507	LXI H,LM2 ;
1683	CDE125	=2508	CALL STRING ; Print the 2nd part of the Length of
		=2509	; file message.
		=2510	;
1686	2AFEEF	=2511	LHLD EPOINT ; Fetch End_of_text pointer
1689	EB	=2512	XCHG ;
168A	2AF0E8	=2513	LHLD MAXTXT ; Fetch Maximum text pointer
		=2514 ;	
		=2515 ;	HL = Maximum text pointer.
		=2516 ;	DE = End_of_text pointer.
		=2517 ;	
168D	CD911F	=2518	CALL SUB16 ; 16 bit subtraction HL = HL - DE
		=2519 ;	
		=2520 ;	HL(Space left) = HL(Maximum text pointer) - DE(End_of_text pointer)
		=2521 ;	;
1690	CDFA25	=2522	CALL PRINTD ; Print the number of bytes left.
		=2523	;
1693	219D2B	=2524	LXI H,LM3 ;
1696	CDE125	=2525	CALL STRING ; Print the 3rd part of the length message
		=2526 ;	
		=2527 ;	Calculate the number of lines in the file by counting line feeds.
		=2528 ;	
1699	010000	=2529	LXI B,0000H ; Init number of lines = zero
169C	E1	=2530	POP H ; Pop len off the stack
169D	7C	=2531	MOV A,H ;
169E	B5	=2532	ORA L ; HL = Zero ?
169F	CAB716	=2533	JZ LEN15 ; Yes - then leave number of lines = 0.
		=2534 ;	
		=2535 ;	-----
		=2536 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 44

LOC	OBJ	LINE	SOURCE STATEMENT
		=2537 ;	If the len # zero then count the number of lines.
		=2538 ;	
16A2	2AFEEF	=2539	LHLD EPOINT ; Fetch End_of_text pointer
16A5	EB	=2540	XCHG ;
16A6	2AFCEF	=2541	LHLD SPOINT ; Fetch Start_of_text pointer
		=2542 ;	
		=2543 ;	HL = Start_of_text_pointer.
		=2544 ;	DE = End_of_text_pointer.
		=2545 ;	
		=2546 LEN05:	
16A9	7E	=2547	MOV A,M ; get character from file.
16AA	FE0A	=2548	CPI LF ; <LF> ?
16AC	C2B016	=2549	JNZ LEN10 ; No - then do not inc line feed counter.
		=2550	
16AF	03	=2551	INX B ;
		=2552 LEN10:	
16B0	CD43F0	=2553	CALL DCMP ; Compare Pointer with end_of_text_pointer.
16B3	23	=2554	INX H ; Inc text pointer.
16B4	DAA916	=2555	JC LEN05 ; If Pointer < End_pointer then loop.
		=2556	
		=2557 LEN15:	
16B7	60	=2558	MOV H,B ;
16B8	69	=2559	MOV L,C ; Move line count from BC to HL.
16B9	CDFA25	=2560	CALL PRINTD ; Print the number of lines in the file as
		=2561	; a decimal number.
16BC	21AD2B	=2562	LXI H,LM4 ; Print the last part on the len message.
16BF	CDE125	=2563	CALL STRING ;
16C2	C9	=2564	RET ; Return from <LENROU>
		=2565 ;	
		=2566 ;=====	
		=2567 ;*	*
		=2568 ;* Function : VERROU (version routine)	*
		=2569 ;* Inputs : None	*
		=2570 ;* Outputs : None	*
		=2571 ;* Destroys : H, L, A and flags.	*
		=2572 ;* Calls : ERRSTR	*
		=2573 ;*	*
		=2574 ;* Description : This routine outputs the editor version	*
		=2575 ;* information to the console, e.g. 'SABus editor V1.1:0'	*
		=2576 ;*	*
		=2577 ;=====	
		=2578 ;	
		=2579 VERROU:	; Start of <VERROU>
16C3	214F2E	=2580	LXI H,VERMSG ;
16C6	CD8723	=2581	CALL ERRSTR ; Output version message
16C9	C9	=2582	RET ;
		=2583 ;	
		=2584 ;	End of <EDTLEN> module.
		2585 SEJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 45

LOC	OBJ	LINE	SOURCE STATEMENT
		2586 ;	
		2587 ;	Include New routine.
		2588 ;	
		2589 \$INCLUDE(EDTNEW.A85)	
		=2590 ;=====	
		=2591 ;*	
		=2592 ;*	Function : NEW (Clears the file in memory.)
		=2593 ;*	Inputs : None.
		=2594 ;*	Outputs : None.
		=2595 ;*	Destroys : Entire file in memory and all registers.
		=2596 ;*	Calls : BLINE, CLRL, STRING, GETKEY, CONCAP, HOME, CLRS
		=2597 ;*	and QINIT.
		=2598 ;*	
		=2599 ;*	Description: This Routine clears the file in memory by
		=2600 ;*	resetting the pointers to the start_of_text pointer.
		=2601 ;*	
		=2602 ;*	NOTE: This routine does ask the user to confirm the command
		=2603 ;*	("clear file in memory (Y/N) ?>")
		=2604 ;*	
		=2605 ;=====	
		=2606 ;	
		=2607 NEWROU:	; Start Of NEW routine.
16CA	CD9D26	=2608	CALL BLINE ;
16CD	CDA826	=2609	CALL CLRL ; Clear command line.
16D0	216A2B	=2610	LXI H,CONFRM ;
16D3	CDE125	=2611	CALL STRING ; Print comfirm deletion prompt.
16D6	CD9323	=2612	CALL GETKEY ; Read a key from the keyboard.
16D9	CDB811	=2613	CALL CONCAP ; Convert to upper case.
16DC	FE59	=2614	CPI 'Y' ;
16DE	C21317	=2615	JNZ NEWREX ; If Not 'Y' then exit and do nothing.
		=2616	; Else
16E1	2AFCEF	=2617	LHLD SPOINT ; EOT_pointer = SOT_pointer.
16E4	22FEEF	=2618	SHLD EPOINT ; Character pointer = SOT
16E7	2202E8	=2619	SHLD CHAR ; Line pointer = SOT
16EA	2200E8	=2620	SHLD POINT ; TEXT(SOT) = LF
16ED	360A	=2621	MVI M,LF ;
16EF	3E00	=2622	MVI A,0 ; Null count = 0
16F1	320BE8	=2623	STA NULLS ; Relative character pos = 0
16F4	320AE8	=2624	STA RELCH ; Lowest correct line = 0
16F7	320EE8	=2625	STA LCLINE ; File_name CRC = 0
16FA	32E7EF	=2626	STA FNCRC ;
16FD	3E17	=2627	MVI A,LSTLIN+1 ;
16FF	3208E8	=2628	STA EOBLIN ; EOB_line = OFF SCREEN
1702	3E00	=2629	MVI A,FALSE ;
1704	3211E8	=2630	STA RESFLG ; Reset cursor restore flag
1707	CDCA26	=2631	CALL HOME ;
170A	CDAE26	=2632	CALL CLRS ; Clear the screen
170D	CDEA12	=2633	CALL QINIT ; Kill all characters in Queue
1710	C31917	=2634	JMP NEWRET ;
		=2635	; New routine exit.
1713	CD9D26	=2637	CALL BLINE ;
1716	CDA826	=2638	CALL CLRL ; Clear prompt from command line.
		=2639 NEWRET:	; Return from <NEWROU>
1719	C9	=2640	RET ;
		=2641 ;	
		=2642 ;	End of <EDTNEW.A85> module
		2643 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 47

LOC	OBJ	LINE	SOURCE STATEMENT
		=2666 ;	
		=2667 ;	
		=2668 ;*	
		=2669 ;*	Function : FINDRO (Find Routine)
		=2670 ;*	Inputs : Command buffer, containing a find command.
		=2671 ;*	Outputs : None.
		=2672 ;*	Destroys : A,B,C,D,E,H,L and flags.
		=2673 ;*	Calls : PACK, ERRSTR, MOVSTR and SEAFS.
		=2674 ;*	
		=2675 ;*	Description : This routine parses the find command and sets
		=2676 ;*	up the parameters for the SEAFS routine.
		=2677 ;*	
		=2678 ;	
		=2679 ;	
		=2680 FINDRO:	
171A	CDE615	=2681	CALL PACK ; Pack the file before looking for string.
171D	2117E8	=2682	LXI H,BUF ;
		=2683 ;	
		=2684 ;	
		=2685 ;	
		=2686 ;	Skip until not tab or blank
		=2687 ;	
		=2688 FIND05:	; Repeat
1720	23	=2689	INX H ; Ptr := Ptr + 1 ;
1721	7E	=2690	MOV A,M ;
1722	FE20	=2691	CPI SPACE ;
1724	CA2017	=2692	JZ FIND05 ;
		=2693 ;	
1727	FE09	=2694	CPI TAB ;
1729	CA2017	=2695	JZ FIND05 ; Until Command_string [Ptr] Not in
		=2696	; [<SPACE> , <TAB>]
		=2697 ;	
172C	FE04	=2698	CPI EOT ; <EOT> ?
172E	C24A17	=2699	JNZ NOTJF ; No - then take character as delimiter.
		=2700 ;	
		=2701 ;	
		=2702 ;	
		=2703 ;	First character (non tab or blank) is the EOT marker, the command
		=2704 ;	is therefore simply an "F". Check to see if the string buffer
		=2705 ;	contains substitute or a find command strings. If the Substitute
		=2706 ;	buffers in tack flag <SUBBIT> false then the find command can
		=2707 ;	assume that the search string is already in the first string buffer.
		=2708 ;	
1731	3A12E8	=2709	LDA SUBBIT ; Load status of buffers.
1734	FEFF	=2710	CPI TRUE ; If Sub_buffers_intact then ERROR.
1736	CA4117	=2711	JZ FINDBE ;
		=2712 ;	
		=2713 ;	
		=2714 ;	
		=2715 ;	Check that there is a non zero length string in the buffer.
		=2716 ;	
1739	3A54EC	=2717	LDA SEASL ; Is there a string in the Find buffer ?
173C	FE00	=2718	CPI 0 ;
173E	C25917	=2719	JNZ FCONT ; Continue with search (buffer and length
		=2720	setup.
		=2721 ;	
		=2722 ;	
		=2723 ;	
		=2724 ;	Zero length string or substitute command in buffes.
		=2725 ;	
		=2726 FINDBE:	
1741	214F2B	=2727	LXI H,NOSTRM ;
1744	CD8723	=2728	CALL ERRSTR ; Print error message.
1747	C36517	=2729	JMP FINDEX ; Exit find routine.
		=2730 ;	
		=2731 ;	
		=2732 ;	
		=2733 ;	The first character following the "F" is not an EOT marker, it
		=2734 ;	must be the string delimiter. Copy the string from the command
		=2735 ;	line into the first string buffer (Search string).

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 48

LOC	OBJ	LINE	SOURCE STATEMENT
		=2736 ;	
		=2737 NOTJF:	
174A	1136EC	=2738 LXI	D,SEABUF ; Set up start of buffer adr for MOVSTR
174D	0600	=2739 MVI	B,NULL ; Set delimiter = NULL, i.e. no delimiter.
174F	CD8A17	=2740 CALL	MOVSTR ; Copy string on command line to string
		=2741	; buffer number 1.
1752	C25F17	=2742 JNZ	FINDER ; If 'MOVSTR' Returns error the goto find_error
		=2743	
1755	79	=2744 MOV	A,C ; Move string length to the Aregister.
1756	3254EC	=2745 STA	SEASL ; Save character count
		=2746 FCONT:	
1759	CDE117	=2747 CALL	SEAFS ; Start search
175C	C36517	=2748 JMP	FINDEX ; Exit from the find routine
		=2749 FINDER:	
175F	21C22B	=2750 LXI	H,STRER1 ;
1762	CD8723	=2751 CALL	ERRSTR ;
		=2752 FINDEX:	
1765	C9	=2753 RET	; < End of find routine >
		=2754 ;	
		=2755 ;=====	
		=2756 ;*	
		=2757 ;* Function : LOOKFS (Look for string)	*
		=2758 ;* Inputs : (CHAR) : Start address for the search.	*
		=2759 ;* (SEABUF) : String to search for.	*
		=2760 ;* Outputs : None.	*
		=2761 ;* Destroys : A,D,E,H,L and flags.	*
		=2762 ;* Calls : DCMP, KEYSTK, STRCMP and FDCMP.	*
		=2763 ;*	*
		=2764 ;* Description : This routine scans the source file, starting	*
		=2765 ;* at the current character position, CHAR, looking for the	*
		=2766 ;* string stored in the SEABUF buffer.	*
		=2767 ;*	*
		=2768 ;=====	
		=2769 ;	
		=2770 LOOKFS:	; Look For String (Starting at CHAR)
1766	2AFEEF	=2771 LHLD	EPOINT ;
1769	EB	=2772 XCHG	;
176A	2A02E8	=2773 LHLD	CHAR ; Setup pointers:
		=2774	; HL = Position in memory to start search.
		=2775	; DE = End of text pointer.
		=2776	
176D	CD43F0	=2777 CALL	DCMP ; Compare pointers.
1770	D28617	=2778 JNC	LFS02 ; If Char_Ptr >= EOT_Ptr then Return.
		=2779	
		=2780 LOKFS:	
		=2781	
1773	CDF223	=2781 CALL	KEYSTK ; Check keyboard for keypress.
1776	0136EC	=2782 LXI	B,SEABUF ; Load start adr of the string to be found.
1779	CD2818	=2783 CALL	STRCMP ; Compare string with memory starting at
		=2784	; the location pointed to by HL.
177C	CA8917	=2785 JZ	LFS05 ; If equal (zero returned) then exit.
		=2786	
177F	23	=2787 INX	H ; Inc pointer to next byte in memory and repeat.
1780	CD3012	=2788 CALL	FDCMP ; Loop until string found or end of file.
1783	C27317	=2789 JNZ	LOKFS ;
		=2790	
		=2791 LFS02:	
1786	3EFF	=2792 MVI	A,OFFH ; If string not found then return nonzero.
1788	B7	=2793 ORA	A ; Clear zero flag.
		=2794 LFS05:	
1789	C9	=2795 RET	; End of <LOOKFS> routine.
		=2796 ;	
		=2797 ;=====	
		=2798 ;*	
		=2799 ;* Function : MOVSTR (Move String)	*
		=2800 ;* Inputs : M : Pointer to command line.	*
		=2801 ;* Outputs : None.	*
		=2802 ;* Destroys : A,B,C,D,E,H,L and flags.	*
		=2803 ;* Calls : None.	*
		=2804 ;*	*
		=2805 ;* Description : This routine copies the search string from	*

LOC	OBJ	LINE	SOURCE STATEMENT
		=2806 ;*	command line, into the search buffer. *
		=2807 ;*	*
		=2808 ;	=====
		=2809 ;	
		=2810	MOVSTR:
178A	OE00	=2811	MVI C,00H ; Reset character counter.
178C	78	=2812	MOV A,B ; Get delimiter.
178D	FE00	=2813	CPI NULL ; Delimiter set ?
178F	C29417	=2814	JNZ DELMOK ; If delimiter set (Not zero) then continue.
1792	46	=2815	MOV B,M ; Else get the next character as the delimiter.
1793	23	=2816	INX H ; Inc pointer past 1st delimiter.
		=2817	DELMOK: ;
1794	7E	=2818	MOV A,M ; Get character from text.
1795	FE04	=2819	CPI EOT ; <EOT> ?
1797	CAA517	=2820	JZ STRMER ; Yes - then, end of string.
		=2821	;
179A	B8	=2822	CMP B ; Character = Delimiter ? (* From B register *)
179B	CAA917	=2823	JZ SDELFO ; Yes - then 2nd delimiter found.
		=2824	;
179E	12	=2825	STAX D ; If not <EOT> or Delimiter then move character
		=2826	; to output buffer.
179F	13	=2827	INX D ; Inc output buffer pointer.
17A0	23	=2828	INX H ; Inc Text pointer.
17A1	0C	=2829	INR C ; Inc character counter.
17A2	C39417	=2830	JMP DELMOK ; Loop.
		=2831	;
		=2832	STRMER: ; <EOT> found before 2nd Delimiter.
17A5	2B	=2833	DCX H ; Dec pointer to point at <EOT>
17A6	3EFF	=2834	MVI A,ERR ; Setup error return.
17A8	B7	=2835	ORA A ;
		=2836	SDELFO: ;
17A9	23	=2837	INX H ; Inc pointer past the delimiter.
17AA	C9	=2838	RET ; Return from <MOVSTR>
		=2839 ;	
		=2840 ;	=====
		=2841 ;*	*
		=2842 ;*	Function : QFINDR (Quote Find routine) *
		=2843 ;*	Inputs : Command line. *
		=2844 ;*	Outputs : None. *
		=2845 ;*	Destroys : A,B,C,D,E,H,L and flags. *
		=2846 ;*	Calls : PACK, ERRSTR, MOVSTR and SEAFS. *
		=2847 ;*	*
		=2848 ;*	Description : This routine parses the find command and sets *
		=2849 ;*	up the parameters for the SEAFS routine. *
		=2850 ;*	*
		=2851 ;	=====
		=2852 ;	
		=2853	QFINDR: ; Start of quote find routine
		=2854	;
17AB	CDE615	=2855	CALL PACK ; Pack the file before search.
17AE	2117E8	=2856	LXI H,BUF ; Load pointer to the start of the command line.
17B1	23	=2857	INX H ; Increment pointer past the quote.
17B2	7E	=2858	MOV A,M ; Load char after the Quote.
17B3	FE04	=2859	CPI EOT ; End_Of_Text ?
17B5	C2D117	=2860	JNZ QF10 ; No - then copy the search string from
		=2861	; the command line into the 1st buffer.
		=2862 ;	
		=2863 ;	-----
		=2864 ;	
		=2865 ;	First character is the EOT marker, the command is therefore simply
		=2866 ;	a "". Check to see if the string buffer contains substitute or
		=2867 ;	a find command strings. If the Substitute buffers in tack flag
		=2868 ;	<SUBBIT> false then the find command can assume that the search
		=2869 ;	string is already in the first string buffer.
		=2870 ;	
17B8	3A12E8	=2871	LDA SUBBIT ; Load status of buffers.
17BB	FEFF	=2872	CPI TRUE ; If Sub_buffers_intact then ERROR.
17BD	CAC817	=2873	JZ QFERR ;
		=2874 ;	
		=2875 ;	-----

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 50

LOC	OBJ	LINE	SOURCE STATEMENT
		=2876 ;	
		=2877 ;	Check that there is a non zero length string in the buffer.
		=2878 ;	
17C0	3A54EC	=2879	LDA SEASL ; Is there a string in the Find buffer ?
17C3	FE00	=2880	CPI 0 ;
17C5	C2DD17	=2881	JNZ QFCONT ; Continie with search (buffer and length
		=2882	; setup.
		=2883 ;	
		=2884 ;	-----
		=2885 ;	
		=2886 ;	Zero length string or substitute command in buffes.
		=2887 ;	
		=2888	QFERR: ;
17C8	214F2B	=2889	LXI H,NOSTRM ;
17CB	CD8723	=2890	CALL ERRSTR ; Print error message.
17CE	C3E017	=2891	JMP QFEXIT ; Exit find routine.
		=2892 ;	
		=2893 ;	-----
		=2894 ;	
		=2895 ;	Copy the search string from the command line to the first string
		=2896 ;	buffer using the quote character as a delimiter. As the quote
		=2897 ;	find command does not need a second delimiter, i.e. "FRED" is
		=2898 ;	a valid command, the error return from the move string command
		=2899 ;	is ignored. This is because the move string command will only
		=2900 ;	produce an error if the EOT marker is encountered on the command
		=2901 ;	line before the second delimiter.
		=2902 ;	
		=2903	QF10: ;
17D1	1136EC	=2904	LXI D,SEABUF ; Load pointer to the start of the 1st string
		=2905	; buffer.
17D4	0627	=2906	MVI B,QUOTE ; Setup delimiter for the string move routine
		=2907	; as the end_of_text character.
17D6	CD8A17	=2908	CALL MOVSTR ; Copy the string from the command line into
		=2909	; the string buffer.
17D9	79	=2910	MOV A,C ; Get the character count.
17DA	3254EC	=2911	STA SEASL ; Save length of string to search for.
		=2912	QFCONT: ;
17DD	CDE117	=2913	CALL SEAFS ; Call string find routine.
		=2914	QFEXIT: ;
17E0	C9	=2915	RET ; Return from the <QFINDER> routine.
		=2916 ;	
		=2917 ;	=====
		=2918 ;*	*
		=2919 ;*	Function : SEAFS (Search for string) *
		=2920 ;*	Inputs : (CHAR) : Current character pointer. *
		=2921 ;*	(SEABUF) : Search buffer *
		=2922 ;*	Outputs : None. *
		=2923 ;*	Destroys : A,B,C,D,E,H,L and flags. *
		=2924 ;*	Calls : LOOKFS, CLRL, ERRSTR, GOBACK, ONSCN, GOTO *
		=2925 ;*	and NEWSN. *
		=2926 ;*	*
		=2927 ;*	Description : This routine calls the LOOKFS routine to *
		=2928 ;*	look for the string and then does the tidying up. *
		=2929 ;*	*
		=2930 ;	=====
		=2931 ;	
		=2932	SEAFS: ; Start of the <SEACH> routine.
17E1	3E00	=2933	MVI A,FALSE ;
17E3	3212E8	=2934	STA SUBBIT ; Mark substitute buffers as having been
		=2935	; destroyed.
		=2936	;
17E6	2A02E8	=2937	LHLD CHAR ;
17E9	23	=2938	INX H ; Inc character pointer so that the next string
17EA	2202E8	=2939	SHLD CHAR ; will be found and not the same one as was
		=2940	; found by the previous search.
		=2941	;
17ED	CD6617	=2942	CALL LOOKFS ; Look for the string (Z=1 - Found)
17F0	CA0618	=2943	JZ SEA05 ; String found.
		=2944 ;	
		=2945 ;	-----

LOC	OBJ	LINE	SOURCE STATEMENT
		=2946 ;	
		=2947 ;	String not found - restore original character pointer and
		=2948 ;	print an error message.
		=2949 ;	
17F3	2A02E8	=2950	LHLD CHAR ;
17F6	2B	=2951	DCX H ; Decrement character pointer.
17F7	2202E8	=2952	SHLD CHAR ;
		=2953	
17FA	CDA826	=2954	CALL CLRL ; Clear the command line
17FD	21DE2B	=2955	LXI H,NOSUCH ;
1800	CD8723	=2956	CALL ERRSTR ; Print error message.
1803	C32718	=2957	JMP SEARET ; Exit from search routine.
		=2958 ;	
		=2959 ;	-----
		=2960 ;	
		=2961 ;	String found.
		=2962 ;	
		=2963	SEA05: ;
1806	EB	=2964	XCHG ; Save pointer to string.
1807	2A00E8	=2965	LHLD POINT ; Load pointer to the start of the line.
180A	E5	=2966	PUSH H ; Save on stack.
180B	EB	=2967	XCHG ;
180C	CD3612	=2968	CALL GOBACK ; Update CHAR and find POINT.
180F	E1	=2969	POP H ; Restore pointer to current line.
1810	CD9512	=2970	CALL ONSCN ; Is the string found on the screen ?
1813	FA2418	=2971	JM NOTOS ; If negative return then line not on screen.
		=2972 ;	
		=2973 ;	-----
		=2974 ;	
		=2975 ;	The line containing the search string is on the screen. Line
		=2976 ;	position given by the line number returned by the <ONSCN>
		=2977 ;	routine. Simply use a direct cursor positioning command to
		=2978 ;	place the cursor in the correct place.
		=2979 ;	
1816	3205E8	=2980	STA LINE ; Update line position.
1819	CDB426	=2981	CALL GOTO ; Move cursor.
181C	3E00	=2982	MVI A,FALSE ;
181E	3211E8	=2983	STA RESFLG ; Reset restore cursor flag.
1821	C32718	=2984	JMP SEARET ; Exit search routine.
		=2985 ;	
		=2986 ;	-----
		=2987 ;	
		=2988 ;	The line containing the search string is not on the screen.
		=2989 ;	Replot the entire screen with the line containing the search
		=2990 ;	string as the top line.
		=2991 ;	
		=2992	NOTOS: ;
1824	CD7F12	=2993	CALL NEWSN ; Plot a new screen.
		=2994 ;	
		=2995 ;	-----
		=2996 ;	
		=2997 ;	Final exit point for the search routine.
		=2998 ;	
		=2999	SEARET: ;
1827	C9	=3000	RET ; Return from the <SEARCH> routine.
		=3001 ;	
		=3002 ;	=====
		=3003 ;	*
		=3004 ;	* Function : STRCMP (String Compare) *
		=3005 ;	* Inputs : HL : Start address of the compare buffer. *
		=3006 ;	* DE : Current file address. *
		=3007 ;	* Outputs : Z = 1 : If a match is found. *
		=3008 ;	* Destroys : A,B,C and flags. *
		=3009 ;	* Calls : CONCAP. *
		=3010 ;	* *
		=3011 ;	* Description : This routine does a case insensitive *
		=3012 ;	* comparison between the string in the search buffer and the *
		=3013 ;	* current position in the file. *
		=3014 ;	* *
		=3015 ;	=====

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 52

LOC	OBJ	LINE	SOURCE STATEMENT
		=3016 ;	
		=3017 STRCMP:	; STRing CoMPare.
1828	E5	=3018 PUSH H	; Save all 16 bit registers.
1829	D5	=3019 PUSH D	
182A	C5	=3020 PUSH B	
182B	3A54EC	=3021 LDA SEASL	; Get the length of the string.
182E	57	=3022 MOV D,A	; setup counter = Length of string.
		=3023 ;	
		=3024 SCOMPL:	
182F	0A	=3025 LDAX B	; Get character.
1830	CDB811	=3026 CALL CONCAP	; Convert to Upper case.
1833	5F	=3027 MOV E,A	; Save upper case letter in E
		=3028 ;	
		=3029 STRC05:	
1834	7E	=3030 MOV A,M	; Get other character.
		=3031 ;	
		=3032 ;	Compare letters after converting to upper case.
		=3033 ;	
1835	CDB811	=3034 CALL CONCAP	; Convert to Upper case.
1838	BB	=3035 CMP E	; Compare letters.
1839	C24218	=3036 JNZ NOTEQ	; Exit loop if characters not equal.
		=3037 ;	; and zero flag clear.
183C	03	=3038 INX B	; Dec pointer.
183D	23	=3039 INX H	; Inc pointer.
183E	15	=3040 DCR D	; Dec counter.
183F	C22F18	=3041 JNZ SCOMPL	; Loop until counter = 0, i.e. the string has
		=3042 ;	; been found.
		=3043 ;	; Zero flag set by counter=0.
		=3044 NOTEQ:	
1842	C1	=3045 POP B	
1843	D1	=3046 POP D	
1844	E1	=3047 POP H	; Restore all 16 bit registers.
1845	C9	=3048 RET	; Return from <STRCMP>.
		=3049 ;	
		=3050 ;	End of <EDTFIND.A85> module.
		3051 SEJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 53

LOC	OBJ	LINE	SOURCE STATEMENT
		3052	\$INCLUDE(EDTSUB.A85)
=3053			-----;
=3054			;
=3055			Substitute module for the UCT SABus editor.;
=3056			-----;
=3057			;
=3058			This module contains (in order) the following subroutines:;
=3059			;
=3060			a) DOSUB - Do substitution.;
=3061			b) LFORLF - Look for linefeed routine.;
=3062			c) SETNSF - Set new screen flag.;
=3063			d) SUBROU - Substitute routine (Handles the "S" command.);
=3064			;
=3065			-----;
=3066			;
=3067			\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=3068 ;	=====
		=3069 ;*	
		=3070 ;*	Function : DOSUB (Do substitution)
		=3071 ;*	Inputs : HL = Pointer to search string that has be
		=3072 ;*	found in the text area.
		=3073 ;*	REPBUFF and REPSL = Buffer that must replace
		=3074 ;*	the search string in the text.
		=3075 ;*	Outputs : Z = 1 : Replace OK
		=3076 ;*	Z = 0 : Replace error (Memory full)
		=3077 ;*	HL = Last character of the replacement string
		=3078 ;*	in the text. (New character pointer)
		=3079 ;*	Destroys : A, B, D, E, H, L and flags.
		=3080 ;*	Calls : PAD.
		=3081 ;*	
		=3082 ;*	Description : This routine performs the substitution once
		=3083 ;*	the search string has been found in the text. The routine
		=3084 ;*	is entered with the HL register pair pointing to the
		=3085 ;*	search string that has been found in the text. This string
		=3086 ;*	must be replaced with the string in the replacement buffer
		=3087 ;*	<REPBUFF>. The length of the replacement string is given by
		=3088 ;*	the byte variable REPSL and the length of the search
		=3089 ;*	string is given by the byte variable SEASL. The process
		=3090 ;*	performed is as follows:
		=3091 ;*	
		=3092 ;*	a) If REPSL > SEASL then PAD the text with nulls to allow
		=3093 ;*	replacement string to be inserted into the text. If
		=3094 ;*	there is a PAD error then exit with the zero flag clear.
		=3095 ;*	
		=3096 ;*	b) Replace the search string with nulls.
		=3097 ;*	
		=3098 ;*	c) Copy the replacement string into the text area.
		=3099 ;*	
		=3100 ;	=====
		=3101 ;	
		=3102	DOSUB:
1846	E5	=3103	PUSH H ; Save start adr of the string found.
1847	3A54EC	=3104	LDA SEASL ; Load character count.
184A	5F	=3105	MOV E,A ;
184B	1600	=3106	MVI D,0 ;
184D	19	=3107	DAD D ; PAD address = Start_of_string + Length.
		=3108	;
184E	47	=3109	MOV B,A ; Move the length of the 1st string to B
184F	3A73EC	=3110	LDA REPSL ; Load replacement string length.
1852	90	=3111	SUB B ; Replacement_string longer the search_string?
1853	FA6B18	=3112	JM DSUB05 ; No - then need not PAD text.
		=3113 ;	
		=3114 ;	-----
		=3115 ;	
		=3116 ;	The text will have to be padded to allow a longer replacement
		=3117 ;	string to be inserted into the text area. The PAD address has
		=3118 ;	been calculated and so has the number of characters to PAD.
		=3119 ;	
1856	EB	=3120	XCHG ; Move PAD address to DE
1857	2A02E8	=3121	LHLD CHAR ; Load character pointer
185A	EB	=3122	XCHG ; Save character pointer in DE, HL=PAD address
185B	2202E8	=3123	SHLD CHAR ; Save PAD address as character pointer.
		=3124	;
		=3125	;
		=3126	;
		=3127	;
		=3128	;
185E	CD8515	=3129	CALL PAD ; PAD the file.
1861	CA6B18	=3130	JZ DSUB05 ; If Pad returned status <> Ok then
		=3131	out of memory error.
1864	EB	=3132	XCHG ;
1865	2202E8	=3133	SHLD CHAR ;
1868	C39418	=3134	JMP DSUBEX ; Exit DOSUB routine with Zero_Flag clear.
		=3135 ;	
		=3136 ;	-----
		=3137 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 55

LOC	OBJ	LINE	SOURCE STATEMENT
		=3138 ;	There are now the correct number of nulls to replace the search
		=3139 ;	string with the replacement string.
		=3140 ;	
		=3141 DSUB05:	
186B	E1	=3142	POP H ; Get start address of string found.
186C	E5	=3143	PUSH H ; Resave.
		=3144 ;	
		=3145 ;	-----
		=3146 ;	
		=3147 ;	Replace all characters in the search string with nulls.
		=3148 ;	
186D	3A54EC	=3149	LDA SEASL ; Load search string character count.
		=3150 DSUB10:	
1870	3600	=3151	MVI M,NULL ; Null all characters in string.
1872	23	=3152	INX H ;
1873	3D	=3153	DCR A ; Dec character counter.
1874	C27018	=3154	JNZ DSUB10 ; Loop until all characters blanked.
		=3155 ;	
		=3156 ;	-----
		=3157 ;	
		=3158 ;	Copy the replacement string into the nulled area.
		=3159 ;	
1877	E1	=3160	POP H ; HL = Start address of new string.
1878	1155EC	=3161	LXI D,REPBUF ; Load start address of the replacement
			string.
187B	3A73EC	=3162	LDA REPSL ; Load replacement string length.
187E	FE00	=3163	CPI 00H ; Zero length 2nd string ?
1880	CA8C18	=3164	JZ DSUB20 ; No string to move if length(2nd string)=0
		=3165	
1883	47	=3166	MOV B,A ;
		=3167 DSUB15:	
1884	1A	=3168	LDAX D ; Load character from replacement string
1885	77	=3169	MOV M,A ; Save in text.
1886	23	=3170	INX H ;
1887	13	=3171	INX D ;
1888	05	=3172	DCR B ; Dec count.
1889	C28418	=3173	JNZ DSUB15 ; Loop until all characters moved.
		=3174	
		=3175 DSUB20:	
188C	E5	=3176	PUSH H ; Save HL.
188D	2A74EC	=3177	LHLD NFINDS ;
1890	23	=3178	INX H ; Inc Number_of_finds
1891	2274EC	=3179	SHLD NFINDS ;
		=3180 DSUBEX:	
1894	E1	=3181	POP H ; Restore HL.
1895	C9	=3182	RET ; Return from the <DSUB> command.
		=3183 ;	
		=3184 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABUS editor

11:47:52 12-APR-88

EDT

PAGE 56

LOC	OBJ	LINE	SOURCE STATEMENT
		=3185 ;=====	
		=3186 ;*	
		=3187 ;*	Function : LFORLF (Look for line feeds)
		=3188 ;*	Inputs : HL = Start of buffer to check.
		=3189 ;*	A = Length of string in buffer.
		=3190 ;*	Outputs : NEWSF(memory) is set if a line feed is found
		=3191 ;*	Destroys : A, B, H, L and flags.
		=3192 ;*	Calls : None.
		=3193 ;*	
		=3194 ;*	Description : This routine checks for line feeds in either
		=3195 ;*	the search or the replace buffers. If a line feed is
		=3196 ;*	found then the screen must be updated.
		=3197 ;*	
		=3198 ;=====	
		=3199 ;	
		=3200 LFORLF:	; Start of look for Line feeds routine.
1896	B7	=3201	ORA A ; Zero length?
1897	CAAB18	=3202	JZ LFLF15 ; Yes - then exit
		=3203	
189A	47	=3204	MOV B,A ; Move string length to Bregister.
		=3205 LFLF05:	
189B	7E	=3206	MOV A,M ; Get character from text.
189C	FE0A	=3207	CPI LF ; If char = Line_feed
189E	C2A618	=3208	JNZ LFLF10 ;
		=3209	; Begin
18A1	3EFF	=3210	MVI A,TRUE ;
18A3	3209E8	=3211	STA NEWSF ; New_screen_flag = True
		=3212	; End;
		=3213 LFLF10:	
18A6	23	=3214	INX H ; Inc buffer pointer.
18A7	05	=3215	DCR B ; Dec buffer len.
18A8	C29B18	=3216	JNZ LFLF05 ; Loop until len = zero.
		=3217	
		=3218 LFLF15:	
18AB	C9	=3219	RET ; Return from <LFORLF>
		=3220 ;	
		=3221 ;=====	
		=3222 ;*	
		=3223 ;*	Function : SETNSF (Set new screen flag)
		=3224 ;*	Inputs : None.
		=3225 ;*	Outputs : NEWSF = True if either the search string or
		=3226 ;*	the replacement string contain line
		=3227 ;*	feeds.
		=3228 ;*	Destroys : A and Flags.
		=3229 ;*	Calls : LFORLF.
		=3230 ;*	
		=3231 ;*	Description : This routine uses the LFORLF routine to scan
		=3232 ;*	the search and replace buffers looking for line feeds. If
		=3233 ;*	either of these buffers contains a line feed then the new
		=3234 ;*	screen flag <NEWSF> is set.
		=3235 ;*	
		=3236 ;=====	
		=3237 ;	
		=3238 SETNSF:	; Set new screen flag routine.
		=3239	
18AC	E5	=3240	PUSH H ; Save regs.
18AD	C5	=3241	PUSH B ;
		=3242	
18AE	3E00	=3243	MVI A,FALSE ;
18B0	3209E8	=3244	STA NEWSF ; New_screen_flag = False.
		=3245	
18B3	2136EC	=3246	LXI H,SEABUF ;
18B6	3A54EC	=3247	LDA SEASL ; HL = Start of buffer, Acc = Length.
18B9	CD9618	=3248	CALL LFORLF ; Look for line feeds.
		=3249	
18BC	2155EC	=3250	LXI H,REPBUF ;
18BF	3A73EC	=3251	LDA REPSL ; HL = Start of buffer, Acc = Length.
18C2	CD9618	=3252	CALL LFORLF ; Look for line feeds.
		=3253	
18C5	C1	=3254	POP B ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 57

LOC	OBJ	LINE	SOURCE STATEMENT
18C6	E1	=3255	POP H ; Restore registers.
		=3256	;
18C7	C9	=3257	RET ; Return from <SETNSF>
		=3258 ;	
		=3259	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 58

LOC	OBJ	LINE	SOURCE STATEMENT
		=3260 ;	=====
		=3261 ;*	*
		=3262 ;*	Function : SUBROU (Substitute routine)
		=3263 ;*	Inputs : Command buffer (in memory) with the first
		=3264 ;*	letter being a "S".
		=3265 ;*	Outputs : None.
		=3266 ;*	Destroys : A, B, C, D, E, H, I and flags.
		=3267 ;*	Calls : PACK, CONCAP, MOVSTR, LOOKFS, DOSUB, BRROU,
		=3268 ;*	STRING, PRINTD, HOME, GOBACK, SETNSF, ONSCN, GOTO,
		=3269 ;*	BLINE, CLRL, PLINE, NEWSN, ERRSTR, QINIT and KEYIN.
		=3270 ;*	*
		=3271 ;*	Description : This routine handles the editor "S" command.
		=3272 ;*	*
		=3273 ;	=====
		=3274 ;	
		=3275 SUBROU:	; Substitute routine.
18C8	CDE615	=3276	CALL PACK ; Pack file before looking for string.
		=3277	;
18CB	3E00	=3278	MVI A, FALSE ;
18CD	3214E8	=3279	STA SUBREP ; Reset Repeat flag.
		=3280	;
18D0	210000	=3281	LXI H, 0000H ;
18D3	2274EC	=3282	SHLD NFINDS ; Reset number of finds.
		=3283	;
18D6	2117E8	=3284	LXI H, BUF ; Load pointer to start of command line.
		=3285 ;	
		=3286 ;	-----
		=3287 ;	
		=3288 ;	Look for the 1st non space character in the command to use as a
		=3289 ;	delimiter. Also check for the substitute next command "SN<EOT>".
		=3290 ;	If the first character encountered is an "N" then it could be a
		=3291 ;	sub next command, but if the character following the "N" is not
		=3292 ;	an <EOT> then treat the "N" as a delimiter.
		=3293 ;	
		=3294 SUB05:	;
18D9	23	=3295	INX H ; Inc pointer to next character.
18DA	7E	=3296	MOV A, M ; Load character from the command line.
18DB	CDB811	=3297	CALL CONCAP ; Convert command letter to upper case.
18DE	FE20	=3298	CPI SPACE ; SPACE ?
18E0	CAD918	=3299	JZ SUB05 ; Loop until (char # SPACE)
		=3300	;
18E3	FE04	=3301	CPI EOT ; Only 'S' typed ?
18E5	CAC919	=3302	JZ SUBER1 ;
		=3303	;
18E8	FE4E	=3304	CPI 'N' ; Command = 'SN' ?
18EA	C20319	=3305	JNZ NOTNXT ; No - Then Fetch the rest of the Command.
		=3306 ;	
		=3307 ;	-----
		=3308 ;	
		=3309 ;	Command thus far = "SN?..."
		=3310 ;	
		=3311 ;	Check for "SN<EOT>" command.
		=3312 ;	
18ED	23	=3313	INX H ; Inc pointer to after the 'SN'
18EE	7E	=3314	MOV A, M ; Load next character.
18EF	FE04	=3315	CPI EOT ; Command = 'SN<EOT>' ?
18F1	CAF818	=3316	JZ SNCOM ; If it is the command = sub next.
		=3317	;
18F4	2B	=3318	DCX H ; Dec pointer to point at the 'N'
18F5	C30319	=3319	JMP NOTNXT ; Continue ('N' Could be the delimiter)
		=3320 ;	
		=3321 ;	-----
		=3322 ;	
		=3323 ;	Command = "SN<EOT>", check that the string buffers contain a
		=3324 ;	substitute command. If they do not (SUBBIT=False) then error..
		=3325 ;	
		=3326 SNCOM:	; Sub Next Command.
18F8	3A12E8	=3327	LDA SUBBIT ; Load SUB_Buffers_In_Tact flag.
18FB	FE00	=3328	CPI FALSE ; If buffers not in tact then Error#4
18FD	CAD519	=3329	JZ SUBER4 ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 59

LOC	OBJ	LINE	SOURCE STATEMENT
		=3330	
1900	C34419	=3331	JMP SUBCON ; Continue with sub.
		=3332 ;	
		=3333 ;	
		=3334 ;	
		=3335 NOTNXT:	
1903	3E00	=3336	MVI A,FALSE ;
1905	3212E8	=3337	STA SUBBIT ; Substitute buffers about to be destroyed.
		=3338	;
1908	1136EC	=3339	LXI D,SEABUF ; Load start adr of the 1st string buffer.
190B	0600	=3340	MVI B,NULL ; Set delimiter to null, i.e. No delimiter set.
190D	CD8A17	=3341	CALL MOVSTR ; Copy 1st string into the string buffer
1910	C2CF19	=3342	JNZ SUBER2 ; If error then exit with message
		=3343	;
1913	79	=3344	MOV A,C ; Move the length of the 1st string to Areg.
1914	FE00	=3345	CPI 00H ; length(1st string) = 0?
1916	CAE119	=3346	JZ SUBER6 ; Yes - Then Substitute error #6.
		=3347	;
1919	3254EC	=3348	STA SEASL ; Save character count
191C	1155EC	=3349	LXI D,REPBUF ; Load start adr of the 2nd string buffer
191F	CD8A17	=3350	CALL MOVSTR ; Copy 2nd string into the 2nd string buffer
		=3351	; using the delimiter from the 1st string.
		=3352 ;	
		=3353 ;	
		=3354 ;	
		=3355 ;	NOTE: A string move error returned when moving the second string is ignored.
		=3356 ;	This allows the user the option of leaving off the 3rd delimiter.
		=3357 ;	
		=3358 ;	i.e. Substitute command:
		=3359 ;	
		=3360 ;	S<Delimiter><Search_String><Delimiter><Replacement_String>[<Delimiter>[A]]
		=3361 ;	
1922	79	=3362	MOV A,C ;
1923	3273EC	=3363	STA REPSL ; Save length of the second string.
		=3364	;
1926	3EFF	=3365	MVI A,TRUE ;
1928	3212E8	=3366	STA SUBBIT ; Substitute buffers OK.
		=3367	;
192B	7E	=3368	MOV A,M ; Get character just after the 2nd delimiter.
192C	FE04	=3369	CPI EOT ; Any options ?
192E	CA4419	=3370	JZ SUBCON ; If no options the continue.
		=3371	;
1931	CDB811	=3372	CALL CONCAP ; Convert to upper case.
1934	FE41	=3373	CPI 'A' ; Option = 'All' ?
1936	C2E719	=3374	JNZ SUBER7 ; No - then exit (Invalid option)
		=3375 ;	
		=3376 ;	
		=3377 ;	
		=3378 ;	Option = All, Set substitute repeat flag and Set Character_Pointer
		=3379 ;	equal to the Start_of_text_pointer (Search the entire
		=3380 ;	text area).
		=3381 ;	
1939	3EFF	=3382	MVI A,TRUE ;
193B	3214E8	=3383	STA SUBREP ; Set repeat sub flag.
		=3384	;
193E	2AFCEF	=3385	LHLD SPOINT ;
1941	2202E8	=3386	SHLD CHAR ; Start Search from the start of the file.
		=3387 ;	
		=3388	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=3389 ;	=====
		=3390 ;	
		=3391 ;	MAIN SUBSTITUTION LOOP.
		=3392 ;	
		=3393 ;	=====
		=3394 ;	
		=3395 SUBCON:	
1944	CD6617	=3396	CALL LOOKFS ; Look for the 1st string.
1947	C25E19	=3397	JNZ SUBER5 ; If string not found then print error message.
		=3398	
194A	CD4618	=3399	CALL DOSUB ; Perform substitution
194D	C2ED19	=3400	JNZ SUBER8 ; Error then exit (Memory full error)
		=3401	
1950	3A14E8	=3402	LDA SUBREP ; Load sub repeat flag.
1953	FEFF	=3403	CPI TRUE ;
1955	C28919	=3404	JNZ SUB37 ; If Subrep=false then exit loop.
		=3405	
1958	2202E8	=3406	SHLD CHAR ; Save pointer to the end of the replaced
		=3407	string as CHAR.
195B	C34419	=3408	JMP SUBCON ; And loop until string not found.
		=3409 ;	
		=3410 ;	-----
		=3411 ;	
		=3412 ;	Error number 5: String not found.
		=3413 ;	
		=3414 ;	If the substitution command had an "ALL" switch then this is the
		=3415 ;	only way of terminating the command. If the "SUBREP" flag is clear
		=3416 ;	then it was a error. If the "SUBREP" flag was set and the number
		=3417 ;	of substitutions is zero then it is also an error. If none of the
		=3418 ;	above are true then there is no error, print the number of
		=3419 ;	substitutions and continue.
		=3420 ;	
		=3421 SUBER5:	
			; String not found error.
195E	3A14E8	=3422	LDA SUBREP ;
1961	FE00	=3423	CPI FALSE ;
1963	CADB19	=3424	JZ SUBE51 ; If SUBREP=FALSE then error
		=3425	
1966	2A74EC	=3426	LHLD NFINDS ;
1969	7C	=3427	MOV A,H ;
196A	B5	=3428	ORA L ;
196B	CADB19	=3429	JZ SUBE51 ; If NFINDS = 0 then error.
		=3430 ;	
		=3431 ;	-----
		=3432 ;	
		=3433 ;	The "SUBREP" flag is set and the number of substitutions is not
		=3434 ;	zero. Move the cursor to the begining of the text, print the number
		=3435 ;	of substitutions performed.
		=3436 ;	
196E	CDD922	=3437	CALL BRROU ; After multiple subs goto start of text.
1971	21372B	=3438	LXI H,ERRCCS ;
1974	CDE125	=3439	CALL STRING ; Move cursor to the last line on the
		=3440	screen and clear it.
1977	2A74EC	=3441	LHLD NFINDS ;
197A	CDFA25	=3442	CALL PRINTD ; Print the number of Subs made.
197D	21692D	=3443	LXI H,SMMSG ; Print Subs_make_message.
1980	CDE125	=3444	CALL STRING ;
1983	CDCA26	=3445	CALL HOME ; Home cursor as expected by BRROU.
1986	C30D1A	=3446	JMP SUBEX ; Exit from <SUB_Routine>
		=3447 ;	
		=3448 ;	-----
		=3449 ;	
		=3450 ;	Search for 1st non null character. This with be the character
		=3451 ;	pointed to by the character pointer "CHAR".
		=3452 ;	
		=3453 SUB37:	
1989	2B	=3454	DCX H ; Dec to last char in the string.
		=3455 SUB40:	
198A	23	=3456	INX H ; Inc to pointer just after the rep string.
198B	7E	=3457	MOV A,M ; Load char after string.
198C	FE00	=3458	CPI NULL ; NULL ?

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 62

LOC	OBJ	LINE	SOURCE STATEMENT
		=3492 ;	=====
		=3493 ;	
		=3494 ;	SUBSTITUTE ERROR EXIT POINTS.
		=3495 ;	
		=3496 ;	=====
		=3497 ;	
		=3498 ;	Substitute error #1, General Syntax error.
		=3499 ;	
		=3500 SUBER1:	
19C9	214B2C	=3501	LXI H,SUBEM1 ;
19CC	C30A1A	=3502	JMP PSUBER ;
		=3503 ;	
		=3504 ;	-----
		=3505 ;	
		=3506 ;	Substitute error #2, General syntax error.
		=3507 ;	
		=3508 SUBER2:	
19CF	214B2C	=3509	LXI H,SUBEM2 ;
19D2	C30A1A	=3510	JMP PSUBER ;
		=3511 ;	
		=3512 ;	-----
		=3513 ;	
		=3514 ;	Substitute error #4, No string in sub buffers.
		=3515 ;	
		=3516 SUBER4:	
19D5	21792C	=3517	LXI H,SUBEM4 ;
19D8	C30A1A	=3518	JMP PSUBER ;
		=3519 ;	
		=3520 ;	-----
		=3521 ;	
		=3522 ;	Substitute error #5, String not found.
		=3523 ;	
		=3524 SUBE51:	
19DB	21982C	=3525	LXI H,SUBEM5 ;
19DE	C30A1A	=3526	JMP PSUBER ;
		=3527 ;	
		=3528 ;	-----
		=3529 ;	
		=3530 ;	Substitute error #6, Zero length 1st string.
		=3531 ;	
		=3532 SUBER6:	
19E1	21AB2C	=3533	LXI H,SUBEM6 ;
19E4	C30A1A	=3534	JMP PSUBER ;
		=3535 ;	
		=3536 ;	-----
		=3537 ;	
		=3538 ;	Substitute error #7, Unknown sub option.
		=3539 ;	
		=3540 SUBER7:	
19E7	21D42C	=3541	LXI H,SUBEM7 ;
19EA	C30A1A	=3542	JMP PSUBER ;
		=3543 ;	
		=3544 ;	-----
		=3545 ;	
		=3546 ;	Substitute error #8, Memory full.
		=3547 ;	
		=3548 SUBER8:	
19ED	3A14E8	=3549	LDA SUBREP ;
19F0	FE00	=3550	CPI FALSE ;
19F2	CA071A	=3551	JZ SUBE85 ; If Sub_repeat = true then
		=3552	begin
19F5	21F02C	=3553	LXI H,SUBM8a ;
19F8	CD8723	=3554	CALL ERRSTR ; Writeln("Memory full error");
19FB	CDEA12	=3555	CALL QINIT ; Input_queue_init;
19FE	CDBE23	=3556	CALL KEYIN ; Acc := Get_key ;
1A01	CDD922	=3557	CALL BRROU ; Beginning_of_range;
1A04	C30D1A	=3558	JMP SUBEX ; end
		=3559	else
		=3560 SUBE85:	begin
1A07	213B2D	=3561	LXI H,SUBM8b ; Writeln("Memory full error");

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 63

LOC	OBJ	LINE	SOURCE STATEMENT
		=3562	; End;
		=3563 ;	-----
		=3564 ;	
		=3565 ;	Substitute final error exit point.
		=3566 ;	
		=3567 PSUBER:	;
1A0A	CD8723	=3568	CALL ERRSTR ;
		=3569 ;	
		=3570 ;	-----
		=3571 ;	
		=3572 ;	Substitute exit point.
		=3573 ;	
		=3574 SUBEX:	;
1A0D	C9	=3575	RET ; End of <SUBROU> routine.
		=3576 ;	
		=3577 ;	End of the <EDTSUB.A85> module.
		3578	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 64

LOC	OBJ	LINE	SOURCE STATEMENT
		3579 ;	
		3580 ;	Include delete routine.
		3581 ;	
		3582	\$INCLUDE(EDTDELROU.A85)
		=3583 ;	-----
		=3584 ;	
		=3585 ;	Delete module for the UCT SABus editor.
		=3586 ;	-----
		=3587 ;	
		=3588 ;	This module contains (in order) the following subroutines:
		=3589 ;	
		=3590 ;	a) DCHARS - Delete n characters from the text.
		=3591 ;	b) DELCRO - Delete character routine (called when the delete
		=3592 ;	key is typed in command mode)
		=3593 ;	c) DELETE - Deletes a single character.
		=3594 ;	d) DELROU - Delete routine (called if first character of the
		=3595 ;	command = 'D'.
		=3596 ;	e) DLINEs - Delete n lines from the text.
		=3597 ;	f) DTEOLR - Delete to end of line routine.
		=3598 ;	g) INCNUL - Increment the null counter and pack if needed.
		=3599 ;	h) NULLUP - Update null counter.
		=3600 ;	
		=3601 ;	-----
		=3602 ;	
		=3603	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=3604 ;	=====
		=3605 ;*	*
		=3606 ;*	Function : DCHARS (Delete characters from file, starting
		=3607 ;*	at the current cursor position).
		=3608 ;*	Inputs : DE : Number of characters to delete.
		=3609 ;*	Outputs : None.
		=3610 ;*	Destroys : All.
		=3611 ;*	Calls : DCMP, NULLUP, GOTO and PLINE.
		=3612 ;*	*
		=3613 ;*	Description : This routine is used by the delete routine to
		=3614 ;*	delete a certain number of characters from the text area
		=3615 ;*	starting at the current cursor position.
		=3616 ;*	*
		=3617 ;*	If any of the characters deleted are end of line markers
		=3618 ;*	(LF) then the screen is rewritten on exit from this
		=3619 ;*	routine, this is done using the NEW_Screen_Flag.
		=3620 ;*	*
		=3621 ;*	The number of characters to be deleted is passed in the DE
		=3622 ;*	register pair. The file null counter is updated by adding
		=3623 ;*	the number of characters deleted (replaced by nulls) to
		=3624 ;*	the null counter and packing the file if it gets to large.
		=3625 ;*	*
		=3626 ;	=====
		=3627 ;	
		=3628	DCHARS: ; Start of delete characters routine <DCHARS>.
1A0E	3E00	=3629	MVI A,FALSE ;
1A10	3209E8	=3630	STA NEWSF ; NEW_Screen_Flag = false
		=3631	;
1A13	42	=3632	MOV B,D ;
1A14	4B	=3633	MOV C,E ; Move the number of characters to be
		=3634	deleted from 'DE' to 'BC'
1A15	C5	=3635	PUSH B ; Save number of chars to delete.
		=3636	;
1A16	2A02E8	=3637	LHLD CHAR ; Load current character pointer.
1A19	EB	=3638	XCHG ;
1A1A	2AFEEF	=3639	LHLD EPOINT ; Load end of text pointers.
1A1D	EB	=3640	XCHG ;
		=3641	DCHRO5: ;
1A1E	CD43F0	=3642	CALL DCMP ;
1A21	CA3F1A	=3643	JZ DCHR15 ; If Char = Epoint then exit.
		=3644	;
1A24	7E	=3645	MOV A,M ; Load current character.
1A25	23	=3646	INX H ; Inc pointer to the next character.
1A26	FE00	=3647	CPI NULL ;
1A28	CA1E1A	=3648	JZ DCHRO5 ; Loop until char # null.
		=3649	;
1A2B	2B	=3650	DCX H ; Dec pointer to NON NULL character.
1A2C	FE0A	=3651	CPI LF ; End of line ?
1A2E	C2361A	=3652	JNZ DCHR10 ;
		=3653	;
1A31	3EFF	=3654	MVI A,TRUE ;
1A33	3209E8	=3655	STA NEWSF ;
		=3656	DCHR10: ;
1A36	3600	=3657	MVI M,NULL ; Remove character by replacing it with a null.
1A38	23	=3658	INX H ; Inc Pointer past the Null.
1A39	0B	=3659	DCX B ; Dec character counter.
1A3A	78	=3660	MOV A,B ;
1A3B	B1	=3661	ORA C ; Set zero flag if BC = zero.
1A3C	C21E1A	=3662	JNZ DCHRO5 ; Loop until Char_Counter = zero OR
		=3663	End_OF_Line = true.
		=3664	DCHR15: ;
1A3F	2202E8	=3665	SHLD CHAR ; Save pointer to the next character as the
		=3666	character pointer CHAR.
1A42	C1	=3667	POP B ; Restore number of characters deleted.
1A43	CD141C	=3668	CALL NULLUP ; Update null count.
		=3669	;
1A46	3E00	=3670	MVI A,FALSE ;
1A48	3211E8	=3671	STA RESFLG ; Reset Cursor_Reset flag.
		=3672	;
1A4B	3A09E8	=3673	LDA NEWSF ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABUS editor

11:47:52 12-APR-88

EDT

PAGE 66

LOC	OBJ	LINE	SOURCE STATEMENT
1A4E	FE00	=3674	CPI FALSE ;
1A50	CA6F1A	=3675	JZ DCHR20 ; If new_screen_flag = true then
		=3676	; rewrite everthing below the current line.
1A53	3E17	=3677	MVI A,LSTLIN+1 ;
1A55	3208E8	=3678	STA EOBLIN ; Move End_Of_Buffer screen pointer off
		=3679	;
1A58	3A07E8	=3680	LDA OLINE ; Get old line position.
1A5B	320EE8	=3681	STA LCLINE ; lowest correct line = current line.
1A5E	3205E8	=3682	STA LINE ; Save as new line.
1A61	3E00	=3683	MVI A,00H ;
1A63	3204E8	=3684	STA COL ; Set col=0
1A66	CDB426	=3685	CALL GOTO ; Move cursor to col zero of the line on
		=3686	; which the charaters were deleted.
1A69	CD7224	=3687	CALL PLINE ;
1A6C	C3801A	=3688	JMP DCHR25 ;
		=3689	DCHR20: ;
1A6F	3A07E8	=3690	LDA OLINE ; Get old line position.
1A72	3205E8	=3691	STA LINE ; Save as new line.
1A75	3E00	=3692	MVI A,00H ;
1A77	3204E8	=3693	STA COL ; Set col=0
1A7A	CDB426	=3694	CALL GOTO ; Move cursor to col zero of the line on
		=3695	; which the charaters were deleted.
1A7D	CD7224	=3696	CALL PLINE ; Reprint the line.
		=3697	DCHR25: ;
1A80	C9	=3698	RET ; Return from <DCHARS>
		=3699	;
		=3700	=====
		=3701	;
		=3702	* Function : DELCRO (Delete character routine) *
		=3703	* Inputs : CHAR : The current character pointer. *
		=3704	* Outputs : None. *
		=3705	* Destroys : Acc and flags *
		=3706	* Calls : DELETE *
		=3707	* *
		=3708	* Description: This routine is called when the DEL key is *
		=3709	* typed. It simply deletes 1 character and increments the *
		=3710	* NULL counter. *
		=3711	* *
		=3712	=====
		=3713	;
		=3714	DELCRO: ; Start of Delete character routine.
1A81	CD881A	=3715	CALL DELETE ; Delete the character
1A84	CD041C	=3716	CALL INCNUL ; Inc null count
1A87	C9	=3717	RET ; Return from <DELCRO>
		=3718	;
		=3719	=====
		=3720	;
		=3721	* Function : DELETE (Delete character before the cursor) *
		=3722	* Inputs : CHAR : Pointer to current character. *
		=3723	* Outputs : None. *
		=3724	* Destroys : D,E,H,L,Acc and flags. *
		=3725	* Calls : FDCMP, RBULF, QCHAR, REVLF, BLIN, PLINE, *
		=3726	* UPDATE and PCUR. *
		=3727	* *
		=3728	* Description : This routine is used to delete the character *
		=3729	* behind the edit cursor (CHAR-1). It is used when the *
		=3730	* delete key is typed from command mode or insert mode. *
		=3731	* *
		=3732	* If the character to be deleted is not a line feed (end of *
		=3733	* line marker) then is is simply replaced by a null and the *
		=3734	* null count increased by one. If the character to be *
		=3735	* deleted is a line feed then the rest of the screen is *
		=3736	* rewritten and the current line appended to the previous *
		=3737	* line. *
		=3738	* *
		=3739	=====
		=3740	;
		=3741	DELETE: ; Start of character delete routine
1A88	2AFCEF	=3742	LHLD SPOINT ; Fetch start of text pointer
1A8B	EB	=3743	XCHG ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 67

LOC	OBJ	LINE	SOURCE STATEMENT
1A8C	2A02E8	=3744	LHLD CHAR ; Fetch current character pointer
		=3745	DELE02: ;
1A8F	CD3012	=3746	CALL FDCMP ; Compare DE and HL
1A92	CAE41A	=3747	JZ DELRET ; If Character-pointer = Start_of_text then
		=3748	; there are no characters to delete.
1A95	2B	=3749	DCX H ; Move pointer 1 character back
1A96	7E	=3750	MOV A,M ; Load current character to delete.
1A97	FE00	=3751	CPI NULL ; Loop until the char is not a null
1A99	CA8F1A	=3752	JZ DELE02 ;
		=3753	;
1A9C	FE0A	=3754	CPI LF ; Start of line ?
1A9E	C2D61A	=3755	JNZ DELE05 ; No - Then handle in simple fashion
		=3756 ;	
		=3757 ;	-----
		=3758 ;	
		=3759 ;	If the character being deleted = LF then
		=3760 ;	append the current line with the previous one.
		=3761 ;	
1AA1	3600	=3762	MVI M,NULL ; Replace the LF with a NUL
1AA3	CD6913	=3763	CALL RBULF ; Go backwards
1AA6	2200E8	=3764	SHLD POINT ; Save new start_of_line_pointer
		=3765 ;	
		=3766 ;	-----
		=3767 ;	
1AA9	3A05E8	=3768	LDA LINE ;
1AAC	B7	=3769	ORA A ;
1AAD	CAB11A	=3770	JZ DELE04 ; If LINE <> 0 then
		=3771	;
1AB0	3D	=3772	DCR A ; Decrement line number
		=3773	DELE04: ;
1AB1	3205E8	=3774	STA LINE ; Update LINE number
1AB4	320EE8	=3775	STA LCLINE ; Update lowest correct line number
1AB7	3E17	=3776	MVI A,LSTLIN+1 ;
1AB9	3208E8	=3777	STA EOBLIN ; Set End-of-buffer off screen.
		=3778	;
1ABC	CD9914	=3779	CALL QCHAR ; Set RELCH
1ABF	CDED26	=3780	CALL REVLF ; Move up one line
1AC2	CD9D26	=3781	CALL BLINE ; Beginning of the line
1AC5	CD7224	=3782	CALL PLINE ; Print the line
		=3783	;
1AC8	3A15E8	=3784	LDA INSFLG ;
1ACB	FEFF	=3785	CPI TRUE ; If Insert_mode = true then
1ACD	CCD513	=3786	CZ UPDATE ; update the rest of the screen.
1ADO	CD0615	=3787	CALL PCUR ; Position cursor
		=3788	;
1AD3	C3E41A	=3789	JMP DELRET ; Exit DELETE routine
		=3790 ;	
		=3791 ;	-----
		=3792 ;	
		=3793 ;	If the character being deleted # LF then simply replace with NUL
		=3794 ;	
		=3795	DELE05: ;
1AD6	3600	=3796	MVI M,NULL ; Blank char
		=3797	;
1AD8	CD9914	=3798	CALL QCHAR ; Set RELCH
1ADB	CD9D26	=3799	CALL BLINE ; Goto the beginning of the current line
1ADE	CD7224	=3800	CALL PLINE ; Reprint the line
1AE1	CD0615	=3801	CALL PCUR ; Position the cursor
		=3802	DELRET: ;
1AE4	C9	=3803	RET ; Return from <DELETE>
		=3804 ;	
		=3805 ;	=====
		=3806 ;*	*
		=3807 ;*	Function : DELROU (Delete routine, called whenever the
		=3808 ;*	the first character in the command line is
		=3809 ;*	'D').
		=3810 ;*	Inputs : Command line, in buffer starting at <BUF>
		=3811 ;*	Outputs : None.
		=3812 ;*	Destroys : All.
		=3813 ;*	Calls : CONCAP, QNUM, READNU, DCHARS and DLINEs.

LOC	OBJ	LINE	SOURCE STATEMENT
		=3814 ;*	
		=3815 ;*	Description : This routine decides on the type of delete
		=3816 ;*	command (character or line delete) and the number of lines
		=3817 ;*	or characters to delete.
		=3818 ;*	
		=3819 ;*	The valid formats for the delete command are as follows:
		=3820 ;*	
		=3821 ;*	'DnnnnL<EOT>' - e.g. 'D10L<EOT>' - Deletes 10 lines.
		=3822 ;*	'DnnnnC<EOT>' - e.g. 'D10C<EOT>' - Deletes 20 charactes.
		=3823 ;*	'DC' - Deletes the current character.
		=3824 ;*	'DL' - Deletes the current line.
		=3825 ;*	
		=3826 ;	=====
		=3827 ;	
		=3828	DELROU:
1AE5	2118E8	=3829	LXI H, BUF+1 ; Load pointer to point to the first character
		=3830	; after the 'D' in the command buffer.
1AE8	7E	=3831	MOV A, M ; Fetch next character on the command line
1AE9	CDB811	=3832	CALL CONCAP ; Convert to upper case
1AEC	FE4C	=3833	CPI 'L' ; 'L' ?
1AEE	C2FE1A	=3834	JNZ NOT1L ; No - Then command <> 'DL'
		=3835	;
1AF1	23	=3836	INX H ;
1AF2	7E	=3837	MOV A, M ; Fetch next character
1AF3	FE04	=3838	CPI EOT ; <EOT> ?
1AF5	C2371B	=3839	JNZ DELER1 ; No - Then command <> 'DL<EOT>'
		=3840	;
		=3841	; Command = 'DL<EOT>'
1AF8	110100	=3842	LXI D, 1 ; Load lines_to_delete = 1
1AFB	C3441B	=3843	JMP DLINE ; Delete n lines
		=3844 ;	
		=3845 ;	-----
		=3846 ;	
		=3847 ;	Command <> 'DL', try 'DC'
		=3848 ;	
		=3849	NOT1L:
1AFE	FE43	=3850	CPI 'C' ; 'C' ? (i.e. command = 'DC' ?).
1B00	C2101B	=3851	JNZ NOT1C ; No - Then not 'DC'
		=3852	;
1B03	23	=3853	INX H ; Command = 'DC?'
1B04	7E	=3854	MOV A, M ; Fetch next character.
1B05	FE04	=3855	CPI EOT ; <EOT> ?
1B07	C2371B	=3856	JNZ DELER1 ; No - Then not 'DC<EOT>'
		=3857	;
		=3858	; Command = 'DC<EOT>'
1B0A	110100	=3859	LXI D, 1 ; Set characters to delete = 1
1B0D	C30E1A	=3860	JMP DCHARS ; Delete characters.
		=3861 ;	
		=3862 ;	-----
		=3863 ;	
		=3864 ;	Command <> 'DC' or 'DL', try 'DnnnnL' and 'DnnnnC'
		=3865 ;	
		=3866	NOT1C:
1B10	CD0613	=3867	CALL QNUM ; Is the 1st character after the 'D' a number ?
1B13	C2371B	=3868	JNZ DELER1 ; No - Then delete command syntax error
		=3869	;
1B16	CD4A26	=3870	CALL READNU ; Read number from the command line into
		=3871	the DE register pair.
1B19	C23D1B	=3872	JNZ DELER2 ; If READNU error then exit with message.
		=3873	;
1B1C	46	=3874	MOV B, M ; Load delete Option 'L' or 'C'
1B1D	23	=3875	INX H ; Inc past Option.
1B1E	7E	=3876	MOV A, M ; Load next character, Should be <EOT>.
1B1F	FE04	=3877	CPI EOT ;
1B21	C2371B	=3878	JNZ DELER1 ; Invalid Syntax if not <EOT>.
		=3879	;
1B24	7A	=3880	MOV A, D ;
1B25	B3	=3881	ORA E ; Set zero flag if DE = zero.
1B26	CA431B	=3882	JZ DELEX1 ; Return if Number read = zero.
		=3883	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 69

LOC	OBJ	LINE	SOURCE STATEMENT
1B29	78	=3884	MOV A,B ; Restore option.
1B2A	CDB811	=3885	CALL CONCAP ; Convert to Upper case.
1B2D	FE4C	=3886	CPI 'L' ;
1B2F	CA441B	=3887	JZ DLINES ; If not 'L' then not a delete line command.
		=3888	;
1B32	FE43	=3889	CPI 'C' ; Delete characters ?
1B34	CA0E1A	=3890	JZ DCHARS ;
		=3891	;
		=3892	DELER1: ;
1B37	21F72B	=3893	LXI H,DERRM1 ;
1B3A	C3401B	=3894	JMP PDELER ;
		=3895	DELER2: ;
1B3D	21382C	=3896	LXI H,DERRM2 ;
		=3897	PDELER: ;
1B40	CD8723	=3898	CALL ERRSTR ; Error exit from delete routine
		=3899	DELEX1: ;
1B43	C9	=3900	RET ; Return from <DELROU>
		=3901 ;	
		=3902 ;	=====
		=3903 ;*	*
		=3904 ;*	Function : DLINES (delete line from the text) *
		=3905 ;*	Inputs : DE : Number of lines to delete. *
		=3906 ;*	POINT : Pointer to the start of the current *
		=3907 ;*	line. *
		=3908 ;*	Outputs : None. *
		=3909 ;*	Destroys : All. *
		=3910 ;*	Calls : PLINE, PACK, GOTO, FDCMP, RBULF, CLRL, BLINE *
		=3911 ;*	STRING and REVLf. *
		=3912 ;*	*
		=3913 ;*	Description : The routine is used to delete n line from the *
		=3914 ;*	file starting at the current line and deleting forwards. *
		=3915 ;*	*
		=3916 ;*	The routine replaces a line at a time with nulls until *
		=3917 ;*	either the end of file is reached (POINT=EPOINT) or the *
		=3918 ;*	correct number of lines have been deleted. *
		=3919 ;*	*
		=3920 ;	=====
		=3921 ;	
		=3922	DLINES: ; Delete LINES routine.
1B44	42	=3923	MOV B,D ; Move number of lines to delete from DE
1B45	4B	=3924	MOV C,E ; to BC.
		=3925	;
1B46	3E00	=3926	MVI A,FALSE ;
1B48	3211E8	=3927	STA RESFLG ; Reset RESTore cursor flag.
1B4B	3E17	=3928	MVI A,LSTLIN+1 ;
1B4D	3208E8	=3929	STA EOBLIN ; Move End_Of_Buffer screen pointer off
		=3930	the screen.
1B50	3A07E8	=3931	LDA OLINE ;
1B53	3205E8	=3932	STA LINE ; Restore old line position.
1B56	320EE8	=3933	STA LCLINE ; Lowest correct line = Old line position
		=3934	(will clear to end of screen).
1B59	3E00	=3935	MVI A,00 ;
1B5B	3204E8	=3936	STA COL ; Set col position = 0.
1B5E	320AE8	=3937	STA RELCH ; Set relative character position = 0.
1B61	CDB426	=3938	CALL GOTO ; Move to the beginning of the current line.
		=3939	;
1B64	2AFEEF	=3940	LHLD EPOINT ;
1B67	EB	=3941	XCHG ; Move end_point to 'DE'
1B68	2A00E8	=3942	LHLD POINT ; Load pointer to the start of the current line
		=3943	DEL1L: ;
1B6B	7E	=3944	MOV A,M ; Load character.
1B6C	3600	=3945	MVI M,NULL ; Blank char with null
1B6E	FE0A	=3946	CPI LF ;
1B70	CA771B	=3947	JZ DELEOL ;
		=3948	;
1B73	23	=3949	INX H ;
1B74	C36B1B	=3950	JMP DEL1L ; Delete next character
		=3951	;
		=3952	DELEOL: ;
1B77	CD3012	=3953	CALL FDCMP ; Pointer = end_pointer ?

LOC	OBJ	LINE	SOURCE STATEMENT
1B7A	CA8D1B	=3954	JZ DELEOB ;
1B7D	23	=3955	INX H ; Inx past <LF>
		=3956	;
1B7E	OB	=3957	DCX B ; Dec line count
1B7F	78	=3958	MOV A,B ;
1B80	B1	=3959	ORA C ; 'BC' = ZERO ?
1B81	C26B1B	=3960	JNZ DEL1L ;
		=3961	;
1B84	2200E8	=3962	SHLD POINT ;
1B87	2202E8	=3963	SHLD CHAR ;
1B8A	C3C11B	=3964	JMP DELEX ;
		=3965 ;	
		=3966 ;	
		=3967 ;	
		=3968	DELEOB: ;
1B8D	2AFCEF	=3969	LHLD SPOINT ;
1B90	EB	=3970	XCHG ;
1B91	2A00E8	=3971	LHLD POINT ;
1B94	CD3012	=3972	CALL FDCMP ;
1B97	CACA1B	=3973	JZ DFILE ; If SPOINT=POINT then entire file deleted.
		=3974	;
1B9A	2B	=3975	DCX H ; Dec point to last line feed in the file
1B9B	22FEEF	=3976	SHLD EPOINT ; Save as end pointer
1B9E	CD6913	=3977	CALL RBULF ; Read backward from old point.
1BA1	2200E8	=3978	SHLD POINT ;
1BA4	2202E8	=3979	SHLD CHAR ; Move to the beginning of the last line.
		=3980	;
1BA7	CDA826	=3981	CALL CLRL ; Clear line
1BAA	CD9D26	=3982	CALL BLINE ; Goto the beginning of the line
1BAD	21272B	=3983	LXI H,EOBMES ;
1BB0	CDE125	=3984	CALL STRING ; Print the [EOB] message
		=3985	;
1BB3	3A05E8	=3986	LDA LINE ;
1BB6	B7	=3987	ORA A ; Set flags
1BB7	CAC11B	=3988	JZ DELEX ;
		=3989	;
1BBA	3D	=3990	DCR A ;
1BBB	3205E8	=3991	STA LINE ;
1BBE	CDED26	=3992	CALL REVLF ; Skip back to previous line.
		=3993	DELEX: ;
1BC1	CDE615	=3994	CALL PACK ; Pack and remove the nulls
1BC4	CD7224	=3995	CALL PLINE ; Print the current line
1BC7	C3DB1B	=3996	JMP DELLEX ; Exit from the delete lines routine.
		=3997 ;	
		=3998 ;	
		=3999 ;	
		=4000 ;	The the start of line pointer = SPOINT
		=4001 ;	
		=4002	DFILE: ;
1BCA	2AFCEF	=4003	LHLD SPOINT ;
1BCD	360A	=4004	MVI M,LF ;
1BCF	22FEEF	=4005	SHLD EPOINT ; Reset End_of_text pointer.
1BD2	2200E8	=4006	SHLD POINT ; Reset Start_of_line_pointer.
1BD5	2202E8	=4007	SHLD CHAR ; Reset character pointer.
1BD8	CD7224	=4008	CALL PLINE ; Print the top line.
		=4009	DELLEX: ;
1BDB	C9	=4010	RET ; Return from <DLINES>
		=4011 ;	
		=4012 ;	=====
		=4013 ;	*
		=4014 ;	Function : DTEOLR (delete to end of line routine) *
		=4015 ;	Inputs : CHAR : Pointer to the current character. *
		=4016 ;	Outputs : None. *
		=4017 ;	Destroys : H,L,Acc and flags. *
		=4018 ;	Calls : INCNUL, CLRL and GOTO. *
		=4019 ;	*
		=4020 ;	Description : This routine deletes all the characters *
		=4021 ;	between the current character pointer (CHAR) and the end *
		=4022 ;	of the current line (marked with a line feed). When a *
		=4023 ;	character is deleted it is simply replaced by a null and *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 71

LOC	OBJ	LINE	SOURCE STATEMENT
		=4024 ;*	the null counter updated. On exit the character pointer
		=4025 ;*	with be left pointing at the line feed on the end of the
		=4026 ;*	line.
		=4027 ;*	
		=4028 ;=====	
		=4029 ;	
		=4030 DTEOLR:	; Start of Delete-to-end-of-line routine
1BDC	2A02E8	=4031 LHL	CHAR ; Get current character pointer.
		=4032 DEOL05:	
1BDF	7E	=4033 MOV	A,M ; Fetch character from text
1BE0	FE0A	=4034 CPI	LF ; End_of_line char ?
1BE2	CAEE1B	=4035 JZ	DEOL10 ; Yes - Then exit
		=4036 ;	
1BE5	3600	=4037 MVI	M,NULL ; Replace character with a null (i.e. del it)
1BE7	CD041C	=4038 CALL	INCNULL ;
		=4039 ;	
1BEA	23	=4040 INX	H ; Inc text pointer to next character
1BEB	C3DF1B	=4041 JMP	DEOL05 ;
		=4042 ;	
		=4043 DEOL10:	
1BEE	2202E8	=4044 SHLD	CHAR ; Save pointer to <LF> as new character
		=4045 ;	pointer.
1BF1	3A07E8	=4046 LDA	OLINE ;
1BF4	3205E8	=4047 STA	LINE ; Restore old_line
1BF7	3A06E8	=4048 LDA	OCOL ;
1BFA	3204E8	=4049 STA	COL ; Restore old_column
		=4050 ;	
1BFD	CDB426	=4051 CALL	GOTO ; Move cursor to old cursor position.
1C00	CDA826	=4052 CALL	CLRL ; Clear to EOL.
		=4053 ;	
1C03	C9	=4054 RET	; Return from <DTEOLR>
		=4055 ;	
		=4056 ;=====	
		=4057 ;*	
		=4058 ;*	Function : INCNULL (Increment null counter).
		=4059 ;*	Inputs : None.
		=4060 ;*	Outputs : None.
		=4061 ;*	Destroys : Acc and flags.
		=4062 ;*	Calls : PACK.
		=4063 ;*	
		=4064 ;*	Description : This routine is called be DELETE whenever a
		=4065 ;*	character is deleted. It adds one to the null counter and
		=4066 ;*	if the counter reaches 256 then the file is automatically
		=4067 ;*	packed to remove all the nulls.
		=4068 ;*	
		=4069 ;=====	
		=4070 ;	
		=4071 INCNULL:	; Start of increment null counter routine
1C04	3A0BE8	=4072 LDA	NULLS ;
1C07	3C	=4073 INR	A ; Increment the NULL counter
1C08	C2101C	=4074 JNZ	INCNO5 ;
		=4075 ;	
1C0B	CDE615	=4076 CALL	PACK ; If Null count = 256 (00H) then pack.
1C0E	3E00	=4077 MVI	A,0 ;
		=4078 INCNO5:	
1C10	320BE8	=4079 STA	NULLS ;
1C13	C9	=4080 RET	
		=4081 ;	
		=4082 ;=====	
		=4083 ;*	
		=4084 ;*	Function : NULLUP (update null counter routine)
		=4085 ;*	Inputs : BC : Number of nulls added to the file.
		=4086 ;*	Outputs : None.
		=4087 ;*	Destroys : Acc and flags.
		=4088 ;*	Calls : PACK.
		=4089 ;*	
		=4090 ;*	Description : The routine updates the file null counter
		=4091 ;*	with the number of characters deleted (replaced by nulls)
		=4092 ;*	and packs the file if this value goes above 255 (0FFH).
		=4093 ;*	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 72

LOC	OBJ	LINE	SOURCE STATEMENT
		=4094 ;	=====
		=4095 ;	
		=4096 NULLUP:	; Start of null counter update routine
1C14	78	=4097	MOV A,B ; Fetch high byte of the number of nulls added
1C15	B7	=4098	ORA A ;
1C16	C2231C	=4099	JNZ NUP05 ; If zero may not have to pack.
		=4100	;
1C19	3A0BE8	=4101	LDA NULLS ;
1C1C	81	=4102	ADD C ; Add low byte to NULL count.
1C1D	320BE8	=4103	STA NULLS ;
1C20	D2261C	=4104	JNC NUP10 ; If null count >255 then pack
		=4105	;
		=4106 NUP05:	; Come here if high byte # zero (must pack)
1C23	CDE615	=4107	CALL PACK ; Pack the file in memory
		=4108 NUP10:	;
1C26	C9	=4109	RET ; Return from <NULLUP> routine.
		=4110 ;	
		=4111 ;	End of editor module <DELROU.A85>
		4112	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 73

LOC	OBJ	LINE	SOURCE STATEMENT
		4113 ;	
		4114 ;	Include cursor movement routines
		4115 ;	
		4116	\$INCLUDE(EDTMOVROU.A85)
		=4117 ;	-----
		=4118 ;	
		=4119 ;	Cursor movement module for the UCT SABus editor.
		=4120 ;	-----
		=4121 ;	
		=4122 ;	This module contains (in order) the following subroutines:
		=4123 ;	
		=4124 ;	
		=4125 ;	a: LINROU - Handles any lines starting with the ASCII
		=4126 ;	character 'L'.
		=4127 ;	b: MOVENL - Moves the cursor n lines down.
		=4128 ;	c: MOVEBL - Moves the cursor n lines up.
		=4129 ;	d: UPROU - Moves the cursor 1 line up.
		=4130 ;	e: DOWNRO - Moves the cursor 1 line down.
		=4131 ;	f: RIGHTR - Moves the cursor 1 character right.
		=4132 ;	g: LEFTRO - Moves the cursor 1 character left.
		=4133 ;	
		=4134 ;	-----
		=4135 ;	
		=4136	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=4137 ;	=====
		=4138 ;*	*
		=4139 ;*	Function : LINEROU (Line Routine) *
		=4140 ;*	Inputs : Command Line. *
		=4141 ;*	Outputs : None. *
		=4142 ;*	Destroys : A,D,E and flags. *
		=4143 ;*	Calls : MOVENL and ERRSTR. *
		=4144 ;*	*
		=4145 ;*	Description : This routine handles the 'L' command. *
		=4146 ;*	*
		=4147 ;	=====
		=4148 ;	
		=4149 LINROU:	; Start of line down routine.
1C27	3A18E8	=4150 LDA BUF+1	; Fetch second character in command buffer.
1C2A	FE04	=4151 CPI EOT	; EOT ?
1C2C	C2381C	=4152 JNZ LINRER	; No - Then line down routine error.
		=4153	;
1C2F	110100	=4154 LXI D,1	; Set line counter = 1.
1C32	CD3F1C	=4155 CALL MOVENL	; Move down N lines (N = DE).
1C35	C33E1C	=4156 JMP LINREX	; Exit routine.
		=4157	;
		=4158 LINRER:	;
1C38	213E2B	=4159 LXI H,IVCMES	;
1C3B	CD8723	=4160 CALL ERRSTR	; Print "invalid command"
		=4161 LINREX:	;
1C3E	C9	=4162 RET	; Return from <LINROU>
		=4163 ;	
		=4164 ;	=====
		=4165 ;*	*
		=4166 ;*	Function : MOVENL (Move N Lines) *
		=4167 ;*	Inputs : Command line. *
		=4168 ;*	Outputs : None. *
		=4169 ;*	Destroys : A,D,E,H,L and flags. *
		=4170 ;*	Calls : RESCUR, QNULLF, DCMP, FDCMP, GOBACK and *
		=4171 ;*	NEWSCN. *
		=4172 ;*	*
		=4173 ;*	Description : This routine handles the [+]nnnnL command. *
		=4174 ;*	*
		=4175 ;	=====
		=4176 ;	
		=4177 MOVENL:	; Start of Move N lines.
1C3F	CD8F25	=4178 CALL RESCUR	; Restore cursor.
1C42	CD7723	=4179 CALL QNULLF	; Null file ?
1C45	CA8F1C	=4180 JZ MFEXIT	; Yes - Then return.
		=4181	;
1C48	3E00	=4182 MVI A,FALSE	;
1C4A	3211E8	=4183 STA RESFLG	; Do not restore cursor after move.
1C4D	212800	=4184 LXI H,MAXMOV	; Load HL register pair with the largest
		=4185	; Number of lines to move and display.
1C50	CD43F0	=4186 CALL DCMP	; Compare number of lines and max number of lines.
1C53	D2811C	=4187 JNC DISMOV	; If number entered < Max number then display.
		=4188 ;	
		=4189 ;	-----
		=4190 ;	
		=4191 ;	If N_Lines > MAXMOV then do not scroll the screen.
		=4192 ;	
		=4193 ;	-----
		=4194 ;	
1C56	42	=4195 MOV B,D	;
1C57	4B	=4196 MOV C,E	; Move number of lines to BC.
1C58	2AFEEF	=4197 LHLD EPOINT	; Load pointer to the end of the text.
1C5B	EB	=4198 XCHG	;
1C5C	2A00E8	=4199 LHLD POINT	; Load start adr of current line.
		=4200	;
		=4201 NXLINE:	;
1C5F	CD3012	=4202 CALL FDCMP	; Text_ptr = Eot_ptr ?
1C62	CA771C	=4203 JZ POSFOU	; Yes - Then position found.
		=4204	;
1C65	7E	=4205 MOV A,M	; Load character for text.
1C66	FE0A	=4206 CPI LF	; End_of_line ?

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 75

LOC	OBJ	LINE	SOURCE STATEMENT
1C68	CA6F1C	=4207	JZ ALINE ;
		=4208	; ;
1C6B	23	=4209	INX H ;
1C6C	C35F1C	=4210	JMP NXLINE ;
		=4211	; ;
		=4212	ALINE: ;
1C6F	0B	=4213	DCX B ; Dec number of lines to move.
1C70	23	=4214	INX H ; Inc past Line feed.
1C71	78	=4215	MOV A,B ;
1C72	B1	=4216	ORA C ; BC = zero ?
1C73	C25F1C	=4217	JNZ NXLINE ; Loop until Number = zero.
1C76	23	=4218	INX H ;
		=4219	; ;
		=4220	POSF0U: ;
1C77	2B	=4221	DCX H ;
1C78	CD3612	=4222	CALL GOBACK ; Using HL as CHAR find POINT.
1C7B	CD7F12	=4223	CALL NEWSN ; New screen.
1C7E	C38F1C	=4224	JMP MFEXIT ;
		=4225	; ;
		=4226	;-----;
		=4227	; ;
		=4228	; If N_Lines <= MAXMOV Then scroll the screen N times. ;
		=4229	; ;
		=4230	;-----;
		=4231	; ;
		=4232	DISMOV: ;
1C81	7A	=4233	MOV A,D ;
1C82	B3	=4234	ORA E ; If number = zero then exit
1C83	CA8F1C	=4235	JZ MFEXIT ;
		=4236	; ;
1C86	1B	=4237	DCX D ; Dec number of lines to move
1C87	D5	=4238	PUSH D ; Save on stack.
1C88	CD361D	=4239	CALL DOWNRO ; Move down.
1C8B	D1	=4240	POP D ;
1C8C	CA811C	=4241	JZ DISMOV ; Loop until number=0 or EOB=true
		=4242	; ;
		=4243	MFEXIT: ;
1C8F	C9	=4244	RET ; Return from <DISMOV>
		=4245	; ;
		=4246	;=====
		=4247	; * ;
		=4248	; * Function : MOVEBL (Move Backwards lines) * ;
		=4249	; * Inputs : Command Line. * ;
		=4250	; * Outputs : None. * ;
		=4251	; * Destroys : A,D,E and flags. * ;
		=4252	; * Calls : RESCUR, QNULLF, DCMP, FDCMP, GOBACK, BRROU * ;
		=4253	; * NEWSN and UPROU. * ;
		=4254	; * ;
		=4255	; * Description : This routine handles the -nnnnL command. * ;
		=4256	; * ;
		=4257	;=====
		=4258	; ;
		=4259	; Same as above but goes backwards. ;
		=4260	; ;
		=4261	MOVEBL: ;
1C90	CD8F25	=4262	CALL RESCUR ; Restore cursor.
1C93	CD7723	=4263	CALL QNULLF ;
1C96	CAE41C	=4264	JZ MBEXIT ;
		=4265	; ;
1C99	3E00	=4266	MVI A,FALSE ;
1C9B	3211E8	=4267	STA RESFLG ; Do not restore cursor after move.
1C9E	212800	=4268	LXI H,MAXMOV ; Load HL register pair with the largest
		=4269	; Number of lines to move and display.
1CA1	CD43F0	=4270	CALL DCMP ; Compare number of lines and max number of lines.
1CA4	D2D61C	=4271	JNC DISBM0V ; If number entered < Max number then display.
		=4272	; ;
1CA7	42	=4273	MOV B,D ;
1CA8	4B	=4274	MOV C,E ; Move number of lines to BC.
1CA9	2AFCEB	=4275	LHLD SPOINT ; Load pointer to the end of the text.
1CAC	EB	=4276	XCHG ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 76

LOC	OBJ	LINE	SOURCE STATEMENT
1CAD	2A00E8	=4277	LHLD POINT ; Load start adr of current line.
		=4278	;
		=4279	MOVB05: ;
1CB0	CD3012	=4280	CALL FDCMP ;
1CB3	CAD01C	=4281	JZ MOVB15 ;
		=4282	;
1CB6	7E	=4283	MOV A,M ; Load character for text.
1CB7	FE0A	=4284	CPI LF ; End_of_line ?
1CB9	CAC01C	=4285	JZ MOVB10 ;
		=4286	;
1CBC	2B	=4287	DCX H ;
1CBD	C3B01C	=4288	JMP MOVB05 ;
		=4289	;
		=4290	MOVB10: ;
1CC0	0B	=4291	DCX B ;
1CC1	2B	=4292	DCX H ;
1CC2	78	=4293	MOV A,B ;
1CC3	B1	=4294	ORA C ; BC = zero ?
1CC4	C2B01C	=4295	JNZ MOVB05 ; Loop until Number = zero.
		=4296	;
1CC7	CD3612	=4297	CALL GOBACK ;
1CCA	CD7F12	=4298	CALL NEWSCHN ;
1CCD	C3E41C	=4299	JMP MBEXIT ;
		=4300	MOVB15: ; If pointer = Start_pointer_then
1CD0	CDD922	=4301	CALL BRROU ; Set pointers for Begin_range_routine.
1CD3	C3E41C	=4302	JMP MBEXIT ;
		=4303	;
		=4304	;
		=4305	;
		=4306	Display cursor movement. ;
		=4307	;
		=4308	;
		=4309	;
		=4310	DISBMOV: ;
1CD6	7A	=4311	MOV A,D ;
1CD7	B3	=4312	ORA E ; If number = zero then exit
1CD8	CAE41C	=4313	JZ MBEXIT ;
		=4314	;
1CDB	1B	=4315	DCX D ; Dec number of lines to move
1CDC	D5	=4316	PUSH D ; Save on stack.
1CDD	CDE51C	=4317	CALL UPROU ; Move down.
1CE0	D1	=4318	POP D ;
1CE1	CAD61C	=4319	JZ DISBMOV ; Loop until number=0 or EOB=true
		=4320	;
		=4321	MBEXIT: ;
1CE4	C9	=4322	RET ;
		=4323	;
		=4324	;
		=4325	;
		=4326	Function : UPROU (UP routine) *
		=4327	Inputs : None. *
		=4328	Outputs : None. *
		=4329	Destroys : A,B,C,D,E,H,L and flags. *
		=4330	Calls : QSOB, RBULF, VERTP, BLIN, REVL, CLRL, PLIN *
		=4331	and PLIN. *
		=4332	;
		=4333	Description : The routine handles the up cursor key. *
		=4334	;
		=4335	;
		=4336	;
		=4337	UPROU: ;
1CE5	CD1913	=4338	CALL QSOB ; Start of buffer ?
1CE8	C2321D	=4339	JNZ UPSOB ;
		=4340	;
1CEB	CD6913	=4341	CALL RBULF ; Read backwards until line feed.
1CEE	2200E8	=4342	SHLD POINT ; Save in pointer.
1CF1	CDBD14	=4343	CALL VERTP ; Position cursor
1CF4	2202E8	=4344	SHLD CHAR ;
1CF7	CD9D26	=4345	CALL BLIN ;
1CFA	CDED26	=4346	CALL REVL ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 77

LOC	OBJ	LINE	SOURCE STATEMENT
		=4347	
1CFD	3A05E8	=4348	LDA LINE ; Load current line number.
1D00	FE00	=4349	CPI 0 ; Top line of the screen ?
1D02	CA0C1D	=4350	JZ TOPOS ; If not Top_of_screen then
		=4351	
1D05	3D	=4352	DCR A ; LINE = LINE - 1
1D06	3205E8	=4353	STA LINE ; Update line number.
		=4354	
1D09	C32D1D	=4355	JMP UPR05
		=4356	
		=4357	TOPOS:
1D0C	3A08E8	=4358	LDA EOBLIN ; Load pos of EOB.
1D0F	FE16	=4359	CPI LSTLIN
1D11	F2181D	=4360	JP EOBOK ; IF EOBLIN < LSTLIN THEN
		=4361	EOBLIN = EOBLIN + 1 ;
1D14	3C	=4362	INR A
1D15	3208E8	=4363	STA EOBLIN
		=4364	
		=4365	EOBOK:
1D18	3A0EE8	=4366	LDA LCLINE ; Load lowest_correct_line pointer.
1D1B	FE16	=4367	CPI LSTLIN ; Is it the at the last line on the page.
1D1D	CA241D	=4368	JZ NOMORE ; If it is, then cannot increment the pointer.
		=4369	
1D20	3C	=4370	INR A ; Inc pointer, because the screen rolled down
		=4371	by one line.
1D21	320EE8	=4372	STA LCLINE ; Save new lowest_correct_line.
		=4373	
		=4374	NOMORE:
1D24	CDA826	=4375	CALL CLRL
1D27	CD7224	=4376	CALL PLINE
1D2A	CD9D26	=4377	CALL BLINE
		=4378	UPR05:
1D2D	3E00	=4379	MVI A,00H ; Set zero flag (Up call OK).
1D2F	C3341D	=4380	JMP UPEXIT
		=4381	
		=4382	UPSOB:
1D32	3EFF	=4383	MVI A,OFFH ; Clear zero flag (i.e. start of buffer).
		=4384	
		=4385	UPEXIT:
1D34	B7	=4386	ORA A ; Set flags before returning.
1D35	C9	=4387	RET ; Return from <UPROU>
		=4388	
		=4389	=====
		=4390	;
		=4391	Function : DOWNRO (DOWN routine)
		=4392	Inputs : None.
		=4393	Outputs : None.
		=4394	Destroys : A,B,C,D,E,H,L and flags.
		=4395	Calls : RULF, NLINE, INCLIN, QCLINE, VERTP, BLINE
		=4396	REVLf, CLRL, PLINE, STRING, CLRS and PLINE.
		=4397	
		=4398	Description : The routine handles the down cursor key.
		=4399	
		=4400	=====
		=4401	
		=4402	DOWNRO:
1D36	CD8A13	=4403	CALL RULF ; Read until line feed. (Adr = HL)
1D39	E5	=4404	PUSH H
1D3A	CDD612	=4405	CALL QE0B
1D3D	E1	=4406	POP H ; Restore start of next line.
1D3E	C2681D	=4407	JNZ DWNEOB
		=4408	
1D41	2200E8	=4409	SHLD POINT ; Save as start of current line.
1D44	CDBD14	=4410	CALL VERTP
1D47	2202E8	=4411	SHLD CHAR ; Save as current char pointer
		=4412	
1D4A	CDE426	=4413	CALL NLINE
1D4D	CDE426	=4414	CALL NLINE ; Down 2 lines.
1D50	CDED26	=4415	CALL REVLf ; Up 1 line (bottom line alway blank)
1D53	CD5A12	=4416	CALL INCLIN ; Inc line num.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 78

LOC	OBJ	LINE	SOURCE STATEMENT
1D56	CDBD12	=4417	CALL QCLINE ; Is this line correct ?
1D59	C8	=4418	RZ ; Return if line ok
		=4419	;
1D5A	CD7224	=4420	CALL PLINE ;
1D5D	3A05E8	=4421	LDA LINE ;
1D60	320EE8	=4422	STA LCLINE ;
1D63	3E00	=4423	MVI A,00H ; Set zero flag.
1D65	C3911D	=4424	JMP DWNRET ;
		=4425	;
		=4426	DWNEOB: ;
1D68	3A05E8	=4427	LDA LINE ;
1D6B	FE16	=4428	CPI LSTLIN ; Last line on the screen ?
1D6D	C2741D	=4429	JNZ NOTBOS ;
		=4430	;
1D70	3D	=4431	DCR A ;
1D71	3205E8	=4432	STA LINE ;
		=4433	NOTBOS: ;
1D74	3208E8	=4434	STA EOBLIN ; Store position of EOB on screen.
1D77	CDE426	=4435	CALL NLINE ;
1D7A	CDE426	=4436	CALL NLINE ;
1D7D	CDED26	=4437	CALL REVLF ; Keep bottom line blank.
1D80	21272B	=4438	LXI H,EOBMES ;
1D83	CDE125	=4439	CALL STRING ;
1D86	CDAE26	=4440	CALL CLRS ;
1D89	CDED26	=4441	CALL REVLF ;
1D8C	CD9D26	=4442	CALL BLINE ;
1D8F	3EFF	=4443	MVI A,OFFH ; Clear zero flag
		=4444	;
		=4445	DWNRET: ;
1D91	B7	=4446	ORA A ; Set flags before returning.
1D92	C9	=4447	RET ;
		=4448	;
		=4449	=====
		=4450	;* *
		=4451	;* Function : RIGHTR (Right routine) *
		=4452	;* Inputs : None. *
		=4453	;* Outputs : None. *
		=4454	;* Destroys : A,B,C,D,E,H,L and flags. *
		=4455	;* Calls : FDCMP, QCHAR, PCUR and DOWNRO *
		=4456	;* *
		=4457	;* Description : The routine handles the right cursor key. *
		=4458	;* *
		=4459	=====
		=4460	;
		=4461	RIGHTR: ; Start of the right cursor move routine;
		=4462	;
1D93	2AFEEF	=4463	LHLD EPOINT ;
1D96	EB	=4464	XCHG ;
1D97	2A02E8	=4465	LHLD CHAR ;
1D9A	CD3012	=4466	CALL FDCMP ; Compare char pointer and End pointers
1D9D	CAC91D	=4467	JZ RCEOB ; Equal then do not move.
		=4468	;
1DA0	2A02E8	=4469	LHLD CHAR ;
		=4470	RCNEB: ; REPEAT
1DA3	23	=4471	INX H ; POINTER := POINTER + 1
1DA4	7E	=4472	MOV A,M ; CHAR := TEXT(POINTER)
1DA5	FE00	=4473	CPI NULL ;
1DA7	CAA31D	=4474	JZ RCNEB ; UNTIL CHAR <> NULL
		=4475	;
1DAA	E5	=4476	PUSH H ;
1DAB	2A02E8	=4477	LHLD CHAR ;
1DAE	7E	=4478	MOV A,M ;
1DAF	E1	=4479	POP H ;
		=4480	;
1DB0	2202E8	=4481	SHLD CHAR ;
1DB3	FE0A	=4482	CPI LF ; LINE FEED ? , IE NOW AT START OF NEXT LINE
1DB5	CAC11D	=4483	JZ NEXTL ; If char = <LF> then goto next_line
		=4484	;
1DB8	CD9914	=4485	CALL QCHAR ; Set new relative character position.
1DBB	CD0615	=4486	CALL PCUR ; Position cursor.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 79

LOC	OBJ	LINE	SOURCE STATEMENT
1DBE	C3C91D	=4487	JMP RCEOB ; Exit <RIGHT_ROUTINE>
		=4488	NEXTL: ;
1DC1	3E00	=4489	MVI A,00H ;
1DC3	320AE8	=4490	STA RELCH ; Reset Relative character pos to Zero.
1DC6	CD361D	=4491	CALL DOWNRO ; Move cursor down 1 line.
		=4492	RCEOB: ;
1DC9	C9	=4493	RET ;
		=4494	;
		=4495	=====
		=4496	;
		=4497	Function : LEFTRO (LEFT routine) *
		=4498	Inputs : None. *
		=4499	Outputs : None. *
		=4500	Destroys : A,B,C,D,E,H,L and flags. *
		=4501	Calls : FDCMP, QCHAR, PCUR, UPROU and RULF. *
		=4502	;
		=4503	Description : The routine handles the left cursor key. *
		=4504	;
		=4505	=====
		=4506	;
		=4507	LEFTRO: ; Left cursor routine.
1DCA	2AFCEF	=4508	LHLD SPOINT ; Load start of text pointer.
1DCD	EB	=4509	XCHG ;
1DCE	2A02E8	=4510	LHLD CHAR ; Load character pointer.
1DD1	CD3012	=4511	CALL FDCMP ;
1DD4	CA001E	=4512	JZ LEFTRE ; If CHAR = SPOINT then exit left routine.
		=4513	;
		=4514	LEFTR1: ; Repeat
1DD7	2B	=4515	DCX H ; Pointer := pointer + 1
1DD8	7E	=4516	MOV A,M ; Char := text(pointer)
1DD9	FE00	=4517	CPI NULL ;
1ddb	CAD71D	=4518	JZ LEFTR1 ; Until Char <> Null
		=4519	;
1DDE	2202E8	=4520	SHLD CHAR ; Save new character pointer.
1DE1	7E	=4521	MOV A,M ;
1DE2	FE0A	=4522	CPI LF ; If <LF> then goto end of previous line.
1DE4	CAF01D	=4523	JZ BEGINL ;
		=4524	;
1DE7	CD9914	=4525	CALL QCHAR ; Set new relative character position.
1DEA	CD0615	=4526	CALL PCUR ; Position cursor.
1DED	C3001E	=4527	JMP LEFTRE ; Exit <LEFT_ROUTINE>
		=4528	;
		=4529	BEGINL: ; Beginning of line.
1DF0	CDE51C	=4530	CALL UPROU ; Move up a line.
1DF3	CD8A13	=4531	CALL RULF ; Move cursor to char after the <LF>
1DF6	2B	=4532	DCX H ; Dec pointer to <LF>.
1DF7	2202E8	=4533	SHLD CHAR ; Save as character pointer.
1DFA	CD9914	=4534	CALL QCHAR ; Set new relative character position.
1DFD	CD0615	=4535	CALL PCUR ; Position cursor.
		=4536	LEFTRE: ;
1E00	C9	=4537	RET ; Exit point for <LEFTRO>.
		=4538	;
		=4539	End of "EDTMOVROU.A85".
		4540	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 81

LOC	OBJ	LINE	SOURCE STATEMENT
		=4560 ;	=====
		=4561 ;*	
		=4562 ;*	Function : CUTROU (CUT routine)
		=4563 ;*	Inputs : Input buffer = command (the 1st character in
		=4564 ;*	the buffer is a 'C')
		=4565 ;*	Outputs : None
		=4566 ;*	Destroys : All
		=4567 ;*	Calls : CONCAP, QNUM, READNU, PACK, FDCMP, SUB16,
		=4568 ;*	KEYSTK, DLINEs and ERRSTR.
		=4569 ;*	
		=4570 ;*	Description: This routine (the one that handles the 'CUT'
		=4571 ;*	command) is called whenever the command line begins with
		=4572 ;*	the letter 'C'. This routine therefore 1st checks for the
		=4573 ;*	rest of the 'CUT' command - the 'UT' part of it.
		=4574 ;*	
		=4575 ;*	The following syntaxs are allowed:
		=4576 ;*	
		=4577 ;*	"CUTnnnnL" - e.g. "CUT12L" - This will move the current
		=4578 ;*	line and the next 11 lines from the text area
		=4579 ;*	into the CUT buffer.
		=4580 ;*	
		=4581 ;*	"CUTL" - This will moves the current line from the text
		=4582 ;*	area into CUT buffer.
		=4583 ;*	
		=4584 ;*	"CUTOL" - This deallocates the CUT buffer, allowing a
		=4585 ;*	bigger text file to be loaded.
		=4586 ;*	
		=4587 ;*	The CUT buffer is dynamically allocated whenever a CUT is
		=4588 ;*	performed, It is therefore not a fixed size. The CUT
		=4589 ;*	buffer occupies the same block of memory as the text area
		=4590 ;*	and it is therefore possible, if the text in memory is
		=4591 ;*	large, that it may be impossible to CUT a large section of
		=4592 ;*	the text.
		=4593 ;*	
		=4594 ;*	The following variables are modified by this routine:
		=4595 ;*	
		=4596 ;*	a) CUTBL - The CUT Buffer Length (the number of
		=4597 ;*	characters in the CUT buffer)
		=4598 ;*	b) CUTLIB - The CUT Lines In Buffer counter (the number
		=4599 ;*	of text lines in the CUT buffer)
		=4600 ;*	c) CUTBUF - The CUT buffer flag is set TRUE when there
		=4601 ;*	is text in the CUT buffer.
		=4602 ;*	d) MAXTXT - The MAXimum TEXT pointer.
		=4603 ;*	
		=4604 ;*	Notes:
		=4605 ;*	
		=4606 ;*	The file is packed before the CUT is performed to make sure
		=4607 ;*	that no NULLs are moved to the CUT buffer.
		=4608 ;*	
		=4609 ;*	The CUT operation is a move operation and the text is there-
		=4610 ;*	fore removed from the text area. This is done using the
		=4611 ;*	delete lines routine DLINEs.
		=4612 ;*	
		=4613 ;	=====
		=4614 ;	
		=4615 CUTROU:	; Start of the <CUTROU> routine
		=4616	;
1E01	3A18E8	=4617 LDA BUF+1	; Point to next character (After the 'C')
1E04	CDB811	=4618 CALL CONCAP	;
1E07	FE55	=4619 CPI 'U'	;
1E09	C2E51E	=4620 JNZ CUTSE	; If Next_character <> 'U' then syntax error.
		=4621	;
1E0C	3A19E8	=4622 LDA BUF+2	;
1E0F	CDB811	=4623 CALL CONCAP	;
1E12	FE54	=4624 CPI 'T'	;
1E14	C2E51E	=4625 JNZ CUTSE	; If Next_character <> 'T' then syntax error.
		=4626 ;	
		=4627 ;	-----
		=4628 ;	
		=4629 ;	The command string is now checked as correct, look for "nnnL" or "L"

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 82

LOC	OBJ	LINE	SOURCE STATEMENT
		=4630 ;	
1E17	211AE8	=4631	LXI H,BUF+3 ; Point to the letter after the 'CUT'
1E1A	7E	=4632	MOV A,M ;
1E1B	CDB811	=4633	CALL CONCAP ;
1E1E	FE4C	=4634	CPI 'L' ; If Next_character <> 'L' then command
1E20	C2301E	=4635	JNZ NCUT1L ; not "CUTL"
		=4636 ;	
1E23	23	=4637	INX H ;
1E24	7E	=4638	MOV A,M ;
1E25	FE04	=4639	CPI EOT ;
1E27	C2D91E	=4640	JNZ CUTER1 ; If Next character <> EOT then syntax error.
		=4641 ;	
1E2A	110100	=4642	LXI D,1 ; Command = "CUTL<EOT>", set number of lines=1.
1E2D	C34C1E	=4643	JMP CUTOK ;
		=4644 ;	
		=4645 ;	
		=4646 ;	
		=4647 ;	Check for "CUT<Digit>....
		=4648 ;	
		=4649	NCUT1L: ;
1E30	CD0613	=4650	CALL QNUM ; Digit ?
1E33	C2D91E	=4651	JNZ CUTER1 ; No then error
		=4652 ;	
1E36	CD4A26	=4653	CALL READNU ; Read the number from the command line.
1E39	C2D91E	=4654	JNZ CUTER1 ;
		=4655 ;	
1E3C	7E	=4656	MOV A,M ; Fetch number terminating character.
1E3D	CDB811	=4657	CALL CONCAP ;
1E40	FE4C	=4658	CPI 'L' ; Check for lines.
1E42	C2D91E	=4659	JNZ CUTER1 ;
		=4660 ;	
1E45	23	=4661	INX H ;
1E46	7E	=4662	MOV A,M ;
1E47	FE04	=4663	CPI EOT ; Check that the 'L' was the last charater.
1E49	C2D91E	=4664	JNZ CUTER1 ;
		=4665 ;	
		=4666 ;	
		=4667 ;	
		=4668 ;	The command is correct and the number of lines to cut is stored in DE.
		=4669 ;	
		=4670	CUTOK: ;
1E4C	7A	=4671	MOV A,D ;
1E4D	B3	=4672	ORA E ;
1E4E	C2631E	=4673	JNZ CUTNZ ; If DE <> Zero then perform CUT.
		=4674 ;	
		=4675 ;	
		=4676 ;	
		=4677 ;	
		=4678 ;	The command = "CUTOL", clear the CUT buffer.
		=4679 ;	
1E51	2AFCEP	=4680	LHLD SPOINT ;
1E54	1100B0	=4681	LXI D,MAXLEN ;
1E57	19	=4682	DAD D ; HL = Start_ptr + Max text length
1E58	22F0E8	=4683	SHLD MAXTXT ;
		=4684 ;	
1E5B	3E00	=4685	MVI A,FALSE ;
1E5D	32EFE8	=4686	STA CUTBUF ;
1E60	C3EB1E	=4687	JMP CUTRET ; Exit cut routine.
		=4688 ;	
		=4689 ;	
		=4690 ;	
		=4691 ;	The number of lines is not zero. (Number of lines = DE)
		=4692 ;	
		=4693	CUTNZ: ;
1E63	42	=4694	MOV B,D ;
1E64	4B	=4695	MOV C,E ; BC = Number of lines to cut.
1E65	EB	=4696	XCHG ; Move Lines to cut into HL
1E66	22F2E8	=4697	SHLD CUTLIB ; Save as the number of lines in the cut buffer
		=4698 ;	
1E69	CDE615	=4699	CALL PACK ; Pack (Do not cut NULLs)

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 83

LOC	OBJ	LINE	SOURCE STATEMENT
1E6C	2AFEEF	=4700	LHLD EPOINT ;
1E6F	EB	=4701	XCHG ;
1E70	2A00E8	=4702	LHLD POINT ; Pointer setup as follows:
		=4703	; HL = Start of lines pointer and
		=4704	; DE = End of text pointer.
		=4705	CUT05: ;
1E73	CD3012	=4706	CALL FDCMP ; Pointer = end_pointer ?
1E76	CA8B1E	=4707	JZ CUT15 ; Yes - Then cut the rest of the file.
		=4708	; ;
1E79	7E	=4709	MOV A,M ; Load character.
1E7A	FE0A	=4710	CPI LF ; End of line character ?
1E7C	CA831E	=4711	JZ CUT10 ; Yes - Then dec lines to cut counter.
		=4712	; ;
1E7F	23	=4713	INX H ;
1E80	C3731E	=4714	JMP CUT05 ; Delete next character
		=4715	; ;
		=4716	CUT10: ;
1E83	23	=4717	INX H ; Inx past <LF>
1E84	0B	=4718	DCX B ; Dec line count
1E85	78	=4719	MOV A,B ;
1E86	B1	=4720	ORA C ; 'BC' = ZERO ?
1E87	C2731E	=4721	JNZ CUT05 ;
		=4722	; ;
1E8A	2B	=4723	DCX H ; Point to the last character to be cut
		=4724	; ;
		=4725	CUT15: ;
		=4726	; ;
		=4727	;-----
		=4728	; ;
		=4729	; HL Points to the last character to be cut.
		=4730	; ;
1E8B	EB	=4731	XCHG ;
1E8C	2A00E8	=4732	LHLD POINT ;
1E8F	EB	=4733	XCHG ;
1E90	CD911F	=4734	CALL SUB16 ; HL = Last_char_to_cut - Start_char_to_cut
		=4735	; (i.e. Size of cut buffer needed).
1E93	23	=4736	INX H ; Inc buffer size.
1E94	22F4E8	=4737	SHLD CUTBL ;
1E97	E5	=4738	PUSH H ;
1E98	1100B0	=4739	LXI D,MAXLEN ;
1E9B	2AFCEF	=4740	LHLD SPOINT ;
1E9E	19	=4741	DAD D ;
1E9F	D1	=4742	POP D ;
1EA0	CD911F	=4743	CALL SUB16 ; HL = Max_text - Cut_buffer_length
		=4744	; ;
1EA3	EB	=4745	XCHG ;
1EA4	2AFEEF	=4746	LHLD EPOINT ;
1EA7	CD43F0	=4747	CALL DCMP ; Start of required cut buffer > End ptr ?
		=4748	; ;
1EAA	D2DF1E	=4749	JNC CUTER2 ; Exit with error message
		=4750	; ;
		=4751	CUT20: ;
1EAD	EB	=4752	XCHG ;
1EAE	2B	=4753	DCX H ;
1EAF	22F0E8	=4754	SHLD MAXTXT ;
1EB2	23	=4755	INX H ;
1EB3	44	=4756	MOV B,H ;
1EB4	4D	=4757	MOV C,L ; BC = Start adr of Cut buffer.
		=4758	; ;
1EB5	2A00E8	=4759	LHLD POINT ; HL = Start of text to cut.
1EB8	EB	=4760	XCHG ;
1EB9	2AF4E8	=4761	LHLD CUTBL ;
1EBC	EB	=4762	XCHG ; DE = Number of bytes to cut.
		=4763	CUT25: ;
1EBD	CDF223	=4764	CALL KEYSTK ;
1EC0	7E	=4765	MOV A,M ; Fetch byte from text.
1EC1	02	=4766	STAX B ; Save in cut buffer.
1EC2	23	=4767	INX H ; Inc text pointer.
1EC3	03	=4768	INX B ; Inc cut buffer pointer
1EC4	1B	=4769	DCX D ; Dec byte counter.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 84

LOC	OBJ	LINE	SOURCE STATEMENT
1EC5	7A	=4770	MOV A,D ;
1EC6	B3	=4771	ORA E ; DE = Zero ?
1EC7	C2BD1E	=4772	JNZ CUT25 ; No - Then loop until it is.
		=4773	;
1ECA	2AF2E8	=4774	LHLD CUTLIB ;
1ECD	EB	=4775	XCHG ; Put line count in DE.
1ECE	CD441B	=4776	CALL DLINES ;
		=4777	;
1ED1	3EFF	=4778	MVI A,TRUE ;
1ED3	32EFE8	=4779	STA CUTBUF ; Set CUT buffer flag, Buffer full.
1ED6	C3EB1E	=4780	JMP CUTRET ; Exit CUT routine.
		=4781 ;	
		=4782 ;	
		=4783 ;	
		=4784 ;	Cut routine error exit points.
		=4785 ;	
		=4786	CUTER1: ; CUT error exit point #1: CUT syntax error
1ED9	217D2D	=4787	LXI H,CUTEM1 ;
1EDC	C3E81E	=4788	JMP CUTERR ;
		=4789	CUTER2: ; CUT error exit point #2: Not enough memory
1EDF	21AD2D	=4790	LXI H,CUTEM2 ;
1EE2	C3E81E	=4791	JMP CUTERR ;
		=4792	CUTSE: ;
1EE5	213E2B	=4793	LXI H,IVCMES ; Print general syntax error message.
		=4794	CUTERR: ;
1EE8	CD8723	=4795	CALL ERRSTR ;
		=4796	CUTRET: ;
1EEB	C9	=4797	RET ; Return from <CUTROU>
		=4798 ;	
		=4799 ;	
		=4800 ;*	*
		=4801 ;*	Function : CUTLRO (CUT buffer length routine) *
		=4802 ;*	Inputs : None *
		=4803 ;*	Outputs : None *
		=4804 ;*	Destroys : H,L,Acc and Flags *
		=4805 ;*	Calls : ERRSTR, PRINTD and STRING. *
		=4806 ;*	*
		=4807 ;*	Description : This routine is used to display the length of *
		=4808 ;*	CUT buffer, in lines and bytes. If the CUT_buffer flag is *
		=4809 ;*	FALSE then there is nothing in the CUT buffer and an error *
		=4810 ;*	message is printed. *
		=4811 ;*	*
		=4812 ;	
		=4813 ;	
		=4814	CUTLRO: ; Start of CUT buffer length routine
1EEC	3AEFE8	=4815	LDA CUTBUF ;
1EEF	FEFF	=4816	CPI TRUE ;
1EF1	C2151F	=4817	JNZ CUTBL5 ; If Cut_buffer full then
		=4818	; begin
1EF4	21232E	=4819	LXI H,CUTLM1 ;
1EF7	CD8723	=4820	CALL ERRSTR ; Write('The cut buffer occupies ');
1EFA	2AF4E8	=4821	LHLD CUTBL ;
1EFD	CDFA25	=4822	CALL PRINTD ; Write(Cut_buffer_length);
1F00	213C2E	=4823	LXI H,CUTLM2 ;
1F03	CDE125	=4824	CALL STRING ; Write(' bytes in ');
1F06	2AF2E8	=4825	LHLD CUTLIB ;
1F09	CDFA25	=4826	CALL PRINTD ; Write(Lines_in_cut_buffer);
1F0C	21472E	=4827	LXI H,CUTLM3 ;
1F0F	CDE125	=4828	CALL STRING ; Write(' lines.')
1F12	C31B1F	=4829	JMP CUTBLE ; end
		=4830	; else
		=4831	CUTBL5: ;
1F15	21DC2D	=4832	LXI H,PASEM1 ;
1F18	CD8723	=4833	CALL ERRSTR ; Writeln('Cut buffer empty');
		=4834	; endif
		=4835	CUTBLE: ;
1F1B	C9	=4836	RET ; Return from <CUTLRO>.
		=4837 ;	
		=4838 ;	
		=4839 ;*	*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 85

LOC	OBJ	LINE	SOURCE STATEMENT
		=4840 ;*	Function : PASTER (PASTE routine) *
		=4841 ;*	Inputs : None (PASTE command in the input buffer) *
		=4842 ;*	Outputs : None. *
		=4843 ;*	Destroys : All. *
		=4844 ;*	Calls : CUTPAD, SUB16, KEYSTK, GOTO, PLINE and ERRSTR *
		=4845 ;*	
		=4846 ;*	Description : This routine handles the PASTE command. It *
		=4847 ;*	checks to that there is text in the CUT buffer and if *
		=4848 ;*	there is it copies it into the text area at the current *
		=4849 ;*	position. *
		=4850 ;*	
		=4851 ;*	Before the routine can copy the text from the CUT buffer it *
		=4852 ;*	checks that there is enough room left in the text area. If *
		=4853 ;*	there is not enough room then an error message is printed. *
		=4854 ;*	
		=4855 ;*	Note: The PAD routine has a special entry point that is *
		=4856 ;*	used by PASTER which pads the text with n NULLS, where n *
		=4857 ;*	is given by the CUTBL (CUT buffer length word). *
		=4858 ;*	
		=4859 ;	=====
		=4860 ;	
		=4861 PASTER:	
1F1C	3AEFE8	=4862	LDA CUTBUF ;
1F1F	FEFF	=4863	CPI TRUE ;
1F21	C2841F	=4864	JNZ PASER1 ; If Cut_buffer <> TRUE then PASTE error.
		=4865	;
1F24	2A02E8	=4866	LHLD CHAR ;
1F27	E5	=4867	PUSH H ; Save old CHAR pointer on the stack
		=4868	;
1F28	2A00E8	=4869	LHLD POINT ;
1F2B	2202E8	=4870	SHLD CHAR ; Setup CHAR = Start of current line.
1F2E	CD9115	=4871	CALL CUTPAD ; PAD starting from the CHAR pointer.
		=4872	;
1F31	E1	=4873	POP H ;
1F32	2202E8	=4874	SHLD CHAR ; Restore CHAR pointer.
1F35	C28A1F	=4875	JNZ PASER2 ; If PAD error the exit.
		=4876	
		=4877 ;	
		=4878 ;	-----
		=4879 ;	
		=4880 ;	There is enough memory to perform the paste.
		=4881 ;	
1F38	2AF4E8	=4882	LHLD CUTBL ;
1F3B	44	=4883	MOV B,H ;
1F3C	4D	=4884	MOV C,L ; BC = CUT buffer length.
1F3D	E5	=4885	PUSH H ;
1F3E	1100B0	=4886	LXI D,MAXLEN ;
1F41	2AFCEF	=4887	LHLD SPOINT ;
1F44	19	=4888	DAD D ; HL = Max top of text
1F45	D1	=4889	POP D ; DE = CUT buffer length
1F46	CD911F	=4890	CALL SUB16 ; HL = Start adr of cut buffer
1F49	EB	=4891	XCHG ;
1F4A	2A00E8	=4892	LHLD POINT ;
		=4893 ;	
		=4894 ;	Register pairs setup as follows:
		=4895 ;	
		=4896 ;	BC = Cut Buffer length.
		=4897 ;	DE = Start adr of the cut buffer
		=4898 ;	HL = Start blank text area (move cut buffer to this area).
		=4899 ;	
		=4900 PAS05:	
1F4D	CDF223	=4901	CALL KEYSTK ; Check the keyboard.
1F50	1A	=4902	LDAX D ; Fetch character from the cut buffer
1F51	77	=4903	MOV M,A ; Put in text area
1F52	13	=4904	INX D ; Inc cut buffer ptr
1F53	23	=4905	INX H ; Inc text ptr
1F54	0B	=4906	DCX B ; Dec byte counter
1F55	78	=4907	MOV A,B ;
1F56	B1	=4908	ORA C ; BC = Zero ?
1F57	C24D1F	=4909	JNZ PAS05 ; No - Then loop and move another character

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 86

LOC	OBJ	LINE	SOURCE STATEMENT
		=4910	
1F5A	3E00	=4911	MVI A,FALSE ;
1F5C	3211E8	=4912	STA RESFLG ; Reset REStore cursor flag.
1F5F	3E17	=4913	MVI A,LSTLIN+1 ;
1F61	3208E8	=4914	STA EOBLIN ; Move End_Of_Buffer screen pointer off
		=4915	; the screen.
1F64	3A07E8	=4916	LDA OLINE ;
1F67	3205E8	=4917	STA LINE ; Restore old line position.
1F6A	320EE8	=4918	STA LCLINE ; Lowest correct line = Old line position
		=4919	; (will clear to end of screen).
1F6D	3E00	=4920	MVI A,00 ;
1F6F	3204E8	=4921	STA COL ; Set col position = 0.
1F72	320AE8	=4922	STA RELCH ; Set relative character position = 0.
1F75	CDB426	=4923	CALL GOTO ; Move to the beginning of the current line.
		=4924	;
1F78	2A00E8	=4925	LHLD POINT ;
1F7B	2202E8	=4926	SHLD CHAR ; Put pointer at the start of the 1st line
		=4927	; copied from the paste buffer.
1F7E	CD7224	=4928	CALL PLINE ; Print the next line.
1F81	C3901F	=4929	JMP PASRET ;
		=4930 ;	
		=4931 ;	-----
		=4932 ;	
		=4933 ;	Paste error exit points.
		=4934 ;	
		=4935	PASER1: ;
1F84	21DC2D	=4936	LXI H,PASEM1 ;
1F87	C38D1F	=4937	JMP PASERR ;
		=4938	PASER2: ;
1F8A	21F02D	=4939	LXI H,PASEM2 ;
		=4940	PASERR: ;
1F8D	CD8723	=4941	CALL ERRSTR ;
		=4942	PASRET: ;
1F90	C9	=4943	RET ; Return from paste routine.
		=4944 ;	
		=4945 ;	=====
		=4946 ;*	*
		=4947 ;*	Function : SUB16 (16 bit subtraction) *
		=4948 ;*	Inputs : DE : 16 bit number. *
		=4949 ;*	HL : 16 bit number. *
		=4950 ;*	Outputs : HL : The value of HL - DE. *
		=4951 ;*	Destroys : Acc and Flags. *
		=4952 ;*	Calls : None. *
		=4953 ;*	*
		=4954 ;*	Description : This routine does a 16 bit subtraction of the *
		=4955 ;*	DE register pair from the HL register pair. *
		=4956 ;*	*
		=4957 ;	=====
		=4958 ;	
		=4959	SUB16: ; Start of <SUB16> routine.
1F91	7D	=4960	MOV A,L ;
1F92	93	=4961	SUB E ;
1F93	6F	=4962	MOV L,A ; 8 bit subtraction: L = L - E (no carry)
		=4963	;
1F94	7C	=4964	MOV A,H ;
1F95	9A	=4965	SBB D ;
1F96	67	=4966	MOV H,A ; 8 bit subtraction: H = H - D (with carry from
		=4967	; the subtraction of E from L.
1F97	C9	=4968	RET ; Return from <SUB16> routine
		=4969 ;	
		=4970 ;	End of CUT/PASTE module (EDTCUTPAS.A85)
		4971	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 87

LOC	OBJ	LINE	SOURCE STATEMENT
		4972 ;	
		4973 ;	Include Insert routine (Plus support routines).
		4974 ;	
		4975 \$INCLUDE(EDTINSERT.A85)	
		=4976 ;	-----
		=4977 ;	
		=4978 ;	Insert module for the UCT SABus editor.
		=4979 ;	-----
		=4980 ;	
		=4981 ;	This module contains (in order) the following subroutines:
		=4982 ;	
		=4983 ;	a) INSERT - The insert routine is used when ever text is being
		=4984 ;	inserted into the text area.
		=4985 ;	b) IROU - The I routine is called when the 'I' command is
		=4986 ;	entered, it simply sets the insert flag.
		=4987 ;	
		=4988 ;	-----
		=4989 ;	
		=4990 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 88

LOC	OBJ	LINE	SOURCE STATEMENT
		=4991 ;	=====
		=4992 ;*	*
		=4993 ;*	Function : IROU (I command routine, goto insert mode) *
		=4994 ;*	Inputs : None. *
		=4995 ;*	Outputs : None. *
		=4996 ;*	Destroys : H,L,Acc and flags. *
		=4997 ;*	Calls : ERRSTR *
		=4998 ;*	*
		=4999 ;*	Description : This routine is called whenever the first *
		=5000 ;*	character of the command line is an 'I'. The routine first *
		=5001 ;*	checks that the command was 'I<EOT>'. If the command was *
		=5002 ;*	correct it sets the INSERT flag and clears then exchange *
		=5003 ;*	flag. The RESCUR (restore cursor flag) is left set. *
		=5004 ;*	*
		=5005 ;	=====
		=5006 ;	
		=5007 IROU:	; Start of 'I' routine.
1F98	3A18E8	=5008 LDA BUF+1	
1F9B	FE04	=5009 CPI EOT	; Command buffer = 'I<EOT>' ?
1F9D	CAA91F	=5010 JZ IROU05	; Yes - Then set insert mode.
		=5011	
		=5012 LXI H,IVCMES	
1FA3	CD8723	=5013 CALL ERRSTR	; Print invalid command error.
1FA6	C3B31F	=5014 JMP IROU10	
		=5015	
		=5016 IROU05:	; Come here if command = 'I<EOT>'
1FA9	3EFF	=5017 MVI A,TRUE	
1FAB	3215E8	=5018 STA INSFLG	; Insert_flag = True.
1FAE	3E00	=5019 MVI A,FALSE	
1FB0	3216E8	=5020 STA XFLG	; Xchange_flag = False.
		=5021	
		=5022 IROU10:	
1FB3	C9	=5023 RET	; Return from <IROU>
		=5024 ;	
		=5025 ;	=====
		=5026 ;*	*
		=5027 ;*	Function : INSERT (Insert characters in the text) *
		=5028 ;*	Inputs : B : The first character to be inserted. *
		=5029 ;*	Outputs : None. *
		=5030 ;*	Destroys : All. *
		=5031 ;*	Calls : IPAD, GETKEY, DELETE, LPRINT, PTOEOL, CLRL *
		=5032 ;*	QCHAR, PCUR, NLINE, REVLF, DOWNRO, UPDATE *
		=5033 ;*	PCUR, SETMODE, ERRSTR, MINIT, QINIT and PACK *
		=5034 ;*	*
		=5035 ;*	Description : This routine is used whenever characters are *
		=5036 ;*	inserted into the text area in memory. *
		=5037 ;*	*
		=5038 ;	=====
		=5039 ;	
		=5040 INSERT:	; Start of character insert routine
1FB4	78	=5041 MOV A,B	
1FB5	FE7F	=5042 CPI DEL	
1FB7	CAC01F	=5043 JZ DELKEY	; IF 1st char del then no PAD.
		=5044	
1FBA	CD7E15	=5045 CALL IPAD	; Pad the file with deletes to allow
		=5046	; the user to insert characters.
1FBD	C23C20	=5047 JNZ IPADER	; IF pad returns non zero then PAD error
		=5048	; (i.e. out of memory).
		=5049 DELKEY:	
1FC0	78	=5050 MOV A,B	; Restore character.
1FC1	2A02E8	=5051 LHLD CHAR	; Get current character pointer.
1FC4	C3D61F	=5052 JMP ENTRY2	
		=5053	
		=5054 INSNXT:	; Insert loop start adr.
		=5055	
1FC7	2A02E8	=5056 LHLD CHAR	; Get current character pointer.
1FCA	7E	=5057 MOV A,M	; Acc := TEXT (pointer)
1FCB	FE00	=5058 CPI NULL	; <NULL> ?
1FCD	C47E15	=5059 CNZ IPAD	; Pad if next character in text <> <NULL>
		=5060	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 89

LOC	OBJ	LINE	SOURCE	STATEMENT
1FD0	C23C20	=5061	JNZ	IPADER ; PAD error => out of memory.
		=5062		;
1FD3	CD9323	=5063	CALL	GETKEY ; Get key from the Keyboard.
		=5064		;
		=5065	ENTRY2:	; Entry point with character in Acc.
1FD6	FE1A	=5066	CPI	MODCCH ;
1FD8	CA2E20	=5067	JZ	EXIT1 ; If Char = Mode_Change then Exit(Insert)
		=5068		;
1FDB	FE1B	=5069	CPI	ESC ;
1FDD	CA3420	=5070	JZ	EXIT2 ; If char = <ESC> Then Exit(Insert)
		=5071		;
1FE0	FE0D	=5072	CPI	CR ;
1FE2	CA0720	=5073	JZ	INSCR ; If char = <CR> Then Start another line.
		=5074		;
1FE5	FE7F	=5075	CPI	DEL ;
1FE7	C2F01F	=5076	JNZ	NOTDEL ; If char = <DEL> Then Delete last character.
		=5077		;
1FEA	CD881A	=5078	CALL	DELETE ;
1FED	C3C71F	=5079	JMP	INSNXT ;
		=5080	NOTDEL:	; Else Text(Pointer) := Character
1FF0	77	=5081	MOV	M,A ;
1FF1	23	=5082	INX	H ; Inc character pointer.
1FF2	2202E8	=5083	SHLD	CHAR ;
		=5084		;
1FF5	CD2F24	=5085	CALL	LPRINT ; Echo the character inserted.
1FF8	CDD124	=5086	CALL	PTOEOL ; Print the rest of the line.
1FFB	CDA826	=5087	CALL	CLRL ; Clear the rest of the line.
1FFE	CD9914	=5088	CALL	QCHAR ; Set relative character position.
2001	CD0615	=5089	CALL	PCUR ; Move cursor to it's new position.
2004	C3C71F	=5090	JMP	INSNXT ; Loop.
		=5091 ;		
		=5092 ;		
		=5093 ;		
		=5094 ;		
		=5095 ;		
		=5096 ;		
		=5097	INSCR:	; Insert <LF> Section.
		=5098		;
2007	CDA826	=5099	CALL	CLRL ; Clear to the end of the line.
		=5100		;
200A	360A	=5101	MVI	M,LF ; Text (Pointer) := <LF>
		=5102		;
200C	3E00	=5103	MVI	A,00H ;
200E	320AE8	=5104	STA	RELCH ; Relch := 0 ; (* Start of a new line *)
		=5105		;
2011	3A05E8	=5106	LDA	LINE ;
2014	320EE8	=5107	STA	LCLINE ; Set Lowest_Correct_Line = Current_line.
		=5108		;
2017	3E17	=5109	MVI	A,LSTLIN+1 ;
2019	3208E8	=5110	STA	EOBLIN ; End_Of_buffer message of screen.
		=5111		;
201C	CDE426	=5112	CALL	NLINE ; Goto next line.
		=5113		;
201F	CDED26	=5114	CALL	REVLf ; Reverse line feed.
2022	CD361D	=5115	CALL	DOWNRO ;
2025	CDD513	=5116	CALL	UPDATE ;
2028	CD0615	=5117	CALL	PCUR ; Position cursor.
202B	C3C71F	=5118	JMP	INSNXT ; Loop.
		=5119 ;		
		=5120 ;		
		=5121 ;		
		=5122 ;		
		=5123 ;		
		=5124	EXIT1:	; 1st Exit point. (Mode_Change_Char typed)
202E	CDA213	=5125	CALL	SETMODE ; Change mode.
2031	C34820	=5126	JMP	INEXIT ; Exit.
		=5127		;
		=5128	EXIT2:	; 2nd Exit point. (<ESC> typed.)
2034	3EFF	=5129	MVI	A,TRUE ;
2036	320FE8	=5130	STA	ESCFLG ; Esc_Flag = True.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 90

LOC	OBJ	LINE	SOURCE STATEMENT
2039	C34820	=5131	JMP INEXIT ; Exit.
		=5132	IPADER: ;
203C	21B52B	=5133	LXI H, MEMFUL ;
203F	CD8723	=5134	CALL ERRSTR ; Print memory full error message.
2042	CD7612	=5135	CALL MINIT ; Mode init (= command mode).
2045	CDEA12	=5136	CALL QINIT ; Queue init (Kill all letters in input queue)
		=5137	INEXIT: ; Final exit from Insert.
2048	CDE615	=5138	CALL PACK ; Pack the file.
204B	C9	=5139	RET ; Return from <INSERT>
		=5140 ;	
		=5141 ;	End of "EDTINSERT.A85" module.
		5142	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 91

LOC	OBJ	LINE	SOURCE STATEMENT
		5143 ;	
		5144 ;	Include FILE_I/O Routines. (These handle the LOAD and SAVE functions)
		5145 ;	
		5146	\$INCLUDE(EDTFILEIO.A85)
		=5147 ;	-----;
		=5148 ;	
		=5149 ;	File I/O module for the UCT SABus editor.
		=5150 ;	-----;
		=5151 ;	
		=5152 ;	This module contains (in order) the following subroutines:
		=5153 ;	
		=5154 ;	a) CHKFN - Check file name. (Confirm use of default filename)
		=5155 ;	b) DELAY - Delay (Delay between sending characters to the PDP)
		=5156 ;	c) FILEIO - Main file IO routine.
		=5157 ;	d) LOADF - Load file.
		=5158 ;	e) LOADRO - Handles the "LOAD" command.
		=5159 ;	f) MAKECOM - Make command.
		=5160 ;	g) QCHAROK - Character OK? (Can the character be sent to the PDP?)
		=5161 ;	h) RFILE - Read file (This does the physical transfer.)
		=5162 ;	i) SAVEF - Save file.
		=5163 ;	j) SAVERO - Handles the "SAVE" command.
		=5164 ;	k) SCTPPF - Send command to the PDP.
		=5165 ;	l) SDPDWE - Send character to the PDP with echo.
		=5166 ;	m) WAITFA - Wait for acknowledge.
		=5167 ;	n) WAITFU - Wait for user (Simpy a half second delay)
		=5168 ;	
		=5169 ;	-----;
		=5170 ;	
		=5171	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 92

LOC	OBJ	LINE	SOURCE STATEMENT
		=5172 ;	=====
		=5173 ;*	
		=5174 ;*	Function : SAVERO (Save routine) *
		=5175 ;*	Inputs : BC = Pointer to the space following the 'SAVE' *
		=5176 ;*	string on the command line. *
		=5177 ;*	Outputs : None. *
		=5178 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=5179 ;*	Calls : PACK and FILEIO. *
		=5180 ;*	
		=5181 ;*	Description : This routine saves the current user text area *
		=5182 ;*	in a file on the PDP. The text area is first packed to *
		=5183 ;*	remove nulls. Most of the work of this routine is done by *
		=5184 ;*	the FILEIO routine. *
		=5185 ;*	
		=5186 ;	=====
		=5187 ;	
		=5188 SAVERO:	; Start of the <SAVERO>
		=5189	;
204C	CDE615	=5190	CALL PACK ; Pack the file before saving it.
204F	3E53	=5191	MVI A,'S' ; Setup KIT save command letter.
2051	CD5B20	=5192	CALL FILEIO ; Perform transfer.
		=5193	;
2054	C9	=5194	RET ; Return from the <SAVERO> routine.
		=5195 ;	
		=5196 ;	=====
		=5197 ;*	
		=5198 ;*	Function : LOADRO (Load routine) *
		=5199 ;*	Inputs : BC = Pointer to the space following the 'LOAD' *
		=5200 ;*	string on the command line. *
		=5201 ;*	Outputs : None. *
		=5202 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=5203 ;*	Calls : FILEIO. *
		=5204 ;*	
		=5205 ;*	Description : This routine loads a file from the PDP into *
		=5206 ;*	the user text area. As above most of the work is done by *
		=5207 ;*	the FILEIO routine. *
		=5208 ;*	
		=5209 ;	=====
		=5210 ;	
		=5211 LOADRO:	; Start of <LOADRO> routine
		=5212	;
2055	3E52	=5213	MVI A,'R' ; Setup KIT recall command letter.
2057	CD5B20	=5214	CALL FILEIO ; Perform transfer.
		=5215	;
205A	C9	=5216	RET ; Return from the <LOADRO> routine
		=5217 ;	
		=5218 ;	=====
		=5219 ;*	
		=5220 ;*	Function : FILEIO - File I/O routine (Used to transfer file *
		=5221 ;*	to and from the PDP/11 by using the SABus *
		=5222 ;*	interface program "KIT") *
		=5223 ;*	
		=5224 ;*	Inputs : A = File_IO command: (KIT command letter) *
		=5225 ;*	A='R' - Load file from PDP into text area. *
		=5226 ;*	A='S' - Save text area on the PDP. *
		=5227 ;*	BC = Pointer to the end of the 'Load' or 'Save' *
		=5228 ;*	command string. *
		=5229 ;*	
		=5230 ;*	Outputs : None. *
		=5231 ;*	Calls : ??? *
		=5232 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=5233 ;*	
		=5234 ;*	Description: *
		=5235 ;*	
		=5236 ;	=====
		=5237 ;	
		=5238 FILEIO:	
205B	32F6E8	=5239	STA COM ; Save file IO command.
205E	CD9D26	=5240	CALL BLINE ; Move cursor to the beginning of the line.
2061	CDA826	=5241	CALL CLRL ; Clear the line.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 93

LOC	OBJ	LINE	SOURCE STATEMENT
2064	CDEE20	=5242	CALL MAKECOM ; Make command (i.e. get file name from the
		=5243	; command string and place it in the
		=5244	; filename buffer.
2067	C2CB20	=5245	JNZ EXFIO ; If No Zero Return then Error.
		=5246	;
206A	CD5521	=5247	CALL SCTPDP ; Send the command to the PDP.
206D	C2B120	=5248	JNZ TIMOER ; If non zero return from Send_command then
		=5249	; Time_out_error.
		=5250	;
2070	CD7122	=5251	CALL WAITFA ; Wait for ACK char (Z = 1 => Ok).
		=5252	; Wait for the reply from the PDP
2073	C2A820	=5253	JNZ LNKERR ; IF return from WAITFA not zero then
		=5254	; there is a link error.
		=5255	;
2076	CD16F0	=5256	CALL GETPDP ; Get error code from the PDP.
2079	FE30	=5257	CPI '0' ; Zero ? (ASCII Zero => No error)
207B	CABA20	=5258	JZ NOFERR ; Yes - No File Error.
		=5259	;
207E	FE34	=5260	CPI '4' ; Error#4 = File does not exist.
2080	C28C20	=5261	JNZ NOTER4 ;
		=5262 ;	
		=5263 ;	
		=5264 ;	
		=5265 ;	If KIT error number 4 then print file not found message.
		=5266 ;	
2083	21C52E	=5267	LXI H,FDNOTE ;
2086	CDE125	=5268	CALL STRING ; Print error message.
2089	C3CB20	=5269	JMP EXFIO ; End exit
		=5270 ;	
		=5271 ;	
		=5272 ;	
		=5273 ;	If KIT error number 5 then print invalid file name message.
		=5274 ;	
		=5275	NOTER4: ;
208C	FE35	=5276	CPI '5' ;
208E	C29A20	=5277	JNZ NOTER5 ; If Error # = 5 Then
		=5278	;
2091	21DC2E	=5279	LXI H,INVNM ; Print (' Invalid file name ').
2094	CDE125	=5280	CALL STRING ;
2097	C3CB20	=5281	JMP EXFIO ; Exit (fileio).
		=5282 ;	
		=5283 ;	
		=5284 ;	
		=5285 ;	Else print general filer error message with number.
		=5286 ;	
		=5287	NOTER5: ;
209A	F5	=5288	PUSH PSW ; Save error number on stack.
209B	21AE2E	=5289	LXI H,PDPDM ;
209E	CDE125	=5290	CALL STRING ; Print general Error message + error number.
20A1	F1	=5291	POP PSW ;
20A2	CD9D24	=5292	CALL PRINT ; Print error number.
20A5	C3CB20	=5293	JMP EXFIO ; Exit file I/O section.
		=5294 ;	
		=5295 ;	
		=5296 ;	
		=5297 ;	If link_error then print link_error message.
		=5298 ;	
		=5299	LNKERR: ;
20A8	21422F	=5300	LXI H,LNKEM ;
20AB	CDE125	=5301	CALL STRING ; Print message
20AE	C3CB20	=5302	JMP EXFIO ; And exit.
		=5303 ;	
		=5304 ;	
		=5305 ;	
		=5306 ;	If there was a time_out_error then print Timeout message.
		=5307 ;	
		=5308	TIMOER: ;
20B1	21852F	=5309	LXI H,TIMOEM ;
20B4	CDE125	=5310	CALL STRING ; Print message
20B7	C3CB20	=5311	JMP EXFIO ; And exit.

LOC	OBJ	LINE	SOURCE STATEMENT
		=5312 ;	
		=5313 ;	
		=5314 ;	
		=5315 ;	If no file error and command = 'R' then load file from PDP.
		=5316 ;	else save file on PDP.
		=5317	NOFERR: ;
20BA	3AF6E8	=5318	LDA COM ;
20BD	FE52	=5319	CPI 'R' ;
20BF	C2C820	=5320	JNZ FIO99 ; If COM = 'R' Then
		=5321	;
20C2	CDCC20	=5322	CALL LOADF ; Load_file
20C5	C3CB20	=5323	JMP EXFIO ; Else
		=5324	FIO99: ;
20C8	CDCA21	=5325	CALL SAVEF ; Save_file
		=5326	EXFIO: ; End
20CB	C9	=5327	RET ; Return from the <FILEIO> routine.
		=5328 ;	
		=5329 ;	=====
		=5330 ;*	*
		=5331 ;*	Function : LOADF (Load file from the PDP using KIT) *
		=5332 ;*	Inputs : None. *
		=5333 ;*	Outputs : None. *
		=5334 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=5335 ;*	Calls : RFILE, ERRSTR, QINIT, KEYIN, WAITFU and BRROU *
		=5336 ;*	*
		=5337 ;*	Description : This routine loads a file from the PDP into *
		=5338 ;*	the user text area on the SABus kit. The routine uses the *
		=5339 ;*	RFILE routine to read the file and then prints the *
		=5340 ;*	correct message, an error message if the transfer fails *
		=5341 ;*	and a 'file loaded' message if there was no error. *
		=5342 ;*	*
		=5343 ;	=====
		=5344 ;	
		=5345	LOADF: ; Start of the <LOADF> routine.
20CC	CD8A21	=5346	CALL RFILE ; Recall file from the PDP.
		=5347	;
20CF	CAE120	=5348	JZ FLOK ; If Returned_status <> Ok then
		=5349	Begin
20D2	219B2F	=5350	LXI H,RFILMF ;
20D5	CD8723	=5351	CALL ERRSTR ; Writeln("Memory full")
20D8	CDEA12	=5352	CALL QINIT ; Kill_chars_in_input_queue.
20DB	CDBE23	=5353	CALL KEYIN ; wait_for_char_from_keyboard
20DE	C3EA20	=5354	JMP LFEXIT ;
		=5355	; else
		=5356	FLOK: ;
20E1	21F12E	=5357	LXI H,FLMESS ; Writeln("File loaded")
20E4	CD8723	=5358	CALL ERRSTR ;
20E7	CDA722	=5359	CALL WAITFU ; Wait for user to read message.
		=5360	LFEXIT: ;
20EA	CDD922	=5361	CALL BRROU ; Reset pointer to the beginning of range.
		=5362	;
20ED	C9	=5363	RET ; Return from <FILEIO> Module.
		=5364 ;	
		=5365 ;	=====
		=5366 ;*	*
		=5367 ;*	Function : Make Command (i.e. 'KIT Filename/<Switch>' *
		=5368 ;*	Inputs : BC Reg Pair = Ptr to 1st character after the *
		=5369 ;*	Command. eg 'LOAD MYFILE.A85' *
		=5370 ;*	(BC) = PTR *
		=5371 ;*	Outputs : The filename will have been stored in the file- *
		=5372 ;*	name buffer starting at <SOFN>, the filename *
		=5373 ;*	is terminated with a <EOT>. If there is an *
		=5374 ;*	error in the filename then this routines *
		=5375 ;*	Returns Non zero. *
		=5376 ;*	Destroys : A,Flags,D,E,H,L *
		=5377 ;*	Calls : CHKFN, STRING *
		=5378 ;*	*
		=5379 ;*	Description : This routine reads a file name from the command *
		=5380 ;*	line and stores it in the file name buffer. It also adds the *
		=5381 ;*	switch and an <EOT> to the filename. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 95

LOC	OBJ	LINE	SOURCE STATEMENT
		=5382 ;*	
		=5383 ;=====	
		=5384 ;	
		=5385 MAKECOM:	; Start of the make command routine
20EE 60		=5386 MOV H,B	
20EF 69		=5387 MOV L,C	; Get pointer to next character from BC.
		=5388	; and put it into HL.
		=5389 RUNB:	
20F0 7E		=5390 MOV A,M	
20F1 FE04		=5391 CPI EOT	; EOT in file name before any chars ?
20F3 CA2721		=5392 JZ EOTIFN	; Yes - the error.
		=5393	
20F6 FE20		=5394 CPI ' '	; SPACE ?
20F8 C2FF20		=5395 JNZ MAKEC5	; No - Then read the rest of the Filename.
		=5396	
20FB 23		=5397 INX H	
20FC C3F020		=5398 JMP RUNB	; Loop until Char # Space
		=5399	
		=5400 MAKEC5:	
20FF 0600		=5401 MVI B,0	; Start CRC
2101 11C9EF		=5402 LXI D,SOFN	
		=5403 CFNLOP:	
2104 7E		=5404 MOV A,M	
2105 FE04		=5405 CPI EOT	; Next character in command line = <EOT> ?
2107 CA1221		=5406 JZ EOFN	; Yes - then End of file name.
		=5407	; else
210A 12		=5408 STAX D	; Store character in output filename.
210B A8		=5409 XRA B	
210C 47		=5410 MOV B,A	; Update CRC
210D 23		=5411 INX H	; Inc file_name_buffer_ptr
210E 13		=5412 INX D	; Inc Command_line_ptr
210F C30421		=5413 JMP CFNLOP	; Loop until exited via <EOT>.
		=5414	
		=5415 ;	
		=5416 ;-----	
		=5417 ;	
		=5418 ; File name has been read, now add switch '//<Switch><EOT>'	
		=5419 ;	
		=5420 EOFN:	
2112 3E2F		=5421 MVI A, '/'	
2114 12		=5422 STAX D	; Place '/' after file name.
2115 13		=5423 INX D	
		=5424	
2116 3AF6E8		=5425 LDA COM	
2119 12		=5426 STAX D	; Add command after the '/'
211A 13		=5427 INX D	
		=5428	
211B 3E04		=5429 MVI A,EOT	
211D 12		=5430 STAX D	; Put the <EOT> after the switch.
211E 78		=5431 MOV A,B	
211F 32E7EF		=5432 STA FNCRC	; Save file name CRC.
2122 3E00		=5433 MVI A,0	; Set acc to zero to indicate Filename OK
2124 C35321		=5434 JMP MFNE	
		=5435 ;	
		=5436 ;-----	
		=5437 ;	
		=5438 ; Section of code that checks for an old file name if one is not	
		=5439 ; given in the SAVE/LOAD command line.	
		=5440 ;	
		=5441 EOTIFN:	; If EOT in filename then ERROR
2127 061E		=5442 MVI B,30	; Load max file name length
2129 0E00		=5443 MVI C,0	; Start CRC = 0
212B 21C9EF		=5444 LXI H,SOFN	; Pointer = Start of file name buffer
		=5445 CRCCL:	
212E 7E		=5446 MOV A,M	; Fetch file name character
212F FE2F		=5447 CPI '/'	; End of file name ?
2131 CA3E21		=5448 JZ CRCC05	
		=5449	; No - then update CRC with character
2134 A9		=5450 XRA C	
2135 4F		=5451 MOV C,A	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 96

LOC	OBJ	LINE	SOURCE STATEMENT
2136	23	=5452	INX H ; Inc filename pointer
2137	05	=5453	DCR B ; Dec character counter
2138	C22E21	=5454	JNZ CRCCL ; Loop until '/' found or 30 charaters
		=5455	; checked.
		=5456	;
213B	C34B21	=5457	JMP NOFILN ; If program reaches here then there is no
		=5458	; '/' in the filename buffer. (i.e. there
		=5459	; is no valid filename in the buffer)
		=5460	CRCC05: ;
213E	3AE7EF	=5461	LDA FNCRC ; Fetch stored CRC
2141	B9	=5462	CMP C ; Compare with calculated CRC
2142	C24B21	=5463	JNZ NOFILN ; If not equal then no filename error
		=5464	;
2145	CD3022	=5465	CALL CHKFN ;
2148	C35421	=5466	JMP MFNEX ; After returning from CHeck_File_Name
		=5467	; the flags will be set as follows.
		=5468	; Z = 1 : Old file name to be used.
		=5469	; Z = 0 : Ignore command.
		=5470	; i.e. Do not set flags before returning.
		=5471	NOFILN: ;
214B	210C2F	=5472	LXI H,NFNAMM ; If there is no file name in the buffer the
214E	CDE125	=5473	CALL STRING ; print the 'No file name' message.
2151	3EFF	=5474	MVI A,OFFH ; and reset zero flag => error.
		=5475	;
		=5476	MFNE: ; Make File Name Exit.
2153	B7	=5477	ORA A ;
		=5478	MFNEX: ;
2154	C9	=5479	RET ; Return from <MAKECOM>
		=5480	;
		=5481	=====
		=5482	;
		=5483	;* Function : SCTPDP (Send Command To PDP) *
		=5484	;* Inputs : A file name stored in memory starting at <SOFN> *
		=5485	;* and terminated by an <EOT> *
		=5486	;* Outputs : None *
		=5487	;* Destroys : A and Flags *
		=5488	;* Calls : DELAY, GETPDP and SNDPDP. *
		=5489	;* *
		=5490	;* Description : This routine sends the KIT command and the filename in *
		=5491	;* memory to the PDP. After sending each character it waits for the *
		=5492	;* character to be echoed. after the command and the file name have *
		=5493	;* been sent it sends a <CR> to terminate the command. *
		=5494	;* *
		=5495	=====
		=5496	;
		=5497	SCTPDP: ; Send command line to the PDP.
		=5498	; Procedure Send_command_to_pdp ;
		=5499	; Begin
2155	21A92E	=5500	LXI H,PDP COM ; Pointer := 1 ; (* start of command *)
		=5501	SCOML1: ;
2158	7E	=5502	MOV A,M ;
2159	FE04	=5503	CPI EOT ;
215B	CA6B21	=5504	JZ EXPDP1 ; While Command_message(Pointer) # EOT do
		=5505	; Begin
215E	CD2222	=5506	CALL DELAY ; Delay;
2161	CD8722	=5507	CALL SPDPWE ; Send_with_echo(Command_message(Pointer));
2164	C28921	=5508	JNZ SCOMEX ; If Timeout then EXIT(SCTPDP);
		=5509	;
2167	23	=5510	INX H ; Pointer := Pointer + 1
2168	C35821	=5511	JMP SCOML1 ; End;
		=5512	EXPDP1: ;
216B	21C9EF	=5513	LXI H,SOFN ; Pointer := 1 ; (* start of file name *)
		=5514	SCOML2: ;
216E	7E	=5515	MOV A,M ;
216F	FE04	=5516	CPI EOT ;
2171	CA8121	=5517	JZ EXPDP2 ; While File_name(Pointer) # EOT Do
		=5518	; Begin
2174	CD2222	=5519	CALL DELAY ; Delay;
2177	CD8722	=5520	CALL SPDPWE ; Send_with_echo(File_name(Pointer));
217A	C28921	=5521	JNZ SCOMEX ; If timeout then EXIT(SCTPDP);

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 97

LOC	OBJ	LINE	SOURCE STATEMENT
		=5522	
217D	23	=5523	INX H ; Pointer := Pointer + 1
217E	C36E21	=5524	JMP SCOML2 ; End;
		=5525	EXPDP2: ;
2181	CD2222	=5526	CALL DELAY ; Delay; (* Delay before sending the <CR> *)
2184	3E0D	=5527	MVI A,CR ;
2186	CD8722	=5528	CALL SPDPWE ; Send_with_echo(Carriage_return);
		=5529	;
		=5530	SCOMEX: ; Exit point for Send_command_to_pdp.
		=5531	;
2189	C9	=5532	RET ; End; (* Return from <SCTPDP>. *)
		=5533	;
		=5534	=====
		=5535	;
		=5536	Function : RFILE (Read file from PDP/11) *
		=5537	Inputs : None. *
		=5538	Outputs : None. *
		=5539	Destroys : All. *
		=5540	Calls : GETPDP and PRINT. *
		=5541	;
		=5542	Description : This routine reads a file from the pdp into memory on *
		=5543	the kit. It is assumed that the KIT command has been sent and that *
		=5544	the KIT ok chars have been received from the PDP. Note <CR>'s are *
		=5545	discarded as the PDP sends a <CR> and a <LF> after every line and *
		=5546	the kit's editor only needs line feeds between the lines. The end *
		=5547	of the file is marker by a <EOFMRK>. *
		=5548	;
		=5549	=====
		=5550	;
		=5551	RFILE: ; Procedure read_file ;
		=5552	; Begin
218A	2AFOE8	=5553	LHLD MAXTXT ;
218D	EB	=5554	XCHG ; DE := Max_text_pointer ;
218E	2AFCEF	=5555	LHLD SPOINT ; Pointer := Start_of_text ;
		=5556	RFILE05: ;
2191	CD16F0	=5557	CALL GETPDP ; Acc := Getpdp ;
2194	FE1A	=5558	CPI EOFMARK ;
2196	CAC221	=5559	JZ RFIL20 ; While Acc <> End_of_file_marker do
		=5560	Begin
2199	FE0D	=5561	CPI CR ; If char <> Carriage_Return then
219B	CA9121	=5562	JZ RFIL05 ; Begin
		=5563	;
219E	77	=5564	MOV M,A ; Text(pointer) := Acc ;
219F	FE0A	=5565	CPI LF ; IF char = Line_feed then
21A1	C2AD21	=5566	JNZ RFIL10 ; Begin
		=5567	;
21A4	CD43F0	=5568	CALL DCMF ; If Pointer >= Max_ptr then memory
		=5569	full error.
21A7	D2BD21	=5570	JNC RFIL15 ; end;
		=5571	;
21AA	22FEEF	=5572	SHLD EPOINT ; End_of_text := pointer;
		=5573	RFILE10: ;
21AD	23	=5574	INX H ; Pointer := Pointer + 1;
		=5575	;
21AE	7C	=5576	MOV A,H ;
21AF	E601	=5577	ANI 00000001B ;
21B1	B5	=5578	ORA L ;
21B2	C29121	=5579	JNZ RFIL05 ; If Pointer MOD 512 = 0 then
		=5580	;
21B5	3E2B	=5581	MVI A,'+' ;
21B7	CD9D24	=5582	CALL PRINT ; Write(' +')
		=5583	;
		=5584	End (* End if *)
21BA	C39121	=5585	JMP RFIL05 ; End; (* End while *)
		=5586	;
		=5587	RFILE15: ; (* Memory full error jump *)
21BD	3EFF	=5588	MVI A,ERR ; Returned_status = error.
21BF	C3C821	=5589	JMP RFIL25 ; Exit the read file routine.
		=5590	RFILE20: ;
21C2	2B	=5591	DCX H ; Pointer := Pointer - 1 ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 98

LOC	OBJ	LINE	SOURCE STATEMENT
21C3	22FEEF	=5592	SHLD EPOINT ; End_pointer := Pointer
21C6	3E00	=5593	MVI A,OK ; Returned status = Ok ;
		=5594	RFIL25: ;
21C8	B7	=5595	ORA A ; Set flags before returning.
21C9	C9	=5596	RET ; End; (* End procedure RFILE *)
		=5597 ;	
		=5598 ;=====	
		=5599 ;*	
		=5600 ;*	Function : SAVEF (Save file on PDP/11) *
		=5601 ;*	Inputs : Start_pointer and End_pointer (Memory) *
		=5602 ;*	Outputs : None. *
		=5603 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=5604 ;*	Calls : SNDDPDP, PRINT, FDCMP and GETPDP. *
		=5605 ;*	
		=5606 ;*	Description : This routine saves the work area as the file name *
		=5607 ;*	found in the file name area 'FNAME'. The file is assumed to be open *
		=5608 ;*	and an '@' read (this indicates no filename error). The transfer *
		=5609 ;*	uses the PDP KIT program, for more details see KIT.RTL source file *
		=5610 ;*	or documentation on 'KIT'. *
		=5611 ;*	
		=5612 ;=====	
		=5613 ;	
		=5614	SAVEF: ; Start of SAVE_File
21CA	2AFCEF	=5615	LHLD SPOINT ;
21CD	EB	=5616	XCHG ;
21CE	2AFEEF	=5617	LHLD EPOINT ;
21D1	EB	=5618	XCHG ; Start pointer = HL register pair
		=5619	; End pointer = DE register pair .
		=5620 ;	
		=5621 ;-----	
		=5622 ;	
		=5623 ;	Wait for Ack char from 'KIT'.
		=5624 ;	
		=5625	NXTLIN: ; Repeat
21D2	CD16F0	=5626	CALL GETPDP ; Char := Getpdp;
21D5	FE40	=5627	CPI ACK ;
21D7	C2D221	=5628	JNZ NXTLIN ; Until Char = ACK
		=5629	;
		=5630	SAVELP: ;
21DA	7C	=5631	MOV A,H ; Load acc with the MSByte of the memory pointer.
21DB	E601	=5632	ANI 00000001B ; Mask off LSBit.
21DD	B5	=5633	ORA L ; Set ZF if pointer = xxxx xxx0 0000 0000
21DE	C2E621	=5634	JNZ NOPLUS ;
		=5635	; If Pointer MOD 512 = 0 then
21E1	3E2B	=5636	MVI A,'+' ;
21E3	CD9D24	=5637	CALL PRINT ; Write ('+'); (* Print plus every block *)
		=5638	;
		=5639	NOPLUS: ;
21E6	CD3012	=5640	CALL FDCMP ; Compare pointer with end_pointer.
21E9	CA0B22	=5641	JZ ENDSAVE ; Exit if equal.
		=5642	;
21EC	7E	=5643	MOV A,M ;
21ED	FE0A	=5644	CPI LF ; If Text(pointer) <> Line_feed then
21EF	CA0222	=5645	JZ ANLINE ; Begin
		=5646	;
21F2	CDB322	=5647	CALL QCHAROK ; IF Character_OK then
21F5	CAFE21	=5648	JZ FSAV05 ; Begin
		=5649	;
21F8	CD1FF0	=5650	CALL SNDDPDP ; Pdpout(Text(pointer));
21FB	CD16F0	=5651	CALL GETPDP ; Acc := Pdpin;
		=5652	FSAV05: ; End
21FE	23	=5653	INX H ; Pointer := Pointer + 1;
21FF	C3DA21	=5654	JMP SAVELP ; End;
		=5655	;
		=5656	ANLINE: ; Else
2202	3E0D	=5657	MVI A,CR ; Begin
2204	CD1FF0	=5658	CALL SNDDPDP ; Pdpout(Carriage_return);
2207	23	=5659	INX H ; Pointer := Pointer + 1;
2208	C3D221	=5660	JMP NXTLIN ; End;
		=5661 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 99

LOC	OBJ	LINE	SOURCE STATEMENT
		=5662 ;	-----
		=5663 ;	
		=5664 ;	Save file exit pointer (Pointer = End_pointer)
		=5665 ;	
		=5666	ENDSAVE: ;
220B	3E0D	=5667	MVI A,CR ;
220D	CD1FF0	=5668	CALL SNDDPDP ; Pdpout(Carriage_return);
		=5669	;
2210	CD7122	=5670	CALL WAITFA ; Wait for Ack line sent.
		=5671	;
2213	3E1A	=5672	MVI A,EOFMARK ; Pdpout(End_of_file_marker);
2215	CD1FF0	=5673	CALL SNDDPDP ;
2218	CD16F0	=5674	CALL GETPDP ; Wait for the echo.
		=5675	;
221B	21FF2E	=5676	LXI H,FSMESS ;
221E	CD8723	=5677	CALL ERRSTR ; Print "File saved" message.
		=5678	;
2221	C9	=5679	RET ; Return from <SAVEF>
		=5680 ;	
		=5681 ;	=====
		=5682 ;*	
		=5683 ;*	Function : DELAY *
		=5684 ;*	Inputs : None. *
		=5685 ;*	Outputs : None. *
		=5686 ;*	Destroys : None. *
		=5687 ;*	Calls : None. *
		=5688 ;*	
		=5689 ;*	Description : This is a simple delay routine used when sending *
		=5690 ;*	characaters to the PDP while it's in MCR mode. *
		=5691 ;*	
		=5692 ;	=====
		=5693 ;	
		=5694	DELAY: ; Start of Delay.
2222	E5	=5695	PUSH H ;
2223	F5	=5696	PUSH PSW ; Save all registers that could get damaged.
		=5697	;
2224	210010	=5698	LXI H,1000H ; Get delay const.
		=5699	DELOOP: ;
2227	2B	=5700	DCX H ; Dec delay counter.
2228	7C	=5701	MOV A,H ;
2229	B5	=5702	ORA L ; Delay counter = Zero ?.
222A	C22722	=5703	JNZ DELOOP ; No - Then loop until it is.
		=5704	;
222D	F1	=5705	POP PSW ; Restore all registers.
222E	E1	=5706	POP H ;
		=5707	;
222F	C9	=5708	RET ; Return from Delay.
		=5709 ;	
		=5710 ;	=====
		=5711 ;*	
		=5712 ;*	Function : CHKFN (Check file name). *
		=5713 ;*	Inputs : None. *
		=5714 ;*	Outputs : Z = 1 : Users wants the old file name used. *
		=5715 ;*	Z = 0 : Users does not want the old file name. *
		=5716 ;*	Destroys : H, L, A and flags. *
		=5717 ;*	Calls : BLINE, CONCAP, CLRL, STRING and KEYIN. *
		=5718 ;*	
		=5719 ;*	Description : This routine check with the user to see if he wants to *
		=5720 ;*	use the old file name in the SAVE/LOAD command that he has just *
		=5721 ;*	typed without a filename. The routine searches for the '/' and *
		=5722 ;*	replaces it with an EOT in order to print it, but this is corrected *
		=5723 ;*	after the file name has been printed. The answer is a single *
		=5724 ;*	key_press and therefore cannot be edited by the user. *
		=5725 ;*	
		=5726 ;	=====
		=5727 ;	
		=5728	CHKFN: ; Start of Check_file_name.
2230	CD9D26	=5729	CALL BLINE ;
2233	CDA826	=5730	CALL CLRL ; Clear command line.
2236	211F2F	=5731	LXI H,QFILN1 ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 100

LOC	OBJ	LINE	SOURCE STATEMENT	
2239	CDE125	=5732	CALL	STRING ; Print 1st part of ? file name message.
223C	21C9EF	=5733	LXI	H,SOFN ; Pointer := Start_of_file_name
		=5734		;
		=5735	CHKFN2:	; Repeat
223F	7E	=5736	MOV	A,M ; Acc := File_name(pointer);
2240	FE2F	=5737	CPI	'/' ;
2242	CA4922	=5738	JZ	BACKSF ;
		=5739		;
2245	23	=5740	INX	H ; Pointer := Pointer + 1
2246	C33F22	=5741	JMP	CHKFN2 ; Until Acc = '/'
		=5742		;
		=5743	BACKSF:	;
2249	3604	=5744	MVI	M,EOT ; File_name(Pointer) := EOT ;
		=5745		;
224B	E5	=5746	PUSH	H ; Save pointer to EOT.
		=5747		;
224C	21C9EF	=5748	LXI	H,SOFN ;
224F	CDE125	=5749	CALL	STRING ; Print File name.
		=5750		;
2252	21262F	=5751	LXI	H,QFILN2 ;
2255	CDE125	=5752	CALL	STRING ; Print 2nd part of message.
		=5753		;
2258	E1	=5754	POP	H ; Restore pointer to the Back slash.
		=5755		;
2259	362F	=5756	MVI	M,'/' ; Restore '/' in KIT command line.
225B	23	=5757	INX	H ;
225C	3AF6E8	=5758	LDA	COM ; Update the Kit switch with the New command.
225F	77	=5759	MOV	M,A ;
2260	CDBE23	=5760	CALL	KEYIN ; Get users answer to the Question.
2263	CDB811	=5761	CALL	CONCAP ; Convert to upper case.
2266	FE59	=5762	CPI	'Y' ; Check to see if answer = Yes.
		=5763		;
2268	F5	=5764	PUSH	PSW ; Save Flags after compare.
2269	CD9D26	=5765	CALL	BLINE ;
226C	CDA826	=5766	CALL	CLRL ; Clear command line.
226F	F1	=5767	POP	PSW ; Restore flags.
		=5768		;
2270	C9	=5769	RET	; Return from <CHKFN>
		=5770		;
		=5771		=====
		=5772		;
		=5773		Function : WAITFA (Wait for ACK) *
		=5774		Inputs : None. *
		=5775		Outputs : Z = 1 : Ack received ok (No invalid characters). *
		=5776		Z = 0 : Ack error (Invalid characters received before *
		=5777		ack character) *
		=5778		Destroys : A and flags. *
		=5779		Calls : GETPDP. *
		=5780		;
		=5781		Description : This routine is used to check incoming characters from *
		=5782		the PDP and wait for the ACK char. The routine discards all control *
		=5783		characters that it receives while waiting for the ACK char, but if *
		=5784		a Non control character is received before an ACK then the routine *
		=5785		returns an error (Z=0). *
		=5786		;
		=5787		=====
		=5788		;
		=5789	WAITFA:	; Start of Wait for ACK routine.
2271	CD16F0	=5790	CALL	GETPDP ; Get char from PDP.
2274	FE40	=5791	CPI	ACK ; Char = ACK ?
2276	CA8322	=5792	JZ	WFAOK ; Yes - Then exit with status OK.
		=5793		;
2279	FE20	=5794	CPI	SPACE ; Char >= Space ?
227B	FA7122	=5795	JM	WAITFA ; No - Loop and wait for ACK char.
		=5796		;
227E	3EFF	=5797	MVI	A,OFFH ; If not ACK and Char >= Space then Link_error.
2280	C38522	=5798	JMP	WFARET ; Exit Wait for ACK routine.
		=5799		;
		=5800	WFAOK:	; Wait for Ack OK.
2283	3E00	=5801	MVI	A,000H ; Set zero flag (Status OK).

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 101

LOC	OBJ	LINE	SOURCE STATEMENT
		=5802	;
		=5803	WFARET: ; Wait for Ack exit point.
2285	B7	=5804	ORA A ; Set flags before returning.
2286	C9	=5805	RET ; Return from <WAITFA>
		=5806	;
		=5807	;=====
		=5808	;* *
		=5809	;* Function : SPDPWE (Send to PDP and wait for the echo with timeout) *
		=5810	;* Inputs : Aregister = Character to send. *
		=5811	;* Outputs : Z = 1 : Echo received OK. (No timeout) *
		=5812	;* Z = 0 : Timeout on echo. *
		=5813	;* Destroys : A and Flags. *
		=5814	;* Calls : SNDPDP, UOTEST and GETPDP. *
		=5815	;* *
		=5816	;* Description : This routine is used to send characters to the PDP and *
		=5817	;* check that the PDP asnwrs within a given time. (Used to check for *
		=5818	;* line_down errors). *
		=5819	;* *
		=5820	;=====
		=5821	;
		=5822	SPDPWE: ; Start of Send_to_pdp_with_echo.
2287	CD1FF0	=5823	CALL SNDPDP ; Send char to PDP.
228A	E5	=5824	PUSH H ;
228B	21FFFF	=5825	LXI H,TOCONST ; Get timeout constant.
		=5826	;
		=5827	WFECHO: ; Wait for echo loop
228E	CD0DF0	=5828	CALL UOTEST ; Char in USART #1 (PDP USART ?)
2291	CA9C22	=5829	JZ NOECHO ; No - Then dec counter and try again.
		=5830	;
2294	CD16F0	=5831	CALL GETPDP ; Fetch char from the USART.
2297	3E00	=5832	MVI A,000H ; Then return with Zero flag set.
2299	C3A422	=5833	JMP SWERET ; Exit Send_with_echo.
		=5834	;
		=5835	NOECHO: ; No Echo.
229C	2B	=5836	DCX H ; Dec timeout counter.
229D	7C	=5837	MOV A,H ;
229E	B5	=5838	ORA L ; HL = Zero ?
229F	C28E22	=5839	JNZ WFECHO ; No - Then there is still time for the
		=5840	echo to be received.
22A2	3EFF	=5841	MVI A,OFFH ; Reset zero flag => Time_out.
		=5842	;
		=5843	SWERET: ;
22A4	B7	=5844	ORA A ; Set flags before returning.
22A5	E1	=5845	POP H ;
22A6	C9	=5846	RET ; Return from <SPDPWE>
		=5847	;
		=5848	;=====
		=5849	;* *
		=5850	;* Function : WAITFU (wait for user) *
		=5851	;* Inputs : None *
		=5852	;* Outputs : None *
		=5853	;* Destroys : Acc and flags *
		=5854	;* Calls : None *
		=5855	;* *
		=5856	;* Description : This routine simply waits for about a second *
		=5857	;* to allow the user to read the message. *
		=5858	;* *
		=5859	;=====
		=5860	;
		=5861	WAITFU: ; Start of <WAITFU> routine
22A7	E5	=5862	PUSH H ; Save HL on stack
22A8	213075	=5863	LXI H,30000 ; HL = Delay constant
		=5864	;
		=5865	WFOU5: ; Repeat
22AB	2B	=5866	DCX H ; HL:=HL-1
22AC	7C	=5867	MOV A,H ;
22AD	B5	=5868	ORA L ;
22AE	C2AB22	=5869	JNZ WFOU5 ; Until HL=0
		=5870	;
22B1	E1	=5871	POP H ; Restore Hl from the stack

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 102

LOC	OBJ	LINE	SOURCE STATEMENT
22B2	C9	=5872	RET ; Return from <WAITFU> routine
		=5873 ;	
		=5874 ;=====	
		=5875 ;*	
		=5876 ;*	Function : QCHAROK (Character OK ?)
		=5877 ;*	Inputs : Acc = Character to check.
		=5878 ;*	Outputs : Z = 0 : If the character is OK.
		=5879 ;*	Z = 1 : If the character is not OK and must
		=5880 ;*	therefore not be sent to the PDP.
		=5881 ;*	Acc : The character with its sign bit clear
		=5882 ;*	Destroys : Flags.
		=5883 ;*	Calls : None.
		=5884 ;*	
		=5885 ;*	Description : This routine is intended to check that a
		=5886 ;*	character in the file being saved will not cause any
		=5887 ;*	problems on the PDP side. A list of characters that can
		=5888 ;*	trouble the PDP are given below:
		=5889 ;*	
		=5890 ;*	<^C> 03H <^S> 13H <ESC> 1BH <^U> 15H
		=5891 ;*	<^R> 12H <^Z> 1AH <DEL> 7FH
		=5892 ;*	
		=5893 ;*	
		=5894 ;=====	
		=5895 ;	
		=5896	QCHAROK: ; Start of <QCHAROK> routine
22B3	E67F	=5897	ANI ASCII ; Clear sign bit
22B5	FE03	=5898	CPI CTRLC ;
22B7	C0	=5899	RNZ ; Return zero if character = CNTRLC
22B8	FE13	=5900	CPI CTRLS ;
22BA	C0	=5901	RNZ ; Return zero if character = XOFF (^S)
22BB	FE1B	=5902	CPI ESC ;
22BD	C0	=5903	RNZ ; Return zero if character = ESC
22BE	FE15	=5904	CPI CTRLU ;
22C0	C0	=5905	RNZ ; Return zero if character = CNTRLU
22C1	FE12	=5906	CPI CTRLR ;
22C3	C0	=5907	RNZ ; Return zero if character = CNTRLR
22C4	FE1A	=5908	CPI CTRLZ ;
22C6	C0	=5909	RNZ ; Return zero if character = CNTRLZ
22C7	FE7F	=5910	CPI DEL ;
22C9	C9	=5911	RET ; Return zero if character = DEL
		=5912 ;	
		=5913 ;	End of file I/O module for the kit editor. (EDTFILEIO.A85)
		5914	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 103

LOC	OBJ	LINE	SOURCE STATEMENT
		5915 ;	
		5916 ;	Include BR/ER/BL/EL Routines.
		5917 ;	
		5918	\$INCLUDE(EDTRANGE.A85)
		=5919 ;	-----;
		=5920 ;	
		=5921 ;	Range module for the UCT SABus editor.
		=5922 ;	-----;
		=5923 ;	
		=5924 ;	This module contains (in order) the following subroutines:
		=5925 ;	
		=5926 ;	a) BLROU - Beginning of line routine.
		=5927 ;	b) BRROU - Beginning of range routine.
		=5928 ;	c) ELROU - End of line routine.
		=5929 ;	d) ERROU - End of range routine.
		=5930 ;	e) QNULLF - Null file ?
		=5931 ;	
		=5932 ;	-----;
		=5933 ;	
		=5934	SEJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=5935 ;	=====
		=5936 ;*	
		=5937 ;*	Function : BLROU (Beginning of line routine) *
		=5938 ;*	Inputs : None. *
		=5939 ;*	Outputs : None. *
		=5940 ;*	Destroys : H,L,Acc and flags. *
		=5941 ;*	Calls : VERTP. *
		=5942 ;*	
		=5943 ;*	Description : This routine moves the cursor to the start of *
		=5944 ;*	the current line. *
		=5945 ;*	
		=5946 ;	=====
		=5947 ;	
		=5948 BLROU:	; Beginning_of_line_routine.
22CA	3E00	=5949 MVI A,0	
22CC	320AE8	=5950 STA RELCH	; Relative char position = 0.
		=5951 ;	
22CF	2A00E8	=5952 LHLD POINT	; Get start of line pointer.
22D2	CDBD14	=5953 CALL VERTP	
22D5	2202E8	=5954 SHLD CHAR	; Save as character pointer.
		=5955 ;	
22D8	C9	=5956 RET	; Return from <BLROU>
		=5957 ;	
		=5958 ;	=====
		=5959 ;*	
		=5960 ;*	Function : BRROU (beginning of range routine) *
		=5961 ;*	Inputs : None. *
		=5962 ;*	Outputs : None. *
		=5963 ;*	Destroys : All. *
		=5964 ;*	Calls : HOME and PLINE. *
		=5965 ;*	
		=5966 ;*	Description : This routine is called to move the cursor to *
		=5967 ;*	the beginning of the file. The terminal cursor is homed *
		=5968 ;*	and the top line printed, the screen updating will be done *
		=5969 ;*	automatically by the UPDATE routine when control returns *
		=5970 ;*	to the main editor loop. All the pointers (except EPOINT) *
		=5971 ;*	are set equal to the start of the file and the relative *
		=5972 ;*	character position is set to zero. *
		=5973 ;*	
		=5974 ;	=====
		=5975 ;	
		=5976 BRROU:	; Beginning_of_range_routine.
		=5977 ;	
22D9	CDCA26	=5978 CALL HOME	; Home cursor
		=5979 ;	
22DC	3E00	=5980 MVI A,0	
22DE	320AE8	=5981 STA RELCH	; Relative character position = 0.
22E1	320EE8	=5982 STA LCLINE	; Lowest correct line pointer = 0.
		=5983 ;	
22E4	2AFCEF	=5984 LHLD SPOINT	; Get Text start pointer.
		=5985 BRROU5:	
22E7	7E	=5986 MOV A,M	; Repeat
22E8	FE00	=5987 CPI NULL	
22EA	23	=5988 INX H	
22EB	CAE722	=5989 JZ BRROU5	; Until char # null
22EE	2B	=5990 DCX H	
		=5991 ;	
22EF	2202E8	=5992 SHLD CHAR	; Char_pointer = Start_of_text_pointer.
22F2	2200E8	=5993 SHLD POINT	; Start_of_line_pointer = Start_of_text_pointer.
		=5994 ;	
22F5	CD7224	=5995 CALL PLINE	; Print the top line.
		=5996 ;	
22F8	3E00	=5997 MVI A,FALSE	
22FA	3211E8	=5998 STA RESFLG	; Reset Restore cursor flag.
		=5999 ;	
22FD	3E17	=6000 MVI A,LSTLIN+1	
22FF	3208E8	=6001 STA EOBLIN	; Init EOB line pointer off the screen, this
		=6002 ;	is makes sure that the UPDATE routine
		=6003 ;	rewrites the screen.
		=6004 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 105

LOC	OBJ	LINE	SOURCE STATEMENT
2302	C9	=6005	RET ; Return from <BRROU>.
		=6006	;
		=6007	=====
		=6008	;
		=6009	Function : ELROU (end of line routine)
		=6010	Inputs : None.
		=6011	Outputs : None.
		=6012	Destroys : All.
		=6013	Calls : QNULLF, RULF and QCHAR.
		=6014	;
		=6015	Description : This routine movee the cursor to the end of
		=6016	the current line.
		=6017	;
		=6018	=====
		=6019	;
		=6020	ELROU: ; Start of ELROU.
		=6021	;
2303	CD7723	=6022	CALL QNULLF ;
2306	CA1323	=6023	JZ ELROUE ; IF Not Null_file then
		=6024	begin
2309	CD8A13	=6025	CALL RULF ; Read_until_line_feed;
230C	2B	=6026	DCX H ; Pointer := pointer - 1;
230D	2202E8	=6027	SHLD CHAR ; Char_tpr := pointer;
2310	CD9914	=6028	CALL QCHAR ; Set_new_relative_char_position
		=6029	End;
		=6030	ELROUE: ;
2313	C9	=6031	RET ;
		=6032	;
		=6033	=====
		=6034	;
		=6035	Function : ERROU (End of range routine)
		=6036	Inputs : None.
		=6037	Outputs : None.
		=6038	Destroys : All.
		=6039	Calls : HOME, QNULLF, RBULF, FDCMP, NEWSCH, UPDATE,
		=6040	QCHAR and PLINE.
		=6041	;
		=6042	Description : This routine moves the cursor to the end of
		=6043	the file. The previous 20 lines are displayed above the
		=6044	cursor.
		=6045	;
		=6046	=====
		=6047	;
		=6048	ERROU: ; Start of End_of_range_routine.
		=6049	;
2314	CDCA26	=6050	CALL HOME ;
		=6051	;
2317	3E00	=6052	MVI A,00H ;
2319	320EE8	=6053	STA LCLINE ; Lowest_correct_line = 0.
		=6054	;
231C	3E00	=6055	MVI A,FALSE ;
231E	3211E8	=6056	STA RESFLG ; Restore_cursor_flag = False.
		=6057	;
2321	2AFEEF	=6058	LHLD EPOINT ;
2324	2202E8	=6059	SHLD CHAR ; Char_pointer = End_pointer.
		=6060	;
2327	CD7723	=6061	CALL QNULLF ; Check for null file.
232A	CA6B23	=6062	JZ NULLF ; IF Start_pointer <> End_pointer
		=6063	Begin
232D	23	=6064	INX H ; Pointer := pointer + 1;
232E	2200E8	=6065	SHLD POINT ; Start_of_line_pointer := pointer;
2331	0E00	=6066	MVI C,0 ;
2333	0614	=6067	MVI B,20 ; Number of lines to skip back
		=6068	BACKL: ;
2335	CD6913	=6069	CALL RBULF ; Read_backwards_until_line_feed;
2338	0C	=6070	INR C ;
2339	2200E8	=6071	SHLD POINT ; Start_of_line_pointer := pointer;
233C	EB	=6072	XCHG ;
233D	2AFCEF	=6073	LHLD SPOINT ;
2340	EB	=6074	XCHG ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 106

LOC	OBJ	LINE	SOURCE STATEMENT
2341	CD3012	=6075	CALL FDCMP ;
2344	CA4B23	=6076	JZ FBEGIN ; At the file beginning so exit the loop
		=6077	;
2347	05	=6078	DCR B ; Dec the count
2348	C23523	=6079	JNZ BACKL ; Keep going back till all done
		=6080	FBEGIN: ;
234B	79	=6081	MOV A,C ;
234C	3D	=6082	DCR A ;
234D	F5	=6083	PUSH PSW ; Save line
234E	CD7F12	=6084	CALL NEWSCHN ; Update the screen
2351	CDD513	=6085	CALL UPDATE ; Update the rest of the screen
2354	F1	=6086	POP PSW ;
2355	3205E8	=6087	STA LINE ;
2358	2AFEEF	=6088	LHLD EPOINT ;
235B	2202E8	=6089	SHLD CHAR ;
235E	23	=6090	INX H ;
235F	2200E8	=6091	SHLD POINT ;
2362	CD6913	=6092	CALL RBULF ; Find the start of the last line
2365	2200E8	=6093	SHLD POINT ;
2368	C36E23	=6094	JMP ER05 ;
		=6095	;
		=6096	NULLF: ; End
		=6097	;
236B	CD7224	=6098	CALL PLINE ; Print_current line.
		=6099	;
		=6100	ER05: ;
236E	3E17	=6101	MVI A,LSTLIN+1 ;
2370	3208E8	=6102	STA EOBLIN ; Init EOB line pointer off the screen.
2373	CD9914	=6103	CALL QCHAR ; Set the relative cursor position.
		=6104	;
2376	C9	=6105	RET ; Return from <ERROU>
		=6106	;
		=6107	=====
		=6108	;* *
		=6109	;* Function : ? Null file. *
		=6110	;* Inputs : None. *
		=6111	;* Outputs : Z = 1 : If Start_ptr = End_ptr. *
		=6112	;* Destroys : Acc and flags. *
		=6113	;* Calls : FDCMP *
		=6114	;* *
		=6115	;* Description: This routine is used to check for a null file *
		=6116	;* i.e. a file with the start and end pointers equal. If the *
		=6117	;* file is a null file then the Zero flag is set. *
		=6118	;* *
		=6119	=====
		=6120	;
		=6121	QNULLF: ; Start of ? null file routine.
2377	E5	=6122	PUSH H ;
2378	D5	=6123	PUSH D ; Save DE,HL regs
		=6124	;
2379	2AFEEF	=6125	LHLD EPOINT ;
237C	EB	=6126	XCHG ;
237D	2AFCEF	=6127	LHLD SPOINT ;
2380	EB	=6128	XCHG ;
		=6129	; Pointer: HL = End_pointer.
		=6130	; DE = Start_pointer.
		=6131	;
2381	CD3012	=6132	CALL FDCMP ; Compare HL with DE.
		=6133	;
2384	D1	=6134	POP D ;
2385	E1	=6135	POP H ; Restore DE,HL regs.
		=6136	;
2386	C9	=6137	RET ; Return from <QNULLF>
		=6138	;
		=6139	End of EDT range routines.
		6140	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 107

LOC	OBJ	LINE	SOURCE STATEMENT
		6141 ;	
		6142 ;	Include I/O Modules
		6143 ;	
		6144	\$INCLUDE(EDTIO.A85)
		=6145 ;	-----;
		=6146 ;	
		=6147 ;	I/O module for the UCT SABus editor.
		=6148 ;	-----;
		=6149 ;	
		=6150 ;	This module contains (in order) the following subroutines:
		=6151 ;	
		=6152 ;	
		=6153 ;	1) ERRSTR ; Error message output routine.
		=6154 ;	2) GETKEY ; Character input routine (Valid character in text) ;
		=6155 ;	3) INCPTR ; Increment input queue pointer ;
		=6156 ;	4) KEYIN ; Character input routine. ;
		=6157 ;	5) KEYSTK ; Keyboard check routine (Stacks incoming chars). ;
		=6158 ;	6) LPRINT ; Letter print routine (prints control chars as ^x). ;
		=6159 ;	7) PBELL ; Bell output routine ;
		=6160 ;	8) PLINE ; Print line. ;
		=6161 ;	9) PRINT ; Character output routine. ;
		=6162 ;	10) PTOEOL ; Print to end of line. ;
		=6163 ;	11) READS ; Read string from keyboard into input buffer. ;
		=6164 ;	12) RESCUR ; Restore cursor position ;
		=6165 ;	13) SCPRESS ; Space compress command line. ;
		=6166 ;	14) STACKC ; Stack input character routine (used by KEYSTK) ;
		=6167 ;	15) STRING ; Message output routine ;
		=6168 ;	
		=6169 ;	-----;
		=6170 ;	
		=6171	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 108

LOC	OBJ	LINE	SOURCE STATEMENT
		=6172 ;	=====
		=6173 ;*	*
		=6174 ;*	Function : ERRSTR (Prints error message on console). *
		=6175 ;*	Inputs : HL : Start address of the error message. *
		=6176 ;*	Outputs : None *
		=6177 ;*	Destroys : H,L,Acc and flags. *
		=6178 ;*	Calls : STRING *
		=6179 ;*	*
		=6180 ;*	Description: This routine is used to output an error *
		=6181 ;*	message, the start address of the message is passed in *
		=6182 ;*	the HL register pair. The routine first uses direct *
		=6183 ;*	cursor positioning to place the cursor on the last line *
		=6184 ;*	of the screen and then outputs the message. *
		=6185 ;*	*
		=6186 ;	=====
		=6187 ;	
		=6188 ERRSTR:	
		=6189	PUSH H ; Start of error string routine
2387	E5	=6190	LXI H,ERRCCS ; Save address of message on stack
2388	21372B	=6191	CALL STRING ; Output the ESC seq to place the cursor
238B	CDE125	=6192	; on the last line of the screen.
238E	E1	=6193	POP H ; Restore the address of the error message
238F	CDE125	=6194	CALL STRING ; Print the message pointed to by HL
2392	C9	=6195	RET ; Return from ERRSTR.
		=6196 ;	
		=6197 ;	=====
		=6198 ;*	*
		=6199 ;*	Function : GETKEY (Get valid insert key). *
		=6200 ;*	Inputs : None *
		=6201 ;*	Outputs : A : The first valid character from the queue. *
		=6202 ;*	Destroys : Acc and flags. *
		=6203 ;*	Calls : KEYIN *
		=6204 ;*	*
		=6205 ;*	Description : This routine fetchs characters from the queue *
		=6206 ;*	(characters typed by the user on the keyboard) until a *
		=6207 ;*	valid text character is found. *
		=6208 ;*	*
		=6209 ;*	The following characters valid text characters: *
		=6210 ;*	*
		=6211 ;*	i : Any character with ASCII value > 20H *
		=6212 ;*	ii : MODCCH (the mode change character) *
		=6213 ;*	iii : The following control characters: *
		=6214 ;*	CR, TAB, QKILLC (Control K) *
		=6215 ;*	iv : The ASCII ESC character (start of cursor moves). *
		=6216 ;*	*
		=6217 ;	=====
		=6218 ;	
		=6219 GETKEY:	
		=6220	CALL KEYIN ; Start of <GETKEY>
2393	CDBE23	=6221 ;	; Get a key.
		=6222 ;	
		=6223 ;	Now check to see if the key is valid.)
		=6224	CPI MODCCH ; Mode change character ?
2396	FE1A	=6225	JZ GKEYOK ; Yes - That's OK.
2398	CAB423	=6226	;
239B	FE0D	=6227	CPI CR ; Carriage return ?
239D	CAB423	=6228	JZ GKEYOK ; Yes - That's OK.
		=6229	;
23A0	FE09	=6230	CPI TAB ; Horiz TAB ?
23A2	CAB423	=6231	JZ GKEYOK ; Yes - That's OK.
		=6232	;
23A5	FE1B	=6233	CPI ESC ; ESC ?
23A7	CAB423	=6234	JZ GKEYOK ; Yes - That's OK.
		=6235	;
23AA	FE0B	=6236	CPI QKILLC ; Queue killing character
23AC	CAB423	=6237	JZ GKEYOK ; Yes - That's OK
		=6238	;
23AF	FE20	=6239	CPI 20H ; Any other character with ASCII >= 20H ?
23B1	FA9323	=6240	JM GETKEY ; No - Then loop.
		=6241	;

LOC	OBJ	LINE	SOURCE STATEMENT
		=6242	GKEYOK: ;
23B4	C9	=6243	RET ; else return with character.
		=6244	;
		=6245	;=====
		=6246	;
		=6247	Function : INCPTR (increment pointer) *
		=6248	Inputs : A : Old input queue pointer *
		=6249	Outputs : A : New (incremented) input queue pointer *
		=6250	Destroys : Acc and Flags *
		=6251	Calls : None *
		=6252	;
		=6253	Description : This routine increments the input queue *
		=6254	pointer by one, with wrap around (i.e. when it reaches the *
		=6255	end of the buffer it is reset to zero. *
		=6256	;
		=6257	;=====
		=6258	;
		=6259	INCPTR: ; Start of increment character buffer pointers
23B5	3C	=6260	INR A ;
23B6	FE78	=6261	CPI KBLN ; If Ptr = End_of_buffer then
23B8	C2BD23	=6262	JNZ INCP05 ;
		=6263	;
23BB	3E00	=6264	MVI A,00H ; Ptr := 0
		=6265	INCP05: ;
23BD	C9	=6266	RET ; Return from <INCPTR> routine
		=6267	;
		=6268	;=====
		=6269	;
		=6270	Function : KEYIN (Get key from keyboard buffer.) *
		=6271	Inputs : None *
		=6272	Outputs : A : Character from top of input queue. *
		=6273	Destroys : Acc and flags. *
		=6274	Calls : INCPTR, KEYSTK, PACK *
		=6275	;
		=6276	Description : This routine (the editors basic character *
		=6277	input routine) fetchs the top character from the input *
		=6278	queue or waits for the a character to appear in the queue *
		=6279	if one is not there on entry to this routine. For more *
		=6280	details of the queue and how it is handled see the KEYSTK *
		=6281	routine. This routine has one other function and that is *
		=6282	to pack the file, if the program finds itself in the *
		=6283	situation where there are no characters in the queue and *
		=6284	the NULL count indicates that there are more than 20 *
		=6285	nulls in the file, this routine will automatically pack *
		=6286	the file. *
		=6287	;
		=6288	;=====
		=6289	;
		=6290	KEYIN: ; Start of KEYIN routine (Get key from user).
23BE	C5	=6291	PUSH B ;
23BF	D5	=6292	PUSH D ; Save all regs except Acc and flags
23C0	E5	=6293	PUSH H ;
		=6294	;
		=6295	NKEYYET: ;
23C1	CDF223	=6296	CALL KEYSTK ; Check keyboard.
23C4	3AEDE8	=6297	LDA UP ; Get buffer upper pointer.
23C7	57	=6298	MOV D,A ;
23C8	3AEDE8	=6299	LDA LO ; Get low buffer pointer.
23CB	BA	=6300	CMP D ; Upper pointer = lower pointer ?
23CC	CAE023	=6301	JZ BEMPTY ; If up=lo then buffer is empty.
		=6302	;
23CF	2175E8	=6303	LXI H,KEYBUF ; Get start adr of keyboard buffer.
23D2	5F	=6304	MOV E,A ;
23D3	1600	=6305	MVI D,0 ; Load 'DE' with lower pointer
23D5	19	=6306	DAD D ; Add pointer to start adr of key buffer
23D6	CDB523	=6307	CALL INCPTR ; Inc lower pointer , to next keypress.
23D9	32EEE8	=6308	STA LO ; Save lower pointer
23DC	7E	=6309	MOV A,M ; Load key press from buffer.
		=6310	;
23DD	C3EE23	=6311	JMP GOTKEY ; Exit with KEY in acc.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 110

LOC	OBJ	LINE	SOURCE STATEMENT
		=6312	;
		=6313	BEMPTY: ;
23E0	3A0BE8	=6314	LDA NULLS ;
23E3	FE0A	=6315	CPI NBPack ; Greater than 20 nulls yet.
23E5	FAEB23	=6316	JM FNULLS ; If nulls < 20 then do not pack
		=6317	;
23E8	CDE615	=6318	CALL PACK ; Pack the file.
		=6319	FNULLS: ;
23EB	C3C123	=6320	JMP NKEYYET ; Loop and repeat it all again. (will exit loop
		=6321	; when a key is typed and the pointer are
		=6322	; changed by the call to keystack).
		=6323	GOTKEY: ;
23EE	E1	=6324	POP H ;
23EF	D1	=6325	POP D ;
23F0	C1	=6326	POP B ; Restore all registers.
		=6327	;
23F1	C9	=6328	RET ; Return from <KEYIN>
		=6329	;
		=6330	=====
		=6331	;* ;
		=6332	;* Function : KEYSTK (Stack any incoming characters) *
		=6333	;* Inputs : None *
		=6334	;* Outputs : None *
		=6335	;* Destroys : None *
		=6336	;* Calls : CCMOVE, CONIN, CONTEST, INCPTR and STACKC *
		=6337	;* ;
		=6338	;* Description : This routine checks the keyboard (RxRdy bit *
		=6339	;* of terminal USART) to see if a character has been received *
		=6340	;* If a character has been received, then it is checked to *
		=6341	;* see if it is a flow control key (Control S and Control Q) *
		=6342	;* or a control_character_cursor move. If it is a flow *
		=6343	;* control key then the NOSEND flag is changed. If it is a *
		=6344	;* control_character_cursor move then the corresponding VT52 *
		=6345	;* escape sequence is packed at the bottom of the input queue. *
		=6346	;* ;
		=6347	;* The input queue: The input queue is a block of memory *
		=6348	;* (Less the 256 characters, as only using byte pointers) *
		=6349	;* with a fixed starting address and 2 byte wide pointers, a *
		=6350	;* top and a bottom pointer. The bottom pointer points to the *
		=6351	;* next character that must be fetched (the oldest character *
		=6352	;* in the queue) and the top pointer points to the next free *
		=6353	;* byte in the queue. If the pointers are equal then the *
		=6354	;* queue is empty. *
		=6355	;* ;
		=6356	=====
		=6357	;
		=6358	KEYSTK: ; Start of key stack routine.
		=6359	;
23F2	F5	=6360	PUSH PSW ; Save acc and flags of stack.
23F3	CDOAF0	=6361	CALL CONTEST ; Check USART for characters.
23F6	CA2D24	=6362	JZ NOKEY ; If no character then exit.
		=6363	;
23F9	E5	=6364	PUSH H ;
23FA	D5	=6365	PUSH D ; Save all other registers.
23FB	C5	=6366	PUSH B ;
		=6367	;
23FC	CD13F0	=6368	CALL CONIN ; Read character from USART.
23FF	FE13	=6369	CPI CTRLS ; ^S ?
2401	C20C24	=6370	JNZ NOTCS ; Yes - Then set NO_send flag.
		=6371	;
2404	3EFF	=6372	MVI A,TRUE ;
2406	3213E8	=6373	STA NOSEND ; Set nosend flag.
2409	C32A24	=6374	JMP KEYEX ; Exit the Key_stack routine.
		=6375	;
		=6376	NOTCS: ;
240C	FE11	=6377	CPI CTRLQ ; ^Q ?
240E	C21924	=6378	JNZ NOTCQ ; Yes - Then clear No_send flag.
		=6379	;
2411	3E00	=6380	MVI A,FALSE ;
2413	3213E8	=6381	STA NOSEND ; Clear No_send flag.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 111

LOC	OBJ	LINE	SOURCE STATEMENT
2416	C32A24	=6382	JMP KEYEX ; Exit the key_stack routine.
		=6383	;
		=6384	NOTCQ: ;
2419	CD7A14	=6385	CALL CCMOVE ; Is the Character a Control_move_char ?
241C	FEFF	=6386	CPI ERR ;
241E	CA2624	=6387	JZ NOTCMC ; No - Then just stack the character.
		=6388	;
2421	3E1B	=6389	MVI A,ESC ;
2423	CDBD25	=6390	CALL STACKC ; Else stack <ESC> and then char returned
		=6391	by the Control_char check routine.
		=6392	NOTCMC: ;
2426	78	=6393	MOV A,B ;
2427	CDBD25	=6394	CALL STACKC ; Stack the char.
		=6395	KEYEX: ;
242A	C1	=6396	POP B ;
242B	D1	=6397	POP D ; Restore all registers.
242C	E1	=6398	POP H ;
		=6399	NOKEY: ;
242D	F1	=6400	POP PSW ; Restore acc and flags.
		=6401	;
242E	C9	=6402	RET ; Return from <KEYSTK>
		=6403	;
		=6404	=====
		=6405	;
		=6406	Function : LPRINT (Letter print). *
		=6407	Inputs : A : ASCII value of character to print *
		=6408	Outputs : None *
		=6409	Destroys : None *
		=6410	Calls : LPRINT, PRINT *
		=6411	;
		=6412	Description : This routine is used to output text characters* *
		=6413	to the console, it has 3 basic functions. *
		=6414	;
		=6415	i : Space expand TAB (a tab stop every 8 characters) *
		=6416	ii : Output control characters a two printable chars. *
		=6417	iii : Keep a column count (used to expand TABs) *
		=6418	;
		=6419	=====
		=6420	;
		=6421	LPRINT: ; Start of letter print routine.
242F	F5	=6422	PUSH PSW ;
2430	FE09	=6423	CPI TAB ; If Char = TAB then
2432	C24524	=6424	JNZ LPNTAB ;
		=6425	Begin
		=6426	LPTAB: Repeat
2435	3E20	=6427	MVI A,SPACE ;
2437	CD2F24	=6428	CALL LPRINT Letter_print(' ');
		=6429	;
243A	3A04E8	=6430	LDA COL ;
243D	E607	=6431	ANI 00000111B ; Until Col MOD 8 = 0
243F	C23524	=6432	JNZ LPTAB ;
2442	C36824	=6433	JMP LPRTX ; End
		=6434	LPNTAB: ;
2445	FE00	=6435	CPI NULL ;
2447	CA6824	=6436	JZ LPRTX ;
		=6437	;
244A	FE20	=6438	CPI SPACE ; Elseif Char < Space AND Char # Null then
244C	F25E24	=6439	JP LPRTOK ; Begin
		=6440	;
244F	3E5E	=6441	MVI A,'^' ;
2451	CD2F24	=6442	CALL LPRINT Letter_print('^');
2454	F1	=6443	POP PSW ;
2455	F5	=6444	PUSH PSW ;
2456	C640	=6445	ADI '@' ;
2458	CD2F24	=6446	CALL LPRINT Letter_print(Char + '@')
245B	C36824	=6447	JMP LPRTX ; End
		=6448	;
		=6449	LPRTOK: ; Else
		=6450	Begin
245E	CD9D24	=6451	CALL PRINT ; Print(Char);

LOC	OBJ	LINE	SOURCE STATEMENT
2461	3A04E8	=6452	LDA COL ;
2464	3C	=6453	INR A ; Col := Col + 1;
2465	3204E8	=6454	STA COL ;
		=6455	; End
		=6456	LPRTEX: ; Endif
2468	F1	=6457	POP PSW ;
2469	C9	=6458	RET ; Return from <LPRINT> routine
		=6459 ;	
		=6460 ;	=====
		=6461 ;*	*
		=6462 ;*	Function : PBELL (Print bell) *
		=6463 ;*	Inputs : None *
		=6464 ;*	Outputs : None *
		=6465 ;*	Destroys : None *
		=6466 ;*	Calls : CONOUT *
		=6467 ;*	*
		=6468 ;*	Description : This routine sends an ASCII bell character to *
		=6469 ;*	the console. *
		=6470 ;*	*
		=6471 ;	=====
		=6472 ;	
		=6473	PBELL: ; Start of <PBELL> routine
246A	F5	=6474	PUSH PSW ; Save acc and flags
		=6475	;
246B	3E07	=6476	MVI A,BELL ;
246D	CD1CF0	=6477	CALL CONOUT ; Console_out(BELL)
		=6478	;
2470	F1	=6479	POP PSW ; Restore acc and flags
		=6480	;
2471	C9	=6481	RET ; Return from <PBELL> routine
		=6482 ;	
		=6483 ;	=====
		=6484 ;*	*
		=6485 ;*	Function : PLINE (print line) *
		=6486 ;*	Inputs : None *
		=6487 ;*	Outputs : None *
		=6488 ;*	Destroys : H,L,D,E,Acc and flags *
		=6489 ;*	Calls : FDCMP, LPRINT, QEOB, CLRL and BLINE *
		=6490 ;*	*
		=6491 ;*	Description : This routine outputs the current line to the *
		=6492 ;*	console. The start of the line is pointed to by the line *
		=6493 ;*	pointer 'POINT' and the line is terminated by an ASCII *
		=6494 ;*	line_feed character 'LF'. The routine also checks for the *
		=6495 ;*	end of the file by comparing the text pointer with the end *
		=6496 ;*	of text pointer. *
		=6497 ;*	*
		=6498 ;	=====
		=6499 ;	
		=6500	PLINE: ; Print_line.
2472	2A00E8	=6501	LHLD POINT ; Get start of line pointer
2475	CDD612	=6502	CALL QEOB ; End of Buffer ?
2478	C29924	=6503	JNZ EPLOOP ; Exit if End_Of_Buffer.
		=6504	;
247B	CD9D26	=6505	CALL BLINE ; Output <CR> and reset column position
		=6506 ;	
247E	2A00E8	=6507	LHLD POINT ; Get pointer to the start of the current line.
2481	EB	=6508	XCHG ;
2482	2AFEEF	=6509	LHLD EPOINT ; Get end of text pointer.
2485	EB	=6510	XCHG ;
		=6511 ;	
		=6512 ;	Pointers setup as follows: HL = Start of line pointer
		=6513 ;	DE = End of text pointer
		=6514 ;	
		=6515	PLOOP: ;
2486	CD3012	=6516	CALL FDCMP ; Compare the char Pointer and the EOT pointer.
2489	CA9924	=6517	JZ EPLOOP ; Exit if equal.
		=6518	;
248C	7E	=6519	MOV A,M ; get the character from the text.
248D	FE0A	=6520	CPI LF ; End of the line ?
248F	CA9924	=6521	JZ EPLOOP ; Exit if End_of_line

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 113

LOC	OBJ	LINE	SOURCE STATEMENT
		=6522	
2492	CD2F24	=6523	CALL LPRINT ; Else print the character.
2495	23	=6524	INX H ; Inc character pointer.
2496	C38624	=6525	JMP PLOOP ; Loop.
		=6526 ;	
		=6527	EPLOOP: ;
2499	CDA826	=6528	CALL CLRL ; Clear the current line.
249C	C9	=6529	RET ; Return from <PLINE>
		=6530 ;	
		=6531 ;	=====
		=6532 ;*	*
		=6533 ;*	Function : PRINT (print character on console) *
		=6534 ;*	Inputs : A : Character to print *
		=6535 ;*	Outputs : None *
		=6536 ;*	Destroys : None *
		=6537 ;*	Calls : KEYSTK and CONOUT *
		=6538 ;*	*
		=6539 ;*	Description : This routine is the basic character output *
		=6540 ;*	routine. The routine filters the output characters and *
		=6541 ;*	only if the character is found in the set of allowed *
		=6542 ;*	characters will the routine print the character. *
		=6543 ;*	*
		=6544 ;	=====
		=6545 ;	
		=6546	PRINT: ; Start of <PRINT> routine
		=6547	; ;
249D	F5	=6548	PUSH PSW ; Save acc and flags
		=6549	PRT05: ; Repeat
249E	CDF223	=6550	CALL KEYSTK ; Stack_any_incomming_characters
		=6551	; ;
24A1	3A13E8	=6552	LDA NOSEND ;
24A4	FEFF	=6553	CPI TRUE ;
24A6	CA9E24	=6554	JZ PRT05 ; Until NOSEND = FALSE;
		=6555	; ;
24A9	F1	=6556	POP PSW ;
		=6557	; ;
		=6558 ;	
		=6559 ;	If char in [LF,CR,ESC,BELL,BACK_SPACE,TAB] or
		=6560 ;	ASCII(char) > SPACE the print_it.
		=6561 ;	
24AA	FE0A	=6562	CPI LF ; If Char = LF
24AC	CACD24	=6563	JZ OKPIT ; then print_it
		=6564	; ;
24AF	FE0D	=6565	CPI CR ; If Char = CR
24B1	CACD24	=6566	JZ OKPIT ; then print_it
		=6567	; ;
24B4	FE1B	=6568	CPI ESC ; If Char = ESC
24B6	CACD24	=6569	JZ OKPIT ; then print_it
		=6570	; ;
24B9	FE07	=6571	CPI BELL ; If Char = BELL
24BB	CACD24	=6572	JZ OKPIT ; then print_it
		=6573	; ;
24BE	FE08	=6574	CPI BSPACE ; If Char = Back_Space
24C0	CACD24	=6575	JZ OKPIT ; then print_it
		=6576	; ;
24C3	FE09	=6577	CPI TAB ; If Char = TAB
24C5	CACD24	=6578	JZ OKPIT ; then print_it
		=6579	; ;
24C8	FE20	=6580	CPI SPACE ; If Char >= SPACE
24CA	FAD024	=6581	JM PRTEX ; then print_it
		=6582	; ;
		=6583	OKPIT: ; OK Print it.
24CD	CD1CF0	=6584	CALL CONOUT ; Output character via Monitor routine.
		=6585	PRTEX: ; Print Exit
24D0	C9	=6586	RET ; Return from <PRINT>.
		=6587 ;	
		=6588 ;	=====
		=6589 ;*	*
		=6590 ;*	Function : PTOEOL (print to end of line) *
		=6591 ;*	Inputs : None *

LOC	OBJ	LINE	SOURCE STATEMENT
		=6592 ;*	Outputs : None *
		=6593 ;*	Destroys : H,L,D,E,Acc and flags. *
		=6594 ;*	Calls : FDCMP, LPRINT *
		=6595 ;*	
		=6596 ;*	Description : This routine is very similar to the PLINE *
		=6597 ;*	routine, except this routine only prints from the current *
		=6598 ;*	character position to the end of the line rather than the *
		=6599 ;*	entire line as is done by the PLINE routine. *
		=6600 ;*	
		=6601 ;	=====
		=6602 ;	
		=6603	PTOEOL: ; Start of Print_to_end_of_line routine
		=6604	
24D1	2A02E8	=6605	LHLD CHAR ; Get current character pointer.
24D4	EB	=6606	XCHG ;
24D5	2AFEEF	=6607	LHLD EPOINT ; Get End_Of_Text pointer.
24D8	EB	=6608	XCHG ;
		=6609	; HL=Char pointer,DE=EOT pointer.
		=6610	PRTLOP: ;
24D9	CD3012	=6611	CALL FDCMP ;
24DC	CAF124	=6612	JZ PTOLEX ; If Char_pointer = EOT pointer then exit
		=6613	
24DF	7E	=6614	MOV A,M ; Get character from the text.
24E0	23	=6615	INX H ;
24E1	FE00	=6616	CPI NULL ; If char=NULL then ignore.
24E3	CAD924	=6617	JZ PRTLOP ; Loop until char # NULL.
		=6618	
24E6	FE0A	=6619	CPI LF ; End of the current line ?
24E8	CAF124	=6620	JZ PTOLEX ; If it is the exit.
		=6621	
24EB	CD2F24	=6622	CALL LPRINT ; Else print the character.
24EE	C3D924	=6623	JMP PRTLOP ; And loop.
		=6624	
		=6625	PTOLEX: ;
24F1	CDA826	=6626	CALL CLRL ; Clear to end of line.
24F4	C9	=6627	RET ; Return from <PTOEOL>
		=6628 ;	
		=6629 ;	=====
		=6630 ;*	
		=6631 ;*	Function : READS (read string) *
		=6632 ;*	Inputs : A : First character of the command. *
		=6633 ;*	Outputs : B : String length. *
		=6634 ;*	Destroys : H,L,B,C,Acc and flags. *
		=6635 ;*	Calls : GOTO, CLRL, KEYIN, PRINT and BLINE. *
		=6636 ;*	
		=6637 ;*	Description : This routine reads a string from the keyboard *
		=6638 ;*	echos it to the screen and places it in a buffer. The *
		=6639 ;*	delete key (DEL or RUBOUT) is supported. The entire line *
		=6640 ;*	can be cancelled by using the cancel key (Control U, as *
		=6641 ;*	used by the PDP-11/23). Control characters typed are *
		=6642 ;*	displayed as two printable ASCII characters, e.g. a line *
		=6643 ;*	feed would be displayed as a Control J '^J'. Certain *
		=6644 ;*	characters cannot be used in command lines, these are the *
		=6645 ;*	mode change character '^Z' and the ESC key. The length of *
		=6646 ;*	the command is checked to make sure that it does not *
		=6647 ;*	exceed the length of the command buffer, this length is *
		=6648 ;*	returned in the B register. *
		=6649 ;*	
		=6650 ;*	On entry to this routine the current column and row *
		=6651 ;*	positions of the cursor are saved, as OLINE and OCOL, *
		=6652 ;*	these values are restored by the RESCUR routine. *
		=6653 ;*	
		=6654 ;	=====
		=6655 ;	
		=6656	READS: ; Start of read string routine.
24F5	F5	=6657	PUSH PSW ; Save Acc and Flags.
		=6658	
24F6	3A05E8	=6659	LDA LINE ;
24F9	3207E8	=6660	STA OLINE ; Save current line position.
		=6661	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 115

LOC	OBJ	LINE	SOURCE STATEMENT	
24FC	3A04E8	=6662	LDA COL	;
24FF	3206E8	=6663	STA OCOL	; Save current col position.
		=6664		;
2502	3E17	=6665	MVI A,BOTLIN	;
2504	3205E8	=6666	STA LINE	; Set line to bottom line.
		=6667		;
2507	3E00	=6668	MVI A,00	;
2509	3204E8	=6669	STA COL	; Set col = zero.
		=6670		;
250C	CDB426	=6671	CALL GOTO	; Move cursor to <TOPLIN,BOTLIN>
250F	CDA826	=6672	CALL CLRL	; Clear the bottom line of the screen
2512	F1	=6673	POP PSW	; Restore Acc and Flags.
		=6674		;
2513	2117E8	=6675	LXI H,BUF	; HL = start adr of buffer.
2516	0600	=6676	MVI B,00H	; Init character counter (1st char in acc)
		=6677		;
2518	FE0D	=6678	CPI CR	; If 1st char is a <CR> then remove it.
251A	C22025	=6679	JNZ READS5	; Jump into the string read routine.
		=6680		;
		=6681	RNEXTC:	;
251D	CDBE23	=6682	CALL KEYIN	; Read char from the keyboard.
		=6683	READS5:	; Init entry point (char already in the A)
2520	FE1B	=6684	CPI ESC	;
2522	CA1D25	=6685	JZ RNEXTC	; Remove ESC chars.
		=6686		;
2525	FE1A	=6687	CPI MODCCH	;
2527	CA1D25	=6688	JZ RNEXTC	; Remove any Mode_Change_Characters.
		=6689		;
252A	FE15	=6690	CPI CTRLU	;
252C	CA7B25	=6691	JZ STRABO	; command line abort character.
		=6692		;
252F	FE0D	=6693	CPI CR	;
2531	CA8925	=6694	JZ STREND	; End of the line ?
		=6695		;
2534	FE7F	=6696	CPI DEL	;
2536	CA5C25	=6697	JZ STRDEL	; The DEL key ?
		=6698		;
2539	FE08	=6699	CPI RUBOUT	;
253B	CA5C25	=6700	JZ STRDEL	; Treat RUBOUT the same way as the DEL key.
		=6701		;
253E	4F	=6702	MOV C,A	; Save char
253F	78	=6703	MOV A,B	; Load # of chars entered thus far.
2540	FE50	=6704	CPI STRLEN	; Buffer full ?
2542	CA1D25	=6705	JZ RNEXTC	; Yes - Then do not accept that last character.
		=6706		;
2545	79	=6707	MOV A,C	; Restore character.
2546	77	=6708	MOV M,A	; Save in buffer.
2547	FE1B	=6709	CPI 27	; Is it a control char ?
2549	F25425	=6710	JP NOTCON	; No - Then goto Not control.
		=6711		;
254C	3E5E	=6712	MVI A,'^'	;
254E	CD9D24	=6713	CALL PRINT	; Print('^')
		=6714		;
2551	79	=6715	MOV A,C	; Restore char.
2552	C640	=6716	ADI 'A' - 1	; Char = Char + 'A' - 1 (1 becomes 'A')
		=6717	NOTCON:	;
2554	CD9D24	=6718	CALL PRINT	; Echo the keypush.
2557	23	=6719	INX H	; Inc pointer.
2558	04	=6720	INR B	; Inc char counter.
2559	C31D25	=6721	JMP RNEXTC	; Loop and read more characters.
		=6722		;
		=6723		;
		=6724		;
		=6725	String delete routine, Come here is <RUBOUT> or <DEL> key typed	
		=6726		;
		=6727	STRDEL:	;
255C	78	=6728	MOV A,B	; Load char counter
255D	FE00	=6729	CPI 00	; Any chars to delete.
255F	CA1D25	=6730	JZ RNEXTC	; No - Then ignore the delete key.
		=6731		;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 116

LOC	OBJ	LINE	SOURCE STATEMENT
2562	05	=6732	DCR B ; Dec char counter
2563	2B	=6733	DCX H ; Dec buffer pointer.
		=6734	;
2564	7E	=6735	MOV A,M ; Load the character.
2565	FE1B	=6736	CPI 27 ; Control character ?.
2567	3E08	=6737	MVI A,BSPACE ;
2569	F26F25	=6738	JP ONLY1BS ; No - Then only need 1 back space.
		=6739	;
256C	CD9D24	=6740	CALL PRINT ; Print 1 back space.
		=6741	ONLY1BS: ;
256F	CD9D24	=6742	CALL PRINT ; Print another backspace.
		=6743	;
2572	CDA826	=6744	CALL CLRL ; Clear the rest of the line.
2575	78	=6745	MOV A,B ; Load character count.
2576	FE00	=6746	CPI 00 ; Char count = zero ?
2578	C21D25	=6747	JNZ RNEXTC ; No - Then there are still characters on the
		=6748	line for the user to edit.
		=6749	;
		=6750 ;	
		=6751 ;	-----
		=6752 ;	
		=6753 ;	Abort string read routine, come here if <CTRL-U> typed.
		=6754 ;	
		=6755	STRABO: ;
257B	CD9D26	=6756	CALL BLINE ; Move cursor to the start of the line.
257E	CDA826	=6757	CALL CLRL ; Clear the line.
2581	3E04	=6758	MVI A,EOT ;
2583	3217E8	=6759	STA BUF ; Put EOT as 1st character in the buffer.
2586	C38E25	=6760	JMP STRRET ; Exit the string read routine.
		=6761 ;	
		=6762 ;	-----
		=6763 ;	
		=6764 ;	Normal string read exit point, come here when <CR> typed.
		=6765 ;	
		=6766	STREND: ;
2589	CD9D24	=6767	CALL PRINT ; Echo the <CR>
258C	3604	=6768	MVI M,EOT ; Mark end off string.
		=6769	;
		=6770	STRRET: ;
258E	C9	=6771	RET ; Return from <GETP> routine.
		=6772 ;	
		=6773 ;	=====
		=6774 ;*	*
		=6775 ;*	Function : RESCUR (Restore cursor) *
		=6776 ;*	Inputs : None *
		=6777 ;*	Outputs : None *
		=6778 ;*	Destroys : A Register. *
		=6779 ;*	Calls : GOTO *
		=6780 ;*	*
		=6781 ;*	Description : This routine restores the old cursor *
		=6782 ;*	position from OLINE/OCOL. *
		=6783 ;*	*
		=6784 ;	=====
		=6785 ;	
		=6786	RESCUR: ; Start of <RESCUR>
258F	3A07E8	=6787	LDA OLINE ;
2592	3205E8	=6788	STA LINE ; Restore old line position.
		=6789	;
2595	3A06E8	=6790	LDA OCOL ;
2598	3204E8	=6791	STA COL ; Restore old col position.
		=6792	;
259B	CDB426	=6793	CALL GOTO ;
259E	C9	=6794	RET ; Return from <RESCUR>
		=6795 ;	
		=6796 ;	=====
		=6797 ;*	*
		=6798 ;*	Function : SCPRESS (Space Compress) *
		=6799 ;*	Inputs : Command line. *
		=6800 ;*	Outputs : Packed command line. *
		=6801 ;*	Destroys : A,D,E,H,L and flags. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 117

LOC	OBJ	LINE	SOURCE STATEMENT
		=6802	; Calls : None.
		=6803	;
		=6804	; Description : This routine removes all the spaces from the
		=6805	; command buffer.
		=6806	;
		=6807	=====
		=6808	;
		=6809	SCPRESS: ; Procedure Space_compression;
		=6810	;
		=6811	Begin
259F	2117E8	=6811	LXI H,BUF ; Pointer := 1;
		=6812	SCP05: ;
25A2	7E	=6813	MOV A,M ;
25A3	FE20	=6814	CPI SPACE ; While Command_line[Pointer] = Space do
25A5	C2AC25	=6815	JNZ SCP10 ; Begin
		=6816	;
25A8	23	=6817	INX H ; Pointer := Pointer + 1
25A9	C3A225	=6818	JMP SCP05 ; End;
		=6819	SCP10: ;
25AC	1117E8	=6820	LXI D,BUF ; Output_Pointer := 1;
		=6821	SCP15: ;
25AF	7E	=6822	MOV A,M ;
25B0	FE04	=6823	CPI EOT ; While Command_Line[Pointer] = EOT do
25B2	CABB25	=6824	JZ SCP20 ; Begin
		=6825	;
25B5	12	=6826	STAX D ; Command_line[Output_pointer]
		=6827	:= Command_line[pointer];
		=6828	;
25B6	13	=6829	INX D ; Output_pointer :=
		=6830	Output_pointer + 1;
25B7	23	=6831	INX H ; Pointer := Pointer + 1;
25B8	C3AF25	=6832	JMP SCP15 ; End;
		=6833	;
		=6834	SCP20: ; End;
25BB	12	=6835	STAX D ; Command_line[Output_pointer] :=
		=6836	Command_line[Pointer];
		=6837	;
25BC	C9	=6838	RET ; End; (* Space_compression *)
		=6839	;
		=6840	=====
		=6841	;
		=6842	; Function : STACKC (Put character into input queue)
		=6843	; Inputs : A : Character to be placed in the queue
		=6844	; Outputs : None
		=6845	; Destroys : H,L,D,E,Acc and flags
		=6846	; Calls : INCPTR and PBELL
		=6847	;
		=6848	; Description : This routine places a character in the input
		=6849	; queue and increments the top pointer. If there is no room
		=6850	; in the input queue then a bell is sent to the console and
		=6851	; the character is ignored.
		=6852	;
		=6853	=====
		=6854	;
		=6855	STACKC: ; Start of stack character routine.
25BD	C5	=6856	PUSH B ;
25BE	47	=6857	MOV B,A ; Save character in the B_register
25BF	2175E8	=6858	LXI H,KEYBUF ; Get start adr of keyboard buffe.
25C2	3AEDE8	=6859	LDA UP ; Get value of upper pointer.
25C5	5F	=6860	MOV E,A ;
25C6	1600	=6861	MVI D,0 ; DE = Upper pointer.
25C8	19	=6862	DAD D ; HL = Start_of_buffer + Upper_pointer.
25C9	CDB523	=6863	CALL INCPTR ;
25CC	4F	=6864	MOV C,A ; Save new UPper pointer in C_register
25CD	3AEDE8	=6865	LDA LO ; Fetch Lower pointer.
25D0	B9	=6866	CMP C ; Equal ?
25D1	CADC25	=6867	JZ STK05 ; Yes - Then the buffer is full
		=6868	;
25D4	70	=6869	MOV M,B ; Put character into the buffer
25D5	79	=6870	MOV A,C ; Move new Upper pointer to A_register
25D6	32EDE8	=6871	STA UP ; Save new upper pointer.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 118

LOC	OBJ	LINE	SOURCE STATEMENT
25D9	C3DF25	=6872	JMP STK10 ;
		=6873	STK05: ;
25DC	CD6A24	=6874	CALL PBELL ; If the buffer is full send a bell to
		=6875	; the screen
		=6876	STK10: ;
25DF	C1	=6877	POP B ;
25E0	C9	=6878	RET ; Return from <STACKC> routine.
		=6879 ;	
		=6880 ;	=====
		=6881 ;	* * *
		=6882 ;	Function : STRING (Prints a message on the console) *
		=6883 ;	Inputs : HL : Pointer to the start of the message *
		=6884 ;	Outputs : None *
		=6885 ;	Destroys : H,L *
		=6886 ;	Calls : PRINT *
		=6887 ;	* * *
		=6888 ;	Description : This routine outputs a string to the console, *
		=6889 ;	the start address of the string is passed in the HL *
		=6890 ;	register pair and the end of the string if marker with an *
		=6891 ;	ASCII <EOT> character. *
		=6892 ;	* * *
		=6893 ;	=====
		=6894 ;	
		=6895	STRING: ; Start of string routine.
25E1	F5	=6896	PUSH PSW ; Save Acc and flags.
		=6897	STLOOP: ;
25E2	7E	=6898	MOV A,M ; Get string character.
25E3	FE04	=6899	CPI EOT ; End_of_text character ?
25E5	CAEF25	=6900	JZ ENDST ; Yes - Then exit string routine.
		=6901	; ;
25E8	CD9D24	=6902	CALL PRINT ; Else Print the character in the ACC.
25EB	23	=6903	INX H ; Inc the text pointer.
25EC	C3E225	=6904	JMP STLOOP ; Loop.
		=6905	; ;
		=6906	ENDST: ; String exit point.
25EF	F1	=6907	POP PSW ; Restore acc and flags.
25F0	C9	=6908	RET ; return from <STRING>
		=6909 ;	
		=6910 ;	End of File <EDTIO.A85>
		6911	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 119

LOC	OBJ	LINE	SOURCE STATEMENT
		6912	\$INCLUDE(EDTDEC.A85)
=6913			-----;
=6914			;
=6915			Decimal routines module for the UCT SABus editor.;
=6916			-----;
=6917			;
=6918			This module contains (in order) the following subroutines:;
=6919			;
=6920			1) DECMUL : Multiply HL register pair by 10;
=6921			2) PRINTD : Print decimal number to terminal.;
=6922			3) READNU : Read decimal number from input stream.;
=6923			;
=6924			-----;
=6925			;
=6926			\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 120

LOC	OBJ	LINE	SOURCE STATEMENT
		=6927 ;	=====
		=6928 ;*	
		=6929 ;*	Function : DECMUL (Decimal multiply)
		=6930 ;*	Inputs : HL = Value to multiply by 10.
		=6931 ;*	Outputs : HL = HL_Entry * 10.
		=6932 ;*	Destroys : Flags.
		=6933 ;*	Calls : None.
		=6934 ;*	
		=6935 ;*	Description: This routine multiplies the value pasted in HL
		=6936 ;*	by 10. Note: No overflow checking is done.
		=6937 ;*	
		=6938 ;	=====
		=6939 ;	
		=6940 DECMUL:	; Start of <DECMUL> routine.
25F1 D5		=6941 PUSH D	; Save DE.
25F2 29		=6942 DAD H	; HL2 := HL_Entry * 2
25F3 54		=6943 MOV D,H	
25F4 5D		=6944 MOV E,L	; DE := HL2
25F5 29		=6945 DAD H	; HL := HL2 * 2
25F6 29		=6946 DAD H	; HL := HL2 * 4
25F7 19		=6947 DAD D	; HL := HL2 * 4 + HL2, (HL = 10 * HL_Entry)
25F8 D1		=6948 POP D	; Restore DE.
25F9 C9		=6949 RET	; Return from <DECMUL>
		=6950 ;	
		=6951 ;	=====
		=6952 ;*	
		=6953 ;*	Function : PRINTD (Print decimal)
		=6954 ;*	Inputs : HL = Value to be printed.
		=6955 ;*	Outputs : None.
		=6956 ;*	Destroys : A, B, C, D, E, H, L and flags.
		=6957 ;*	Calls : PRINT.
		=6958 ;*	
		=6959 ;*	Description: This routine prints a decimal number on the
		=6960 ;*	screen. The value to be printed is pasted as a 16 bit
		=6961 ;*	binary number in the HL register pair. The method used to
		=6962 ;*	calculate the digits is subtraction of negative powers of
		=6963 ;*	ten until negative, this given the decimal digits.
		=6964 ;*	
		=6965 ;	=====
		=6966 ;	
		=6967 PRINTD:	; Start of <PRINTD> routine.
25FA 7C		=6968 MOV A,H	; Check if HL zero first
25FB B5		=6969 ORA L	
25FC C20526		=6970 JNZ OUTD05	; No - print it normally
		=6971	
25FF 3E30		=6972 MVI A,'0'	; Print "0".
2601 CD9D24		=6973 CALL PRINT	
2604 C9		=6974 RET	; and return
		=6975 OUTD05:	
2605 3E00		=6976 MVI A,FALSE	; suppress leading zeroes
2607 3210E8		=6977 STA PZRFLG	
260A 113E26		=6978 LXI D,PWR10	; point to table of powers of 10
		=6979 OUTD10:	
260D EB		=6980 XCHG	; table pointer into HL
260E 4E		=6981 MOV C,M	; put power of 10 into BC
260F 23		=6982 INX H	
2610 46		=6983 MOV B,M	
2611 23		=6984 INX H	
2612 EB		=6985 XCHG	; put number back to HL
2613 78		=6986 MOV A,B	; check to see if we're at the end
2614 B1		=6987 ORA C	; last table entry = 0
2615 C8		=6988 RZ	
		=6989	
2616 3E00		=6990 MVI A,0	; initialise digit to zero
2618 E5		=6991 PUSH H	; dummy push
		=6992 OUTD15:	
2619 33		=6993 INX SP	; correct stack for push
261A 33		=6994 INX SP	
261B E5		=6995 PUSH H	; save HL in case we go too far
261C 3C		=6996 INR A	; bump up digit

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 121

LOC	OBJ	LINE	SOURCE STATEMENT
261D	09	=6997	DAD B ; subtract the power of 10
261E	DA1926	=6998	JC OUTD15 ; can still take off more... go back
		=6999	;
2621	E1	=7000	POP H ; else restore HL again
2622	C62F	=7001	ADI '0'-1 ; convert to ASCII and decrement
2624	4F	=7002	MOV C, A ; put in C ready to output
2625	FE30	=7003	CPI '0' ; Is it zero?
2627	C23226	=7004	JNZ OUTD20 ; no - print it
		=7005	;
262A	3A10E8	=7006	LDA PZRFLG ; else check if we're suppressing
262D	FE00	=7007	CPI FALSE ; leading zeroes
262F	CA0D26	=7008	JZ OUTD10 ; Yes - don't print the zero
		=7009	OUTD20: ;
2632	79	=7010	MOV A,C ;
2633	CD9D24	=7011	CALL PRINT ; print the number
2636	3EFF	=7012	MVI A,TRUE ; must print zeroes from now on
2638	3210E8	=7013	STA PZRFLG ;
263B	C30D26	=7014	JMP OUTD10 ; loop back for next power
		=7015 ;	
		=7016 ;	
		=7017 ;	
		=7018 ;	Table of negative powers of TEN, Zero used to mark end of table.
		=7019 ;	
		=7020	PWR10:
263E	F0D8	=7021	DW -10000 ; - (10E4)
2640	18FC	=7022	DW -1000 ; - (10E3)
2642	9CFF	=7023	DW -100 ; - (10E2)
2644	F6FF	=7024	DW -10 ; - (10E1)
2646	FFFF	=7025	DW -1 ; - (10E0)
2648	0000	=7026	DW 0 ; 0 marks the last entry
		=7027 ;	
		=7028 ;	=====
		=7029 ;	*
		=7030 ;	Function : READNU : (READ NUmber from command line) *
		=7031 ;	Inputs : HL = Current character in the command line *
		=7032 ;	i.e. The 1st digit of the number *
		=7033 ;	Outputs : DE = Value read from the input string *
		=7034 ;	HL = Pointer to the 1st invalid digit. *
		=7035 ;	Zero_flag set if a valid number is read. *
		=7036 ;	Destroys : A, B, C, D, E, H, L and flags. *
		=7037 ;	Calls : QNUM and DECMUL. *
		=7038 ;	*
		=7039 ;	Description: This routine reads a decimal number from the *
		=7040 ;	an input line and returns the value in DE, the number of *
		=7041 ;	digits is limited to <DBLEN>. Leading zeros are removed. *
		=7042 ;	If there is an error in the number then the routine *
		=7043 ;	returns non_zero. *
		=7044 ;	*
		=7045 ;	=====
		=7046 ;	
		=7047	READNU: ; Start of read number routine.
264A	1171E8	=7048	LXI D,DIGBUF ;
264D	0E05	=7049	MVI C,DBLEN+1 ;
264F	0600	=7050	MVI B,FALSE ; Set number started = False.
		=7051	GETDIG: ;
2651	7E	=7052	MOV A,M ; Get char from command line.
2652	CD0613	=7053	CALL QNUM ; number ?
2655	C27726	=7054	JNZ LASTDIG ; If not number then exit.
		=7055	;
2658	78	=7056	MOV A,B ; Get number_started flag.
2659	FEFF	=7057	CPI TRUE ;
265B	CA6A26	=7058	JZ RNUM05 ; If Number_started = True then continue.
		=7059	;
265E	7E	=7060	MOV A,M ; Restore Digit .
265F	FE30	=7061	CPI '0' ;
2661	C26826	=7062	JNZ RNUM02 ;
		=7063	;
2664	23	=7064	INX H ;
2665	C35126	=7065	JMP GETDIG ; Leading zero, so ignore it and try next digit
		=7066	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 122

LOC	OBJ	LINE	SOURCE STATEMENT
		=7067	RNUM02:
2668	06FF	=7068	MVI B,TRUE
		=7069	RNUM05:
266A	7E	=7070	MOV A,M
266B	23	=7071	INX H
266C	12	=7072	STAX D
266D	13	=7073	INX D
266E	0D	=7074	DCR C
266F	C25126	=7075	JNZ GETDIG
		=7076	
		=7077	RNERR:
2672	3EFF	=7078	MVI A,OFFH
2674	C39B26	=7079	JMP RNUMEX
		=7080	
		=7081	LASTDIG:
2677	E5	=7082	PUSH H
2678	3E05	=7083	MVI A,DBLEN+1
267A	91	=7084	SUB C
267B	4F	=7085	MOV C,A
267C	1171E8	=7086	LXI D,DIGBUF
267F	210000	=7087	LXI H,0000H
2682	B7	=7088	ORA A
2683	CA9726	=7089	JZ NUMREAD
		=7090	
		=7091	NEXTDIG:
2686	CDF125	=7092	CALL DECMUL
2689	1A	=7093	LDAX D
268A	DE30	=7094	SBI '0'
268C	D5	=7095	PUSH D
268D	1600	=7096	MVI D,00H
268F	5F	=7097	MOV E,A
2690	19	=7098	DAD D
2691	D1	=7099	POP D
2692	13	=7100	INX D
2693	0D	=7101	DCR C
2694	C28626	=7102	JNZ NEXTDIG
		=7103	
		=7104	NUMREAD:
2697	EB	=7105	XCHG
		=7106	
2698	E1	=7107	POP H
2699	3E00	=7108	MVI A,00H
		=7109	RNUMEX:
269B	B7	=7110	ORA A
		=7111	
269C	C9	=7112	RET
		=7113	
		=7114	End of <EDTDEC.A85>
		7115	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 123

LOC	OBJ	LINE	SOURCE STATEMENT
		7116	\$INCLUDE(EDTESC.A85)
		=7117	-----
		=7118	;
		=7119	ESC sequence (VT52 assumed) module for the UCT SABus editor.
		=7120	-----
		=7121	;
		=7122	This module contains (in order) the following subroutines:
		=7123	;
		=7124	a) BLINE ; Beginning of line.
		=7125	b) CLRL ; Clear to end of line.
		=7126	c) CLRS ; Clear to end of screen.
		=7127	d) GOTO ; Direct cursor positioning.
		=7128	e) HOME ; Home cursor.
		=7129	f) MOVL ; Move cursor left.
		=7130	g) MOVR ; Move cursor right.
		=7131	h) NLINE ; Next line.
		=7132	i) REVLF ; Reverse line feed.
		=7133	j) SNDCOM ; Send <ESC> sequence to console
		=7134	;
		=7135	-----
		=7136	;
		=7137	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=7138 ;	=====
		=7139 ;*	
		=7140 ;*	Function : BLINE (Beginning of line routine) *
		=7141 ;*	Inputs : None *
		=7142 ;*	Outputs : None *
		=7143 ;*	Destroys : Acc and flags *
		=7144 ;*	Calls : PRINT *
		=7145 ;*	
		=7146 ;*	Description : This routine outputs a <CR> to the console *
		=7147 ;*	moving the cursor to column zero of the current line. *
		=7148 ;*	
		=7149 ;	=====
		=7150 ;	
		=7151 BLINE:	; Start of <BLINE> routine
269D	3E0D	=7152 MVI A,CR	;
269F	CD9D24	=7153 CALL PRINT	; Print a <CR>.
		=7154 ;	
26A2	3E00	=7155 MVI A,0	;
26A4	3204E8	=7156 STA COL	; Set col position = zero.
		=7157 ;	
26A7	C9	=7158 RET	; Return from <BLINE> routine
		=7159 ;	
		=7160 ;	=====
		=7161 ;*	
		=7162 ;*	Function : CLRL (Clear to end of line routine) *
		=7163 ;*	Inputs : None *
		=7164 ;*	Outputs : None *
		=7165 ;*	Destroys : Acc and flags *
		=7166 ;*	Calls : SNDCOM *
		=7167 ;*	
		=7168 ;*	Description : This routine clear the current line from the *
		=7169 ;*	cursor position to the end of the line. *
		=7170 ;*	
		=7171 ;*	VT52 Command = <ESC><Clear line code> *
		=7172 ;*	
		=7173 ;	=====
		=7174 ;	
		=7175 CLRL:	; Start of <CLRL> routine
26A8	3E4B	=7176 MVI A,CLRLC	; Load clear to end of line command.
26AA	CDF326	=7177 CALL SNDCOM	; Send the command.
26AD	C9	=7178 RET	; Return from <CLRL> routine
		=7179 ;	
		=7180 ;	=====
		=7181 ;*	
		=7182 ;*	Function : CLRS (clear screen routine) *
		=7183 ;*	Inputs : None *
		=7184 ;*	Outputs : None *
		=7185 ;*	Destroys : Acc and Flags *
		=7186 ;*	Calls : SNDCOM *
		=7187 ;*	
		=7188 ;*	Description : The routine clears the screen from the cursor *
		=7189 ;*	to the end of the screen. *
		=7190 ;*	
		=7191 ;*	VT52 Command = <ESC><Clear screen code> *
		=7192 ;*	
		=7193 ;	=====
		=7194 ;	
		=7195 CLRS:	; Start of <CLRS> routine
26AE	3E4A	=7196 MVI A,CLRSC	; Load clear to end of screen command.
26B0	CDF326	=7197 CALL SNDCOM	; Send the command.
26B3	C9	=7198 RET	; Return from <CLRS> routine
		=7199 ;	
		=7200 ;	
		=7201 ;	=====
		=7202 ;*	
		=7203 ;*	Function : GOTO (Goto <COL><LINE> position on screen) *
		=7204 ;*	Inputs : LINE = Line to move to. *
		=7205 ;*	COL = Column to move to. *
		=7206 ;*	Outputs : None. *
		=7207 ;*	Destroys : Acc and flags. *

LOC	OBJ	LINE	SOURCE STATEMENT
		=7208 ;*	Calls : PRINT. *
		=7209 ;*	*
		=7210 ;*	Description: This routine move the cursor to a given *
		=7211 ;*	point on the screen. The values of COL *
		=7212 ;*	and line are stored in memory. *
		=7213 ;*	*
		=7214 ;*	Before the COL and LINE values can be send an offset *
		=7215 ;*	has to be added to give a printable character. The *
		=7216 ;*	offset is equated (OFFSET) and is simply added. *
		=7217 ;*	*
		=7218 ;*	VT52 Command format *
		=7219 ;*	*
		=7220 ;*	<ESC> <Goto command code> <Line+offset> <Column+offset> *
		=7221 ;*	*
		=7222 ;*	=====
		=7223 ;*	*
		=7224 GOTO:	; Start of direct cursor positioning routine.
26B4	3E59	=7225	MVI A,GOTOC ;
26B6	CDF326	=7226	CALL SNDCOM ; Send GOTO command
		=7227	;
26B9	3A05E8	=7228	LDA LINE ;
26BC	C620	=7229	ADI OFFSET ;
26BE	CD9D24	=7230	CALL PRINT ; Print Chr(Offset+line)
		=7231	;
26C1	3A04E8	=7232	LDA COL ;
26C4	C620	=7233	ADI OFFSET ;
26C6	CD9D24	=7234	CALL PRINT ; Print Chr(Offset+col)
		=7235	;
26C9	C9	=7236	RET ; Return from Goto routine.
		=7237 ;	;
		=7238 ;*	=====
		=7239 ;*	*
		=7240 ;*	Function : HOME (Homes the terminal cursor) *
		=7241 ;*	Inputs : None. *
		=7242 ;*	Outputs : None. *
		=7243 ;*	Destroys : Acc and flags. *
		=7244 ;*	Calls : PRINT. *
		=7245 ;*	*
		=7246 ;*	Description: This routine simply homes the cursor and *
		=7247 ;*	resets the line and col counters to zero. *
		=7248 ;*	*
		=7249 ;*	VT52 command format = <ESC> <Home cursor code> *
		=7250 ;*	*
		=7251 ;*	=====
		=7252 ;*	*
		=7253 HOME:	; Start of <HOME> routine.
26CA	3E00	=7254	MVI A,00 ;
26CC	3204E8	=7255	STA COL ; Set horiz position = 0.
26CF	3205E8	=7256	STA LINE ; Set vert position = 0.
		=7257	;
26D2	3E48	=7258	MVI A,HOMEC ; Load home cursor command code.
26D4	CDF326	=7259	CALL SNDCOM ; Send the command.
26D7	C9	=7260	RET ; Return from <HOME> routine
		=7261 ;	;
		=7262 ;*	=====
		=7263 ;*	*
		=7264 ;*	Function : MOVL (move cursor left routine) *
		=7265 ;*	Inputs : None *
		=7266 ;*	Outputs : None *
		=7267 ;*	Destroys : Acc and flags *
		=7268 ;*	Calls : SNDCOM *
		=7269 ;*	*
		=7270 ;*	Description : This routine moves the cursor one character *
		=7271 ;*	left. *
		=7272 ;*	*
		=7273 ;*	VT52 command format = <ESC> <Move cursor left code> *
		=7274 ;*	*
		=7275 ;*	=====
		=7276 ;*	*
		=7277 MOVL:	; Start of <MOVL> routine

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 126

LOC	OBJ	LINE	SOURCE STATEMENT
26D8	3E44	=7278	MVI A,LEFTC ; Load cursor left command letter.
26DA	CDF326	=7279	CALL SNDCOM ; Send the command.
26DD	C9	=7280	RET ; Return from <MOVL> routine
		=7281 ;	
		=7282 ;=====	
		=7283 ;*	
		=7284 ;*	Function : MOVR (move cursor 1 character right) *
		=7285 ;*	Inputs : None *
		=7286 ;*	Outputs : None *
		=7287 ;*	Destroys : Acc and flags *
		=7288 ;*	Calls : SNDCOM *
		=7289 ;*	
		=7290 ;*	Description : This routine moves the cursor one character *
		=7291 ;*	to the right. *
		=7292 ;*	
		=7293 ;*	VT52 command format = <ESC> <move cursor right code> *
		=7294 ;*	
		=7295 ;=====	
		=7296 ;	
		=7297	MOVR:
26DE	3E43	=7298	MVI A,RIGHTC ; Load cursor right command letter.
26E0	CDF326	=7299	CALL SNDCOM ; Send the command.
26E3	C9	=7300	RET ;
		=7301 ;	
		=7302 ;=====	
		=7303 ;*	
		=7304 ;*	Function : NLINE (next line) *
		=7305 ;*	Inputs : None *
		=7306 ;*	Outputs : None *
		=7307 ;*	Destroys : Acc and flags *
		=7308 ;*	Calls : PRINT *
		=7309 ;*	
		=7310 ;*	Description : This routine moves the cursor to the beginning*
		=7311 ;*	of the next line. *
		=7312 ;*	
		=7313 ;=====	
		=7314 ;	
		=7315	NLINE: ; Start of <NLINE> routine
26E4	CD9D26	=7316	CALL BLINE ; Move cursor to the beginning of line
26E7	3E0A	=7317	MVI A,LF ;
26E9	CD9D24	=7318	CALL PRINT ; Print a <LF>.
26EC	C9	=7319	RET ; Return from <NLINE>
		=7320 ;	
		=7321 ;=====	
		=7322 ;*	
		=7323 ;*	Function : REVLF (reverse line feed) *
		=7324 ;*	Inputs : None *
		=7325 ;*	Outputs : None *
		=7326 ;*	Destroys : Acc and flags *
		=7327 ;*	Calls : SNDCOM *
		=7328 ;*	
		=7329 ;*	Description : This routine reverse line feeds the screen. *
		=7330 ;*	If the cursor was on the top line of the screen then the *
		=7331 ;*	screen is scrolled one line down and clears the top line. *
		=7332 ;*	
		=7333 ;*	VT52 command format = <ESC> <reverse line feed code> *
		=7334 ;*	
		=7335 ;=====	
		=7336 ;	
		=7337	REVLF:
26ED	3E49	=7338	MVI A,REVLFC ; Load reverse line feed command.
26EF	CDF326	=7339	CALL SNDCOM ; Send the command
26F2	C9	=7340	RET ;
		=7341 ;	
		=7342 ;=====	
		=7343 ;*	
		=7344 ;*	Function : SNDCOM (send command) *
		=7345 ;*	Inputs : A : Command code *
		=7346 ;*	Outputs : None *
		=7347 ;*	Destroys : None *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 127

LOC	OBJ	LINE	SOURCE STATEMENT
		=7348 ;*	Calls : PRINT *
		=7349 ;*	*
		=7350 ;*	Description : This routine outputs an <ESC> character *
		=7351 ;*	followed by the command code passed in the accumulator. *
		=7352 ;*	*
		=7353 ;	=====
		=7354 ;	
		=7355	SNDCOM:
26F3	F5	=7356	PUSH PSW ; Save the command code.
26F4	3E1B	=7357	MVI A,ESC ;
26F6	CD9D24	=7358	CALL PRINT ; Print an <ESC> character.
26F9	F1	=7359	POP PSW ; Restore command letter
26FA	CD9D24	=7360	CALL PRINT ; and print it.
26FD	C9	=7361	RET ; Return from all screen routines.
		=7362 ;	
		=7363 ;	End of module <EDTESC.A85>
		7364	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 128

LOC	OBJ	LINE	SOURCE STATEMENT
		7365	\$INCLUDE(EDTDEFINE.A85)
=7366			-----;
=7367			;
=7368			Key definition module of the SABus editor.
=7369			;
=7370			The following routines are contained in this module.
=7371			;
=7372			a) DEFINED : Implements key definitions.
=7373			b) DEFROU : Allows the keys to be defined.
=7374			c) FETCHD : Fetch information about deinition n.
=7375			d) KILL1C : Kill 1 character
=7376			e) KILLC : Kill character (including control characters)
=7377			f) PUTD : Put information about definition n.
=7378			g) PUTKD : Moves definition string from input buffer to the
=7379			definition data area.
=7380			h) READKD : Read delimited key definition from console.
=7381			;
=7382			-----;
=7383			
=7384			\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 128

LOC	OBJ	LINE	SOURCE STATEMENT
		7365	\$INCLUDE(EDTDEFINE.A85)
=7366			-----;
=7367			;
=7368			Key definition module of the SABus editor.
=7369			;
=7370			The following routines are contained in this module.
=7371			;
=7372			a) DEFINED : Implements key definitions.
=7373			b) DEFROU : Allows the keys to be defined.
=7374			c) FETCHD : Fetch information about deinition n.
=7375			d) KILL1C : Kill 1 character
=7376			e) KILLC : Kill character (including control characters)
=7377			f) PUTD : Put information about definition n.
=7378			g) PUTKD : Moves definition string from input buffer to the
=7379			definition data area.
=7380			h) READKD : Read delimited key definition from console.
=7381			;
=7382			-----;
=7383			;
=7384			\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=7385 ;	=====
		=7386 ;*	
		=7387 ;*	Function : DEFINED
		=7388 ;*	Inputs : B : 2nd character in the function key <ESC>
		=7389 ;*	sequence, in the range '0'..'9'
		=7390 ;*	Outputs : None
		=7391 ;*	Destroys : A, B, C, D, E, H, L and flags.
		=7392 ;*	Calls : FETCHD and STACKC.
		=7393 ;*	
		=7394 ;*	Description : This routine, using the key number passed,
		=7395 ;*	checks to see if the key typed has been defined and if it
		=7396 ;*	has then the definition string is copied from the data
		=7397 ;*	area to the input queue. If the key is undefined, a length
		=7398 ;*	of zero, then an error message is printed.
		=7399 ;*	
		=7400 ;	=====
		=7401 ;	
		=7402 DEFINED:	; Start of <DEFINED> routine
		=7403	;
26FE	78	=7404	MOV A,B ; Restore 2nd character in ESC seq
26FF	D630	=7405	SUI '0' ; convert to range 0..9
2701	CD6027	=7406	CALL FETCHD ; Fetch entry data.
		=7407	; B : Length of definition in bytes.
		=7408	; DE : Start adr of definition in data area.
		=7409	;
2704	78	=7410	MOV A,B ; Move length byte to acc
2705	FE00	=7411	CPI 0 ; Zero length string ? (Zero => undefined)
2707	CA1927	=7412	JZ DEFR05 ; Yes - then exit with error message
		=7413 ;	
		=7414 ;	-----
		=7415 ;	
		=7416 ;	Key is defined, move string to input buffer
		=7417 ;	
270A	EB	=7418	XCHG ; Move start address of definition to HL.
		=7419 DEFR02:	;
270B	7E	=7420	MOV A,M ; Fetch character from
270C	E5	=7421	PUSH H ; Save pointer
270D	CDBD25	=7422	CALL STACKC ; Put character in input Q
2710	E1	=7423	POP H ; Restore pointer
2711	23	=7424	INX H ; Inc string pointer
2712	05	=7425	DCR B ;
2713	C20B27	=7426	JNZ DEFR02 ; Loop for all keys in buffer
		=7427	;
2716	C31F27	=7428	JMP DEFR20 ; Exit from routine
		=7429 ;	
		=7430 ;	-----
		=7431 ;	
		=7432 ;	Key is undefined, Print an error message.
		=7433 ;	
		=7434 DEFR05:	;
2719	219A2E	=7435	LXI H,UNDEFK ;
271C	CD8723	=7436	CALL ERRSTR ; Print error message
		=7437 DEFR20:	; Defined exit point
271F	3E00	=7438	MVI A,FALSE ;
2721	320FE8	=7439	STA ESCFLG ; Reset ESC flag.
2724	C9	=7440	RET
		=7441 ;	
		=7442 ;	=====
		=7443 ;*	
		=7444 ;*	Function : DEFROU (define routine)
		=7445 ;*	Inputs : BC : Pointer to the next character in the
		=7446 ;*	command line.
		=7447 ;*	Outputs : None
		=7448 ;*	Destroys : A, B, C, D, E, H, L and flags.
		=7449 ;*	Calls : BLINE, CLRL, STRING, READKD, PUTKD and ERRSTR
		=7450 ;*	
		=7451 ;*	Description : This routine is called when a 'DEFn' command
		=7452 ;*	is typed. The routine 1st checks to see if 'n' is in the
		=7453 ;*	range '0'..'9'. If 'n' is not in this range then an error
		=7454 ;*	message is printed. If 'n' is in range then the user is

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 130

LOC	OBJ	LINE	SOURCE STATEMENT
		=7455 ;*	prompted for a delimited key definition - i.e. this 1st *
		=7456 ;*	character typed is not part of the definition but is used *
		=7457 ;*	as a delimiter, this allows the return key to be entered *
		=7458 ;*	as part of the definition. The definition entry is *
		=7459 ;*	terminated when the 2nd delimiter is entered. *
		=7460 ;*	*****
		=7461 ;	*****
		=7462 ;	*****
		=7463 DEFROU:	; Start of <DEFROU> routine.
2725	60	=7464	MOV H,B ;
2726	69	=7465	MOV L,C ; Move pointer to TERMCH to HL
2727	7E	=7466	MOV A,M ; Fetch TERMCH
2728	FE30	=7467	CPI '0' ; Check for limits
272A	FA5927	=7468	JM DEF20 ;
272D	FE3A	=7469	CPI '9'+1 ;
272F	F25927	=7470	JP DEF20 ; If n<'0' or n>'9' then error.
		=7471 ;	;
2732	D630	=7472	SUI '0' ; Convert to binary
2734	3215E9	=7473	STA KEYNUM ; Save defined key
2737	23	=7474	INX H ; Check that the next character is a EOT
2738	7E	=7475	MOV A,M ; Fetch Character
2739	FE04	=7476	CPI EOT ; EOT ?
273B	C25927	=7477	JNZ DEF20 ; If not EOT then DEF syntax error
		=7478 ;	;
		=7479 ;	-----
		=7480 ;	;
		=7481 ;	N was within range and terminated, now fetch pointers
		=7482 ;	;
273E	CD9D26	=7483	CALL BLINE ; Move cursor to beginning of the line
2741	CDA826	=7484	CALL CLRL ; Clear to end of line
2744	21642E	=7485	LXI H,DEFMES ;
2747	CDE125	=7486	CALL STRING ; Print key definition message
274A	CDCD27	=7487	CALL READKD ; Read key def from keyboard
274D	CDA427	=7488	CALL PUTKD ; Put def in table
2750	CD9D26	=7489	CALL BLINE ; Move cursor to the beginning of the line
2753	CDA826	=7490	CALL CLRL ; Clear the line.
2756	C35F27	=7491	JMP DEF25 ; exit.
		=7492 ;	;
		=7493 DEF20:	; Define key range error (DEFn n=0..9)
2759	21802E	=7494	LXI H,DEFERR ;
275C	CD8723	=7495	CALL ERRSTR ; Print Define key error message
		=7496 DEF25:	;
275F	C9	=7497	RET ; Return from <DEFROU>
		=7498 ;	;
		=7499 ;	=====
		=7500 ;*	*****
		=7501 ;*	Function : FETCHD (fetch key definition for key n) *
		=7502 ;*	Inputs : A : Key number (in the range 0..9) *
		=7503 ;*	Outputs : B : Length of the definition string. *
		=7504 ;*	DE : Start address of the definition string. *
		=7505 ;*	Destroys : A,B,D,E,H,L and flags. *
		=7506 ;*	Calls : None *
		=7507 ;*	*****
		=7508 ;*	Description : This routine fetches the length and address *
		=7509 ;*	of key definition n, where n is in the range 0..9. If the *
		=7510 ;*	length, returned in the B register, is zero then this *
		=7511 ;*	means that the key is undefined. The length and start *
		=7512 ;*	of the 10 entries are stored in a 30 byte table with each *
		=7513 ;*	entry having the following format: *
		=7514 ;*	*****
		=7515 ;*	1st byte 2nd byte 3rd byte *
		=7516 ;*	*****
		=7517 ;*	LSByte of start MSByte of start Definition *
		=7518 ;*	address address length *
		=7519 ;*	*****
		=7520 ;	=====
		=7521 ;	*****
		=7522 FETCHD:	; Fetch defined key parameters
2760	47	=7523	MOV B,A ;
2761	87	=7524	ADD A ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 131

LOC	OBJ	LINE	SOURCE STATEMENT
2762	80	=7525	ADD B ; Acc = Acc * 3
2763	21F7E8	=7526	LXI H,DEFTAB ; HL = Start of vector table
2766	1600	=7527	MVI D,0 ;
2768	5F	=7528	MOV E,A ; DE = Entry number * 4
2769	19	=7529	DAD D ; HL = Start of entry number n
276A	5E	=7530	MOV E,M ;
276B	23	=7531	INX H ;
276C	56	=7532	MOV D,M ; DE = Pointer to start of entry
		=7533	;
276D	23	=7534	INX H ;
276E	46	=7535	MOV B,M ; B = Length of key definition
		=7536	;
276F	C9	=7537	RET ; Return from <FETCHD> routine
		=7538	;
		=7539	=====
		=7540	;
		=7541	Function : KILL1C (Kill one character) *
		=7542	Inputs : None *
		=7543	Outputs : None *
		=7544	Destroys : None *
		=7545	Calls : PRINT *
		=7546	;
		=7547	Description : This routine is used when the delete key is *
		=7548	typed to back_space over the character and replace it with *
		=7549	a space. The routine sends the following sequence of *
		=7550	characters to the console: *
		=7551	;
		=7552	<back_space> <space> <back_space> *
		=7553	;
		=7554	=====
		=7555	;
		=7556	KILL1C: ; Start of <KILL1C> routine
2770	F5	=7557	PUSH PSW ; Save Acc and flags
		=7558	;
2771	3E08	=7559	MVI A,BSPACE ;
2773	CD9D24	=7560	CALL PRINT ; Output a back space
2776	3E20	=7561	MVI A,SPACE ;
2778	CD9D24	=7562	CALL PRINT ; Output a space
277B	3E08	=7563	MVI A,BSPACE ;
277D	CD9D24	=7564	CALL PRINT ; Output a back space
		=7565	;
2780	F1	=7566	POP PSW ; Restore Acc and flags
2781	C9	=7567	RET ; Return from <KILL1C> routine
		=7568	;
		=7569	=====
		=7570	;
		=7571	Function : KILLC (kill character) *
		=7572	Inputs : A : ASCII character to be killed. *
		=7573	Outputs : None *
		=7574	Destroys : Flags. *
		=7575	Calls : KILL1C *
		=7576	;
		=7577	Description : This routine uses KILL1C to delete characters *
		=7578	from the screen, if the character is a control character - *
		=7579	ASCII value less than that of a space - then two character *
		=7580	positions are cleared. *
		=7581	;
		=7582	=====
		=7583	;
		=7584	KILLC: ; Start of <KILLC> routine
2782	CD7027	=7585	CALL KILL1C ; Always kill a least 1 character position
2785	FE20	=7586	CPI SPACE ; If Char < Space then
2787	F28D27	=7587	JP KILL05 ; Kill another character position
278A	CD7027	=7588	CALL KILL1C ;
		=7589	KILL05: ;
278D	C9	=7590	RET ; Return from <KILLC>
		=7591	;
		=7592	=====
		=7593	;
		=7594	Function : PUTD (put key definition data) *

LOC	OBJ	LINE	SOURCE STATEMENT
		=7595 ;*	Inputs : A : Key number in the range 0..9 *
		=7596 ;*	B : length of the key definition. *
		=7597 ;*	DE : Start address of the key definition. *
		=7598 ;*	Outputs : None *
		=7599 ;*	Destroys : Acc and flags *
		=7600 ;*	Calls : None *
		=7601 ;*	*
		=7602 ;*	Description : This routine saves the information, passed in *
		=7603 ;*	the registers, about the definition in the key definition *
		=7604 ;*	table. *
		=7605 ;*	*
		=7606 ;	=====
		=7607 ;	
		=7608 PUTD:	; Fetch defined key parameters
278E	C5	=7609	PUSH B ;
278F	E5	=7610	PUSH H ; Save registers.
2790	D5	=7611	PUSH D ;
		=7612	;
2791	4F	=7613	MOV C,A ; Move key number to C register
2792	87	=7614	ADD A ;
2793	81	=7615	ADD C ; Acc = Acc * 3
2794	21F7E8	=7616	LXI H,DEFTAB ; HL = Start of vector table
2797	1600	=7617	MVI D,0 ;
2799	5F	=7618	MOV E,A ; DE = Entry number * 3
279A	19	=7619	DAD D ; HL = Start of entry number n
279B	D1	=7620	POP D ;
		=7621	;
279C	73	=7622	MOV M,E ; Save LSByte
279D	23	=7623	INX H ;
279E	72	=7624	MOV M,D ; Save MSByte
279F	23	=7625	INX H ;
27A0	70	=7626	MOV M,B ; Save len
27A1	E1	=7627	POP H ;
27A2	C1	=7628	POP B ;
27A3	C9	=7629	RET ; Return from <PUTD> routine
		=7630 ;	
		=7631 ;	=====
		=7632 ;*	*
		=7633 ;*	Function : PUTKD (put key definition) *
		=7634 ;*	Inputs : A : length of string in input buffer. *
		=7635 ;*	Outputs : None *
		=7636 ;*	Destroys : B,C,D,E,H and L *
		=7637 ;*	Calls : PUTD *
		=7638 ;*	*
		=7639 ;*	Description : This routine calculates the address in the *
		=7640 ;*	key definition data area where the definition for key n is *
		=7641 ;*	to be stored. This address along with the length of the *
		=7642 ;*	definition is then stored in the key definition table and *
		=7643 ;*	the definition copied from the input buffer to the data *
		=7644 ;*	area at the address calculated. *
		=7645 ;*	*
		=7646 ;	=====
		=7647 ;	
		=7648 PUTKD:	; Start of <PUTKD> routine
27A4	F5	=7649	PUSH PSW ; Save string length (A register) on stack
		=7650	;
27A5	3A15E9	=7651	LDA KEYNUM ; Fetch key number
27A8	115000	=7652	LXI D,STRLEN ; DE = String length
27AB	21C6E8	=7653	LXI H,KDTAB-STRLEN ; HL = Address of buffer number -1
27AE	3C	=7654	INR A ;
		=7655 PUTKD5:	;
27AF	19	=7656	DAD D ;
27B0	3D	=7657	DCR A ;
27B1	C2AF27	=7658	JNZ PUTKD5 ;
		=7659	; HL = Key_definition_data_area
		=7660	; + String_length * key_number
27B4	EB	=7661	XCHG ;
27B5	2118E8	=7662	LXI H,BUF+1 ; HL = Start address of definition in input
		=7663	buffer
27B8	F1	=7664	POP PSW ; Restore input buffer length

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 133

LOC	OBJ	LINE	SOURCE STATEMENT
		=7665	
27B9	3D	=7666	DCR A ; Skip 1st delimiter
27BA	CACC27	=7667	JZ PUTKDX ; Exit if zero len string
		=7668	
27BD	47	=7669	MOV B,A ; Put len in B
27BE	3A15E9	=7670	LDA KEYNUM ; Fetch key number
27C1	CD8E27	=7671	CALL PUTD ; Save definition information for key n
		=7672	PUTKDL: ;
27C4	7E	=7673	MOV A,M ;
27C5	12	=7674	STAX D ;
27C6	23	=7675	INX H ; Copy key definition from the input buffer
27C7	13	=7676	INX D ; to the definition data area.
27C8	05	=7677	DCR B ;
27C9	C2C427	=7678	JNZ PUTKDL ;
		=7679	
		=7680	PUTKDX: ;
27CC	C9	=7681	RET ; Return from <PUTKD>
		=7682	;
		=7683	=====
		=7684	;
		=7685	* Function : READKD (read key definition) *
		=7686	* Inputs : None *
		=7687	* Outputs : A : Length of string in input buffer. *
		=7688	* Destroys : All *
		=7689	* Calls : KEYIN, KILLC, BLINE, CLRL and LPRINT *
		=7690	;
		=7691	* Description : This routine reads a delimited string from *
		=7692	* keyboard into the input buffer, the 1st character typed is *
		=7693	* used as the delimiter, but this can be changed by using *
		=7694	* the delete key. The routine checks that the buffer is *
		=7695	* never overflowed. Every character, except the delete key, *
		=7696	* can be used as a delimiter and every key except the delete *
		=7697	* key and the delimiter can be entered as part of the string. *
		=7698	;
		=7699	=====
		=7700	;
		=7701	READKD: ; Start of routine to read key def.
27CD	0600	=7702	MVI B,0 ; Init counter = zero
27CF	2117E8	=7703	LXI H,BUF ; Fetch start adr of buffer
27D2	CD8E23	=7704	CALL KEYIN ; Fetch the 1st character (Delimiter)
27D5	FE7F	=7705	CPI DEL ; 1st character = delete ?
27D7	CA2128	=7706	JZ RDEF20 ; Yes - then exit
		=7707	
27DA	F5	=7708	PUSH PSW ; Save character
27DB	CD9D26	=7709	CALL BLINE ;
27DE	CDA826	=7710	CALL CLRL ; Clear the prompt line
27E1	F1	=7711	POP PSW ; Restore character
		=7712	;
27E2	CD2F24	=7713	CALL LPRINT ; Echo the delimiter
27E5	77	=7714	MOV M,A ; Save delimiter as 1st character in buffer
27E6	23	=7715	INX H ; Increment buffer pointer (to BUF+1)
27E7	04	=7716	INR B ; Increment character counter (to 1)
		=7717	RDEF10: ;
27E8	CD8E23	=7718	CALL KEYIN ; Fetch a character from the keyboard
27EB	FE7F	=7719	CPI DEL ; Delete key ?
27ED	CA0C28	=7720	JZ RDEF15 ; Yes - then handle delete key
		=7721	
27F0	CD2F24	=7722	CALL LPRINT ; Echo the character
27F3	77	=7723	MOV M,A ; Save character in the buffer
27F4	3A17E8	=7724	LDA BUF ; Fetch delimiter
27F7	BE	=7725	CMP M ; Was the last character = delimiter ?
27F8	CA2128	=7726	JZ RDEF20 ; Yes - Then exit
		=7727	
27FB	23	=7728	INX H ; Increment buffer pointer
27FC	04	=7729	INR B ; Increment character counter
27FD	78	=7730	MOV A,B ; Move character count to the Acc
27FE	FE50	=7731	CPI STRLEN ; Buffer full ?
2800	C2E827	=7732	JNZ RDEF10 ; Loop until buffer full or delimiter.
		=7733	;
2803	2B	=7734	DCX H ; Dec pointer to last byte in buffer

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 134

LOC	OBJ	LINE	SOURCE STATEMENT
2804	7E	=7735	MOV A,M ; Fetch character to be removed
2805	CD8227	=7736	CALL KILLC ; Kill the character
2808	05	=7737	DCR B ; Dec counter
2809	C3E827	=7738	JMP RDEF10 ; Loop and wait for delimiter
		=7739	RDEF15: ;
280C	05	=7740	DCR B ;
280D	C21928	=7741	JNZ RDEF17 ; If Buffer not empty then simply kill char
		=7742	;
2810	CD9D26	=7743	CALL BLINE ; Clear the command line
2813	CDA826	=7744	CALL CLRL ;
2816	C3CD27	=7745	JMP READKD ; If delimiter deleted restart
		=7746	;
		=7747	RDEF17: ; Else dec pointer and loop.
2819	2B	=7748	DCX H ;
281A	7E	=7749	MOV A,M ;
281B	CD8227	=7750	CALL KILLC ; Remove the character from the line.
281E	C3E827	=7751	JMP RDEF10 ; Loop and fetch next character
		=7752	;
		=7753	RDEF20: ; Delimiter typed. (Set A = len and exit)
2821	78	=7754	MOV A,B ;
2822	C9	=7755	RET ; Return from <READKD>
		=7756 ;	
		=7757 ;	End of Define key routines
		7758	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 135

LOC	OBJ	LINE	SOURCE STATEMENT
		7759 ;	
		7760 ;	Include Data area and tables.
		7761 ;	
		7762	\$INCLUDE(EDTTABLE.A85)
		=7763 ;	-----;
		=7764 ;	;
		=7765 ;	Editor Command Tables
		=7766 ;	-----;
		=7767 ;	;
		=7768 ;	Editor commands (those typed on the last line of the screen while in
		=7769 ;	command mode) are grouped into 2 types:
		=7770 ;	;
		=7771 ;	a) Those the can be identified by the 1st character of the command
		=7772 ;	e.g. A substitute command 'S/FRED/JIM/'
		=7773 ;	;
		=7774 ;	b) Those that require an entire string to identify them
		=7775 ;	e.g. A save command 'SAVE FRED.A85'
		=7776 ;	;
		=7777 ;	Another typed of command is a cursor movement command, i.e. the typing off
		=7778 ;	one of the cursor keys. These keys produce an escape sequence, an <ESC>
		=7779 ;	character followed by a character in the range 'A'..'D'.
		=7780 ;	;
		=7781 ;	-----;
		=7782	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 136

LOC	OBJ	LINE	SOURCE STATEMENT
		=7783 ;	
		=7784 ;	Command/cursor movement tables. (ESC codes)
		=7785 ;	
		=7786 CMTAB:	; Start of Table.
		=7787	;
2823	41	=7788 DB	UPC ; Cursor up code.
2824	E51C	=7789 DW	UPROU ; Cursor uP vector.
		=7790	;
2826	42	=7791 DB	DOWNC ; Cursor down code.
2827	361D	=7792 DW	DOWNRO ; Cursor down vector.
		=7793	;
2829	43	=7794 DB	RIGHTC ; Cursor right code.
282A	931D	=7795 DW	RIGHTR ; Cursor right vector.
		=7796	;
282C	44	=7797 DB	LEFTC ; Cursor left code.
282D	CA1D	=7798 DW	LEFTR ; Cursor left vector.
		=7799	;
282F	FF	=7800 DB	ETABLE ; End of table.
		=7801 ;	
		=7802 ;	-----
		=7803 ;	
		=7804 ;	As some terminals/keyboards do not have cursor keys it has been made
		=7805 ;	possible to used 4 control keys as cursor keys, one for each of the
		=7806 ;	4 directions. A table is given below to allow the program to check
		=7807 ;	a given control code with the list of control keys being used as
		=7808 ;	cursor keys. The <ESC> sequence character into which the control code
		=7809 ;	must be translated is given as the second byte of each entry.
		=7810 ;	
		=7811 ;	Cursor movement tables (control codes)
		=7812 ;	
		=7813 CCMOVT:	; Start of Control to Esc_code table.
		=7814	;
2830	12	=7815 DB	CONTUP ; Cursor up (Control code)
2831	41	=7816 DB	UPC ; Cursor up (ESC code)
		=7817	;
2832	03	=7818 DB	CONTD ; Cursor down (Control code)
2833	42	=7819 DB	DOWNC ; Cursor down (ESC code)
		=7820	;
2834	06	=7821 DB	CONTRI ; Cursor right (Control code)
2835	43	=7822 DB	RIGHTC ; Cursor right (ESC code)
		=7823	;
2836	04	=7824 DB	CONTR ; Cursor left (Control code)
2837	44	=7825 DB	LEFTC ; Cursor left (ESC code)
		=7826	;
2838	FF	=7827 DB	ETABLE ; End of table.
		=7828 ;	
		=7829 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 137

LOC	OBJ	LINE	SOURCE STATEMENT
		=7830 ;	-----
		=7831 ;	
		=7832 ;	The list of commands of type A follows.
		=7833 ;	
		=7834 ;	Editor command table (where the command is a letter with options)
		=7835 ;	
		=7836 COMTAB:	; Start of table.
		=7837	;
2839	27	=7838	DB QUOTE ; Quote = Find routine.
283A	AB17	=7839	DW QFINDR ; Vector to the quote find routine.
		=7840	;
283C	46	=7841	DB 'F' ; 'F' also = find routine.
283D	1A17	=7842	DW FINDRO ; Vector to find routine.
		=7843	;
283F	44	=7844	DB 'D' ; 'D' = Delete routine.
2840	E51A	=7845	DW DELROU ; Vector to the delete routine.
		=7846	;
2842	53	=7847	DB 'S' ; 'S' = Substitute routine.
2843	C818	=7848	DW SUBROU ; Vector to the Sub routine.
		=7849	;
2845	4C	=7850	DB 'L' ; 'L' = Line down routine.
2846	271C	=7851	DW LINROU ; Vector to line down routine.
		=7852	;
2848	49	=7853	DB 'I' ; 'I' = Insert mode.
2849	981F	=7854	DW IROU ; Vector to set insert mode routine.
		=7855	;
284B	43	=7856	DB 'C' ; 'C' = Cut.
284C	011E	=7857	DW CUTROU ; Vector to cut routine.
		=7858	;
284E	FF	=7859	DB ETABLE ; End of table.
		=7860 ;	
		=7861 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 138

LOC	OBJ	LINE	SOURCE STATEMENT
		=7862 ;	-----
		=7863 ;	
		=7864 ;	Editor command table (where the command is a word with no options)
		=7865 ;	
		=7866 ;	NOTE: load, save and define entries come 1st so that there
		=7867 ;	entry numbers are zero, one and two.
		=7868 ;	
		=7869 ;	-----
		=7870 ;	
0000		=7871 LOADEN EQU 0	; Load entry number.
0001		=7872 SAVEEN EQU 1	; Save entry number.
0002		=7873 DEFEN EQU 2	; Define entry number.
		=7874 ;	
		=7875 ;	-----
		=7876 ;	
		=7877 STRCOM:	; Start of table.
		=7878	;
284F	4C4F4144 04	=7879 DB	'LOAD',EOT ;
2854	5520	=7880 DW	LOADRO ; Load PDP file routine.
		=7881	;
2856	53415645 04	=7882 DB	'SAVE',EOT ;
285B	4C20	=7883 DW	SAVERO ; Save PDP file routine.
		=7884	;
285D	44454604	=7885 DB	'DEF',EOT ;
2861	2527	=7886 DW	DEFROU ; Define key routine
		=7887	;
2863	4C454E04	=7888 DB	'LEN',EOT ;
2867	6616	=7889 DW	LENROU ; Length of file routine.
		=7890	;
2869	425204	=7891 DB	'BR',EOT ;
286C	D922	=7892 DW	BRROU ; Beginning of range routine.
		=7893	;
286E	424C04	=7894 DB	'BL',EOT ;
2871	CA22	=7895 DW	BLROU ; Beginning of line routine.
		=7896	;
2873	455204	=7897 DB	'ER',EOT ;
2876	1423	=7898 DW	ERROU ; End of range routine.
		=7899	;
2878	454C04	=7900 DB	'EL',EOT ;
287B	0323	=7901 DW	ELROU ; End of line routine.
		=7902	;
287D	45584954 04	=7903 DB	'EXIT',EOT ;
2882	5C16	=7904 DW	EXITRO ; Exit routine.
		=7905	;
2884	45584904	=7906 DB	'EXI',EOT ;
2888	5C16	=7907 DW	EXITRO ; Second form of "EXIT"
		=7908	;
288A	455804	=7909 DB	'EX',EOT ;
288D	5C16	=7910 DW	EXITRO ; Third form of "EXIT"
		=7911	;
288F	4E455704	=7912 DB	'NEW',EOT ;
2893	CA16	=7913 DW	NEWROU ; NEW routine.
		=7914	;
2895	44454C04	=7915 DB	'DEL',EOT ;
2899	DC1B	=7916 DW	DTEOLR ; Delete to End_of_line
		=7917	;
289B	50415354 4504	=7918 DB	'PASTE',EOT ;
28A1	1C1F	=7919 DW	PASTER ; Paste routine.
		=7920	;
28A3	4355544C 454E04	=7921 DB	'CUTLEN',EOT. ;
28AA	EC1E	=7922 DW	CUTLRO ; Cut buffer len routine.
		=7923	;
28AC	56455204	=7924 DB	'VER',EOT ;
28B0	C316	=7925 DW	VERROU ; Version routine.
		=7926	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 139

L0C	OBJ	LINE	SOURCE STATEMENT
28B2	48454C50	=7927	DB 'HELP',EOT ;
	04		
28B7	BA28	=7928	DW HELPRO ; Help routine.
		=7929	;
28B9	FF	=7930	DB ETABLE ; End Of Table Marker.
		=7931 ;	
		=7932 ;	End of Editor tables. (EDTTABLE.A85).
		7933 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 140

LOC	OBJ	LINE	SOURCE STATEMENT
		7934	\$INCLUDE(EDTHELP.A85)
		=7935	;
		=7936	;
		=7937	EDTHELP module for the UCT SABus editor.
		=7938	;
		=7939	;
		=7940	This module contains (in other) the following subroutine:
		=7941	;
		=7942	a) HELPRO - Displayes a summy of the commands.
		=7943	;
		=7944	;
		=7945	;
		=7946	;
		=7947	;
		=7948	* Function : HELPRO (Help routine). *
		=7949	* Inputs : None. *
		=7950	* Outputs : None. *
		=7951	* Destroys : All. *
		=7952	* Calls : NEWSN, CLRS, QINIT, KEYIN, STRING and HOME. *
		=7953	;
		=7954	* Description : This routine clears the screen and displays *
		=7955	* a summary of the editor commands. It the reinitialises the *
		=7956	* input queue and fetches a key from the keyboard. The *
		=7957	* reason why the queue is reset it that if there were any *
		=7958	* characters in the queue they would cause the editor to *
		=7959	* exit from the HELP routine and redisplay the text. *
		=7960	;
		=7961	;
		=7962	;
		=7963	HELPRO: ; Start of help routine
28BA	CDCA26	=7964	CALL HOME ; Home the cursor
28BD	CDAE26	=7965	CALL CLRS ; Clear the rest of the screen
28C0	21D028	=7966	LXI H,HELMES ;
28C3	CDE125	=7967	CALL STRING ; Print the help message
28C6	CDEA12	=7968	CALL QINIT ; Kill the input queue
28C9	CDBE23	=7969	CALL KEYIN ; Fetch a key from the keyboard
28CC	CD7F12	=7970	CALL NEWSN ; Reinitialise the screen
28CF	C9	=7971	RET ; Return from the editor help routine
		=7972	;
		=7973	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 141

LOC	OBJ	LINE	SOURCE STATEMENT
		=7974 ;	-----
		=7975 ;	
		=7976 ;	Editor help message.
		=7977 ;	
		=7978 HELPMES:	
28D0	0A4C4F41	=7979	DB LF,'LOAD/SAVE <filename>'
	442F5341		
	5645203C		
	66696C65		
	6E616D65		
	3E		
28E5	0D0A4445	=7980	DB CR,LF,'DEFn (Define function key n)'
	466E2028		
	44656669		
	6E652066		
	756E6374		
	696F6E20		
	6B657920		
	6E29		
2903	0D0A4C45	=7981	DB CR,LF,'LEN (Display file length)'
	4E202844		
	6973706C		
	61792066		
	696C6520		
	6C656E67		
	746829		
291E	0D0A4252	=7982	DB CR,LF,'BR, ER, BL and EL'
	2C204552		
	2C20424C		
	20616E64		
	20454C		
2931	0D0A4558	=7983	DB CR,LF,'EXIT'
	4954		
2937	0D0A4E45	=7984	DB CR,LF,'NEW'
	57		
293C	0D0A4445	=7985	DB CR,LF,'DEL (Delete to end of line)'
	4C202844		
	656C6574		
	6520746F		
	20656E64		
	206F6620		
	6C696E65		
	29		
2959	0D0A4355	=7986	DB CR,LF,'CUT[n]L (Copy n lines into the CUT buffer)'
	545B6E5D		
	4C202843		
	6F707920		
	6E206C69		
	6E657320		
	696E746F		
	20746865		
	20435554		
	20627566		
	66657229		
2985	0D0A5041	=7987	DB CR,LF,'PASTE (Copy the CUT buffer into the text)'
	53544520		
	28436F70		
	79207468		
	65204355		
	54206275		
	66666572		
	20696E74		
	6F207468		
	65207465		
	787429		
		=7988	
29B0	0D0A5645	=7989	DB CR,LF,'VER (Display version information)'
	52202844		
	6973706C		
	61792076		

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 142

LOC	OBJ	LINE	SOURCE STATEMENT
	65727369		
	6F6E2069		
	6E666F72		
	6D617469		
	6F6E29		
29D3	0D0A4355	=7990	DB CR,LF,'CUTLEN (Display CUT buffer length)'
	544C454E		
	20284469		
	73706C61		
	79204355		
	54206275		
	66666572		
	206C656E		
	67746829		
29F7	0D0A463C	=7991	DB CR,LF,'F<Delimiter><Search_str><Delimiter>'
	44656C69		
	6D697465		
	723E3C53		
	65617263		
	685F7374		
	723E3C44		
	656C696D		
	69746572		
	3E		
2A1C	0D0A533C	=7992	DB CR,LF,'S<Delimiter><Search_str><Delimiter><Replacement_str><Delimiter>[A]'
	44656C69		
	6D697465		
	723E3C53		
	65617263		
	685F7374		
	723E3C44		
	656C696D		
	69746572		
	3E3C5265		
	706C6163		
	656D656E		
	745F7374		
	723E3C44		
	656C696D		
	69746572		
	3E5B415D		
2A60	0D0A273C	=7993	DB CR,LF,'''<Search_str>'
	53656172		
	63685F73		
	74723E		
2A6F	0D0A445B	=7994	DB CR,LF,'D[n]C/D[n]L (Delete n characters/lines)'
	6E5D432F		
	445B6E5D		
	4C202844		
	656C6574		
	65206E20		
	63686172		
	61637465		
	72732F6C		
	696E6573		
	29		
2A98	0D0A5B2D	=7995	DB CR,LF,'[-][n]L (Move cursor n lines up/down)'
	5D5B6E5D		
	4C20284D		
	6F766520		
	63757273		
	6F72206E		
	206C696E		
	65732075		
	702F646F		
	776E29		
2ABF	0D0A4920	=7996	DB CR,LF,'I (Insert mode)'
	28496E73		
	65727420		
	6D6F6465		

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 143

LOC	OBJ	LINE	SOURCE STATEMENT
	29		
2AD0	0D0A3C45	=7997	DB CR,LF,'<ESC>n (execute key definition n)'
	53433E6E		
	20286578		
	65637574		
	65206B65		
	79206465		
	66696E69		
	74696F6E		
	206E29		
		=7998	
2AF3	0D0A5E5A	=7999	DB CR,LF,'^Z (Goto next mode)'
	2028476F		
	746F206E		
	65787420		
	6D6F6465		
	29		
2B08	0D0A0A09	=8000	DB CR,LF,LF,TAB,TAB,'<Hit any key to continue>'
	093C4869		
	7420616E		
	79206B65		
	7920746F		
	20636F6E		
	74696E75		
	653E		
2B26	04	=8001	DB EOT
		=8002 ;	
		=8003 ;	End of <EDTHELP.A85> module.
		8004 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 144

LOC	OBJ	LINE	SOURCE STATEMENT
		8005	\$INCLUDE(EDTMES.A85)
		=8006	=====;
		=8007	;
		=8008	Message data used by the UCT SABus editor.
		=8009	;
		=8010	;
		=8011	NOTE: All messages are terminated by the special end of text character
		=8012	- <EOT>. This can be changed to use the sign bit as a last
		=8013	character indicator by simply changing this message file and the
		=8014	message output routine, STRING, in the EDTIO module.
		=8015	;
		=8016	=====;
		=8017	;
2B27	5B456E64 206F6620 42756666 65725D04	=8018	EOBMES: DB '[End of Buffer]',EOT
		=8019	;
		=8020	Error message control string (Printed just before an error message).
		=8021	;
2B37	1B593720	=8022	ERRCCS: DB ESC, GOTOC, 23+OFFSET, 0+OFFSET ; Move cursor to the start
		=8023	; of the last line.
2B3B	1B4B	=8024	DB ESC,CLRLC ; Clear line.
2B3D	04	=8025	DB EOT ; End of message
		=8026	;
2B3E	07496E76 616C6964 20436F6D 6D616E64 04	=8027	IVCMES: DB Bell , 'Invalid Command',EOT
2B4F	074E6F20 73747269 6E672069 6E206669 6E642062 75666665 722E04	=8028	NOSTRM: DB Bell , 'No string in find buffer.',EOT
		=8029	;
2B6A	44656C65 74652066 696C6520 696E206D 656D6F72 79202859 2F4E2920 3F3E04	=8030	CONFRM: DB 'Delete file in memory (Y/N) ?>',EOT
		=8031	;
2B89	46696C65 20757365 732004	=8032	LM1: DB 'File uses ',EOT
2B94	20627974 65732C20 04	=8033	LM2: DB ' bytes, ',EOT
2B9D	2072656D 61696E69 6E672C20 696E2004	=8034	LM3: DB ' remaining, in ',EOT
2BAD	206C696E 65732E04	=8035	LM4: DB ' lines.',EOT
		=8036	;
2BB5	074D656D 6F727920 66756C6C 04	=8037	MEMFUL: DB Bell, 'Memory full',EOT
		=8038	;
2BC2	5365636F 6E642064 656C696D 69746572 206E6F74	=8039	STRER1: DB 'Second delimiter not found.',EOT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 145

LOC	OBJ	LINE	SOURCE STATEMENT
	20666F75		
	6E642E04		
2BDE	074E6F20	=8040 NOSUCH: DB	Bell, 'No such string in file.',EOT
	73756368		
	20737472		
	696E6720		
	696E2066		
	696C652E		
	04		
		=8041 ;	
2BF7	07	=8042 DERRM1: DB	Bell
2BF8	53796E74	=8043 DB	'Syntax error, formats = "DnnnnL"'
	61782065		
	72726F72		
	2C20666F		
	726D6174		
	73203D20		
	22446E6E		
	6E6E4C22		
2C18	2C202244	=8044 DB	', "DnnC", "DL", "DC" and "DEL".'
	6E6E4322		
	2C202244		
	4C222C20		
	22444322		
	20616E64		
	20224445		
	4C222E		
2C37	04	=8045 DB	EOT
2C38	074E756D	=8046 DERRM2: DB	Bell,'Number too large.',EOT
	62657220		
	746F6F20		
	6C617267		
	652E04		
		=8047 ;	
		=8048 SUBEM1:	
2C4B	0753796E	=8049 SUBEM2: DB	Bell, 'Syntax error, format = "S/String1/String2/"',EOT
	74617820		
	6572726F		
	722C2066		
	6F726D61		
	74203D20		
	22532F53		
	7472696E		
	67312F53		
	7472696E		
	67322F22		
	2E04		
2C79	4E6F2073	=8050 SUBEM4: DB	'No string in substitute buffer.',EOT
	7472696E		
	6720696E		
	20737562		
	73697475		
	74652062		
	75666665		
	722E04		
2C98	07537472	=8051 SUBEM5: DB	Bell,'String not found.',EOT
	696E6720		
	6E6F7420		
	666F756E		
	642E04		
2CAB	53756273	=8052 SUBEM6: DB	'Substitute error, zero length 1st string.',EOT
	69747574		
	65206572		
	726F722C		
	207A6572		
	6F206C65		
	6E677468		
	20317374		
	20737472		
	696E672E		

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 146

LOC	OBJ	LINE	SOURCE STATEMENT
	04		
2CD4	556E6B6E	=8053 SUBEM7: DB	'Unknown substitution option.',EOT
	6F776E20		
	73756273		
	69747574		
	696F6E20		
	6F707469		
	6F6E2E04		
2CF0	074D656D	=8054 SUBM8a: DB	Bell, 'Memory full: Substitutions could not be made,'
	6F727920		
	66756C6C		
	3A205375		
	62737469		
	74757469		
	6F6E7320		
	636F756C		
	64206E6F		
	74206265		
	206D6164		
	652C		
2D1E	203C5072	=8055 DB	' <Press any key to continue>',EOT
	65737320		
	616E7920		
	6B657920		
	746F2063		
	6F6E7469		
	6E75653E		
	04		
2D3B	074D656D	=8056 SUBM8b: DB	Bell, 'Memory full: Substitution could not be made.',EOT
	6F727920		
	66756C6C		
	3A205375		
	62737469		
	74757469		
	6F6E2063		
	6F756C64		
	206E6F74		
	20626520		
	6D616465		
	2E04		
		=8057 ;	
2D69	20537562	=8058 SMMMSG: DB	' Substitutions made.',EOT
	73697475		
	74696F6E		
	73206D61		
	64652E04		
		=8059 ;	
		=8060 ;	CUT/PASTE error messages.
		=8061 ;	
2D7D	07435554	=8062 CUTEM1: DB	Bell, 'CUT syntax error, format = "CUTnnnL" or "CUTL"',EOT
	2073796E		
	74617820		
	6572726F		
	722C2066		
	6F726D61		
	74203D20		
	22435554		
	6E6E6E4C		
	22206F72		
	20224355		
	544C2204		
2DAD	07435554	=8063 CUTEM2: DB	Bell, 'CUT error, insufficient memory to perform CUT',EOT
	20657272		
	6F722C20		
	696E7375		
	66666963		
	69656E74		
	206D656D		
	6F727920		
	746F2070		

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 147

LOC	OBJ	LINE	SOURCE STATEMENT
	6572666F		
	726D2043		
	555404		
		=8064 ;	
2DDC	07506173	=8065 PASEM1: DB	Bell,'Paste buffer empty',EOT
	74652062		
	75666665		
	7220656D		
	70747904		
2DF0	07504153	=8066 PASEM2: DB	Bell,'PASTE error, insufficient memory to perform PASTE',EOT
	54452065		
	72726F72		
	2C20696E		
	73756666		
	69636965		
	6E74206D		
	656D6F72		
	7920746F		
	20706572		
	666F726D		
	20504153		
	544504		
		=8067 ;	
2E23	54686520	=8068 CUTLM1: DB	'The cut buffer occupies ',EOT
	63757420		
	62756666		
	6572206F		
	63637570		
	69657320		
	04		
2E3C	20627974	=8069 CUTLM2: DB	' bytes in ',EOT
	65732069		
	6E2004		
2E47	206C696E	=8070 CUTLM3: DB	' lines.',EOT
	65732E04		
		=8071 ;	
		=8072 ;	Version message.
		=8073 ;	
2E4F	0D534142	=8074 VERMSG: DB	CR,'SABus editor V',VERSION+'0' ,'. '
	75732065		
	6469746F		
	72205631		
	2E		
2E60	313A	=8075 DB	REVISION+'0',': '
2E62	30	=8076 DB	RELEASE+'0'
2E63	04	=8077 DB	EOT
		=8078 ;	
		=8079 ;	Define key messages.
		=8080 ;	
2E64	456E7465	=8081 DEFMES: DB	'Enter delimited definition:',EOT
	72206465		
	6C696D69		
	74656420		
	64656669		
	6E697469		
	6F6E3A04		
2E80	07446566	=8082 DEFERR: DB	Bell,'Define key, syntax error',EOT
	696E6520		
	6B65792C		
	2073796E		
	74617820		
	6572726F		
	7204		
2E9A	07556E64	=8083 UNDEFK: DB	Bell,'Undefined key',EOT
	6566696E		
	6564206B		
	657904		
		=8084 ;	
		=8085 ;	FILE_I/O Messages.
		=8086 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 148

LOC	OBJ	LINE	SOURCE STATEMENT
2EA9	4B495420 04	=8087 PDP COM: DB	'KIT ',EOT ; KIT command
		=8088 ;	
2EAE	070D4B49 54202D2D 20457272 6F72206E 756D6265 722004	=8089 PDPEM: DB	Bell,CR,'KIT -- Error number ',EOT
2EC5	070D4669 6C652064 6F657320 6E6F7420 65786973 742E04	=8090 FDNOTE: DB	Bell,CR,'File does not exist.',EOT
2EDC	070D496E 76616C69 64206669 6C65206E 616D652E 04	=8091 INV FNM: DB	Bell,CR,'Invalid file name.',EOT
		=8092 ;	
2EF1	0746696C 65206C6F 61646564 2E04	=8093 FLMESS: DB	Bell, 'File loaded.',EOT
2EFF	0746696C 65207361 7665642E 04	=8094 FSMESS: DB	Bell, 'File saved.',EOT
		=8095 ;	
2F0C	0D4E6F20 66696C65 6E616D65 20676976 656E04	=8096 NFNAMM: DB	CR,'No filename given',EOT
2F1F	0D557365 202204	=8097 QFILN1: DB	CR,'Use "',EOT
2F26	22206173 20746865 2066696C 65206E61 6D652028 592F4E29 203F3E04	=8098 QFILN2: DB	'" as the file name (Y/N) ?>',EOT
		=8099 ;	
2F42	070D4C69 6E6B2065 72726F72 2C20696E 76616C69 64206563 686F3A	=8100 LNKEM: DB	Bell,CR,'Link error, invalid echo:'
2F5D	20757365 72206E6F 74206C6F 67676564 20696E2F 6F746865 72207461 736B2072 756E6E69 6E672E04	=8101 DB	' user not logged in/other task running.',EOT
2F85	070D4C69 6E6B2074 696D656F 75742065 72726F72 2E04	=8102 TIMOEM: DB	Bell,CR,'Link timeout error.',EOT;
2F9B	070D4D65	=8103 RFILMF: DB	Bell,CR,'Memory full: The entire file could not be loaded.'

LOC	OBJ	LINE	SOURCE STATEMENT
	6D6F7279		
	2066756C		
	6C3A2054		
	68652065		
	6E746972		
	65206669		
	6C652063		
	6F756C64		
	206E6F74		
	20626520		
	6C6F6164		
	65642E		
2FCE	203C5072	=8104	DB ' <Press any key to continue>',EOT
	65737320		
	616E7920		
	6B657920		
	746F2063		
	6F6E7469		
	6E75653E		
	04		
		=8105 ;	
		=8106 ;	End of message data. (EDTMES.A85)
		8107 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 150

LOC	OBJ	LINE	SOURCE STATEMENT
		8108	\$INCLUDE(EDTDATA.A85)
		=8109	\$LIST
		=8110	=====
		=8111	;
		=8112	;
			Data area (Used to store variables.)
		=8113	;
		=8114	=====
		=8115	;
E800		=8116	ORG DATA
		=8117	;
		=8118	-----
		=8119	;
		=8120	a : Memory pointers:
		=8121	;
E800		=8122	POINT: DS WORD ; Pointer to the 1st char in the current line.
E802		=8123	CHAR: DS WORD ; Pointer to character at cursor.
		=8124	;
		=8125	-----
		=8126	;
		=8127	b : Line and column values:
		=8128	;
E804		=8129	COL: DS BYTE ; Current Column position.
E805		=8130	LINE: DS BYTE ; Current line position.
		=8131	;
E806		=8132	OCOL: DS BYTE ; Saved column position.
E807		=8133	OLINE: DS BYTE ; Saved line position.
		=8134	;
		=8135	-----
		=8136	;
		=8137	c : Counters:
		=8138	;
E808		=8139	EOBLIN: DS BYTE ; Position of the EOB on the screen.
		=8140	; EOBLIN=LSTLIN+1 means that it is off the
		=8141	; screen.
E809		=8142	NEWSF: DS BYTE ; New_screen_flag.
		=8143	;
E80A		=8144	RELCH: DS BYTE ; Relative character position within a line.
E80B		=8145	NULLS: DS BYTE ; Null counter (When this reaches a given value
		=8146	; the file is automatically packed.
		=8147	;
E80C		=8148	TPADL: DS WORD ; Temp variable used by pad routine.
		=8149	; Temp_PAD_length.
		=8150	;
		=8151	-----
		=8152	;
		=8153	d : Flags:
		=8154	;
E80E		=8155	LCLINE: DS BYTE ; FLAG : Lowest correct line.
E80F		=8156	ESCFLG: DS BYTE ; FLAG : Is this command part of an ESC seq.
E810		=8157	PZRFLG: DS BYTE ; FLAG : Used by 'PRINTD' to suppress zeros
E811		=8158	RESFLG: DS BYTE ; FLAG : Restore cursor after command ?
E812		=8159	SUBBIT: DS BYTE ; FLAG : SUB_Buffers_In_Tact.
E813		=8160	NOSEND: DS BYTE ; FLAG : Flow control flag. True=^S received.
E814		=8161	SUBREP: DS BYTE ; FLAG : Repeat substitution flag.
		=8162	;
		=8163	-----
		=8164	;
		=8165	e : Mode flags:
		=8166	;
		=8167	INSFLG = False and XFLG = False Command mode.
		=8168	INSFLG = True and XFLG = False Insert mode.
		=8169	INSFLG = False and XFLG = True Exchange mode.
		=8170	;
		=8171	INSFLG = True and XFLG = True *** Invalid ***
		=8172	;
E815		=8173	INSFLG: DS BYTE ; FLAG : Insert mode (ON/OFF).
E816		=8174	XFLG: DS BYTE ; FLAG : Exchange mode (ON/OFF).
		=8175	;
		=8176	-----
		=8177	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 151

LOC	OBJ	LINE	SOURCE STATEMENT
		=8178 ;	f : Buffers:
		=8179 ;	
E817		=8180 BUF: DS	STRLEN ; Command line buffer.
E867		=8181 TCOMBU: DS	MAXCL ; Temp COMmand Buffer.
		=8182 ;	
E871		=8183 DIGBUF: DS	DBLEN ; Read number routine digit buffer.
		=8184 ;	
		=8185 ;	-----
		=8186 ;	
		=8187 ;	g : Keyboard buffer structure.
		=8188 ;	
E875		=8189 KEYBUF: DS	KBLLEN ; Keyboard buffer.
		=8190 ;	
E8ED		=8191 UP: DS	BYTE ; Upper pointer within the keyboard buffer.
E8EE		=8192 LO: DS	BYTE ; Lower pointer within the keyboard buffer.
		=8193 ;	
		=8194 ;	-----
		=8195 ;	
		=8196 ;	h : CUT/PASTE data area.
		=8197 ;	
E8EF		=8198 CUTBUF: DS	BYTE ; FLAG : CUT buffer full flag.
E8F0		=8199 MAXTXT: DS	WORD ; Top of possible text area.
E8F2		=8200 CUTLIB: DS	WORD ; Cut buffer lines in buffer.
E8F4		=8201 CUTBL: DS	WORD ; Cut buffer length.
		=8202 ;	
		=8203 ;	-----
		=8204 ;	
		=8205 ;	i : File_I/O data area.
		=8206 ;	
E8F6		=8207 COM: DS	BYTE ; Stores the original command .
		=8208 ;	
		=8209 ;	-----
		=8210 ;	
		=8211 ;	j : Key definition data area
		=8212 ;	
		=8213 ;	Each key record looks like this.
		=8214 ;	
		=8215 ;	VECTOR: DS WORD ; Start address of the character string
		=8216 ;	length: DS BYTE ; length of the character string
		=8217 ;	
		=8218 ;	(If the length byte = zero then key not defined)
		=8219 ;	
E8F7		=8220 DEFTAB: DS	10*(WORD+BYTE) ; Leave space for 10 keys.
E915		=8221 KEYNUM: DS	1 ; Temp byte to store keynumber.
E916		=8222 KDTAB: DS	10*80 ; Allow 80 bytes per key.
		=8223 ;	
		=8224 ;	-----
		=8225 ;	
		=8226 ;	k : Find and replace buffers and string lengths.
		=8227 ;	
EC36		=8228 SEABUF: DS	30 ; Search buffer.
EC54		=8229 SEASL: DS	BYTE ; Search buffer string length.
EC55		=8230 REPBUF: DS	30 ; Replacement buffer.
EC73		=8231 REPSL: DS	BYTE ; Replacement buffer string length.
		=8232 ;	
EC74		=8233 NFINDS: DS	WORD ; Counter for the number of substitutions make.
		=8234 ;	
		=8235 ;	-----
		=8236 ;	
		=8237 ;	l : Editor stack area.
		=8238 ;	
EC76		=8239	DS STKLEN ; Reserve stack area.
		=8240	;
		=8241 NEWSP:	;
		=8242 ;	Initial new stack pointer at the end of the
		=8243 ;	stack area. (Pre-decrement stack system)
		=8244 ;	-----
		=8245 ;	
		=8246 ;	There is 38. bytes of free space in the monitor data area
		=8247 ;	starting at location 0EFC9H.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT

PAGE 152

LOC	OBJ	LINE	SOURCE STATEMENT
		=8248 ;	
EFC9		=8249	ORG OEFC9H
		=8250 ;	
EFC9		=8251 SOFN: DS 30	; Pdp file name.
EFE7		=8252 FNCRC: DS BYTE	; File name CRC.
		=8253 ;	
EFFC		=8254	ORG OEFFCH
		=8255 ;	
EFFC		=8256 SPOINT: DS WORD	; Start of text pointer.
EFFE		=8257 EPOINT: DS WORD	; End of text pointer.
		=8258 ;	
		=8259 ;	End of Data area. (EDTDATA.A85)
		8260 ;	
1000		8261	END START

0 error(s) detected

User symbols

ACK	0040 A	ALINE	1C6F A	ANLINE	2202 A	ASCBIN	F031 A	ASCII	007F A	BACKL	2335 A	BACKSF	2249 A
BBUF	117F A	BBUFLP	1182 A	BEGINL	1DF0 A	BELL	0007 A	BEMPTY	23E0 A	BLINE	269D A	BLKWS	1575 A
BLROU	22CA A	BNKLOP	15D1 A	BOTLIN	0017 A	BREAK	F025 A	BRROU	22D9 A	BRROU5	22E7 A	BSPACE	0008 A
BUF	E817 A	BYTE	0001 A	CCEXIT	11C4 A	CCMEX	1497 A	CCMOV1	147F A	CCMOV2	148E A	CCMOV4	1493 A
CCMOVE	147A A	CCMOVT	2830 A	CER	116E A	CFNLOP	2104 A	CHAR	E802 A	CHARF	162B A	CHKFN	2230 A
CHKFN2	223F A	CI	F013 A	CITEST	F00A A	CLRL	26A8 A	CLRLC	004B A	CLRS	26AE A	CLRSC	004A A
CMTAB	2823 A	CO	F01C A	COL	E804 A	COLSET	152A A	COM	E8F6 A	COMERR	1161 A	COMLOP	1006 A
COMM	108F A	COMTAB	2839 A	CONCAP	11B8 A	CONFRM	2B6A A	CONIN	F013 A	CONOUT	F01C A	CONDO	0003 A
CONTES	F00A A	CONTLE	0004 A	CONTRI	0006 A	CONTRO	001F A	CONTUP	0012 A	COPY	118B A	CPYEX	1186 A
CR	000D A	CRCC05	213E A	CRCCCL	212E A	CTRLC	0003 A	CTRLK	000B A	CTRLQ	0011 A	CTRLR	0012 A
CTRLS	0013 A	CTRLU	0015 A	CTRLZ	001A A	CUT05	1E73 A	CUT10	1E83 A	CUT15	1E8B A	CUT20	1EAD A
CUT25	1EBD A	CUTBL	E8F4 A	CUTBL5	1F15 A	CUTBLE	1F1B A	CUTBUF	E8EF A	CUTEM1	2D7D A	CUTEM2	2DAD A
CUTER1	1ED9 A	CUTER2	1EDF A	CUTERR	1EE8 A	CUTLIB	E8F2 A	CUTLM1	2E23 A	CUTLM2	2E3C A	CUTLM3	2E47 A
CUTLRO	1EEC A	CUTNZ	1E63 A	CUTOK	1E4C A	CUTPAD	1591 A	CUTRET	1EEB A	CUTROU	1E01 A	CUTSE	1EE5 A
DATA	E800 A	DBLEN	0004 A	DCHARS	1A0E A	DCHR05	1A1E A	DCHR10	1A36 A	DCHR15	1A3F A	DCHR20	1A6F A
DCHR25	1A80 A	DCMP	F043 A	DCMPI	F046 A	DECMUL	25F1 A	DEF20	2759 A	DEF25	275F A	DEFEN	0002 A
DEFERR	2E80 A	DEFINE	26FE A	DEFMES	2E64 A	DEFRO2	270B A	DEFRO5	2719 A	DEFRO20	271F A	DEFROU	2725 A
DEFTAB	E8F7 A	DEL	007F A	DELLI	1B6B A	DELAY	2222 A	DELCRO	1A81 A	DELE02	1A8F A	DELE04	1AB1 A
DELE05	1AD6 A	DELEOB	1B8D A	DELEOL	1B77 A	DELER1	1B37 A	DELER2	1B3D A	DELETE	1A88 A	DELEX	1BC1 A
DELEX1	1B43 A	DELKEY	1FC0 A	DELLEX	1BDB A	DELNOK	1794 A	DELOOP	2227 A	DELPRO	1443 A	DELRET	1AE4 A
DELROU	1AE5 A	DEOLO5	1BDF A	DEOL10	1BEE A	DERRM1	2BF7 A	DERRM2	2C38 A	DFILE	1BCA A	DIGBUF	E871 A
DISMO	1CD6 A	DISMOV	1C81 A	DLINES	1B44 A	DOSUB	1846 A	DOWNC	0042 A	DOWNRO	1D36 A	DSUB05	186B A
DSUB10	1870 A	DSUB15	1884 A	DSUB20	188C A	DSUBEX	1894 A	DTEOLR	1BDC A	DWNEOB	1D68 A	DWNRET	1D91 A
ECHO	F034 A	EDTIE	122F A	EDTINI	11C5 A	ELROU	2303 A	ELROUE	2313 A	ENDCM	1450 A	ENDCOM	1174 A
ENDSAV	220B A	ENDST	25EF A	ENDTAB	1356 A	ENTRY2	1FD6 A	EOB05	12E8 A	EOBLIN	E808 A	EOBMES	2B27 A
EOBOK	1D18 A	EOFMAR	001A A	EOFN	2112 A	EOT	0004 A	EOTIFN	2127 A	EPLOOP	2499 A	EPOINT	EFFE A
EPTRFO	164B A	ER05	236E A	ERR	00FF A	ERRCCS	2B37 A	ERROU	2314 A	ERRSTR	2387 A	ESC	001B A
ESCFLG	E80F A	ESCSEQ	1060 A	ETABLE	00FF A	EXCOM	1152 A	EXPIO	20CB A	EXIT1	202E A	EXIT2	2034 A
EXITRO	165C A	EXPDP1	216B A	EXPDP2	2181 A	FALSE	0000 A	FBEGIN	234B A	FCONT	1759 A	FDCMP	1230 A
FDMOTE	2EC5 A	FETCHD	2760 A	FILEIO	205B A	FIND05	1720 A	FINDBE	1741 A	FINDER	175F A	FINDEX	1765 A
FINDRO	171A A	FIO99	20C8 A	FLMESS	2EF1 A	FLOK	20E1 A	FNCRC	EFE7 A	FNULLS	23EB A	FSAV05	21FE A
FSMESS	2EFF A	GETINH	F055 A	GET2NM	F058 A	GET3NM	F05B A	GETCM	F04F A	GETDIG	2651 A	GETECH	F02B A
GETHX	F052 A	GETKEY	2393 A	GETPDP	F016 A	GETUC	F028 A	GETWD	F02E A	GKEYOK	23B4 A	GOB05	1245 A
GOBACK	1236 A	GOTKEY	23EE A	GOTO	26B4 A	GOTOC	0059 A	GOTOCM	13BF A	GOTOIM	13CA A	GOTOXM	13B2 A
GTSPC	14FB A	HELPME	28D0 A	HELPRO	28BA A	HOME	26CA A	HOMEC	0048 A	INCLX	1266 A	INCLIN	125A A
INCNO5	1C10 A	INCNUI	1C04 A	INCP05	23BD A	INCPTR	23B5 A	INEXIT	2048 A	INITDL	126C A	INITDT	1267 A
INSCR	2007 A	INSERT	1FB4 A	INSFLG	E815 A	INSNT	1FC7 A	INVFM	2EDC A	IPAD	157E A	IPADER	203C A
IPADLE	0014 A	IPROMP	003C A	IROU	1F98 A	IROU05	1FA9 A	IROU10	1FB3 A	IVCMES	2B3E A	K	0400 A
KBLEN	0078 A	KDTAB	E916 A	KEYBUF	E875 A	KEYEX	242A A	KEYIN	23BE A	KEYNUM	E915 A	KEYSTK	23F2 A
KILL05	278D A	KILLIC	2770 A	KILLC	2782 A	LASTDI	2677 A	LASTLF	1224 A	LC	12D2 A	LCLINE	E80E A
LEFTC	0044 A	LEFTRI	1DD7 A	LEFTRE	1E00 A	LEFTRO	1DCA A	LEN05	16A9 A	LEN10	16B0 A	LEN15	16B7 A
LENROU	1666 A	LF	000A A	LFEXIT	20EA A	LFLF05	189B A	LFLF10	18A6 A	LFLF15	18AB A	LFORLP	1896 A
LFS02	1786 A	LFS05	1789 A	LINE	E805 A	LINRER	1C38 A	LINREX	1C3E A	LINROU	1C27 A	LM1	2B89 A
LM2	2B94 A	LM3	2B9D A	LM4	2BAD A	LNKEM	2F42 A	LNKERR	20A8 A	LNOTC	12CD A	LO	E8EE A
LOADEN	0000 A	LOADF	20CC A	LOADRO	2055 A	LOOKFS	1773 A	LPNTAB	2445 A	LPNTAB	2445 A	LPRINT	242F A
LPRTEX	2468 A	LPRTOX	245E A	LPTAB	2435 A	LSTLIN	0016 A	MAKEC5	20FF A	MAKECO	20EE A	MASK	001F A
MAXCL	000A A	MAXCOL	004F A	MAXLEN	B000 A	MAXMOV	0028 A	MAXTXT	E8F0 A	MBEXIT	1CE4 A	MCEX	1505 A
MCTAB	14FF A	MEMER	15DF A	MEMFUL	2BB5 A	MFEXIT	1C8F A	MFNE	2153 A	MFNEX	2154 A	MINCOL	0000 A
MINIT	1276 A	MODCCH	001A A	MODCOL	14F0 A	MONDAT	EF00 A	MORTC	1196 A	MOVBO5	1CB0 A	MOVBI0	1CC0 A
MOVBI5	1CD0 A	MOVCU5	144A A	MOVCUR	142E A	MOVEBL	1C90 A	MOVENL	1C3F A	MOVL	26D8 A	MOVR	26DE A

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus editor

11:47:52 12-APR-88

EDT PAGE 153

LOC	OBJ	LINE	SOURCE STATEMENT
MOVSTR 178A A	NBPACK 000A A	NCUT1L 1E30 A	NDEL 1033 A
NEWREX 1713 A	NEWROU 16CA A	NEWSCN 127F A	NEWSF E809 A
NFINDS EC74 A	NFNAMM 2F0C A	NKEYYE 23C1 A	NLINE 26E4 A
NOFERR 20BA A	NOFILE 122D A	NOFILM 214B A	NOKEY 242D A
NOSEND E813 A	NOSTRM 2B4F A	NOSUCH 2BDE A	NOTIC 1B10 A
NOTCMC 242E A	NOTCON 2554 A	NOTCQ 2419 A	NOTCS 240C A
NOTE 1348 A	NOTEQ 1842 A	NOTER4 208C A	NOTES5 209A A
NOTJF 174A A	NOTLET 12FD A	NOTMIN 1114 A	NOTNUM 1315 A
NOTSC1 1142 A	NOTX 1554 A	NULL 0000 A	NULLF 236B A
NUPO5 1C23 A	NUP10 1C26 A	NXLINE 1C5F A	NXTCH 150F A
OK 0000 A	OKLET 1302 A	OKPIT 24CD A	OLINE E807 A
ONS15 12B5 A	ONS20 12BA A	ONSCN 1295 A	OUTD05 2605 A
OUTMSG F03D A	PACK 15E6 A	PACK05 15F3 A	PACK10 1613 A
PADLOP 15B9 A	PADLS 1595 A	PADRET 15E5 A	PAS05 1F4D A
PASER2 1F8A A	PASERR 1F8D A	PASRET 1F90 A	PASTER 1F1C A
PDELER 1B40 A	PDPOM 2E9A A	PDPEN 2EAE A	PLINE 2472 A
POSFOU 1C77 A	PRINT 249D A	PRINTD 25FA A	PROG 1000 A
PRVAL F037 A	PSPACE 1441 A	PSUBER 1A0A A	PTOEL 24D1 A
PUTK05 27AF A	PUTKDL 27C4 A	PUTKDX 27CC A	PWR10 263E A
QCHAR5 14A2 A	QCHARO 22B3 A	QCLINE 12BD A	QCLRET 12D4 A
QF10 17D1 A	QFCOMT 17DD A	QFERR 17C8 A	QFEXIT 17E0 A
QINIT 12EA A	QKILLC 000B A	QLET 12F3 A	QLETIX 1304 A
QSOB 1319 A	QSOB05 132A A	QTAB 132C A	QTABEX 1366 A
QUITRO 165C A	QUOTE 0027 A	RBULF 1369 A	RBULFE 1388 A
RDEF10 27E8 A	RDEF15 280C A	RDEF17 2819 A	RDEF20 2821 A
READS5 2520 A	REDOS 19C3 A	RELCH E80A A	RELEAS 0000 A
RESPLG E811 A	REVISI 0001 A	REVL 26ED A	REVLFC 0049 A
RFIL20 21C2 A	RFIL25 21C8 A	RFILE 218A A	RFILMF 2F9B A
RNEXTC 251D A	RNUM02 2668 A	RNUM05 266A A	RNUMEX 269B A
RULFL 1391 A	RUNB 20F0 A	SAVEEN 0001 A	SAVEF 21CA A
SCH10 1472 A	SCOMEX 2189 A	SCOML1 2158 A	SCOML2 216E A
SCP15 25AF A	SCP20 25BB A	SCPRES 259F A	SCTPDP 2155 A
SEAF5 17E1 A	SEARCH 145D A	SEARET 1827 A	SEASL EC54 A
SKIPNC 11A8 A	SMEXIT 13D4 A	SMMSG 2D69 A	SNCOM 18F8 A
SOFN EFC9 A	SOLFO 1255 A	SPACE 0020 A	SPDPWE 2287 A
START 1000 A	STK05 25DC A	STK10 25DF A	STKLEN 00C8 A
STRCMP 1828 A	STRCOM 284F A	STRDEL 255C A	STREND 2589 A
STRMER 17A5 A	STRRET 258E A	SUB05 18D9 A	SUB16 1F91 A
SUBCON 1944 A	SUBE51 19DB A	SUBE85 1A07 A	SUBEM1 2C4B A
SUBEM6 2CAB A	SUBEM7 2CD4 A	SUBER1 19C9 A	SUBER2 19CF A
SUBER7 19E7 A	SUBER8 19ED A	SUBEX 1A0D A	SUBM8A 2CF0 A
SWERET 22A4 A	TAB 0009 A	TCOMBU E867 A	TIMOEM 2F85 A
TOPOS 1D0C A	TPADL E80C A	TRANSP F049 A	TRUE FFFF A
UI1 F019 A	UNDEFK 2E9A A	UO0 F01F A	UO1 F022 A
UP10 13FC A	UPC 0041 A	UPDATE 13D5 A	UPEXIT 1D34 A
USART 0000 A	VALDL F04C A	VERMSG 2E4F A	VERROU 16C3 A
VERT15 14E8 A	VERT20 14EF A	VERTP 14BD A	WAITFA 2271 A
WFECO 228E A	WFOU5 22AB A	WORD 0002 A	XCHAR 1409 A
XPROMP 005E A			

APPENDIX G

ASM LISTING

The following pages contain an index of the symbols used in the assembler source listing. In order to keep the index reasonably short, the local symbols, those defined and referenced within a subroutine, are not listed.

As well as the symbol name and page reference, a symbol type is also given. The possible symbol types are:

- a) CONSTANT (in this case a value is also given)
- b) SUBROUTINE
- c) BYTE ARRAY (in this case the length is also given)
- d) WORD ARRAY (in this case the length, in bytes, is also given)
- e) STRUCTURE
- f) TABLE
- g) MESSAGE

Some of the symbol types may be preceded by the word ABSOLUTE. This simply means that they are stored at fixed addresses in the memory map and storage is not dynamically allocated in the assembler data area.

NOTE: The page reference given in the index below is the page on which the symbol is defined.

<u>SYMBOL</u>	<u>DESCRIPTION</u>	<u>PAGE REFERENCE</u>
AACK	CONSTANT, VALUE = 006H.....	G-10
AADD	SUBROUTINE.....	G-136
ABS	SUBROUTINE.....	G-136
ADDP	SUBROUTINE.....	G-28
ADDSYM	CONSTANT, VALUE = '+'.....	G-11
ADIV	SUBROUTINE.....	G-137
ADRINS	SUBROUTINE.....	G-28
AINIT	SUBROUTINE.....	G-130
AMULT	SUBROUTINE.....	G-137
ANDC	CONSTANT, VALUE = 03H.....	G-11
ANDROU	SUBROUTINE.....	G-138

ANEG	SUBROUTINE.....	G-138
APOP	SUBROUTINE.....	G-130
APUSH	SUBROUTINE.....	G-131
ARPS	TABLE.....	G-107
ASCSUB	CONSTANT, VALUE = 01AH.....	G-11
ASP	WORD VARIABLE.....	G-159
ASTACK	BYTE ARRAY, LENGTH = ASTKL.....	G-159
ASTKL	CONSTANT, VALUE = 15 * WORD.....	G-12
ASUB	SUBROUTINE.....	G-139
BASE	BYTE VARIABLE.....	G-159
BASEAD	TABLE.....	G-103
BBUF	SUBROUTINE.....	G-33
BCC	SUBROUTINE.....	G-78
BELL	CONSTANT, VALUE = 007H.....	G-10
BITSPB	CONSTANT, VALUE = 8.....	G-10
BITSPW	CONSTANT, VALUE = 16.....	G-10
BREAK	SUBROUTINE.....	G-78
BRIEFF	BYTE VARIABLE.....	G-158
BRIEFR	SUBROUTINE.....	G-53
BSPACE	CONSTANT, VALUE = 008H.....	G-10
BYTE	CONSTANT, VALUE = 1.....	G-10
BYTES	BYTE VARIABLE.....	G-160
CAN	CONSTANT, VALUE = 018H.....	G-11
CENT	BYTE VARIABLE.....	G-160
CHKSUM	BYTE VARIABLE.....	G-159
CLOSEFILE	SUBROUTINE.....	G-79
CMPLAB	SUBROUTINE.....	G-57
CMPOP	SUBROUTINE.....	G-68
CODBUF	BYTE ARRAY, LENGTH = 3.....	G-159
CODE	SUBROUTINE.....	G-111
CONBL	CONSTANT, VALUE = 10.....	G-12
CONCOP	SUBROUTINE.....	G-33
CONFLG	BYTE VARIABLE.....	G-160
CONHEX	SUBROUTINE.....	G-80
CONJT	TABLE.....	G-53
CONLET	SUBROUTINE.....	G-34
CONTAB	TABLE.....	G-53
CONTROL	SUBROUTINE.....	G-50
CONVHV	SUBROUTINE.....	G-57
CONVPO	SUBROUTINE.....	G-68
COPYL	SUBROUTINE.....	G-35
CR	CONSTANT, VALUE = 00DH.....	G-10
CRLF	SUBROUTINE.....	G-80
CSHIFT	SUBROUTINE.....	G-139
CTRLMASK	CONSTANT, VALUE = 0011111B.....	G-11
CTRLQ	CONSTANT, VALUE = CTRLMASK AND 'Q'.....	G-11
CTRLS	CONSTANT, VALUE = CTRLMASK AND 'S'.....	G-11
CTRLZ	CONSTANT, VALUE = CTRLMASK AND 'Z'.....	G-11
D16	CONSTANT, VALUE = 0001B.....	G-11
D16ROU	SUBROUTINE.....	G-104
D8	CONSTANT, VALUE = 0010B.....	G-11
D8ROU	SUBROUTINE.....	G-104
DBGENB	SUBROUTINE.....	G-41
DBLOCK	CONSTANT, VALUE = 0E800H.....	G-12
DBROU	SUBROUTINE.....	G-41

DC1	CONSTANT, VALUE = 011H.....	G-10
DC2	CONSTANT, VALUE = 012H.....	G-11
DC3	CONSTANT, VALUE = 013H.....	G-11
DC4	CONSTANT, VALUE = 014H.....	G-11
DCMP	SUBROUTINE.....	G-29
DEFLPP	CONSTANT, VALUE = 60.....	G-12
DEFPF	CONSTANT, VALUE = FALSE.....	G-12
DEL	CONSTANT, VALUE = 07FH.....	G-11
DINIT	SUBROUTINE.....	G-132
DIVC	CONSTANT, VALUE = 09H.....	G-11
DIVMOD	SUBROUTINE.....	G-140
DIVSYM	CONSTANT, VALUE = '/'.....	G-11
DLE	CONSTANT, VALUE = 010H.....	G-10
DPUSH	SUBROUTINE.....	G-132
DREG	CONSTANT, VALUE = 0100B.....	G-11
DREGRO	SUBROUTINE.....	G-105
DSP	WORD VARIABLE.....	G-159
DSROU	SUBROUTINE.....	G-44
DSTACK	BYTE ARRAY, LENGTH = DSTKL.....	G-159
DSTKL	CONSTANT, VALUE = 21.....	G-12
DWGENW	SUBROUTINE.....	G-44
DWROU	SUBROUTINE.....	G-45
EJECTR	SUBROUTINE.....	G-51
EM	CONSTANT, VALUE = 019H.....	G-11
ENDM1	MESSAGE.....	G-156
ENDM2	MESSAGE.....	G-156
ENDM3	MESSAGE.....	G-156
ENDROU	SUBROUTINE.....	G-46
ENQ	CONSTANT, VALUE = 005H.....	G-10
ENTS	BYTE VARIABLE.....	G-159
EOFMARK	CONSTANT, VALUE = CTRLZ.....	G-12
EOT	CONSTANT, VALUE = 004H.....	G-10
EPOINT	ABSOLUTE WORD VARIABLE.....	G-12
EQUFLG	BYTE VARIABLE.....	G-158
EQUROU	SUBROUTINE.....	G-46
EQUSTR	MESSAGE.....	G-155
ERMES	MESSAGE.....	G-156
ERRBUF	BYTE ARRAY, LENGTH = 10.....	G-158
ERRORP	WORD VARIABLE.....	G-158
ERRORS	WORD VARIABLE.....	G-158
ESC	CONSTANT, VALUE = 01BH.....	G-11
ETABLE	CONSTANT, VALUE = 0FFH.....	G-12
ETB	CONSTANT, VALUE = 017H.....	G-11
ETX	CONSTANT, VALUE = 003H.....	G-10
EXOPER	SUBROUTINE.....	G-126
FALSE	CONSTANT, VALUE = NOT TRUE.....	G-10
FATALF	BYTE VARIABLE.....	G-158
FATALN	CONSTANT, VALUE = 99.....	G-12
FATM1	MESSAGE.....	G-155
FATM2	MESSAGE.....	G-156
FATMSG	MESSAGE.....	G-156
FETCH	SUBROUTINE.....	G-58
FETCHN	SUBROUTINE.....	G-54
FF	CONSTANT, VALUE = 00CH.....	G-10
FILEOUT	SUBROUTINE.....	G-80

FNAME	BYTE ARRAY, LENGTH = 30.....	G-159
FPHEX	SUBROUTINE.....	G-81
FS	CONSTANT, VALUE = 01CH.....	G-11
GCFLAG	BYTE VARIABLE.....	G-158
GETNSC	SUBROUTINE.....	G-35
GETOPR	SUBROUTINE.....	G-103
GS	CONSTANT, VALUE = 01DH.....	G-11
HASH	SUBROUTINE.....	G-59
HASHV	BYTE VARIABLE.....	G-159
HEAD1	MESSAGE.....	G-156
HEAD2	MESSAGE.....	G-156
HEADER	SUBROUTINE.....	G-82
HIGHC	CONSTANT, VALUE = 0EH.....	G-11
HIGHRO	SUBROUTINE.....	G-141
HT	CONSTANT, VALUE = 009H.....	G-10
HVINC	BYTE VARIABLE.....	G-159
INSERT	SUBROUTINE.....	G-60
K	CONSTANT, VALUE = 1024.....	G-10
KITACK	CONSTANT, VALUE = '@'.....	G-12
KITIN	CONSTANT, VALUE = 0F013H.....	G-76
KITMSG	MESSAGE.....	G-156
KITOUT	CONSTANT, VALUE = 0F01CH.....	G-76
KITTEST	CONSTANT, VALUE = 0F00AH.....	G-76
LABBL	CONSTANT, VALUE = 6.....	G-12
LABELB	STRUCTURE.....	G-159
LABELE	CONSTANT, VALUE = 0FFH.....	G-12
LASTPC	WORD VARIABLE.....	G-160
LERRS	BYTE VARIABLE.....	G-158
LF	CONSTANT, VALUE = 00AH.....	G-10
LINEB	BYTE ARRAY, LENGTH = MAXBL.....	G-159
LINEBL	BYTE VARIABLE.....	G-159
LINEBP	WORD VARIABLE.....	G-159
LINEN	WORD VARIABLE.....	G-158
LISTRO	SUBROUTINE.....	G-50
LOWC	CONSTANT, VALUE = 0DH.....	G-11
LOWLIM	BYTE VARIABLE.....	G-160
LOWROU	SUBROUTINE.....	G-141
LPOINT	WORD VARIABLE.....	G-158
LPP	BYTE VARIABLE.....	G-158
LSTFIL	SUBROUTINE.....	G-51
M10	MESSAGE.....	G-152
M15	MESSAGE.....	G-152
M20	MESSAGE.....	G-153
M25	MESSAGE.....	G-153
M30	MESSAGE.....	G-154
M35	MESSAGE.....	G-154
M40	MESSAGE.....	G-155
M5	MESSAGE.....	G-151
M8085	CONSTANT, VALUE = TRUE.....	G-12
MAXBL	CONSTANT, VALUE = 132.....	G-12
MAXCOD	CONSTANT, VALUE = 0FFFFH.....	G-12
MAXPOL	CONSTANT, VALUE = 50.....	G-12
MESSAG	MESSAGE.....	G-151
MINCOD	CONSTANT, VALUE = 0000H.....	G-12
MINUSC	CONSTANT, VALUE = 06H.....	G-11

MODC	CONSTANT, VALUE = 0AH.....	G-11
MODROU	SUBROUTINE.....	G-142
MONDATA	CONSTANT, VALUE = 0EF00H.....	G-12
MOREH	BYTE VARIABLE.....	G-158
MSB	CONSTANT, VALUE = 10000000B.....	G-12
MULSYM	CONSTANT, VALUE = '*'.....	G-11
MULT	WORD VARIABLE.....	G-159
MULTC	CONSTANT, VALUE = 08H.....	G-11
NAK	CONSTANT, VALUE = 015H.....	G-11
NEGC	CONSTANT, VALUE = 07H.....	G-11
NEVERF	CONSTANT, VALUE = 080H.....	G-12
NEWPAG	BYTE VARIABLE.....	G-158
NEWSP	STRUCTURE.....	G-160
NEXTHV	SUBROUTINE.....	G-62
NEXTID	BYTE VARIABLE.....	G-160
NEXTPL	SUBROUTINE.....	G-82
NLSTRO	SUBROUTINE.....	G-51
NOBRRO	SUBROUTINE.....	G-53
NOENDM	MESSAGE.....	G-156
NOFILM	MESSAGE.....	G-156
NOOP	CONSTANT, VALUE = 0000B.....	G-11
NOPC	CONSTANT, VALUE = 00H.....	G-11
NOTC	CONSTANT, VALUE = 04H.....	G-11
NOTROU	SUBROUTINE.....	G-142
NSYMRO	SUBROUTINE.....	G-51
NUL	CONSTANT, VALUE = 000H.....	G-10
OBJBA	WORD VARIABLE.....	G-159
OBJBP	WORD VARIABLE.....	G-159
OBJBUF	BYTE ARRAY, LENGTH = OBJLEN.....	G-159
OBJC	BYTE VARIABLE.....	G-159
OBJFLG	BYTE VARIABLE.....	G-159
OBJFRO	SUBROUTINE.....	G-52
OBJINIT	SUBROUTINE.....	G-55
OBJLEN	CONSTANT, VALUE = 16.....	G-12
OBJNA	WORD VARIABLE.....	G-159
OCODBUF	SUBROUTINE.....	G-113
OENTS	BYTE VARIABLE.....	G-160
OFFMSB	CONSTANT, VALUE = 01111111B.....	G-12
OKGETV	SUBROUTINE.....	G-120
OPBUF	BYTE ARRAY, LENGTH = CONBL.....	G-160
OPBUFL	CONSTANT, VALUE = 6.....	G-12
OPCODE	BYTE VARIABLE.....	G-160
OPENFILE	SUBROUTINE.....	G-83
OPERE	CONSTANT, VALUE = 000H.....	G-12
OPERF	SUBROUTINE.....	G-121
OPERS	BYTE VARIABLE.....	G-160
OPOP	SUBROUTINE.....	G-132
OPTAB	TABLE.....	G-70
OPTABL	CONSTANT, VALUE = 81.....	G-73
OPUSH	SUBROUTINE.....	G-133
ORC	CONSTANT, VALUE = 01H.....	G-11
ORGROU	SUBROUTINE.....	G-48
ORROU	SUBROUTINE.....	G-143
OSP	WORD VARIABLE.....	G-159
OSTACK	BYTE ARRAY, LENGTH = OSTKL.....	G-159

OSTKL	CONSTANT, VALUE = 30.....	G-12
P1INIT	SUBROUTINE.....	G-16
P1LINE	SUBROUTINE.....	G-16
P2INIT	SUBROUTINE.....	G-20
P2LABL	SUBROUTINE.....	G-23
P2LINE	SUBROUTINE.....	G-21
P2POPR	SUBROUTINE.....	G-24
PADR	SUBROUTINE.....	G-91
PAERRS	SUBROUTINE.....	G-97
PAFLAG	BYTE VARIABLE.....	G-160
PAGLEN	SUBROUTINE.....	G-52
PAGNUM	WORD VARIABLE.....	G-158
PASDEF	BYTE VARIABLE.....	G-159
PASSN	BYTE VARIABLE.....	G-160
PC	WORD VARIABLE.....	G-160
PCDEF	BYTE VARIABLE.....	G-160
PCODEL	SUBROUTINE.....	G-91
PDPFLG	BYTE VARIABLE.....	G-158
PDPIN	CONSTANT, VALUE = 0F016H.....	G-76
PDPOUT	CONSTANT, VALUE = 0F01FH.....	G-76
PDPSTR	SUBROUTINE.....	G-84
PDPTEST	CONSTANT, VALUE = 0F00DH.....	G-76
PEQUL	SUBROUTINE.....	G-93
PERR	SUBROUTINE.....	G-97
PERROR	SUBROUTINE.....	G-98
PFLAG	BYTE VARIABLE.....	G-160
PHL	SUBROUTINE.....	G-85
PHOTAB	TABLE.....	G-75
PLINE	SUBROUTINE.....	G-94
PLINES	BYTE VARIABLE.....	G-158
PLUSC	CONSTANT, VALUE = 05H.....	G-11
PLUSOL	BYTE VARIABLE.....	G-159
PNERRS	SUBROUTINE.....	G-99
POINT	WORD VARIABLE.....	G-158
PRINT	SUBROUTINE.....	G-85
PRINTD	SUBROUTINE.....	G-100
PRINTH	SUBROUTINE.....	G-86
PRTCB	SUBROUTINE.....	G-94
PRTLAB	SUBROUTINE.....	G-64
PTEXT	SUBROUTINE.....	G-95
PWR10	TABLE.....	G-101
PZRFLG	BYTE VARIABLE.....	G-160
QCONTAB	SUBROUTINE.....	G-36
QEFIL	SUBROUTINE.....	G-29
QEQU	SUBROUTINE.....	G-29
QLET	SUBROUTINE.....	G-37
QLLET	SUBROUTINE.....	G-38
QNUM	SUBROUTINE.....	G-150
QNUML	SUBROUTINE.....	G-119
QOP	SUBROUTINE.....	G-69
QOPERS	SUBROUTINE.....	G-126
QPHOP	SUBROUTINE.....	G-74
QREG	SUBROUTINE.....	G-109
QSLOPS	SUBROUTINE.....	G-128
QTAB	SUBROUTINE.....	G-36

QTABSL	BYTE VARIABLE.....	G-160
QTABSP	SUBROUTINE.....	G-30
QUOTE	CONSTANT, VALUE = 027H.....	G-11
QUOTEF	SUBROUTINE.....	G-118
READNU	SUBROUTINE.....	G-148
REGS	TABLE.....	G-110
REXP	SUBROUTINE.....	G-115
RPARF	SUBROUTINE.....	G-123
RPAROU	SUBROUTINE.....	G-107
RPATH	CONSTANT, VALUE = 0110B.....	G-11
RPLIM	CONSTANT, VALUE = 0111B.....	G-11
RPLROU	SUBROUTINE.....	G-108
RPSROU	SUBROUTINE.....	G-106
RPSTK	CONSTANT, VALUE = 0101B.....	G-11
RS	CONSTANT, VALUE = 01EH.....	G-11
RSTN	CONSTANT, VALUE = 1000B.....	G-11
RSTNRO	SUBROUTINE.....	G-108
RULF	SUBROUTINE.....	G-30
RUNTOB	SUBROUTINE.....	G-30
SADR	WORD VARIABLE.....	G-160
SCMP	SUBROUTINE.....	G-65
SCODE	CONSTANT, VALUE = 01000H.....	G-12
SEMILF	SUBROUTINE.....	G-31
SEMROS	TABLE.....	G-126
SENDHR	SUBROUTINE.....	G-87
SENDLINE	SUBROUTINE.....	G-86
SHIFTL	SUBROUTINE.....	G-143
SHIFTR	SUBROUTINE.....	G-143
SHLC	CONSTANT, VALUE = 0CH.....	G-11
SHLROU	SUBROUTINE.....	G-144
SHRC	CONSTANT, VALUE = 0BH.....	G-11
SHRROU	SUBROUTINE.....	G-144
SI	CONSTANT, VALUE = 00FH.....	G-10
SIGN	BYTE VARIABLE.....	G-159
SO	CONSTANT, VALUE = 00EH.....	G-10
SOH	CONSTANT, VALUE = 001H.....	G-10
SP2MES	MESSAGE.....	G-155
SPACE	CONSTANT, VALUE = 020H.....	G-11
SPACES	SUBROUTINE.....	G-88
SPOINT	ABSOLUTE WORD VARIABLE.....	G-12
SREG	CONSTANT, VALUE = 0011B.....	G-11
SREGRO	SUBROUTINE.....	G-105
STABLE	ABSOLUTE STRUCTURE.....	G-12
STACKL	CONSTANT, VALUE = 70 * WORD.....	G-12
START	MAIN PROGRAM.....	G-13
STARTM	MESSAGE.....	G-155
STDMES	MESSAGE.....	G-155
STDUMP	SUBROUTINE.....	G-63
STKRPS	TABLE.....	G-106
STRING	SUBROUTINE.....	G-89
STROPS	TABLE.....	G-127
STX	CONSTANT, VALUE = 002H.....	G-10
SUBSYM	CONSTANT, VALUE = '-'.....	G-11
SWOPB	SUBROUTINE.....	G-39
SYMFLG	BYTE VARIABLE.....	G-160

SYMROU	SUBROUTINE.....	G-51
SYMSOL	BYTE VARIABLE.....	G-158
SYN	CONSTANT, VALUE = 016H.....	G-11
TAB	CONSTANT, VALUE = 009H.....	G-10
TABINT	SUBROUTINE.....	G-62
TD	BYTE VARIABLE.....	G-159
TEMPLB	BYTE ARRAY, LENGTH = LABBL.....	G-159
TERMHR	MESSAGE.....	G-156
TERMKIT	MESSAGE.....	G-156
TERMOUT	SUBROUTINE.....	G-89
TRUE	CONSTANT, VALUE = 0FFH.....	G-10
UD8	CONSTANT, VALUE = 1001B.....	G-11
UD8ROU	SUBROUTINE.....	G-109
UNDEF	CONSTANT, VALUE = 80H.....	G-10
UNFLOW	SUBROUTINE.....	G-131
UNSIGN	SUBROUTINE.....	G-145
UPLIM	BYTE VARIABLE.....	G-160
US	CONSTANT, VALUE = 01FH.....	G-11
VT	CONSTANT, VALUE = 00BH.....	G-10
WORD	CONSTANT, VALUE = 2.....	G-10
XORC	CONSTANT, VALUE = 02H.....	G-11
XORROU	SUBROUTINE.....	G-146


```

LOC   OBJ      LINE      SOURCE STATEMENT
1 $TITLE('UCT 8085 Assembler')
2 $PAGELENGTH(75)
3 ;
4 ;*****
5 ;*
6 ;*           8085 Assembler
7 ;*           -----
8 ;*
9 ;*           Language : 8085 Assembler
10 ;*          Author   : J.G.Jack
11 ;*          Date     : 1983
12 ;*
13 ;*           Main file : ASM.A85
14 ;*
15 ;*           The assembler also consists of the following INCLUDE files:
16 ;*
17 ;*           1) ASMEQU.A85           Assembler equates
18 ;*           2) ASMCHARF.A85        Character functions
19 ;*           3) ASMDIREC.A85        Directive routines
20 ;*           4) ASMCONT.A85         Assembler controls
21 ;*           5) ASMSTABLE.A85       Symbol table routines
22 ;*           6) ASMSTDMP.A85        Symbol table dump
23 ;*           7) ASMOPCODE.A85       Opcode lookup routine
24 ;*           8) ASMPCODE.A85        Directive lookup routine
25 ;*           9) ASMPLINE.A85        Line output routine
26 ;*          10) ASMPERR.A85          Error reporting routine
27 ;*          11) ASMIO.A85           Assembler I/O routines
28 ;*          12) ASMPRINTD.A85       Print number in decimal routine
29 ;*          13) ASMGETOPR.A85       Operand fetch routines
30 ;*          14) ASMCODE.A85         Code generation routine
31 ;*          15) ASMCALC.A85         Expression evaluation routine
32 ;*          16) ASMCALCO.A85        Exp eval operand lookup routine
33 ;*          17) ASMCALCS.A85        Exp eval stack routines
34 ;*          18) ASMCALCA.A85        Exp eval arithmetic routines
35 ;*          19) ASMREADNU.A85       Read number routine
36 ;*          20) ASMMMSG.A85         Assembler error message file
37 ;*          21) ASMDBLOCK.A85       Data block
38 ;*          22) ASMPASS1.A85        Pass 1 routine. (With init).
39 ;*          23) ASMPASS2.A85        Pass 2 routine. (With init).
40 ;*          24) ASMGEN.A85          General routines.
41 ;*          25) ASMBIO.A85          Basic I/O routines.
42 ;*
43 ;*****
44 ;
45           NAME      ASM
46 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT
		47	\$INCLUDE(ASMEQU.A85)
		= 48 ;	-----
		= 49 ;	
		= 50 ;	Equates Used by the UCT SABus assembler.
		= 51 ;	
		= 52 ;	The equates are divided into the following subsections:
		= 53 ;	
		= 54 ;	A : Logical equates.
		= 55 ;	B : Storage size equates.
		= 56 ;	C : Word and bytes sizes in bits.
		= 57 ;	D : ASCII constants.
		= 58 ;	E : Codes for Arithmetic operators.
		= 59 ;	F : Opcode types:
		= 60 ;	G : General Equates (Markers etc)
		= 61 ;	H : Data structure sizes:
		= 62 ;	H.1 : Buffer sizes.
		= 63 ;	H.2 : Stack sizes.
		= 64 ;	H.3 : Main assembler stack length.
		= 65 ;	I : Memory usage equates.
		= 66 ;	J : Defaults
		= 67 ;	
		= 68 ;	-----
		= 69 ;	
		= 70 ;	A : Logical equates.
		= 71 ;	
00FF		= 72 TRUE	EQU OFFH ; Logical TRUE.
FF00		= 73 FALSE	EQU NOT TRUE ; Logical FALSE.
0080		= 74 UNDEF	EQU 80H ; Undefined.
		= 75 ;	
		= 76 ;	-----
		= 77 ;	
		= 78 ;	B : Storage size Equates (In terms of bytes).
		= 79 ;	
0001		= 80 BYTE	EQU 1 ; 1 Byte = 1 Byte.
0002		= 81 WORD	EQU 2 ; 1 Word = 2 Bytes.
0400		= 82 K	EQU 1024 ; 1 Kbytes = 1024 bytes.
		= 83 ;	
		= 84 ;	-----
		= 85 ;	
		= 86 ;	C : Word and bytes sizes in bits.
		= 87 ;	
0010		= 88 BITSPW	EQU 16 ; Bits per word = 16.
0008		= 89 BITSPB	EQU 8 ; Bits per byte = 8.
		= 90 ;	
		= 91 ;	-----
		= 92 ;	
		= 93 ;	D : A S C I I c o n s t a n t s .
		= 94 ;	
		= 95 ;	Control characters:
		= 96 ;	
0000		= 97 NUL	EQU 000H ; ASCII null character
0001		= 98 SOH	EQU 001H ; ASCII Start of header character
0002		= 99 STX	EQU 002H ; ASCII Start of transmission character
0003		= 100 ETX	EQU 003H ; ASCII End of transmission character
0004		= 101 EOT	EQU 004H ; ASCII End of text marker.
0005		= 102 ENQ	EQU 005H ; ASCII Enquire character
0006		= 103 ACK	EQU 006H ; ASCII Ack character
0007		= 104 BELL	EQU 007H ; ASCII Bell character.
0008		= 105 BSPACE	EQU 008H ; ASCII Back space.
0009		= 106 HT	EQU 009H ; ASCII Horiz tab character.
0009		= 107 TAB	EQU 009H ; ASCII Horiz tab character.
000A		= 108 LF	EQU 00AH ; ASCII Line feed character.
000B		= 109 VT	EQU 00BH ; ASCII Vertical tab character
000C		= 110 FF	EQU 00CH ; ASCII Form Feed character.
000D		= 111 CR	EQU 00DH ; ASCII Carriage return character.
000E		= 112 SO	EQU 00EH ; ASCII Shift out character
000F		= 113 SI	EQU 00FH ; ASCII Shift in character
		= 114 ;	
0010		= 115 DLE	EQU 010H ; ASCII DLE character.
0011		= 116 DC1	EQU 011H ; ASCII Device control #1 (Xon)

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 3

LOC	OBJ	LINE	SOURCE STATEMENT
0012		= 117 DC2 EQU 012H	; ASCII Device control #2
0013		= 118 DC3 EQU 013H	; ASCII Device control #3 (Xoff)
0014		= 119 DC4 EQU 014H	; ASCII Device control #4
0015		= 120 NAK EQU 015H	; ASCII Negative ACK character
0016		= 121 SYN EQU 016H	; ASCII Sync character
0017		= 122 ETB EQU 017H	; ASCII End of transmission buffer.
0018		= 123 CAN EQU 018H	; ASCII Cancel.
0019		= 124 EM EQU 019H	; ASCII
001A		= 125 ASCSUB EQU 01AH	; ASCII Submit character.
001B		= 126 ESC EQU 01BH	; ASCII Escape character.
001C		= 127 FS EQU 01CH	; ASCII Field separator.
001D		= 128 GS EQU 01DH	; ASCII Group separator
001E		= 129 RS EQU 01EH	; ASCII Record separator
001F		= 130 US EQU 01FH	; ASCII Unit separator.
		= 131 ;	
		= 132 ;	More control characters:
		= 133 ;	
001F		= 134 CTRLMASK EQU 00011111B	; Mask off bottom 5 bits
		= 135 ;	
0013		= 136 CTRLS EQU CTRLMASK AND 'S'	; Control 'S'
0011		= 137 CTRLQ EQU CTRLMASK AND 'Q'	; Control 'Q'
001A		= 138 CTRLZ EQU CTRLMASK AND 'Z'	; Control 'Z'
		= 139 ;	
		= 140 ;	Printable characters:
		= 141 ;	
0020		= 142 SPACE EQU 020H	; ASCII Space.
0027		= 143 QUOTE EQU 027H	; Single quote ('')
007F		= 144 DEL EQU 07FH	; ASCII Delete character.
		= 145 ;	
		= 146 ;	-----
		= 147 ;	
		= 148 ;	D : Codes for Arithmetic operators.
		= 149 ;	
0000		= 150 NOPC EQU 00H	; Nop
0001		= 151 ORC EQU 01H	; OR Operand
0002		= 152 XORC EQU 02H	; XOR Operand
0003		= 153 ANDC EQU 03H	; AND Operand
0004		= 154 NOTC EQU 04H	; NOT Operand
0005		= 155 PLUSC EQU 05H	; '+'
0006		= 156 MINUSC EQU 06H	; '-'
0007		= 157 NEGC EQU 07H	; Negation
0008		= 158 MULTC EQU 08H	; '*'
0009		= 159 DIVC EQU 09H	; '/'
000A		= 160 MODC EQU 0AH	; MOD Operand
000B		= 161 SHRC EQU 0BH	; SHR Operand
000C		= 162 SHLC EQU 0CH	; SHL Operand
000D		= 163 LOWC EQU 0DH	; LOW Operand
000E		= 164 HIGHC EQU 0EH	; HIGH Operand
		= 165 ;	
		= 166 ;	Single character operators:
		= 167 ;	
002B		= 168 ADDSYM EQU '+'	; Addition Symbol.
002D		= 169 SUBSYM EQU '-'	; Subtraction Symbol.
002A		= 170 MULSYM EQU '*'	; Multiplication Symbol.
002F		= 171 DIVSYM EQU '/'	; Division Symbol.
		= 172 ;	
		= 173 ;	-----
		= 174 ;	
		= 175 ;	F : Opcode types:
		= 176 ;	
0000		= 177 NOOP EQU 0000B	; No operand
0001		= 178 D16 EQU 0001B	; 16 bit data field
0002		= 179 D8 EQU 0010B	; 8 bit data field
0003		= 180 SREG EQU 0011B	; Source register
0004		= 181 DREG EQU 0100B	; Destination register
0005		= 182 RPSTK EQU 0101B	; Register pair (Stack)
0006		= 183 RPATH EQU 0110B	; Register pair (Arithmetic)
0007		= 184 RPLIM EQU 0111B	; Register pair (Limited)
0008		= 185 RSTN EQU 1000B	; RST number
0009		= 186 UD8 EQU 1001B	; Unsigned 8 bit data field

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 4

LOC	OBJ	LINE	SOURCE STATEMENT
		= 187 ;	
		= 188 ;	
		= 189 ;	
		= 190 ;	G : General Equates (Markers etc)
		= 191 ;	
0080		= 192 NEVERF EQU 080H	; Dummy character used in tables (never found).
00FF		= 193 ETABLE EQU OFFH	; End of table marker
0040		= 194 KITACK EQU '@'	; Kit ack charater.
001A		= 195 EOFMARK EQU CTRLZ	; Kit EOF character = ^Z
0080		= 196 MSB EQU 10000000B	; Byte with it's MSBit set.
007F		= 197 OFFMSB EQU 01111111B	; Byte with all but it's MSBit set.
		= 198 ;	
		= 199 ;	
		= 200 ;	
		= 201 ;	H.1 : Buffer sizes.
		= 202 ;	
0006		= 203 LABBL EQU 6	; Label buffer length.
0006		= 204 OPBUFL EQU 6	; Opcode buffer length.
000A		= 205 CONBL EQU 10	; Control buffer length.
		= 206 ;	
		= 207 ;	H.2 : Stack sizes.
		= 208 ;	
001E		= 209 ASTKL EQU 15 * WORD	; Arithmetic stack.
001E		= 210 OSTKL EQU 30	; Operator stack.
0015		= 211 DSTKL EQU 21	; Read_number Digit stack.
		= 212 ;	
		= 213 ;	H.3 : Main asm stack length.
		= 214 ;	
008C		= 215 STACKL EQU 70 * WORD	;
		= 216 ;	
0084		= 217 MAXBL EQU 132	; Max Line buffer length (Used by fileio).
0032		= 218 MAXPOL EQU 50	; Max Plus Signs on any line.
0010		= 219 OBJLEN EQU 16	; Max Intel Paper Tape format record len.
		= 220 ;	
		= 221 ;	
		= 222 ;	
		= 223 ;	I : Memory usage equates.
		= 224 ;	
EF00		= 225 MONDATA EQU 0EF00H	; Start of monitor data area.
EFFC		= 226 SPOINT EQU MONDATA+100H-4	;
EFFE		= 227 EPOINT EQU MONDATA+100H-2	;
		= 228 ;	
E000		= 229 STABLE EQU 0E000H	; Start address of the Symbol table.
		= 230 ;	; (Size = 256 * 8 = 2K)
		= 231 ;	
1000		= 232 SCODE EQU 01000H	; Start address of the Assembler.
E800		= 233 DBLOCK EQU 0E800H	; Start address of the assembler data area
		= 234 ;	; (Size +/- 256 bytes , includes the stack)
		= 235 ;	
0000		= 236 MINCOD EQU 0000H	; Lower bound of object code area
OFFF		= 237 MAXCOD EQU OFFFH	; Upper bound of object code area
		= 238 ;	
		= 239 ;	
		= 240 ;	
		= 241 ;	J : Defaults
		= 242 ;	
FF00		= 243 DEFPP EQU FALSE	; Default = print OFF.;
003C		= 244 DEFLPP EQU 60	; Default Lines Per Page = 60
		= 245 ;	
00FF		= 246 LABELE EQU OFFH	; Label expected.
0000		= 247 OPERE EQU 000H	; Operator expected.
		= 248 ;	
0063		= 249 FATALN EQU 99	; Fatal error number.
		= 250 ;	
00FF		= 251 M8085 EQU TRUE	; Gen assembler for 8085 and not 8080.
		= 252 ;	
		= 253 ;	End of equates. (EQUATES.A85)
		= 254 SEJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 5

LOC	OBJ	LINE	SOURCE STATEMENT
		255	
1000		256	ORG SCODE ; Start Program at Start_Code.
		257	;
		258	START: ; Assembler entry point.
		259	;
		260	-----
		261	;
		262	;
		263	Start of pass #1.
		264	-----
		265	;
		266	SPASS1: ; Start of Pass 1 label.
		267	;
1000	31FDE9	268	LXI SP,NEWSP ; Setup new stack pointer.
1003	3E00	269	MVI A,FALSE ;
1005	3220E8	270	STA PDPFLG ; PDP_flag = False (* Output to console *)
1008	2AFEEF	271	LHLD EPOINT ;
100B	EB	272	XCHG ;
100C	2AFCEF	273	LHLD SPOINT ;
100F	CD2313	274	CALL DCMF ; If EPOINT > SPOINT then assemble file.
1012	DA1E10	275	JC OKASMF ;
		276	;
		277	;
1015	21072F	277	LXI H,NOFILM ;
1018	CDB51F	278	CALL STRING ; Print "No File in Memory" message
101B	C38110	279	JMP FEXASM ; and Exit assembler.
		280	OKASMF: ;
		281	;
		282	-----
		283	;
		284	;
		285	There is a file in memory, so start pass 1.
101E	CD8510	286	CALL P1INIT ; Pass 1 init.
		287	NLINE1: ; Next line loop for pass 1.
1021	CDAF10	288	CALL P1LINE ; Pass 1 Line routine.
1024	3A70E9	289	LDA PASSN ;
1027	FE01	290	CPI 1 ; Pass Number still 1 ?
1029	C23D10	291	JNZ SPAS2 ; If not then Start pass 2.
		292	;
102C	CD3113	293	CALL QEFILE ; Check for end of file.
102F	DA2110	294	JC NLINE1 ; If Not end of file then loop.
		295	;
		296	;
		297	-----
		298	;
		299	There was no end in the file, set pass number = 2 and print message.
		300	;
1032	3E02	301	MVI A,2 ; Set pass Number = 2.
1034	3270E9	302	STA PASSN ;
1037	21182F	303	LXI H,NOENDM ; Print "No END in file" Message.
103A	CDB51F	304	CALL STRING ;
		305	;
		306	-----
		307	;
		308	;
		309	Start pass #2.
		310	-----
		311	;
		312	SPAS2: ;
103D	CD3511	313	CALL P2INIT ; Pass 2 initialize
		314	NLINE2: ; Pass 2 loop.
1040	CD7311	315	CALL P2LINE ; Pass 2 line routine.
1043	CD9820	316	CALL PLINE ; Print line.
1046	CD0321	317	CALL PAERRS ; Print any error messages.
		318	;
1049	3A15E8	319	LDA FATALF ;
104C	FEFF	320	CPI TRUE ;
104E	CA7E10	321	JZ EXASM ; IF Fatal exit flag = true then Exit.
		322	;
1051	3A70E9	323	LDA PASSN ;
1054	FE02	324	CPI 2 ; If Pass number # 2 then exit.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 6

LOC	OBJ	LINE	SOURCE STATEMENT
1056	C26510	325	JNZ EPAS2 ; Else
		326	;
1059	CD3113	327	CALL QEFILF ; If NOT End_Of_File
105C	DA4010	328	JC NLINE2 ; Then loop to P2LINE.
		329	;
105F	21182F	330	LXI H,NOENDM ; Print "No END in file" message.
1062	CDB51F	331	CALL STRING ;
		332	EPAS2: ; End of pass 2.
1065	21362F	333	LXI H,ENDM1 ;
1068	CDB51F	334	CALL STRING ; Print "End of assembly:"
106B	CD8B21	335	CALL PNERRS ; Print Error summary.
		336	;
106E	3A6CE9	337	LDA SYMFLG ;
1071	FE80	338	CPI UNDEF ; If Symbol print flag = Undefined then
1073	C27910	339	JNZ SYMDEF ;
1076	3A69E9	340	LDA PFLAG ; Use Print flag as Symbol print flag.
		341	SYMDEF: ; Symbol defined.
		342	;
1079	FEFF	343	CPI TRUE ; If Symbol table print flag = true then
		344	; print the symbol table.
107B	CC2719	345	CZ STDUMP ;
		346	EXASM: ; Exit asm.
107E	CD2F1D	347	CALL CLOSEFILE ; Close output file (Only if there was one)
		348	FEXASM: ;
1081	CD831D	349	CALL CRLF ;
1084	76	350	HLT ; Return to monitor.
		351	;
		352	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 7

LOC	OBJ	LINE	SOURCE STATEMENT
		353 ;	=====
		354 ;	
		355 ;	INCLUDE EXTERNAL FILES.
		356 ;	
		357 ;	=====
		358 ;	
		359 ;	PASS 1 ROUTINE (With pass 1 init routine).
		360 ;	
		361	\$INCLUDE(ASMPASS1.A85)
=		362 ;	-----;
=		363 ;	;
=		364 ;	Pass number 1 module for the UCT SABus assembler ;
=		365 ;	----- ;
=		366 ;	;
=		367 ;	This module contains (in order) the following subroutines: ;
=		368 ;	;
=		369 ;	a) P1INIT - Pass number 1 initialise routine ;
=		370 ;	b) P1LINE - Pass number 1 process line routine ;
=		371 ;	;
=		372 ;	-----;
=		373 ;	
=		374	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		= 375 ;	=====
		= 376 ;*	*
		= 377 ;*	Function : P1INIT. (Pass 1 initialise)
		= 378 ;*	Inputs : None.
		= 379 ;*	Outputs : None.
		= 380 ;*	Destroys : H, L, Acc and Flags.
		= 381 ;*	Calls : TABINT and STRING.
		= 382 ;*	*
		= 383 ;*	Description: This routine inits all variables for pass
		= 384 ;*	number 1.
		= 385 ;*	*
		= 386 ;	=====
		= 387 ;	
		= 388 P1INIT:	; Start of <P1INIT> routine
		= 389	;
1085	CD1719	= 390	CALL TABINT ; Init Symbol table.
		= 391	;
1088	218B2E	= 392	LXI H,STARTM ;
108B	CDB51F	= 393	CALL STRING ; Print Start of pass 1 message.
		= 394	;
108E	210000	= 395	LXI H,0000H ;
1091	2265E9	= 396	SHLD PC ; Setup PC = 0000.
		= 397	;
1094	3E80	= 398	MVI A,UNDEF ;
1096	326CE9	= 399	STA SYMFLG ; Symbol table print flag = Undefined.
		= 400	;
1099	3E00	= 401	MVI A,FALSE ;
109B	3269E9	= 402	STA PFLAG ; Asm listing print flag = False.
		= 403	;
109E	3EFF	= 404	MVI A,TRUE ;
10A0	326BE9	= 405	STA PCDEF ; PC defined = true.
		= 406	;
10A3	3E01	= 407	MVI A,1 ;
10A5	3270E9	= 408	STA PASSN ; Passnumber = 1.
		= 409	;
10A8	2AFCEF	= 410	LHLD SPOINT ;
10AB	2200E8	= 411	SHLD POINT ; Set text Pointer = Start_of_text.
		= 412	;
10AE	C9	= 413	RET ; Return from <P1INIT> routine
		= 414 ;	
		= 415 ;	=====
		= 416 ;*	*
		= 417 ;*	Function : P1LINE. (Pass number 1 line routine.
		= 418 ;*	Inputs : POINT(MEMORY) = Start of line to process.
		= 419 ;*	Outputs : None.
		= 420 ;*	Destroys : A, B, C, D, E, H, L and flags.
		= 421 ;*	Calls : CONTROL, QLET, COPYL, RUNTOB, COPY, QEQU,
		= 422 ;*	EQUROU, ADRINS, SEMILF, ADDPC, QPHOP, PERR
		= 423 ;*	and RULF.
		= 424 ;*	*
		= 425 ;*	Description: This routine is called for each line of the
		= 426 ;*	source file during pass #1.
		= 427 ;*	*
		= 428 ;	=====
		= 429 ;	
		= 430 P1LINE:	; Start of Pass 1 process line.
10AF	3E00	= 431	MVI A,0 ;
10B1	3214E8	= 432	STA LERRS ; Set Line error count = zero.
		= 433	;
10B4	2116E8	= 434	LXI H,ERRBUF ;
10B7	2210E8	= 435	SHLD ERRORP ; Error_buffer_ptr = Start_of_error_buffer.
		= 436 ;	
		= 437 ;	-----
		= 438 ;	
		= 439 ;	Start of pass 1 process line, 1st check for a control line.
		= 440 ;	
10BA	2A00E8	= 441	LHLD POINT ;
10BD	7E	= 442	MOV A,M ; Get 1st char . (test for label) .
10BE	FE24	= 443	CPI '\$' ; Is the 1st char a dollar . (control line)
10C0	C2C910	= 444	JNZ NOCON1 ; Skip control routine call

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 9

LOC	OBJ	LINE	SOURCE STATEMENT
		= 445	
10C3	CD7516	= 446	CALL CONTROL ; Process control line.
		= 447	
10C6	C33111	= 448	JMP ENDLIN ; End pass 1 process line.
		= 449 ;	
		= 450 ;	
		= 451 ;	
		= 452 ;	It's not a control line, now check for a label.
		= 453 ;	
		= 454	NOCON1: ; Areg = Current asm character.
10C9	CD5C14	= 455	CALL QLET ; Letter ?
10CC	C2FA10	= 456	JNZ NOLABP ; If there is a letter then it must be a label
		= 457 ;	
		= 458 ;	
		= 459 ;	
		= 460 ;	The first character on the line was a letter, there is therefore a
		= 461 ;	label in this line. Copy label string into label buffer. If the
		= 462 ;	label string is terminated with a colon then it is entered
		= 463 ;	into the symbol table with the current PC as it's value. If the
		= 464 ;	label string is not terminated with colon then the line must be an
		= 465 ;	equate line, check for equate directive in the opcode field.
		= 466 ;	
10CF	21E3E8	= 467	LXI H,LABELB ;
10D2	CDBF13	= 468	CALL COPYL ; Copy label into label buffer
10D5	2A00E8	= 469	LHLD POINT ; Update asm pointer.
10D8	7E	= 470	MOV A,M ; Fetch label terminating character.
10D9	FE3A	= 471	CPI ':' ; Colon ? (I.e. Is it a label ?)
10DB	CAF310	= 472	JZ ADRSET ; Yes - Then place in symbol table.
		= 473 ;	
		= 474 ;	
		= 475 ;	
		= 476 ;	There is no colon, it must be an equate.
		= 477 ;	
10DE	CD5F13	= 478	CALL RUNTOB ; Skip blanks and tabs
10E1	2153E9	= 479	LXI H,OPBUF ;
10E4	CD8313	= 480	CALL COPY ; Copy Opcode into opcode buffer.
10E7	CD4013	= 481	CALL QEUQ ; Opcode_buffer = 'EQU' ?
10EA	C22411	= 482	JNZ EQUERR ; No - Then equate error.
		= 483 ;	
10ED	CD0216	= 484	CALL EQUROU ; Else call equate routine.
10F0	C33111	= 485	JMP ENDLIN ;
		= 486 ;	
		= 487 ;	
		= 488 ;	
		= 489 ;	If there is a colon on the label then come here.
		= 490 ;	
		= 491	ADRSET: ;
10F3	23	= 492	INX H ; Inc past colon.
10F4	2200E8	= 493	SHLD POINT ;
10F7	CD0F13	= 494	CALL ADRINS ; Insert adr in symbol table.
		= 495 ;	
		= 496 ;	
		= 497 ;	
		= 498 ;	There was no label, look for an opcode.
		= 499 ;	
		= 500	NOLABP: ;
10FA	CD5F13	= 501	CALL RUNTOB ; Skip tabs and blanks
10FD	CD6E13	= 502	CALL SEMILF ; Is it a blank line ?
1100	CA3111	= 503	JZ ENDLIN ; Yes - Then exit.
		= 504	
		= 505	
1103	2153E9	= 506	LXI H,OPBUF ;
1106	CD8313	= 507	CALL COPY ; Copy next string into opcode buffer.
1109	CD1C1A	= 508	CALL QOP ; Opcode ?
110C	C21811	= 509	JNZ P1NTOP ; No then pass 1 not opcode.
		= 510	
110F	3A5DE9	= 511	LDA BYTES ;
1112	CD0013	= 512	CALL ADDPC ; Add number of bytes in the inst to PC.
1115	C33111	= 513 ;	JMP ENDLIN ;
		= 514 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 10

LOC	OBJ	LINE	SOURCE STATEMENT
		= 515 ;	
		= 516 ;	It's not an opcode (Could it be a directive ?)
		= 517 ;	
		= 518 PINTOP:	
1118	CDAB1C	= 519	CALL QPHOP ; Ph opcode ?
111B	C22C11	= 520	JNZ UNDEFO ; No - Then it's undefined.
		= 521	
111E	213111	= 522	LXI H,ENDLIN ;
1121	E5	= 523	PUSH H ; Set up return address.
1122	EB	= 524	XCHG ; Move directive address from DE to HL
1123	E9	= 525	PCHL ; Jump to directive routine.
		= 526 ;	
		= 527 ;	=====
		= 528 ;	
		= 529 ;	Pass number 1 error messages.
		= 530 ;	
		= 531 EQUERR:	; Equate error.
1124	3E0E	= 532	MVI A,14 ;
1126	CD3E21	= 533	CALL PERR ; Gen error #14 (Equate directive error)
1129	C33111	= 534	JMP ENDLIN ; and exit.
		= 535 ;	
		= 536 ;	-----
		= 537 ;	
		= 538 UNDEFO:	; Undefined opcode error.
112C	3E01	= 539	MVI A,1 ;
112E	CD3E21	= 540	CALL PERR ; Gen error #1 (Undefined opcode)
		= 541 ;	
		= 542 ;	-----
		= 543 ;	
		= 544 ENDLIN:	; End of pass 1 line.
1131	CD4D13	= 545	CALL RULF ;
1134	C9	= 546	RET ; Return from pass 1 proc line routine.
		= 547 ;	
		= 548 ;	End of module <ASMPASS1>
		549 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 11

LOC	OBJ	LINE	SOURCE STATEMENT
		550 ;	
		551 ;	-----
		552 ;	
		553 ;	P A S S 2 R O U T I N E (With pass 2 init routine).
		554 ;	
		555	\$INCLUDE(ASMPASS2.A85)
		= 556 ;	-----
		= 557 ;	
		= 558 ;	Pass number 2 module for the UCT SABus assembler ;
		= 559 ;	----- ;
		= 560 ;	
		= 561 ;	This module contains (in order) the following subroutines: ;
		= 562 ;	
		= 563 ;	a) P2INIT - Pass number 2 init routine. ;
		= 564 ;	b) P2LINE - Pass number 2 process line routine. ;
		= 565 ;	c) P2LABL - Pass number 2 process label routine. ;
		= 566 ;	d) P2POPR - Pass number 2 process opcode routine. ;
		= 567 ;	
		= 568 ;	-----
		= 569 ;	
		= 570	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 12

LOC	OBJ	LINE	SOURCE STATEMENT
		= 571 ;	=====
		= 572 ;*	*
		= 573 ;*	Function : P2INIT. (Pass 2 initialise)
		= 574 ;*	Inputs : None.
		= 575 ;*	Outputs : None.
		= 576 ;*	Destroys : H, L, Acc and Flags.
		= 577 ;*	Calls : STRING.
		= 578 ;*	*
		= 579 ;*	Description: This routine inits all variables for pass
		= 580 ;*	number 2.
		= 581 ;*	*
		= 582 ;	=====
		= 583 ;	
		= 584	P2INIT: ; Start of P2INIT.
1135	3E00	= 585	MVI A,DEFPF ;
1137	3269E9	= 586	STA PFLAG ; Set print flag = Default print flag.
		= 587	;
113A	3E3C	= 588	MVI A,DEFLPP ;
113C	320AE8	= 589	STA LPP ; Setup default lines per page.
		= 590	;
113F	3E00	= 591	MVI A,FALSE ;
1141	3205E8	= 592	STA NEWPAG ; Reset New Page Flag.
1144	3215E8	= 593	STA FATALF ; Reset fatal error flag.
		= 594	;
1147	3EFF	= 595	MVI A,TRUE ;
1149	326BE9	= 596	STA PCDEF ; Set PC defined flag.
		= 597	;
114C	210000	= 598	LXI H,0000H ;
114F	2206E8	= 599	SHLD LINEN ; Zero Line number.
1152	2265E9	= 600	SHLD PC ; Program counter.
1155	2212E8	= 601	SHLD ERRORS ; Error count.
1158	2208E8	= 602	SHLD PAGNUM ; Set page number = 0.
		= 603	;
115B	2AFCEF	= 604	LHLD SPOINT ;
115E	2200E8	= 605	SHLD POINT ; Set Pointer = Start of Text.
		= 606	;
1161	21AA2E	= 607	LXI H,SP2MES ;
1164	CDB51F	= 608	CALL STRING ; Print Start of pass 2 message.
		= 609	;
1167	3E00	= 610	MVI A,FALSE ;
1169	320DE8	= 611	STA BRIEFF ; Set brief flag to false
116C	3220E8	= 612	STA PDPFLG ; Reset PDP print flag (Note: after
		= 613	Start of pass 2 message).
116F	32C7E8	= 614	STA OBJFLG ; Reset OBJECT file flag.
		= 615	;
1172	C9	= 616	RET ; Return from Pass 2 Init.
		= 617 ;	
		= 618	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 13

LOC	OBJ	LINE	SOURCE STATEMENT
		= 619 ;	=====
		= 620 ;*	*
		= 621 ;*	Function : P2LINE. (Pass 2 process line)
		= 622 ;*	Inputs : POINT(Memory) = Start of line to process.
		= 623 ;*	Outputs : None.
		= 624 ;*	Destroys : All.
		= 625 ;*	Calls : CONTROL, QLET, QTABSP, SEMILF, PERR, COPYL,
		= 626 ;*	RUNTOB, COPY, QEQU, P2LABL, QOP, QPHOP, RULF
		= 627 ;*	and P2POPR.
		= 628 ;*	*
		= 629 ;*	DESCRIPTION: This routine is called for each line of
		= 630 ;*	the source file during pass #2.
		= 631 ;*	*
		= 632 ;	=====
		= 633 ;	
		= 634 P2LINE:	; Pass 2 Line Routine (called for every line
		= 635	; in the source file)
1173	3EFF	= 636 MVI A,TRUE	
1175	320EE8	= 637 STA GCFLAG	; Generate_Code_Flag = TRUE.
1178	326AE9	= 638 STA PAFLAG	; Print_Address_Flag = TRUE.
		= 639	
117B	3E00	= 640 MVI A,FALSE	
117D	320FE8	= 641 STA EQUFLG	; Equate_flag = FALSE.
1180	326EE9	= 642 STA CONFLG	; Control_Line_Flag = FALSE.
		= 643	
1183	3E00	= 644 MVI A,00H	
1185	3214E8	= 645 STA LERRS	; Line Error count = 0.
		= 646	
1188	2116E8	= 647 LXI H,ERRBUF	
118B	2210E8	= 648 SHLD ERRORP	; Error buffer pointer = ERRBUF.
		= 649	
118E	2A65E9	= 650 LHLD PC	
1191	2267E9	= 651 SHLD LASTPC	; Update LASTPC with the Current PC.
		= 652	
1194	2A06E8	= 653 LHLD LINEN	
1197	23	= 654 INX H	; Inc line Number.
1198	2206E8	= 655 SHLD LINEN	
		= 656 ;	
		= 657 ;	-----
		= 658 ;	
		= 659 ;	Start to process line.
		= 660 ;	
119B	2A00E8	= 661 LHLD POINT	
119E	2202E8	= 662 SHLD LPOINT	; Update Last_Pointer = Current Pointer.
11A1	7E	= 663 MOV A,M	; Fetch the 1st char from the text.
11A2	FE24	= 664 CPI '\$'	; Control Line ?
11A4	C2AD11	= 665 JNZ NOCON2	; No - then skip Control routine.
		= 666	
11A7	CD7516	= 667 CALL CONTROL	; Call CONTROL.
11AA	C34212	= 668 JMP P2ENDL	; Exit Pass 2 line.
		= 669 ;	
		= 670 ;	-----
		= 671 ;	
		= 672 ;	Not control line, is there a label on the line ?
		= 673 ;	
		= 674 NOCON2:	; Not control.
11AD	CD5C14	= 675 CALL QLET	; Is it a letter ?
11B0	CAC811	= 676 JZ P2LABR	; Yes - then handle labels
		= 677	
11B3	7E	= 678 MOV A,M	; Restore character.
		= 679	
11B4	CD4713	= 680 CALL QTABSP	; If Char in [Tab, Space] then
11B7	CAFF11	= 681 JZ P2POP	; read Opcode.
		= 682	
11BA	CD6E13	= 683 CALL SEMILF	
11BD	CAFF11	= 684 JZ P2POP	; If Char IN [LF, ';'] then Exit via P2POP.
		= 685	
11C0	3E27	= 686 MVI A,39	; Else
11C2	CD3E21	= 687 CALL PERR	; Gen error 39 (Invalid label char).
11C5	C34212	= 688 JMP P2ENDL	

LOC	OBJ	LINE	SOURCE STATEMENT
		= 689 ;	
		= 690 ;	
		= 691 ;	
		= 692 ;	The first character on the line was a letter, handle label.
		= 693 ;	It could be a label (with a colon) or an equate line.
		= 694 ;	
		= 695 P2LABR:	
11C8	21E3E8	= 696	LXI H,LABELB ;
11CB	CDBF13	= 697	CALL COPYL ; Copy the String into the Label buffer.
11CE	2A00E8	= 698	LHLD POINT ; Save new asm pointer.
11D1	7E	= 699	MOV A,M ; Get the 1st character after the LABEL.
11D2	FE3A	= 700	CPI ':' ; COLON ?, A Label or an Equate ?
11D4	CAEE11	= 701	JZ P2ADRS ; A Colon => a Label.
		= 702 ;	
		= 703 ;	
		= 704 ;	
		= 705 ;	Label string not terminated by a colon, check for equate directive)
		= 706 ;	
11D7	CD5F13	= 707	CALL RUNTOB ; Skip tabs and blanks
11DA	2153E9	= 708	LXI H,OPBUF ;
11DD	CD8313	= 709	CALL COPY ; Copy opcode field in the opcode buffer
11E0	CD4013	= 710	CALL QEQU ; Equate ?
11E3	CAF411	= 711	JZ P2EQU ; Yes - then process equate directive
		= 712 ;	
11E6	3E0E	= 713	MVI A,14 ; Else
11E8	CD3E21	= 714	CALL PERR ; Error #14 (Equate error)
11EB	C34212	= 715	JMP P2ENDL ; Exit P2 process line
		= 716 ;	
		= 717 ;	
		= 718 ;	
		= 719 ;	Label found on line (aaaaaa:) Check symbol table.
		= 720 ;	
		= 721 P2ADRS:	; Pass 2 Adr (i.e. Label found)
11EE	CD4612	= 722	CALL P2LABL ;
11F1	C3FF11	= 723	JMP P2POP ;
		= 724 ;	
		= 725 ;	
		= 726 ;	
		= 727 ;	Equate found on line (aaaaaa EQU <Expression>).
		= 728 ;	
		= 729 P2EQU:	; Pass 2 Equate.
11F4	3EFF	= 730	MVI A,TRUE ;
11F6	320FE8	= 731	STA EQUFLG ; Equate_Flag = TRUE.
		= 732 ;	
11F9	CD0216	= 733	CALL EQUROU ;
11FC	C34212	= 734	JMP P2ENDL ;
		= 735 ;	
		= 736 ;	
		= 737 ;	
		= 738 ;	P2 process opcode.
		= 739 ;	
		= 740 P2POP:	; Pass 2 Process opcode.
11FF	2A00E8	= 741	LHLD POINT ; Load HL with current asm pointer
1202	CD5F13	= 742	CALL RUNTOB ; Skip until not tab of blank.
1205	7E	= 743	MOV A,M ;
1206	CD6E13	= 744	CALL SEMILF ; If character = ';' or LF then comment_line
1209	CA2A12	= 745	JZ COMLIN ;
		= 746 ;	
120C	2153E9	= 747	LXI H,OPBUF ;
120F	CD8313	= 748	CALL COPY ; Get the next string into the opcode buffer.
1212	CD1C1A	= 749	CALL QOP ; Is it an Opcode ?
1215	C21E12	= 750	JNZ P2OPNF ; No - The check for an assembler directive.
		= 751 ;	
		= 752 P2OPF:	; Pass 2 Opcode found.
1218	CD8012	= 753	CALL P2POPR ;
121B	C34212	= 754	JMP P2ENDL ;
		= 755 ;	
		= 756 ;	
		= 757 ;	
		= 758 ;	Pass 2 check for Ph Opcode.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 15

LOC	OBJ	LINE	SOURCE STATEMENT
		= 759 ;	
		= 760 P2OPNF:	
121E	CDAB1C	= 761	CALL QPHOP ; Is it an assembler directive ?
1221	C23212	= 762	JNZ P2PHNF ; No then Error.
		= 763	
1224	214212	= 764	LXI H,P2ENDL ;
1227	E5	= 765	PUSH H ; Setup return adr on the stack.
1228	EB	= 766	XCHG ; Move the start adr of the Assm directive
		= 767	; routine from DE -> HL.
1229	E9	= 768	PCHL ; Indirect jump to the routine.
		= 769 ;	
		= 770 ;=====	
		= 771 ;	
		= 772 ;	P2 process line exit points:
		= 773 ;	
		= 774 COMLIN:	; Comment line.
122A	3E00	= 775	MVI A,FALSE ;
122C	326AE9	= 776	STA PAFLAG ; Reset Print adr Flag.
122F	C34212	= 777	JMP P2ENDL ; Exit pass 2 line Routine.
		= 778 ;	
		= 779 ;=====	
		= 780 ;	
		= 781 ;	If the opcode field is not a directive or an opcode check for
		= 782 ;	an equate directive, even if it is there it is still an error.
		= 783 ;	
		= 784 P2PHNF:	; Pass 2 Ph Opcode not found.
1232	CD4013	= 785	CALL QEQU ;
1235	C23D12	= 786	JNZ P2PHNE ;
		= 787	
1238	3E26	= 788	MVI A,38 ;
123A	C33F12	= 789	JMP P2EPRT ;
		= 790 ;	
		= 791 ;=====	
		= 792 ;	
		= 793 ;	P2 error exit (pass 2 undefined opcode)
		= 794 P2PHNE:	
123D	3E01	= 795	MVI A,1 ; Therefore unknown Opcode.
		= 796 P2EPRT:	
123F	CD3E21	= 797	CALL PERR ; Print error message.
		= 798	
		= 799 P2ENDL:	; Pass 2 End of line.
1242	CD4D13	= 800	CALL RULF ; Read until line feed.
		= 801 ENDL2:	
1245	C9	= 802	RET ; Return from <P2LINE> routine.
		= 803 ;	
		= 804 ;=====	
		= 805 ;*	
		= 806 ;*	Function : P2LABEL (Pass 2 label on line)
		= 807 ;*	Inputs : HL = Pointer to colon terminator.
		= 808 ;*	Outputs : None.
		= 809 ;*	Destroys : H, L, D, E, A and flags.
		= 810 ;*	Calls : FETCH, DCMF, PERR and ADRINS.
		= 811 ;*	
		= 812 ;*	Description: This routine handles the case where there is a
		= 813 ;*	label on a line during pass 2. The label string has been
		= 814 ;*	copied into the label buffer. There routine attempts to
		= 815 ;*	lookup the symbol. If the symbol is undefined then it is
		= 816 ;*	defined by the routine. If the symbol is defined in the
		= 817 ;*	current pass then a multiply defined symbol error is
		= 818 ;*	printed. If the symbol was defined in pass 1 but has a
		= 819 ;*	a different value to the current program counter then a
		= 820 ;*	phase error is generated.
		= 821 ;*	
		= 822 ;=====	
		= 823 ;	
		= 824 P2LABEL:	; Start of P2LABEL.
		= 825	
1246	23	= 826	INX H ; Skip past ':' .
1247	2200E8	= 827	SHLD POINT ; Update asm pointer
124A	CD2E18	= 828	CALL FETCH ; Fetch the label in the label buffer

```

LOC   OBJ       LINE      SOURCE STATEMENT
124D  CA5612     = 829      JZ      P2ADRD      ; Defined ? - Yes then check value and
                  = 830                      ; the pass during which it was defined.
                  = 831 ;
                  = 832 ;-----
                  = 833 ;
                  = 834 ;      Label was not previously defined, so define it.
                  = 835 ;
                  = 836 P2ADRU:                ;
1250  CDOF13     = 837      CALL   ADRINS      ; Insert address in the symbol table.
1253  C37F12     = 838      JMP    P2LABE      ; Exit from P2 label routine.
                  = 839 ;
                  = 840 ;-----
                  = 841 ;
                  = 842 ;      The label has been previously defined. Check defining pass number.
                  = 843 ;
                  = 844 P2ADRD:                ; Pass number 2, Adr defined.
1256  3AF3E8     = 845      LDA     PASDEF      ;
1259  47         = 846      MOV     B,A        ;
125A  3A70E9     = 847      LDA     PASSN      ;
125D  B8         = 848      CMP     B          ; Pass_defined = current_pass ?
125E  CA7A12     = 849      JZ      P2MULD      ; Yes - Then label multiply defined.
                  = 850 ;
                  = 851 ;-----
                  = 852 ;
                  = 853 ;      The label was defined on the previous pass, check value.
                  = 854 ;      and set defined pass = 2.
                  = 855 ;
1261  2A63E9     = 856      LHLD    SADR        ; Get start adr of entry in symbol table.
1264  7E         = 857      MOV     A,M        ;
1265  F680       = 858      ORI     MSB        ; Set MSBit.
1267  77         = 859      MOV     M,A        ;
                  = 860 ;
1268  2AE9E8     = 861      LHLD    LABELB+LABBL ; Fetch value of label.
126B  EB         = 862      XCHG                     ;
126C  2A65E9     = 863      LHLD    PC          ; load HL with PC / DE = label.
126F  CD2313     = 864      CALL    DCMF        ; Compare label_value and PC.
1272  CA7F12     = 865      JZ      P2LABE      ; Equal - Yes then no Phase error.
                  = 866 ;
                  = 867 ;-----
                  = 868 ;
                  = 869 ;      Phase error error exit point.
                  = 870 ;
1275  3E14       = 871      MVI     A,20        ; Gen "Phase error" message.
1277  C37C12     = 872      JMP     P2LBER      ;
                  = 873 ;
                  = 874 ;-----
                  = 875 ;
                  = 876 ;      Multiply defined symbol error exit point.
                  = 877 ;
                  = 878 P2MULD:                ;
127A  3E04       = 879      MVI     A,4        ; Else ( gen Multiply defined symbol error )
                  = 880 P2LBER:                ;
127C  CD3E21     = 881      CALL    PERR        ; Print error message and exit.
                  = 882 P2LABE:                ;
127F  C9         = 883      RET                     ; Return from <P2LABL>
                  = 884 ;
                  = 885 ;=====
                  = 886 ;*
                  = 887 ;* Function : P2POPR (Pass 2 operator routine) *
                  = 888 ;* Inputs : POINT(Memory) = Assembly pointer. *
                  = 889 ;* Outputs : None. *
                  = 890 ;* Destroys : A, B, C, D, E, H, L and flags. *
                  = 891 ;* Calls : RUNTOB, SEMILF, GETOPR, PERR, OCODBUF and ADDPC *
                  = 892 ;*
                  = 893 ;* Description: This routine handles the operand fetch and *
                  = 894 ;* code generation for opcodes during pass 2. *
                  = 895 ;*
                  = 896 ;=====
                  = 897 ;
                  = 898 P2POPR:                ;

```


UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 17

LOC	OBJ	LINE	SOURCE STATEMENT
1280	2A00E8	= 899	LHLD POINT ;
1283	CD5F13	= 900	CALL RUNTOB ; Move asm pointer to next char.
1286	3A5EE9	= 901	LDA OPERS ; Get operand count for the opcode.
1289	FE00	= 902	CPI 0 ; Zero operands ?
128B	CAC912	= 903	JZ NOOPER ; Yes - then goto No Operands.
		= 904	;
128E	7E	= 905	MOV A,M ;
128F	CD6E13	= 906	CALL SEMILF ; <LF> or ';' ?
1292	CAE712	= 907	JZ ERRNOP ; Yes - error (No Operands)
		= 908	;
		= 909	GETOPS: ; Get operands.
1295	3A5EE9	= 910	LDA OPERS ;
1298	E60F	= 911	ANI 00001111B ; Mask off 1st 4 bit Operand spec.
129A	CD1222	= 912	CALL GETOPR ; Get the Operand
129D	C2DB12	= 913	JNZ NOOPOK ; If Getopr error return then exit.
		= 914	;
12A0	3A5EE9	= 915	LDA OPERS ;
12A3	E6F0	= 916	ANI 11110000B ; Mask Off 2nd 4 bit Operand spec.
12A5	0F	= 917	RRC ;
12A6	0F	= 918	RRC ;
12A7	0F	= 919	RRC ; Shift 4 bits righth (into lower nibble).
12A8	0F	= 920	RRC ;
12A9	B7	= 921	ORA A ; Set flags.
12AA	47	= 922	MOV B,A ;
12AB	CAC912	= 923	JZ NOOPER ; If Upper spec zero then no second operand.
		= 924	;
12AE	2A00E8	= 925	LHLD POINT ;
12B1	7E	= 926	MOV A,M ; Check that the 2 operands are seperated
		= 927	by a comma.
12B2	FE2C	= 928	CPI ',' ;
12B4	C2EC12	= 929	JNZ ERRNSO ; If not then error.
		= 930	;
12B7	23	= 931	INX H ; Else inc past the comma.
12B8	CD5F13	= 932	CALL RUNTOB ;
12BB	7E	= 933	MOV A,M ;
12BC	CD6E13	= 934	CALL SEMILF ; <LF> or ';' ?
12BF	CAEC12	= 935	JZ ERRNSO ; Yes - error (There should be operands)
		= 936	;
12C2	78	= 937	MOV A,B ; Restore operand spec.
12C3	CD1222	= 938	CALL GETOPR ; Get 2nd operand.
12C6	C2DB12	= 939	JNZ NOOPOK ; If getopr returns error then exit.
		= 940	;
		= 941	NOOPER: ;
12C9	2A00E8	= 942	LHLD POINT ;
12CC	CD5F13	= 943	CALL RUNTOB ; Skip.
12CF	7E	= 944	MOV A,M ; Get Character.
12D0	CD6E13	= 945	CALL SEMILF ; <LF> OR ';' ?
12D3	CADB12	= 946	JZ NOOPOK ; Yes - Then no other characters on the line
		= 947	;
12D6	3E25	= 948	MVI A,37 ; Else Error (Extra chars in line)
12D8	CD3E21	= 949	CALL PERR ;
		= 950	NOOPOK: ;
12DB	3A5FE9	= 951	LDA OPCODE ; Fetch the Opcode.
12DE	32DFE8	= 952	STA CODBUF ; Save opcode as the 1st byte of the inst code.
12E1	CD0724	= 953	CALL OCODBUF ; Generate the code bytes.
12E4	C3F912	= 954	JMP P2POPE ;
		= 955 ;	
		= 956 ;	=====
		= 957 ;	
		= 958	ERRNOP: ; Error - No Operands
12E7	3E18	= 959	MVI A,24 ;
12E9	C3EE12	= 960	JMP P2OERR ;
		= 961 ;	
		= 962 ;	-----
		= 963 ;	
		= 964	ERRNSO: ; Error - no Second operand found.
12EC	3E19	= 965	MVI A,25 ;
		= 966	P2OERR: ;
12EE	CD3E21	= 967	CALL PERR ; Gen Error message.
12F1	3E00	= 968	MVI A,0 ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM PAGE 18

LOC	OBJ	LINE	SOURCE STATEMENT
12F3	32E0E8	= 969	STA CODBUF+1 ;
12F6	32E1E8	= 970	STA CODBUF+2 ; Kill data bytes in code buffer
		= 971	P2POPE: ;
12F9	3A5DE9	= 972	LDA BYTES ;
12FC	CD0013	= 973	CALL ADDPC ; Inc program count byte # bytes.
12FF	C9	= 974	RET ;
		= 975 ;	
		= 976 ;	End of module <ASMPASS2>.
		977	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 19

LOC	OBJ	LINE	SOURCE STATEMENT
		978 ;	
		979 ;	
		980 ;	
		981 ;	GENERAL ROUTINES.
		982 ;	
		983 \$INCLUDE(ASMGEN.A85)	
		= 984 ;	
		= 985 ;	
		= 986 ;	ASMGEN.A85 module for the UCT SABus assembler
		= 987 ;	
		= 988 ;	
		= 989 ;	This module contains (in order) the following subroutines:
		= 990 ;	
		= 991 ;	a) ADDPC - Add A register to the program counter.
		= 992 ;	b) ADRINS - Address insert.
		= 993 ;	c) DCMF - Compare HL and DE.
		= 994 ;	d) QEFILF - End of file ?
		= 995 ;	e) QEFU - Equate directive ?
		= 996 ;	f) QTABSP - Tab or Space ?
		= 997 ;	g) RULF - Read until line feed
		= 998 ;	h) RUNTOB - Read until not tab or blank
		= 999 ;	i) SEMILF - Semi colon or line feed ?
		=1000 ;	
		=1001 ;	
		=1002 ;	
		=1003 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 20

LOC	OBJ	LINE	SOURCE STATEMENT
		=1004 ;	=====
		=1005 ;*	*
		=1006 ;*	Function : ADDPC (Add accumulator to program counter). *
		=1007 ;*	Inputs : A : Number of bytes in the instruction. *
		=1008 ;*	Outputs : None. (PC in memory updated). *
		=1009 ;*	Destroys : Flags. *
		=1010 ;*	Calls : None. *
		=1011 ;*	*
		=1012 ;*	Description : This routine simply adds the number of bytes *
		=1013 ;*	in the instruction (passed in the A register) to the *
		=1014 ;*	program counter stored in memory. *
		=1015 ;*	*
		=1016 ;	=====
		=1017 ;	
		=1018 ADDPC:	; Start of add to PC routine
1300	E5	=1019	PUSH H ;
1301	D5	=1020	PUSH D ; Save pointers.
		=1021	;
1302	2A65E9	=1022	LHLD PC ; Fetch PC
1305	5F	=1023	MOV E,A ;
1306	1600	=1024	MVI D,0 ; DE = A (Number of bytes to add)
1308	19	=1025	DAD D ; HL = HL (* Old PC *) + DE (* Bytes to add *)
1309	2265E9	=1026	SHLD PC ; Save PC
		=1027	;
130C	D1	=1028	POP D ; Restore pointers
130D	E1	=1029	POP H ;
130E	C9	=1030	RET ; Return from <ADDPC>
		=1031 ;	
		=1032 ;	=====
		=1033 ;*	*
		=1034 ;*	Function : ADRINS (Insert address into symbol table). *
		=1035 ;*	Inputs : None. *
		=1036 ;*	Outputs : None. *
		=1037 ;*	Destroys : A and Flags. *
		=1038 ;*	Calls : INSERT *
		=1039 ;*	*
		=1040 ;*	Description : This routine is called when a line has a *
		=1041 ;*	label, the label string will have been copied into the *
		=1042 ;*	label buffer by the COPY routine. This routine checks to *
		=1043 ;*	see if the current program counter is defined and if it is *
		=1044 ;*	then the label in the label buffer is inserted into the *
		=1045 ;*	symbol table as having a value equal to the current PC. *
		=1046 ;*	*
		=1047 ;	=====
		=1048 ;	
		=1049 ADRINS:	; Start of <ADRINS> routine
130F	E5	=1050	PUSH H ;
1310	3A6BE9	=1051	LDA PCDEF ;
1313	FE00	=1052	CPI FALSE ;
1315	CA2113	=1053	JZ PCNDEF ; If PC defined then.
		=1054	;
1318	2A65E9	=1055	LHLD PC ;
131B	22E9E8	=1056	SHLD LABELB+6 ;
131E	CDA918	=1057	CALL INSERT ; Insert(Symbol_in_label_buffer,PC)
		=1058	;
		=1059 PCNDEF:	;
1321	E1	=1060	POP H ;
1322	C9	=1061	RET ; Return from <ADRINS>
		=1062 ;	
		=1063 ;	-----
		=1064 ;	
		=1065 ; FUNCTION :	DCMP
		=1066 ; INPUTS :	DE and HL to be compared
		=1067 ; OUTPUTS :	CY flag clear if HL >= DE
		=1068 ;	set if HL < DE
		=1069 ;	Z flag set if HL = DE
		=1070 ; CALLS :	NONE
		=1071 ; DESTROYS :	F/F's
		=1072 ; STACK USAGE :	2 bytes
		=1073 ; DESCRIPTION :	DCMP compares DE and HL (unsigned 16-bit comparison), setting

LOC	OBJ	LINE	SOURCE STATEMENT
		=1074 ;	the carry and zero flags according to the result of HL - DE.
		=1075 ;	
		=1076 DCMF:	
1323	C5	=1077	PUSH B ; Save BC so there's a place to put A
1324	47	=1078	MOV B,A ; Save A
1325	7D	=1079	MOV A,L ; Get lower byte of HL
1326	93	=1080	SUB E ; Subtract lower byte of DE
1327	4F	=1081	MOV C,A ; Save result in C
1328	7C	=1082	MOV A,H ; Upper byte of HL
1329	9A	=1083	SBB D ; Complete 16-bit subtraction - CY set/cleared
132A	DA2E13	=1084	JC DCP05 ; Carry flag set so result can't be zero
132D	B1	=1085	ORA C ; Set Z flag if HL = DE (CY clear)
		=1086 DCP05:	
132E	78	=1087	MOV A,B ; Restore A
132F	C1	=1088	POP B ; And BC
1330	C9	=1089	RET
		=1090 ;	
		=1091 ;=====	
		=1092 ;*	*
		=1093 ;*	Function : QEFILE (End of file ?).
		=1094 ;*	Inputs : None.
		=1095 ;*	Outputs : C = 1: No end of file.
		=1096 ;*	C = 0: End of file.
		=1097 ;*	Destroys : Flags.
		=1098 ;*	Calls : DCMF
		=1099 ;*	*
		=1100 ;*	Description : This routine checks for end of file by
		=1101 ;*	comparing the current assembly pointer and the end of text
		=1102 ;*	pointer. If the assembly pointer is greater than the end
		=1103 ;*	of text pointer then the end of file has been reached.
		=1104 ;*	*
		=1105 ;=====	
		=1106 ;	
		=1107 QEFILE:	; Start of <QEFILE> routine.
1331	E5	=1108	PUSH H ;
1332	D5	=1109	PUSH D ; Save pointers.
		=1110	;
1333	2AFEEF	=1111	LHLD EPOINT ;
1336	EB	=1112	XCHG ; Fetch the pointers, HL = Current asm pointer
1337	2A00E8	=1113	LHLD POINT ; DE = End of text pointer
		=1114	;
133A	CD2313	=1115	CALL DCMF ; Compare the pointers.
		=1116 ;	
		=1117 ;	If HL < DE then the carry flag is set.
		=1118 ;	i.e. The end of the file has been reached in the carry flag is clear.
		=1119 ;	
		=1120	;
133D	D1	=1121	POP D ; Restore pointers.
133E	E1	=1122	POP H ;
133F	C9	=1123	RET ; Return from <QEFILE>
		=1124 ;	
		=1125 ;=====	
		=1126 ;*	*
		=1127 ;*	Function : QEQU (Equate ?)
		=1128 ;*	Inputs : None.
		=1129 ;*	Outputs : Z = 1: If the string in the opcode buffer is
		=1130 ;*	'EQU' else Z = '0.
		=1131 ;*	Destroys : H,L,A and Flags.
		=1132 ;*	Calls : CMPOP
		=1133 ;*	*
		=1134 ;*	Description : This routine checks to see if the opcode
		=1135 ;*	string in the opcode buffer is the equate directive.
		=1136 ;*	*
		=1137 ;=====	
		=1138 ;	
		=1139 QEQU:	; Start of <QEQU> routine.
1340	21872E	=1140	LXI H,EQUSTR ; HL = Start of 4 character string 'EQU'
1343	CDF219	=1141	CALL CMPOP ; Compare with opcode in OPBUF.
1346	C9	=1142	RET ; Return with the result of the compare.
		=1143 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 22

LOC	OBJ	LINE	SOURCE STATEMENT
		=1144 ;	=====
		=1145 ;*	*
		=1146 ;*	Function : QTABSP (? TAB or Space)
		=1147 ;*	Inputs : Acc = Character to be checked.
		=1148 ;*	Outputs : Z = 1 : If char = Space or Tab, else Z=0.
		=1149 ;*	Destroys : Flags.
		=1150 ;*	Calls : None.
		=1151 ;*	*
		=1152 ;*	Description: This routine simply checks to see if the
		=1153 ;*	character in the acc is a space or a tab.
		=1154 ;*	*
		=1155 ;	=====
		=1156 ;	
		=1157 QTABSP:	; Start of ? Tab/space
1347	FE09	=1158 CPI TAB	;
1349	C8	=1159 RZ	; If TAB then return with Z = 1.
		=1160	;
134A	FE20	=1161 CPI SPACE	;
134C	C9	=1162 RET	; Compare with space and return.
		=1163 ;	
		=1164 ;	=====
		=1165 ;*	*
		=1166 ;*	Function : RULF (Read until line feed).
		=1167 ;*	Inputs : The assembly pointer <POINT> in memory.
		=1168 ;*	Outputs : An updated assembly pointer.
		=1169 ;*	Destroys : H,L,A and Flags.
		=1170 ;*	Calls : None.
		=1171 ;*	*
		=1172 ;*	Description : This routine scans the input line, in memory,
		=1173 ;*	until a line feed (The end of line marker) is found. The
		=1174 ;*	assembly pointer is placed one character passed the line
		=1175 ;*	feed, i.e the first character of the next line.
		=1176 ;*	*
		=1177 ;	=====
		=1178 ;	
		=1179 RULF:	; Start of <RULF> routine.
134D	2A00E8	=1180 LHLD POINT	; Fetch current asm pointer
		=1181 RULFL:	;
1350	7E	=1182 MOV A,M	; Fetch character (* Pointer = HL *)
1351	FE0A	=1183 CPI LF	; <LF> ?
1353	CA5A13	=1184 JZ EXRULF	; Yes - then exit, line feed found
		=1185	;
1356	23	=1186 INX H	; Inc text pointer
1357	C35013	=1187 JMP RULFL	; and loop
		=1188	;
		=1189 EXRULF:	; Read until line feed exit point
135A	23	=1190 INX H	; Inc to character after the line feed
135B	2200E8	=1191 SHLD POINT	; Save as new asm pointer.
135E	C9	=1192 RET	; Return from <RULF>
		=1193 ;	
		=1194 ;	=====
		=1195 ;*	*
		=1196 ;*	Function : RUNTOB (Read until not tab or blank).
		=1197 ;*	Inputs : HL = Current assembly pointer.
		=1198 ;*	Outputs : HL = First character that is not a space or a
		=1199 ;*	tab.
		=1200 ;*	Destroys : H,L,A and Flags.
		=1201 ;*	Calls : QTABSP
		=1202 ;*	*
		=1203 ;*	Description : This routine scans the input line looking for
		=1204 ;*	a character that is not a Tab or a Space. When it is found
		=1205 ;*	the pointer is returned in the HL register pair and the
		=1206 ;*	current assembly pointer <POINT> is updated.
		=1207 ;*	*
		=1208 ;	=====
		=1209 ;	
		=1210 ;	HL = Current assembly ptr.
		=1211 ;	
		=1212 RUNTOB:	; Start of <RUNTOB>
		=1213	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 23

LOC	OBJ	LINE	SOURCE STATEMENT
135F	7E	=1214	MOV A,M ; While TEXT(Ptr) = Space or TEXT(Ptr) = Tab do
1360	CD4713	=1215	CALL QTABSP ;
1363	C26A13	=1216	JNZ NSKPTS ;
		=1217	;
1366	23	=1218	INX H ; Ptr := Ptr + 1;
1367	C35F13	=1219	JMP RUNTOB ;
		=1220	;
		=1221	NSKPTS: ;
		=1222	;
136A	2200E8	=1223	SHLD POINT ; Update the current assembly pointer.
136D	C9	=1224	RET ; Return from <RUNTOB>
		=1225 ;	
		=1226 ;=====	
		=1227 ;*	
		=1228 ;* Function : SEMILF (? Semi-colon or Line feed).	*
		=1229 ;* Inputs : A = Character to check.	*
		=1230 ;* Outputs : Z = 1 : If character = <LF> or <'>.	*
		=1231 ;* Z = 0 : if not <LF> or <'>	*
		=1232 ;* Destroys : Flags.	*
		=1233 ;* Calls : None.	*
		=1234 ;*	*
		=1235 ;* Description: This routine checks to see if the char passed	*
		=1236 ;* to it is a <LF> or <'>.	*
		=1237 ;*	*
		=1238 ;=====	
		=1239 ;	
		=1240 SEMILF:	;
136E	FE3B	=1241	CPI ';' ; Start of Semi-colon or Line-feed routine
1370	C8	=1242	RZ ; Return zero if semi-colon
		=1243	;
1371	FE0A	=1244	CPI LF ;
1373	C9	=1245	RET ; Return zero if Line-feed
		=1246 ;	
		=1247 ;	End of <ASMGEN> module.
		1248	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 24

LOC	OBJ	LINE	SOURCE STATEMENT
		1249 ;	
		1250 ;	
		1251 ;	
		1252 ;	S T R I N G F U N C T I O N S (COPY/COPYL/QLET/QLEET/SWOPB)
		1253 ;	
		1254	\$INCLUDE(ASMCHARF.A85)
		=1255 ;	
		=1256 ;	
		=1257 ;	Character functions module for the UCT SABus assembler ;
		=1258 ;	----- ;
		=1259 ;	
		=1260 ;	This module contains (in order) the following subroutines: ;
		=1261 ;	
		=1262 ;	a) BBUF - Blank buffer ;
		=1263 ;	b) CONCOP - Control copy, copies a string from source text ;
		=1264 ;	into a buffer (buffer address passed as parameter). ;
		=1265 ;	c) CONLET - Convert a letter from lower to upper case. ;
		=1266 ;	d) COPY - Copy a string from the source text into a buffer. ;
		=1267 ;	e) COPYL - Copy a label from the source text into the label ;
		=1268 ;	buffer. ;
		=1269 ;	f) GETNSC - Get next string character. ;
		=1270 ;	g) QCONTAB - Search control table for a string in the opcode ;
		=1271 ;	buffer. ;
		=1272 ;	h) QLET - Letter ? ('A'..'Z', '@', '?') ;
		=1273 ;	i) QLEET - Label letter ? ('A'..'Z', '@', '?', '_', '0'..'9') ;
		=1274 ;	j) QTAB - Search table for the string in the opcode buffer. ;
		=1275 ;	k) SWOPB - Swop the opcode and temp_buffers. ;
		=1276 ;	
		=1277 ;	----- ;
		=1278 ;	
		=1279	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=1280 ;	=====
		=1281 ;*	*
		=1282 ;*	Function : BBUF (Blank buffer) *
		=1283 ;*	Inputs : HL = Start adr of buffer to blank. *
		=1284 ;*	Acc = Len of buffer. *
		=1285 ;*	Outputs : None. *
		=1286 ;*	Destroys : Acc and flags. *
		=1287 ;*	Calls : None. *
		=1288 ;*	*
		=1289 ;*	DESCRIPTION: This routines clears the buffer and fills *
		=1290 ;*	it with spaces. The start adr and length *
		=1291 ;*	of the buffer are passed in registers. *
		=1292 ;*	*
		=1293 ;	=====
		=1294 ;	
		=1295 BBUF:	; Start of Blank buffer.
1374	E5	=1296 PUSH H	; Save HL
		=1297 BBUFPL:	; Repeat
		=1298	; begin
1375	3620	=1299 MVI M, ' '	; Buffer(ptr) := Space;
1377	23	=1300 INX H	; Ptr := Ptr + 1;
1378	3D	=1301 DCR A	; Counter = counter - 1
1379	C27513	=1302 JNZ BBUFPL	; end
		=1303	; until Counter = 0
		=1304	;
137C	E1	=1305 POP H	; Restore HL.
137D	C9	=1306 RET	; Return from <BBUF>
		=1307 ;	
		=1308 ;	=====
		=1309 ;*	*
		=1310 ;*	Function : CONCOPY / COPY (Control + normal copy.) *
		=1311 ;*	Inputs : HL = Buffer to copy string into. *
		=1312 ;*	Current_text_ptr : POINT. *
		=1313 ;*	Outputs : None. *
		=1314 ;*	Destroys : All. *
		=1315 ;*	Calls : BBUF, CONLET and QLET. *
		=1316 ;*	*
		=1317 ;*	Description: This routine copies the string starting at *
		=1318 ;*	the current text_ptr into the string buf *
		=1319 ;*	starting at the PTR passed in HL. *
		=1320 ;*	*
		=1321 ;*	The Max length of the strings are as follows: *
		=1322 ;*	*
		=1323 ;*	Control string : Max len = 10 *
		=1324 ;*	Normal string : Max len = 6 *
		=1325 ;*	*
		=1326 ;*	Note: If the string in the text is longer than the buffer *
		=1327 ;*	then the string is read until a non valid string *
		=1328 ;*	char is found, but the extra characters are ignored *
		=1329 ;*	*
		=1330 ;*	Note: All characters are converted to upper case. *
		=1331 ;*	*
		=1332 ;*	This means that if a label is "Print_char" then only the *
		=1333 ;*	1st 6 characters "Print_" will be read into the string *
		=1334 ;*	buffer to be stored in the symbol table. *
		=1335 ;*	*
		=1336 ;	=====
		=1337 ;	
		=1338 CONCOPY:	; Start of control copy routine.
137E	3E0A	=1339 MVI A,10	; Setup long buffer length for control copy.
1380	C38513	=1340 JMP COPY05	;
		=1341 COPY:	; Normal COPY entry point.
1383	3E06	=1342 MVI A,6	; Setup buffer length.
		=1343 COPY05:	;
1385	47	=1344 MOV B,A	; Save buffer length in 'B'
1386	E5	=1345 PUSH H	; Save start of buffer.
1387	CD7413	=1346 CALL BBUF	; Blank buffe.
138A	EB	=1347 XCHG	; Move start adr of buffer to DE.
138B	2A00E8	=1348 LHLD POINT	;
138E	EB	=1349 XCHG	; Pointer now setup as follows.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 26

LOC	OBJ	LINE	SOURCE STATEMENT
		=1350	; HL = Start adr of buffer.
		=1351	; DE = Current asm pointer.
		=1352	MORETC:
138F	1A	=1353	LDAX D ; Get char from text.
1390	CD5C14	=1354	CALL QLET ; Letter ?
1393	C2AC13	=1355	JNZ COPYEX ; No - then Exit copy routine.
		=1356	;
1396	1A	=1357	LDAX D ; Restore letter.
1397	CDB213	=1358	CALL CONLET ; Convert Letter to Upper case.
139A	77	=1359	MOV M,A ; Save letter in output buffer.
139B	23	=1360	INX H ; Inc buffer pointer.
139C	13	=1361	INX D ; Inc text pointer.
139D	05	=1362	DCR B ; Dec char counter.
139E	C28F13	=1363	JNZ MORETC ; Loop if there is still space in the output
		=1364	; buffer i.e Char < Max_chars.
		=1365	SKIPNL:
		=1366	;
		=1367	; If buffer is full and there are still chars
		=1368	; in the string then read until non_letter
		=1369	; char found.
13A1	1A	=1370	LDAX D ; Get text char.
13A2	CD5C14	=1371	CALL QLET ; Letter ?
13A5	C2AC13	=1372	JNZ COPYEX ; No - Then exit copy.
		=1373	;
13A8	13	=1374	INX D ; Inc text pointer.
13A9	C3A113	=1375	JMP SKIPNL ; Loop until non letter char found.
		=1376	;
		=1377	COPYEX:
13AC	EB	=1378	XCHG ; Copy exit point.
13AD	2200E8	=1379	SHLD POINT ; Save new asm pointer.
		=1380	;
13B0	E1	=1381	POP H ; Pop start adr of the buffer from the stack.
13B1	C9	=1382	RET ; Return from <COPY>
		=1383 ;	
		=1384 ;=====	
		=1385 ;*	
		=1386 ;* Function : CONLET (CONvert LETter)	*
		=1387 ;* Inputs : Acc = Upper/lower case character.	*
		=1388 ;* Outputs : Acc = Upper case character.	*
		=1389 ;* Destroys : Acc and flags.	*
		=1390 ;* Calls : None.	*
		=1391 ;*	*
		=1392 ;* Description: This routine converts the character from	*
		=1393 ;* lower to upper case. (Only if the input	*
		=1394 ;* char is a lower case character, i.e.	*
		=1395 ;* char >= 'a' AND char <= 'z'.)	*
		=1396 ;*	*
		=1397 ;=====	
		=1398 ;	
		=1399	CONLET: ; Start of <CONLET>.
13B2	FE61	=1400	CPI 'a' ;
13B4	FABE13	=1401	JM CONLEX ; If char < 'a' then not lower case char.
		=1402	;
13B7	FE7B	=1403	CPI 'z' + 1 ;
13B9	F2BE13	=1404	JP CONLEX ; If char >= 'z'+1 then not lower case char
		=1405	;
		=1406	; else lower case.
13BC	C6E0	=1407	ADI 'A'-'a' ; Convert to Capital letter .
		=1408	;
		=1409	CONLEX: ;
13BE	C9	=1410	RET ; Return from <CONLET>
		=1411 ;	
		=1412 ;=====	
		=1413 ;*	*
		=1414 ;* Function : COPYL (Copy label)	*
		=1415 ;* Inputs : HL = Start adr of string buffer.	*
		=1416 ;* Current_memory_ptr : POINT.	*
		=1417 ;* Outputs : None.	*
		=1418 ;* Destroys : All.	*
		=1419 ;* Calls : BBUF, CONLET and QLET.	*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 27

LOC	OBJ	LINE	SOURCE STATEMENT
		=1420 ;*	
		=1421 ;*	Description: As above except that other Label_chars can
		=1422 ;*	occur in the string. (Uses QLLET instead
		=1423 ;*	of QLET). These 2 routines could be
		=1424 ;*	combined if extra space is needed.
		=1425 ;*	
		=1426 ;=====	
		=1427 ;	
		=1428 COPYL:	; Start of copy label routine.
13BF	E5	=1429	PUSH H ; Push start adr of buffer onto the stack.
13C0	3E06	=1430	MVI A,LABBL ; Set up buffer length.
13C2	CD7413	=1431	CALL BBUF ; Blank buffer.
		=1432	;
13C5	EB	=1433	XCHG ; Move start adr of buffer to DE.
13C6	2A00E8	=1434	LHLD POINT ;
13C9	EB	=1435	XCHG ;
		=1436	;
13CA	0606	=1437	MVI B,LABBL ; Max number of letters.
		=1438	;
		=1439 MORTC:	;
13CC	1A	=1440	LDAX D ;
13CD	CD7C14	=1441	CALL QLLET ;
13D0	C2E913	=1442	JNZ CPYEX ;
		=1443	;
13D3	1A	=1444	LDAX D ;
13D4	CDB213	=1445	CALL CONLET ; Convert letter to Upper case.
13D7	77	=1446	MOV M,A ;
13D8	23	=1447	INX H ;
13D9	13	=1448	INX D ;
13DA	05	=1449	DCR B ;
13DB	C2CC13	=1450	JNZ MORTC ;
		=1451	;
		=1452 SKIPNC:	;
13DE	1A	=1453	LDAX D ;
13DF	CD7C14	=1454	CALL QLLET ;
13E2	C2E913	=1455	JNZ CPYEX ;
		=1456	;
13E5	13	=1457	INX D ;
13E6	C3DE13	=1458	JMP SKIPNC ; Skip characters until valid temp char.
		=1459	;
		=1460 CPYEX:	;
13E9	EB	=1461	XCHG ;
13EA	2200E8	=1462	SHLD POINT ;
13ED	E1	=1463	POP H ; Pop start adr of the buffer from the stack.
13EE	C9	=1464	RET ; Return from <COPYL>
		=1465 ;	
		=1466 ;=====	
		=1467 ;*	
		=1468 ;*	Function : GETNSC (Get next string char)
		=1469 ;*	Inputs : HL = Asm pointer.
		=1470 ;*	Outputs : Z = 1 : Terminating quote found.
		=1471 ;*	Z = 0 and Acc = Char if terminating quote
		=1472 ;*	not found.
		=1473 ;*	HL = Next char in source file.
		=1474 ;*	Destroys : Acc and Flags.
		=1475 ;*	Calls : None.
		=1476 ;*	
		=1477 ;*	Description: This routine is used to read char within
		=1478 ;*	strings. If there is a Line_feed the asm pointer must not
		=1479 ;*	be incremented and of there is a double quote in the
		=1480 ;*	string it is not the terminating quote, but a quote in the
		=1481 ;*	string and is therefore returned as a quote.
		=1482 ;*	
		=1483 ;*	e.g. A string with a single quote = '''
		=1484 ;*	
		=1485 ;=====	
		=1486 ;	
		=1487 GETNSC:	; Start of Get next string char routine.
13EF	7E	=1488	MOV A,M ; Get 1st character from string.
13F0	FE0A	=1489	CPI LF ; Is it a line feed ?

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 28

LOC	OBJ	LINE	SOURCE	STATEMENT
13F2	CA0B14	=1490	JZ	NXTCEX ; Yes - then exit with LF in Acc.
		=1491		;
13F5	FE27	=1492	CPI	QUOTE ; Is it a quote character
13F7	CAFE13	=1493	JZ	QFOUND ; Yes - then Quote found in string.
		=1494		;
13FA	23	=1495	INX	H ; Else use char in Acc as next string char.
13FB	C30B14	=1496	JMP	NXTCEX ; Exit Next string char routine.
		=1497		;
		=1498	QFOUND:	;
13FE	23	=1499	INX	H ; Inc pointer to next character.
13FF	7E	=1500	MOV	A,M ; Get the next char from memory.
1400	FE27	=1501	CPI	QUOTE ; Another quote ?
1402	C20914	=1502	JNZ	NOSQ ; No second quote .
		=1503		;
1405	23	=1504	INX	H ; Inc pointer to char after 2nd quote.
1406	C30B14	=1505	JMP	NXTCEX ;
		=1506		;
		=1507	NOSQ:	;
		=1508		No second quote found . (i.e. end string)
1409	3E00	=1508	MVI	A,00H ;
		=1509	NXTCEX:	;
140B	B7	=1510	ORA	A ; Set flags before returning.
140C	C9	=1511	RET	;
		=1512		Return from <GETNSC>.
		=1513		;
		=1514		*****
		=1515		* Function : QCONTAB and QTAB (Checks for string in table.) *
		=1516		* Inputs : String in OPBuffer (Len = 6 or 10). *
		=1517		* Outputs : S = 0 and Acc = Position of the string in *
		=1518		* the buffer, if found. *
		=1519		* S = 1 : If string not found. *
		=1520		* Destroys : D,E,H,L,B,Acc and flags. *
		=1521		* Calls : None. *
		=1522		* *
		=1523		* Description: This routines uses a simple search to find the *
		=1524		* string (in the Opcode_buffer) in buffer string buffer *
		=1525		* given. In the case of QCONTAB the buffer is fixed and is *
		=1526		* the CONTAB (table of valid control strings). *
		=1527		* *
		=1528		* In the case of QTAB the start adr of the buffer is given *
		=1529		* in DE. The table has the following format: *
		=1530		* *
		=1531		* Start_of_table: <String #1>, EOT *
		=1532		* <String #2>, EOT *
		=1533		* ' ' *
		=1534		* <String #N>, EOT *
		=1535		* End_of_table_marker. *
		=1536		* *
		=1537		*****
		=1538		;
		=1539	QCONTAB:	;
		=1540		Control String entry point.
140D	115B17	=1540	LXI	D,CONTAB ; Get start adr of Control command
		=1541		Table.
1410	3E0A	=1542	MVI	A,CONBL ; Max len of control string = CONBL
1412	C31714	=1543	JMP	QTAB05 ; Continue with table search.
		=1544	QTAB:	;
		=1545		Normal ertry point for QTAB.
1415	3E06	=1545	MVI	A,LABBL ; Normal QTABLE function has str len=6
		=1546	QTAB05:	;
1417	0600	=1547	MVI	B,00H ; Set up string counter.
1419	326FE9	=1548	STA	QTABSL ; Save acc as Max str len.
		=1549	QTABL:	;
		=1550		Q TABLE Loop.
141C	2153E9	=1550	LXI	H,OPBUF ; Get start of string to search for.
141F	3A6FE9	=1551	LDA	QTABSL ; Get Max length of string.
1422	4F	=1552	MOV	C,A ; Put max len in 'C'
		=1553	QTABNB:	;
1423	1A	=1554	LDAX	D ; Get char from Table.
1424	FE04	=1555	CPI	EOT ;
1426	CA4914	=1556	JZ	SFOUND ; If char = EOT then String found.
		=1557		;
1429	FEFF	=1558	CPI	ETABLE ;
142B	CA4414	=1559	JZ	ENDTAB ; If char = End_table then Not found.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 29

LOC	OBJ	LINE	SOURCE STATEMENT
		=1560	
142E	BE	=1561	CMP M ; Compare with char in buffer.
142F	C23814	=1562	JNZ NOTE ; If Char # Char_in_buffer then
		=1563	; The strings are not equal.
1432	13	=1564	INX D ; Else Inc pointers
1433	23	=1565	INX H ;
1434	0D	=1566	DCR C ; Dec char counter.
1435	C32314	=1567	JMP QTABNB ; Loop.
		=1568	;
		=1569	NOTE: ; If not equal then
1438	13	=1570	INX D ; Inc pointer in table.
1439	1A	=1571	LDAX D ;
143A	FE04	=1572	CPI EOT ;
143C	C23814	=1573	JNZ NOTE ; Until Table_char = EOT.
		=1574	;
		=1575	ENDENT: ;
143F	13	=1576	INX D ;
1440	04	=1577	INR B ;
1441	C31C14	=1578	JMP QTABL ;
		=1579	;
		=1580	ENDTAB: ;
1444	3EFF	=1581	MVI A,OFFH ;
1446	C35A14	=1582	JMP QTABX ; Exit <QTAB>
		=1583	;
		=1584	SFOUND: ; String found.
1449	79	=1585	MOV A,C ; Load number of charaters left in OPBUF.
144A	B7	=1586	ORA A ;
144B	CA5914	=1587	JZ QTABEX ; If all chars equal then OK
		=1588	;
		=1589	CHKBLK: ; Else check that the rest of the buffer if
		=1590	; full of spaces.
144E	7E	=1591	MOV A,M ; Get char from buffer.
144F	FE20	=1592	CPI ' ' ; Space ?
1451	C23F14	=1593	JNZ ENDENT ; If char # Space then try the next entry
		=1594	; in the table.
		=1595	;
1454	23	=1596	INX H ; Inc pointer.
1455	0D	=1597	DCR C ; Dec chars_left counter.
1456	C24E14	=1598	JNZ CHKBLK ; Loop until chars_left counter = zero.
		=1599	;
		=1600	QTABEX: ; Exit point if String found in table.
1459	78	=1601	MOV A,B ;
		=1602	QTABX: ; ? Table exit point.
145A	B7	=1603	ORA A ;
145B	C9	=1604	RET ; Return from <QTAB>
		=1605	;
		=1606	;=====
		=1607	;*
		=1608	;* Function : QLET (? Valid Letter) *
		=1609	;* Inputs : Acc = Character to be checked. *
		=1610	;* Outputs : Z = 1 : If char is a label character. *
		=1611	;* Z = 0 : If char non label character. *
		=1612	;* Destroys : Acc and flags. *
		=1613	;* Calls : CONLET *
		=1614	;*
		=1615	;* Description: This routine is used to check that a char *
		=1616	;* is a valid starting letter in a label. *
		=1617	;*
		=1618	;* Note: The valid starting chars in labels are : *
		=1619	;*
		=1620	;* '@', '?', 'a' to 'z' and 'A' to 'Z'. *
		=1621	;*
		=1622	;=====
		=1623	;
		=1624	QLET: ; Start of <QLET>
145C	FE40	=1625	CPI '@' ;
145E	CA7314	=1626	JZ LETOK ; If Char = '@' then it's OK
		=1627	;
1461	FE3F	=1628	CPI '?' ;
1463	CA7314	=1629	JZ LETOK ; If Char = '?' then it's OK

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 30

LOC	OBJ	LINE	SOURCE STATEMENT
		=1630	
1466	CDB213	=1631	CALL CONLET ; Convert to upper case
1469	FE41	=1632	CPI 'A' ;
146B	FA7814	=1633	JM LETNOK ; If Char <= 'A' then it's not OK
		=1634	
146E	FE5B	=1635	CPI 'Z'+1 ;
1470	F27814	=1636	JP LETNOK ; If Char >= 'Z'+1 then it's not OK
		=1637	
		=1638	LETOK: ; Else it's OK.
1473	3E00	=1639	MVI A,00H ; Setup Zero = OK return.
1475	C37A14	=1640	JMP QLETEX ;
		=1641	
		=1642	LETNOK: ;
1478	3EFF	=1643	MVI A,OFFH ; Setup Non zero = Error return.
		=1644	QLETEX: ;
147A	B7	=1645	ORA A ;
147B	C9	=1646	RET ; Return from <QLET>
		=1647 ;	
		=1648 ;	=====
		=1649 ;	*
		=1650 ;	Function : QLELET (? Label Letter) *
		=1651 ;	Inputs : Acc = Character to be checked. *
		=1652 ;	Outputs : Z = 1: If character was a valid label char. *
		=1653 ;	Z = 0: If character non label char. *
		=1654 ;	Destroys : Acc and flags. *
		=1655 ;	Calls : QLELET *
		=1656 ;	*
		=1657 ;	Description: This routine is used to check if an input *
		=1658 ;	character is a valid label character or *
		=1659 ;	not. The valid label chars are as follows *
		=1660 ;	*
		=1661 ;	'a'-'z', 'A'-'Z', '0'-'9', '_', '@', '?' *
		=1662 ;	*
		=1663 ;	=====
		=1664 ;	
		=1665	QLELET: ; Start of <QLELET>
147C	C5	=1666	PUSH B ;
147D	47	=1667	MOV B,A ;
147E	CD5C14	=1668	CALL QLELET ; Test for letter.
1481	78	=1669	MOV A,B ;
1482	C1	=1670	POP B ;
		=1671	
1483	CA9D14	=1672	JZ QLELETX ; If char = Letter then Label_letter = true.
		=1673	
1486	FE5F	=1674	CPI '_' ; If char = '_' then Label_letter = true.
1488	CA9A14	=1675	JZ LLETOK ;
		=1676	
148B	FE30	=1677	CPI '0' ; If Char < '0' then Label_letter = false.
148D	FA9514	=1678	JM NOTLL ;
		=1679	
1490	FE3A	=1680	CPI '9'+1 ; If Char < '9'+1 then Label_letter = true.
1492	FA9A14	=1681	JM LLETOK ;
		=1682	
		=1683	NOTLL: ;
1495	3EFF	=1684	MVI A,OFFH ; Reset Zero flag => not a Label Letter.
1497	C39C14	=1685	JMP QLLX ; Exit (And set flags.)
		=1686	
		=1687	LLETOK: ;
149A	3E00	=1688	MVI A,00H ; Set zero flag => a label letter.
		=1689	QLLX: ;
149C	B7	=1690	ORA A ; Set flags and exit.
		=1691	QLELETX: ;
149D	C9	=1692	RET ; Return from <QLELET>
		=1693 ;	
		=1694 ;	=====
		=1695 ;	*
		=1696 ;	Function : SWOPB (Swop buffers) *
		=1697 ;	Inputs : None. *
		=1698 ;	Outputs : None. *
		=1699 ;	Destroys : None. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 31

LOC	OBJ	LINE	SOURCE STATEMENT
		=1700 ;*	Calls : None. *
		=1701 ;*	*
		=1702 ;*	DESCRIPTION: This routine swaps the 2 label buffers. *
		=1703 ;*	*
		=1704 ;	=====
		=1705 ;	
		=1706 SWOPB:	; Start of <SWOPB>
149E	F5	=1707	PUSH PSW ;
149F	E5	=1708	PUSH H ;
14A0	D5	=1709	PUSH D ;
14A1	C5	=1710	PUSH B ; Save all registers.
		=1711	;
14A2	21E3E8	=1712	LXI H,LABELB ; HL = Start of buffer #1
14A5	11EBE8	=1713	LXI D,TEMPLB ; DE = Start of buffer #2
14A8	0606	=1714	MVI B,LABELL ; B = Len of label buffer.
		=1715	;
		=1716 SWOPBL:	; For I = 1 to Label_buffer_len do
		=1717	; begin
14AA	4E	=1718	MOV C,M ; C_reg := Buffer1(I);
14AB	1A	=1719	LDAX D ; A_reg := Buffer2(I);
14AC	77	=1720	MOV M,A ; Buffer1(I) := A_reg ;
14AD	79	=1721	MOV A,C ; A_reg := C_reg;
14AE	12	=1722	STAX D ; Buffer2(I) := A_reg;
14AF	23	=1723	INX H ; Buffer1_ptr := Buffer1_ptr + 1;
14B0	13	=1724	INX D ; Buffer2_ptr := Buffer2_ptr + 1;
14B1	05	=1725	DCR B ;
14B2	C2AA14	=1726	JNZ SWOPBL ; end;
		=1727	;
14B5	C1	=1728	POP B ; Restore all registers
14B6	D1	=1729	POP D ;
14B7	E1	=1730	POP H ;
14B8	F1	=1731	POP PSW ;
		=1732	;
14B9	C9	=1733	RET ; Return from <SWOPB>
		=1734 ;	
		=1735 ;	End of Character functions module. (CHARFUNC.A85).
		1736 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 32

LOC	OBJ	LINE	SOURCE STATEMENT
		1737 ;	
		1738 ;	
		1739 ;	
		1740 ;	ASSEMBLER DIRECTIVES. (DW/DB/ORG/END/DS)
		1741 ;	
		1742	\$INCLUDE(ASMDIREC.A85)
		=1743 ;	
		=1744 ;	
		=1745 ;	Assembler directives module for the UCT SABus assembler
		=1746 ;	
		=1747 ;	
		=1748 ;	This module contains (in order) the following subroutines:
		=1749 ;	
		=1750 ;	a) DBGENB - Generate byte. (Used by the Define Byte routine)
		=1751 ;	b) DBROU - Define Byte routine.
		=1752 ;	c) DSROU - Define Storage routine.
		=1753 ;	d) DWGENW - Generate word. (Used by the Define Word routine)
		=1754 ;	e) DWROU - Define Word routine.
		=1755 ;	f) ENDROU - End directive routine.
		=1756 ;	g) EQUROU - Equate directive routine.
		=1757 ;	h) ORGROU - Origin directive routine.
		=1758 ;	
		=1759 ;	
		=1760 ;	
		=1761 ;	
		=1762 ;	
		=1763 ;	
		=1764 ;	Assembler directive routines.
		=1765 ;	
		=1766 ;	
		=1767 ;	a) EQU : Equate routine.
		=1768 ;	b) END : End of source file directive.
		=1769 ;	c) DS : Define storage directive.
		=1770 ;	d) ORG : Set origin point directive.
		=1771 ;	e) DW : Define word directive.
		=1772 ;	f) DB : Define byte directive.
		=1773 ;	
		=1774 ;	
		=1775 ;	
		=1776	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=1777 ;	
		=1778 ;=====	
		=1779 ;*	
		=1780 ;* Function : DBGENB (Define byte, Generate byte)	*
		=1781 ;* Inputs : Acc = Byte to Generate.	*
		=1782 ;* Outputs : None.	*
		=1783 ;* Destroys : Acc, B and flags.	*
		=1784 ;* Calls : CODE.	*
		=1785 ;*	*
		=1786 ;* Description: This routine handles the code generation for	*
		=1787 ;* the DB routine. As above if it's pass one then no code is	*
		=1788 ;* generated and only the PC is changed.	*
		=1789 ;*	*
		=1790 ;=====	
		=1791 ;	
		=1792 DBGENB:	; Gen byte value as data.
14BA	E5	=1793	PUSH H ; Save HL
14BB	47	=1794	MOV B,A ; Save Value in B register.
14BC	2A65E9	=1795	LHLD PC ; Get Program counter.
14BF	3A70E9	=1796	LDA PASSN ;
14C2	FE02	=1797	CPI 2 ;
14C4	C2CB14	=1798	JNZ SKIPCG ; If pass_number = 2 then
		=1799	;
14C7	78	=1800	MOV A,B ; Code(Data_byte);
14C8	CD8923	=1801	CALL CODE ;
		=1802 SKIPCG:	;
14CB	23	=1803	INX H ; Inc program counter.
14CC	2265E9	=1804	SHLD PC ;
14CF	E1	=1805	POP H ;
14D0	C9	=1806	RET ; Return from <DBGENB>
		=1807 ;	
		=1808 ;=====	
		=1809 ;*	*
		=1810 ;* Function : DBROU (Define byte routine.)	*
		=1811 ;* Inputs : Current_asm_ptr (POINT)	*
		=1812 ;* Outputs : None.	*
		=1813 ;* Destroys : All.	*
		=1814 ;* Calls : RUNTOB, REXP, DBGENB, GETNSC and PERR.	*
		=1815 ;*	*
		=1816 ;* Description: This routine handles the define byte (DB)	*
		=1817 ;* directive. There is one problem when evaluating a DB	*
		=1818 ;* directive, that is that there are 2 types of entries on a	*
		=1819 ;* DB line. These are 'strings' and 'expressions'.	*
		=1820 ;*	*
		=1821 ;* The problem is that a expression char start of looking	*
		=1822 ;* like a string.	*
		=1823 ;*	*
		=1824 ;* e.g. 'A' + 80H (Is an expression that starts like a string.	*
		=1825 ;*	*
		=1826 ;* The way the assembler knows is by the number of characters	*
		=1827 ;* in the string. If there is only one character then it is	*
		=1828 ;* treated as an expression.	*
		=1829 ;*	*
		=1830 ;* format:	*
		=1831 ;*	*
		=1832 ;* [<label>] DB <String/Exp>[,<String/Exp>, ...] [;<comment>]	*
		=1833 ;*	*
		=1834 ;=====	
		=1835 ;	
		=1836 DBROU:	; Start of Define Byte routine.
		=1837	;
14D1	2A00E8	=1838	LHLD POINT ; Fetch asm pointer
14D4	CD5F13	=1839	CALL RUNTOB ; Skip blanks and tabs
14D7	7E	=1840	MOV A,M ; Fetch character
14D8	CD6E13	=1841	CALL SEMILF ; Char = <LF> or ';' ?
14DB	CA7315	=1842	JZ NOEXPF ; Yes - then error (No data after DB)
		=1843 ;	
		=1844 ;-----	
		=1845 ;	
		=1846 NEXTDB:	; DB loop.

ASM LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 34

LOC	OBJ	LINE	SOURCE STATEMENT
14DE	FE27	=1847	CPI QUOTE ; Char = quote ?
14EO	CA1715	=1848	JZ DBSTR ; Yes - then the entry could be a string
		=1849	DBEXP: ;
14E3	CD1C24	=1850	CALL REXP ; Read expression
14E6	7C	=1851	MOV A,H ; Move high byte of the result to Areg
14E7	FE00	=1852	CPI 0 ; IF HIGH Result # 0 and HIGH Result # \$FF
14E9	CAFB14	=1853	JZ DBVALO ;
		=1854	;
14EC	FEFF	=1855	CPI OFFH ;
14EE	CAFB14	=1856	JZ DBVALO ;
		=1857	;
			Then
14F1	3EOA	=1858	MVI A,10 ; Begin
14F3	CD3E21	=1859	CALL PERR ; Perr (10)
14F6	3E00	=1860	MVI A,0 ; Data_byte := 0
14F8	C3FC14	=1861	JMP DBVALC ; End
		=1862	DBVALO: ; Else
14FB	7D	=1863	MOV A,L ; Data_byte := LOW Result ;
		=1864	DBVALC: ;
14FC	CDBA14	=1865	CALL DBGENB ; Generate code (Data_Byte)
14FF	2A00E8	=1866	LHLD POINT ; Load asm pointer
1502	CD5F13	=1867	CALL RUNTOB ; Skip blanks and tabs
1505	7E	=1868	MOV A,M ; Fetch character
1506	CD6E13	=1869	CALL SEMILF ; Char = LF or Char = ';' ?
1509	CA7815	=1870	JZ DBEXIT ; Yes then exit
		=1871	;
150C	23	=1872	INX H ; Skip terminator
150D	2200E8	=1873	SHLD POINT ;
		=1874	;
		=1875	;
		=1876	;
1510	CD5F13	=1877	CALL RUNTOB ;
1513	7E	=1878	MOV A,M ;
1514	C3DE14	=1879	JMP NEXTDB ; Loop for next section.
		=1880 ;	
		=1881 ;	
		=1882 ;	
		=1883 ;	If the 1st character in the DB entry is a quote then come here.
		=1884 ;	
		=1885	DBSTR: ; DB string.
1517	54	=1886	MOV D,H ;
1518	5D	=1887	MOV E,L ; Save pointer to 1st quote in DE.
1519	23	=1888	INX H ; Now points to the 1st char after the quote.
151A	CDEF13	=1889	CALL GETNSC ; Get the next char from the string.
151D	CA4D15	=1890	JZ ENDSTR ; Zero => end of string.
		=1891	;
1520	FE0A	=1892	CPI LF ;
1522	CA6315	=1893	JZ NRQ ; If line feed then no right quote.
		=1894	;
1525	4F	=1895	MOV C,A ; Save char just read in the C register.
1526	CDEF13	=1896	CALL GETNSC ; Get the next char.
1529	CA4615	=1897	JZ SCHAR ; If it is the goto single char.
		=1898	;
152C	F5	=1899	PUSH PSW ; Save char just read.
152D	79	=1900	MOV A,C ; Gen first byte.
152E	CDBA14	=1901	CALL DBGENB ;
1531	F1	=1902	POP PSW ; Restore char just read.
1532	CDBA14	=1903	CALL DBGENB ; Gen byte for second character.
		=1904	GENIT: ;
1535	CDEF13	=1905	CALL GETNSC ; Get another character
1538	CA4D15	=1906	JZ ENDSTR ;
		=1907	;
153B	FE0A	=1908	CPI LF ;
153D	CA6315	=1909	JZ NRQ ;
		=1910	;
1540	CDBA14	=1911	CALL DBGENB ; Generate byte.
1543	C33515	=1912	JMP GENIT ;
		=1913	;
		=1914 ;	
		=1915 ;	
		=1916 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 35

LOC	OBJ	LINE	SOURCE STATEMENT
		=1917 ;	If single character in quotes then treat it as an expression.
		=1918 ;	
		=1919 SCHAR:	
1546	EB	=1920	XCHG ; Restore pointer to 1st quote.
1547	2200E8	=1921	SHLD POINT ;
154A	C3E314	=1922	JMP DBEXP ; Treat the single char string as part of
		=1923	; an expression.
		=1924 ENDSTR:	
154D	CD5F13	=1925	CALL RUNTOB ; Read until not tab or blank
1550	7E	=1926	MOV A,M ; Fetch next character
1551	CD6E13	=1927	CALL SEMILF ; Semi-colon or linefeed ?
1554	CA7815	=1928	JZ DBEXIT ; Yes - then end of DB line
		=1929	
1557	FE2C	=1930	CPI ',' ; Comma ?
1559	C26B15	=1931	JNZ DBSTRE ; No - then error
		=1932	
155C	23	=1933	INX H ; Yes - skip comma and read next item on line.
155D	2200E8	=1934	SHLD POINT ; Point to next char after the comma.
1560	C3D114	=1935	JMP DBROU ; loop
		=1936 ;	
		=1937 ;	=====
		=1938 ;	
		=1939 ;	DB error exit points:
		=1940 ;	
		=1941 NRQ:	
1563	3E10	=1942	MVI A,16 ; No right quote found.
1565	CD3E21	=1943	CALL PERR ; Gen error #16
1568	C37815	=1944	JMP DBEXIT ; and exit
		=1945 ;	
		=1946 ;	-----
		=1947 ;	
		=1948 DBSTRE:	
156B	3E1E	=1949	MVI A,30 ; DB String error.
156D	CD3E21	=1950	CALL PERR ; Gen error #30
1570	C37815	=1951	JMP DBEXIT ; and exit
		=1952 ;	
		=1953 ;	-----
		=1954 ;	
		=1955 NOEXPF:	
1573	3E1D	=1956	MVI A,29 ; No Expression found (eq: FRED: DB ;Comment
1575	CD3E21	=1957	CALL PERR ; Gen error #29
		=1958 ;	
		=1959 ;	-----
		=1960 ;	
		=1961 DBEXIT:	
		=1962	; DB exit point
1578	C9	=1963	RET ; Return from <DBROU> routine
		=1964 ;	
		=1965 SEJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 36

LOC	OBJ	LINE	SOURCE STATEMENT
		=1966 ;	=====
		=1967 ;*	*
		=1968 ;*	Function : DSROU (Define storage routine)
		=1969 ;*	Inputs : Current_asm_ptr (POINT)
		=1970 ;*	Outputs : None.
		=1971 ;*	Destroys : All.
		=1972 ;*	Calls : RUNTOB, REXP, APUSH, APOP and PERR.
		=1973 ;*	*
		=1974 ;*	Description: This routine handles the DS directive.
		=1975 ;*	*
		=1976 ;*	format: [<label>] DS <exp> [<comment>]
		=1977 ;*	*
		=1978 ;*	If there is an error in the <exp> then the program
		=1979 ;*	counter is no longer defined and zero is added to the
		=1980 ;*	PC (As it is undefined it does not matter.)
		=1981 ;*	*
		=1982 ;*	If there is no error the Value of the <exp> is added to
		=1983 ;*	the PC, but no code is generated.
		=1984 ;*	*
		=1985 ;	=====
		=1986 ;	
		=1987 DSROU:	; Start of <DSROU>
1579	2A00E8	=1988 LHL	POINT ; Fetch asm pointer.
157C	CD5F13	=1989 CALL	RUNTOB ; Read_until_not_tab_or_blank.
157F	CD1C24	=1990 CALL	REXP ; Read expression from text.
1582	CA8D15	=1991 JZ	DSEXPO ; If there is no error in the expression
		=1992	; then leave PCDEF flag unchanged
1585	3E00	=1993 MVI	A,FALSE ;
1587	326BE9	=1994 STA	PCDEF ; PCDEF = False .
158A	210000	=1995 LXI	H,0000H ; Set DS Value = 0, i.e. do not inc PC.
		=1996	;
		=1997 DSEXPO:	;
158D	7C	=1998 MOV	A,H ; Load MSByte of value read.
158E	B7	=1999 ORA	A ; Set flags.
158F	F29A15	=2000 JP	DSVPOS ; If DS value is positive then DS value ok
		=2001	;
1592	3E05	=2002 MVI	A,5 ; Else error ("DS value must be positive")
1594	CD3E21	=2003 CALL	PERR ;
1597	210000	=2004 LXI	H,0000H ; Set DS value = 0000H
		=2005	;
		=2006 DSVPOS:	; DS value now OK.
		=2007	;
159A	EB	=2008 XCHG	; Move DS value to DE
159B	2A65E9	=2009 LHL	PC ; Load PC in HL
159E	19	=2010 DAD	D ;
		=2011	; HL = DS_value + Program_counter.
159F	2265E9	=2012 SHLD	PC ; Save as new PC.
15A2	3E00	=2013 MVI	A,FALSE ;
15A4	320EE8	=2014 STA	GCFLAG ; Gen_code_flag = False.
		=2015	;
15A7	C9	=2016 RET	; Return from <DSROU>
		=2017 ;	
		=2018 ;	=====
		=2019 ;*	*
		=2020 ;*	Function : DWGENW (Define word, Generate word)
		=2021 ;*	Inputs : HL = word to generate.
		=2022 ;*	Outputs : None.
		=2023 ;*	Destroys : Acc and flags.
		=2024 ;*	Calls : CODE.
		=2025 ;*	*
		=2026 ;*	Description: This routine is used to generate words for the
		=2027 ;*	DW routine. The word to Gen is passed in HL and the ADR is
		=2028 ;*	taken from PC and PC is incremented by 2. If it's pass 1
		=2029 ;*	then no code is generated and all that happens is that the
		=2030 ;*	PC is incremented.
		=2031 ;*	*
		=2032 ;	=====
		=2033 ;	
		=2034 DWGENW:	; Start of the <DWGENW> routine.
15A8	E5	=2035 PUSH	H ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 37

LOC	OBJ	LINE	SOURCE STATEMENT
15A9	D5	=2036	PUSH D ; Save H,L,D and E regs.
		=2037	
15AA	EB	=2038	XCHG ; Move value to DE.
15AB	2A65E9	=2039	LHLD PC ; Get current code adr.
15AE	3A70E9	=2040	LDA PASSN ;
15B1	FE01	=2041	CPI 1 ;
15B3	CAC315	=2042	JZ SKIPC ; If pass_number = 1 then skip_code_gen
		=2043	
15B6	7B	=2044	MOV A,E ; Acc = LSByte of Value to gen.
15B7	CD8923	=2045	CALL CODE ; Gen code byte.
15BA	23	=2046	INX H ; Inc PC.
15BB	7A	=2047	MOV A,D ; Acc = MSByte of value to gen.
15BC	CD8923	=2048	CALL CODE ; Gen code byte.
15BF	23	=2049	INX H ; Inc PC.
15C0	C3C515	=2050	JMP EXDWGW ; Exit DW_code_gen_routine.
		=2051	
		=2052	SKIPC: ; If pass_number = 1
15C3	23	=2053	INX H ; PC = PC + 2
15C4	23	=2054	INX H ;
		=2055	EXDWGW: ;
15C5	2265E9	=2056	SHLD PC ; Save HL as new PC.
		=2057	
15C8	D1	=2058	POP D ;
15C9	E1	=2059	POP H ; Restore H,L,D and E regs.
15CA	C9	=2060	RET ; Return from <DWGENW>
		=2061 ;	
		=2062 ;=====	
		=2063 ;*	
		=2064 ;* Function : DWROU (Define word routine.)	*
		=2065 ;* Inputs : Current_asm_ptr (POINT).	*
		=2066 ;* Outputs : None.	*
		=2067 ;* Destroys : None.	*
		=2068 ;* Calls : RUNTOB, REXP, PERR and DWGENW.	*
		=2069 ;*	*
		=2070 ;* Description: This routine handles the define word (DW)	*
		=2071 ;* directive. There must be at least 1 number in the operand	*
		=2072 ;* field and there can be more seperated by commas.	*
		=2073 ;*	*
		=2074 ;* format: [<label>] DW <exp>[,<exp>, ...] [<comment>]	*
		=2075 ;*	*
		=2076 ;=====	
		=2077 ;	
		=2078	DWROU: ; Start of <DWROU>
15CB	2A00E8	=2079	LHLD POINT ; Fetch Current_asm_ptr
15CE	CD5F13	=2080	CALL RUNTOB ; Read_until_not_tab_or_blank.
		=2081	
15D1	7E	=2082	MOV A,M ; Fetch 1st character of <Exp>
		=2083	
15D2	CD6E13	=2084	CALL SEMILF ; If Char in [';' , LF , ','] then
15D5	CAF415	=2085	JZ NOEXP ; No_exp_error.
15D8	FE2C	=2086	CPI ',' ;
15DA	CAF415	=2087	JZ NOEXP ;
		=2088	
15DD	CD1C24	=2089	CALL REXP ; Else read expression from text.
15E0	CDA815	=2090	CALL DWGENW ; gen_word.(Even if there is an error in
		=2091	the word, to keep the PC correct)
		=2092	
15E3	2A00E8	=2093	LHLD POINT ; Fetch asm_ptr (Points to REXP terminator)
		=2094	; i.e. one of the following [';' , ',' , LF]
15E6	7E	=2095	MOV A,M ; Fetch REXP terminator.
15E7	CD6E13	=2096	CALL SEMILF ;
15EA	CAF915	=2097	JZ DWEXIT ; If char = ';' or <LF> then exit DW_routine.
		=2098	
		=2099	; Else it must be a comma.(Therefore there
		=2100	must be another word to read.)
15ED	23	=2101	INX H ; Inc asm_ptr past the comma.
15EE	2200E8	=2102	SHLD POINT ; Update asm_ptr.
15F1	C3CB15	=2103	JMP DWROU ; The terminating char must be a comma
		=2104	; (To exit REXP the char must be one of the
		=2105	following [LF , ';' , ',']

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 38

LOC	OBJ	LINE	SOURCE STATEMENT
		=2106	; So inc past comma an read next exp.
		=2107 NOEXP:	;
15F4	3E18	=2108	MVI A,24 ; If no expression then gen error #24.
15F6	CD3E21	=2109	CALL PERR ;
		=2110	;
		=2111 DWEXIT:	;
15F9	C9	=2112	RET ; Return from <DWROU>
		=2113 ;	
		=2114 ;	=====
		=2115 ;*	*
		=2116 ;*	Function : ENDROU (End routine.) *
		=2117 ;*	Inputs : Current_asm_ptr (POINT). *
		=2118 ;*	Outputs : None. *
		=2119 ;*	Destroys : Acc and Flags. *
		=2120 ;*	Calls : None. *
		=2121 ;*	*
		=2122 ;*	Description: If an END directive is encountered in the text *
		=2123 ;*	then the pass_number is simply incremented to indicate *
		=2124 ;*	that the next pass should start. *
		=2125 ;*	*
		=2126 ;*	format: [<label>] END [<Symbol>] [;<comment>] *
		=2127 ;*	*
		=2128 ;	=====
		=2129 ;	
		=2130 ENDROU:	; Start of <ENDROU>
15FA	3A70E9	=2131	LDA PASSN ;
15FD	3C	=2132	INR A ; Pass_number = Pass_number + 1
15FE	3270E9	=2133	STA PASSN ;
1601	C9	=2134	RET ; Return from <ENDROU>
		=2135 ;	
		=2136 ;	=====
		=2137 ;*	*
		=2138 ;*	Function : EQUROU (EQUate ROUTine) *
		=2139 ;*	Inputs : Current_asm_ptr (POINT) *
		=2140 ;*	Outputs : None. *
		=2141 ;*	Destroys : All. *
		=2142 ;*	Calls : SWOPB, RUNTOB, REXP and INSERT. *
		=2143 ;*	*
		=2144 ;*	Description: This routine handles the EQU directive by *
		=2145 ;*	reading an expression from the chars following the EQU *
		=2146 ;*	code and then inserting the symbol into the Symbol table *
		=2147 ;*	if there was no error in the expression. *
		=2148 ;*	*
		=2149 ;*	format: <label> EQU <exp> [;<comment>] *
		=2150 ;*	*
		=2151 ;	=====
		=2152 ;	
		=2153 EQUROU:	; Start of Equate routine.
1602	CD9E14	=2154	CALL SWOPB ; Save Symbol to be defined in the Temp_label
		=2155	; buffer (The label_buffer will be used by
		=2156	; the read_expression routine.)
1605	2A00E8	=2157	LHLD POINT ; Fetch assembly pointer.
		=2158	;
1608	CD5F13	=2159	CALL RUNTOB ; Read_until_not_tab_or_blank.
160B	CD1C24	=2160	CALL REXP ; Read expression from the source.
160E	CD9E14	=2161	CALL SWOPB ; Restore symbol to be defined.
		=2162	;
1611	F5	=2163	PUSH PSW ; Save returned status.
1612	E5	=2164	PUSH H ; Save value returned.
		=2165	;
1613	CD2E18	=2166	CALL FETCH ; Fetch the value.
1616	C24C16	=2167	JNZ NODEF ; IF Undefined then put in symbol table.
		=2168 ;	
		=2169 ;	-----
		=2170 ;	
		=2171 ;	The symbol is already in the symbol table, check that it was not
		=2172 ;	defined during the current pass.
		=2173 ;	
1619	3A70E9	=2174	LDA PASSN ; Fetch the current pass number
161C	47	=2175	MOV B,A ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 39

LOC	OBJ	LINE	SOURCE STATEMENT
161D	3AF3E8	=2176	LDA PASDEF ; Fetch the defined pass number ?
1620	B8	=2177	CMP B ; Pass_defined = current_pass ?
1621	C22E16	=2178	JNZ DIFPAS ; No - Then check for phase error.
		=2179 ;	
		=2180 ;	
		=2181 ;	
		=2182 ;	Defined in this pass = Multiply defined symbol.
		=2183 ;	
1624	E1	=2184	POP H ;
1625	F1	=2185	POP PSW ; Tidy up stack.
		=2186	;
1626	3E04	=2187	MVI A,4 ;
1628	CD3E21	=2188	CALL PERR ; Print multiply defined system error.
162B	C35416	=2189	JMP EQUEND ;
		=2190 ;	
		=2191 ;	
		=2192 ;	
		=2193 ;	Defined in the previous pass, check to see if the values are the same ?
		=2194 ;	
		=2195	DIFPAS: ;
162E	E1	=2196	POP H ; Restore value from the stack
162F	F1	=2197	POP PSW ; Tidy stack
1630	EB	=2198	XCHG ; Move value to DE
1631	2AE9E8	=2199	LHLD LABELB+LABBL ; Get value of symbol just read.
1634	CD2313	=2200	CALL DCMP ; Values equal ?
1637	C24416	=2201	JNZ PHSERR ; No - then phase error
		=2202 ;	
		=2203 ;	
		=2204 ;	
		=2205 ;	Symbol was defined during the previous pass and had the same value.
		=2206 ;	Therefore do nothing. (Set MSBit of 1st char in indicate that it
		=2207 ;	was defined during the 2nd pass.)
		=2208 ;	
163A	2A63E9	=2209	LHLD SADR ; Get start adr of 1st character of entry.
163D	3E80	=2210	MVI A,MSB ;
163F	B6	=2211	ORA M ; Acc = 1st_char OR 1000000B
1640	77	=2212	MOV M,A ; 1st_char = Acc.
		=2213	;
1641	C35416	=2214	JMP EQUEND ; exit equ routine
		=2215 ;	
		=2216 ;	
		=2217 ;	
		=2218 ;	Phase error.
		=2219 ;	
		=2220	PHSERR: ;
1644	3E14	=2221	MVI A,20 ;
1646	CD3E21	=2222	CALL PERR ; Print "Phase error message"
1649	C35416	=2223	JMP EQUEND ; Exit equ routine
		=2224 ;	
		=2225 ;	
		=2226 ;	
		=2227 ;	Not defined.
		=2228 ;	
		=2229	NODEF: ;
164C	E1	=2230	POP H ; Restore expression value from the stack
164D	F1	=2231	POP PSW ; Restore the expression read status.
		=2232	;
164E	22E9E8	=2233	SHLD LABELB+6 ; Save value read in the symbol packet.
		=2234	;
1651	CCA918	=2235	CZ INSERT ; If the Rexp routine returned zero.
		=2236	;
		=2237	;(i.e. The were no errors in the expression
			then put it in the sym table.)
		=2238	EQUEND: ;
1654	C9	=2239	RET ; Return from <EQUOROU>
		=2240 ;	
		=2241 ;	
		=2242 ;	
		=2243 ;	* Function : ORGROU (Origin point routine). *
		=2244 ;	* Inputs : Current_asm_ptr (POINT). *
		=2245 ;	* Outputs : None. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 40

LOC	OBJ	LINE	SOURCE STATEMENT
		=2246 ;*	Destroys : All. *
		=2247 ;*	Calls : RUNTOB and REXP. *
		=2248 ;*	
		=2249 ;*	Description: This routine sets the new origin point for the *
		=2250 ;*	program. (i.e. It changes the PC). *
		=2251 ;*	
		=2252 ;*	If the value of the expression is undefined (i.e. There is *
		=2253 ;*	an error in it) then the PC becomes undefined. *
		=2254 ;*	
		=2255 ;*	format: [<label>] ORG <exp> [;<comment>] *
		=2256 ;*	
		=2257 ;	=====
		=2258 ;	
		=2259 ORGROU:	; Start of <ORGROU>
		=2260	;
1655	2A00E8	=2261	LHLD POINT ; Fetch current_asm_ptr
1658	CD5F13	=2262	CALL RUNTOB ; Read_until_no_tab_or_blank
165B	CD1C24	=2263	CALL REXP ; Read expression from text.
165E	C26F16	=2264	JNZ ORGERR ; Return if error in value read from the
		=2265	; text.
1661	3EFF	=2266	MVI A,TRUE ;
1663	326BE9	=2267	STA PCDEF ; PC_defined = True.
1666	2265E9	=2268	SHLD PC ; PC = Expression_value
1669	2267E9	=2269	SHLD LASTPC ; Last_PC = Expression_value
166C	C37416	=2270	JMP ORGREX ;
		=2271	;
		=2272 ORGERR:	;
166F	3E00	=2273	MVI A,FALSE ;
1671	326BE9	=2274	STA PCDEF ; PC_defined = False
		=2275 ORGREX:	;
1674	C9	=2276	RET ; Return from <ORGROU>
		=2277 ;	
		=2278 ;	End of directive routines. (ASMDIREC.A85)
		2279 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 41

LOC	OBJ	LINE	SOURCE STATEMENT
2280			;
2281			-----
2282			;
2283			CONTROL ROUTINES.
2284			;
2285			(\$LIST/\$LISTFILE/\$NOLIST/\$SYMBOLS/\$NOSYMBOLS/\$EJECT/\$PAGELENGTH)
2286			;
2287			\$INCLUDE(ASMCONT.A85)
=2288			-----
=2289			;
=2290			Controls module for the UCT SABus assembler
=2291			-----
=2292			;
=2293			This module contains the following subroutines:
=2294			;
=2295			a) CONTROL - This routine implements all controls.
=2296			;
=2297			(The following are all subroutines of CONTROL)
=2298			;
=2299			i) LISTRO - List routine.
=2300			ii) NLSTRO - No list routine.
=2301			iii) EJECTR - Page eject routine.
=2302			iv) SYMROU - Symbols routine.
=2303			v) NSYMRO - No symbols routine.
=2304			vi) LSTFIL - List file routine.
=2305			vii) OBJFRO - Object file routine.
=2306			viii) PAGLEN - Page length routine.
=2307			ix) BRIEFR - Brief line format routine.
=2308			x) NOBRRO - No brief line format routine.
=2309			;
=2310			b) FETCHN - Fetch file name from the source text.
=2311			c) OBJINIT - Initialize object output variables.
=2312			;
=2313			-----
=2314			;
=2315			;
=2316			;
=2317			-----
=2318			;
=2319			Controls
=2320			;
=2321			a) \$LIST (Start assembler listing.)
=2322			b) \$NOLIST (Stop assembler listing.)
=2323			c) \$EJECT (Start new page in listing file.)
=2324			d) \$SYMBOLS (Print symbol table after assembly.)
=2325			e) \$NOSYMBOLS (Do not print symbols after asm.)
=2326			f) \$LISTFILE(filename) (Send listing to a file on the PDP.)
=2327			g) \$OBJECTFILE(filename) (Send object to a file on the PDP.)
=2328			h) \$PAGELENGTH(exp) (Sent length of listing pages.)
=2329			i) \$BRIEF (set print line mode to brief)
=2330			j) \$NOBRIEF (Set print line mode to full)
=2331			;
=2332			-----
=2333			;
=2334			\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=2335 ;	=====
		=2336 ;*	
		=2337 ;*	Function : CONTROL. *
		=2338 ;*	Inputs : HL = Ptr to '\$' (Start of control). *
		=2339 ;*	Outputs : None. *
		=2340 ;*	Destroys : All. *
		=2341 ;*	Calls : CONCOPY,PERR,QCONTAB and 1 of the control *
		=2342 ;*	routines. *
		=2343 ;*	
		=2344 ;*	Description: This routine handles the assembler control *
		=2345 ;*	lines. (See above for Syntax.) *
		=2346 ;*	
		=2347 ;	=====
		=2348 ;	
		=2349	CONTROL: ; Main entry point for control lines.
		=2350	;
1675	23	=2351	INX H ; Inc asm ptr past '\$'
1676	2200E8	=2352	SHLD POINT ; Save assembly pointer.
		=2353	;
1679	3A70E9	=2354	LDA PASSN ;
167C	FE01	=2355	CPI 1 ;
167E	CAAD16	=2356	JZ CONTEX ; If Pass Number = 1 then ignore controls.
		=2357	;
1681	3EFF	=2358	MVI A,TRUE ;
1683	326EE9	=2359	STA CONFLG ; Set control line flag.
1686	3E00	=2360	MVI A,FALSE ;
1688	326AE9	=2361	STA PAFLAG ; Reset print adr flag.
		=2362	;
168B	2153E9	=2363	LXI H,OPBUF ;
168E	CD7E13	=2364	CALL CONCOF ; Copy the string into the Opcode_buffer.
1691	CD0D14	=2365	CALL QCONTAB ; Is the string in the opcode_buffer in the
		=2366	control_strings table ?
1694	FAA816	=2367	JM NOTCON ; No - Then it's not a valid control.
		=2368	; Else control number = Acc.
1697	2600	=2369	MVI H,0 ;
1699	6F	=2370	MOV L,A ; HL = Entry_number.
169A	29	=2371	DAD H ; HL = Entry_number * 2
169B	11AD17	=2372	LXI D,CONJT ;
169E	19	=2373	DAD D ; Add start adr of control jump table
		=2374	; to twice entry number.
169F	7E	=2375	MOV A,M ;
16A0	23	=2376	INX H ;
16A1	66	=2377	MOV H,M ;
16A2	6F	=2378	MOV L,A ; (HL)(HL+1) = Pointer to control routine.
16A3	11AD16	=2379	LXI D,CONTEX ;
16A6	D5	=2380	PUSH D ; Setup return adr.
16A7	E9	=2381	PCHL ; Indirect jump to control routine.
		=2382	;
		=2383	NOTCON: ; If not valid control then exit
16A8	3E1C	=2384	MVI A,28 ; print error message .
16AA	CD3E21	=2385	CALL PERR ;
		=2386	;
		=2387	CONTEX: ; Return point for Control directives.
		=2388	;
16AD	C9	=2389	RET ; Return from <CONTROL>
		=2390	;
		=2391 ;	
		=2392 ;	-----
		=2393 ;	
		=2394 ;	Routines for control directives.
		=2395 ;	
		=2396 ;	-----
		=2397 ;	
		=2398 ;	a) \$LIST control.
		=2399 ;	
		=2400	LISTRO: ; Start of control routine (a).
16AE	3EFF	=2401	MVI A,TRUE ;
16B0	3269E9	=2402	STA PFLAG ; Set print flag.
16B3	2A08E8	=2403	LHLD PAGNUM ;
16B6	7C	=2404	MOV A,H ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 43

LOC	OBJ	LINE	SOURCE STATEMENT
16B7	B5	=2405	ORA L ; If Page Number = 0 then print header.
16B8	C2C616	=2406	JNZ LSTREX ;
		=2407	;
16BB	3A20E8	=2408	LDA PDPFLG ;
16BE	FEFF	=2409	CPI TRUE ;
16C0	CCE31D	=2410	CZ HEADER ; If Pdpflg then Print_header.
		=2411	;
16C3	CD101E	=2412	CALL NEXTPL ;
		=2413	LSTREX: ; List routine Exit.
16C6	C9	=2414	RET ;
		=2415	;
		=2416	;
		=2417	-----
		=2418	;
		=2419	;
		=2420	b) \$NOLIST control.
		=2421	NLSTRO: ; Start of Control routine (b).
16C7	3E00	=2422	MVI A,FALSE ;
16C9	3269E9	=2423	STA PFLAG ;
16CC	C9	=2424	RET ;
		=2425	;
		=2426	-----
		=2427	;
		=2428	c) \$EJECT control.
		=2429	;
		=2430	EJECTR: ; Start of control routine (c).
16CD	3EFF	=2431	MVI A,TRUE ;
16CF	3205E8	=2432	STA NEWPAG ; Set new page Flag.
16D2	C9	=2433	RET ;
		=2434	;
		=2435	-----
		=2436	;
		=2437	d) \$SYMBOLS control.
		=2438	;
		=2439	SYMROU: ; Start of control routine (d).
16D3	3EFF	=2440	MVI A,TRUE ;
16D5	326CE9	=2441	STA SYMFLG ; Set symbol_print_flag.
16D8	C9	=2442	RET ;
		=2443	;
		=2444	-----
		=2445	;
		=2446	e) \$NOSYMBOLS control.
		=2447	;
		=2448	NSYMRO: ; Start of control routine (e).
16D9	3E00	=2449	MVI A,FALSE ;
16DB	326CE9	=2450	STA SYMFLG ; Reset symbol_print_flag.
16DE	C9	=2451	RET ;
		=2452	;
		=2453	-----
		=2454	;
		=2455	f) \$LISTFILE(File_name) control.
		=2456	;
		=2457	LSTFIL: ; Start of control routine (f).
		=2458	;
16DF	CDC117	=2459	CALL FETCHN ; Fetch a file name.
16E2	C2FA16	=2460	JNZ LSTFER ; If error then exit with error message.
		=2461	;
16E5	23	=2462	INX H ; Inc asm pointer to <LF> .
16E6	2200E8	=2463	SHLD POINT ;
		=2464	;
16E9	CD4A1E	=2465	CALL OPENFILE ; Open the output file.
16EC	C2FF16	=2466	JNZ LSTFEX ;
		=2467	;
16EF	3EFF	=2468	MVI A,TRUE ;
16F1	3220E8	=2469	STA PDPFLG ; Set FDP flag.
16F4	CDAE16	=2470	CALL LISTRO ;
16F7	C3FF16	=2471	JMP LSTFEX ;
		=2472	;
		=2473	LSTFER: ;
16FA	3E06	=2474	MVI A,6 ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 44

LOC	OBJ	LINE	SOURCE STATEMENT
16FC	CD3E21	=2475	CALL PERR ;
		=2476	LSTFEX: ;
16FF	C9	=2477	RET ;
		=2478	;
		=2479	-----
		=2480	;
		=2481	;
		=2482	g) \$OBJECTFILE(File_name) control.
		=2483	OBJFRO: ; Start of control routine (g).
1700	CDC117	=2484	CALL FETCHN ; Fetch a file name from the source text.
1703	C21317	=2485	JNZ OBJFER ; If error then exit with message.
		=2486	;
1706	23	=2487	INX H ; Inc asm pointer to <LF>.
1707	2200E8	=2488	SHLD POINT ;
		=2489	;
170A	CD4A1E	=2490	CALL OPENFILE ; Open the output file.
170D	CCF017	=2491	CZ OBJINIT ; If the file opened correctly then init
		=2492	object file output variables.
1710	C31817	=2493	JMP OBJFEX ;
		=2494	OBJFER: ;
1713	3E06	=2495	MVI A,6 ; Print error message
1715	CD3E21	=2496	CALL PERR ;
		=2497	OBJFEX: ;
1718	C9	=2498	RET ; Return from \$OBJECTFILE control routine.
		=2499	;
		=2500	-----
		=2501	;
		=2502	h) \$PAGELENGTH (Expression) control.
		=2503	;
		=2504	PAGLEN: ; Start of Control routine (h).
1719	2A00E8	=2505	LHLD POINT ;
171C	7E	=2506	MOV A,M ;
171D	FE28	=2507	CPI '(' ;
171F	C24417	=2508	JNZ PAGLE1 ; If 1st # '(' Then page_len syntax error.
		=2509	;
1722	23	=2510	INX H ; Inc asm_ptr to after '('
1723	2200E8	=2511	SHLD POINT ;
1726	CD5029	=2512	CALL READNU ; Read number from Text. (Into HL)
1729	EB	=2513	XCHG ; Move number read to DE.
172A	2A00E8	=2514	LHLD POINT ;
172D	7E	=2515	MOV A,M ; Get READNU terminator
172E	FE29	=2516	CPI ')' ; Termch = ')' ?
1730	C24417	=2517	JNZ PAGLE1 ; No - then syntax error.
		=2518	;
1733	7A	=2519	MOV A,D ;
1734	B7	=2520	ORA A ; Value read <= 255 ?
1735	C24417	=2521	JNZ PAGLE1 ; No - Then page_len value error.
		=2522	;
1738	7B	=2523	MOV A,E ;
1739	FE05	=2524	CPI 5 ; Value read < 5 ?
173B	FA4917	=2525	JM PAGLE2 ; Yes - Then value error.
		=2526	;
173E	320AE8	=2527	STA LPP ; Else set new Lines_per_page.
1741	C34E17	=2528	JMP PAGLEX ;
		=2529	;
		=2530	PAGLE1: ; Page Length Error #1 : Syntax error.
1744	3E1F	=2531	MVI A,31 ;
1746	C34B17	=2532	JMP PAGLE3 ;
		=2533	;
		=2534	PAGLE2: ; Page Length Error #2 : Number too small
1749	3E20	=2535	MVI A,32 ; (Min page length = 5)
		=2536	PAGLE3: ;
174B	CD3E21	=2537	CALL PERR ;
		=2538	;
		=2539	PAGLEX: ;
174E	C9	=2540	RET ; Return from \$PAGELENGTH control.
		=2541	;
		=2542	-----
		=2543	;
		=2544	i) \$BRIEF control.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 45

LOC	OBJ	LINE	SOURCE STATEMENT
		=2545 ;	
		=2546 BRIEFR:	
174F	3EFF	=2547 MVI A,TRUE	;
1751	320DE8	=2548 STA BRIEFF	; Set brief flag
1754	C9	=2549 RET	;
		=2550 ;	
		=2551 ;	
		=2552 ;	
		=2553 ;	J) \$NOBRIEF control.
		=2554 ;	
		=2555 NOBRRO:	;
1755	3E00	=2556 MVI A,FALSE	;
1757	320DE8	=2557 STA BRIEFF	; Reset brief flag
175A	C9	=2558 RET	;
		=2559 ;	
		=2560 ;	
		=2561 ;	
		=2562 ;	TABLE OF CONTROLS.
		=2563 ;	
		=2564 ;	
		=2565 ;	
		=2566 ;	
		=2567 CONTAB:	; Table of control commands.
175B	4C495354	=2568 DB 'LIST',EOT	;
	04		
1760	4E4F4C49	=2569 DB 'NOLIST',EOT	;
	535404		
1767	454A4543	=2570 DB 'EJECT',EOT	;
	5404		
176D	53594D42	=2571 DB 'SYMBOLS',EOT	;
	4F4C5304		
1775	4E4F5359	=2572 DB 'NOSYMBOLS',EOT	;
	4D424F4C		
	5304		
177F	4C495354	=2573 DB 'LISTFILE',EOT	;
	46494C45		
	04		
1788	4F424A45	=2574 DB 'OBJECTFILE',EOT	;
	43544649		
	4C4504		
1793	50414745	=2575 DB 'PAGELENGTH',EOT	;
	4C454E47		
	544804		
179E	42524945	=2576 DB 'BRIEF',EOT	;
	4604		
17A4	4E4F4252	=2577 DB 'NOBRIEF',EOT	;
	49454604		
17AC	FF	=2578 DB ETABLE	; End of control string table.
		=2579 ;	
		=2580 ;	Vector table for the above control strings.
		=2581 ;	
		=2582 CONJT:	; Control command jump table.
17AD	AE16	=2583 DW LISTRO	;
17AF	C716	=2584 DW NLSTRO	;
17B1	CD16	=2585 DW EJECTR	;
17B3	D316	=2586 DW SYMROU	;
17B5	D916	=2587 DW NSYMRO	;
17B7	DF16	=2588 DW LSTFIL	;
17B9	0017	=2589 DW OBJFRO	;
17BB	1917	=2590 DW PAGLEN	;
17BD	4F17	=2591 DW BRIEFR	;
17BF	5517	=2592 DW NOBRRO	;
		=2593 ;	
		=2594 ;	
		=2595 ;*	
		=2596 ;*	Function : FETCHN (Fetch file name)
		=2597 ;*	Inputs : Assembler pointer (in memory <POINT>).
		=2598 ;*	Outputs : Z = 1: File name read correctly.
		=2599 ;*	Z = 0: Error in file name.
		=2600 ;*	Destroys : B, H, L, D, E, A and flags.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 46

LOC	OBJ	LINE	SOURCE STATEMENT
		=2601 ;*	Calls : None.
		=2602 ;*	
		=2603 ;*	Description : This routine attempts to read a file name from *
		=2604 ;*	the source text. The format of the file name is given *
		=2605 ;*	below: *
		=2606 ;*	
		=2607 ;*	'(' <character>, <character> .. <character> ')'
		=2608 ;*	
		=2609 ;*	The maximum length of the file name (i.e. the number of *
		=2610 ;*	characters is 30. The characters are copied from the *
		=2611 ;*	source line into the file name buffer. On exit the last *
		=2612 ;*	character in the file name has it's top bit set to *
		=2613 ;*	indicate that it is the last character in the file name. *
		=2614 ;*	
		=2615 ;	=====
		=2616 ;	
		=2617	FETCHN: ; Start of the <FETCHN> routine
17C1	2A00E8	=2618	LHLD POINT ; HL = asm pointer
17C4	7E	=2619	MOV A,M ; Check that the next char = '('
17C5	FE28	=2620	CPI '(' ;
17C7	C2E217	=2621	JNZ FFNERR ; If not then file name error.
		=2622	;
17CA	23	=2623	INX H ; Inc pointer to the next character
17CB	1121E8	=2624	LXI D,FNAME ; DE = Start address of the file name buffer
17CE	061E	=2625	MVI B,30 ; Setup counter with max length of filename.
		=2626	FFN05: ;
17D0	7E	=2627	MOV A,M ; Get next char from text.
17D1	FE29	=2628	CPI ')' ; End of file name character ?
17D3	CAE717	=2629	JZ FFNEOF ; Yes - then exit and indicate OK
		=2630	;
17D6	FE0A	=2631	CPI LF ; End of line reached ?
17D8	CAE217	=2632	JZ FFNERR ; Yes - then control syntax error
		=2633	;
17DB	12	=2634	STAX D ; Store the character in the file name buffer
17DC	23	=2635	INX H ; Inc asm pointer
17DD	13	=2636	INX D ; Inc file name buffer pointer
17DE	05	=2637	DCR B ; Dec character counter
17DF	C2D017	=2638	JNZ FFN05 ; Loop if less than 30 characters
		=2639 ;	
		=2640 ;	-----
		=2641 ;	
		=2642 ;	Fetch file name error exit pointer (return with zero flag clear)
		=2643 ;	
		=2644	FFNERR: ;
17E2	3EFF	=2645	MVI A,OFFH ; A = 255;
17E4	C3EE17	=2646	JMP FFNRET ; Exit from the fetch file name routine
		=2647 ;	
		=2648 ;	-----
		=2649 ;	
		=2650 ;	Fetch file name exit pointer (return with zero flag set)
		=2651 ;	
		=2652	FFNEOF: ;
17E7	1B	=2653	DCX D ; Dec pointer to last character in the filename
17E8	1A	=2654	LDAX D ;
17E9	F680	=2655	ORI MSB ; Set MSBit => end of filename.
17EB	12	=2656	STAX D ;
		=2657	;
17EC	3E00	=2658	MVI A,000H ; A = 0
		=2659	FFNRET: ;
17EE	B7	=2660	ORA A ; Set flags before returning.
17EF	C9	=2661	RET ; Return from the <FETCHN> routine.
		=2662 ;	
		=2663 ;	=====
		=2664 ;*	
		=2665 ;*	Function : OBJINIT (Object output init)
		=2666 ;*	Inputs : None.
		=2667 ;*	Outputs : None.
		=2668 ;*	Destroys : A, H, and L.
		=2669 ;*	Calls : None.
		=2670 ;*	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 47

LOC	OBJ	LINE	SOURCE STATEMENT
		=2671 ;	* Description : This routine is called to initialize the
		=2672 ;	* variables used by the object output routine.
		=2673 ;	*
		=2674 ;	=====
		=2675 ;	
		=2676	OBJINIT: ; Start of <OBJINIT> routine.
		=2677	;
17F0	3EFF	=2678	MVI A,TRUE ;
17F2	32C7E8	=2679	STA OBJFLG ; Set OBJECT flag.
		=2680	;
		=2681 ;	
		=2682 ;	Init object file variables.
		=2683 ;	
17F5	21C8E8	=2684	LXI H,OBJBUF ;
17F8	22D8E8	=2685	SHLD OBJBP ; Object_buffer_ptr = Object_buffer.
		=2686	;
17FB	3E00	=2687	MVI A,0 ;
17FD	32DEE8	=2688	STA OBJC ; Object_byte_counter = 0
		=2689	;
1800	210000	=2690	LXI H,0000H ;
1803	22DAE8	=2691	SHLD OBJBA ; Object_base_adr = 0000H
1806	22DCE8	=2692	SHLD OBJNA ; Object_next_adr = 0000H
		=2693	;
1809	C9	=2694	RET ; Return from <OBJINIT> routine.
		=2695 ;	
		=2696 ;	End of control routines (CONTROL.A85)
		2697	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 48

LOC	OBJ	LINE	SOURCE STATEMENT
		2698 ;	
		2699 ;	-----
		2700 ;	
		2701 ;	SYMBOL TABLE ROUTINES.
		2702 ;	
		2703	\$INCLUDE(ASMSTABLE.A85)
		=2704 ;	-----
		=2705 ;	
		=2706 ;	Symbol table module for the UCT SABus assembler
		=2707 ;	-----
		=2708 ;	
		=2709 ;	This module contains (in order) the following subroutines:
		=2710 ;	
		=2711 ;	a) CMPLAB - Compare labels.
		=2712 ;	b) CONVHV - Convert hash value to label address.
		=2713 ;	c) FETCH - Lookup symbol in symbol table.
		=2714 ;	d) HASH - Hash a label to a 8 bit value.
		=2715 ;	e) INSERT - Insert a label and value into the symbol table.
		=2716 ;	f) NEXTHV - Next hash value routine.
		=2717 ;	g) TABINT - Initialize the symbol table.
		=2718 ;	
		=2719 ;	-----
		=2720 ;	
		=2721 ;	
		=2722 ;	
		=2723 ;	-----
		=2724 ;	
		=2725 ;	Symbol table size and format information:
		=2726 ;	-----
		=2727 ;	
		=2728 ;	The assembler maintains a 256 entry symbol table. This table is used
		=2729 ;	by the assembler to store symbols and their values (e.g. if the
		=2730 ;	user defined the symbol 'LF' as having the value 0AH then this
		=2731 ;	information would be stored in the symbol table.
		=2732 ;	
		=2733 ;	The table consists of 256 8 byte entries, the format of each entry
		=2734 ;	is as follows:
		=2735 ;	
		=2736 ;	<Character1> .. <Character2> <LSByte of value> <MSByte of value>
		=2737 ;	
		=2738 ;	Notes: a) The sign bit of the 1st character indicates on which pass
		=2739 ;	the symbol was defined. (0=Pass 1, 1=Pass 2).
		=2740 ;	
		=2741 ;	b) The label buffer is space filled.
		=2742 ;	
		=2743 ;	The symbol lookup technique used is hashing. The hash function was
		=2744 ;	derived empirical.
		=2745 ;	
		=2746 ;	-----
		=2747 ;	
		=2748	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 49

LOC	OBJ	LINE	SOURCE STATEMENT
		=2749	;=====
		=2750	;* *
		=2751	;* Function : CMPLAB (Compare label) *
		=2752	;* Inputs : HL = Start adr of an entry in the S table. *
		=2753	;* and a Label in the Label_buffer. *
		=2754	;* Outputs : Z = 1 : Entry label = label in buffer. *
		=2755	;* Z = 0 : Not equal. *
		=2756	;* Destroys : D, E, H, L, B, C, Acc and Flags. *
		=2757	;* Calls : None. *
		=2758	;* *
		=2759	;* Description: This routine compares the label in the *
		=2760	;* label_buffer with the label pointed to by HL (i.e. An *
		=2761	;* entry in the S table). *
		=2762	;* *
		=2763	;=====
		=2764	;
		=2765	CMPLAB: ; Start of <CMPLAB>
180A	11E3E8	=2766	LXI D,LABELB ; Load DE With start of label buffer.
180D	0606	=2767	MVI B,LABBL ; Load B with length of label.
		=2768	CNXTCH: ;
180F	1A	=2769	LDAX D ; Load acc with char from label buffer.
1810	E67F	=2770	ANI OFFMSB ;
1812	4F	=2771	MOV C,A ; Save 7 bit data in C register.
1813	7E	=2772	MOV A,M ;
1814	E67F	=2773	ANI OFFMSB ; Mask off lower 7 bits.
1816	B9	=2774	CMP C ;
1817	C22018	=2775	JNZ CMPLX ; No - then the labels are not the same.
		=2776	;
181A	13	=2777	INX D ; Inc pointer in label buffer.
181B	23	=2778	INX H ; Inc pointer in symbol table.
181C	05	=2779	DCR B ; Dec byte counter.
181D	C20F18	=2780	JNZ CNXTCH ; Loop until all chars have been checked.
		=2781	CMPLX: ;
1820	C9	=2782	RET ; Return after all characters found to
		=2783	; be equal (zero flag set).
		=2784	;
		=2785	;=====
		=2786	;* *
		=2787	;* Function : CONVHV (Convert Hash value to start adr) *
		=2788	;* Inputs : Acc = Hash value. *
		=2789	;* Outputs : HL = Start adr of the entry in the table. *
		=2790	;* Destroys : H and L *
		=2791	;* Calls : None. *
		=2792	;* *
		=2793	;* Description: This routine converts the hash value in the *
		=2794	;* acc to the start adr of the entry in the symbol table. *
		=2795	;* (The start adr of the symbol table = <STABLE> and the *
		=2796	;* length of each entry is <WORD>+<LABBL> = 8. *
		=2797	;* *
		=2798	;=====
		=2799	;
		=2800	CONVHV: ; Start of <CONVHV>
1821	D5	=2801	PUSH D ; Save DE registers.
1822	1100E0	=2802	LXI D,STABLE ; Get start adr of Hash Table.
1825	6F	=2803	MOV L,A ;
1826	2600	=2804	MVI H,0 ; HL = Hash value.
		=2805	;
1828	29	=2806	DAD H ;
1829	29	=2807	DAD H ;
182A	29	=2808	DAD H ; HL = 8 * Hash value
		=2809	;
182B	19	=2810	DAD D ; HL = 8 * Hash value + Adr (Hash_table)
		=2811	;
182C	D1	=2812	POP D ; Restore DE registers.
182D	C9	=2813	RET ; Return from <CONVHV>
		=2814	;
		=2815	;=====
		=2816	;* *
		=2817	;* Function : FETCH *
		=2818	;* Inputs : Label (In Label_buffer, length LABBL) *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 50

LOC	OBJ	LINE	SOURCE STATEMENT
		=2819 ;*	Outputs : Z = 0 : If label not found in table. *
		=2820 ;*	Z = 1 : If the label is in the table. *
		=2821 ;*	HL and (Label+6) = Value of label *
		=2822 ;*	
		=2823 ;*	Destroys : A, B, H, L, D, E and flags. *
		=2824 ;*	Calls : HASH, CONVHV and CMPLAB. *
		=2825 ;*	
		=2826 ;*	Description: This routine is the symbol table fetch routine *
		=2827 ;*	that is used to lookup symbols in the symbol table. The *
		=2828 ;*	search system used is Hashing with a quadratic search on *
		=2829 ;*	clashes. The entire table is set to zero when the *
		=2830 ;*	assembler is run so that a slot with the 1st char = Null *
		=2831 ;*	(00H) is not in use. *
		=2832 ;*	
		=2833 ;	=====
		=2834 ;	
		=2835	FETCH: ; Start of symbol table fetch routine.
182E	3E00	=2836	MVI A,00H ;
1830	32F2E8	=2837	STA HVINC ; Zero hash value inc.
1833	CD7018	=2838	CALL HASH ; Hash label in buffer Acc = HASH_Value.
1836	32F1E8	=2839	STA HASHV ; Store hash value.
		=2840	;
		=2841	TRYA: ;
1839	CD2118	=2842	CALL CONVHV ; Convert hash value to start adr of label.
183C	2263E9	=2843	SHLD SADR ; Save start address of entry in symbol table.
183F	7E	=2844	MOV A,M ; Load 1st byte of label in symbol table
1840	B7	=2845	ORA A ; Set flags.
1841	CA6C18	=2846	JZ STEREX ; If byte zero the no label.
		=2847	;
1844	CD0A18	=2848	CALL CMPLAB ; Compare label in table and buffer.
1847	C26618	=2849	JNZ CLASH ; If labels not equal then clash.
		=2850	;
		=2851	; Else fetch the Value of the Label (HL)
		=2852 ;	
		=2853 ;	-----
		=2854 ;	
		=2855 ;	Come here if label defined (Setup value in lookup packed and
		=2856 ;	calculate defined pass from sign bit of first character.
		=2857 ;	
		=2858	NOREX: ;
184A	5E	=2859	MOV E,M ; Load MSByte of value from table (D).
184B	23	=2860	INX H ; Inc pointer to LSByte .
184C	56	=2861	MOV D,M ; Load LSByte of value from table (E).
184D	EB	=2862	XCHG ; DE <-> HL . (Value now in HL).
184E	22E9E8	=2863	SHLD LABELB+LABBL ; Store value in fetch calling block.
		=2864	;
1851	2A63E9	=2865	LHLD SADR ; fetch start adr of entry.
1854	7E	=2866	MOV A,M ; Get 1st character of the entry.
1855	E680	=2867	ANI MSB ; Check the MSBit.
1857	3E01	=2868	MVI A,1 ; If MSBit clear then
1859	CA5E18	=2869	JZ P1DEFN ; Acc = 1
		=2870	; else
185C	3E02	=2871	MVI A,2 ; Acc = 2.
		=2872	P1DEFN: ;
185E	32F3E8	=2873	STA PASDEF ; Pass_defined = Acc.
		=2874	;
1861	3E00	=2875	MVI A,00H ; Zero Acc. (Z=1 , fetch OK)
1863	C36E18	=2876	JMP FETCHE ; Exit fetch routine.
		=2877 ;	
		=2878 ;	-----
		=2879 ;	
		=2880 ;	Come here if the hashed label position is occupied but not the same
		=2881 ;	as the label in the label buffer. (A clash)
		=2882 ;	
		=2883 ;	Start implementation of quadratic search. .
		=2884 ;	
		=2885	CLASH: ;
1866	CDFC18	=2886	CALL NEXTHV ; Calculate the next hash value.
1869	C23918	=2887	JNZ TRYA ; Loop if not end of search.
		=2888 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 51

LOC	OBJ	LINE	SOURCE STATEMENT
		=2889 ;	-----
		=2890 ;	
		=2891 ;	Fetch symbol error exit point. (Symbol undefined)
		=2892 ;	
		=2893 STEREX:	; Fetch Error exit point.
186C	3EFF	=2894 MVI A,0FFH	; Set Acc # Zero (=> Fetch error)
		=2895 ;	
		=2896 ;	-----
		=2897 ;	
		=2898 ;	Fetch symbol exit point. Set flags and return.
		=2899 ;	
		=2900 FETCHE:	; Fetch Exit point.
186E	B7	=2901 ORA A	; Set Flags before returning.
186F	C9	=2902 RET	; Return from <FETCH>
		=2903 ;	
		=2904 ;	=====
		=2905 ;*	*
		=2906 ;*	* Function : HASH (Hashing function routine) *
		=2907 ;*	* Inputs : Label in Label_buffer, LABBL chars long. *
		=2908 ;*	* Outputs : Hash_value in Acc. (Range 000H to 0FFH) *
		=2909 ;*	* Destroys : H, L, A and flags *
		=2910 ;*	* Calls : None. *
		=2911 ;*	*
		=2912 ;*	* Description: This routine hashes the label in the label *
		=2913 ;*	* buffer and returns the hash value in the acc. The most *
		=2914 ;*	* important thing about this routine is that it's fast as it *
		=2915 ;*	* is called a large number of times. *
		=2916 ;*	*
		=2917 ;	=====
		=2918 ;	
		=2919 HASH:	
1870	C5	=2920 PUSH B	;
1871	D5	=2921 PUSH D	;
1872	21E3E8	=2922 LXI H,LABELB	; Get start address of label buffer
1875	11FFFF	=2923 LXI D,0FFFFH	; Initialise the CRC to all 1's
1878	0606	=2924 MVI B,LABBL	; Put the block length into B
		=2925 NXTCRC:	;
187A	7E	=2926 MOV A,M	; Get the data byte
187B	23	=2927 INX H	; Point to the next byte
		=2928 ;	
		=2929 ;	The following code shifts a single byte in A into the CRC held in
		=2930 ;	D,E. The CCITT polynomial (x16 + x12 + x5 + 1) is used. A high-
		=2931 ;	speed method is employed which shifts a whole byte into the CRC
		=2932 ;	in a single pass.
		=2933 ;	
187C	AA	=2934 XRA D	; Make E7..E0
187D	4F	=2935 MOV C,A	; Save it temporarily
187E	0F	=2936 RRC	; Shift E7..E4 down to the LS nibble
187F	0F	=2937 RRC	;
1880	0F	=2938 RRC	;
1881	0F	=2939 RRC	;
1882	E60F	=2940 ANI 00001111B	; Isolate the E7..E4 bits
1884	A9	=2941 XRA C	; Make F7..F0 in A
1885	4F	=2942 MOV C,A	; Save F7..F0 in C
1886	0F	=2943 RRC	; Move F7..F3 down
1887	0F	=2944 RRC	;
1888	0F	=2945 RRC	;
1889	57	=2946 MOV D,A	; Save the rotated F7..F0 for later
188A	0F	=2947 RRC	;
188B	E6F0	=2948 ANI 11110000B	; Isolate F3..F0
188D	AB	=2949 XRA E	; XOR with the LSB of the CRC
188E	AA	=2950 XRA D	; XOR with the rotated F2..F0,F7..F3
188F	5F	=2951 MOV E,A	; Put the result back for later
1890	7A	=2952 MOV A,D	; Get the rotated F7..F0 back
1891	53	=2953 MOV D,E	; Juggle a bit with saved values
1892	E6E0	=2954 ANI 11100000B	;
1894	5F	=2955 MOV E,A	; Save for later
1895	AA	=2956 XRA D	; Undo F2..F0 from the MS byte
1896	57	=2957 MOV D,A	; Save the MS byte again
1897	7B	=2958 MOV A,E	; Get F2..F0 back

LOC	OBJ	LINE	SOURCE STATEMENT
1898	A9	=2959	XRA C ; XOR it to form the LS byte
1899	5F	=2960	MOV E,A ; Saving it..
		=2961 ;	
		=2962 ;	The updated CRC is now in D,E.
		=2963 ;	
189A	05	=2964	DCR B ; At the end yet?
189B	C27A18	=2965	JNZ NXXCRC ; No - shift the next byte in
189E	7A	=2966	MOV A,D ; Then return the CRC in complemented form
189F	2F	=2967	CMA ;
18A0	57	=2968	MOV D,A ;
18A1	7B	=2969	MOV A,E ;
18A2	2F	=2970	CMA ;
18A3	5F	=2971	MOV E,A ;
18A4	7A	=2972	MOV A,D ;
18A5	AB	=2973	XRA E ;
18A6	D1	=2974	POP D ;
18A7	C1	=2975	POP B ;
18A8	C9	=2976	RET ; Return with the hash value in A.
		=2977 ;	
		=2978 ;	=====
		=2979 ;*	*
		=2980 ;*	Function : INSERT (Symbol table insert routine.) *
		=2981 ;*	Inputs : Label and value in Label_buffer. *
		=2982 ;*	Outputs : Z = 1 : Insert OK (Table not full or *
		=2983 ;*	multiply defined label) *
		=2984 ;*	Z = 0 : Insert failed. *
		=2985 ;*	Destroys : *
		=2986 ;*	Calls : NEXTHV, HASH, CONVHV and CMPLAB. *
		=2987 ;*	*
		=2988 ;*	Description: This routine is used to insert symbols into *
		=2989 ;*	the symbol table. The routine implements hashing and a *
		=2990 ;*	quadratic search on a clash. *
		=2991 ;*	*
		=2992 ;	=====
		=2993 ;	
		=2994	INSERT: ; Start of Insert routine.
18A9	3E00	=2995	MVI A,00H ;
18AB	32F2E8	=2996	STA HVINC ; Zero Hash_value_inc
18AE	CD7018	=2997	CALL HASH ; Hash label.
18B1	32F1E8	=2998	STA HASHV ; Store hash value .
		=2999	TRYAI: ;
18B4	CD2118	=3000	CALL CONVHV ; Convert to start adr of label in table.
18B7	7E	=3001	MOV A,M ; Load 1st byte from label.
18B8	FE00	=3002	CPI 0 ; 1st char = 0 ?
18BA	CAD118	=3003	JZ FREE ; Yes - Then there is no label in this
		=3004	slot in the Hash table.
		=3005	;
18BD	CD0A18	=3006	CALL CMPLAB ; Compare label in buffer with label in
		=3007	symbol table .
18C0	CAF318	=3008	JZ MULDEF ; If they are the same then jump to 'MULDEF'
		=3009	(Multiply defined label).
18C3	CDFC18	=3010	CALL NEXTHV ; Calculate the next hash value
18C6	C2B418	=3011	JNZ TRYAI ; If not end of search the loop and try again.
		=3012 ;	
		=3013 ;	-----
		=3014 ;	
		=3015 ;	If end of search the the symbol table is full (print a message).
		=3016 ;	;
		=3017	TABFUL: ;
18C9	3E07	=3018	MVI A,7 ; Gen error #7 (Symbol table full)
18CB	CD3E21	=3019	CALL PERR ;
18CE	C3F818	=3020	JMP INSER ;
		=3021 ;	
		=3022 ;	-----
		=3023 ;	
		=3024 ;	If slot is free then insert the symbol into the table.
		=3025 ;	
		=3026	FREE: ; Free slot in the table found.
		=3027	;
18D1	0608	=3028	MVI B,LABBL+WORD ; Load B with byte counter.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 53

LOC	OBJ	LINE	SOURCE STATEMENT
18D3	11E3E8	=3029	LXI D,LABELB ; Load DE with start adr of label buffer.
18D6	E5	=3030	PUSH H ; Save start adr of entry.
		=3031	MOVLOP: ;
18D7	1A	=3032	LDAX D ; Load byte from lab buffer.
18D8	77	=3033	MOV M,A ; Save byte in symbol table.
18D9	23	=3034	INX H ;
18DA	13	=3035	INX D ; Inc both pointers.
18DB	05	=3036	DCR B ; Dec byte counter.
18DC	C2D718	=3037	JNZ MOVLOP ;
		=3038	; ;
18DF	E1	=3039	POP H ; Restore start adr of entry.
		=3040	;
		=3041	;
		=3042	;
		=3043	;
		=3044	Have now copied the 8 byte entry from the lookup packet to the
		=3045	symbol table, add the pass definition information to the symbol.
		=3046	;
		=3047	If Pass_number = 2 then Set MSBit of 1st character.
18E0	3A70E9	=3048	LDA PASSN ;
18E3	FE01	=3049	CPI 1 ; IF Pass_number <> 2 then
18E5	3E00	=3050	MVI A,0 ; Acc = 0
18E7	CAEC18	=3051	JZ P1DEF ; else
		=3052	; Acc = 10000000B
18EA	3E80	=3053	MVI A,MSB ;
		=3054	P1DEF: ; Acc = (M) OR Acc
18EC	B6	=3055	ORA M ; (M) = Acc
18ED	77	=3056	MOV M,A ;
		=3057	;
		=3058	;
		=3059	;
		=3060	Setup OK return and exit.
		=3061	;
18EE	3E00	=3062	MVI A,00H ; Zero acc (Z=1, Insert OK)
18F0	C3FA18	=3063	JMP INSRET ; Exit insert routine.
		=3064	;
		=3065	;
		=3066	;
		=3067	Multiple definition of label (print error and exit)
		=3068	;
		=3069	MULDEF: ;
18F3	3E04	=3070	MVI A,4 ; Gen Error #4 (Multipy defined symbol)
18F5	CD3E21	=3071	CALL PERR ;
		=3072	;
		=3073	;
		=3074	;
		=3075	Error exit point for the INSERT routine.
		=3076	;
		=3077	INSER: ;
18F8	3EFF	=3078	MVI A,OFFH ; Insert Error return.
		=3079	;
		=3080	;
		=3081	;
		=3082	Main exit point for the insert routine.
		=3083	;
		=3084	INSRET: ; Insert Exit point.
18FA	B7	=3085	ORA A ; Set flags before returning.
18FB	C9	=3086	RET ; Return from <INSERT>
		=3087	;
		=3088	=====
		=3089	;
		=3090	Function : NEXTHV (Next hash value) *
		=3091	Inputs : None. *
		=3092	Outputs : Z = 1 : If the end of the quadratic search *
		=3093	has been reached. *
		=3094	Z = 0 and Areg = next hash value : If there *
		=3095	is still an unsearched slot in the *
		=3096	symbol table. *
		=3097	Destroys : A and flags. *
		=3098	Calls : None. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 54

LOC	OBJ	LINE	SOURCE STATEMENT
		=3099 ;*	
		=3100 ;*	Description : This routine implements a quadratic seach of
		=3101 ;*	of the symbol table.
		=3102 ;*	
		=3103 ;=====	
		=3104 ;	
		=3105 NEXTHV:	
18FC	C5	=3106	PUSH B ; Save B register
18FD	3AF1E8	=3107	LDA HASHV ;
1900	47	=3108	MOV B,A ; Temp save in B reg.
1901	3AF2E8	=3109	LDA HVINC ;
1904	3C	=3110	INR A ; Inc Hash_Value_inc.
1905	32F2E8	=3111	STA HVINC ;
1908	CA1519	=3112	JZ NXTHVE ; If hash_value_inc = Zero then we have tried
		=3113	; the entire table and not found an empty slot
		=3114	; Therefore the symbol table is full.
		=3115	;
190B	80	=3116	ADD B ; Add Hash_value_inc to the hash_value
190C	32F1E8	=3117	STA HASHV ; Store new Hash_value.
190F	3EFF	=3118	MVI A,OFFH ;
1911	B7	=3119	ORA A ; Clear zero flag.
1912	3AF1E8	=3120	LDA HASHV ; Restore new hash value
		=3121 NXTHVE:	;
1915	C1	=3122	POP B ; Restore B register
1916	C9	=3123	RET ; Return from <NEXTHV> routine
		=3124 ;	
		=3125 ;=====	
		=3126 ;*	
		=3127 ;*	Function : TABINT (Symbol table init)
		=3128 ;*	Inputs : None.
		=3129 ;*	Outputs : None.
		=3130 ;*	Destroys : H,L,D,E
		=3131 ;*	Calls : None.
		=3132 ;*	
		=3133 ;*	Description: This routine simply inits the symbols
		=3134 ;*	by filling it with zero's. There are 256
		=3135 ;*	entries in the table and each one has a
		=3136 ;*	length or (LABBL + WORD).
		=3137 ;*	
		=3138 ;=====	
		=3139 ;	
		=3140 TABINT:	;
1917	2100E0	=3141	LXI H,STABLE ; Load HL with start of symbol table.
191A	110008	=3142	LXI D,256*(LABBL+WORD)
		=3143 NEXTBY:	;
191D	3600	=3144	MVI M,00H ; Store a zero in the table
191F	23	=3145	INX H ; Inc table pointer to the next byte.
1920	1B	=3146	DCX D ; Dec byte counter.
1921	7A	=3147	MOV A,D ;
1922	B3	=3148	ORA E ;
1923	C21D19	=3149	JNZ NEXTBY ; Loop to next byte until counter = 0.
		=3150	;
1926	C9	=3151	RET ; Return from <TABINT>
		=3152 ;	
		=3153 ;	End of symbol table routines (STABLE.A85)
		3154 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 55

LOC	OBJ	LINE	SOURCE STATEMENT
		3155	\$INCLUDE(ASMSTDMP.A85)
		=3156	;
		=3157	;
		=3158	Symbol table dump module for the UCT SABus assembler
		=3159	;
		=3160	;
		=3161	This module contains (in order) the following subroutines:
		=3162	;
		=3163	a) PRTLAB - Print label.
		=3164	b) SCMP - String compare routine.
		=3165	c) STDUMP - Main routine, symbol table dump.
		=3166	;
		=3167	;
		=3168	;
		=3169	=====
		=3170	;
		=3171	Function : STDUMP (Symbol table dump).
		=3172	Inputs : None.
		=3173	Outputs : None.
		=3174	Destroys : All.
		=3175	Calls : NEXTPL, STRING, SCMP, PRTLAB, CONVHV and SWOPB.
		=3176	;
		=3177	Description: This routine prints a sorted symbol table.
		=3178	;
		=3179	This is done by finding the smallest label in the symbol
		=3180	table and then finding the next smallest and so on until
		=3181	there are no more symbols in the table. This is a very
		=3182	slow method, but it must be used as the symbol table must
		=3183	not be changed in any way as it is used by the debugger.
		=3184	;
		=3185	=====
		=3186	;
		=3187	Symbol table dump routine. (Prints a sorted table).
		=3188	;
		=3189	STDUMP: ; Start of Symbol table dump routine <STDUMP>
		=3190	;
1927	CD101E	=3191	CALL NEXTPL ; Start another line.
192A	219C2E	=3192	LXI H,STDMES ;
192D	CDB51F	=3193	CALL STRING ; Print (" Symbol Table ") message.
1930	CD101E	=3194	CALL NEXTPL ; Another line.
1933	3E00	=3195	MVI A,0 ;
1935	3204E8	=3196	STA SYMSOL ; Init Symbols_on_line = 0.
1938	210000	=3197	LXI H,0 ;
193B	22EBE8	=3198	SHLD TEMPLB ;
193E	22EDE8	=3199	SHLD TEMPLB+2 ;
1941	22EFE8	=3200	SHLD TEMPLB+4 ; TEMPLB := (0,0,0,0,0,0) ;
		=3201	;
		=3202	PRTNI: ;
1944	217F7F	=3203	LXI H,7F7FH ;
1947	22E3E8	=3204	SHLD LABELB ;
194A	22E5E8	=3205	SHLD LABELB+2 ;
194D	22E7E8	=3206	SHLD LABELB+4 ; LABELB := (#HEX 7F(6)#) ;
		=3207	;
1950	0E00	=3208	MVI C,0 ; Init loop counter.
		=3209	FORIL: ;
1952	79	=3210	MOV A,C ; Load Entry number to check.
1953	CD2118	=3211	CALL CONVHV ; Convert entry number to a pointer.
1956	7E	=3212	MOV A,M ; Load 1st character of the entry.
1957	FE00	=3213	CPI 0 ; 1st char zero ? (i.e. Entry ?)
1959	CA8219	=3214	JZ NEXTI ; Skip the entry if the 1st char=0.
		=3215	;
195C	11EBE8	=3216	LXI D,TEMPLB ;
195F	CDD819	=3217	CALL SCMP ; Compare ENT(I) with TEMPLB.
1962	F28219	=3218	JP NEXTI ;
		=3219	;
1965	11E3E8	=3220	LXI D,LABELB ; Compare ENT(I) with LABELB.
1968	CDD819	=3221	CALL SCMP ;
196B	FA8219	=3222	JM NEXTI ;
		=3223	;
196E	CA8219	=3224	JZ NEXTI ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 56

LOC	OBJ	LINE	SOURCE STATEMENT	
		=3225		;
1971	11E3E8	=3226	LXI D,LABELB	; IF ENT(I) > TEMPSB AND ENT(I) < LABELB
		=3227		; THEN LABELB := END(I).
1974	0606	=3228	MVI B,LABBL	;
1976	79	=3229	MOV A,C	; HASHV := I
1977	32F1E8	=3230	STA HASHV	;
		=3231	SSD10:	;
197A	7E	=3232	MOV A,M	; Load char from ENT(I).
197B	12	=3233	STAX D	; Save char in LABELB.
197C	23	=3234	INX H	;
197D	13	=3235	INX D	;
197E	05	=3236	DCR B	;
197F	C27A19	=3237	JNZ SSD10	; Move all 6 characters.
		=3238		;
		=3239	NEXTI:	;
1982	0C	=3240	INR C	; Inc loop counter.
1983	C25219	=3241	JNZ FORIL	; Loop until c=0
		=3242		;
1986	3AE3E8	=3243	LDA LABELB	; If LABELB[1] # CHR\$(127) THEN
1989	FE7F	=3244	CPI 7FH	;
198B	CA9719	=3245	JZ ENDOFT	; BEGIN
		=3246		;
198E	CD9E14	=3247	CALL SWOPB	; TEMPLB := ENT(I) ;
1991	CD9B19	=3248	CALL PRTLAB	; WRITELN(ENT(I).NAME,ENT(I).VALUE)
		=3249		;
1994	C34419	=3250	JMP PRTNI	; END
		=3251	ENDOFT:	;
1997	CD101E	=3252	CALL NEXTPL	; Another new line.
		=3253		;
199A	C9	=3254	RET	; Return from <STDUMP>
		=3255	;	
		=3256	;	=====
		=3257	;	*
		=3258	;	* Function : PRTLAB (Print label)
		=3259	;	* Inputs : Hash_value of the label to be printed.
		=3260	;	* Outputs : None.
		=3261	;	* Destroys : All.
		=3262	;	* Calls : CONVHV,PRINTH,NEXTPL and PRINT
		=3263	;	*
		=3264	;	* Description: This routine prints the Symbol table entry
		=3265	;	* specified by the current HASH value in memory. The routine
		=3266	;	* first converts the hash value to the start adr of the entry
		=3267	;	* and then prints the 6 letter label, a space and then the 4
		=3268	;	* digit hex value.
		=3269	;	*
		=3270	;	* The routine also maintains a count of the number of symbols
		=3271	;	* on the line and adds a <CR><LF> every 6 symbols.
		=3272	;	*
		=3273	;	=====
		=3274	;	
		=3275	PRTLAB:	; Start of <PRTLAB>
		=3276		;
199B	3AF1E8	=3277	LDA HASHV	; Get hash value of the label to be printed.
199E	CD2118	=3278	CALL CONVHV	; Get start adr of the label by converting
		=3279		; the hash value to the start adr of the entry
		=3280		; in the symbol table.
19A1	0606	=3281	MVI B,LABBL	; Init char counter to label_buffer_len
		=3282		;
		=3283	PRTL10:	;
19A3	7E	=3284	MOV A,M	; Get char from symbol table.
19A4	CDC61E	=3285	CALL PRINT	; Print char.
19A7	23	=3286	INX H	; Inc pointer.
19A8	05	=3287	DCR B	; Dec char counter.
19A9	C2A319	=3288	JNZ PRTL10	; Loop until char counter = zero.
		=3289		;
19AC	3E20	=3290	MVI A,' '	;
19AE	CDC61E	=3291	CALL PRINT	; Print a space.
		=3292		;
19B1	5E	=3293	MOV E,M	; Load 1sbyte of the value.
19B2	23	=3294	INX H	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 57

LOC	OBJ	LINE	SOURCE STATEMENT
19B3	7E	=3295	MOV A,M ; Load Msbyte of the value
		=3296	;
		=3297	;
19B4	CDDC1E	=3298	CALL PRINTH ; Now print the value of the label.
19B7	7B	=3299	MOV A,E ; Acc = Msbyte.
19B8	CDDC1E	=3300	CALL PRINTH ; Acc = Lsbyte.
		=3301	;
19BB	3E20	=3302	MVI A, ' ' ;
19BD	CDC61E	=3303	CALL PRINT ; Print a space
19C0	CDC61E	=3304	CALL PRINT ; and another.
		=3305	;
19C3	3A04E8	=3306	LDA SYMSOL ;
19C6	3C	=3307	INR A ; Inc symbols on line counter.
19C7	3204E8	=3308	STA SYMSOL ;
19CA	FE06	=3309	CPI 6 ;
19CC	C2D719	=3310	JNZ PLABEX ; If symbols_on_line = 6 then
		=3311	; Begin
19CF	3E00	=3312	MVI A,0 ;
19D1	3204E8	=3313	STA SYMSOL ; Symbols_on_line := 0 ;
19D4	CD101E	=3314	CALL NEXTPL ; Writeln
		=3315	; End;
		=3316	;
		=3317	PLABEX: ; Print label exit point.
19D7	C9	=3318	RET ; Return from <PRTLAB>
		=3319	;
		=3320	=====
		=3321	;
		=3322	Function : SCMP (String compare) *
		=3323	;
		=3324	Inputs : HL = Pointer to the start of string buf #1 *
		=3325	DE = Pointer to the start of string buf #2 *
		=3326	;
		=3327	Outputs : Z=1 : If Lab #1 = Lab #2 *
		=3328	Z=0, S=1 : If Lab #1 < Lab #2 *
		=3329	Z=0, S=0 : If Lab #1 > Lab #2 *
		=3330	;
		=3331	Destroys : Acc and Flags *
		=3332	;
		=3333	Calls : None. *
		=3334	;
		=3335	Description: This routine compares the two six char buffers *
		=3336	pointed to by DE and HL. *
		=3337	;
		=3338	=====
		=3339	;
		=3340	SCMP: ; Start of <SCMP> String compare routine.
19D8	D5	=3341	PUSH D ;
19D9	E5	=3342	PUSH H ; Save H,L,D and E.
19DA	C5	=3343	PUSH B ;
19DB	0606	=3344	MVI B,LABELL ; Init character counter = Label_buf_len.
		=3345	;
		=3346	SCMP05: ;
19DD	7E	=3347	MOV A,M ; Fetch character from buffer #1
19DE	E67F	=3348	ANI OFFMSB ; Mask off MSBit of 1st character.
19E0	4F	=3349	MOV C,A ; Store in C register.
		=3350	;
19E1	1A	=3351	LDAX D ; Get char from Buffer #2
19E2	E67F	=3352	ANI OFFMSB ; Mask off MSBit of 2nd character.
		=3353	;
19E4	B9	=3354	CMP C ; Compare it with char in buffer #1
19E5	C2EE19	=3355	JNZ SCMP10 ; If not equal then exit.
		=3356	;
19E8	23	=3357	INX H ; else Inc pointer within buffer #1
19E9	13	=3358	INX D ; and Inc pointer within buffer #2
19EA	05	=3359	DCR B ; Dec char counter.
19EB	C2DD19	=3360	JNZ SCMP05 ; Loop until character counter = zero.
		=3361	;
		=3362	SCMP10: ; String comp exit point.
19EE	C1	=3363	POP B ;
19EF	E1	=3364	POP H ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 59

LOC	OBJ	LINE	SOURCE STATEMENT
		3370 ;	
		3371 ;	-----
		3372 ;	
		3373 ;	OPCODE LOOKUP ROUTINES.
		3374 ;	
		3375 \$INCLUDE(ASMOPCODE.A85)	
		=3376 ;	-----
		=3377 ;	
		=3378 ;	Opcode lookup module for the UCT SABus assembler ;
		=3379 ;	----- ;
		=3380 ;	
		=3381 ;	This module contains (in order) the following subroutines: ;
		=3382 ;	
		=3383 ;	a) CMPOP - Compare opcode. ;
		=3384 ;	b) CONVPO - Convert opcode number to table address. ;
		=3385 ;	c) QOP - 8085 assembler opcode ? ;
		=3386 ;	
		=3387 ;	-----
		=3388 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 60

LOC	OBJ	LINE	SOURCE STATEMENT
		=3389 ;	=====
		=3390 ;*	*
		=3391 ;*	Function : CMPPOP (Compare opcode)
		=3392 ;*	Inputs : HL = Start address of entry.
		=3393 ;*	Outputs : Z = 1 : If Opcode_buffer = Entry pointed to
		=3394 ;*	by the HL register pair.
		=3395 ;*	Z = 0 : If there no equal.
		=3396 ;*	*
		=3397 ;*	S = 1 : If Opcode_buffer < Entry.
		=3398 ;*	S = 0 : If Opcode_buffer > Entry.
		=3399 ;*	*
		=3400 ;*	Destroys : C,A,D,E and Flags.
		=3401 ;*	Calls : None.
		=3402 ;*	*
		=3403 ;*	Description: This routine simply compares the string
		=3404 ;*	pointed to by the HL register pair (An entry in the opcode
		=3405 ;*	table) and the string in the Opcode_buffer.
		=3406 ;*	*
		=3407 ;	=====
		=3408 ;	
		=3409 CMPPOP:	; Compare Opcode buffer with
		=3410	; Opcode pointer to by SADR.
19F2	1153E9	=3411 LXI D,OPBUF	; Get start adr of OPCODE buffer.
19F5	0E04	=3412 MVI C,4	; Get len of Opcode string field.
		=3413 CMPPOP:	;
19F7	1A	=3414 LDAX D	; Get Opcode buffer.
19F8	BE	=3415 CMP M	; Compare with letter from table
19F9	C20C1A	=3416 JNZ CMPOPE	; If not Equal Return.
		=3417	;
19FC	23	=3418 INX H	; Else try next letter.
19FD	13	=3419 INX D	; Inc both pointers.
19FE	0D	=3420 DCR C	; Dec counter
19FF	C2F719	=3421 JNZ CMPPOP	; And loop until count = 0.
		=3422	;
1A02	1A	=3423 LDAX D	; Get 5th byte.
1A03	FE20	=3424 CPI ' '	;
1A05	C20C1A	=3425 JNZ CMPOPE	; Check that is a space.
		=3426	;
1A08	13	=3427 INX D	;
1A09	1A	=3428 LDAX D	; Get 6th byte.
1A0A	FE20	=3429 CPI ' '	; Check that is a space.
		=3430	;
		=3431 CMPOPE:	;
1A0C	C9	=3432 RET	; Return from <CMPPOP>
		=3433 ;	
		=3434 ;	=====
		=3435 ;*	*
		=3436 ;*	Function : CONVPO (Convert opcode number to address)
		=3437 ;*	Inputs : Acc = Entry number.
		=3438 ;*	Outputs : HL = Start address of entry.
		=3439 ;*	Destroys : D, E and flags.
		=3440 ;*	Calls : None.
		=3441 ;*	*
		=3442 ;*	Description: This converts a given opcode table entry
		=3443 ;*	number to the start adr of the entry. Each entry is 7
		=3444 ;*	bytes long.
		=3445 ;*	*
		=3446 ;	=====
		=3447 ;	
		=3448 CONVPO:	; Start of <CONVPO> routine
		=3449	;
1A0D	217B1A	=3450 LXI H,OPTAB	; Set HL = Start adr of the Table.
		=3451	; length of 1 entry.
1A10	1600	=3452 MVI D,00H	;
1A12	5F	=3453 MOV E,A	; Set DE = Opcode table entry number.
1A13	1D	=3454 DCR E	; Set DE = Entry number - 1
		=3455	;
1A14	19	=3456 DAD D	;
1A15	19	=3457 DAD D	;
1A16	19	=3458 DAD D	;

LOC	OBJ	LINE	SOURCE STATEMENT
1A17	19	=3459	DAD D ;
1A18	19	=3460	DAD D ;
1A19	19	=3461	DAD D ;
1A1A	19	=3462	DAD D ; HL = <OPTAB> + (Entry number - 1) * 7
		=3463	;
1A1B	C9	=3464	RET ; Return from <CONVPO>
		=3465 ;	
		=3466 ;	=====
		=3467 ;	*
		=3468 ;	* Function : QOP (? Opcode) *
		=3469 ;	* Inputs : Opcode buffer (Contains string read from source) *
		=3470 ;	* Outputs : Z Flag : Clear if Opcode found in table. *
		=3471 ;	* Set if Opcode no in table. *
		=3472 ;	* Destroys : A, B, C, D, E, H, L and Flags. *
		=3473 ;	* Calls : CONVPO and CMPPOP. *
		=3474 ;	*
		=3475 ;	* Description: This routine tries to find the opcode passed *
		=3476 ;	* in the OPCODE buffer in the opcode table. If then opcode *
		=3477 ;	* is found the the opcode info is stored in the opcode data *
		=3478 ;	* area. i.e. *
		=3479 ;	*
		=3480 ;	* BYTES : The number of bytes used by the opcode. *
		=3481 ;	* OPCODE : The opcode, without any registers masked in. *
		=3482 ;	* OPERS : The operands needed by the opcode. Given as 2 *
		=3483 ;	* nibbles. (Upper and lower in the byte.) *
		=3484 ;	*
		=3485 ;	* Note: The routine uses a binary search. *
		=3486 ;	*
		=3487 ;	=====
		=3488 ;	
		=3489 ;	
		=3490	QOP: ; Start of Opcode lookup routine <QOP>
1A1C	3E00	=3491	MVI A,0 ;
1A1E	3260E9	=3492	STA LOWLIM ; Set lower limit = 0
1A21	3E51	=3493	MVI A,OPTABL ;
1A23	3261E9	=3494	STA UPLIM ; Set upper limit = Number_of_opcodes
		=3495	;
		=3496	NEXTOP: ;
1A26	3A60E9	=3497	LDA LOWLIM ;
1A29	47	=3498	MOV B,A ; B_reg = Lower limit
1A2A	3A61E9	=3499	LDA UPLIM ; A_reg = Upper limit
1A2D	80	=3500	ADD B ; A_reg = Upper_limit + Lower_limit
1A2E	1F	=3501	RAR ; A_reg = (Upper_limit + Lower_limit)/2
1A2F	3262E9	=3502	STA CENT ; Save entry number to check.
		=3503	;
1A32	CD0D1A	=3504	CALL CONVPO ; Convert position to start adr of the entry.
1A35	2263E9	=3505	SHLD SADR ; Save start adr.
1A38	CDF219	=3506	CALL CMPPOP ; Compare opcode in buffer with entry.
1A3B	CA621A	=3507	JZ OCODEF ; Equal - Then opcode found.
		=3508	;
1A3E	FA4A1A	=3509	JM MODUPL ; If Opcode_buffer > Entry_checked then
		=3510	begin
1A41	3A62E9	=3511	LDA CENT ;
1A44	3260E9	=3512	STA LOWLIM ; Lower_limit = entry_checked
1A47	C3501A	=3513	JMP CONTIN ; end
		=3514	MODUPL: ; Else
		=3515	begin
1A4A	3A62E9	=3516	LDA CENT ; Upper_limit = entry_checked
1A4D	3261E9	=3517	STA UPLIM ; end;
		=3518	;
		=3519	CONTIN: ;
1A50	3A60E9	=3520	LDA LOWLIM ;
1A53	47	=3521	MOV B,A ;
1A54	3A61E9	=3522	LDA UPLIM ;
1A57	90	=3523	SUB B ; A_reg = Upper_limit - Lower_limit.
1A58	FE02	=3524	CPI 2 ; If Difference >= 2 then
1A5A	F2261A	=3525	JP NEXTOP ; Loop
		=3526	Else
1A5D	3EFF	=3527	MVI A,OFFH ; Error(Opcode not in table)
1A5F	C3791A	=3528	JMP QOPRET ;

```

LOC   OBJ       LINE      SOURCE STATEMENT
                                =3529 ;
                                =3530 ;-----
                                =3531 ;
                                =3532 ;      Opcode has been found (Set opcode variables.)
                                =3533 ;
                                =3534 OPCODEF:
1A62  2A63E9     =3535      LHLD   SADR           ; Get start adr of correct entry.
1A65  23         =3536      INX    H               ;
1A66  23         =3537      INX    H               ;
1A67  23         =3538      INX    H               ;
1A68  23         =3539      INX    H               ; Inc ptr 4 times ( Past Opcode string ).
                                =3540 ;
1A69  7E         =3541      MOV    A,M            ;
1A6A  325DE9     =3542      STA    BYTE$         ; Fetch BYTE variable.
1A6D  23         =3543      INX    H               ;
                                =3544 ;
1A6E  7E         =3545      MOV    A,M            ;
1A6F  325EE9     =3546      STA    OPERS          ; Fetch OPERS variable.
1A72  23         =3547      INX    H               ;
                                =3548 ;
1A73  7E         =3549      MOV    A,M            ;
1A74  325FE9     =3550      STA    OPCODE         ; Fetch OPCODE variable.
                                =3551 ;
1A77  3E00       =3552      MVI    A,00H          ; Return zero (Opcode found)
                                =3553 QOPRET:
1A79  B7         =3554      ORA    A               ; Set flags before returning.
                                =3555 ;
1A7A  C9         =3556      RET                    ; Return from <QOP>
                                =3557 ;
                                =3558 ;-----
                                =3559 ;*
                                =3560 ;*      Assembler opcode data area.
                                =3561 ;*-----
                                =3562 ;*
                                =3563 ;*      The format for the data area is as follows .
                                =3564 ;*
                                =3565 ;*      'opcode , in ASCII ' , N , Operand info , opcode
                                =3566 ;*
                                =3567 ;*      Where  1) The opcode string is stored as a 4 bytes
                                =3568 ;*              space filled ASCII field.
                                =3569 ;*
                                =3570 ;*              2) N is the number of bytes for the inst
                                =3571 ;*
                                =3572 ;*              3) The operand info has the following form:
                                =3573 ;*
                                =3574 ;*              2 four bit fields , where each field has
                                =3575 ;*              the following meaning
                                =3576 ;*
                                =3577 ;*      4 bit field      meaning      place to mask
                                =3578 ;*      -----      -
                                =3579 ;*
                                =3580 ;*      0000      No operand
                                =3581 ;*      0001      16 bit data
                                =3582 ;*      0010      8 bit data
                                =3583 ;*      0011      Des register      ..RRR...
                                =3584 ;*      0100      Src register      .....RRR
                                =3585 ;*      0101      Reg pr (STK)      ..RP....
                                =3586 ;*      0110      Reg pr (ARITH)    ..rp....
                                =3587 ;*      0111      Reg pr (LIMIT)   ...R....
                                =3588 ;*      1000      Value 0 to 7      ..NNN...
                                =3589 ;*      1001      Unsigned 8 bit value
                                =3590 ;*
                                =3591 ;-----
                                =3592 ;
                                =3593 OPTAB:
1A7B  41434920   =3594      DB      'ACI ' ,2,D8  + NOOP  SHL 4 , OCEH
                                0202CE
1A82  41444320   =3595      DB      'ADC ' ,1,SREG + NOOP  SHL 4 , 088H
                                010388
1A89  41444420   =3596      DB      'ADD ' ,1,SREG + NOOP  SHL 4 , 080H

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 63

LOC	OBJ	LINE	SOURCE STATEMENT
	010380		
1A90	41444920	=3597	DB 'ADI ',2,D8 + NOOP SHL 4 , 0C6H
	0202C6		
1A97	414E4120	=3598	DB 'ANA ',1,SREG + NOOP SHL 4 , 0A0H
	0103A0		
1A9E	414E4920	=3599	DB 'ANI ',2,D8 + NOOP SHL 4 , 0E6H
	0202E6		
1AA5	43414C4C	=3600	DB 'CALL',3,D16 + NOOP SHL 4 , 0CDH
	0301CD		
1AAC	43432020	=3601	DB 'CC ',3,D16 + NOOP SHL 4 , 0DCH
	0301DC		
1AB3	434D2020	=3602	DB 'CM ',3,D16 + NOOP SHL 4 , 0FCH
	0301FC		
1ABA	434D4120	=3603	DB 'CMA ',1,NOOP + NOOP SHL 4 , 02FH
	01002F		
1AC1	434D4320	=3604	DB 'CMC ',1,NOOP + NOOP SHL 4 , 03FH
	01003F		
1AC8	434D5020	=3605	DB 'CMP ',1,SREG + NOOP SHL 4 , 0B8H
	0103B8		
1ACF	434E4320	=3606	DB 'CNC ',3,D16 + NOOP SHL 4 , 0D4H
	0301D4		
1AD6	434E5A20	=3607	DB 'CNZ ',3,D16 + NOOP SHL 4 , 0C4H
	0301C4		
1ADD	43502020	=3608	DB 'CP ',3,D16 + NOOP SHL 4 , 0F4H
	0301F4		
1AE4	43504520	=3609	DB 'CPE ',3,D16 + NOOP SHL 4 , 0ECH
	0301EC		
1AEB	43504920	=3610	DB 'CPI ',2,D8 + NOOP SHL 4 , 0FEH
	0202FE		
1AF2	43504F20	=3611	DB 'CPO ',3,D16 + NOOP SHL 4 , 0E4H
	0301E4		
1AF9	435A2020	=3612	DB 'CZ ',3,D16 + NOOP SHL 4 , 0CCH
	0301CC		
1B00	44414120	=3613	DB 'DAA ',1,NOOP + NOOP SHL 4 , 027H
	010027		
1B07	44414420	=3614	DB 'DAD ',1,RPATH + NOOP SHL 4 , 009H
	010609		
1B0E	44435220	=3615	DB 'DCR ',1,DREG + NOOP SHL 4 , 005H
	010405		
1B15	44435820	=3616	DB 'DCX ',1,RPATH + NOOP SHL 4 , 00BH
	01060B		
1B1C	44492020	=3617	DB 'DI ',1,NOOP + NOOP SHL 4 , 0F3H
	0100F3		
1B23	45492020	=3618	DB 'EI ',1,NOOP + NOOP SHL 4 , 0FBH
	0100FB		
1B2A	484C5420	=3619	DB 'HLT ',1,NOOP + NOOP SHL 4 , 076H
	010076		
1B31	494E2020	=3620	DB 'IN ',2,UD8 + NOOP SHL 4 , 0DBH
	0209DB		
1B38	494E5220	=3621	DB 'INR ',1,DREG + NOOP SHL 4 , 004H
	010404		
1B3F	494E5820	=3622	DB 'INX ',1,RPATH + NOOP SHL 4 , 003H
	010603		
1B46	4A432020	=3623	DB 'JC ',3,D16 + NOOP SHL 4 , 0DAH
	0301DA		
1B4D	4A4D2020	=3624	DB 'JM ',3,D16 + NOOP SHL 4 , 0FAH
	0301FA		
1B54	4A4D5020	=3625	DB 'JMP ',3,D16 + NOOP SHL 4 , 0C3H
	0301C3		
1B5B	4A4E4320	=3626	DB 'JNC ',3,D16 + NOOP SHL 4 , 0D2H
	0301D2		
1B62	4A4E5A20	=3627	DB 'JNZ ',3,D16 + NOOP SHL 4 , 0C2H
	0301C2		
1B69	4A502020	=3628	DB 'JP ',3,D16 + NOOP SHL 4 , 0F2H
	0301F2		
1B70	4A504520	=3629	DB 'JPE ',3,D16 + NOOP SHL 4 , 0EAH
	0301EA		
1B77	4A504F20	=3630	DB 'JPO ',3,D16 + NOOP SHL 4 , 0E2H
	0301E2		
1B7E	4A5A2020	=3631	DB 'JZ ',3,D16 + NOOP SHL 4 , 0CAH

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 64

LOC	OBJ	LINE	SOURCE STATEMENT
	0301CA		
1B85	4C444120	=3632	DB 'LDA ',3,D16 + NOOP SHL 4 , 03AH
	03013A		
1B8C	4C444158	=3633	DB 'LDAX',1,RPLIM + NOOP SHL 4 , 00AH
	01070A		
1B93	4C484C44	=3634	DB 'LHLD',3,D16 + NOOP SHL 4 , 02AH
	03012A		
1B9A	4C584920	=3635	DB 'LXI ',3,RPATH + D16 SHL 4 , 001H
	031601		
1BA1	4D4F5620	=3636	DB 'MOV ',1,DREG + SREG SHL 4 , 040H
	013440		
1BA8	4D564920	=3637	DB 'MVI ',2,DREG + D8 SHL 4 , 006H
	022406		
1BAF	4E4F5020	=3638	DB 'NOP ',1,NOOP + NOOP SHL 4 , 000H
	010000		
1BB6	4F524120	=3639	DB 'ORA ',1,SREG + NOOP SHL 4 , 0B0H
	0103B0		
1BBD	4F524920	=3640	DB 'ORI ',2,D8 + NOOP SHL 4 , 0F6H
	0202F6		
1BC4	4F555420	=3641	DB 'OUT ',2,UD8 + NOOP SHL 4 , 0D3H
	0209D3		
1BCB	5043484C	=3642	DB 'PCHL',1,NOOP + NOOP SHL 4 , 0E9H
	0100E9		
1BD2	504F5020	=3643	DB 'POP ',1,RPSTK + NOOP SHL 4 , 0C1H
	0105C1		
1BD9	50555348	=3644	DB 'PUSH',1,RPSTK + NOOP SHL 4 , 0C5H
	0105C5		
1BE0	52414C20	=3645	DB 'RAL ',1,NOOP + NOOP SHL 4 , 017H
	010017		
1BE7	52415220	=3646	DB 'RAR ',1,NOOP + NOOP SHL 4 , 01FH
	01001F		
1BEE	52432020	=3647	DB 'RC ',1,NOOP + NOOP SHL 4 , 0D8H
	0100D8		
1BF5	52455420	=3648	DB 'RET ',1,NOOP + NOOP SHL 4 , 0C9H
	0100C9		
		=3649	
		=3650	IF M8085
1BFC	52494D20	=3651	DB 'RIM ',1,NOOP + NOOP SHL 4 , 020H
	010020		
		=3652	ENDIF
		=3653	
1C03	524C4320	=3654	DB 'RLC ',1,NOOP + NOOP SHL 4 , 007H
	010007		
1C0A	524D2020	=3655	DB 'RM ',1,NOOP + NOOP SHL 4 , 0F8H
	0100F8		
1C11	524E4320	=3656	DB 'RNC ',1,NOOP + NOOP SHL 4 , 0D0H
	0100D0		
1C18	524E5A20	=3657	DB 'RNZ ',1,NOOP + NOOP SHL 4 , 0C0H
	0100C0		
1C1F	52502020	=3658	DB 'RP ',1,NOOP + NOOP SHL 4 , 0F0H
	0100F0		
1C26	52504520	=3659	DB 'RPE ',1,NOOP + NOOP SHL 4 , 0E8H
	0100E8		
1C2D	52504F20	=3660	DB 'RPO ',1,NOOP + NOOP SHL 4 , 0E0H
	0100E0		
1C34	52524320	=3661	DB 'RRC ',1,NOOP + NOOP SHL 4 , 00FH
	01000F		
1C3B	52535420	=3662	DB 'RST ',1,RSTN + NOOP SHL 4 , 0C7H
	0108C7		
1C42	525A2020	=3663	DB 'RZ ',1,NOOP + NOOP SHL 4 , 0C8H
	0100C8		
1C49	53424220	=3664	DB 'SBB ',1,SREG + NOOP SHL 4 , 098H
	010398		
1C50	53424920	=3665	DB 'SBI ',2,D8 + NOOP SHL 4 , 0DEH
	0202DE		
1C57	53484C44	=3666	DB 'SHLD',3,D16 + NOOP SHL 4 , 022H
	030122		
		=3667	
		=3668	IF M8085
1C5E	53494D20	=3669	DB 'SIM ',1,NOOP + NOOP SHL 4 , 030H

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 65

LOC	OBJ	LINE	SOURCE STATEMENT
	010030		
		=3670	ENDIF
		=3671	
1C65	5350484C	=3672	DB 'SPHL',1,NOOP + NOOP SHL 4 , 0F9H
	0100F9		
1C6C	53544120	=3673	DB 'STA ',3,D16 + NOOP SHL 4 , 032H
	030132		
1C73	53544158	=3674	DB 'STAX',1,RPLIM + NOOP SHL 4 , 002H
	010702		
1C7A	53544320	=3675	DB 'STC ',1,NOOP + NOOP SHL 4 , 037H
	010037		
1C81	53554220	=3676	DB 'SUB ',1,SREG + NOOP SHL 4 , 090H
	010390		
1C88	53554920	=3677	DB 'SUI ',2,D8 + NOOP SHL 4 , 0D6H
	0202D6		
1C8F	58434847	=3678	DB 'XCHG',1,NOOP + NOOP SHL 4 , 0EBH
	0100EB		
1C96	58524120	=3679	DB 'XRA ',1,SREG + NOOP SHL 4 , 0A8H
	0103A8		
1C9D	58524920	=3680	DB 'XRI ',2,D8 + NOOP SHL 4 , 0EEH
	0202EE		
1CA4	5854484C	=3681	DB 'XTHL',1,NOOP + NOOP SHL 4 , 0E3H
	0100E3		
		=3682 ;	
0051		=3683	OPTABL EQU ((\$ - OPTAB) / 7) + 1
		=3684 ;	
		=3685 ;	End of Opcode lookup routine (OPCODE.A85)
		3686	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 66

LOC	OBJ	LINE	SOURCE STATEMENT
		3687	\$INCLUDE(ASMPPCODE.A85)
		=3688	;
		=3689	;
		=3690	Assembler directive lookup module for the UCT SABus assembler
		=3691	;
		=3692	;
		=3693	This module contains the following subroutine:
		=3694	;
		=3695	a) QPHOP - Assembler directive ?
		=3696	;
		=3697	;
		=3698	;
		=3699	;
		=3700	;
		=3701	Function : QPHOP (? Ph opcode)
		=3702	Inputs : Opcode_buffer (contains the string to be
		=3703	check against the Ph opcode table)
		=3704	Outputs : If not in table then Non Zero returned.
		=3705	If String found in table the Zero return
		=3706	and the Adr of the Directive routine is
		=3707	returned in DE.
		=3708	Destroys : D, E, H, L, Acc and Flags.
		=3709	Calls : None
		=3710	;
		=3711	Description: This routine is use to find out if the
		=3712	opcode_string in the Opcode_buffer is an assmebler
		=3713	directive. If the String is not found in the table the
		=3714	routine returns with the zero flag clear. If string is
		=3715	found then the zero flag is set and the address of the
		=3716	directive routine is returned in the DE reg pair. The
		=3717	format of the table is as follows.
		=3718	;
		=3719	'Ph Opcode #1 string', EOT, ADR (Dir routine #1)
		=3720	'Ph Opcode #2 string', EOT, ADR (Dir routine #2)
		=3721	etc
		=3722	'Ph Opcode #N string', EOT, ADR (Dir routine #N)
		=3723	ETABLE (* End of table marker. *)
		=3724	;
		=3725	;
		=3726	;
		=3727	QPHOP: ; Start of ? Assembler directive routine.
1CAB	C5	=3728	PUSH B ;
1CAC	21F11C	=3729	LXI H,PHOTAB ; Get start adr of Ph opcode table.
		=3730	NXTPHO: ;
1CAF	1153E9	=3731	LXI D,OPBUF ; Get start adr of Opcode_buffer.
1CB2	0606	=3732	MVI B,OPBUFL ; Get length of Opcode_buffer.
		=3733	NXTPHC: ;
1CB4	7E	=3734	MOV A,M ; If char = EOT then
1CB5	FE04	=3735	CPI EOT ;
1CB7	CADD1C	=3736	JZ PHOPF ; Ph opcode found.
		=3737	; else
1CBA	1A	=3738	LDAX D ;
1CBB	BE	=3739	CMP M ; If char # Opbuf char then
1CBC	C2C51C	=3740	JNZ NOTT1 ; Not that Ph opcode.
		=3741	; else
1CBF	23	=3742	INX H ; inc pointer.
1CC0	13	=3743	INX D ;
1CC1	05	=3744	DCR B ;
1CC2	C2B41C	=3745	JNZ NXTPHC ; Loop until counter = 0.
		=3746	NOTT1: ;
1CC5	7E	=3747	MOV A,M ; If Not the entry then try next.
1CC6	FE04	=3748	CPI EOT ;
1CC8	CACF1C	=3749	JZ ENDPHO ; Loop until Table_char = 'EOT.
		=3750	;
1CCB	23	=3751	INX H ;
1CCC	C3C51C	=3752	JMP NOTT1 ;
		=3753	;
		=3754	ENDPHO: ; Pointer now pointing to the EOT of the
		=3755	; last entry.(skip to 1st character of the
		=3756	; Next entry.)

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 67

LOC	OBJ	LINE	SOURCE STATEMENT
1CCF	23	=3757	INX H ;
1CD0	23	=3758	INX H ;
1CD1	23	=3759	INX H ;
1CD2	7E	=3760	MOV A,M ; Get 1st character of the next entry.
1CD3	FEFF	=3761	CPI ETABLE ; Char = End_of_table ?
1CD5	C2AF1C	=3762	JNZ NXTPHO ; No then Loop and check that entry.
		=3763	;
		=3764	QHPERR: ; Else (* end of table and Opcode not found *)
1CD8	3EFF	=3765	MVI A,OFFH ; Reset zero flag.
1CDA	C3EE1C	=3766	JMP QPHOPR ; Exit routine.
		=3767	;
		=3768	PHOPF: ; If Opcode found then check that the rest of
		=3769	the Opcode buffer if blank.
1CDD	1A	=3770	LDAX D ; Get Opcode_buffer_char
1CDE	FE20	=3771	CPI ' ' ; Space ?
1CE0	C2CF1C	=3772	JNZ ENDPHO ; No - then not that Ph opcode so try the next.
		=3773	;
1CE3	13	=3774	INX D ; Inc opcode buffer pointer.
1CE4	05	=3775	DCR B ; Dec character counter.
1CE5	C2DD1C	=3776	JNZ PHOPF ; loop until end of opcode buffer.
		=3777	;
1CE8	23	=3778	INX H ; Opcode buffer = Dir string #X and the rest of
		=3779	the buffer is blank so can now read dir rou
		=3780	adr from the next 2 bytes in the table.
1CE9	5E	=3781	MOV E,M ;
1CEA	23	=3782	INX H ;
1CEB	56	=3783	MOV D,M ; DE = Adr of dir routine.
1CEC	3E00	=3784	MVI A,00H ; Setup zero return.
		=3785	QPHOPR: ;
1CEE	C1	=3786	POP B ;
1CEF	B7	=3787	ORA A ; Set flags before returning.
1CF0	C9	=3788	RET ; Return from <QPHOP>
		=3789 ;	
		=3790 ;	
		=3791 ;	
		=3792 ;	Assembler directive lookup table.
		=3793 ;	
		=3794 ;	
		=3795 ;	
		=3796	PHOTAB: ; Start of table.
		=3797	;
1CF1	454E4404	=3798	DB 'END',EOT ; End of text directive.
1CF5	FA15	=3799	DW ENDROU ;
		=3800	;
1CF7	4F524704	=3801	DB 'ORG',EOT ; Origin directive.
1CFB	5516	=3802	DW ORGROU ;
		=3803	;
1CFD	445304	=3804	DB 'DS',EOT ; Define Storage directive.
1D00	7915	=3805	DW DSROU ;
		=3806	;
1D02	445704	=3807	DB 'DW',EOT ; Define Word directive.
1D05	CB15	=3808	DW DWROU ;
		=3809	;
1D07	444204	=3810	DB 'DB',EOT ; Define Byte directive.
1D0A	D114	=3811	DW DBROU ;
		=3812	;
1D0C	FF	=3813	DB ETABLE ; End of table marker.
		=3814 ;	
		=3815 ;	End of Phopcode lookup routines. (PHOPCODE.A85)
		3816	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 68

LOC	OBJ	LINE	SOURCE STATEMENT
		3817 ;	
		3818 ;-----	
		3819 ;	
		3820 ;	I N P U T / O U T P U T R O U T I N E S .
		3821 ;	
		3822 \$INCLUDE(ASMBIO.A85)	
		=3823 ;-----	
		=3824 ;	
		=3825 ;	ASMBIO module for the UCT SABus assembler ;
		=3826 ;	----- ;
		=3827 ;	
		=3828 ;	This module contains the basic I/O equates used in the SABUS 8085 ;
		=3829 ;	assembler. These equates should be changed if the assembler is ;
		=3830 ;	to be run on another machine. ;
		=3831 ;	
		=3832 ;-----	
		=3833 ;	
F013		=3834 KITIN EQU	0F013H ; Keyboard character input routine.
F01C		=3835 KITOUT EQU	0F01CH ; VDU character output routine.
F00A		=3836 KITTEST EQU	0F00AH
F016		=3837 PDPIN EQU	0F016H ; PDP character input routine.
F01F		=3838 PDPOUT EQU	0F01FH ; PDP character output routine.
F00D		=3839 PDPTEST EQU	0F00DH
		=3840 ;	
		=3841 ;	END <ASMBIO.A85>
		3842 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 69

LOC	OBJ	LINE	SOURCE STATEMENT
		3843	\$INCLUDE(ASMIO.A85)
=3844			-----;
=3845			;
=3846			Input/Output routines used by 8085 assembler. ;
=3847			;
=3848			A list of the routines to be found in this file follows: ;
=3849			;
=3850			a) BCC - updates the hex record BCC. ;
=3851			b) BREAK - Xon/Xoff implementation routine. ;
=3852			c) CLOSEFILE - Closes any file that may have been opened on ;
=3853			on the PDP. ;
=3854			d) CONHEX - Converts binary to ASCII (hex digits). ;
=3855			e) CRLF - Sends a CR and LF to the output device. ;
=3856			f) FILEOUT - Sends the ASCII character passed in the A reg ;
=3857			to the file on the PDP. ;
=3858			g) FPHEX - Same as PRINTH, but output to the PDP. ;
=3859			h) HEADER - Prints the list file page header. ;
=3860			i) NEXTPL - Starts the next print line of the list file. ;
=3861			j) OPENFILE - Opens a file on the PDP, name of file passed ;
=3862			in the file name buffer (FNAME). ;
=3863			k) PDPSTR - Sends a string of ASCII characters to the PDP ;
=3864			l) PHL - Prints the HL register pair in hexadecimal. ;
=3865			m) PRINT - Send the character in the Areg to the ;
=3866			output device (User's terminal or the PDP). ;
=3867			n) PRINTH - Prints the A as a 2 character value. ;
=3868			o) SENDLINE - Sends the buffered line to the PDP. ;
=3869			p) SENDHR - This sends the INTEL hex record to the PDP. ;
=3870			q) SPACES - Prints n spaces (n is passed in the A reg). ;
=3871			r) STRING - Outputs a string of ASCII characters. ;
=3872			s) TERMOUT - Sends a character to the user's terminal. ;
=3873			;
=3874			-----;
=3875			;
=3876			\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 70

LOC	OBJ	LINE	SOURCE STATEMENT
		=3877 ;	=====
		=3878 ;*	*
		=3879 ;*	Function : BCC
		=3880 ;*	Inputs : Acc = Byte to be added to the Checksum
		=3881 ;*	Outputs : None
		=3882 ;*	Destroys : None
		=3883 ;*	Calls : None
		=3884 ;*	*
		=3885 ;*	Description: This Subroutine simply adds the Data byte in
		=3886 ;*	the Acc to the checksum in memory.
		=3887 ;*	*
		=3888 ;	=====
		=3889 ;	
		=3890 BCC:	; Update Block Check Character.
1D0D	F5	=3891	PUSH PSW ;
1D0E	C5	=3892	PUSH B ; Save Registers that will be destroyed.
1D0F	47	=3893	MOV B,A ; Save Data byte in B reg.
1D10	3AE2E8	=3894	LDA CHKSUM ; Get old checksum.
1D13	80	=3895	ADD B ; Add data byte.
1D14	32E2E8	=3896	STA CHKSUM ; Save new checksum.
1D17	C1	=3897	POP B ; Restore Regs.
1D18	F1	=3898	POP PSW ;
1D19	C9	=3899	RET ; Return from <BCC>
		=3900 ;	
		=3901 ;	-----
		=3902 ;	
		=3903 ; FUNCTION :	BREAK
		=3904 ; INPUTS :	NONE
		=3905 ; OUTPUTS :	NONE
		=3906 ; CALLS :	GETUC
		=3907 ; DESTROYS :	A, F/F's
		=3908 ; STACK USAGE :	4 bytes
		=3909 ; DESCRIPTION :	BREAK checks whether a character has been typed on the console
		=3910 ;	and takes the appropriate action. If it is a control-S (XOFF)
		=3911 ;	the routine waits until a control-Q (XON) is entered before
		=3912 ;	returning. Any other characters are lost.
		=3913 ;	
		=3914 BREAK:	
1D1A	CD0AF0	=3915	CALL KITTEST ; Test if console holds a character
1D1D	C8	=3916	RZ ; No - return with carry clear
1D1E	CD13F0	=3917	CALL KITIN ; Fetch received character
1D21	E67F	=3918	ANI 01111111B ; Strip off parity bit
1D23	FE13	=3919	CPI CTRLS ; Is it a control-S (XOFF)?
1D25	C0	=3920	RNZ ; No - ignore character
		=3921 BRK05:	
1D26	CD13F0	=3922	CALL KITIN ; Fetch another character from console
1D29	FE11	=3923	CPI CTRLQ ; Is it a control-Q (XON)?
1D2B	C2261D	=3924	JNZ BRK05 ; No - loop until it is
1D2E	C9	=3925	RET ; Yes - return with carry clear
		=3926 ;	
		=3927 ;	=====
		=3928 ;*	*
		=3929 ;*	Function : CLOSEFILE
		=3930 ;*	Inputs : None.
		=3931 ;*	Outputs : None.
		=3932 ;*	Destroys : A,H,L and Flags.
		=3933 ;*	Calls : SENDHR, FILEOUT, SENDLINE, PDPOUT, STRING and
		=3934 ;*	PNERRS.
		=3935 ;*	*
		=3936 ;*	Description : This routine closes and open file on the PDP.
		=3937 ;*	There are two flags can be used to see if any file has
		=3938 ;*	been opened. These are PDPFLG, set true if the listfile
		=3939 ;*	is open and OBJFLG, set true is the objectfile is open
		=3940 ;*	on the PDP.
		=3941 ;*	*
		=3942 ;*	If neither of the files are open then the routine simply
		=3943 ;*	returns. If the file open is an object file then then
		=3944 ;*	buffered hex bytes are send following by the INTEL paper
		=3945 ;*	format file terminator record '0000001FF'.
		=3946 ;*	*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 71

LOC	OBJ	LINE	SOURCE STATEMENT
		=3947 ;*	The file is then closed by sending any buffered characters*
		=3948 ;*	and an EOF character. *
		=3949 ;*	*
		=3950 ;	=====
		=3951 ;	
		=3952	CLOSEFILE: ; Start of close PDP file routine.
1D2F	3A20E8	=3953	LDA PDPFLG ;
1D32	FEFF	=3954	CPI TRUE ;
1D34	CA561D	=3955	JZ CLOSE ; If PDPFLG = True then close the file.
		=3956	;
1D37	3AC7E8	=3957	LDA OBJFLG ;
1D3A	FE00	=3958	CPI FALSE ;
1D3C	CA781D	=3959	JZ NOOPNF ; If OBJFLG = False then there was no open
		=3960	file, so exit file close file routine.
		=3961 ;	
		=3962 ;	-----
		=3963 ;	
		=3964 ;	Close object format (INTEL paper tape format) file here.
		=3965 ;	
1D3F	CD401F	=3966	CALL SENDHR ; If then file was an object file then send
		=3967	the remaining code in the code buffer.
1D42	21FB2E	=3968	LXI H,TERMHR ; Send terminating record to the PDP.
		=3969	CLOSE5: ;
1D45	7E	=3970	MOV A,M ;
1D46	B7	=3971	ORA A ; Set sign flag if end of string.
1D47	FA511D	=3972	JM CLOSE7 ; If end_of_string then exit.
		=3973	;
1D4A	CD8E1D	=3974	CALL FILEOUT ; Else print the character
1D4D	23	=3975	INX H ;
1D4E	C3451D	=3976	JMP CLOSE5 ; and loop.
		=3977	CLOSE7: ;
1D51	E67F	=3978	ANI OFFMSB ; Remove the top bit of the terminating char
1D53	CD8E1D	=3979	CALL FILEOUT ; and send it.
		=3980 ;	
		=3981 ;	-----
		=3982 ;	
		=3983 ;	Close the file here (Send the line buffer if line_len <> zero).
		=3984 ;	
		=3985	CLOSE: ; Close the open file.
1D56	3AC5E8	=3986	LDA LINEBL ;
1D59	B7	=3987	ORA A ;
1D5A	C4F51E	=3988	CNZ SENDLINE ; Send the line if length <> zero
		=3989	;
1D5D	3E1A	=3990	MVI A,EOFMARK ; Send the End_of_file marker to the PDP
1D5F	CD1FF0	=3991	CALL PDPOUT ;
		=3992	;
1D62	3A20E8	=3993	LDA PDPFLG ;
1D65	FE00	=3994	CPI FALSE ; Was the file Open a list file ?
1D67	CA781D	=3995	JZ NOOPNF ; No the the Error summary went to the terminal
		=3996	So exit CLOSEFILE.
		=3997	;
1D6A	3E00	=3998	MVI A,FALSE ; Else
1D6C	3220E8	=3999	STA PDPFLG ; Reset PDP flag now that the file has been
		=4000	Closed.
1D6F	21362F	=4001	LXI H,ENDM1 ;
1D72	CDB51F	=4002	CALL STRING ; Print end of assembly message.
1D75	CD8B21	=4003	CALL PNERRS ; And print the error summary.
		=4004	;
		=4005	NOOPNF: ;
1D78	C9	=4006	RET ; Return from <CLOSEFILE>
		=4007 ;	
		=4008 ;	=====
		=4009 ;*	*
		=4010 ;*	Function : CONHEX (Convert to hexadecimal) *
		=4011 ;*	Inputs : A = Four bit binary number (0..15) *
		=4012 ;*	Outputs : A = Hexadecimal equivalent ('0'..'9', 'A'..'F') *
		=4013 ;*	Destroys : A and Flags *
		=4014 ;*	Calls : None. *
		=4015 ;*	*
		=4016 ;*	Description : This routine converts a binary nibble to it's *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 72

LOC	OBJ	LINE	SOURCE STATEMENT
		=4017	* equivalent hexadecimal character. *
		=4018	* *
		=4019	=====
		=4020	;
		=4021	CONHEX: ; Start of <CONHEX> routine
1D79	FE0A	=4022	CPI OAH ; Is the value greater than 9 (i.e. letter).
1D7B	FA801D	=4023	JM NUMER ; Number between 0 and 9.
		=4024	;
1D7E	C607	=4025	ADI 'A' - '9' - 1 ;
		=4026	NUMER: ;
1D80	C630	=4027	ADI '0' ; Convert to ASCII.
1D82	C9	=4028	RET ; Return from <CONHEX>
		=4029	;
		=4030	=====
		=4031	* *
		=4032	* Function : CRLF (Carriage return and linefeed) *
		=4033	* Inputs : None. *
		=4034	* Outputs : None. *
		=4035	* Destroys : Acc and Flags. *
		=4036	* Calls : PRINT. *
		=4037	* *
		=4038	* Description: This routine sends <CR><LF> to the output *
		=4039	* device, the console or a list file. *
		=4040	* *
		=4041	=====
		=4042	;
		=4043	CRLF: ; Start of <CRLF>
1D83	3E0A	=4044	MVI A,LF ; Load acc with ASCII for a <lf>.
1D85	CDC61E	=4045	CALL PRINT ; Print the character in the Acc.
		=4046	;
1D88	3E0D	=4047	MVI A,CR ; Load acc with ASCII for a <cr>.
1D8A	CDC61E	=4048	CALL PRINT ; Print the character in the Acc.
		=4049	;
1D8D	C9	=4050	RET ; Return from <CRLF>.
		=4051	;
		=4052	=====
		=4053	* *
		=4054	* Function : FILEOUT (Output a character to the file) *
		=4055	* Inputs : A = Character to be sent to the file *
		=4056	* Outputs : None *
		=4057	* Destroys : None *
		=4058	* Calls : SENDLINE *
		=4059	* *
		=4060	* Description: This routine is used as the common output *
		=4061	* routine for all characters being sent to the open file on *
		=4062	* the PDP. *
		=4063	* *
		=4064	* All this routine does is buffer the characters checking for *
		=4065	* the special ones. Some characters are ignored, these are *
		=4066	* <LF>s and <BEL>s. If the character is a <CR> then the *
		=4067	* current record is sent to the file on the PDP and the *
		=4068	* pointers are setup ready for the next one. *
		=4069	* *
		=4070	* NOTE: This routine does bound checking on the output buffer *
		=4071	* and if too many characters are received then the *
		=4072	* extra are ignored. *
		=4073	* *
		=4074	=====
		=4075	;
		=4076	FILEOUT: ; Start of File output routine.
1D8E	E5	=4077	PUSH H ;
1D8F	D5	=4078	PUSH D ;
1D90	F5	=4079	PUSH PSW ; Save H,L,D,E,A and flags.
		=4080	;
1D91	57	=4081	MOV D,A ; Save character to output in D
1D92	FE0A	=4082	CPI LF ; Character = <LF> ?
1D94	CAC61D	=4083	JZ FIOEX ; Yes - Ignore it and exit
		=4084	;
1D97	FE07	=4085	CPI BEL ; Character = <BEL> ?
1D99	CAC61D	=4086	JZ FIOEX ; Yes - Ignore it and exit

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 74

LOC	OBJ	LINE	SOURCE STATEMENT
1DE2	C9	=4157	RET ; Return from <FPHEX>
		=4158 ;	
		=4159 ;=====	
		=4160 ;*	
		=4161 ;*	Function : HEADER (Print list file header)
		=4162 ;*	Inputs : None.
		=4163 ;*	Outputs : None.
		=4164 ;*	Destroys : A,H,L and Flags.
		=4165 ;*	Calls : PRINT, CRLF, STRING and PRINTD.
		=4166 ;*	
		=4167 ;*	Description : This routine prints a list file page header.
		=4168 ;*	
		=4169 ;=====	
		=4170 ;	
		=4171	HEADER: ; Start of <HEADER> routine
1DE3	3E0C	=4172	MVI A,FF ;
1DE5	CDC61E	=4173	CALL PRINT ; Print a Form feed.
		=4174	;
1DE8	CD831D	=4175	CALL CRLF ; Start new listing line.
		=4176	;
1DEB	21692F	=4177	LXI H,HEAD1 ; Print 1st line of Page Header.
1DEE	CDB51F	=4178	CALL STRING ;
		=4179	;
1DF1	2A08E8	=4180	LHLD PAGNUM ;
1DF4	23	=4181	INX H ; Inc Page number before printing.
1DF5	2208E8	=4182	SHLD PAGNUM ;
1DF8	CDB721	=4183	CALL PRINTD ; Print the Page number (in Decimal)
		=4184	;
1DFB	CD831D	=4185	CALL CRLF ;
1DFE	CD831D	=4186	CALL CRLF ; Leave 2nd line of page Header blank.
1E01	218C2F	=4187	LXI H,HEAD2 ; Print 3rd line of Page Header.
1E04	CDB51F	=4188	CALL STRING ;
1E07	CD831D	=4189	CALL CRLF ;
		=4190	;
1E0A	3E03	=4191	MVI A,3 ;
1E0C	320BE8	=4192	STA PLINES ; Set lines on page = 3 (lines in the header)
		=4193	;
1E0F	C9	=4194	RET ; Return from the <HEADER> routine.
		=4195 ;	
		=4196 ;=====	
		=4197 ;*	
		=4198 ;*	Function : NEXTPL. (Next print line.)
		=4199 ;*	Inputs : None.
		=4200 ;*	Outputs : None.
		=4201 ;*	Destroys : Acc and Flags.
		=4202 ;*	Calls : CRLF and HEADER.
		=4203 ;*	
		=4204 ;*	Description: This routine is called after every listing
		=4205 ;*	line has been printed. It sends a <CR> &
		=4206 ;*	<LF> to the screen, (If PDPFLG=False),
		=4207 ;*	or it checkes of the number of lines on
		=4208 ;*	the current page and starts a new page if
		=4209 ;*	it is needed.
		=4210 ;*	
		=4211 ;*	Note: If the new_page Flag is set (By \$EJECT control)
		=4212 ;*	then a new page is started.
		=4213 ;*	
		=4214 ;=====	
		=4215 ;	
		=4216	NEXTPL: ; Start of next print line routine.
1E10	E5	=4217	PUSH H ;
1E11	D5	=4218	PUSH D ;
1E12	C5	=4219	PUSH B ;
		=4220	;
1E13	3A20E8	=4221	LDA PDPFLG ;
1E16	FEFF	=4222	CPI TRUE ; If Not PDPFLG then just print <CR><LF>
1E18	C2431E	=4223	JNZ SCREEN ;
		=4224	;
1E1B	3A05E8	=4225	LDA NEWPAG ;
1E1E	FEFF	=4226	CPI TRUE ; If NEWPAGE = true then

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 75

LOC	OBJ	LINE	SOURCE STATEMENT
1E20	C22E1E	=4227	JNZ NOTNP ;
		=4228	;
1E23	CDE31D	=4229	CALL HEADER ; Print a header (Top of next page)
1E26	3E00	=4230	MVI A,FALSE ;
1E28	3205E8	=4231	STA NEWPAG ; Reset New Page Flag.
1E2B	C33C1E	=4232	JMP NOTBOP ;
		=4233	;
		=4234	NOTNP: ;
1E2E	3A0AE8	=4235	LDA LPP ;
1E31	67	=4236	MOV H,A ;
1E32	3A0BE8	=4237	LDA PLINES ;
1E35	BC	=4238	CMP H ; If Lines_Per_Page < Page_LINES then
		=4239	; Print header and start another page.
1E36	FA3C1E	=4240	JM NOTBOP ;
		=4241	;
1E39	CDE31D	=4242	CALL HEADER ; Print page header.
		=4243	NOTBOP: ;
1E3C	3A0BE8	=4244	LDA PLINES ;
1E3F	3C	=4245	INR A ; Inc Number of lines on the page.
1E40	320BE8	=4246	STA PLINES ;
		=4247	SCREEN: ;
1E43	CD831D	=4248	CALL CRLF ; Start new line on console.
		=4249	NXTPL: ;
1E46	C1	=4250	POP B ;
1E47	D1	=4251	POP D ;
1E48	E1	=4252	POP H ; Restore registers.
		=4253	;
1E49	C9	=4254	RET ; Return from <NEXTPL>
		=4255	;
		=4256	=====
		=4257	;
		=4258	Function : OPENFILE (Open PDP output file) *
		=4259	Inputs : None. *
		=4260	Outputs : Z=1 : File open correctly *
		=4261	Z=0 : Error while trying to open file. *
		=4262	Destroys : A, H, L and Flags. *
		=4263	Calls : PDPSTR, PDPIN and PERR. *
		=4264	;
		=4265	Description : This routine opens an output file on the PDP *
		=4266	via the KIT program. *
		=4267	;
		=4268	If either a LIST or an OBJECT file is already open then *
		=4269	an error is reported. *
		=4270	;
		=4271	This routine initializes the file output pointers and *
		=4272	buffer length counter. *
		=4273	;
		=4274	NOTE: See documentation for more information about the KIT *
		=4275	program. *
		=4276	;
		=4277	=====
		=4278	;
		=4279	OPENFILE: ; Start of open PDP file routine.
1E4A	3A20E8	=4280	LDA PDPFLG ;
1E4D	FE00	=4281	CPI FALSE ;
1E4F	C29A1E	=4282	JNZ OPENE2 ; If List file already open then error.
		=4283	;
1E52	3AC7E8	=4284	LDA OBJFLG ;
1E55	FE00	=4285	CPI FALSE ;
1E57	C29A1E	=4286	JNZ OPENE2 ; If Object file already open then error.
		=4287	;
1E5A	21F42E	=4288	LXI H,KITMSG ;
1E5D	CDA31E	=4289	CALL PDPSTR ; Send 'KIT '
1E60	2121E8	=4290	LXI H,FNAME ;
1E63	CDA31E	=4291	CALL PDPSTR ; Send file name.
1E66	21F82E	=4292	LXI H,TERMKIT ;
1E69	CDA31E	=4293	CALL PDPSTR ; Send '/S<CR>'
		=4294	;
		=4295	OPEN05: ;
1E6C	CD16F0	=4296	CALL PDPIN ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 76

LOC	OBJ	LINE	SOURCE STATEMENT
1E6F	FE40	=4297	CPI KITACK ;
1E71	C26C1E	=4298	JNZ OPEN05 ;
		=4299	;
1E74	CD16F0	=4300	CALL PDPIN ; Fetch KIT error code.
1E77	FE30	=4301	CPI '0' ; No Error ?
1E79	C2951E	=4302	JNZ OPENE1 ;
		=4303	;
		=4304	OPEN10: ;
1E7C	CD16F0	=4305	CALL PDPIN ;
1E7F	FE40	=4306	CPI KITACK ;
1E81	C27C1E	=4307	JNZ OPEN10 ; Wait for Second ACK.
		=4308	;
1E84	213FE8	=4309	LXI H,LINEB ;
1E87	22C3E8	=4310	SHLD LINEBP ; Setup line pointer to the start of the buffer
1E8A	3E00	=4311	MVI A,0 ; Set zero flag = File opened OK.
1E8C	32C5E8	=4312	STA LINEBL ; Setup character counter.
1E8F	32C6E8	=4313	STA PLUSOL ; Setup Plus Counter.
1E92	C3A11E	=4314	JMP OPENFE ;
		=4315 ;	
		=4316 ;	
		=4317 ;	
		=4318 ;	If the status returned by KIT (the xfer task running on the PDP)
		=4319 ;	is not zero (no error) then print an error message indicating
		=4320 ;	the fact that the file could not be opened.
		=4321 ;	
		=4322	OPENE1: ; Open file error number 1
1E95	3E08	=4323	MVI A,8 ;
1E97	C39C1E	=4324	JMP OPENEE ; Exit file open routine with error #8
		=4325 ;	
		=4326 ;	
		=4327 ;	
		=4328 ;	If either the OBJFLG or the PDPFLG are set then a file is already
		=4329 ;	open and as another cannot be opened, print an error message.
		=4330 ;	
		=4331	OPENE2: ; Open file error number 2
1E9A	3E24	=4332	MVI A,36 ; Exit file open routine with error #36
		=4333	OPENEE: ;
1E9C	CD3E21	=4334	CALL PERR ; Print an error message.
1E9F	3EFF	=4335	MVI A,OFFH ; Clear Zero flag, error return.
		=4336 ;	
		=4337 ;	
		=4338 ;	
		=4339 ;	Final Open file exit point
		=4340 ;	
		=4341	OPENFE: ; Open file final exit
1EA1	B7	=4342	ORA A ; Set flags before returning
1EA2	C9	=4343	RET ; Return from <OPENFILE>
		=4344 ;	
		=4345 ;	=====
		=4346 ;*	
		=4347 ;*	Function : PDPSTR (PDP string routine) *
		=4348 ;*	Inputs : HL = Start address of the string. *
		=4349 ;*	Outputs : None. *
		=4350 ;*	Destroys : H and L. *
		=4351 ;*	Calls : PDPIN and PDPOUT. *
		=4352 ;*	
		=4353 ;*	Description : This routine sends a string of characters to *
		=4354 ;*	the PDP. The start address of the string is passed in the *
		=4355 ;*	HL register pair and the end of the string is marked by *
		=4356 ;*	the last character having it's sign bit set. The routine *
		=4357 ;*	waites for the echo from the PDP before returning. *
		=4358 ;*	
		=4359 ;	=====
		=4360 ;	
		=4361	PDPSTR: ; Start of the <PDPSTR> routine.
1EA3	F5	=4362	PUSH PSW ; Save the Acc and flags.
		=4363	PDPS05: ;
1EA4	7E	=4364	MOV A,M ; Fetch the character
1EA5	B7	=4365	ORA A ; Set flags.
1EA6	FAB31E	=4366	JM PDPS10 ; Check sign bit (Signbit.set => end of string)

LOC	OBJ	LINE	SOURCE STATEMENT
		=4367	
1EA9	CD1FF0	=4368	CALL PDPOUT ; Send the character to the PDP
1EAC	CD16F0	=4369	CALL PDPIN ; Wait for the echo
1EAF	23	=4370	INX H ; Inc pointer
1EB0	C3A41E	=4371	JMP PDPS05 ; and loop until sign bit set.
		=4372 PDPS10:	; Exit point.
1EB3	E67F	=4373	ANI OFFMSB ; Remove the top bit of the last character
1EB5	CD1FF0	=4374	CALL PDPOUT ; Send it to the PDP
1EB8	CD16F0	=4375	CALL PDPIN ; Wait for the echo
1EBB	F1	=4376	POP PSW ; Restore the registers
1EBC	C9	=4377	RET ; Return from the <PDPSTR> routine
		=4378 ;	
		=4379 ;=====	
		=4380 ;*	
		=4381 ;*	Function : PHL (Print HL register pair) *
		=4382 ;*	Inputs : HL = 16 number to print. *
		=4383 ;*	Outputs : None. *
		=4384 ;*	Destroys : A and Flags. *
		=4385 ;*	Calls : PRINTH. *
		=4386 ;*	
		=4387 ;*	Description : This routine outputs the HL register pair *
		=4388 ;*	(a 16 bit number) as 4 hexadecimal digits. *
		=4389 ;*	
		=4390 ;=====	
		=4391 ;	
		=4392 PHL:	; Start of the <PHL> routine
		=4393	
1EBD	7C	=4394	MOV A,H ; Load acc with msbyte of pointer
1EBE	CDDC1E	=4395	CALL PRINTH ; Output as hex value.
		=4396	
1EC1	7D	=4397	MOV A,L ; Load acc with lsbyte of pointer
1EC2	CDDC1E	=4398	CALL PRINTH ; Output as hex value.
		=4399	
1EC5	C9	=4400	RET ; Return from the <PHL> routine
		=4401 ;	
		=4402 ;=====	
		=4403 ;*	
		=4404 ;*	Function : PRINT *
		=4405 ;*	Inputs : A reg = ASCII character to print. *
		=4406 ;*	Outputs : None. *
		=4407 ;*	Destroys : Acc and Flags *
		=4408 ;*	Calls : FILEOUT and TERMOUT. *
		=4409 ;*	
		=4410 ;*	Description: This is the basic assembler output routine and *
		=4411 ;*	it therefore directs the output to terminal or the PDP *
		=4412 ;*	depending on the \$LISTFILE(<Filename>) control. *
		=4413 ;*	
		=4414 ;=====	
		=4415 ;	
		=4416 PRINT:	; Start of character output routine.
1EC6	C5	=4417	PUSH B ;
1EC7	47	=4418	MOV B,A ;
1EC8	3A20E8	=4419	LDA PDFFLG ;
1ECB	FE00	=4420	CPI FALSE ; If PDP_Flag = TRUE Then
1ECD	78	=4421	MOV A,B ; Begin
1ECE	C1	=4422	POP B ;
		=4423	
1ECF	CAD81E	=4424	JZ NOFOUT ;
1ED2	CD8E1D	=4425	CALL FILEOUT ; FILEOUT(ASCII_Character)
1ED5	C3DB1E	=4426	JMP PRTE ; End
		=4427	; Else
		=4428 NOFOUT:	; Begin
1ED8	CDCB1F	=4429	CALL TERMOUT ; TERMINAL_OUT(ASCII_Character)
		=4430 PRTE:	; End;
		=4431	
1EDB	C9	=4432	RET ; Return from <PRINT> routine.
		=4433 ;	
		=4434 ;=====	
		=4435 ;*	
		=4436 ;*	Function : PRINTH (Print hexadecimal) *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 78

LOC	OBJ	LINE	SOURCE STATEMENT
		=4437 ;*	Inputs : A = 8 bit number to print. *
		=4438 ;*	Outputs : None. *
		=4439 ;*	Destroys : A and Flags. *
		=4440 ;*	Calls : CONHEX and PRINT. *
		=4441 ;*	
		=4442 ;*	Description : This routine prints the 8 bit A register as *
		=4443 ;*	2 hexadecimal digits. The "PRINT" output routine is used *
		=4444 ;*	to output the characters. *
		=4445 ;*	
		=4446 ;*	=====
		=4447 ;	
		=4448	PRINTH: ; Start of the <PRINTH> routine
1EDC D5		=4449	PUSH D ; Save the DE register pair
		=4450	;
1EDD 5F		=4451	MOV E,A ; Save value in E reg.
1EDE E6F0		=4452	ANI 11110000B ; Mask off most sig nibble of value.
1EE0 1F		=4453	RAR ;
1EE1 1F		=4454	RAR ; Shift acc 4 bits right.
1EE2 1F		=4455	RAR ;
1EE3 1F		=4456	RAR ;
1EE4 CD791D		=4457	CALL CONHEX ;
1EE7 CDC61E		=4458	CALL PRINT ; Output the char.
		=4459	;
1EEA 7B		=4460	MOV A,E ; Restore orig value.
1EEB E60F		=4461	ANI 00001111B ; Mask off least sig nibble of value.
1EED CD791D		=4462	CALL CONHEX ;
1EF0 CDC61E		=4463	CALL PRINT ; Output the char.
		=4464	;
1EF3 D1		=4465	POP D ; Restore DE
1EF4 C9		=4466	RET ; Return from <PRINTH>.
		=4467 ;	
		=4468 ;*	=====
		=4469 ;*	
		=4470 ;*	Function : SENDLINE. *
		=4471 ;*	Inputs : Character_count - Memory - (LINEBL) *
		=4472 ;*	Character_buffer - Memory - (LINEB+) *
		=4473 ;*	Outputs : None *
		=4474 ;*	Destroys : H,L,E,Acc and Flags *
		=4475 ;*	Calls : PDPIN, PDPOUT and KITOUT. *
		=4476 ;*	
		=4477 ;*	Description: This routine sends the line stored in Memory to *
		=4478 ;*	the PDP. It also places a plus sign on the screen for every *
		=4479 ;*	line sent and starts a new line every 50. It transfer to *
		=4480 ;*	the PDP is done using "KIT". See documentation for more *
		=4481 ;*	details about "KIT". *
		=4482 ;*	
		=4483 ;*	=====
		=4484 ;	
		=4485	SENDLINE: ; Send line to the PDP.
1EF5 3AC5E8		=4486	LDA LINEBL ;
1EF8 FE00		=4487	CPI 0 ; Zero length line ?
1EFA CA0D1F		=4488	JZ SLEXIT ; Yes - then do not send it.
		=4489	;
1EFD 5F		=4490	MOV E,A ; Else, Save character count in E reg.
1EFE 213FE8		=4491	LXI H,LINEB ; Get start adr of line buffer.
		=4492	SND05: ; For I:= 1 to Char_count [E reg] Do
1F01 7E		=4493	MOV A,M ; Begin
1F02 CD1FF0		=4494	CALL PDPOUT ; Pdp_out(line_buffer(I)) ;
1F05 CD16F0		=4495	CALL PDPIN ; Dummy := Pdp_int ; (* Get Echo *)
1F08 23		=4496	INX H ;
1F09 1D		=4497	DCR E ;
1FOA C2011F		=4498	JNZ SND05 ; End;
		=4499	SLEXIT: ;
1F0D 3E0D		=4500	MVI A,CR ; Pdp_out(CR) ;
1F0F CD1FF0		=4501	CALL PDPOUT ;
		=4502	SND10: ; Repeat
1F12 CD16F0		=4503	CALL PDPIN ; ACC = Pdp_in ; (* wait for KITACK *)
1F15 FE40		=4504	CPI KITACK ; Until ACC = KITACK
1F17 C2121F		=4505	JNZ SND10 ;
		=4506	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 79

LOC	OBJ	LINE	SOURCE STATEMENT
1F1A	3AC7E8	=4507	LDA OBJFLG ; Check To see if the File is a list or
1F1D	FEFF	=4508	CPI TRUE ; Object file. (if Object then no plus
1F1F	CA3F1F	=4509	JZ NOPLUS ; Signs on the screen)
		=4510	;
1F22	3AC6E8	=4511	LDA PLUSOL ;
1F25	FE32	=4512	CPI MAXPOL ; If PLUS_On_Line = MAX Plus_On_Line
1F27	C2361F	=4513	JNZ SND15 ;
		=4514	;
1F2A	3E0D	=4515	MVI A,CR ; Send a CR + LF to the terminal.
1F2C	CD1CF0	=4516	CALL KITOUT ;
1F2F	3E0A	=4517	MVI A,LF ;
1F31	CD1CF0	=4518	CALL KITOUT ;
		=4519	;
1F34	3E00	=4520	MVI A,0 ; Set Plus Signs = 0
		=4521	;
		=4522	SND15: ; Endif
		=4523	;
1F36	3C	=4524	INR A ; Inc Plus Signs on the line
1F37	32C6E8	=4525	STA PLUSOL ;
		=4526	;
1F3A	3E2B	=4527	MVI A,'+' ; Send a Plus Sign to the terminal
1F3C	CD1CF0	=4528	CALL KITOUT ;
		=4529	NOPLUS: ;
1F3F	C9	=4530	RET ; Return from <SEND_LINE>
		=4531	;
		=4532	=====
		=4533	;
		=4534	Function : SENDHR (Send Hex Record) *
		=4535	Inputs : A data block in memory consisting of the *
		=4536	following data items: *
		=4537	;
		=4538	Name: Function *
		=4539	-----
		=4540	OBJC The number of bytes in the record. *
		=4541	OBJBA The start adr of the Record. *
		=4542	;
		=4543	Outputs : None *
		=4544	Destroys : A, B, H, L and flags *
		=4545	Calls : FILEOUT, FPHEX and BCC *
		=4546	;
		=4547	Description: This subroutine outputs a Hex record with *
		=4548	the given base_adr and byte count from the data *
		=4549	starting at OBJBUF in memory. *
		=4550	;
		=4551	NOTE: The checksum is calculated as follows. *
		=4552	;
		=4553	Checksum := -((Byte_count + Low base_adr + High base_adr + *
		=4554	+ Record_type + sum_of_data_byte) mod 256) *
		=4555	;
		=4556	Record_type = 0 : data record. *
		=4557	Record_type = 1 : EOF record. *
		=4558	;
		=4559	=====
		=4560	;
		=4561	SENDHR: ; Send Hex record.
1F40	3ADEE8	=4562	LDA OBJC ;
1F43	FE00	=4563	CPI 0 ;
1F45	CAA51F	=4564	JZ SNDREX ; If Byte_count <> 0 then
		=4565	begin ;
1F48	3E3A	=4566	MVI A,':' ;
1F4A	CD8E1D	=4567	CALL FILEOUT ; Fileout(':') ;
1F4D	3E00	=4568	MVI A,0 ;
1F4F	32E2E8	=4569	STA CHKSUM ; Checksum := 0 ;
		=4570	;
1F52	3ADEE8	=4571	LDA OBJC ;
1F55	CD0D1D	=4572	CALL BCC ; Checksum:=(Checksum+Byte_count) mod 256;
1F58	CDCA1D	=4573	CALL FPHEX ; Fileout(Hex\$(Byte_count));
		=4574	;
1F5B	2ADAE8	=4575	LHLD OBJBA ;
1F5E	7C	=4576	MOV A,H ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 80

LOC	OBJ	LINE	SOURCE STATEMENT
1F5F	CD0D1D	=4577	CALL BCC ;
1F62	CDCA1D	=4578	CALL FPHEX ;
		=4579	Checksum:=(Checksum+Low Base_adr) mod 256;
			Fileout(Hex\$(Low base_adr));
1F65	7D	=4580	MOV A,L ;
1F66	CD0D1D	=4581	CALL BCC ;
1F69	CDCA1D	=4582	CALL FPHEX ;
		=4583	Checksum:=(Checksum+high base_adr)mod 256;
			Fileout(Hex\$(high base_adr));
1F6C	3E00	=4584	MVI A,0 ;
1F6E	CDCA1D	=4585	CALL FPHEX ;
		=4586	Fileout(Hex\$(0)); (* Data type = 0 *)
1F71	21C8E8	=4587	LXI H,OBJBUF ;
1F74	3ADEE8	=4588	LDA OBJC ;
1F77	47	=4589	MOV B,A ;
		=4590	OBLOOP: ;
1F78	7E	=4591	MOV A,M ;
1F79	CD0D1D	=4592	CALL BCC ;
1F7C	CDCA1D	=4593	CALL FPHEX ;
1F7F	23	=4594	INX H ;
1F80	05	=4595	DCR B ;
1F81	C2781F	=4596	JNZ OBLOOP ;
		=4597	End;
1F84	3AE2E8	=4598	LDA CHKSUM ;
1F87	2F	=4599	CMA ;
1F88	3C	=4600	INR A ;
1F89	CDCA1D	=4601	CALL FPHEX ;
1F8C	3E0D	=4602	MVI A,CR ;
1F8E	CD8E1D	=4603	CALL FILEOUT ;
		=4604	Fileout(CR);
1F91	3E00	=4605	MVI A,0 ;
1F93	32DEE8	=4606	STA OBJC ;
1F96	21C8E8	=4607	LXI H,OBJBUF ;
1F99	22D8E8	=4608	SHLD OBJBP ;
1F9C	210000	=4609	LXI H,0 ;
1F9F	22DAE8	=4610	SHLD OBJBA ;
1FA2	22DCE8	=4611	SHLD OBJNA ;
		=4612	SNDREX: ;
1FA5	C9	=4613	RET ;
		=4614 ;	
		=4615 ;	=====
		=4616 ;*	*
		=4617 ;*	Function : SPACES (Print n spaces) *
		=4618 ;*	Inputs : A = Number of spaces to print. *
		=4619 ;*	Outputs : None. *
		=4620 ;*	Destroys : None. *
		=4621 ;*	Calls : PRINT. *
		=4622 ;*	*
		=4623 ;*	Description : This routine sends n spaces to the output *
		=4624 ;*	device. *
		=4625 ;*	*
		=4626 ;	=====
		=4627 ;	
		=4628	SPACES: ; Start of SPACES routine.
1FA6	F5	=4629	PUSH PSW ; Save A and flags.
1FA7	C5	=4630	PUSH B ; Save B and C
1FA8	47	=4631	MOV B,A ; Move spaces_to_print to B_reg
		=4632	SPLOOP: ;
1FA9	3E20	=4633	MVI A,' ' ;
1FAB	CDC61E	=4634	CALL PRINT ; Output a space
1FAE	05	=4635	DCR B ; Dec spaces_to_print counter
1FAF	C2A91F	=4636	JNZ SPLOOP ; If not zero then loop and print another space
		=4637	;
1FB2	C1	=4638	POP B ; Restore B and C
1FB3	F1	=4639	POP PSW ; Restore A and Flags
		=4640	;
1FB4	C9	=4641	RET ; Return from <SPACES>
		=4642 ;	
		=4643 ;	=====
		=4644 ;*	*
		=4645 ;*	Function : STRING. (string output routine) *
		=4646 ;*	Inputs : HL = Start adr of string to print. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 81

LOC	OBJ	LINE	SOURCE STATEMENT
		=4647 ;	Outputs : None. *
		=4648 ;	Destroys : H and L registers. *
		=4649 ;	Calls : PRINT. *
		=4650 ;	;
		=4651 ;	Description: This routine prints the given string on the *
		=4652 ;	screen. The start adr of the string is passed in the HL *
		=4653 ;	register pair and the string is terminated by the last char *
		=4654 ;	in the string having it's MSBit set. *
		=4655 ;	;
		=4656 ;	=====
		=4657 ;	;
		=4658	STRING: ; Start of <STRING>
1FB5	F5	=4659	PUSH PSW ; Save Acc and flags.
		=4660	STLOOP: ;
1FB6	7E	=4661	MOV A,M ; Get next character from string.
1FB7	B7	=4662	ORA A ; Test for end_of_string (MSBit set)
1FB8	FAC41F	=4663	JM ENDST ; Yes - Then end of string has been reached.
		=4664	;
1FBB	E67F	=4665	ANI OFFMSB ; Remove MSBit.
1FBD	CDC61E	=4666	CALL PRINT ; Print the character in the Acc.
1FC0	23	=4667	INX H ; Inc string pointer to next character.
1FC1	C3B61F	=4668	JMP STLOOP ; Loop until <EOT> found.
		=4669	;
		=4670	ENDST: ; End of string.
1FC4	E67F	=4671	ANI OFFMSB ;
1FC6	CDC61E	=4672	CALL PRINT ; Print the last character.
1FC9	F1	=4673	POP PSW ; Restore Acc and flags.
1FCA	C9	=4674	RET ; Return from <STRING>.
		=4675 ;	;
		=4676 ;	=====
		=4677 ;	;
		=4678 ;	Function : TERMOUT (Terminal output routine) *
		=4679 ;	Inputs : A = Character to output. *
		=4680 ;	Outputs : None. *
		=4681 ;	Destroys : None. *
		=4682 ;	Calls : BREAK and KITOUT. *
		=4683 ;	;
		=4684 ;	Description : This routine outputs a character to the *
		=4685 ;	console (with flow control). *
		=4686 ;	;
		=4687 ;	=====
		=4688 ;	;
		=4689	TERMOUT: ; Start of <TERMOUT> routine
		=4690 ;	;
		=4691 ;	Implement flow control via the monitor break routine.
		=4692 ;	;
1FCB	F5	=4693	PUSH PSW ;
1FCC	CD1A1D	=4694	CALL BREAK ; Use monitor flow control routine.
1FCF	F1	=4695	POP PSW ;
		=4696 ;	;
		=4697 ;	Output the character via the monitor output routine.
		=4698 ;	;
1FD0	CD1CF0	=4699	CALL KITOUT ; Send character to the Terminal.
		=4700 ;	;
1FD3	C9	=4701	RET ; Return from the <TERMOUT> routine
		=4702 ;	;
		=4703 ;	End of assembler I/O routines. (IO.A85)
		4704	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 82

LOC	OBJ	LINE	SOURCE STATEMENT
		4705	\$INCLUDE(ASMPLINE.A85)
=4706			-----
=4707			;
=4708			Line output routine module for the UCT SABus assembler
=4709			-----
=4710			;
=4711			This module contains (in order) the following subroutines:
=4712			;
=4713			a) PADR - Print address routine.
=4714			b) PCODEL - Print code line routine.
=4715			c) PEQUL - Print equate line routine.
=4716			d) PLINE - Print line routine (MAIN ROUTINE)
=4717			e) PRTCB - Print code byte routine.
=4718			f) PTEXT - Print text routine.
=4719			;
=4720			-----
=4721			;
=4722			\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=4723 ;	=====
		=4724 ;*	*
		=4725 ;*	Function : PADR (Print address). *
		=4726 ;*	Inputs : None. *
		=4727 ;*	Outputs : None. *
		=4728 ;*	Destroys : H, L, A and Flags. *
		=4729 ;*	Calls : PHL and SPACES. *
		=4730 ;*	*
		=4731 ;*	Description: This routine prints the Program_counter if the *
		=4732 ;*	Print_adr_flag = True, otherwise it prints 4 spaces. *
		=4733 ;*	*
		=4734 ;	=====
		=4735 ;	
		=4736 PADR:	; Start of Print adr routine.
1FD4	3A6AE9	=4737 LDA	PAFLAG ;
1FD7	FE00	=4738 CPI	FALSE ;
1FD9	CAE51F	=4739 JZ	DONTPA ; If Print_adr_flag = true then
		=4740 ;	
1FDC	2A67E9	=4741 LHLD	LASTPC ;
1FDF	CDBD1E	=4742 CALL	PHL ; Print LASTPC as 4 digit hex number
1FE2	C3EA1F	=4743 JMP	PADREX ;
		=4744 ;	; Else
		=4745 DONTPA:	;
1FE5	3E04	=4746 MVI	A,4 ; Print 4 spaces (Same len as hex number)
1FE7	CDA61F	=4747 CALL	SPACES ;
		=4748 ;	
		=4749 PADREX:	;
1FEA	C9	=4750 RET	; Return from <PADR>
		=4751 ;	
		=4752 ;	=====
		=4753 ;*	*
		=4754 ;*	Function : PCODEL (Print code line). *
		=4755 ;*	Inputs : None. *
		=4756 ;*	Outputs : None. *
		=4757 ;*	Destroys : All. *
		=4758 ;*	Calls : PADR, SPACES, DCMPL, PRTCB, PTEXT and NEXTPL. *
		=4759 ;*	*
		=4760 ;*	Description: This routine prints the listing line for a *
		=4761 ;*	code statement. The format of a code statement listing *
		=4762 ;*	line is as follows: *
		=4763 ;*	*
		=4764 ;*	AAAA CCCCCCCC NNNN <TAB> <Source statement> *
		=4765 ;*	CCCCCCCC *
		=4766 ;*	CCCC *
		=4767 ;*	*
		=4768 ;*	Where: AAAA = Code adr. *
		=4769 ;*	CCCCCCCC = Code bytes. *
		=4770 ;*	NNNN = Line number. *
		=4771 ;*	*
		=4772 ;*	Note: The code bytes can take more that one line to *
		=4773 ;*	print. Every Code listing line can print 4 hex *
		=4774 ;*	code bytes, therefore if a source line *
		=4775 ;*	generates 10 bytes (As above) then it will *
		=4776 ;*	need 3 listing lines to print it. *
		=4777 ;*	*
		=4778 ;	=====
		=4779 ;	
		=4780 PCODEL:	; Start of <PCODEL>
1FEB	CDD41F	=4781 CALL	PADR ; Print the assembler address .
		=4782 ;	
1FEE	3E01	=4783 MVI	A,1 ;
1FF0	CDA61F	=4784 CALL	SPACES ; Print a space
		=4785 ;	
1FF3	0604	=4786 MVI	B,4 ; Init bytes left on line = 4
		=4787 ;	
1FF5	3E00	=4788 MVI	A,FALSE ;
1FF7	320CE8	=4789 STA	MOREH ; Init More_hex = False.
		=4790 ;	
1FFA	3A0EE8	=4791 LDA	GCFLAG ;
1FFD	FE00	=4792 CPI	FALSE ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 84

LOC	OBJ	LINE	SOURCE STATEMENT
1FFF	CA3520	=4793	JZ NOMORE ; IF Gen_code_flag = true then Print_code.
		=4794	;
2002	2A65E9	=4795	LHLD PC ;
2005	EB	=4796	XCHG ;
2006	2A67E9	=4797	LHLD LASTPC ; DE = New PC / HL = old PC .
		=4798	PNHEX: ;
2009	CD2313	=4799	CALL DCMP ; New_PC = Last_PC ?
200C	CA3520	=4800	JZ NOMORE ; Yes - then all the code has been printed.
		=4801	;
200F	7E	=4802	MOV A,M ;
2010	CDC620	=4803	CALL PRTCB ; Print code byte.
2013	23	=4804	INX H ; Inc Code byte pointer.
2014	05	=4805	DCR B ; Dec bytes left on line.
2015	C20920	=4806	JNZ PNHEX ; Loop until all code printed or there are 4
		=4807	code bytes already on the current line.
		=4808	;
2018	CD2313	=4809	CALL DCMP ; If The 4th byte was the last then
		=4810	do not set more_hex flag.
201B	CA3E20	=4811	JZ ENDPH ;
		=4812	;
201E	3EFF	=4813	MVI A,TRUE ;
2020	320CE8	=4814	STA MOREH ; More_hex = True.
		=4815 ;	
		=4816 ;	-----
		=4817 ;	
		=4818 ;	If the brief flag is set then add a plus sign to the 4 code bytes
		=4819 ;	and then only print 1 space and no 2.
		=4820 ;	
2023	3A0DE8	=4821	LDA BRIEFF ;
2026	FE00	=4822	CPI FALSE ; Brief flag = False ?
2028	CA3E20	=4823	JZ ENDPH ;
		=4824	;
202B	3E2B	=4825	MVI A,'+' ;
202D	CDC61E	=4826	CALL PRINT ; Print a plus sign
2030	3E01	=4827	MVI A,1 ;
2032	C34020	=4828	JMP ENDPH1 ; Print 1 space and the text line
		=4829 ;	
		=4830 ;	-----
		=4831 ;	
		=4832 ;	Less that 4 code bytes were printed, Pad to correct column.
		=4833 ;	
		=4834	NOMORE: ; No more Hex value to print.
		=4835	; (Pad to correct column).
2035	3E02	=4836	MVI A,2 ;
2037	CDA61F	=4837	CALL SPACES ; Print 2 spaces per byte.
203A	05	=4838	DCR B ; Dec bytes left on line until = zero.
203B	C23520	=4839	JNZ NOMORE ;
		=4840	;
		=4841 ;	
		=4842 ;	-----
		=4843 ;	
		=4844 ;	Code bytes have been printed, now print source line.
		=4845 ;	
		=4846	ENDPH: ;
203E	3E02	=4847	MVI A,2 ; Print 2 spaces
		=4848	ENDPH1: ;
2040	CDA61F	=4849	CALL SPACES ; Print the number of spaces given by Areg
		=4850	;
2043	2267E9	=4851	SHLD LASTPC ; Save current code gen pointer .
2046	CDE020	=4852	CALL PTEXT ; Print the source text line.
		=4853 ;	
		=4854 ;	-----
		=4855 ;	
		=4856 ;	Source line has been printed. (Print extra code lines if needed.)
		=4857 ;	
2049	3A0CE8	=4858	LDA MOREH ;
204C	FE00	=4859	CPI FALSE ;
204E	CA8520	=4860	JZ EXPLIN ; If More_hex = false then exit.
		=4861	;
2051	3A0DE8	=4862	LDA BRIEFF ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 85

LOC	OBJ	LINE	SOURCE STATEMENT
2054	FEFF	=4863	CPI TRUE ;
2056	CA8520	=4864	JZ EXPLIN ; If brief_flag = True then exit.
		=4865	;
2059	2A65E9	=4866	LHLD PC ;
205C	EB	=4867	XCHG ;
205D	2A67E9	=4868	LHLD LASTPC ;
		=4869	PNLOH: ;
2060	3E08	=4870	MVI A,8 ; Space to correct column.
2062	CDA61F	=4871	CALL SPACES ;
		=4872	;
2065	0604	=4873	MVI B,4 ; Init code bytes left on line.
		=4874	PNXTHD: ;
2067	CD2313	=4875	CALL DCMP ; Last_PC = PC ?
206A	CA8220	=4876	JZ ENDHEX ; Yes - then all code bytes printed.
		=4877	;
206D	7E	=4878	MOV A,M ; Fetch code byte.
206E	CDC620	=4879	CALL PRTCB ; Print code byte.
2071	23	=4880	INX H ; Inc Last_PC
2072	05	=4881	DCR B ; Dec code bytes on line
2073	C26720	=4882	JNZ PNXTHD ; Loop until 4 bytes printed or
		=4883	all bytes printed.
		=4884	;
2076	CD101E	=4885	CALL NEXTPL ; Next print line.
2079	CD2313	=4886	CALL DCMP ; Was the Last byte on that line the last
		=4887	byte for this source line ?
207C	CA8520	=4888	JZ EXPLIN ; Yes - Then exit print code line routine.
207F	C36020	=4889	JMP PNLOH ;
		=4890	;
		=4891	ENDHEX: ;
2082	CD101E	=4892	CALL NEXTPL ; Next print line.
		=4893	;
		=4894	EXPLIN: ; Exit point for Print line routine.
		=4895	;
2085	C9	=4896	RET ; Return from <PCODEL>
		=4897	;
		=4898	=====
		=4899	;
		=4900	Function : PEQUL (Print equate line.) *
		=4901	Inputs : None. *
		=4902	Outputs : None. *
		=4903	Destroys : All. *
		=4904	Calls : FETCH, PHL, SPACES, DCMP, NEXTPL, PRINTD and *
		=4905	PRINT. *
		=4906	;
		=4907	Description: This routine prints the listing line for an *
		=4908	equate source line. The format of an equate listing line *
		=4909	is as follows: *
		=4910	;
		=4911	XXXX LLLL <TAB> <LABEL> EQU <Expression> *
		=4912	(Exp value) (Line #) (Source statement) *
		=4913	;
		=4914	=====
		=4915	;
		=4916	PEQUL: ; Start of <PEQUL>
2086	CD2E18	=4917	CALL FETCH ; Fetch value in label buffer.
2089	2AE9E8	=4918	LHLD LABELB+6 ; HL = Value of defined symbol.
208C	CDBD1E	=4919	CALL PHL ; Print HL as 4 digit hex number.
208F	3E0B	=4920	MVI A,11 ;
2091	CDA61F	=4921	CALL SPACES ; Print 11 spaces.
2094	CDE020	=4922	CALL PTEXT ; Print the source text
2097	C9	=4923	RET ; Return from <PEQUL>
		=4924	;
		=4925	=====
		=4926	;
		=4927	Function : PLINE (Print line routine) *
		=4928	Inputs : None. *
		=4929	Outputs : None. *
		=4930	Destroys : All. *
		=4931	Calls : SPACES, PCODEL and PEQUL. *
		=4932	;

LOC	OBJ	LINE	SOURCE STATEMENT
		=4933 ;*	Description: This routine is called after each line in pass *
		=4934 ;*	#2, so that it can print the listing line if required. A *
		=4935 ;*	listing line is printed in the following cases. *
		=4936 ;*	
		=4937 ;*	a) The print_flag (PFLAG) is set. *
		=4938 ;*	b) It was a control line. *
		=4939 ;*	c) There were errors on the line. *
		=4940 ;*	
		=4941 ;*	NOTE: There are 2 types of listing lines. These are: *
		=4942 ;*	
		=4943 ;*	Code lines - e.g. MVI A,12 and *
		=4944 ;*	Equate lines - e.g. FRED EQU 1234H *
		=4945 ;*	
		=4946 ;*	These 2 types have different listing formats. *
		=4947 ;*	
		=4948 ;*	=====
		=4949 ;*	
		=4950	PLINE: ; Start of Print line routine.
2098	3A6EE9	=4951	LDA CONFLG ;
209B	FEFF	=4952	CPI TRUE ;
209D	CAAF20	=4953	JZ PRINTL ; If Control_flag = True then print_line.
		=4954	;
20A0	3A14E8	=4955	LDA LERRS ;
20A3	B7	=4956	ORA A ;
20A4	C2AF20	=4957	JNZ PRINTL ; If Errors_on_line > 0 then Print_line.
		=4958	;
		=4959	NLERRS: ;
20A7	3A69E9	=4960	LDA PFLAG ;
20AA	FE00	=4961	CPI FALSE ;
20AC	CAC520	=4962	JZ NPRINT ; If Print_flag = True then print_line.
		=4963	; (Else Do not print anything.)
		=4964	PRINTL: ;
20AF	3E03	=4965	MVI A,3 ;
20B1	CDA61F	=4966	CALL SPACES ; Print 3 spaces before the line starts
		=4967	;
20B4	3A0FE8	=4968	LDA EQUFLG ;
20B7	FEFF	=4969	CPI TRUE ;
20B9	CAC220	=4970	JZ EQULIN ; If Equate_flag <> true then
		=4971	;
20BC	CDEB1F	=4972	CALL PCODEL ; Print_code_line
20BF	C3C520	=4973	JMP NPRINT ;
		=4974	EQULIN: ; Else
		=4975	;
20C2	CD8620	=4976	CALL PEQUL ; Print_equate_line ;
		=4977	;
		=4978	NPRINT: ;
20C5	C9	=4979	RET ; Return from <PLINE>
		=4980 ;	
		=4981 ;*	=====
		=4982 ;*	
		=4983 ;*	Function : PRTCB (Print Code byte) *
		=4984 ;*	Inputs : A = Code byte. *
		=4985 ;*	Outputs : None. *
		=4986 ;*	Destroys : None. *
		=4987 ;*	Calls : PRINT and PRINTH. *
		=4988 ;*	
		=4989 ;*	Description: This routine prints a 2 character code for the *
		=4990 ;*	code byte. If the object is being to the PDP then the code *
		=4991 ;*	byte is printed as follows '--', otherwise it is printed *
		=4992 ;*	as 2 hex digits. (e.g. '2A') *
		=4993 ;*	
		=4994 ;*	=====
		=4995 ;*	
		=4996	PRTCB: ; Print code byte.
20C6	F5	=4997	PUSH PSW ;
20C7	3AC7E8	=4998	LDA OBJFLG ;
20CA	FE00	=4999	CPI FALSE ;
20CC	CADB20	=5000	JZ PRTCBH ; If Object_flag = True then
		=5001	;
20CF	3E2D	=5002	MVI A,'-' ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 87

LOC	OBJ	LINE	SOURCE STATEMENT
20D1	CDC61E	=5003	CALL PRINT ;
20D4	CDC61E	=5004	CALL PRINT ; Write ('--') (* Object sent to PDP *)
20D7	F1	=5005	POP PSW ;
20D8	C3DF20	=5006	JMP PRTCBF ;
		=5007	; Else
		=5008	PRTCBH: ;
20DB	F1	=5009	POP PSW ;
20DC	CDDC1E	=5010	CALL PRINTH ; Write Code_byte as 2 digit hex number.
		=5011	;
		=5012	PRTCBF: ;
20DF	C9	=5013	RET ; Return from <PRTCB>
		=5014	;
		=5015	;=====
		=5016	;*
		=5017	;* Function : PTEXT (Print source text line) *
		=5018	;* Inputs : None. *
		=5019	;* Outputs : None. *
		=5020	;* Destroys : H, L, D, E, A and flags. *
		=5021	;* Calls : PRINTD, DCMPL, NEXTPL and PRINT. *
		=5022	;* *
		=5023	;* Description : This routine prints the source text for the *
		=5024	;* current line. The line number is also printed. *
		=5025	;* *
		=5026	;=====
		=5027	;
		=5028	PTEXT: ; Start of <PTEXT> line
20E0	2A06E8	=5029	LHLD LINEN ;
20E3	CDB721	=5030	CALL PRINTD ; Print the line number in decimal.
		=5031	;
20E6	3E09	=5032	MVI A,TAB ;
20E8	CDC61E	=5033	CALL PRINT ; Print a TAB char.
		=5034	;
20EB	2A00E8	=5035	LHLD POINT ; Fetch the current assembly pointer
20EE	EB	=5036	XCHG ; DE = ptr to line feed at the end of the
		=5037	; current source line.
20EF	2A02E8	=5038	LHLD LPOINT ; Hl = Pointer to the start of the current
		=5039	; source line.
		=5040	PTEXTL: ; Repeat
20F2	7E	=5041	MOV A,M ; Acc := Text (Text_ptr);
20F3	FE0A	=5042	CPI LF ; Acc <> LF then
20F5	C4C61E	=5043	CNZ PRINT ; Write(chr(Acc));
20F8	23	=5044	INX H ; Text_ptr := Text_ptr + 1
20F9	CD2313	=5045	CALL DCMPL ;
20FC	C2F220	=5046	JNZ PTEXTL ; Until Text_ptr = Last_char + 1
		=5047	;
20FF	CD101E	=5048	CALL NEXTPL ; Next_print_line.
		=5049	;
2102	C9	=5050	RET ; Return from the <PTEXT> routine
		=5051	;
		=5052	; End of Print line module (PRINTLIN.A85)
		5053	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 88

LOC	OBJ	LINE	SOURCE STATEMENT
		5054	\$INCLUDE(ASMPERR.A85)
		=5055	-----
		=5056	;
		=5057	ASMPERR module for the UCT SABus assembler ;
		=5058	----- ;
		=5059	;
		=5060	This module contains (in order) the following subroutines: ;
		=5061	;
		=5062	a) PAERRS - Print any errors. ;
		=5063	b) PERR - Print error (This routine saves the error number) ;
		=5064	c) PERROR - Print error (This routine prints an error message) ;
		=5065	d) PNERRS - Print the number of errors. ;
		=5066	;
		=5067	-----
		=5068	;
		=5069	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 89

LOC	OBJ	LINE	SOURCE STATEMENT
		=5070 ;	=====
		=5071 ;*	*
		=5072 ;*	Function : PAERRS (Print Any Errors.) *
		=5073 ;*	Inputs : None *
		=5074 ;*	Outputs : None *
		=5075 ;*	Destroys : None *
		=5076 ;*	Calls : PHL, PERROR and STRING. *
		=5077 ;*	*
		=5078 ;*	Description: This routine prints the error messages at the *
		=5079 ;*	end of each line. If a fatal error is found then the fatal *
		=5080 ;*	error flag, 'FATALF', is set. The error message is printed *
		=5081 ;*	by calling 'PERROR' with the error number in the A reg. *
		=5082 ;*	*
		=5083 ;	=====
		=5084 ;	
		=5085 PAERRS:	; Start of <PAERRS> routine
2103	3A14E8	=5086 LDA LERRS	;
2106	FE00	=5087 CPI 00H	; Are there any errors in the line ?
2108	CA3D21	=5088 JZ NERRIL	; No - then Return from PAERRS.
		=5089	; Else output the error messages.
210B	47	=5090 MOV B,A	; B = Number of errors on the line.
210C	2116E8	=5091 LXI H,ERRBUF	; HL = Start of error number buffer.
		=5092 PNXTER:	;
210F	7E	=5093 MOV A,M	; Fetch error number from buffer.
2110	FE63	=5094 CPI FATALN	; Fatal Error number ?
2112	CA2021	=5095 JZ FATALE	; YES - Print error message and exit.
		=5096	;
2115	CD6021	=5097 CALL PERROR	; Print error meesage.
2118	23	=5098 INX H	; Inc pointer to next error number.
2119	05	=5099 DCR B	; Dec error_messages_left couunter.
211A	C20F21	=5100 JNZ PNXTER	; Loop if there any more error messages
		=5101	; to print.
211D	C33D21	=5102 JMP NERRIL	; Exit print any errors.
		=5103 FATALE:	;
2120	3EFF	=5104 MVI A,TRUE	;
2122	3215E8	=5105 STA FATALF	; Set fatal flag.
		=5106	;
2125	21BB2E	=5107 LXI H,FATM1	;
2128	CDB51F	=5108 CALL STRING	; Print 1st part of fatal error message.
212B	210000	=5109 LXI H,MINCOD	;
212E	CDBD1E	=5110 CALL PHL	; Print Min code address
2131	21E62E	=5111 LXI H,FATM2	;
2134	CDB51F	=5112 CALL STRING	; Print 2nd part of fatal error message.
2137	21FF0F	=5113 LXI H,MAXCOD	;
213A	CDBD1E	=5114 CALL PHL	; Print Max code address
		=5115	;
		=5116 NERRIL:	; No errors in line.
213D	C9	=5117 RET	; Return from <PAERRS>
		=5118 ;	
		=5119 ;	=====
		=5120 ;*	*
		=5121 ;*	Function : PERR (Print Error.) *
		=5122 ;*	Inputs : A register = Error number. *
		=5123 ;*	Outputs : None *
		=5124 ;*	Destroys : A register and flags. *
		=5125 ;*	Calls : None. *
		=5126 ;*	*
		=5127 ;*	Description: This routine is called when ever an error is *
		=5128 ;*	detected during assembly. It stores the error numbers in *
		=5129 ;*	an array (length = 10.) so that at the end of the *
		=5130 ;*	current line these messages can be printed. This routine *
		=5131 ;*	has bound checking. *
		=5132 ;*	*
		=5133 ;	=====
		=5134 ;	
		=5135 PERR:	; Start of <PERR> routine.
213E	C5	=5136 PUSH B	;
213F	47	=5137 MOV B,A	; Save error number in B register.
		=5138	;
2140	FE00	=5139 CPI 00	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 90

LOC	OBJ	LINE	SOURCE STATEMENT
2142	CA5E21	=5140	JZ PRTEE ; If Error number = zero then EXIT.
		=5141	;
2145	3A14E8	=5142	LDA LERRS ;
2148	FE0A	=5143	CPI 10 ; If Errors_on_Line = 10 then EXIT.
214A	CA5E21	=5144	JZ PRTEE ;
		=5145	;
214D	E5	=5146	PUSH H ; Save HL reg pair.
		=5147	;
214E	2A10E8	=5148	LHLD ERRORP ; Get pointer to error free location in
		=5149	; error number array.
2151	70	=5150	MOV M,B ; Save error number in array.
2152	23	=5151	INX H ;
2153	2210E8	=5152	SHLD ERRORP ; Inc and save new error array pointer.
		=5153	;
2156	3A14E8	=5154	LDA LERRS ;
2159	3C	=5155	INR A ; Inc Errors_on_line counter .
215A	3214E8	=5156	STA LERRS ;
		=5157	;
215D	E1	=5158	POP H ; Restore HL reg pair.
		=5159	PRTEE: ; Print error Exit.
215E	C1	=5160	POP B ;
215F	C9	=5161	RET ; Return from <PERR>.
		=5162 ;	
		=5163 ;=====	
		=5164 ;*	
		=5165 ;* Function : PERROR (Print Error)	*
		=5166 ;* Inputs : A reg = Error number.	*
		=5167 ;* Outputs : None	*
		=5168 ;* Destroys : None	*
		=5169 ;* Calls : STRING and NEXTPL.	*
		=5170 ;*	*
		=5171 ;* Description: This routine prints error message number 'N',	*
		=5172 ;* where 'N' is the error number passed in the Acc. The	*
		=5173 ;* routine simply counts <EOT>'s until it finds the 'N'th	*
		=5174 ;* error message and the prints it.	*
		=5175 ;*	*
		=5176 ;* Note: The end of each string (i.e. error message) is marker	*
		=5177 ;* by having the sign bit (D7) of the last character set.*	*
		=5178 ;*	*
		=5179 ;=====	
		=5180 ;	
		=5181 PERROR:	; Print Error message routine.
2160	F5	=5182	PUSH PSW ;
2161	E5	=5183	PUSH H ;
2162	C5	=5184	PUSH B ; Save registers that will be destroyed.
2163	4F	=5185	MOV C,A ; Save error number.
2164	21282F	=5186	LXI H,ERMES ; Get start adr of General Error message.
2167	CDB51F	=5187	CALL STRING ; Print It
216A	211D2A	=5188	LXI H,MESSAG ; Get start adr of error messages.
		=5189	NEXTST: ;
216D	0D	=5190	DCR C ;
216E	CA7A21	=5191	JZ PRTIT ;
		=5192	PERRNC: ; Repeat
2171	7E	=5193	MOV A,M ; acc := messages[ptr];
2172	23	=5194	INX H ; ptr := ptr + 1;
2173	B7	=5195	ORA A ;
2174	F27121	=5196	JP PERRNC ;
		=5197	;
2177	C36D21	=5198	JMP NEXTST ; until acc < 0 (* Sign bit set *)
		=5199	;
		=5200	PRTIT: ;
217A	CDB51F	=5201	CALL STRING ; Print the error message now that the correct
		=5202	; number have been skipped.
217D	2A12E8	=5203	LHLD ERRORS ;
2180	23	=5204	INX H ; Inc error counter.
2181	2212E8	=5205	SHLD ERRORS ;
		=5206	;
2184	CD101E	=5207	CALL NEXTPL ; Start another print line.
		=5208	NOPRT: ;
2187	C1	=5209	POP B ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 91

LOC	OBJ	LINE	SOURCE STATEMENT
2188	E1	=5210	POP H ; Restore registers.
2189	F1	=5211	POP PSW ;
218A	C9	=5212	RET ; Return from <PERROR>
		=5213 ;	
		=5214 ;=====	
		=5215 ;*	
		=5216 ;*	Function : PNERRS (Print Number ERRorS) *
		=5217 ;*	Inputs : None *
		=5218 ;*	Outputs : None *
		=5219 ;*	Destroys : H,L,Acc and Flags *
		=5220 ;*	Calls : STRING and PRINTD. *
		=5221 ;*	
		=5222 ;*	Description: This routine prints the error report for the *
		=5223 ;*	assembly. There are three possible types of error reports. *
		=5224 ;*	
		=5225 ;*	1) If Fatal_flag = True then "Fatal error" *
		=5226 ;*	
		=5227 ;*	2) If Fatal_flag = False and number_errors = 0 then *
		=5228 ;*	"no errors found" *
		=5229 ;*	
		=5230 ;*	3) If Fatal_flag = False and number_errors > 0 then *
		=5231 ;*	"NNNN Error(s) found" *
		=5232 ;*	
		=5233 ;=====	
		=5234 ;	
		=5235 PNERRS:	; Print end of assembly message with
		=5236	; error count (if there were any)
218B	3A15E8	=5237	LDA FATALF ;
218E	FEFF	=5238	CPI TRUE ;
2190	C29C21	=5239	JNZ NOTFEX ; If Fatal_Flag = TRUE Errors = Fatal exit.
		=5240	;
2193	21EA2E	=5241	LXI H,FATMSG ; Print "Fatal EXIT"
2196	CDB51F	=5242	CALL STRING ;
2199	C3B621	=5243	JMP ENDEM ;
		=5244	;
		=5245 NOTFEX:	; Else
219C	2A12E8	=5246	LHLD ERRORS ;
219F	7D	=5247	MOV A,L ;
21A0	B4	=5248	ORA H ;
21A1	CAB021	=5249	JZ NOERRS ; If NOERRS # zero then
		=5250	;
21A4	CDB721	=5251	CALL PRINTD ; Print the Number of Errors.
21A7	21482F	=5252	LXI H,ENDM2 ;
21AA	CDB51F	=5253	CALL STRING ; Print "Error(s) found"
21AD	C3B621	=5254	JMP ENDEM ;
		=5255 NOERRS:	; Else
21B0	21582F	=5256	LXI H,ENDM3 ;
21B3	CDB51F	=5257	CALL STRING ; Print "No errors found"
		=5258 ENDEM:	; Endif
21B6	C9	=5259	RET ; Endif
		=5260 ;	
		=5261 ;	End of Print_error module. (PERR.A85)
		5262	SEJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		5263	\$INCLUDE(ASMPRINTD.A85)
		=5264	-----
		=5265	;
		=5266	Decimal output routine for the UCT SABus assembler
		=5267	-----
		=5268	;
		=5269	-----
		=5270	;
		=5271	=====
		=5272	;
		=5273	Function : PRINTD (Print Decimal)
		=5274	Inputs : HL = 16 bit value to be printed.
		=5275	Outputs : None
		=5276	Destroys : All
		=5277	Calls : SPACES, PRINT
		=5278	;
		=5279	Description: This routine prints the 16 bit value in the HL
		=5280	reg pair as a 4 digit negative number. Leading zeros are
		=5281	supressed.
		=5282	;
		=5283	=====
		=5284	;
		=5285	PRINTD: ; Start of the <PRINTD> routine
21B7	7C	=5286	MOV A, H ; Check if HL zero first
21B8	B5	=5287	ORA L ;
21B9	C2C721	=5288	JNZ OUTD05 ; No - print it normally
		=5289	;
21BC	3E03	=5290	MVI A,3 ;
21BE	CDA61F	=5291	CALL SPACES ;
21C1	3E30	=5292	MVI A,'0' ; Print " 0" , right justified 0 .
21C3	CDC61E	=5293	CALL PRINT ;
21C6	C9	=5294	RET ; and return
		=5295	OUTD05: ;
21C7	3E00	=5296	MVI A, FALSE ; suppress leading zeroes
21C9	326DE9	=5297	STA PZRFLG ;
21CC	110822	=5298	LXI D, PWR10 ; point to table of powers of 10
		=5299	OUTD10: ;
21CF	EB	=5300	XCHG ; table pointer into HL
21D0	4E	=5301	MOV C, M ; put power of 10 into BC
21D1	23	=5302	INX H ;
21D2	46	=5303	MOV B, M ;
21D3	23	=5304	INX H ;
21D4	EB	=5305	XCHG ; put number back to HL
21D5	78	=5306	MOV A, B ; check to see if we're at the end
21D6	B1	=5307	ORA C ; last table entry = 0
21D7	C8	=5308	RZ ;
		=5309	;
21D8	3E00	=5310	MVI A, 0 ; initialise digit to zero
21DA	E5	=5311	PUSH H ; dummy push
		=5312	OUTD15: ;
21DB	33	=5313	INX SP ; correct stack for push
21DC	33	=5314	INX SP ;
21DD	E5	=5315	PUSH H ; save HL in case we go too far
21DE	3C	=5316	INR A ; bump up digit
21DF	09	=5317	DAD B ; subtract the power of 10
21E0	DADB21	=5318	JC OUTD15 ; can still take off more... go back
		=5319	;
21E3	E1	=5320	POP H ; else restore HL again
21E4	C62F	=5321	ADI '0'-1 ; convert to ASCII and decrement
21E6	4F	=5322	MOV C, A ; put in C ready to output
21E7	FE30	=5323	CPI '0' ; Is it zero?
21E9	C2F421	=5324	JNZ OUTD20 ; no - print it
		=5325	;
21EC	3A6DE9	=5326	LDA PZRFLG ; else check if we're suppressing
21EF	FE00	=5327	CPI FALSE ; leading zeroes
21F1	CA0022	=5328	JZ PADWB ; Yes - don't print the zero
		=5329	OUTD20: ;
21F4	79	=5330	MOV A,C ;
21F5	CDC61E	=5331	CALL PRINT ; print the number
21F8	3EFF	=5332	MVI A, TRUE ; must print zeroes from now on

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 93

LOC	OBJ	LINE	SOURCE STATEMENT
21FA	326DE9	=5333	STA PZRFLG ;
21FD	C3CF21	=5334	JMP OUTD10 ; loop back for next power
		=5335	;
2200	3E20	=5336	PADWB: MVI A, ' ' ; Print a space to pad the number to 4 chars.
2202	CDC61E	=5337	CALL PRINT ;
2205	C3CF21	=5338	JMP OUTD10 ;
		=5339	;
		=5340	PWR10: ;
2208	18FC	=5341	DW -1000 ; Negative powers of ten .
220A	9CFF	=5342	DW -100 ;
220C	F6FF	=5343	DW -10 ;
220E	FFFF	=5344	DW -1 ;
2210	0000	=5345	DW 0 ; 0 marks the last entry
		=5346 ;	
		=5347 ;	End of the <ASMPRINTD.A85> file.
		5348	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 94

LOC	OBJ	LINE	SOURCE STATEMENT
		5349 ;	
		5350 ;	
		5351 ;	-----
		5352 ;	
		5353 ;	CODE GENERATION ROUTINES.
		5354 ;	
		5355	\$INCLUDE(ASMGETOPR.A85)
		=5356 ;	-----
		=5357 ;	
		=5358 ;	Get operand module for the UCT SABus assembler ;
		=5359 ;	----- ;
		=5360 ;	
		=5361 ;	This module contains (in order) the following subroutines: ;
		=5362 ;	
		=5363 ;	a) GETOPR - Get operand routine. (calls one of the following ;
		=5364 ;	subroutines: ;
		=5365 ;	
		=5366 ;	i) D16ROU - 16 bit data routine. ;
		=5367 ;	ii) D8ROU - 8 bit data routine. ;
		=5368 ;	iii) SREGRO - Source register routine. ;
		=5369 ;	iv) DREGRO - Destination register routine. ;
		=5370 ;	v) RPSROU - Register pair (Stack) routine. ;
		=5371 ;	vi) RPAROU - Register pair (Arithmetic) routine. ;
		=5372 ;	vii) RPLROU - Register pair (Limited) routine. ;
		=5373 ;	viii) RSTNRO - RST number routine. ;
		=5374 ;	ix) UD8ROU - Unsigned 8 bit data routine. ;
		=5375 ;	
		=5376 ;	b) QREG - Register ? ;
		=5377 ;	
		=5378 ;	-----
		=5379 ;	
		=5380	\$EJECT

G-103

LOC	OBJ	LINE	SOURCE STATEMENT
		=5451 ;	
		=5452 ;=====	
		=5453 ;*	
		=5454 ;* Function : (1) D16ROU (Operand = 16 bit word)	*
		=5455 ;* Inputs : None.	*
		=5456 ;* Outputs : None.	*
		=5457 ;* Destroys : A, H, L and flags.	*
		=5458 ;* Calls : REXP	*
		=5459 ;*	*
		=5460 ;* Description: This routine simply reads (or tries to read),	*
		=5461 ;* an expression from then source. This routine never returns	*
		=5462 ;* with an error because any error message needed will be	*
		=5463 ;* printed by the REXP routine.	*
		=5464 ;*	*
		=5465 ;=====	
		=5466 ;	
		=5467 D16ROU:	; Data 16 (e.g. LXI H,'D16')
		=5468	;
223E	CD1C24	=5469 CALL REXP	; Read the expression.
2241	22E0E8	=5470 SHLD CODBUF+1	; Save the word in the code buffer.
2244	3E00	=5471 MVI A,0	; Set acc=zero i.e. No error (Rexp will print
		=5472	; it's own error message.)
2246	C9	=5473 RET	; Return from <D16ROU>
		=5474 ;	
		=5475 ;=====	
		=5476 ;*	
		=5477 ;* Function : (2) D8ROU (Operand = 8 bit byte)	*
		=5478 ;* Inputs : None.	*
		=5479 ;* Outputs : None.	*
		=5480 ;* Destroys : All.	*
		=5481 ;* Calls : REXP	*
		=5482 ;*	*
		=5483 ;* Description: As above this routine reads an expression as an	*
		=5484 ;* operand for an opcode and stores the result in the	*
		=5485 ;* Code_buffer. The routine also checks that the word read	*
		=5486 ;* can be represented as an 8 bit value.	*
		=5487 ;*	*
		=5488 ;* The 2 valid words are: a) 00XXH (MSByte = 00H)	*
		=5489 ;* b) FFXXH (MSByte = FFH)	*
		=5490 ;*	*
		=5491 ;* NOTE: If format B, the lower byte must be negative to allow	*
		=5492 ;* it to be sign extended to 16 bits and therefore be	*
		=5493 ;* representable in 8 bits.	*
		=5494 ;*	*
		=5495 ;=====	
		=5496 ;	
		=5497 D8ROU:	; (2) Signed 8 bit value
2247	CD1C24	=5498 CALL REXP	; Read the expression.
		=5499	;
224A	7C	=5500 MOV A,H	;
224B	FE00	=5501 CPI 0	;
224D	CA6122	=5502 JZ D810	; If MSByte of result = 0 then number < 256
		=5503	; therefore 8 bit data.
2250	3C	=5504 INR A	;
2251	C25A22	=5505 JNZ D805	; If MSByte of result <> 0FFH then the value
		=5506	; can not be expressed as an 8 bit value.
		=5507 ;	
		=5508 ;-----	
		=5509 ;	
		=5510 ;	Now check that the lower byte if negative. If it is then sign
		=5511 ;	extending it to 16 bits would yield an 0FFH in the MSByte.
		=5512 ;	
2254	7D	=5513 MOV A,L	; Fetch LSByte of result
2255	E680	=5514 ANI MSB	; Mask off MSBit
2257	C26122	=5515 JNZ D810	; Set - then value negative - value OK.
		=5516	;
		=5517 D805:	;
225A	3E0A	=5518 MVI A,10	;
225C	CD3E21	=5519 CALL PERR	; Else data error. (> 8 bits).
225F	2E00	=5520 MVI L,00H	; Set byte to be generated to zero

LOC	OBJ	LINE	SOURCE STATEMENT
		=5521	;
		=5522	D810: ;
2261	7D	=5523	MOV A,L ; Fetch LSByte of result
2262	32E0E8	=5524	STA CODBUF+1 ; Store as 2nd byte of code.
		=5525	;
2265	3E00	=5526	MVI A,0 ; Setup OK return.
2267	C9	=5527	RET ; Return from <D8ROU>
		=5528	;
		=5529	=====
		=5530	;* *
		=5531	;* Function : (3) SREGRO (Operand = source register) *
		=5532	;* Inputs : None. *
		=5533	;* Outputs : None. *
		=5534	;* Destroys : All. *
		=5535	;* Calls : QREG *
		=5536	;* *
		=5537	;* Description: This routine reads the next character in the *
		=5538	;* source and checks that it's in the set of valid registers. *
		=5539	;* If it is then the opcode is updated with the register mask. *
		=5540	;* *
		=5541	;* Source_register bit position: sss *
		=5542	;* *
		=5543	;* Where "sss" is the register code for one of the 8 regs. *
		=5544	;* *
		=5545	;=====
		=5546	;
		=5547	SREGRO: ; Start of <SREGRO>
2268	2A00E8	=5548	LHLD POINT ;
226B	7E	=5549	MOV A,M ; Acc = next source character.
226C	CD5E23	=5550	CALL QREG ; Acc = Reg ? (i.e. A,B,C,D,E,H,L,M).
226F	FA8122	=5551	JM EROPRE ; No - Then error.
		=5552	;
2272	47	=5553	MOV B,A ; Else update Opcode.
2273	3A5FE9	=5554	LDA OPCODE ;
2276	B0	=5555	ORA B ; Opcode = opcode OR Reg_mask
2277	325FE9	=5556	STA OPCODE ;
227A	23	=5557	INX H ; Inc past reg.
227B	2200E8	=5558	SHLD POINT ; Save new assembly pointer.
227E	3E00	=5559	MVI A,0 ; Setup OK return
2280	C9	=5560	RET ; Return from <SREGRO>
		=5561	;
		=5562	;-----
		=5563	;
		=5564	EROPRE: ; Error in operand register expected.
2281	3E15	=5565	MVI A,21 ; Error #21
2283	C32422	=5566	JMP EEGETO ; Exit getopr routine (with an error)
		=5567	;
		=5568	=====
		=5569	;* *
		=5570	;* Function : (4) DREGRO (Operand = Destination_reg.) *
		=5571	;* Inputs : None. *
		=5572	;* Outputs : None. *
		=5573	;* Destroys : All. *
		=5574	;* Calls : QREG. *
		=5575	;* *
		=5576	;* Description: This routine is basically the same as the *
		=5577	;* routine above <SREGRO>. The only change is the position of *
		=5578	;* the register code in opcode. *
		=5579	;* *
		=5580	;* Destination_register bit position: ..dd d... *
		=5581	;* *
		=5582	;* Where "ddd" is the register code for one of the 8 regs. *
		=5583	;* *
		=5584	;=====
		=5585	;
		=5586	DREGRO: ; Start of <DREGRO>
2286	2A00E8	=5587	LHLD POINT ;
2289	7E	=5588	MOV A,M ; Acc = next source char.
228A	CD5E23	=5589	CALL QREG ; Acc in Valid_registers ?
228D	FA8122	=5590	JM EROPRE ; No - then register expected error.

LOC	OBJ	LINE	SOURCE STATEMENT
		=5591	
2290	07	=5592	RLC
2291	07	=5593	RLC
2292	07	=5594	RLC
		=5595	; Acc = 8 * Acc (i.e. Have moved register
			; number from ".....rrr" to "...rrr..."
2293	47	=5596	MOV B,A
2294	3A5FE9	=5597	LDA OPCODE
2297	B0	=5598	ORA B
			; Opcode = Opcode OR Des_reg_code.
2298	325FE9	=5599	STA OPCODE
229B	23	=5600	INX H
			; Inc text pointer to after reg.
229C	2200E8	=5601	SHLD POINT
			; Save new ptr.
229F	3E00	=5602	MVI A,0
			; Setup OK return.
		=5603	
22A1	C9	=5604	RET
			; Return from <DREGRO>
		=5605	
		=5606	;=====
		=5607	;
		=5608	;* Function : (5) RPSROU (Operand = Reg_pair_stack). *
		=5609	;* Inputs : None. *
		=5610	;* Outputs : None. *
		=5611	;* Destroys : All. *
		=5612	;* Calls : COPY and QTAB. *
		=5613	;* *
		=5614	;* Description: This routine tries to read a register pair *
		=5615	;* operand from the source line and mask it into the opcode. *
		=5616	;* *
		=5617	;* Register_pair_stack bit position: ..RR *
		=5618	;* *
		=5619	;* Where "RR" is one of the 4 register_pair codes. *
		=5620	;* *
		=5621	;=====
		=5622	;
		=5623	RPSROU: ; Start of <RPSROU>
22A2	2153E9	=5624	LXI H,OPBUF ;
22A5	CD8313	=5625	CALL COPY ; Copy the next string into the string buffer
		=5626	; It should be a Stack_register pair.
		=5627	;
22A8	11C522	=5628	LXI D,STKRPS ; Init Start_of_table_ptr = Stack_reg_pairs.
22AB	CD1514	=5629	CALL QTAB ; Is the string in the OPBUF in the table
		=5630	; those starting adr is STKRPS ?
22AE	FAC022	=5631	JM EROPSE ; No - Then error (Stack_reg_pair expected)
		=5632	;
22B1	07	=5633	RLC ;
22B2	07	=5634	RLC ;
22B3	07	=5635	RLC ;
22B4	07	=5636	RLC ; Acc = 16 * Acc. (i.e. Have moved reg_pair
		=5637	; value from ".....RR" to "...RR...."
22B5	47	=5638	MOV B,A ;
22B6	3A5FE9	=5639	LDA OPCODE ;
22B9	B0	=5640	ORA B ; Opcode = Opcode OR Reg_pair_code
22BA	325FE9	=5641	STA OPCODE ;
22BD	3E00	=5642	MVI A,0 ; Setup OK return.
22BF	C9	=5643	RET ; Return from <SRPROU>
		=5644	;
		=5645	;
		=5646	;
		=5647	;
		=5648	EROPSE: ;
22C0	3E1A	=5648	MVI A,26 ; Error #26 (Stack_reg_pair expected).
22C2	C32422	=5649	JMP EEGETO ; Exit Getopr with error.
		=5650	;
		=5651	;
		=5652	;
		=5653	; Table of valid Stack register pairs. (i.e. Reg pairs that can be
		=5654	pushed onto the stack)
		=5655	;
		=5656	;
		=5657	;
		=5658	STKRPS: ; Start of Table.
		=5659	;
22C5	4204	=5660	DB 'B' ,EOT ; BE reg pair = 00b

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 99

LOC	OBJ	LINE	SOURCE STATEMENT
22C7	4404	=5661	DB 'D',EOT ; DE reg pair = 01b
22C9	4804	=5662	DB 'H',EOT ; HL reg pair = 10b
22CB	50535704	=5663	DB 'PSW',EOT ; Acc and Flags = 11b
		=5664	;
22CF	FF	=5665	DB ETABLE ; End of table.
		=5666 ;	
		=5667 ;=====	
		=5668 ;*	
		=5669 ;* Function : (6) RPAROU (Operand = reg_pair_arith.)	*
		=5670 ;* Inputs : None.	*
		=5671 ;* Outputs : None.	*
		=5672 ;* Destroys : All.	*
		=5673 ;* Calls : QTAB and COPY.	*
		=5674 ;*	*
		=5675 ;* Description: This routine tries to read one of the 4	*
		=5676 ;* arithmetic register pairs from the source line.	*
		=5677 ;*	*
		=5678 ;* Register_pair_arithmetic bit position: ..RR	*
		=5679 ;*	*
		=5680 ;* Where "RR" is one of the 4 arithmetic register_pair codes.	*
		=5681 ;*	*
		=5682 ;=====	
		=5683 ;	
		=5684 RPAROU:	; Start of <RPAROU>
22D0	2153E9	=5685	LXI H,OPBUF ;
22D3	CD8313	=5686	CALL COPY ; Copy string into opcode_buffer
22D6	11F322	=5687	LXI D,ARPS ;
22D9	CD1514	=5688	CALL QTAB ; Opcode_buffer IN Arithmetic_register_pairs ?
22DC	FAEE22	=5689	JM EROPAE ; No - Then Error in Operands.
		=5690	;
22DF	07	=5691	RLC ;
22E0	07	=5692	RLC ;
22E1	07	=5693	RLC ;
22E2	07	=5694	RLC ; Shift ".... ..RR" to "..RR"
22E3	47	=5695	MOV B,A ;
22E4	3A5FE9	=5696	LDA OPCODE ;
22E7	B0	=5697	ORA B ;
22E8	325FE9	=5698	STA OPCODE ; and mask it into the Opcode.
		=5699	;
22EB	3E00	=5700	MVI A,0 ; Setup zero return = OK.
22ED	C9	=5701	RET ;
		=5702	;
		=5703 EROPAE:	;
22EE	3E16	=5704	MVI A,22 ; Err #22 = "Error in operands
		=5705	; Arith register pair expected"
22F0	C32422	=5706	JMP EEGETO ;
		=5707 ;	
		=5708 ;-----	
		=5709 ;	
		=5710 ; Table of valid arithmetic register pairs (i.e. Reg pairs that can	
		=5711 ; be Dad'ed with HL)	
		=5712 ;	
		=5713 ;-----	
		=5714 ;	
		=5715 ARPS:	; Start of table.
		=5716	;
22F3	4204	=5717	DB 'B',EOT ; BE register pair = 00b
22F5	4404	=5718	DB 'D',EOT ; DE register pair = 01b
22F7	4804	=5719	DB 'H',EOT ; HE register pair = 10b
22F9	535004	=5720	DB 'SP',EOT ; Stack pointer = 11b
		=5721	;
22FC	FF	=5722	DB ETABLE ; End of table.
		=5723 ;	
		=5724 ;=====	
		=5725 ;*	*
		=5726 ;* Function : (7) RPLROU (Operand = reg_pair_limited).	*
		=5727 ;* Inputs : Current_asm_ptr (POINT)	*
		=5728 ;* Outputs : None.	*
		=5729 ;* Destroys : A, H, L and Flags	*
		=5730 ;* Calls : CONLET.	*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM PAGE 100

LOC	OBJ	LINE	SOURCE STATEMENT
		=5731 ;*	
		=5732 ;*	Description: This routine is used to read a limited
		=5733 ;*	register pair from the source line.
		=5734 ;*	
		=5735 ;*	Note: The limited register pairs are (BC) and (DE).
		=5736 ;*	
		=5737 ;	=====
		=5738 ;	
		=5739 RPLROU:	; Start of RPLROU.
22FD	2A00E8	=5740	LHLD POINT ;
2300	7E	=5741	MOV A,M ; Acc = Next source char.
2301	CDB213	=5742	CALL CONLET ; Convert acc to upper case.
2304	FE42	=5743	CPI 'B' ;
2306	CA1323	=5744	JZ BCRP ; If Char = 'B' then BC register pair.
		=5745	; ;
2309	FE44	=5746	CPI 'D' ;
230B	CA1823	=5747	JZ DERP ; If Char = 'D' then DE register pair.
		=5748	; ;
230E	3E1B	=5749	MVI A,27 ; Else errpr #27 (Limited RP expected.)
2310	C32422	=5750	JMP EEGETO ;
		=5751	; ;
		=5752 BCRP:	; Set RP mask = BC register mask.
2313	3E00	=5753	MVI A,00000000B ;
2315	C31A23	=5754	JMP MASKIN ;
		=5755	; ;
		=5756 DERP:	; Set RP mask = DE register mask.
2318	3E10	=5757	MVI A,00010000B ;
		=5758	; ;
		=5759 MASKIN:	; ;
231A	47	=5760	MOV B,A ;
231B	3A5FE9	=5761	LDA OPCODE ;
231E	B0	=5762	ORA B ; Opcode = Opcode OR Limited_register_pair
231F	325FE9	=5763	STA OPCODE ;
2322	23	=5764	INX H ;
2323	2200E8	=5765	SHLD POINT ; Asm_ptr := asm_ptr + 1 (Skip char.)
2326	3E00	=5766	MVI A,0 ; Setup OK return.
2328	C9	=5767	RET ; Return from <RPLROU>
		=5768 ;	
		=5769 ;	=====
		=5770 ;*	
		=5771 ;*	Function : (8) RSTNRO (Operand = RST number (0-7))
		=5772 ;*	Inputs : None.
		=5773 ;*	Outputs : None.
		=5774 ;*	Destroys : All.
		=5775 ;*	Calls : REXP.
		=5776 ;*	
		=5777 ;*	Description: This routine read the RST number from the
		=5778 ;*	source line. The value must be in the range 0 to 7.
		=5779 ;*	
		=5780 ;*	RST value bit position: ..NN N...
		=5781 ;*	
		=5782 ;*	Where NNN is the RST value (Range 0 to 7)
		=5783 ;*	
		=5784 ;	=====
		=5785 ;	
		=5786 RSTNRO:	; Start of <RSTROU>
2329	CD1C24	=5787	CALL REXP ; Read expression from source.
		=5788	; ;
232C	7D	=5789	MOV A,L ;
232D	E6F8	=5790	ANI 11111000B ;
232F	B4	=5791	ORA H ; Value returned = 0000 0000 0000 ONNN b ?
		=5792	; (Where NNN = RST number)
2330	CA3823	=5793	JZ NUMLIM ; Yes - Then RST value in range.
		=5794	; ;
2333	3E17	=5795	MVI A,23 ; Else error #23 (RST number must be <= 7)
2335	C32422	=5796	JMP EEGETO ; Exit getopr with error.
		=5797	; ;
		=5798 NUMLIM:	; ;
2338	3A5FE9	=5799	LDA OPCODE ;
233B	47	=5800	MOV B,A ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 101

LOC	OBJ	LINE	SOURCE STATEMENT
233C	7D	=5801	MOV A,L ;
233D	07	=5802	RLC ;
233E	07	=5803	RLC ;
233F	07	=5804	RLC ; Acc = Acc * 8 (i.e. Have moved ".....NNN"
		=5805	; to "...NNN..."
2340	B0	=5806	ORA B ;
2341	325FE9	=5807	STA OPCODE ; Opcode = Opcode OR RST_field.
2344	3E00	=5808	MVI A,0 ; Setup OK return.
2346	C9	=5809	RET ; Return from <RSTNRO>
		=5810 ;	
		=5811 ;=====	
		=5812 ;*	
		=5813 ;* Function : (9) UD8ROU - Operand = Unsigned 8 bit value	*
		=5814 ;* Inputs : None.	*
		=5815 ;* Outputs : None.	*
		=5816 ;* Destroys : All.	*
		=5817 ;* Calls : REXP	*
		=5818 ;*	*
		=5819 ;* Description: As above this routine reads an expression as an	*
		=5820 ;* operand for an opcode and stores the result in the	*
		=5821 ;* Code_buffer. The routine also checks that the word read	*
		=5822 ;* can be represented as an unsigned 8 bit value.	*
		=5823 ;*	*
		=5824 ;* The valid format is: 00XXH (MSByte = 00H)	*
		=5825 ;*	*
		=5826 ;=====	
		=5827 ;	
		=5828 UD8ROU:	; (9) Unsigned 8 bit value
2347	CD1C24	=5829	CALL REXP ; Read the expression.
		=5830	;
234A	7C	=5831	MOV A,H ; Check upper byte
234B	FE00	=5832	CPI 0 ;
234D	CA5723	=5833	JZ UD805 ; If MSByte of result = 0 then number < 256
		=5834	; therefore 8 bit unsigned value.
2350	3E28	=5835	MVI A,40 ;
2352	CD3E21	=5836	CALL PERR ; Else data error. (> 8 bits).
2355	2E00	=5837	MVI L,00H ; Setup byte to generate = zero
		=5838	;
		=5839 UD805:	;
2357	7D	=5840	MOV A,L ; Fetch LSByte of result.
2358	32E0E8	=5841	STA CODBUF+1 ; Store as 2nd byte of code.
		=5842	;
235B	3E00	=5843	MVI A,0 ; Setup OK return.
235D	C9	=5844	RET ; Return from <UD8ROU>
		=5845 ;	
		=5846 ;=====	
		=5847 ;*	
		=5848 ;* Function : QREG (Register ?).	*
		=5849 ;* Inputs : Acc = Character to check.	*
		=5850 ;* Outputs : Acc = Register mask (3 bits).	*
		=5851 ;* If not valid register then Negative number	*
		=5852 ;* returned.	*
		=5853 ;* Destroys : A,B,C and Flags.	*
		=5854 ;* Calls : CONLET	*
		=5855 ;*	*
		=5856 ;* Description: This routine checks to see if the char passed	*
		=5857 ;* in the Acc is a valid register.	*
		=5858 ;*	*
		=5859 ;=====	
		=5860 ;	
		=5861 QREG:	; Start of <QREG>
235E	E5	=5862	PUSH H ; Save HL
235F	0E00	=5863	MVI C,00H ; Set counter = 0
2361	CDB213	=5864	CALL CONLET ; Convert letter in the acc to upper case
2364	47	=5865	MOV B,A ;
2365	218023	=5866	LXI H,REGS ;
		=5867 NXTREG:	;
2368	7E	=5868	MOV A,M ; Get character from Register table.
2369	FE04	=5869	CPI EOT ; End of register table ?
236B	CA7723	=5870	JZ NOTREG ; Yes- Then char <> register.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 102

LOC	OBJ	LINE	SOURCE STATEMENT
		=5871	
236E	B8	=5872	CMP B ; Register_table_char = Input_char ?
236F	CA7C23	=5873	JZ REGF ; Yes - Then character is a register.
		=5874	
2372	0C	=5875	INR C ; Inc counter
2373	23	=5876	INX H ; Inc register table pointer.
2374	C36823	=5877	JMP NXTREG ; Loop.
		=5878	
		=5879	NOTREG: ;
2377	3EFF	=5880	MVI A,OFFH ; Char not register = Error return.
2379	C37D23	=5881	JMP QREGEX ;
		=5882	REGF: ;
237C	79	=5883	MOV A,C ; Char is a register = Acc = register mask.
		=5884	QREGEX: ;
237D	B7	=5885	ORA A ; Set flags before returning.
237E	E1	=5886	POP H ;
237F	C9	=5887	RET ; Return from <QREG>
		=5888	
2380	42434445	=5889	REGS: DB 'BCDEHLMA',EOT
	484C4D41		
	04		
		=5890	;
		=5891	;
			End of Get operand routines. (GETOPR.A85)
		5892	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 103

LOC	OBJ	LINE	SOURCE STATEMENT
		5893	\$INCLUDE(ASMCODE.A85)
		=5894	=====
		=5895	;
		=5896	Function : CODE
		=5897	Inputs : A register = Code byte
		=5898	HL Register pair = Code generation pointer.
		=5899	Outputs : None
		=5900	Destroys : None
		=5901	Calls : DCMF, PERR and SENDHR.
		=5902	;
		=5903	Description: This routine does one of two things depending
		=5904	on the state of the OBJECT_FLAG.
		=5905	;
		=5906	Object_Flag = False. (i.e. Code to RAM)
		=5907	-----
		=5908	;
		=5909	This Section does the code generation by simply placing
		=5910	the code byte in memory at the adr given by the HL
		=5911	register pair.
		=5912	;
		=5913	The Section also does bound checking on the address to
		=5914	make sure that it does not over write the source or the
		=5915	assembler code.
		=5916	;
		=5917	Object_flag = True. (i.e. Code to Disk)
		=5918	-----
		=5919	;
		=5920	This Section does the code generation by writing the bytes
		=5921	to disk in the form of a Intel Paper tape format file.
		=5922	(For more details see Documentation.)
		=5923	;
		=5924	Note: No bound checking is done if the file is being
		=5925	written to disk.
		=5926	;
		=5927	PROCEDURE CODE (CODE_ADR[HL] : INTEGER
		=5928	, CODE_BYTE[ACC] : CHAR);
		=5929	BEGIN
		=5930	IF NOT OBJECT_FLAG THEN
		=5931	(* Object to Ram *)
		=5932	IF (CODE_ADR <= MAX_CODE_ADR) AND
		=5933	(CODE_ADR >= MIN_CODE_ADR) THEN
		=5934	MEMORY(CODE_ADR) := CODE_BYTE
		=5935	ELSE
		=5936	PRINT_ERROR (FATAL_ERROR)
		=5937	END;
		=5938	ELSE
		=5939	(* Object to PDP as Hex format file *)
		=5940	IF OBJ_NEXT_ADR = CODE_ADR THEN
		=5941	BEGIN
		=5942	OBJBUF(OBJ_COUNT) := CODE_BYTE;
		=5943	OBJ_COUNT := OBJ_COUNT + 1 ;
		=5944	OBJ_NEXT_ADR := OBJ_NEXT_ADR + 1 ;
		=5945	IF OBJ_COUNT = OBJ_LEN THEN
		=5946	SEND_HEX_BUFFER
		=5947	END
		=5948	ELSE
		=5949	BEGIN
		=5950	SEND_HEX_BUFFER;
		=5951	OBJBUF(OBJ_COUNT) := CODE_BYTE
		=5952	BASE_ADR := CODE_ADR ;
		=5953	OBJ_NEXT_ADR := CODE_ADR + 1
		=5954	OBJ_COUNT := OBJ_COUNT + 1
		=5955	END
		=5956	END;
		=5957	;
		=5958	=====
		=5959	;
2389	C5	=5960	CODE: ; Generate code byte.
238A	D5	=5961	PUSH B ;
		=5962	PUSH D ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 104

LOC	OBJ	LINE	SOURCE STATEMENT
238B	E5	=5963	PUSH H ;
238C	F5	=5964	PUSH PSW ;
		=5965	;
238D	3AC7E8	=5966	LDA OBJFLG ; Object to DISK ?
2390	FEFF	=5967	CPI TRUE ;
2392	CABC23	=5968	JZ TOFILE ;
		=5969	;
		=5970	;
		=5971	;
2395	110010	=5971	LXI D,MAXCOD+1 ;
2398	CD2313	=5972	CALL DCMP ;
239B	D2AC23	=5973	JNC ERRCOD ; If PC > MAXCODE then error
		=5974	;
239E	110000	=5975	LXI D,MINCOD ;
23A1	CD2313	=5976	CALL DCMP ;
23A4	DAAC23	=5977	JC ERRCOD ; If PC < MINCODE then error.
		=5978	;
23A7	F1	=5979	POP PSW ;
23A8	77	=5980	MOV M,A ; Else put byte in memory.
23A9	C3B723	=5981	JMP RAMEND ;
		=5982	;
		=5983	ERRCOD: ;
23AC	3E63	=5984	MVI A,FATALN ;
23AE	CD3E21	=5985	CALL PERR ; Generate a FATAL error message.
23B1	3E00	=5986	MVI A,FALSE ;
23B3	320EE8	=5987	STA GCFLAG ;
23B6	F1	=5988	POP PSW ;
		=5989	RAMEND: ;
23B7	E1	=5990	POP H ;
23B8	D1	=5991	POP D ;
23B9	C30524	=5992	JMP CODEEX ;
		=5993	;
		=5994	;
		=5995	;
		=5996	;
		=5997	TOFILE: ; Object to file.
23BC	EB	=5998	XCHG ; Move Code_adr to DE.
23BD	2ADCE8	=5999	LHLD OBJNA ; Get Obj next adr.
23C0	CD2313	=6000	CALL DCMP ; Code_adr = Obj_next_adr ?
23C3	CAE423	=6001	JZ OBJCON ; Yes - then object cont.
		=6002	;
23C6	CD401F	=6003	CALL SENDHR ; Else send current Record.
23C9	3E01	=6004	MVI A,1 ;
23CB	32DEE8	=6005	STA OBJC ; Obj_count = 1
23CE	F1	=6006	POP PSW ;
23CF	21C8E8	=6007	LXI H,OBJBUF ;
23D2	77	=6008	MOV M,A ; Save Byte at start of object buffer.
23D3	23	=6009	INX H ;
23D4	22D8E8	=6010	SHLD OBJBP ; Save Ptr to next free byte as OBJ_PTR
23D7	E1	=6011	POP H ; Restore Code_adr
23D8	22DAE8	=6012	SHLD OBJBA ; Save as Base_adr.
23DB	23	=6013	INX H ;
23DC	22DCE8	=6014	SHLD OBJNA ; Save Code_adr + 1 as Obj_Next_adr
23DF	2B	=6015	DCX H ; Restore HL to value from the stack.
23E0	D1	=6016	POP D ; Remove DE from stack
23E1	C30524	=6017	JMP CODEEX ; Exit Code Routine.
		=6018	;
		=6019	;
		=6020	;
		=6021	;
		=6022	;
		=6023	OBJCON: ; Object cont section.
23E4	F1	=6024	POP PSW ; Restore PSW, acc contains the data byte.
23E5	2AD8E8	=6025	LHLD OBJBP ;
23E8	77	=6026	MOV M,A ; Save byte in Object buffer.
23E9	23	=6027	INX H ; Inc obj_buf_ptr
23EA	22D8E8	=6028	SHLD OBJBP ;
		=6029	;
23ED	2ADCE8	=6030	LHLD OBJNA ;
23F0	23	=6031	INX H ; Inc Obj_next_adr.
23F1	22DCE8	=6032	SHLD OBJNA ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 105

LOC	OBJ	LINE	SOURCE STATEMENT
		=6033	
23F4	3ADEE8	=6034	LDA OBJC ;
23F7	3C	=6035	INR A ; Inc Object_byte_count.
23F8	32DEE8	=6036	STA OBJC ;
		=6037	
23FB	FE10	=6038	CPI OBJLEN ; Object buffer full ?
23FD	C20324	=6039	JNZ OBJBNF ; No - then Exit.
		=6040	
2400	CD401F	=6041	CALL SENDHR ; Yes - the send the record.
		=6042	
		=6043	OBJBNF: ; Object Buffer not full
2403	E1	=6044	POP H ; Pop HL and DE.
2404	D1	=6045	POP D ;
		=6046	CODEEX: ;
2405	C1	=6047	POP B ;
2406	C9	=6048	RET ; Return from <CODE>.
		=6049	;
		=6050	=====
		=6051	;* *
		=6052	;* Function : OCODBUF (Output code buffer) *
		=6053	;* Inputs : None. *
		=6054	;* Outputs : None. *
		=6055	;* Destroys : A, B, D, E, H, L and flags. *
		=6056	;* Calls : CODE. *
		=6057	;* *
		=6058	;* Description: This routine outputs the code buffer via the *
		=6059	;* CODE routine. Each byte is passed to the Code routine in *
		=6060	;* the acc with it's address in the HL register pair. *
		=6061	;* *
		=6062	=====
		=6063	;
		=6064	OCODBUF: ; Out code buffer.
2407	3A5DE9	=6065	LDA BYTES ; Get byte count for inst.
240A	47	=6066	MOV B,A ;
240B	11DFE8	=6067	LXI D,CODBUF ;
240E	2A65E9	=6068	LHLD PC ;
		=6069	OCODBL: ;
2411	1A	=6070	LDAX D ; Get code byte from the code buffer.
2412	CD8923	=6071	CALL CODE ; Gen byte.
2415	13	=6072	INX D ; Inc buffer pointer
2416	23	=6073	INX H ; Inc PC
2417	05	=6074	DCR B ; Dec byte counter
2418	C21124	=6075	JNZ OCODBL ; Loop until bytes left = zero.
		=6076	;
241B	C9	=6077	RET ; Return from <OCODBUF>
		=6078	;
		=6079	End of code generation routines (CODE.A85)
		6080	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 106

LOC	OBJ	LINE	SOURCE STATEMENT
		6081 ;	
		6082 ;	-----
		6083 ;	
		6084 ;	EXPRESS I O N EVALUATION ROUTINES.
		6085 ;	
		6086	\$INCLUDE(ASMCALC.A85)
		=6087 ;	-----
		=6088 ;	
		=6089 ;	Expression evaluation module for the UCT SABus assembler ;
		=6090 ;	----- ;
		=6091 ;	
		=6092 ;	This module contains (in order) the following subroutines: ;
		=6093 ;	
		=6094 ;	a) REXP - Main read expression routine. ;
		=6095 ;	
		=6096 ;	Subroutines of REXP: ;
		=6097 ;	
		=6098 ;	i) OKGETV - Ok get value. ;
		=6099 ;	ii) OPERF - Operator found. ;
		=6100 ;	iii) QNUML - ? limited number. ;
		=6101 ;	iv) QUOTEF - Quote found. ;
		=6102 ;	v) RPARF - Right parenthesis found. ;
		=6103 ;	
		=6104 ;	-----
		=6105 ;	
		=6106	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=6107 ;	=====
		=6108 ;*	*
		=6109 ;*	Function : REXP (Read expression) *
		=6110 ;*	Inputs : HL = Start of expression. *
		=6111 ;*	Outputs : Z = 1 : Expression read OK (value = HL) *
		=6112 ;*	Z = 0 : Expression error. (HL = Value = 0) *
		=6113 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=6114 ;*	Calls : AINIT, RUNTOB, QOPERS, OPERF, RPARF, OPUSH *
		=6115 ;*	QUOTE, OKGETV, FERR, OPOP, EXOPER, SEMILF *
		=6116 ;*	and APOP. *
		=6117 ;*	*
		=6118 ;*	Description : This routine reads an expression from the *
		=6119 ;*	source line. *
		=6120 ;*	*
		=6121 ;	=====
		=6122 ;	
		=6123 REXP:	
241C	CDE226	=6124	CALL AINIT ; Entry point for the Read_expression routine
		=6125	; Init arithmetic and operator stacks
		=6126	
241F	3EFF	=6127	MVI A, LABEL
2421	3252E9	=6128	STA NEXTID ; Set flag to indicate that the next symbol
		=6129 ;	; in the expression should be a label/const
		=6130 ;	-----
		=6131 ;	
		=6132 ;	Main expression evaluation loop.
		=6133 ;	
		=6134 NXCIE:	
2424	2A00E8	=6135	LHLD POINT ; Load assembler pointer and load char
2427	CD5F13	=6136	CALL RUNTOB ; Skip characters until char#tab or char#space
		=6137	
242A	CD6E13	=6138	CALL SEMILF ;
242D	CA8424	=6139	JZ ENDEXP ; End of expression is indicated by one of the
		=6140	; following characters ',', '; or LF
2430	FE2C	=6141	CPI ',' ;
2432	CA8424	=6142	JZ ENDEXP ;
		=6143 ;	
		=6144 ;	-----
		=6145 ;	
		=6146 ;	Tabs and blanks skipped, next character is not an expression
		=6147 ;	terminator (LF, Semi-colon, comma). Check for operator.
		=6148 ;	
2435	CD4C26	=6149	CALL QOPERS ; Operator ?
2438	CA4424	=6150	JZ REXP05 ; No - Then Look for next possible symbol
		=6151	; in the expression.
		=6152	
243B	CD7D25	=6153	CALL OPERF ; Else Process operator.
243E	C2B724	=6154	JNZ ERREX ; If operator error then exit with error
		=6155	
2441	C38124	=6156	JMP REXP99 ; Else continue expression evaluation.
		=6157 ;	
		=6158 ;	-----
		=6159 ;	
		=6160 ;	Not an operator, is it a rightleft parenthesis ?
		=6161 ;	
		=6162 REXP05:	
2444	2A00E8	=6163	LHLD POINT ; Not an Operator.
2447	7E	=6164	MOV A, M ; Fetch asm pointer
2448	FE29	=6165	CPI ')' ; Fetch current character from the text.
244A	C25624	=6166	JNZ REXP10 ; Char = right par ?
		=6167	; No then continue.
		=6168	
244D	CDFE25	=6169	CALL RPARF ; Else Process right parenthesis.
2450	C2B724	=6170	JNZ ERREX ; If error then exit with error.
		=6171	
2453	C38124	=6172	JMP REXP99 ; Else continue with expression evaluation
		=6173 ;	
		=6174 ;	-----
		=6175 ;	
		=6176 ;	Not right parenthesis, check for left parenthesis.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 108

LOC	OBJ	LINE	SOURCE STATEMENT
		=6177	REXP10: ;
2456	FE28	=6178	CPI '(' ; left Par ?
2458	C26524	=6179	JNZ REXP15 ; No then continue.
		=6180	;
		=6181	;
		=6182	;
245B	CD9227	=6183	CALL OPUSH ; Else process left parenthesis by simply
245E	23	=6184	INX H ; pushing it onto the operator stack.
245F	2200E8	=6185	SHLD POINT ; Push left parenthesis.
2462	C38124	=6186	JMP REXP99 ; Inc pointer past '('
		=6187	;
		=6188	;
		=6189	;
		=6190	;
		=6191	Not left parenthesis, Could it be a character in quotes ?
		=6192	REXP15: ;
2465	FE27	=6193	CPI QUOTE ; Char = Quote ?
2467	C27324	=6194	JNZ REXP20 ; No - then continue
		=6195	;
246A	CDDC24	=6196	CALL QUOTEF ; Else process quote character.
246D	C2B724	=6197	JNZ ERREX ; If error then exit with error message
		=6198	;
2470	C38124	=6199	JMP REXP99 ; Continue with expression evaluation
		=6200	;
		=6201	;
		=6202	;
		=6203	Not a character in quotes, it must be a factor (Symbol, number
		=6204	PC indicator - i.e. Dollar).
		=6205	;
		=6206	NOTE: First check that the next ID expected is a factor.
		=6207	;
		=6208	REXP20: ;
2473	3A52E9	=6209	LDA NEXTID ;
2476	FEFF	=6210	CPI LABELE ; Next_ID = Label_Expected ?
2478	C2A224	=6211	JNZ REXPE1 ; No - then error.
		=6212	;
247B	CD2C25	=6213	CALL OKGETV ; Else - fetch value from the source line.
247E	C2B724	=6214	JNZ ERREX ; If error then exit
		=6215	;
		=6216	;
		=6217	;
		=6218	End of expression evaluation loop.
		=6219	;
		=6220	REXP99: ;
2481	C32424	=6221	JMP NXTCIE ;
		=6222	;
		=6223	;
		=6224	;
		=6225	;
		=6226	End of main calculation loop.
		=6227	;
		=6228	ENDEXP: ; Normal Exit point for <REXP>
2484	3A52E9	=6229	LDA NEXTID ; Check that next ID = operator expected.
2487	FE00	=6230	CPI OPERE ; Label-expected ?
2489	C2AA24	=6231	JNZ REXPE2 ; No - then error.
		=6232	;
248C	2200E8	=6233	SHLD POINT ; Fetch assembly pointer
		=6234	ENDEX: ;
248F	CD7927	=6235	CALL OPOP ; Pull top operator
2492	FE24	=6236	CPI '\$' ; Bottom of stack marker ?
2494	CAD524	=6237	JZ EXEXP ; Yes - then exit with every thing OK
		=6238	;
2497	FE28	=6239	CPI '(' ; Left parenthesis ?
2499	CAB224	=6240	JZ REXPE3 ; Yes - then the expression has excess left
		=6241	parenthesis.
		=6242	;
249C	CD2326	=6243	CALL EXOPER ; Else execute operators left on the stack
249F	C38F24	=6244	JMP ENDEX ; Loop until bottom of stack found or error.
		=6245	;
		=6246	;

LOC	OBJ	LINE	SOURCE STATEMENT
		=6247 ;	
		=6248 ;	Read expression error exit points:
		=6249 ;	
		=6250 REXPE1:	
24A2	3E0F	=6251	MVI A,15 ; Read expression error #1
24A4	CD3E21	=6252	CALL PERR ;
24A7	C3B724	=6253	JMP ERREX ; Gen operator expected error.
		=6254 ;	
		=6255 ;	-----
		=6256 ;	
		=6257 REXPE2:	
24AA	3E0D	=6258	MVI A,13 ; Read expression error #2
24AC	CD3E21	=6259	CALL PERR ;
24AF	C3B724	=6260	JMP ERREX ; Gen factor expected error.
		=6261 ;	
		=6262 ;	-----
		=6263 ;	
		=6264 REXPE3:	
24B2	3E0B	=6265	MVI A,11 ; Read expression error #3
24B4	CD3E21	=6266	CALL PERR ;
		=6267 ;	Gen excess left parenthesis error.
		=6268 ;	-----
		=6269 ;	
		=6270 ;	Final error exit point (Look for valid expression terminator).
		=6271 ;	
		=6272 ERREX:	
24B7	2A00E8	=6273	LHLD POINT ; Error exit point for <REXP>
		=6274	WFORVE: ; Fetch assembly pointer
24BA	7E	=6275	MOV A,M ;
24BB	CD6E13	=6276	CALL SEMILF ; While Text(Pointer) <> Semi-colon or
24BE	CACA24	=6277	JZ ERREX1 ; Text(Pointer) <> Line-feed or
		=6278	;
24C1	FE2C	=6279	CPI ',' ;
24C3	CACA24	=6280	JZ ERREX1 ; Text(Pointer) <> Comma Do
		=6281	;
24C6	23	=6282	INX H ;
24C7	C3BA24	=6283	JMP WFORVE ; Pointer := Pointer + 1;
		=6284 ;	
		=6285 ;	-----
		=6286 ;	
		=6287 ERREX1:	
24CA	2200E8	=6288	SHLD POINT ;
		=6289	Save new assembly pointer (Points to
24CD	210000	=6290	LXI H,0000H ; a valid expression terminator.
24D0	3EFF	=6291	MVI A,OFFH ; Reset value to Zero
24D2	C3DA24	=6292	JMP REXPFE ; Clear Zero flag => Error.
		=6293 ;	Exit.
		=6294 ;	-----
		=6295 ;	
		=6296 EXEXP:	
24D5	CDFD26	=6297	CALL APOP ; Expression exit point for OK expression
24D8	3E00	=6298	MVI A,0 ; Pull the expression value off the stack
		=6299 ;	Set zero flag => Expression OK
		=6300 ;	-----
		=6301 ;	
		=6302 ;	Read expression final exit point.
		=6303 ;	
		=6304 REXPFE:	
24DA	B7	=6305	ORA A ; Read expression final exit
24DB	C9	=6306	RET ; Set flags before returning.
		=6307 ;	Return from <REXP> routine
		=6308 ;	=====
		=6309 ;	*
		=6310 ;	Function : QUOTEF (Quote found). *
		=6311 ;	Inputs : HL = Asm pointer (Pointing to a quote) *
		=6312 ;	Outputs : Z : Return status (1=OK, 0=Fails). *
		=6313 ;	Destroys : A, H, L and flags. *
		=6314 ;	Calls : GETNSC, APUSH and PERR. *
		=6315 ;	*
		=6316 ;	Description : This routine reads an single quoted character *

LOC	OBJ	LINE	SOURCE STATEMENT
		=6317 ;*	from the source text. The ASCII value of the character is *
		=6318 ;*	pushed, as a 16 bit number, onto the arithmetic stack. *
		=6319 ;*	*
		=6320 ;*	The correct syntax for a single quoted character is as *
		=6321 ;*	follows: '<Character' *
		=6322 ;*	*
		=6323 ;*	Note: This assembler will only allow single quoted *
		=6324 ;*	characters to be used in expressions. *
		=6325 ;*	*
		=6326 ;	=====
		=6327 ;	
		=6328	QUOTE: ; Quote found routine
24DC	23	=6329	INX H ; Inc pointer to char after quote.
24DD	CDEF13	=6330	CALL GETNSC ; Get the next string character
24E0	CAFB24	=6331	JZ EMPTYQ ; If GET Next String Character returns zero the
		=6332	; terminating quote found.
24E3	FE0A	=6333	CPI LF ; Is the character a line feed.
24E5	CA0025	=6334	JZ NORQ ; If it is the the right quote is missing
		=6335 ;	
		=6336 ;	-----
		=6337 ;	
		=6338 ;	The character following the first quote is not a second quote or
		=6339 ;	a linefeed, so treat it as an ASCII constant and push it (as a 16
		=6340 ;	bit number, with the top byte zeroed) onto the arithmetic stack.
		=6341 ;	
24E8	E5	=6342	PUSH H ; Save asm pointer
24E9	6F	=6343	MOV L,A ; Move character to L reg.
24EA	2600	=6344	MVI H,0 ; Zero H reg , HL = ASC(Character)
24EC	CD2327	=6345	CALL APUSH ; Push ASCII value onto arith stack
24EF	E1	=6346	POP H ; Restore asm pointer
		=6347 ;	
		=6348 ;	-----
		=6349 ;	
		=6350 ;	There was at least 1 character in the quotes. Now check that
		=6351 ;	there was only one, the next string character should return zero.
		=6352 ;	
24F0	CDEF13	=6353	CALL GETNSC ; Fetch next string char, should be a quote.
24F3	CA0D25	=6354	JZ CHARFO ; Quote? Yes - then Right quote found.
		=6355 ;	
		=6356 ;	-----
		=6357 ;	
		=6358 ;	The second Get_string_character did not return zero - i.e. there is
		=6359 ;	more than one character in quotes. (This is an error).
		=6360 ;	
24F6	3E13	=6361	MVI A,19 ; More than one character (Error exit)
24F8	C30225	=6362	JMP QERROR ; Exit via quote error point.
		=6363 ;	
		=6364 ;	-----
		=6365 ;	
		=6366 ;	The first call to Get_string_character returned zero - i.e. there are
		=6367 ;	no characters between the quotes ""'".
		=6368 ;	
		=6369	EMPTYQ: ; Empty quote in Exp = Error.
24FB	3E0D	=6370	MVI A,13 ;
24FD	C30225	=6371	JMP QERROR ;
		=6372 ;	
		=6373 ;	-----
		=6374 ;	
		=6375 ;	The second call to Get_string_character returned nonzero and the
		=6376 ;	character returned was a linefeed. "<Character><LF>"
		=6377 ;	
		=6378	NORQ: ; No write quote found in string
2500	3E10	=6379	MVI A,16 ;
		=6380 ;	
		=6381 ;	-----
		=6382 ;	
		=6383 ;	Quote found error exit point.
		=6384 ;	
		=6385	QERROR: ; Quote routine error exit point.
2502	2200E8	=6386	SHLD POINT ; Save asm pointer to after the error.

LOC	OBJ	LINE	SOURCE STATEMENT
2505	CD3E21	=6387	CALL PERR ; Print the error message (Error_number = Areg)
2508	3EFF	=6388	MVI A,OFFH ; Reset zero flag (error return)
250A	C31725	=6389	JMP QUOTER ; Exit quote-routine with error
		=6390 ;	
		=6391 ;	
		=6392 ;	
		=6393 ;	If the quote section has the correct syntax then come here -
		=6394 ;	OK exit point. ("<Character>")
		=6395 ;	
		=6396	CHARFO: ; Character found.
250D	2200E8	=6397	SHLD POINT ; Save new pointer. Points to character
		=6398	; after the last quote.
2510	3E00	=6399	MVI A,OPERE ;
2512	3252E9	=6400	STA NEXTID ; Next_ID_flag = Operator expected
2515	3E00	=6401	MVI A,0 ; Return zero (Character OK)
		=6402 ;	
		=6403 ;	
		=6404 ;	
		=6405 ;	Final exit point for the quote found routine.
		=6406 ;	
		=6407	QUOTER: ; Quote found return
2517	B7	=6408	ORA A ; Set flags before returning
2518	C9	=6409	RET ; Return from <QUOTEF>
		=6410 ;	
		=6411 ;	=====
		=6412 ;	*
		=6413 ;	Function : QNUML (? Number limited) *
		=6414 ;	Inputs : Acc = Digit to be checked. *
		=6415 ;	Outputs : Z=1 : Digit is in the range 0 to 9. *
		=6416 ;	Z=0 : Otherwise. *
		=6417 ;	Destroys : A and flags. *
		=6418 ;	Calls : None *
		=6419 ;	*
		=6420 ;	Description: This routine is used to decide if a value is a *
		=6421 ;	constant or a label. (i.e. does it start with a Letter A *
		=6422 ;	to Z or does it start with a Limited number 0 to 9. *
		=6423 ;	*
		=6424 ;	=====
		=6425 ;	
		=6426	QNUML: ; ? Number routine.
2519	FE30	=6427	CPI '0' ; Compare digit with '0'
251B	FA2825	=6428	JM NOTLN5 ; If Digit < '0' then it's not a number.
		=6429	;
251E	FE3A	=6430	CPI '9'+1 ; Compare digit with '9'+1
2520	F22825	=6431	JP NOTLN5 ; If digit >= '9'+1 then it's not a number.
		=6432	;
2523	3E00	=6433	MVI A,00H ; Else digit is a number.
2525	C32A25	=6434	JMP QNUMLE ; Set status = 0 (OK) and return.
		=6435	;
		=6436	NOTLN5: ; If not number (0 to 9) then
2528	3EFF	=6437	MVI A,OFFH ; Set status <> 0 (ERR) and return.
		=6438	QNUMLE: ;
252A	B7	=6439	ORA A ; Set flags.
252B	C9	=6440	RET ; Return from <QNUML>
		=6441 ;	
		=6442 ;	=====
		=6443 ;	*
		=6444 ;	Function : OKGETV (Ok get value) *
		=6445 ;	Inputs : HL = Asm pointer. *
		=6446 ;	Outputs : Z : Return status (1=OK, 0=fails) *
		=6447 ;	Destroys : H, L, A and flags. *
		=6448 ;	Calls : COPYL, FETCH, PERR, QNUML, READNU, QLET and *
		=6449 ;	APUSH. *
		=6450 ;	*
		=6451 ;	Description : This routine attempts to read a value *
		=6452 ;	(i.e. a number, a dollar or a label) and push it onto the *
		=6453 ;	arithmetic stack. *
		=6454 ;	*
		=6455 ;	=====
		=6456 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 112

LOC	OBJ	LINE	SOURCE STATEMENT
		=6457	OKGETV: ; Routine to read value from text.
252C	7E	=6458	MOV A,M ; Fetch current source character
252D	FE24	=6459	CPI '\$' ; Char = '\$' => Value = Program counter.
252F	C23E25	=6460	JNZ NPCVAL ;
		=6461 ;	
		=6462 ;	
		=6463 ;	
		=6464 ;	The 1st character is a dollar ('\$'), so value = Current PC.
		=6465 ;	
2532	23	=6466	INX H ; Inc pointer past the Dollar.
2533	2200E8	=6467	SHLD POINT ;
2536	2A65E9	=6468	LHLD PC ; Value read to HL = current PC
2539	3E00	=6469	MVI A,0 ; Zero return => value read ok.
253B	C37125	=6470	JMP VREAD ; Exit Read value routine.
		=6471 ;	
		=6472 ;	
		=6473 ;	
		=6474 ;	Check to see if the 1st character = number ('0' to '9')
		=6475 ;	
		=6476	NPCVAL: ; Char not dollar.
253E	CD1925	=6477	CALL QNUML ; Digit between 0 and 9 ?
2541	C24A25	=6478	JNZ NOTLN ;
		=6479	;
2544	CD5029	=6480	CALL READNU ; Read the number from the source.
2547	C37125	=6481	JMP VREAD ;
		=6482 ;	
		=6483 ;	
		=6484 ;	
		=6485 ;	Not a number, check for valid symbol start character.
		=6486 ;	
		=6487	NOTLN: ;
254A	7E	=6488	MOV A,M ; Fetch character.
254B	CD5C14	=6489	CALL QLET ; Letter ?
254E	C26A25	=6490	JNZ NOTVSC ; No - then it's not a valid symbol character.
		=6491 ;	
		=6492 ;	
		=6493 ;	
		=6494 ;	The 1st character is a letter, so fetch the symbol value.
		=6495 ;	
		=6496	FETCHV: ;
2551	21E3E8	=6497	LXI H,LABELB ; Else fetch the value as a label.
2554	CDBF13	=6498	CALL COPYL ; Copy label into the label buffer.
2557	CD2E18	=6499	CALL FETCH ; Fetch the label from the symbol table.
		=6500	; (a zero return => value read OK)
255A	2AE9E8	=6501	LHLD LABELB+LABBL ; HL = Value read, A = Read status.
255D	CA7125	=6502	JZ VREAD ;
		=6503 ;	
		=6504 ;	
		=6505 ;	
		=6506 ;	If FETCH returns non-zero then undefined symbol error.
		=6507 ;	
2560	3E02	=6508	MVI A,2 ; Gen 'Undefined symbol' error.
2562	CD3E21	=6509	CALL PERR ;
2565	3EFF	=6510	MVI A,OFFH ; Error return => Value not ok.
2567	C37125	=6511	JMP VREAD ;
		=6512 ;	
		=6513 ;	
		=6514 ;	
		=6515 ;	Character not a valid starting character for a symbol, => ERROR.
		=6516 ;	
		=6517	NOTVSC: ;
256A	3E27	=6518	MVI A,39 ;
256C	CD3E21	=6519	CALL PERR ; Generate 'Invalid symbol character' error
256F	3EFF	=6520	MVI A,OFFH ;
		=6521 ;	
		=6522 ;	
		=6523 ;	
		=6524 ;	Exit point for value fetch routine, Value in HL and Status in Acc.
		=6525 ;	
		=6526	VREAD: ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 113

LOC	OBJ	LINE	SOURCE STATEMENT
2571	F5	=6527	PUSH PSW ; Save Value read status.
2572	CD2327	=6528	CALL APUSH ; Save value on the Arith stack.
2575	3E00	=6529	MVI A,OPERE ;
2577	3252E9	=6530	STA NEXTID ; Set Nextid = Operator.
257A	F1	=6531	POP PSW ; Restore value read status.
257B	B7	=6532	ORA A ; Set flags before returning.
257C	C9	=6533	RET ; Return from value read routine.
		=6534 ;	
		=6535 ;=====	
		=6536 ;	
		=6537 ;	Function : OPERF (Operator found routine) *
		=6538 ;	Inputs : HL = Asm pointer. (See text for details) *
		=6539 ;	Outputs : Z : Return status (1=OK, 0=fails). *
		=6540 ;	Destroys : H, L, B, C, A and flags. *
		=6541 ;	Calls : PERR, OPUSH, OPOP and EXOPER. *
		=6542 ;	
		=6543 ;	Description : This routine handles operators. The routine *
		=6544 ;	checks the NEXTID flag to see if an operator or a factor *
		=6545 ;	was expected. The following is a table of which operators *
		=6546 ;	are valid with the NEXTID flag set to OPERE or LABEL. *
		=6547 ;	
		=6548 ;	Op_num Operator OPER_Expected LABEL_Expected *
		=6549 ;	
		=6550 ;	1 OR Valid Invalid *
		=6551 ;	2 XOR Valid Invalid *
		=6552 ;	3 AND Valid Invalid *
		=6553 ;	4 NOT Invalid (*) Valid *
		=6554 ;	5 '+' Valid Invalid *
		=6555 ;	6 & 7 '-' Valid (Minus) Valid (Negate) *
		=6556 ;	8 '*' Valid Invalid *
		=6557 ;	9 '/' Valid Invalid *
		=6558 ;	10 MOD Valid Invalid *
		=6559 ;	11 SHR Valid Invalid *
		=6560 ;	12 SHL Valid Invalid *
		=6561 ;	13 LOW Invalid (*) Valid *
		=6562 ;	14 HIGH Invalid (*) Valid *
		=6563 ;	
		=6564 ;	(*) These operators are referred too as TYPE1 operators. *
		=6565 ;	
		=6566 ;	Note: The assembly pointer in the HL register pair on entry *
		=6567 ;	to this routine points to the operator if it is a *
		=6568 ;	single character ('+', '-', '*', '/') or to the last *
		=6569 ;	character of the string operator ('HIGH' etc). *
		=6570 ;	
		=6571 ;=====	
		=6572 ;	
		=6573	OPERF: ; Routine for operators.
257D	23	=6574	INX H ;
257E	2200E8	=6575	SHLD POINT ; Save new asm pointer.
2581	4F	=6576	MOV C,A ; Operator in the C register.
		=6577 ;	
		=6578 ;	Check for Operators off the type HIGH/LOW/NOT
		=6579 ;	
2582	FE0E	=6580	CPI HIGHC ; Oper = HIGH ?
2584	CAAF25	=6581	JZ TYPE1 ; Yes - Oper of type 1
		=6582	
2587	FE0D	=6583	CPI LOWC ; Oper = LOW ?
2589	CAAF25	=6584	JZ TYPE1 ; Yes - Oper of type 1
		=6585	
258C	FE04	=6586	CPI NOTC ; Oper = NOT ?
258E	CAAF25	=6587	JZ TYPE1 ; Yes - Oper of type 1
		=6588	
2591	3A52E9	=6589	LDA NEXTID ;
2594	FE00	=6590	CPI OPERE ; NEXT_ID_Flag = operator ?
2596	CACA25	=6591	JZ OKOPER ; Yes - then Operator OK.
		=6592	
2599	79	=6593	MOV A,C ; Restore Operator
259A	FE06	=6594	CPI MINUSC ; Oper = MINUS ?
259C	CAC225	=6595	JZ NEGATE ; Yes then MINUS = NEGATE.
		=6596	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 114

LOC	OBJ	LINE	SOURCE STATEMENT
259F	FE05	=6597	CPI PLUSC ; Oper = PLUS ?
25A1	CAF525	=6598	JZ ENDOPR ; Yes then Ignore Plus sign (e.g. +2 = 2)
		=6599	;
25A4	3E0D	=6600	MVI A,13 ; Else Error
25A6	CD3E21	=6601	CALL PERR ; Operator found, but Label expected.
25A9	3EFF	=6602	MVI A,OFFH ;
25AB	B7	=6603	ORA A ; Reset zero flag => error.
25AC	C3FD25	=6604	JMP OPERET ;
		=6605 ;	
		=6606 ;	
		=6607 ;	
		=6608 ;	Operator of TYPE 1 (HIGH/LOW/NOT).
		=6609 ;	
		=6610	TYPE1: ;
25AF	3A52E9	=6611	LDA NEXTID ;
25B2	FEFF	=6612	CPI LABELE ; NEXT_ID_Flag = Label ?
25B4	CAE225	=6613	JZ STKIT ; Yes - then stack the operator.
		=6614	;
25B7	3E0F	=6615	MVI A,15 ; Else Type 1 operator found
25B9	CD3E21	=6616	CALL PERR ; but Operator expected.
25BC	3EFF	=6617	MVI A,OFFH ;
25BE	B7	=6618	ORA A ; Reset Zero flag => error.
25BF	C3FD25	=6619	JMP OPERET ;
		=6620 ;	
		=6621 ;	
		=6622 ;	
		=6623 ;	If a factor is expected and a minus sign is encountered then
		=6624 ;	assume it is a unary minus.
		=6625 ;	
		=6626	NEGATE: ; If Negate then
25C2	3E07	=6627	MVI A,NEGC ;
25C4	CD9227	=6628	CALL OPUSH ; Opush(NEGAtE_Command)
25C7	C3FD25	=6629	JMP OPERET ;
		=6630 ;	
		=6631 ;	
		=6632 ;	
		=6633 ;	The operator is OK and expected (i.e. NEXTID was set to OPERE).
		=6634 ;	
		=6635	OKOPER: ;
25CA	79	=6636	MOV A,C ; Get operator from C register.
25CB	47	=6637	MOV B,A ;
25CC	CD7927	=6638	CALL OPOP ; Get operator from the stack.
25CF	CD9227	=6639	CALL OPUSH ; Save on the stack incase it is not used.
		=6640	;
25D2	FE24	=6641	CPI '\$' ; Bottom of stack ?
25D4	CAE225	=6642	JZ STKIT ; Yes - Then place character just read on
		=6643	the stack.
25D7	FE28	=6644	CPI '(' ;
25D9	C2DE25	=6645	JNZ NOTLP ; Left bracket has precedence of 0.
		=6646	;
25DC	3E00	=6647	MVI A,00H ;
		=6648	NOTLP: ;
25DE	B8	=6649	CMP B ; Compare value just read with the one from
		=6650	the top of the stack.
25DF	F2E925	=6651	JP OPERGT ;
		=6652	;
		=6653	STKIT: ; Stack operator just read.
25E2	79	=6654	MOV A,C ;
25E3	CD9227	=6655	CALL OPUSH ;
25E6	C3F525	=6656	JMP ENDOPR ;
		=6657	OPERGT: ;
25E9	CD7927	=6658	CALL OPOP ; Get operator from the top of the stack.
25EC	47	=6659	MOV B,A ; Save it in B.
25ED	79	=6660	MOV A,C ; Get operator just read
25EE	CD9227	=6661	CALL OPUSH ; and push it onto the stack.
25F1	78	=6662	MOV A,B ;
25F2	CD2326	=6663	CALL EXOPER ; Execute the operator that was on top of the
		=6664	Stack.
		=6665	ENDOPR: ;
25F5	3EFF	=6666	MVI A,LABELE ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 115

LOC	OBJ	LINE	SOURCE STATEMENT
25F7	3252E9	=6667	STA NEXTID ; Set next ID flag = label expected.
25FA	3E00	=6668	MVI A,0 ;
25FC	B7	=6669	ORA A ; Set zero flag => operator OK.
		=6670	OPERET: ;
25FD	C9	=6671	RET ; Return from <OPERF> routine.
		=6672 ;	
		=6673 ;	=====
		=6674 ;*	*
		=6675 ;*	Function : RPARF (Right parenthesis found) *
		=6676 ;*	Inputs : HL = Pointer to right parenthesis. *
		=6677 ;*	Outputs : Z = 1: Parenthesis processed OK. *
		=6678 ;*	Z = 0: Error in expression. *
		=6679 ;*	Destroys : H, L, A and flags. *
		=6680 ;*	Calls : OPOP, PERR and EXOPER. *
		=6681 ;*	*
		=6682 ;*	Description : This routine processes a right parenthesis in *
		=6683 ;*	the source line. The routine unstacks and executes *
		=6684 ;*	operators from the operator stack until either the bottom *
		=6685 ;*	of stack marker or a left parenthesis is found. *
		=6686 ;*	*
		=6687 ;*	If the bottom of stack marker is found then an excess right *
		=6688 ;*	parenthesis error has occurred. If a left parenthesis is *
		=6689 ;*	found then it is simply unstacked and discarded. *
		=6690 ;*	*
		=6691 ;	=====
		=6692 ;	
		=6693	RPARF: ; Right parenthesis found.
25FE	23	=6694	INX H ;
25FF	2200E8	=6695	SHLD POINT ; Inc pointer past ')' and save
		=6696 ;	
		=6697 ;	-----
		=6698 ;	
		=6699 ;	Right parenthesis loop: pull operators off the operator stack
		=6700 ;	and execute them until either a left parenthesis is found
		=6701 ;	or the bottom of the stack is reached. If the bottom of the
		=6702 ;	stack is reached then print an error message and return with
		=6703 ;	the zero flag clear.
		=6704 ;	
		=6705	RPARL: ;
2602	CD7927	=6706	CALL OPOP ; Pop operator off the Operator stack.
2605	FE28	=6707	CPI '(' ; Operator = Left parenthesis ?
2607	CA1526	=6708	JZ RPAROK ; Yes then Exit <RPARF> with Status OK.
		=6709	;
260A	FE24	=6710	CPI '\$' ; Bottom of stack ?
260C	CA1A26	=6711	JZ EXRP ; Yes - then error excess right parenthesis.
		=6712	;
260F	CD2326	=6713	CALL EXOPER ; Execute Operator.
2612	C30226	=6714	JMP RPARL ; Loop.
		=6715 ;	
		=6716 ;	-----
		=6717 ;	
		=6718 ;	Right parenthesis routine OK exit point.
		=6719 ;	
		=6720	RPAROK: ; Right Par routine (Left par found on
		=6721	; Operator stack = return status OK)
2615	3E00	=6722	MVI A,0 ;
2617	C32126	=6723	JMP RPARE ;
		=6724 ;	
		=6725 ;	-----
		=6726 ;	
		=6727 ;	Excess right parenthesis routine.
		=6728 ;	
		=6729	EXRP: ; To many Right pars.
261A	3E0C	=6730	MVI A,12 ; Gen error #12.
261C	CD3E21	=6731	CALL PERR ;
261F	3EFF	=6732	MVI A,OFFH ; Reset zero Flag = Error.
		=6733 ;	
		=6734 ;	-----
		=6735 ;	
		=6736 ;	Right parenthesis exit point:

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 116

LOC	OBJ	LINE	SOURCE STATEMENT
		=6737 ;	
		=6738 RPARE:	
2621	B7	=6739	ORA A ; Set flags before returning
2622	C9	=6740	RET ; Return from then <RPARE> routine.
		=6741 ;	
		=6742 ;	End of asm CALC module. (CALC.A85)
		6743 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 117

LOC	OBJ	LINE	SOURCE STATEMENT
		6744	\$INCLUDE(ASMCALCO.A85)
		=6745	-----;
		=6746	;
		=6747	Arithmetic operator module for the UCT SABus assembler
		=6748	-----;
		=6749	;
		=6750	This module contains (in order) the following subroutines:
		=6751	;
		=6752	a) EXOPER - Execute operator.
		=6753	b) QOPERS - Operator ?
		=6754	c) QSLOPS - Single character operator ?
		=6755	;
		=6756	-----;
		=6757	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 118

LOC	OBJ	LINE	SOURCE STATEMENT
		=6758 ;	
		=6759 ;=====	
		=6760 ;*	
		=6761 ;* Function : EXOPER (Execute Operator)	*
		=6762 ;* Inputs : Acc = Operator type (range 1 to 14)	*
		=6763 ;* Outputs : None.	*
		=6764 ;* Destroys : All	*
		=6765 ;* Calls : One of the 14 Arithmetic evaluation routines.	*
		=6766 ;*	*
		=6767 ;* Description: This routine jumps to 1 of the 14 arithmetic	*
		=6768 ;* evaluation routines given the Operator type in the Acc.	*
		=6769 ;*	*
		=6770 ;* The routine uses the table of vectors starting a SEMROS as	*
		=6771 ;* a jump table.	*
		=6772 ;*	*
		=6773 ;=====	
		=6774 ;	
		=6775 EXOPER:	; Start of Execute operator routine.
2623	112E26	=6776 LXI D,SEMROS-2	;
2626	2600	=6777 MVI H,00H	;
2628	6F	=6778 MOV L,A	; HL = Operator type.
2629	29	=6779 DAD H	; HL = 2 * Operator type.
262A	19	=6780 DAD D	; HL = SEMROS - 2 + 2 * (Operator type)
		=6781 ;	
262B	5E	=6782 MOV E,M	;
262C	23	=6783 INX H	;
262D	56	=6784 MOV D,M	; DE = Vector to operator routine.
262E	EB	=6785 XCHG	;
262F	E9	=6786 PCHL	; Indirect jump Via HL to the correct routine.
		=6787 ;	
		=6788 ;-----	
		=6789 ;	
		=6790 ; Table of vectors to the arithmetic evaluation routines.	
		=6791 ;	
		=6792 SEMROS:	;
2630	CB28	=6793 DW ORROU	; OR Routine.
2632	3F29	=6794 DW XORROU	; XOR Routine.
2634	1028	=6795 DW ANDROU	; AND Routine.
2636	BE28	=6796 DW NOTROU	; NOT Routine.
2638	B627	=6797 DW AADD	; Addition Routine. (+).
263A	3328	=6798 DW ASUB	; Subtraction Routine. (-).
263C	2128	=6799 DW ANEG	; Negation Routine. (-).
263E	E327	=6800 DW AMULT	; Multiplication Routine. (*).
2640	D427	=6801 DW ADIV	; Division Routine. (/).
2642	B728	=6802 DW MODROU	; MOD Routine.
2644	0629	=6803 DW SHRROU	; SHR Routine.
2646	EE28	=6804 DW SHLROU	; SHL Routine.
2648	AE28	=6805 DW LOWROU	; LOW Routine.
264A	A428	=6806 DW HIGHRO	; HIGH Routine.
		=6807 ;	
		=6808 ;=====	
		=6809 ;*	*
		=6810 ;* Function : QOPERS (Operator ?)	*
		=6811 ;* Inputs : A = Current character,	*
		=6812 ;* (POINT):memory = Current assembly pointer	*
		=6813 ;* Outputs : Z=1 : If next symbol is not an operator.	*
		=6814 ;* Z=0 and A=Operator type : If next symbol is an	*
		=6815 ;* operator.	*
		=6816 ;* Destroys : A, H, L, D, E and flags	*
		=6817 ;* Calls : QSLOPS, QLET, COPY and QTAB.	*
		=6818 ;*	*
		=6819 ;* Description :	*
		=6820 ;*	*
		=6821 ;=====	
		=6822 ;	
		=6823 QOPERS:	; Check to see if the next symbol is an operator
264C	CDB626	=6824 CALL QSLOPS	; Is the next char a single letter operator
		=6825	; i.e. ('-', '/', '*', '+')
264F	C28526	=6826 JNZ QOPEX	; If not zero then acc=operand number,
		=6827 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 119

LOC	OBJ	LINE	SOURCE STATEMENT
2652	7E	=6828	MOV A,M ; Fetch character from the text.
2653	CD5C14	=6829	CALL QLET ; Is it a letter ?
2656	C28326	=6830	JNZ NOTSOP ; No - then it's not a string operator.
		=6831	;
2659	2A00E8	=6832	LHLD POINT ; Fetch the assembly pointer
265C	E5	=6833	PUSH H ; Save assembly pointer (just in case the
		=6834	; string read is not a string operator)
265D	2153E9	=6835	LXI H,OPBUF ; HL = Start address of opcode buffer
2660	CD8313	=6836	CALL COPY ; Copy string into Opcode buffer.
2663	2A00E8	=6837	LHLD POINT ; Fetch new assembly pointer
2666	7E	=6838	MOV A,M ; Fetch COPY terminating character.
2667	CD7C14	=6839	CALL QLLET ; Could it be part of a symbol ?
266A	CA7F26	=6840	JZ NSOP ; Yes - Then it's not a string operand
		=6841	;
266D	118726	=6842	LXI D,STROPS ; DE = Start address of string operand table.
2670	CD1514	=6843	CALL QTAB ; Check to see if the string in the Opcode
		=6844	; buffer (i.e. the string just read) is
		=6845	; in the String_Operator table.
2673	FA7F26	=6846	JM NSOP ; No - then it's not a string Operator.
		=6847	;
2676	3C	=6848	INR A ; Else inc acc to give Operator value
		=6849	; between 1 and 14.
2677	E1	=6850	POP H ; Pop Saved pointer of the stack
		=6851	; but disgard it as there was a string
		=6852	; Operator in the text.
2678	2A00E8	=6853	LHLD POINT ; Get the new asm pointer
267B	2B	=6854	DCX H ; Dec pointer to last char of the string
		=6855	; Operator.
267C	C38526	=6856	JMP QOPEX ; Exit <QOPERF>
		=6857	;
		=6858	NSOP: ; If not string Operator then
267F	E1	=6859	POP H ; Restore old asm pointer.
2680	2200E8	=6860	SHLD POINT ;
		=6861	NOTSOP: ;
2683	3E00	=6862	MVI A,NOPC ; Set operand type to NOPC (i.e. not an opcode)
		=6863	QOPEX: ;
2685	B7	=6864	ORA A ; Set flags and exit
2686	C9	=6865	RET ;
		=6866	;
		=6867	-----
		=6868	;
		=6869	Table of String operators (Special character Operators such as '*'
		=6870	are also included in the table, but will never be found.).
		=6871	;
		=6872	STROPS: ;
2687	4F5204	=6873	DB 'OR',EOT ;
268A	584F5204	=6874	DB 'XOR',EOT ;
268E	414E4404	=6875	DB 'AND',EOT ;
2692	4E4F5404	=6876	DB 'NOT',EOT ;
2696	8004	=6877	DB NEVERF,EOT ; PLUS
2698	8004	=6878	DB NEVERF,EOT ; MINUS
269A	8004	=6879	DB NEVERF,EOT ; NEGATE
269C	8004	=6880	DB NEVERF,EOT ; MULT
269E	8004	=6881	DB NEVERF,EOT ; DIVIDE
26A0	4D4F4404	=6882	DB 'MOD',EOT ;
26A4	53485204	=6883	DB 'SHR',EOT ;
26A8	53484C04	=6884	DB 'SHL',EOT ;
26AC	4C4F5704	=6885	DB 'LOW',EOT ;
26B0	48494748	=6886	DB 'HIGH',EOT ;
	04		
26B5	FF	=6887	DB ETABLE ; End of table marker.
		=6888	;
		=6889	=====
		=6890	;
		=6891	* Function : QSLOPS (? Single letter operands) *
		=6892	* Inputs : Acc = Character to be checked. *
		=6893	* Outputs : Acc = Operator code (NOF Code = the character *
		=6894	* was not a single letter Operator) *
		=6895	* Destroys : A and flags. *
		=6896	* Calls : None. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 120

LOC	OBJ	LINE	SOURCE STATEMENT
		=6897 ;*	
		=6898 ;*	Description: This routine is used to check the char in the
		=6899 ;*	Areg against the set of single letter Operator:
		=6900 ;*	
		=6901 ;*	['*', '/', '+', '-']
		=6902 ;*	
		=6903 ;*	If the char is found to be in the set then the Operator
		=6904 ;*	code for the operator is returned, otherwise the Nop code
		=6905 ;*	is returned.
		=6906 ;*	
		=6907 ;=====	
		=6908 ;	
		=6909 QSLOPS:	; Start of <QSLOPS>
26B6	FE2B	=6910	CPI ADDSYM ;
26B8	CACF26	=6911	JZ PLUS ; If Char = ADDSYM ('+') then Operator = PLUSC
		=6912	;
26BB	FE2D	=6913	CPI SUBSYM ;
26BD	CAD426	=6914	JZ MINUS ; If Char = SUBSYM ('-') then Operator = MINUSC
		=6915	;
26C0	FE2F	=6916	CPI DIVSYM ;
26C2	CAD926	=6917	JZ DIVIDE ; If Char = DIVSYM ('/') then Operator = DIVC
		=6918	;
26C5	FE2A	=6919	CPI MULSYM ;
26C7	CADE26	=6920	JZ MULTSF ; If char = MULSYM ('*') then Operator = MULTC
		=6921	;
26CA	3E00	=6922	MVI A,NOPC ; Else char not single letter operator
		=6923	; Operator = NOPC
26CC	C3E026	=6924	JMP QSOPEX ;
		=6925 PLUS:	;
26CF	3E05	=6926	MVI A,PLUSC ;
26D1	C3E026	=6927	JMP QSOPEX ;
		=6928 MINUS:	;
26D4	3E06	=6929	MVI A,MINUSC ;
26D6	C3E026	=6930	JMP QSOPEX ;
		=6931 DIVIDE:	;
26D9	3E09	=6932	MVI A,DIVC ;
26DB	C3E026	=6933	JMP QSOPEX ;
		=6934 MULTSF:	;
26DE	3E08	=6935	MVI A,MULTC ;
		=6936 QSOPEX:	;
26E0	B7	=6937	ORA A ; Set flags before returning.
26E1	C9	=6938	RET ; Return from <QSLOPS>
		=6939 ;	
		=6940 ;	End of module <ASMICALCO.A85> file.
		6941 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 121

LOC	OBJ	LINE	SOURCE STATEMENT
		6942	\$INCLUDE(ASMCALCS.A85)
		=6943 ;	-----;
		=6944 ;	;
		=6945 ;	Arithmetic stack operations module for the UCT SABus assembler ;
		=6946 ;	-----;
		=6947 ;	;
		=6948 ;	This module contains (in order) the following subroutines: ;
		=6949 ;	;
		=6950 ;	a) AINIT - Arithmetic and operator stack initialise routine. ;
		=6951 ;	b) APOP - Arithmetic value POP routine. ;
		=6952 ;	c) APUSH - Arithmetic value PUSH routine. ;
		=6953 ;	d) DINIT - Digit stack initialise routine. ;
		=6954 ;	e) DPUSH - Digit PUSH routine ;
		=6955 ;	f) OPOP - Operator POP routine. ;
		=6956 ;	g) OPUSH - Operator PUSH routine. ;
		=6957 ;	;
		=6958 ;	-----;
		=6959 ;	;
		=6960 ;	;
		=6961 ;	-----;
		=6962 ;	;
		=6963 ;	Stacks managed by the routines in this file: ;
		=6964 ;	;
		=6965 ;	Arithmetic stack: Stack pointer = ASP ;
		=6966 ;	Entry counter = ENTS ;
		=6967 ;	Length (in entries) = ASTKL/2 ;
		=6968 ;	Initial stack pointer = ASTACK ;
		=6969 ;	Stack operation = Word/Post_increment. ;
		=6970 ;	Routines supplied = AINIT, APUSH and APOP. ;
		=6971 ;	;
		=6972 ;	Operator stack: Stack pointer = OSP ;
		=6973 ;	Entry counter = OENTS ;
		=6974 ;	Length (in entries) = OSTKL ;
		=6975 ;	Initial stack pointer = OSTACK ;
		=6976 ;	Stack operation = Byte/Post_increment. ;
		=6977 ;	Routines supplied = AINIT, OPUSH and OPOP. ;
		=6978 ;	;
		=6979 ;	Digit stack: Stack pointer = DSP ;
		=6980 ;	Entry counter = TD ;
		=6981 ;	Length (in entries) = DSTKL ;
		=6982 ;	Initial stack pointer = DSTACK ;
		=6983 ;	Stack operation = Byte/Post_increment. ;
		=6984 ;	Routines supplied = DINIT and DPUSH. ;
		=6985 ;	;
		=6986 ;	-----;
		=6987 ;	;
		=6988	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 122

LOC	OBJ	LINE	SOURCE STATEMENT
		=6989 ;	=====
		=6990 ;*	*
		=6991 ;*	Function : AINIT (Arithmetic and operator stack initialise)*
		=6992 ;*	Inputs : None.
		=6993 ;*	Outputs : None.
		=6994 ;*	Destroys : A and flags.
		=6995 ;*	Calls : None.
		=6996 ;*	*
		=6997 ;*	Description : This routine initialises both the Arithmetic
		=6998 ;*	and Operator stack pointers as well as the entry counters
		=6999 ;*	for both these stacks. The entry counters are used to
		=7000 ;*	protect against stack under/over-flow. A bottom of stack
		=7001 ;*	marker is also places at the bottom of the operator stack.
		=7002 ;*	*
		=7003 ;	=====
		=7004 ;	
		=7005 AINIT:	; Arithmetic Init.
26E2	E5	=7006	PUSH H ;
26E3	21F4E8	=7007	LXI H,ASTACK ;
26E6	2245E9	=7008	SHLD ASP ; Init Arithmetic stack pointer.
		=7009	;
26E9	2112E9	=7010	LXI H,OSTACK ;
26EC	3624	=7011	MVI M,'\$' ; Place '\$' on the bottom of the OP stack
		=7012	; as a marker.
26EE	23	=7013	INX H ; Inc OSP to next free byte on the stack.
26EF	2247E9	=7014	SHLD OSP ; Save new OSP.
		=7015	;
26F2	3E00	=7016	MVI A,00H ;
26F4	3250E9	=7017	STA ENTS ; Init Arithmetic stack entry counter = zero.
26F7	3C	=7018	INR A ;
26F8	3251E9	=7019	STA OENTS ; Init Operator stack entry counter = 1.
		=7020	; (The '\$' on the bottom of the stack)
26FB	E1	=7021	POP H ;
26FC	C9	=7022	RET ; Return from <AINIT>
		=7023 ;	
		=7024 ;	=====
		=7025 ;*	*
		=7026 ;*	Function : APOP (Arithmetic stack POP routine)
		=7027 ;*	Inputs : None.
		=7028 ;*	Outputs : HL = Top value of the Astack.
		=7029 ;*	Destroys : H, L, A and flags.
		=7030 ;*	Calls : PERR.
		=7031 ;*	*
		=7032 ;*	Description : This routine pulls the top value off the
		=7033 ;*	arithmetic stack. This routine implements arithmetic stack
		=7034 ;*	underflow protection by checking for a non zero entry
		=7035 ;*	counter. The value is returned in HL, the entry counter
		=7036 ;*	and arithmetic stack pointer are updated.
		=7037 ;*	*
		=7038 ;	=====
		=7039 ;	
		=7040 APOP:	; Arithmetic stack POP Function.
26FD	D5	=7041	PUSH D ;
26FE	F5	=7042	PUSH PSW ;
		=7043	;
26FF	2A45E9	=7044	LHLD ASP ; Load Arithmetic Stack Pointer.
2702	3A50E9	=7045	LDA ENTS ; Load Arithmetic Stack Entry counter.
2705	B7	=7046	ORA A ; If entry counter = zero then STACK UNDERFLOW.
2706	CA1827	=7047	JZ UNFLOW ;
		=7048	;
2709	3D	=7049	DCR A ; Else Dec counter
270A	3250E9	=7050	STA ENTS ;
270D	2B	=7051	DCX H ;
270E	56	=7052	MOV D,M ; and Fetch the word off the A stack.
270F	2B	=7053	DCX H ;
2710	5E	=7054	MOV E,M ;
2711	2245E9	=7055	SHLD ASP ; Save the new A stack pointer.
2714	EB	=7056	XCHG ;
2715	C32027	=7057	JMP EXAPOP ;
		=7058	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 123

LOC	OBJ	LINE	SOURCE STATEMENT
		=7059	UNFLOW: ;
2718	210000	=7060	LXI H,0000H ; Setup dummy result = zero.
271B	3E12	=7061	MVI A,18 ; Gen Error Number 18 (Stack Underflow)
271D	CD3E21	=7062	CALL PERR ;
		=7063	;
		=7064	EXAPOP: ; Exit APOP Routine.
2720	F1	=7065	POP PSW ;
2721	D1	=7066	POP D ;
2722	C9	=7067	RET ; Return From <APOP>
		=7068 ;	
		=7069 ;=====	
		=7070 ;*	
		=7071 ;*	Function : APUSH (Arithmetic stack PUSH routine) *
		=7072 ;*	Inputs : HL = Number to push onto the Astack. *
		=7073 ;*	Outputs : None. *
		=7074 ;*	Destroys : None. *
		=7075 ;*	Calls : PERR. *
		=7076 ;*	
		=7077 ;*	Description : This routine pushes a value onto the Astack. *
		=7078 ;*	The number is passed to the routine in the HL register *
		=7079 ;*	pair. The routine checks for stack overflow by checking *
		=7080 ;*	that the entry counter is not equal to the maximum entry *
		=7081 ;*	count. *
		=7082 ;*	
		=7083 ;=====	
		=7084 ;	
		=7085	APUSH: ; Arithmetic Push Routine.
		=7086	;
2723	F5	=7087	PUSH PSW ;
2724	E5	=7088	PUSH H ;
2725	D5	=7089	PUSH D ; Save registers
		=7090	;
2726	3A50E9	=7091	LDA ENTS ; Get A stack Entry Count.
2729	FE0F	=7092	CPI ASTKL/2 ; Max entries ?
272B	CA4027	=7093	JZ APUSHE ; Yes - The A stack Overflow.
		=7094	;
272E	3C	=7095	INR A ; Inc Astack entry counter.
272F	3250E9	=7096	STA ENTS ; and update.
		=7097	;
2732	EB	=7098	XCHG ; Save the number to be pushed in DE
2733	2A45E9	=7099	LHLD ASP ; Fetch the Astack pointer
2736	73	=7100	MOV M,E ; Save LSByte first
2737	23	=7101	INX H ; Inc Astack pointer
2738	72	=7102	MOV M,D ; Save MSByte
2739	23	=7103	INX H ; Inc Astack pointer
273A	2245E9	=7104	SHLD ASP ; Save new Astack pointer
273D	C34527	=7105	JMP APUSHX ; exit APUSH routine
		=7106	;
		=7107	APUSHE: ; APUSH Error.
2740	3E22	=7108	MVI A,34 ;
2742	CD3E21	=7109	CALL PERR ; Gen Error #34 (A stack Overflow)
		=7110	;
		=7111	APUSHX: ; Exit from APUSH.
2745	D1	=7112	POP D ;
2746	E1	=7113	POP H ;
2747	F1	=7114	POP PSW ;
2748	C9	=7115	RET ; Return from <APUSH>
		=7116 ;	
		=7117 ;=====	
		=7118 ;*	
		=7119 ;*	Function : DINIT (Digit buffer initialise) *
		=7120 ;*	Inputs : None. *
		=7121 ;*	Outputs : None. *
		=7122 ;*	Destroys : A and flags. *
		=7123 ;*	Calls : None. *
		=7124 ;*	
		=7125 ;*	Description : This routine initialises the digit buffer and *
		=7126 ;*	digit counter (TD). This buffer is used while reading *
		=7127 ;*	numbers from the source text. *
		=7128 ;*	

LOC	OBJ	LINE	SOURCE STATEMENT
		=7129 ;	=====
		=7130 ;	
		=7131 DINIT:	; Digit stack init routine.
2749	E5	=7132	PUSH H ;
274A	3E00	=7133	MVI A,00H ;
274C	324EE9	=7134	STA TD ; Init total digit counter = 0.
274F	2130E9	=7135	LXI H,DSTACK ;
2752	2249E9	=7136	SHLD DSP ; Init Digit stack pointer.
2755	E1	=7137	POP H ;
2756	C9	=7138	RET ; Return from <DINIT>
		=7139 ;	
		=7140 ;	=====
		=7141 ;*	*
		=7142 ;*	Function : DPUSH (Digit buffer push routine) *
		=7143 ;*	Inputs : A = Digit to be pushed. *
		=7144 ;*	Outputs : None. *
		=7145 ;*	Destroys : A and flags. *
		=7146 ;*	Calls : PERR. *
		=7147 ;*	*
		=7148 ;*	Description : This routine pushes digits onto the digit *
		=7149 ;*	stack. Overflow protection is implemented. *
		=7150 ;*	*
		=7151 ;	=====
		=7152 ;	
		=7153 DPUSH:	; Digit Push Routine.
2757	C5	=7154	PUSH B ;
2758	47	=7155	MOV B,A ; Save digit in B reg.
		=7156	;
2759	3A4EE9	=7157	LDA TD ; Get total digit count.
275C	FE15	=7158	CPI DSTKL ; Digit Count = Length of digit buffer ?
275E	CA7227	=7159	JZ DPUSHE ; Yes then Digit stack overflow.
		=7160	;
2761	3C	=7161	INR A ; Inc Total digit counter.
2762	324EE9	=7162	STA TD ;
		=7163	;
2765	E5	=7164	PUSH H ;
2766	2A49E9	=7165	LHLD DSP ; Get digit Stack Pointer.
2769	70	=7166	MOV M,B ; Save Digit On stack.
276A	23	=7167	INX H ; Inc D Stack pointer.
276B	2249E9	=7168	SHLD DSP ;
		=7169	;
276E	E1	=7170	POP H ;
276F	C37727	=7171	JMP DPUSHX ; Exit DPUSH routine.
		=7172	;
		=7173 DPUSHE:	; D PUSH Error.
2772	3E21	=7174	MVI A,33 ;
2774	CD3E21	=7175	CALL PERR ; Gen Error Number 33 (D STK Overflow)
		=7176 DPUSHX:	;
2777	C1	=7177	POP B ;
2778	C9	=7178	RET ; Return from <DPUSH>
		=7179 ;	
		=7180 ;	=====
		=7181 ;*	*
		=7182 ;*	Function : OPOP (Operator stack POP routine) *
		=7183 ;*	Inputs : None. *
		=7184 ;*	Outputs : A = Operator off the stack. *
		=7185 ;*	Destroys : A and flags. *
		=7186 ;*	Calls : PERR. *
		=7187 ;*	*
		=7188 ;*	Description : This routine pulls operator off the operator *
		=7189 ;*	stack. The routine implements stack underflow protection *
		=7190 ;*	and will not allow the operator bottom of stack marker to *
		=7191 ;*	be removed. *
		=7192 ;*	*
		=7193 ;	=====
		=7194 ;	
		=7195 OPOP:	; Operator POP Routine.
2779	E5	=7196	PUSH H ; Save the HL register pair
277A	2A47E9	=7197	LHLD OSP ; Fetch the operator stack pointer
277D	2B	=7198	DCX H ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 125

LOC	OBJ	LINE	SOURCE STATEMENT
277E	7E	=7199	MOV A,M ; Fetch the last operator pushed onto the stack
277F	F5	=7200	PUSH PSW ; Save operator fetched flags
		=7201 ;	
		=7202 ;	
		=7203 ;	
		=7204 ;	Never allow the bottom of stack pointer to be removed. If the
		=7205 ;	entry counter drops to zero then restore it to one and return
		=7206 ;	the bottom of stack marker as the operator.
		=7207 ;	
2780	3A51E9	=7208	LDA OENTS ;
2783	3D	=7209	DCR A ; Dec operator stack entry counter
2784	C28927	=7210	JNZ EXOPOP ; If Entry counter = zero then
		=7211 ;	
2787	23	=7212	INX H ; Inc Ostack pointer
2788	3C	=7213	INR A ; Inc entry counter
		=7214	EXOPOP: ;
2789	3251E9	=7215	STA OENTS ; Save the new operator stack entry counter
278C	2247E9	=7216	SHLD OSP ; Save the new operator stack pointer
		=7217 ;	
278F	F1	=7218	POP PSW ; Restore the operator fetched
2790	E1	=7219	POP H ;
2791	C9	=7220	RET ; Return from <OPOP>
		=7221 ;	
		=7222 ;	=====
		=7223 ;*	*
		=7224 ;*	Function : OPUSH (Operator stack PUSH routine) *
		=7225 ;*	Inputs : A = Operator to push onto the stack. *
		=7226 ;*	Outputs : None. *
		=7227 ;*	Destroys : None. *
		=7228 ;*	Calls : PERR. *
		=7229 ;*	*
		=7230 ;*	Description : The routine pushes operators onto the *
		=7231 ;*	operator stack. The routine implements stack overflow *
		=7232 ;*	protection by checking and updateing the operator stack *
		=7233 ;*	entry counter. *
		=7234 ;*	*
		=7235 ;	=====
		=7236 ;	
		=7237	OPUSH: ; Operator Push routine.
		=7238	;
2792	F5	=7239	PUSH PSW ;
2793	C5	=7240	PUSH B ; Save A, B, C and flags.
		=7241	;
2794	47	=7242	MOV B,A ; Move operator to the B register.
2795	3A51E9	=7243	LDA OENTS ; Fetch operator stack entry counter
2798	FE1E	=7244	CPI OSTKL ; Entry counter = Length of operator stack ?
279A	CAAE27	=7245	JZ OPUSHE ; Yes - Then operator stack overflow
		=7246	;
279D	3C	=7247	INR A ; Inc operator stack entry counter
279E	3251E9	=7248	STA OENTS ; and save.
		=7249	;
27A1	E5	=7250	PUSH H ; Save HL regsiter pair
27A2	2A47E9	=7251	LHLD OSP ; Load operator stack pointer
27A5	70	=7252	MOV M,B ; Save operator fromn B register onto stack
27A6	23	=7253	INX H ; Inc stack pointer
27A7	2247E9	=7254	SHLD OSP ; Save new stack pointer
27AA	E1	=7255	POP H ; restore HL register pair
		=7256	;
27AB	C3B327	=7257	JMP OPUSHX ; Exit OPUSH routine
		=7258	;
		=7259	OPUSHE: ; OPUSH error (Operator stack overflow)
27AE	3E23	=7260	MVI A,35 ;
27B0	CD3E21	=7261	CALL PERR ; Gen error #35
		=7262	;
		=7263	OPUSHX: ;
27B3	C1	=7264	POP B ;
27B4	F1	=7265	POP PSW ; Restore B, C, A and flags.
		=7266	;
27B5	C9	=7267	RET ; Return from <OPUSH>
		=7268 ;	

ASM LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 126

LOC	OBJ	LINE	SOURCE STATEMENT
		=7269 ;	End of <ASMCALCS.A85> file
		7270	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 127

LOC	OBJ	LINE	SOURCE STATEMENT
		7271	\$INCLUDE(ASMCA.LCA.A85)
=7272			-----
=7273			;
=7274			Arithmetic evaluation module for the UCT SABus assembler
=7275			-----
=7276			;
=7277			This module contains (in order) the following subroutines:
=7278			;
=7279		a)	AADD - Add the top two value on the Astack.
=7280		b)	ABS - Calculate the absolute value of the number on top
=7281			of the Astack.
=7282		c)	ADIV - Divide the top two values on the Astack.
=7283		d)	AMULT - Multiply top two values on the Astack.
=7284		e)	ANDROU - Implement AND function.
=7285		f)	ANEG - Negate the top of the Astack (2's complement).
=7286		g)	ASUB - Subtract the top two values of the arithmetic
=7287			stack, placing the result on the stack.
=7288		h)	CSHIFT - Check shift value.
=7289		i)	DIVMOD - Divide the top two values on the Astack giving
=7290			both the result of the DIV and MOD functions.
=7291		j)	HIGHROU - Calculate the HIGH byte of the value on the top
=7292			of the Astack.
=7293		k)	LOWROU - Calculate the LOW byte of the value on the Astack
=7294		l)	MODROU - Calculate the MOD of the top two values on the
=7295			Astack.
=7296		m)	NOTROU - Implement NOT function.
=7297		n)	ORROU - Calculate the OR of the top two values op the
=7298			Astack.
=7299		o)	SHIFTL - Shift the HL register pair 1 bit left.
=7300		p)	SHIFTR - Shift the HL register pair 1 bit right.
=7301		q)	SHLROU - Implement Shift Left function.
=7302		r)	SHRROU - Implement Shift Right function.
=7303		s)	UNSIGN - Unsign the top of the Astack.
=7304		t)	XORROU - Calculate the XOR of the top two values on the
=7305			Astack.
=7306			;
=7307			-----
=7308			;
=7309			\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 128

LOC	OBJ	LINE	SOURCE STATEMENT
		=7310 ;	
		=7311 ;=====	
		=7312 ;*	
		=7313 ;* Function : AADD (Arithmetic addition)	*
		=7314 ;* Inputs : The top 2 numbers on the A stack.	*
		=7315 ;* Outputs : The top number on the A stack.	*
		=7316 ;* Destroys : Acc and Flags.	*
		=7317 ;* Calls : APOP and APUSH.	*
		=7318 ;*	*
		=7319 ;* Description: This routine performs an arithmetic addition	*
		=7320 ;* on the Arithmetic stack.	*
		=7321 ;*	*
		=7322 ;=====	
		=7323 ;	
		=7324 AADD:	; Start of AADD (16 bit addition.)
27B6	D5	=7325	PUSH D ;
27B7	E5	=7326	PUSH H ; Save D,E,H,L Registers.
		=7327	;
27B8	CDFD26	=7328	CALL APOP ;
27BB	EB	=7329	XCHG ;
27BC	CDFD26	=7330	CALL APOP ; Fetch the Operands from the A stack.
		=7331	;
27BF	19	=7332	DAD D ; 16 bit addition (HL = HL + DE)
		=7333	;
27C0	CD2327	=7334	CALL APUSH ; Push result onto the A stack.
		=7335	;
27C3	E1	=7336	POP H ; Restore D,E,H,L Registers.
27C4	D1	=7337	POP D ;
27C5	C9	=7338	RET ; Return from <AADD>
		=7339 ;	
		=7340 ;=====	
		=7341 ;*	
		=7342 ;* Function : ABS (Absolute value)	*
		=7343 ;* Inputs : Top of Astack.	*
		=7344 ;* Outputs : Top of Astack. (The result)	*
		=7345 ;* Destroys : A and flags.	*
		=7346 ;* Calls : APOP, APUSH and ANEG.	*
		=7347 ;*	*
		=7348 ;* Description : The routine calculates the absolute value of	*
		=7349 ;* the number on the Astack. If the sign bit of the number is	*
		=7350 ;* set (i.e. a negative number) then the number is	*
		=7351 ;* complemented and returned to the Astack.	*
		=7352 ;*	*
		=7353 ;=====	
		=7354 ;	
		=7355 ABS:	; Start of <ABS>
		=7356	;
27C6	E5	=7357	PUSH H ; Save HL register pair
		=7358	;
27C7	CDFD26	=7359	CALL APOP ; Fetch the number off the Astack
27CA	CD2327	=7360	CALL APUSH ; Restore number
		=7361	;
27CD	7C	=7362	MOV A,H ; Move the MSByte to the Areg
27CE	B7	=7363	ORA A ;
27CF	FC2128	=7364	CM ANEG ; If the sign_bit of the number is set then
		=7365	; complement the number to make it
		=7366	; positive.
		=7367	;
27D2	E1	=7368	POP H ; Restore the HL register pair
		=7369	;
27D3	C9	=7370	RET ; Return from <ABS>
		=7371 ;	
		=7372 ;=====	
		=7373 ;*	
		=7374 ;* Function : ADIV (Arithmetic division)	*
		=7375 ;* Inputs : The 2 top numbers on the Arith stack.	*
		=7376 ;* Outputs : The top number on the Arith stack.	*
		=7377 ;* Destroys : Acc,H,L,D,E and Flags.	*
		=7378 ;* Calls : DIVMOD, APOP and APUSH.	*
		=7379 ;*	*

LOC	OBJ	LINE	SOURCE STATEMENT
		=7450 ;*	
		=7451 ;*	Function : ANDROU (AND routine)
		=7452 ;*	Inputs : The top two values on the Astack.
		=7453 ;*	Outputs : The top value on the Astack.
		=7454 ;*	Destroys : H, L, D, E, A and flags.
		=7455 ;*	Calls : APOP and APUSH.
		=7456 ;*	
		=7457 ;*	Description : This function performs a bitwise AND on the
		=7458 ;*	top two numbers on the Astack. The result is then pushed
		=7459 ;*	onto the Astack.
		=7460 ;*	
		=7461 ;*	Bit 1 Bit 2 Bit 1 AND Bit 2
		=7462 ;*	-----
		=7463 ;*	0 0 0
		=7464 ;*	0 1 0
		=7465 ;*	1 0 0
		=7466 ;*	1 1 1
		=7467 ;*	
		=7468 ;*	=====
		=7469 ;	
		=7470 ANDROU:	; Start of <ANDROU>
		=7471	;
2810	CDFD26	=7472	CALL APOP ;
2813	EB	=7473	XCHG ;
2814	CDFD26	=7474	CALL APOP ;
		=7475	;
2817	7C	=7476	MOV A,H ;
2818	A2	=7477	ANA D ;
2819	67	=7478	MOV H,A ; H reg = (H reg) AND (D reg).
281A	7D	=7479	MOV A,L ;
281B	A3	=7480	ANA E ;
281C	6F	=7481	MOV L,A ; L reg = (L reg) AND (E reg).
		=7482	;
281D	CD2327	=7483	CALL APUSH ;
		=7484	;
2820	C9	=7485	RET ; Return from <ANDROU>
		=7486 ;	
		=7487 ;*	=====
		=7488 ;*	
		=7489 ;*	Function : ANEG (2's complement negation)
		=7490 ;*	Inputs : The value on the top of the arithmetic stack.
		=7491 ;*	Outputs : The negated value is returned to the stack.
		=7492 ;*	Destroys : None
		=7493 ;*	Calls : APUSH and APOP.
		=7494 ;*	
		=7495 ;*	Description: This routine negates the TOS (Top_of_stack)
		=7496 ;*	(Uses 2's comp negation)
		=7497 ;*	
		=7498 ;*	=====
		=7499 ;	
		=7500 ANEG:	; Start of <ANEG> (16 bit Negation)
2821	E5	=7501	PUSH H ;
2822	F5	=7502	PUSH PSW ; Save H,L,Acc,Flags
		=7503	;
2823	CDFD26	=7504	CALL APOP ; Fetch Operand from the Stack.
		=7505	;
2826	7C	=7506	MOV A,H ;
2827	2F	=7507	CMA ;
2828	67	=7508	MOV H,A ; H = NOT H (complement all bits)
2829	7D	=7509	MOV A,L ;
282A	2F	=7510	CMA ;
282B	6F	=7511	MOV L,A ; L = NOT L
282C	23	=7512	INX H ; HL = HL + 1 (To get 2's complement)
		=7513	;
282D	CD2327	=7514	CALL APUSH ; Push result onto the A stack.
		=7515	;
2830	F1	=7516	POP PSW ;
2831	E1	=7517	POP H ; Restore H,L,Acc,Flags
2832	C9	=7518	RET ; Return from <ANEG>
		=7519 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 131

LOC	OBJ	LINE	SOURCE STATEMENT
		=7520 ;	
		=7521 ;=====	
		=7522 ;*	
		=7523 ;* Function : ASUB (Arithmetic subtraction)	*
		=7524 ;* Inputs : The top 2 numbers on the A stack.	*
		=7525 ;* Outputs : The top number on the A stack.	*
		=7526 ;* Destroys : Acc and Flags.	*
		=7527 ;* Calls : APOP and APUSH.	*
		=7528 ;*	*
		=7529 ;* Description: This routine performs a subtraction on the	*
		=7530 ;* Arithmetic stack.	*
		=7531 ;*	*
		=7532 ;=====	
		=7533 ;	
		=7534 ASUB:	; Subtraction routine.
2833	D5	=7535	PUSH D ;
2834	E5	=7536	PUSH H ; Save DE,HL registers.
		=7537	;
2835	CDFD26	=7538	CALL APOP ;
2838	EB	=7539	XCHG ;
2839	CDFD26	=7540	CALL APOP ; Pop the 2 values of the A Stack.
		=7541	;
283C	7D	=7542	MOV A,L ;
283D	93	=7543	SUB E ; 16 bit subtraction
283E	6F	=7544	MOV L,A ;
283F	7C	=7545	MOV A,H ; HL = HL - DE
2840	9A	=7546	SBB D ;
2841	67	=7547	MOV H,A ;
		=7548	;
2842	CD2327	=7549	CALL APUSH ; Push result onto the A stack.
		=7550	;
2845	E1	=7551	POP H ; Restore DE,HL registers.
2846	D1	=7552	POP D ;
2847	C9	=7553	RET ; Return from <ASUB>
		=7554 ;	
		=7555 ;=====	
		=7556 ;*	*
		=7557 ;* Function : CSHIFT (Check shift)	*
		=7558 ;* Inputs : HL = Shift count.	*
		=7559 ;* Outputs : L = Corrected shift value.	*
		=7560 ;* Z' = 1 : If corrected shift value = zero.	*
		=7561 ;* Z = 0 : If corrected shift value <> zero.	*
		=7562 ;* Destroys : A and flags.	*
		=7563 ;* Calls : PERR.	*
		=7564 ;*	*
		=7565 ;* Description : This routine checks for a valid shift count	*
		=7566 ;* i.e. 0..15, if the shift count is invalid then a	*
		=7567 ;* corrected shift count of zero is returned and an error	*
		=7568 ;* message is printed.	*
		=7569 ;*	*
		=7570 ;=====	
		=7571 ;	
		=7572 CSHIFT:	; Start of <CSHIFT>
		=7573	;
2848	7D	=7574	MOV A,L ; Check that the second operand is of the
2849	E6F0	=7575	ANI 11110000B ; form 000000000000XXXX binary.
284B	B4	=7576	ORA H ;
284C	C25328	=7577	JNZ SHIFTE ; If not zero then shift value error.
		=7578	;
284F	7D	=7579	MOV A,L ; Move LSByte of the number to the acc.
2850	C35A28	=7580	JMP CSHRET ; And exit Check_shift routine
		=7581	;
		=7582 SHIFTE:	;
2853	3E03	=7583	MVI A,3 ; Gen error #3 (Shift out off range >15)
2855	CD3E21	=7584	CALL PERR ;
2858	3E00	=7585	MVI A,00H ; Load acc with zero. i.e. Do not shift the
		=7586	; number at all.
		=7587 CSHRET:	;
285A	B7	=7588	ORA A ; Set flags before returning.
285B	C9	=7589	RET ; Return from <CSHIFT>

LOC	OBJ	LINE	SOURCE STATEMENT
		=7590 ;	
		=7591 ;=====	
		=7592 ;*	
		=7593 ;* Function : DIVMOD (Division / Remainder)	*
		=7594 ;* Inputs : Top 2 values on the arithmetic stack.	*
		=7595 ;* Outputs : Top 2 Values on the arithmetic stack.	*
		=7596 ;* TOS = Signed result.	*
		=7597 ;* TOS-1 = Remainder.	*
		=7598 ;* Destroys : Acc and Flags.	*
		=7599 ;* Calls : UNSIGN, APUSH and ANEG.	*
		=7600 ;*	*
		=7601 ;* Description: This routine divides the TOS by the TOS-1 and	*
		=7602 ;* returns (On the arithmetic stack) the answer and the	*
		=7603 ;* remainder.	*
		=7604 ;*	*
		=7605 ;=====	
		=7606 ;	
		=7607 DIVMOD:	; Start of <DIVMOD>
285C	E5	=7608 PUSH H	;
285D	D5	=7609 PUSH D	; Save regs.
285E	C5	=7610 PUSH B	;
		=7611	;
285F	CD1E29	=7612 CALL UNSIGN	; Unsign the 2 numbers on the top of the
		=7613	; stack and move Unsigned(TOS) = HL,
		=7614	; Unsigned(TOS-1) = DE and
		=7615	; Sign(Result) = SIGN.
2862	44	=7616 MOV B,H	;
2863	4D	=7617 MOV C,L	; Move HL to BC.
		=7618	;
2864	7A	=7619 MOV A,D	;
2865	2F	=7620 CMA	; Complement D reg.
2866	57	=7621 MOV D,A	;
2867	7B	=7622 MOV A,E	;
2868	2F	=7623 CMA	; Complement E reg.
2869	5F	=7624 MOV E,A	;
286A	13	=7625 INX D	; Get 2's complement of DE.
		=7626	;
286B	210000	=7627 LXI H,0000H	; Zero result.
286E	3E11	=7628 MVI A,11H	; Init Shift counter.
		=7629	;
		=7630 DIVLP:	;
2870	E5	=7631 PUSH H	;
2871	19	=7632 DAD D	;
2872	D27628	=7633 JNC RSTRHL	;
		=7634	;
2875	E3	=7635 XTHL	;
		=7636 RSTRHL:	;
2876	E1	=7637 POP H	;
2877	F5	=7638 PUSH PSW	;
		=7639	;
2878	79	=7640 MOV A,C	;
2879	17	=7641 RAL	;
287A	4F	=7642 MOV C,A	;
287B	78	=7643 MOV A,B	;
287C	17	=7644 RAL	;
287D	47	=7645 MOV B,A	; Shift BC 1 bit Left.
		=7646	;
287E	7D	=7647 MOV A,L	;
287F	17	=7648 RAL	;
2880	6F	=7649 MOV L,A	;
2881	7C	=7650 MOV A,H	;
2882	17	=7651 RAL	;
2883	67	=7652 MOV H,A	; Shift HL 1 bit Left.
		=7653	;
2884	F1	=7654 POP PSW	;
2885	3D	=7655 DCR A	; Dec shift counter
2886	C27028	=7656 JNZ DIVLP	; Loop until shift counter = zero.
		=7657	;
2889	B7	=7658 ORA A	; Acc = Zero (Clear carry flag).
288A	7C	=7659 MOV A,H	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 133

LOC	OBJ	LINE	SOURCE STATEMENT
288B	1F	=7660	RAR ;
288C	57	=7661	MOV D,A ;
288D	7D	=7662	MOV A,L ;
288E	1F	=7663	RAR ;
288F	5F	=7664	MOV E,A ; DE = HL shifted 1 bit right.
		=7665 ;	
		=7666 ;	Answer = BC reg pair.
		=7667 ;	Remainder = DE reg pair.
		=7668 ;	
2890	EB	=7669	XCHG ; Move remainder to HL.
2891	CD2327	=7670	CALL APUSH ; Push result onto arithmetic stack.
		=7671 ;	
2894	60	=7672	MOV H,B ;
2895	69	=7673	MOV L,C ; Move answer from BC to HL.
2896	CD2327	=7674	CALL APUSH ; Push answer onto the Arithmetic stack.
		=7675 ;	
2899	3A4FE9	=7676	LDA SIGN ;
289C	B7	=7677	ORA A ;
289D	FC2128	=7678	CM ANEG ; If sign_bit(SIGN) = 1 then Negate Result.
		=7679 ;	
28A0	C1	=7680	POP B ;
28A1	D1	=7681	POP D ; Restore Regs.
28A2	E1	=7682	POP H ;
		=7683 ;	
28A3	C9	=7684	RET ; Return from <DIVMOD>
		=7685 ;	
		=7686 ;	=====
		=7687 ;*	
		=7688 ;*	Function : HIGHRO (High routine). *
		=7689 ;*	Inputs : Top value on Astack. *
		=7690 ;*	Outputs : Top value on Astack (Result of HIGH function) *
		=7691 ;*	Destroys : H, L, A and flags. *
		=7692 ;*	Calls : APOP and APUSH. *
		=7693 ;*	
		=7694 ;*	Description: This routine implements the assembler HIGH *
		=7695 ;*	function by clearing the high byte of the result and *
		=7696 ;*	moving the high byte of the input value to the low byte *
		=7697 ;*	of the result. *
		=7698 ;*	
		=7699 ;	=====
		=7700 ;	
		=7701	HIGHRO: ; Start of <HIGHRO>
		=7702	;
28A4	CDFD26	=7703	CALL APOP ; Pop value off the stack.
28A7	6C	=7704	MOV L,H ; Move high byte to the low byte.
28A8	2600	=7705	MVI H,00H ; Zero the high byte.
28AA	CD2327	=7706	CALL APUSH ; Push the result onto the A stack.
		=7707	;
28AD	C9	=7708	RET ; Return from <HIGHRO>
		=7709 ;	
		=7710 ;	=====
		=7711 ;*	
		=7712 ;*	Function : LOWROU (LOW routine) *
		=7713 ;*	Inputs : Top value on the Astack. *
		=7714 ;*	Outputs : Top value on the Astack. (The result) *
		=7715 ;*	Destroys : H, L, A and flags. *
		=7716 ;*	Calls : APOP and APUSH. *
		=7717 ;*	
		=7718 ;*	Description : This routine implements the assembler LOW *
		=7719 ;*	function by simply clearing the high byte of the input *
		=7720 ;*	number and saving it as the result. *
		=7721 ;*	
		=7722 ;	=====
		=7723 ;	
		=7724	LOWROU: ; Start of <LOWROU>
		=7725	;
28AE	CDFD26	=7726	CALL APOP ; Pop value from the top of the stack.
28B1	2600	=7727	MVI H,00H ; Set the high byte to zero.
28B3	CD2327	=7728	CALL APUSH ; Push the result onto the A stack.
		=7729	;

LOC	OBJ	LINE	SOURCE STATEMENT
28B6	C9	=7730	RET ; Return from <LOWROU>
		=7731 ;	
		=7732 ;	=====
		=7733 ;*	*
		=7734 ;*	Function : MODROU (MOD routine) *
		=7735 ;*	Inputs : Top 2 numbers on the Astack. *
		=7736 ;*	Outputs : Top number on the A stack. *
		=7737 ;*	Destroys : H, L, Acc and Flags. *
		=7738 ;*	Calls : DIVMOD AND APOP. *
		=7739 ;*	*
		=7740 ;*	Description: This routine calculates the MOD of the TOS and *
		=7741 ;*	the TOS-1 and returns it on the Astack. (As with ADIV it *
		=7742 ;*	uses DIVMOD). *
		=7743 ;*	*
		=7744 ;	=====
		=7745 ;	
		=7746 MODROU:	; Start of <MODROU>
		=7747	;
28B7	CD5C28	=7748	CALL DIVMOD ; Calculate DIV and MOD.
28BA	CDFD26	=7749	CALL APOP ; Pop result off stack, leave the remainder.
		=7750	;
28BD	C9	=7751	RET ; Return from <MODROU>
		=7752 ;	
		=7753 ;	=====
		=7754 ;*	*
		=7755 ;*	Function : NOTROU (NOT routine) *
		=7756 ;*	Inputs : Top value on Astack. *
		=7757 ;*	Outputs : Top value of Astack. (The result) *
		=7758 ;*	Destroys : H, L, A and flags. *
		=7759 ;*	Calls : APOP and APUSH. *
		=7760 ;*	*
		=7761 ;*	Description : This routine implements the assembler NOT *
		=7762 ;*	function by simply inverting the number on the Astack. :
		=7763 ;*	*
		=7764 ;	=====
		=7765 ;	
		=7766 NOTROU:	; Start of <NOTROU>
		=7767	;
28BE	CDFD26	=7768	CALL APOP ; Pop value from the stack.
		=7769	;
28C1	7C	=7770	MOV A,H ;
28C2	2F	=7771	CMA ;
28C3	67	=7772	MOV H,A ; Comp H reg.
28C4	7D	=7773	MOV A,L ;
28C5	2F	=7774	CMA ;
28C6	6F	=7775	MOV L,A ; Comp L reg.
		=7776	;
28C7	CD2327	=7777	CALL APUSH ; Push result onto the A stack.
		=7778	;
28CA	C9	=7779	RET ; Return from <NOTROU>
		=7780 ;	
		=7781 ;	=====
		=7782 ;*	*
		=7783 ;*	Function : ORROU (OR routine) *
		=7784 ;*	Inputs : Two top numbers on the Astack. *
		=7785 ;*	Outputs : Top number on the Astack. (The result) *
		=7786 ;*	Destroys : H, L, D, E, A and flags. *
		=7787 ;*	Calls : APOP and APUSH. *
		=7788 ;*	*
		=7789 ;*	Description : The routine implements the assembler OR *
		=7790 ;*	function by using the 8085 OR function. *
		=7791 ;*	*
		=7792 ;*	Bit 1 Bit 2 Bit 1 OR Bit 2 *
		=7793 ;*	-----
		=7794 ;*	0 0 0 *
		=7795 ;*	0 1 1 *
		=7796 ;*	1 0 1 *
		=7797 ;*	1 1 1 *
		=7798 ;*	*
		=7799 ;	=====

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 135

LOC	OBJ	LINE	SOURCE STATEMENT
		=7800 ;	
		=7801 ORROU:	; Start of <ORROU>
		=7802	
28CB	CDFD26	=7803	CALL APOP ; Fetch the first number from the Astack
28CE	EB	=7804	XCHG ; Move it to DE
28CF	CDFD26	=7805	CALL APOP ; Fetch the second number from the Astack
		=7806	
28D2	7C	=7807	MOV A,H ;
28D3	B2	=7808	ORA D ;
28D4	67	=7809	MOV H,A ; H = H OR D
28D5	7D	=7810	MOV A,L ;
28D6	B3	=7811	ORA E ;
28D7	6F	=7812	MOV L,A ; L = L OR E
		=7813	
28D8	CD2327	=7814	CALL APUSH ; Save result on the Arithmetic stack
		=7815	
28DB	C9	=7816	RET ; Return from <ORROU>
		=7817 ;	
		=7818 ;=====	
		=7819 ;*	
		=7820 ;* Function : SHIFTL (Shift left)	*
		=7821 ;* Inputs : HL = 16 bit number to be shifted.	*
		=7822 ;* Outputs : HL = Shifted 16 bit number.	*
		=7823 ;* Destroys : Acc and Flags	*
		=7824 ;* Calls : None.	*
		=7825 ;*	*
		=7826 ;* Description: This routine simply shifts the HL register	*
		=7827 ;* pairs 1 bit left. The carry bit is cleared before the	*
		=7828 ;* shift is carried out so that bit(0) of the result is	*
		=7829 ;* always 0.	*
		=7830 ;*	*
		=7831 ;* Bit(15) of the input word is shifted into the carry bit.	*
		=7832 ;*	*
		=7833 ;=====	
		=7834 ;	
		=7835 SHIFTL:	; Start of <SHIFTL>
28DC	37	=7836	STC ;
28DD	3F	=7837	CMC ; Clear Carry.
		=7838	
28DE	7D	=7839	MOV A,L ;
28DF	17	=7840	RAL ; Shift L reg 1 bit Left. (Bit 0 = 0)
28E0	6F	=7841	MOV L,A ;
		=7842	
28E1	7C	=7843	MOV A,H ;
28E2	17	=7844	RAL ; Shift H reg 1 bit Left.
28E3	67	=7845	MOV H,A ; (bit 0 = old bit 7 of L reg)
		=7846	
28E4	C9	=7847	RET ; Return from <SHIFTL>
		=7848 ;	
		=7849 ;=====	
		=7850 ;*	
		=7851 ;* Function : SHIFTR (Shift Right)	*
		=7852 ;* Inputs : HL = 16 bit number to be shifted.	*
		=7853 ;* Outputs : HL = Shifted 16 bit number.	*
		=7854 ;* Destroys : Acc and Flags.	*
		=7855 ;* Calls : None.	*
		=7856 ;*	*
		=7857 ;* Description: This routine shifts the HL register pair 1 bit	*
		=7858 ;* right. Carry is cleared before the shift so that bit(15)	*
		=7859 ;* will always zero.	*
		=7860 ;*	*
		=7861 ;=====	
		=7862 ;	
		=7863 SHIFTR:	; Start of <SHIFTR>
28E5	37	=7864	STC ;
28E6	3F	=7865	CMC ; Clear Carry.
		=7866	
28E7	7C	=7867	MOV A,H ;
28E8	1F	=7868	RAR ; Shift H 1 bit Right.
28E9	67	=7869	MOV H,A ; (bit 7 = 0)

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 136

LOC	OBJ	LINE	SOURCE STATEMENT
		=7870	
28EA	7D	=7871	MOV A,L
28EB	1F	=7872	RAR
28EC	6F	=7873	MOV L,A
		=7874	
28ED	C9	=7875	RET
		=7876	
		=7877	=====
		=7878	;
		=7879	Function : SHLROU (Shift left routine)
		=7880	Inputs : Top two values on the Astack.
		=7881	Outputs : Top value on the Astack.
		=7882	Destroys : H, L, D, E, A, B and flags.
		=7883	Calls : APOP, SHIFTL, CSHIFT and APUSH.
		=7884	;
		=7885	Description : This routine implements the assembler SHL
		=7886	function, The syntax of the SHL function is as follows:
		=7887	;
		=7888	<Number to shift> SHL <Shift count>
		=7889	;
		=7890	This routine uses a simple shift and not a rotate (i.e.
		=7891	the top bit is lost after every shift).
		=7892	;
		=7893	The routine does bound checking on the shift count which
		=7894	must be in the range 0..15
		=7895	;
		=7896	=====
		=7897	;
		=7898	SHLROU: ; Start of <SHLROU>
		=7899	;
28EE	CDFD26	=7900	CALL APOP ; Pop number of times to shift off the stack
28F1	CD4828	=7901	CALL CSHIFT ; Check that the number of bits that the
		=7902	number must be shifted is within range
28F4	CA0529	=7903	JZ SHLRET ; If second operand zero then leave number
		=7904	on the stack unchanged
28F7	47	=7905	MOV B,A ; Save number of shifts in B register
28F8	CDFD26	=7906	CALL APOP ; Pop number to be shifted from the stack
		=7907	;
		=7908	SHLLOP: ;
28FB	CDDC28	=7909	CALL SHIFTL ; Shift the number 1 bit left
28FE	05	=7910	DCR B ;
28FF	C2FB28	=7911	JNZ SHLLOP ; Loop until counter = zero
		=7912	;
2902	CD2327	=7913	CALL APUSH ; Push result onto the stack
		=7914	SHLRET: ;
2905	C9	=7915	RET ; Return from <SHLROU>
		=7916	;
		=7917	=====
		=7918	;
		=7919	Function : SHRROU (Shift right routine)
		=7920	Inputs : Top two values on the Astack.
		=7921	Outputs : Top value on the Astack.
		=7922	Destroys : H, L, D, E, A, B and flags.
		=7923	Calls : APOP, SHIFTR, CSHIFT and APUSH.
		=7924	;
		=7925	Description : This routine implements the assembler SHR
		=7926	function, The syntax of the SHR function is as follows:
		=7927	;
		=7928	<Number to shift> SHR <Shift count>
		=7929	;
		=7930	This routine uses a simple shift and not a rotate (i.e.
		=7931	the top bit is lost after every shift).
		=7932	;
		=7933	The routine does bound checking on the shift count which
		=7934	must be in the range 0..15
		=7935	;
		=7936	=====
		=7937	;
		=7938	SHRROU: ; Start of <SHRROU>
		=7939	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 137

LOC	OBJ	LINE	SOURCE STATEMENT
2906	CDFD26	=7940	CALL APOP ; Fetch the shift count from the stack
2909	CD4828	=7941	CALL CSHIFT ; Bound check the shift count
290C	CA1D29	=7942	JZ SHRRET ; If shift count = zero then do not shift the
		=7943	; number.
290F	47	=7944	MOV B,A ; Move the shift count to the B register.
2910	CDFD26	=7945	CALL APOP ; Fetch the number to be shifted.
		=7946	SHRLOP ;
2913	CDE528	=7947	CALL SHIFTR ; Shift the number 1 bit to the right
2916	05	=7948	DCR B ; Dec loop counter
2917	C21329	=7949	JNZ SHRLOP ; and loop until counter = zero
		=7950	;
291A	CD2327	=7951	CALL APUSH ; Push the result onto the Astack
		=7952	SHRRET ;
291D	C9	=7953	RET ; Return from <SHRROU>
		=7954 ;	
		=7955 ;=====	
		=7956 ;*	
		=7957 ;*	Function : UNSIGN (Unsign the top two number on the Astack)*
		=7958 ;*	Inputs : The top 2 numbers on the Astack.*
		=7959 ;*	Outputs : HL = First unsigned number off the Astack,*
		=7960 ;*	DE = Second unsigned number off the Astack.*
		=7961 ;*	SIGN (In memory) = Sign of the result of*
		=7962 ;*	multiplication or division.*
		=7963 ;*	Destroys : H, L, D, E, A and flags.*
		=7964 ;*	Calls : APOP, APUSH and ABS.*
		=7965 ;*	
		=7966 ;*	Description: This routine calculates the sign of the result*
		=7967 ;*	and stores in in <SIGN>.*
		=7968 ;*	
		=7969 ;*	Note: Sign1 Sign2 Signres*
		=7970 ;*	
		=7971 ;*	Negative * or / Negative = Positive 1 1 0*
		=7972 ;*	Negative * or / Positive = Negative 1 0 1*
		=7973 ;*	Positive * or / Negative = Negative 0 1 1*
		=7974 ;*	Positive * or / Positive = Positive 0 0 0*
		=7975 ;*	
		=7976 ;*	i.e. Sign_bit(1) XOR Sign_bit(2) = Sign_bit(result)*
		=7977 ;*	
		=7978 ;=====	
		=7979 ;	
		=7980	UNSIGN: ; Start of <UNSIGN>
291E	CDFD26	=7981	CALL APOP ;
2921	EB	=7982	XCHG ;
2922	CDFD26	=7983	CALL APOP ; Get the 2 numbers of the stack.
		=7984	; (DE = Num1, HL = Num2)
		=7985	;
2925	7A	=7986	MOV A,D ;
2926	AC	=7987	XRA H ; XOR sign bits.
2927	324FE9	=7988	STA SIGN ; and store as sign of result.
		=7989	;
292A	CD2327	=7990	CALL APUSH ;
292D	CDC627	=7991	CALL ABS ; Calculate absolute value of HL
2930	CDFD26	=7992	CALL APOP ;
2933	EB	=7993	XCHG ;
		=7994	;
2934	CD2327	=7995	CALL APUSH ;
2937	CDC627	=7996	CALL ABS ; Calculate absolute value of DE
293A	CDFD26	=7997	CALL APOP ;
293D	EB	=7998	XCHG ;
		=7999	;
293E	C9	=8000	RET ; Return from <UNSIGN>
		=8001 ;	
		=8002 ;=====	
		=8003 ;*	
		=8004 ;*	Function : XORROU (Exclusive OR routine)*
		=8005 ;*	Inputs : Top two values on the Astack.*
		=8006 ;*	Outputs : Top value on the Astack. (The result)*
		=8007 ;*	Destroys : H, L, D, E, A and flags.*
		=8008 ;*	Calls : APOP and APUSH.*
		=8009 ;*	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 138

LOC	OBJ	LINE	SOURCE STATEMENT
		=8010 ;*	Description : This routine implements the assembler XOR
		=8011 ;*	function by exclusive ORing the top two numbers on the
		=8012 ;*	Astack. The routine uses the 8085 XRA instruction.
		=8013 ;*	
		=8014 ;*	Bit 1 Bit 2 Bit 1 XOR Bit 2
		=8015 ;*	-----
		=8016 ;*	0 0 0
		=8017 ;*	0 1 1
		=8018 ;*	1 0 1
		=8019 ;*	1 1 0
		=8020 ;*	
		=8021 ;	=====
		=8022 ;	
		=8023 XORROU:	; Start of <XORROU>
		=8024	;
293F	CDFD26	=8025	CALL APOP ; Fetch the first number off the Astack
2942	EB	=8026	XCHG ; move it to DE
2943	CDFD26	=8027	CALL APOP ; Fetch the second number off the Astack
		=8028	;
2946	7C	=8029	MOV A,H ;
2947	AA	=8030	XRA D ;
2948	67	=8031	MOV H,A ; H = H XOR D
2949	7D	=8032	MOV A,L ;
294A	AB	=8033	XRA E ;
294B	6F	=8034	MOV L,A ; L = L XOR E
		=8035	;
294C	CD2327	=8036	CALL APUSH ; Push the result onto the Astack.
		=8037	;
294F	C9	=8038	RET ; Return from <XORROU>
		=8039 ;	
		=8040 ;	End of <ASMCALCA.A85> file.
		8041 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 139

LOC	OBJ	LINE	SOURCE STATEMENT
		8042	\$INCLUDE(ASMREADNU.A85)
		=8043	-----
		=8044	;
		=8045	ASMREADNU module for the UCT SABus assembler ;
		=8046	----- ;
		=8047	;
		=8048	This module contains (in order) the following subroutines: ;
		=8049	;
		=8050	a) READNU - Main read number routine ;
		=8051	b) QNUM - Number ? ;
		=8052	;
		=8053	-----
		=8054	;
		=8055	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 140

LOC	OBJ	LINE	SOURCE STATEMENT
		=8056 ;	=====
		=8057 ;*	*
		=8058 ;*	Function : READ Number (From Text) *
		=8059 ;*	Inputs : Asm Pointer (POINT) = 1st digit of number. *
		=8060 ;*	Outputs : HL = Number. *
		=8061 ;*	Destroys : All *
		=8062 ;*	Stack Usage : Used 4 Bytes of Arithmetic stack + *
		=8063 ;*	4 Bytes of 8085 stack. *
		=8064 ;*	Calls : DINIT, CONLET, QNUM, DPUSH, APUSH, AMULT, *
		=8065 ;*	AADD, APOP and PERR. *
		=8066 ;*	*
		=8067 ;*	Description: This routine Reads a number from the *
		=8068 ;*	input text (Start pointer = POINT) and returns *
		=8069 ;*	it in the HL register Pair as a 16 bit value. *
		=8070 ;*	*
		=8071 ;*	The number formats supported are as follows: *
		=8072 ;*	*
		=8073 ;*	i) Decimal (Default base), eg 1234 *
		=8074 ;*	ii) Decimal (Format two), eg 1234D *
		=8075 ;*	iii) Octal eg 3344O *
		=8076 ;*	iv) Octal (Format two), eg 3455Q *
		=8077 ;*	v) Hexidecimal , eg 0A7B3H *
		=8078 ;*	vi) Binary , eg 1010B *
		=8079 ;*	vii) Binary (Format two), eg 0110Y *
		=8080 ;*	*
		=8081 ;	=====
		=8082 ;	
		=8083 READNU:	; Start of <READNU> routine.
		=8084	;
2950	CD4927	=8085 CALL DINIT	; Init digit stack.
2953	2A00E8	=8086 LHLD POINT	; HL = Current assembly pointer.
		=8087 NEXTD:	; Loop here after each digit.
2956	7E	=8088 MOV A,M	; Fetch next character.
2957	CDB213	=8089 CALL CONLET	; Convert to upper case (If it is a letter)
295A	CDFB29	=8090 CALL QNUM	; Is it a number ?
295D	FA6729	=8091 JM NOTNUM	; No then goto NOT NUMBER.
		=8092	;
2960	CD5727	=8093 CALL DPUSH	; It's a Number (push it onto the digit stack)
2963	23	=8094 INX H	; Inc pointer.
2964	C35629	=8095 JMP NEXTD	; Loop for next digit.
		=8096 ;	
		=8097 ;	-----
		=8098 ;	
		=8099 ;	All the digits have been pushed onto the digit stack.
		=8100 ;	;
		=8101 NOTNUM:	; Not Number.
2967	7E	=8102 MOV A,M	; Refetch the character that is not a number.
2968	CDB213	=8103 CALL CONLET	; Convert to Upper case.
296B	FE48	=8104 CPI 'H'	; Hex number ?
296D	CA9329	=8105 JZ HEXNUM	;
		=8106	;
2970	FE4F	=8107 CPI 'O'	; Octal number ?
2972	CA9829	=8108 JZ OCTAL	;
		=8109	;
2975	FE51	=8110 CPI 'Q'	; Octal format 2 ?
2977	CA9829	=8111 JZ OCTAL	;
		=8112	;
297A	FE59	=8113 CPI 'Y'	; Binary Format 2 ?
297C	CAB029	=8114 JZ BIN05	;
		=8115	; If there is no valid Terminator then check
		=8116	; the last digit (It could be 'B' or 'D')
		=8117	;
297F	2B	=8118 DCX H	; Dec pointer to last Digit.
2980	7E	=8119 MOV A,M	;
2981	CDB213	=8120 CALL CONLET	; Convert previous character to upper case.
2984	FE42	=8121 CPI 'B'	; Binary number ?
2986	CAA929	=8122 JZ BINARY	;
		=8123	;
2989	FE44	=8124 CPI 'D'	; Decimal ?
298B	CA9D29	=8125 JZ DGIVEN	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 141

LOC	OBJ	LINE	SOURCE STATEMENT
		=8126	
		=8127	DECNUM: ; Decimal Number given.
298E	3E0A	=8128	MVI A,10 ; Set base = 10.
2990	C3B229	=8129	JMP BASEF ;
		=8130	
		=8131	HEXNUM: ; Hex number given.
2993	3E10	=8132	MVI A,16 ; Set base = 16.
2995	C3B229	=8133	JMP BASEF ;
		=8134	
		=8135	OCTAL: ; Octal number given.
2998	3E08	=8136	MVI A,8 ; Set base = 8.
299A	C3B229	=8137	JMP BASEF ;
		=8138	
		=8139	DGIVEN: ; Decimal Number (with trailing 'D')
299D	3A4EE9	=8140	LDA TD ; Dec total Digit count.
29A0	3D	=8141	DCR A ;
29A1	324EE9	=8142	STA TD ;
29A4	3E0A	=8143	MVI A,10 ; Set base = 10.
29A6	C3B229	=8144	JMP BASEF ;
		=8145	
		=8146	BINARY: ; Binary number.
29A9	3A4EE9	=8147	LDA TD ; Dec total Digit count.
29AC	3D	=8148	DCR A ;
29AD	324EE9	=8149	STA TD ;
		=8150	BIN05: ;
29B0	3E02	=8151	MVI A,2 ; Set base = 2.
		=8152	
		=8153 ;	
		=8154 ;	
		=8155 ;	
		=8156 ;	The base of the number is in the A register and the memory variable
		=8157 ;	TD contains the number of digits in the number.
		=8158 ;	
		=8159	BASEF: ; Once Base found Calculate Total Value.
29B2	324DE9	=8160	STA BASE ; Store acc as base.
29B5	23	=8161	INX H ; Inc pointer to char after the last digit
		=8162	; In the string.
29B6	2200E8	=8163	SHLD POINT ; Save as asm Pointer.
29B9	3A4EE9	=8164	LDA TD ;
29BC	47	=8165	MOV B,A ; B Reg = Total digit count.
29BD	3A4DE9	=8166	LDA BASE ;
29C0	4F	=8167	MOV C,A ; C Reg = Base.
29C1	210000	=8168	LXI H,0000H ; Set Number = 0000.
29C4	CD2327	=8169	CALL APUSH ; Push Number onto Arithmetic stack.
29C7	2130E9	=8170	LXI H,DSTACK ; Get start adr of Digit stack.
		=8171	
		=8172	NEXTDI: ; Next Digit.
29CA	79	=8173	MOV A,C ; Get base.
29CB	3D	=8174	DCR A ;
29CC	BE	=8175	CMP M ; Compare digit with base.
29CD	DAF229	=8176	JC ERRDIG ; If digit > base then error.
		=8177	
29D0	5E	=8178	MOV E,M ;
29D1	1600	=8179	MVI D,0 ; Setup DE = Digit. (0. to 15.)
29D3	E5	=8180	PUSH H ; Save Digit stack pointer.
29D4	2600	=8181	MVI H,0 ;
29D6	69	=8182	MOV L,C ; Setup HL = Base.
29D7	CD2327	=8183	CALL APUSH ; Push base onto A_Stack.
29DA	CDE327	=8184	CALL AMULT ; Multiply Answer thus far with Base.
29DD	EB	=8185	XCHG ;
29DE	CD2327	=8186	CALL APUSH ; Push Digit onto the stack.
29E1	CDB627	=8187	CALL AADD ; Add digit to answer thus far.
29E4	E1	=8188	POP H ; Restore Digit stack pointer.
29E5	23	=8189	INX H ; Inc D_Stack pointer.
29E6	05	=8190	DCR B ;
29E7	C2CA29	=8191	JNZ NEXTDI ; If digits left <> 0 then loop to next digit.
		=8192	
29EA	CDFD26	=8193	CALL APOP ; Pop result of the A_Stack
		=8194	
29ED	3E00	=8195	MVI A,00H ; Set zero flag => Number read correctly.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 142

LOC	OBJ	LINE	SOURCE STATEMENT
29EF	C3F929	=8196	JMP RDNUME ;
		=8197	;
		=8198	ERRDIG: ; Error digit (i.e. Digit > base)
29F2	3E11	=8199	MVI A,17 ;
29F4	CD3E21	=8200	CALL PERR ; Gen error #17
29F7	3EFF	=8201	MVI A,OFFH ; Reset zero flag => Error in Number.
		=8202	;
		=8203	RDNUME: ; Read number Exit.
29F9	B7	=8204	ORA A ; Set flags.
29FA	C9	=8205	RET ; Return from <READNU>
		=8206 ;	
		=8207 ;=====	
		=8208 ;*	
		=8209 ;*	Function : QNUM (? Number) *
		=8210 ;*	Inputs : Acc = Digit to be checked. *
		=8211 ;*	Outputs : Acc = value of digit, if the character is a *
		=8212 ;*	valid Hex digit (i.e. 0 to 9 and A to F) *
		=8213 ;*	If the character is outside these ranges *
		=8214 ;*	then a negative value is returned. *
		=8215 ;*	Destroys : Acc and Flags. *
		=8216 ;*	Stack Usage : 2 bytes. *
		=8217 ;*	Calls : None. *
		=8218 ;*	
		=8219 ;*	Description: This routine is used by the READNU routine to *
		=8220 ;*	check that the characters is reads are valid digits. *
		=8221 ;*	
		=8222 ;=====	
		=8223 ;	
		=8224	QNUM: ; Start of Q number routine.
		=8225 ;	
		=8226 ;	Check that the character is in the range '0' to 'F'
		=8227 ;	
29FB	FE30	=8228	CPI '0' ;
29FD	FA192A	=8229	JM ERRN ; If Char < '0' then not a number.
		=8230	;
2A00	FE47	=8231	CPI 'F'+1 ;
2A02	F2192A	=8232	JP ERRN ; If char >= 'F'+1 then not a number.
		=8233 ;	
		=8234 ;	Now check to see if it is a Letter (A to F) or
		=8235 ;	a Limited number (0 to 9).
		=8236 ;	
		=8237 ;	
2A05	FE3A	=8238	CPI '9'+1 ;
2A07	FA142A	=8239	JM NUMBER ; If Character < '9'+1 then it's a Limited num.
		=8240	;
2A0A	FE41	=8241	CPI 'A' ;
2A0C	FA192A	=8242	JM ERRN ; If Character < 'A' then it's not a number.
		=8243	;
		=8244	LETTER: ; Else it's in the range 'A' to 'F'
2A0F	D637	=8245	SUI 'A'-10 ;
2A11	C31C2A	=8246	JMP QNUMEX ; => value = Character - 'A' + 10
		=8247	;
		=8248	NUMBER: ; Character in the range '0' to '9'
2A14	D630	=8249	SUI '0' ;
2A16	C31C2A	=8250	JMP QNUMEX ; => value = character - '0'
		=8251	;
		=8252	ERRN: ; Character not in range
2A19	3EFF	=8253	MVI A,OFFH ;
2A1B	B7	=8254	ORA A ; => value = -1
		=8255	QNUMEX: ;
2A1C	C9	=8256	RET ; Return from <QNUM>
		=8257 ;	
		=8258 ;	End of <READNU> module.
		8259	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 143

LOC	OBJ	LINE	SOURCE STATEMENT
		8260 ;	
		8261 ;-----	
		8262 ;	
		8263 ;	M E S S A G E S .
		8264 ;	
		8265 \$INCLUDE(ASMMSG.A85)	
		=8266 ;-----	
		=8267 ;	
		=8268 ;	Message data : Error messages 1 to 40.
		=8269 ;	
		=8270 ;-----	
		=8271 ;	
		=8272 MESSAG:	
2A1D	556E6465	=8273	DB 'Undefined OPCOD', 'E'+MSB
	66696E65		
	64204F50		
	434F44C5		
2A2D	556E6465	=8274	DB 'Undefined symbo', '1'+MSB
	66696E65		
	64207379		
	6D626FEC		
2A3D	53686966	=8275	DB 'Shift error, value > 1','5'+MSB
	74206572		
	726F722C		
	2076616C		
	7565203E		
	2031B5		
2A54	4D756C74	=8276	DB 'Multiply defined symbo', '1'+MSB
	69706C79		
	20646566		
	696E6564		
	2073796D		
	626FEC		
		=8277 M5:	
2A6B	44532065	=8278	DB 'DS expression must yield positive resul', 't'+MSB
	78707265		
	7373696F		
	6E206D75		
	73742079		
	69656C64		
	20706F73		
	69746976		
	65207265		
	73756CF4		
2A93	244C4953	=8279	DB '\$LISTFILE/\$OBJECTFILE -- Error in file nam', 'e'+MSB
	5446494C		
	452F244F		
	424A4543		
	5446494C		
	45202D2D		
	20457272		
	6F722069		
	6E206669		
	6C65206E		
	616DE5		
2ABE	53796D62	=8280	DB 'Symbol table ful', '1'+MSB
	6F6C2074		
	61626C65		
	2066756C		
	EC		
2ACF	244C4953	=8281	DB '\$LISTFILE/\$OBJECTFILE -- File could not be opene', 'd'+MSB
	5446494C		
	452F244F		
	424A4543		
	5446494C		
	45202D2D		
	2046696C		
	6520636F		
	756C6420		
	6E6F7420		

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 144

LOC	OBJ	LINE	SOURCE STATEMENT
	6265206F 70656E65 E4		
2B00	4E6F2065 6E642066 6F756EE4	=8282	DB 'No end foun', 'd'+MSB
		=8283 M10:	
2B0C	56616C75 65206F75 74206F66 2072616E 67652C20 62797465 2076616C 75652065 78706563 7465E4	=8284	DB 'Value out of range, byte value expecte', 'd'+MSB
2B33	45786365 7373206C 65667420 70617265 6E746865 7365F3	=8285	DB 'Excess left parenthese', 's'+MSB
2B4A	45786365 73732072 69676874 20706172 656E7468 657365F3	=8286	DB 'Excess right parenthese', 's'+MSB
2B62	46616374 6F722065 78706563 7465E4	=8287	DB 'Factor expecte', 'd'+MSB
2B71	496E636F 72726563 74207573 65206F66 206C6162 656C2066 69656C64 2C206C61 62656C73 206D7573 74206861 76652063 6F6C6F6E 73AE	=8288	DB 'Incorrect use of label field, labels must have colons', '.'+MSB
		=8289 M15:	
2BA7	4F706572 61746F72 20657870 65637465 E4	=8290	DB 'Operator expecte', 'd'+MSB
2BB8	52696768 74207175 6F746520 6E6F7420 666F756E E4	=8291	DB 'Right quote not foun', 'd'+MSB
2BCD	496E7661 6C696420 64696769 F4	=8292	DB 'Invalid digi', 't'+MSB
2BDA	41726974 686D6574 69632073 7461636B 20756E64 6572666C 6FF7	=8293	DB 'Arithmetic stack underflo', 'w'+MSB

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 145

LOC	OBJ	LINE	SOURCE STATEMENT
2BF4	4D6F7265 20746861 6E206F6E 65206368 61726163 74657220 696E2071 756F7465 F3	=8294	DB 'More than one character in quote', 's'+MSB
		=8295 M20:	
2C15	50686173 65206572 726FF2	=8296	DB 'Phase erro', 'r'+MSB
2C20	4572726F 7220696E 206F7065 72616E64 2C207265 67697374 65722065 78706563 7465E4	=8297	DB 'Error in operand, register expecte', 'd'+MSB
2C43	4572726F 7220696E 206F7065 72616E64 2C206172 6974686D 65746963 20726567 69737465 72207061 69722065 78706563 7465E4	=8298	DB 'Error in operand, arithmetic register pair expecte', 'd'+MSB
2C76	52535420 76616C75 65203E20 B7	=8299	DB 'RST value > ', '7'+MSB
2C83	4E6F206F 70657261 6E642066 6F756EE4	=8300	DB 'No operand foun', 'd'+MSB
		=8301 M25:	
2C93	326E6420 6F706572 616E6420 6E6F7420 666F756E E4	=8302	DB '2nd operand not foun', 'd'+MSB
2CA8	4572726F 7220696E 206F7065 72616E64 2C207374 61636B20 72656769 73746572 20706169 72206578 70656374 65E4	=8303	DB 'Error in operand, stack register pair expecte', 'd'+MSB
2CD6	4572726F 7220696E 206F7065 72616E64 2C206C69 6D697465 64207265 67697374	=8304	DB 'Error in operand, limited register pair expecte', 'd'+MSB

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 146

LOC	OBJ	LINE	SOURCE STATEMENT
	65722070		
	61697220		
	65787065		
	637465E4		
2D06	496C6C65	=8305	DB 'Illegal contro', 'l'+MSB
	67616C20		
	636F6E74		
	726FEC		
2D15	4E6F2065	=8306	DB 'No entry in DB directiv', 'e'+MSB
	6E747279		
	20696E20		
	44422064		
	69726563		
	746976E5		
		=8307 M30:	
2D2D	45786365	=8308	DB 'Excess characters in lin', 'e'+MSB
	73732063		
	68617261		
	63746572		
	7320696E		
	206C696E		
	E5		
2D46	24504147	=8309	DB '\$PAGELENGTH -- Syntax error or Length > 25', '5'+MSB
	454C454E		
	47544820		
	2D2D2053		
	796E7461		
	78206572		
	726F7220		
	6F72204C		
	656E6774		
	68203E20		
	3235B5		
2D71	24504147	=8310	DB '\$PAGELENGTH -- Page length < 5', '.'+MSB
	454C454E		
	47544820		
	2D2D2050		
	61676520		
	6C656E67		
	7468203C		
	2035AE		
2D90	44696769	=8311	DB 'Digit stack overflow, number to lon', 'g'+MSB
	74207374		
	61636B20		
	6F766572		
	666C6F77		
	2C206E75		
	6D626572		
	20746F20		
	6C6F6EE7		
2DB4	41726974	=8312	DB 'Arithmetic stack overflo', 'w'+MSB
	686D6574		
	69632073		
	7461636B		
	206F7665		
	72666C6F		
	F7		
		=8313 M35:	
2DCD	4F706572	=8314	DB 'Operator stack overflo', 'w'+MSB
	61746F72		
	20737461		
	636B206F		
	76657266		
	6C6FF7		
2DE4	4F6E6C79	=8315	DB 'Only ONE file can be opened', 'd'+MSB
	204F4E45		
	2066696C		
	65206361		
	6E206265		
	206F7065		

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 147

LOC	OBJ	LINE	SOURCE STATEMENT
	6E65E4		
2DFF	556E6578	=8316	DB 'Unexpected extra chars in lin', 'e'+MSB
	70656374		
	65642065		
	78747261		
	20636861		
	72732069		
	6E206C69		
	6EE5		
2E1D	496E7661	=8317	DB 'Invalid use of EQUate directiv', 'e'+MSB
	6C696420		
	75736520		
	6F662045		
	51556174		
	65206469		
	72656374		
	6976E5		
2E3C	496C6C65	=8318	DB 'Illegal character in symbo', 'l'+MSB
	67616C20		
	63686172		
	61637465		
	7220696E		
	2073796D		
	626FEC		
		=8319 M40:	
2E57	56616C75	=8320	DB 'Value out of range, unsigned byte value expecte', 'd'+MSB
	65206F75		
	74206F66		
	2072616E		
	67652C20		
	756E7369		
	676E6564		
	20627974		
	65207661		
	6C756520		
	65787065		
	637465E4		
		=8321 ;	
		=8322 ;	
		=8323 ;	
		=8324 ;	General assembler messages.
		=8325 ;	
		=8326 ;	
		=8327 ;	
2E87	45515520	=8328 EQUSTR: DB	'EQU '
		=8329 ;	
2E8B	53746172	=8330 STARTM: DB	'Start of pass 1' ,CR,LF+MSB
	74206F66		
	20706173		
	7320310D		
	8A		
2E9C	53796D62	=8331 STD MES: DB	'Symbol table',CR,LF+MSB
	6F6C2074		
	61626C65		
	0D8A		
2EAA	53746172	=8332 SP2MES: DB	'Start of pass 2' ,CR,LF+MSB
	74206F66		
	20706173		
	7320320D		
	8A		
		=8333 ;	
2EBB	072A2A20	=8334 FATM1: DB	BELL, '** FATAL EXIT - Code address out of range'
	46415441		
	4C204558		
	4954202D		
	20436F64		
	65206164		
	64726573		
	73206F75		
	74206F66		

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 148

LOC	OBJ	LINE	SOURCE STATEMENT
	2072616E		
	6765		
2EE5	A0	=8335	DB ' '+MSB
2EE6	20746FA0	=8336	FATM2: DB ' to', ' '+MSB
		=8337	;
2EEA	46617461	=8338	FATMSG: DB 'Fatal EXI', 'T'+MSB
	6C204558		
	49D4		
		=8339	;
2EF4	4B4954A0	=8340	KITMSG: DB 'KIT', ' '+MSB
2EF8	2F538D	=8341	TERMKIT: DB '/S',CR+MSB
2EFB	3A303030	=8342	TERMHR: DB ':00000001FF',CR+MSB
	30303030		
	3146468D		
		=8343	;
2F07	4E6F2066	=8344	NOFILM: DB 'No file in memor', 'y'+MSB
	696C6520		
	696E206D		
	656D6F72		
	F9		
2F18	4E6F2045	=8345	NOENDM: DB 'No END in file',CR,LF+MSB
	4E442069		
	6E206669		
	6C650D8A		
		=8346	;
2F28	072A2A2A	=8347	ERMES: DB BELL,'*** ERROR - ', ' '+MSB
	20455252		
	4F52202D		
	20A0		
		=8348	;
2F36	0D0A456E	=8349	ENDM1: DB CR,LF,'End of Assembly', ':'+MSB
	64206F66		
	20417373		
	656D626C		
	79BA		
2F48	20457272	=8350	ENDM2: DB ' Error(s) found', ' '+MSB
	6F722873		
	2920666F		
	756E64AE		
2F58	204E6F20	=8351	ENDM3: DB ' No Errors found', ' '+MSB
	4572726F		
	72732066		
	6F756E64		
	AE		
		=8352	;
		=8353	;
		=8354	;
2F69	55435420	=8355	HEAD1: DB 'UCT 8080/8085 Assembler, V1.0 PAGE', ' '+MSB
	38303830		
	2F383038		
	35204173		
	73656D62		
	6C65722C		
	2056312E		
	30205041		
	4745A0		
2F8C	2020204C	=8356	HEAD2: DB ' LOC OBJ LINE SOURCE STATEMEN', 'T'+MSB
	4F432020		
	2020204F		
	424A2020		
	20204C49		
	4E452020		
	20202020		
	2020534F		
	55524345		
	20535441		
	54454D45		
	4ED4		
		=8357	;
		=8358	;
			End of message module (MESSAGE.A85).

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 149

LOC	OBJ	LINE	SOURCE STATEMENT
		8359 ;	
2FB9		8360 ZZCODE EQU	\$-1 ; Address of last code byte
		8361 SEJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 150

LOC	OBJ	LINE	SOURCE STATEMENT
		8362 ;	
		8363 ;	
		8364 ;	
		8365 ;	DATA AREA.
		8366 ;	
		8367 \$INCLUDE(ASMDBLOCK.A85)	
		=8368 ;	
		=8369 ;	
		=8370 ;	Kit assembler data area.
		=8371 ;	
		=8372 ;	
		=8373 ;	
		=8374 ;	
E800		=8375	ORG DBLOCK ; Start of DATA area.
		=8376 ;	
		=8377 ;	
		=8378 ;	
		=8379 ;	A) Text pointers.
		=8380 ;	
E800		=8381 POINT: DS	WORD ; Current_asm_pointer.
E802		=8382 LPOINT: DS	WORD ; Start of line text pointer.
		=8383 ;	
		=8384 ;	
		=8385 ;	
		=8386 ;	B) Listing format variables.
		=8387 ;	
E804		=8388 SYMSOL: DS	BYTE ; Symbols on line (Used by the symbol
		=8389	; table dump routine.)
E805		=8390 NEWPAG: DS	BYTE ; Set By \$EJECT to indicate that a New page
		=8391	; is needed.
		=8392	;
E806		=8393 LINEN: DS	WORD ; Listing file, Line number.
E808		=8394 PAGNUM: DS	WORD ; Listing file, Page Number.
E80A		=8395 LPP: DS	BYTE ; Listing file, Lines_Per_Page.
E80B		=8396 PLINES: DS	BYTE ; Listing file, Lines on the page.
		=8397	;
E80C		=8398 MOREH: DS	BYTE ; More code to print on next listing line
		=8399	; flag (i.e. more the 4 bytes of code
		=8400	; from a given source line)
		=8401	;
E80D		=8402 BRIEFF: DS	BYTE ; Brief list flag - when set to true by the
		=8403	; \$BRIEF control only one line of output
		=8404	; is produced even if there are more than
		=8405	; 4 bytes of object code produced.
		=8406	;
E80E		=8407 GCFLAG: DS	BYTE ; Generate code flag.
		=8408	; True = print hex for code between Last_PC
		=8409	; and PC (Used for DS directives)
		=8410	;
E80F		=8411 EQUFLG: DS	BYTE ; EQU_line flag.
		=8412	; The listing line is printed with the value
		=8413	; of the symbol printed in the adr field and
		=8414	; is therefore printed by a separate routine.
		=8415 ;	
		=8416 ;	
		=8417 ;	
		=8418 ;	C) Error message variables.
		=8419 ;	
E810		=8420 ERRORP: DS	WORD ; Error buffer pointer.
E812		=8421 ERRORS: DS	WORD ; Total error count for entire assembly.
E814		=8422 LERRS: DS	BYTE ; Counter for Errors on each line.
E815		=8423 FATALF: DS	BYTE ; Fatal error flag.
E816		=8424 ERRBUF: DS	10*BYTE ; Buffer to hold Error numbers.
		=8425 ;	
		=8426 ;	
		=8427 ;	
		=8428 ;	D) Data used by PDF List file routines.
		=8429 ;	
E820		=8430 PDPFLG: DS	BYTE ; Output flag:
		=8431	; True => Listing to PDP.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM PAGE 151

LOC	OBJ	LINE	SOURCE STATEMENT
		=8432	
E821		=8433 FNAME: DS	30*BYTE ; False => Listing = Console/Null.
E83F		=8434 LINEB: DS	MAXBL ; Output file name buffer.
E8C3		=8435 LINEBP: DS	WORD ; Output line buffer.
E8C5		=8436 LINEBL: DS	BYTE ; Pointer in output buffer.
E8C6		=8437 PLUSOL: DS	BYTE ; Line_buffer_len (# of chars in line_buffer)
		=8438	; Plus Signs on the terminal. (used when
		=8439 ;	; Sending a list file to the PDP)
		=8440 ;	-----
		=8441 ;	
		=8442 ;	E) Data used by PDP object file routines.
		=8443 ;	
E8C7		=8444 OBJFLG: DS	BYTE ; Object output flag:
		=8445	; TRUE => Object to PDP.
		=8446	; FALSE => Object to RAM.
		=8447	;
E8C8		=8448 OBJBUF: DS	OBJLEN ; Buffer to store object bytes before
		=8449	; they are saved on the PDP.
		=8450	;
E8D8		=8451 OBJBP: DS	WORD ; Object buffer pointer. (to next free byte
		=8452	; in the buffer.)
		=8453	;
E8DA		=8454 OBJBA: DS	WORD ; Adr of the 1st byte in the object buffer.
		=8455	;
E8DC		=8456 OBJNA: DS	WORD ; Next adr expected by the Object save routine.
		=8457	;
E8DE		=8458 OBJC: DS	BYTE ; Object byte counter (i.e. The number of
		=8459	; bytes in the object buffer)
		=8460	;
E8DF		=8461 CODBUF: DS	3 * BYTE ; Code buffer.
E8E2		=8462 CHKSUM: DS	BYTE ; Temp check sum.
		=8463 ;	
		=8464 ;	-----
		=8465 ;	
		=8466 ;	F) Data used by symbol table routines.
		=8467 ;	
E8E3		=8468 LABELB: DS	LABBL ; 6 Characters for Label field.
E8E9		=8469	DS WORD ; 1 Word for addr field.
E8EB		=8470 TEMPLB: DS	LABBL ; 6 Characters for temp label field.
		=8471	;
E8F1		=8472 HASHV: DS	BYTE ; Hash value.
E8F2		=8473 HVINC: DS	BYTE ; Amount to inc hash value (used by quad
		=8474	; search method).
E8F3		=8475 PASDEF: DS	BYTE ; Pass defined. (1 or 2).
		=8476 ;	
		=8477 ;	-----
		=8478 ;	
		=8479 ;	G) Data area used for rexp and arithmetic routines .
		=8480 ;	
		=8481 ;	i) Stack areas.
		=8482 ;	
E8F4		=8483 ASTACK: DS	ASTKL ; Allow max of 15 words on the stack
E912		=8484 OSTACK: DS	OSTKL ; Allow max of 30 opers on astack
E930		=8485 DSTACK: DS	DSTKL ; Allow for 16 binary bits + terminator.
		=8486 ;	
		=8487 ;	ii) Stack pointers.
		=8488 ;	
E945		=8489 ASP: DS	WORD ; Arithmetic stack pointer .
E947		=8490 OSP: DS	WORD ; Operator stack pointer .
E949		=8491 DSP: DS	WORD ; Digit stack pointer.
		=8492 ;	
		=8493 ;	iii) General use.
		=8494 ;	
E94B		=8495 MULT: DS	WORD ; Temp word used by amult routine.
E94D		=8496 BASE: DS	BYTE ; Base of number read in.
E94E		=8497 TD: DS	BYTE ; Number digits in the number read in.
E94F		=8498 SIGN: DS	BYTE ; Sign of number used by mult.
		=8499	;
E950		=8500 ENTS: DS	BYTE ; Arithmetic entries, (Number of word on the
		=8501	; Arithmetic stack)

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM

PAGE 152

LOC	OBJ	LINE	SOURCE STATEMENT
E951		=8502	OENTS: DS BYTE ; Operator entries (Number of entries on the
		=8503	; operator stack)
E952		=8504	NEXTID: DS BYTE ; Next_identifier expected.
		=8505	; label or operator ?
		=8506 ;	
		=8507 ;	
		=8508 ;	
		=8509 ;	H) Data used by opcode lookup routines.
		=8510 ;	
		=8511 ;	i) Opcode buffers. (Note len(Control_buffer) > len(Opcode_buffer))
		=8512 ;	
E953		=8513	OPBUF: DS OPBUFL ; Characters for Opcode field.
E959		=8514	DS CONBL-OPBUFL ; Add extra for control buffer.
		=8515 ;	
		=8516 ;	ii) Packet returned byte opcode lookup routine.
		=8517 ;	
E95D		=8518	BYTES: DS BYTE ; Number of bytes in instruction.
E95E		=8519	OPERS: DS BYTE ; Operands needed by opcode.
E95F		=8520	OPCODE: DS BYTE ; Opcode byte (Null operands).
		=8521 ;	
		=8522 ;	iii) Binary search variables.
		=8523 ;	
E960		=8524	LOWLIM: DS BYTE ; Lower limit for binary search.
E961		=8525	UPLIM: DS BYTE ; Upper limit for binary search.
E962		=8526	CENT: DS BYTE ; Centre of range.
E963		=8527	SADR: DS WORD ; Start adr of the entry being checked.
		=8528 ;	
		=8529 ;	
		=8530 ;	
		=8531 ;	I) Assembly counters. (Program counters.)
		=8532 ;	
E965		=8533	PC: DS WORD ; Assembly program counter.
E967		=8534	LASTPC: DS WORD ; Start of line program counter.
		=8535 ;	
		=8536 ;	
		=8537 ;	
		=8538 ;	J) Flags.
		=8539 ;	
E969		=8540	PFLAG: DS BYTE ; Print flag (Set/Reset by \$LIST/\$NOLIST)
E96A		=8541	PAFLAG: DS BYTE ; Print adr Flag.
E96B		=8542	PCDEF: DS BYTE ; Program counter defined flag.
E96C		=8543	SYMFLG: DS BYTE ; Symbol table print Flag.
E96D		=8544	PZRFLG: DS BYTE ; Print zero flag (Used by PRINTD)
E96E		=8545	CONFLG: DS BYTE ; Set by control to indicate that the current
		=8546	; lines is a control line.
		=8547 ;	
		=8548 ;	
		=8549 ;	
		=8550 ;	K) General byte variables.
		=8551 ;	
E96F		=8552	QTABSL: DS BYTE ; Qtable String length.
E970		=8553	PASSN: DS BYTE ; Assembly pass number, 1 or 2.
		=8554 ;	
E971		=8555	DS STACKL
		=8556	NEWSP: ; New stack
		=8557 ;	
		=8558 ;	End of Data area (DBLOCK.A85).
		8559	
E9FC		8560	ZZDATA EQU \$-1 ; Address of last data byte
		8561	
1000		8562	END START ; The program stops HERE

0 error(s) detected

User symbols

AACK 0006 A	AADD 27B6 A	ABS 27C6 A	ADDP 1300 A	ADDSYM 002B A	ADIV 27D4 A	ADRINS 130F A
ADRSET 10F3 A	AINIT 26E2 A	AMULT 27E3 A	ANDC 0003 A	ANDROU 2810 A	ANEG 2821 A	APOP 26FD A
APUSH 2723 A	APUSHE 2740 A	APUSHX 2745 A	ARPS 22F3 A	ASCSub 001A A	ASP E945 A	ASTACK E8F4 A

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT 8085 Assembler

08:53:34 13-APR-88

ASM PAGE 153

LOC	OBJ	LINE	SOURCE STATEMENT
ASTKL	001E A	ASUB 2833 A	BASE E94D A BASEAD 222C A BASEF 29B2 A BBUF 1374 A BBUFLP 1375 A
BCC	1D0D A	BCRP 2313 A	BELL 0007 A BIN05 29B0 A BINARY 29A9 A BITSPB 0008 A BITSPW 0010 A
BREAK	1D1A A	BRIEFF E80D A	BRIEFR 174F A BRK05 1D26 A BSPACE 0008 A BYTE 0001 A BYTES E95D A
CAN	0018 A	CENT E962 A	CHARFO 250D A CHKBLK 144E A CHKSUM E8E2 A CLASH 1866 A CLOSE 1D56 A
CLOSE5	1D45 A	CLOSE7 1D51 A	CLOSEF 1D2F A CMPLAB 180A A CMPLX 1820 A CMPLOP 19F7 A CMPOP 19F2 A
CMPOPE	1A0C A	CNXTCH 180F A	CODBUF E8DF A CODE 2389 A CODEEX 2405 A COMLIN 122A A CONBL 000A A
CONCOP	137E A	CONFLG E96E A	CONHEX 1D79 A CONJT 17AD A CONLET 13B2 A CONLEX 13BE A CONTAB 175B A
CONTEX	16AD A	CONTIN 1A50 A	CONTRO 1675 A CONVHV 1821 A CONVOP 2219 A CONVPO 1A0D A COPY 1383 A
COPY05	1385 A	COPYEX 13AC A	COPYL 13BF A CPYEX 13E9 A CR 000D A CRLF 1D83 A CSHIFT 2848 A
CSHRET	285A A	CTRLMA 001F A	CTRLQ 0011 A CTRLS 0013 A CTRLZ 001A A D16 0001 A D16ROU 223E A
D8	0002 A	D805 225A A	D810 2261 A D8ROU 2247 A DBEXIT 1578 A DBEXP 14E3 A DBGENB 14BA A
DBLOCK	E800 A	DBROU 14D1 A	DBSTR 1517 A DBSTRE 156B A DBVALC 14FC A DBVALO 14FB A DC1 0011 A
DC2	0012 A	DC3 0013 A	DC4 0014 A DCPM 1323 A DCP05 132E A DECNUM 298E A DEFLPP 003C A
DEFFF	FF00 A	DEL 007F A	DERP 2318 A DIGIVEN 299D A DIFPAS 162E A DINIT 2749 A DIVC 0009 A
DIVIDE	26D9 A	DIVLP 2870 A	DIVMOD 285C A DIVSYM 002F A DLE 0010 A DONTPA 1FE5 A DPUSH 2757 A
DPUSHE	2772 A	DPUSHX 2777 A	DREG 0004 A DREGRO 2286 A DSEXPO 158D A DSP E949 A DSROU 1579 A
DSTACK	E930 A	DSTKL 0015 A	DSVPOS 159A A DWEXIT 15F9 A DWGENW 15A8 A DWROU 15CB A EGETO 2224 A
EJECTR	16CD A	EM 0019 A	EMPTYQ 24FB A ENDEM 21B6 A ENDENT 143F A ENDEX 248F A ENDEXP 2484 A
ENDHEX	2082 A	ENDLIN 1131 A	ENDLP2 1245 A ENDM1 2F36 A ENDM2 2F48 A ENDM3 2F58 A ENDOFT 1997 A
ENDOPR	25F5 A	ENDPH 203E A	ENDPH1 2040 A ENDPHO 1CCF A ENDROU 15FA A ENDST 1FC4 A ENDSTR 154D A
ENDTAB	1444 A	ENQ 0005 A	ENTS E950 A EOPMAR 001A A EOT 0004 A EPAS2 1065 A EPOINT EFPE A
EQUEND	1654 A	EQU 1124 A	EQUFLG E80F A EQULIN 20C2 A EQUROU 1602 A EQUSTR 2E87 A ERNES 2F28 A
EROPAE	22EE A	EROPRE 2281 A	EROPSE 22C0 A ERRBUF E816 A ERRCOD 23AC A ERRDIG 29F2 A ERREX 24B7 A
ERREX1	24CA A	ERRN 2A19 A	ERRNOP 12E7 A ERRNSO 12EC A ERRORP E810 A ERRORS E812 A ESC 001B A
ETABLE	00FF A	ETB 0017 A	ETX 0003 A EXAPOP 2720 A EXASM 107E A EXDWG 15C5 A EXEXP 24D5 A
EXOPER	2623 A	EXOPOP 2789 A	EXPLIN 2085 A EXRP 261A A EXRULF 135A A FALSE FF00 A FATALE 2120 A
FATALF	E815 A	FATALN 0063 A	FATM1 2EBB A FATM2 2EE6 A FATMSG 2EEA A FETCH 182E A FETCHE 186E A
FETCHN	17C1 A	FETCHV 2551 A	FEKASM 1081 A FF 000C A FPN05 17D0 A FFNEOF 17E7 A FPNERR 17E2 A
FFNRET	17EE A	FILEOU 1D8E A	FIOEX 1DC6 A FNAME E821 A FORIL 1952 A FPHEX 1DCA A FREE 18D1 A
FS	001C A	GCFLAG E80E A	GENIT 1535 A GETNSC 13EF A GETOPR 2212 A GETOPS 1295 A GOPRET 222A A
GS	001D A	HASH 1870 A	HASHV E8F1 A HEAD1 2F69 A HEAD2 2F8C A HEADER 1DE3 A HEXNUM 2993 A
HIGHC	000E A	HIGHRO 284A A	HT 0009 A HVINC E8F2 A INSER 18F8 A INSERT 18A9 A INSR 18FA A
K	0400 A	KITACK 0040 A	KITIN F013 A KITMSG 2EF4 A KITOUT F01C A KITTES F00A A LABBL 0006 A
LABELB	E8E3 A	LABEL 00FF A	LASTPC E967 A LERRS E814 A LETNOK 1478 A LETOK 1473 A LETTER 2A0F A
LF	000A A	LINEB E83F A	LINEBL E8C5 A LINEBP E8C3 A LINEN E806 A LISTRO 16AE A LLETOK 149A A
LOWC	000D A	LOWLIN E960 A	LOWROU 28AE A LPOINT E802 A LPP E80A A LSTFER 16FA A LSTPEX 16FF A
LSTFIL	16DF A	LSTREX 16C6 A	M10 2B0C A M15 2BA7 A M20 2C15 A M25 2C93 A M30 2D2D A
M35	2DCD A	M40 2E57 A	M5 2A6B A M8085 00FF A MASKIN 231A A MAXBL 0084 A MAXCOD 0FFF A
MAXPOL	0032 A	MESSAG 2A1D A	MINCOD 0000 A MINUS 26D4 A MINUSC 0006 A MODC 000A A MODROU 28B7 A
MODUPL	1A4A A	MONDAT EF00 A	MOREH E80C A MORETC 138F A MORTC 13CC A MOVLOP 18D7 A MSB 0080 A
MULDEF	18F3 A	MULSYM 002A A	MULT E94B A MULTC 0008 A MULTSF 26DE A NAK 0015 A NEGATE 25C2 A
NEGC	0007 A	NERRIL 213D A	NEVERP 0080 A NEWPAG E805 A NEWSF E9FD A NEXTBY 191D A NEXTD 2956 A
NEXTDB	14DE A	NEXTDI 29CA A	NEXTHV 18FC A NEXTI 1982 A NEXTID E952 A NEXTOP 1A26 A NEXTPL 1E10 A
NEXTST	216D A	NLERS 20A7 A	NLINE1 1021 A NLINE2 1040 A NLSTRO 16C7 A NOADD 27FD A NOBRRO 1755 A
NOCON1	10C9 A	NOCON2 11AD A	NODEF 164C A NOENDM 2F18 A NOERRS 21B0 A NOEXP 15F4 A NOEXPF 1573 A
NOFILM	2F07 A	NOFOUT 1E0B A	NOLABP 10FA A NOMORE 2035 A NOOP 0000 A NOOPER 12C9 A NOOPNF 1D78 A
NOOPOK	12DB A	NOPC 0000 A	NOPLUS 1F3F A NOPRT 2187 A NOREX 184A A NORQ 2500 A NOSQ 1409 A
NOTBOP	1E3C A	NOTC 0004 A	NOTCON 16AB A NOTE 1438 A NOTFEX 219C A NOTLL 1495 A NOTLN 254A A
NOTLN5	2528 A	NOTLP 25DE A	NOTNP 1E2E A NOTNUM 2967 A NOTREG 2377 A NOTROU 28BE A NOTSOP 2683 A
NOTT1	1CC5 A	NOTVSC 256A A	NPCVAL 253E A NPRINT 20C5 A NRQ 1563 A NSKPTS 136A A NSOP 267F A
NSYNRO	16D9 A	NUL 0000 A	NUMBER 2A14 A NUMER 1D80 A NUMLIN 2338 A NXTBIT 27F0 A NXTCEX 140B A
NXTCIE	2424 A	NXTCRC 187A A	NXTHVE 1915 A NXTPHC 1CB4 A NXTPHO 1CAF A NXTPL 1E46 A NXTREG 2368 A
OBJBA	E8DA A	OBJBNF 2403 A	OBJBP E8D8 A OBJBUF E8C8 A OBJC E8DE A OBJCON 23E4 A OBJFER 1713 A
OBJFEX	1718 A	OBJFLG E8C7 A	OBJFRO 1700 A OBJINI 17F0 A OBJLEN 0010 A OBJNA E8DC A OBLOP 1F78 A
OCODBL	2411 A	OCODBU 2407 A	OCODEF 1A62 A OCTAL 2998 A OENTS E951 A OFFMSB 007F A OKASMF 101E A
OKGETV	252C A	OKOPER 25CA A	OPBUF E953 A OPBUFL 0006 A OPCODE E95F A OPEN05 1E6C A OPEN10 1E7C A
OPENE1	1E95 A	OPENE2 1E9A A	OPENE3 1E9C A OPENFE 1EA1 A OPENFI 1EA4 A OPER 0000 A OPERET 25FD A
OPERF	257D A	OPERT 25E9 A	OPERS E95E A OPOP 2779 A OPTAB 1A7B A OPTABL 0051 A OPUSH 2792 A
OPUSHE	27AE A	OPUSHX 27B3 A	ORC 0001 A ORGERR 166F A ORGREX 1674 A ORGROU 1655 A ORROU 28CB A
OSP	E947 A	OSTACK E912 A	OSTKL 001E A OUTD05 21C7 A OUTD10 21CF A OUTD15 21DB A OUTD20 21F4 A
P1DEF	18EC A	P1DEFN 185E A	P1INIT 1085 A P1LINE 10AF A P1TOP 1118 A P2ADRD 1256 A P2ADRS 11EE A
P2ADRU	1250 A	P2ENDL 1242 A	P2EPRT 123F A P2EQU 11F4 A P2INIT 1135 A P2LABE 127F A P2LABL 1246 A
P2LABR	11C8 A	P2LBER 127C A	P2LINE 1173 A P2MULD 127A A P2OERR 12EE A P2OPF 1218 A P2OPNF 121E A
P2PHNE	123D A	P2PHNF 1232 A	P2POP 11FF A P2POPE 12F9 A P2POPR 1280 A P2PDR 1F04 A PADREX 1FEA A
PADWB	2200 A	PAERS 2103 A	PAFLAG E96A A PAGLE1 1744 A PAGLE2 1749 A PAGLE3 174B A PAGLEN 1719 A
PAGLEX	174E A	PAGNUM E808 A	PASDEF E8F3 A PASSN E970 A PDEF E965 A PCDEF E96B A PCNDEF 1321 A
PCODEL	1FEB A	PDPFLG E820 A	PDPIN F016 A PDPOUT F01F A PDP05 1EA4 A PDP10 1EB3 A PDPSTR 1EA3 A
PDPFES	F00D A	PEQL 2086 A	PERR 213E A PERRNC 2171 A PERROR 2160 A PFLAG E969 A PHL 1EBD A
PHOFF	1CDD A	PHOTAB 1CF1 A	PHSERR 1644 A PLABEX 19D7 A PLINE 2098 A PLINES E80B A PLUS 26CF A
PLUSC	0005 A	PLUSOL E8C6 A	PNERRS 218B A PNHEX 2009 A PNLOH 2060 A PNXTER 210F A PNXTHD 2067 A

PAGE 154

G-162

APPENDIX H

DEB LISTING

The following pages contain an index of the symbols used in the debugger source listing. In order to keep the index reasonably short, the local symbols, those defined and referenced within a subroutine, are not listed.

As well as the symbol name and page reference, a symbol type is also given. The possible symbol types are:

- a) CONSTANT (in this case a value is also given)
- b) SUBROUTINE
- c) BYTE ARRAY (in this case the length is also given)
- d) WORD ARRAY (in this case the length, in bytes, is also given)
- e) STRUCTURE
- f) TABLE
- g) MESSAGE

Some of the symbol types may be preceded by the word ABSOLUTE. This simply means that they are stored at fixed addresses in the memory map and storage is not dynamically allocated in the debugger data area.

NOTE: The page reference given in the index below is the page on which the symbol is defined.

<u>SYMBOL</u>	<u>DESCRIPTION</u>	<u>PAGE REFERENCE</u>
ACK	CONSTANT, VALUE = 006H.....	H-17
ADDBC	SUBROUTINE.....	H-118
ADDLEN	SUBROUTINE.....	H-172
ADJUST	SUBROUTINE.....	H-112
ASCBUF	BYTE ARRAY, LENGTH = 16.....	H-195
ASCII	CONSTANT, VALUE = 01111111B.....	H-16
ASUB	CONSTANT, VALUE = 01AH.....	H-17
BCMP	SUBROUTINE.....	H-77
BELL	CONSTANT, VALUE = 007H.....	H-17
BITSPB	CONSTANT, VALUE = 8.....	H-16
BITSPW	CONSTANT, VALUE = 16.....	H-16
BREAK	SUBROUTINE.....	H-69

BS	CONSTANT, VALUE = 008H.....	H-17
BYTE	CONSTANT, VALUE = 1.....	H-16
BYTECOUNT	BYTE VARIABLE.....	H-196
BYTEDUMP	SUBROUTINE.....	H-152
BYTEROU	SUBROUTINE.....	H-154
BYTEVAR	CONSTANT, VALUE = 00000000B.....	H-16
CAN	CONSTANT, VALUE = 018H.....	H-17
CHKKEY	SUBROUTINE.....	H-118
CHKMRD	SUBROUTINE.....	H-23
CHKMWR	SUBROUTINE.....	H-24
CHKPRD	SUBROUTINE.....	H-24
CHKPWR	SUBROUTINE.....	H-25
CI	CONSTANT, VALUE = 0F013H.....	H-64
CITEST	CONSTANT, VALUE = 0F00AH.....	H-64
CLPTR	WORD VARIABLE.....	H-195
CLRLINE	CONSTANT, VALUE = 'K'.....	H-18
CLRMSG	MESSAGE.....	H-139
CLRS	CONSTANT, VALUE = 'J'.....	H-17
CLRSCN	MESSAGE.....	H-109
CMDERM	MESSAGE.....	H-70
CMDERR	SUBROUTINE.....	H-69
CMDLINE	BYTE ARRAY, LENGTH = 80.....	H-195
CMDTAB	TABLE.....	H-82
CO	CONSTANT, VALUE = 0F01CH.....	H-64
CODATA	CONSTANT, VALUE = 0.....	H-64
COMMA	CONSTANT, VALUE = ','.....	H-17
CONHEX	SUBROUTINE.....	H-69
COPYBYTES	SUBROUTINE.....	H-119
COPYENTRY	SUBROUTINE.....	H-130
COPYSYM	SUBROUTINE.....	H-87
COSTAT	CONSTANT, VALUE = 1.....	H-64
CR	CONSTANT, VALUE = 00DH.....	H-17
CRLF	SUBROUTINE.....	H-70
CSTATUS	BYTE VARIABLE.....	H-196
CTRLMASK	CONSTANT, VALUE = 00011111B.....	H-16
CTRLQ	CONSTANT, VALUE = CTRLMASK AND 'Q'.....	H-17
CTRLS	CONSTANT, VALUE = CTRLMASK AND 'S'.....	H-17
CTRLZ	CONSTANT, VALUE = CTRLMASK AND 'Z'.....	H-17
D16FETCH	SUBROUTINE.....	H-38
D8FETCH	SUBROUTINE.....	H-38
DATA16	CONSTANT, VALUE = 4.....	H-26
DATA8	CONSTANT, VALUE = 3.....	H-26
DC1	CONSTANT, VALUE = 011H.....	H-17
DC2	CONSTANT, VALUE = 012H.....	H-17
DC3	CONSTANT, VALUE = 013H.....	H-17
DC4	CONSTANT, VALUE = 014H.....	H-17
DCMP	SUBROUTINE.....	H-77
DEB	MAIN PROGRAM.....	H-20
DEBINIT	SUBROUTINE.....	H-21
DECIDE	SUBROUTINE.....	H-39

DECLINES	SUBROUTINE.....	H-138
DEFCOUNT	CONSTANT, VALUE = 1.....	H-18
DEFROU	SUBROUTINE.....	H-158
DEL	CONSTANT, VALUE = 07FH.....	H-17
DIGBUF	BYTE ARRAY, LENGTH = 16.....	H-195
DIGCOUNT	BYTE VARIABLE.....	H-195
DIGIPTR	WORD VARIABLE.....	H-195
DIGOPTR	WORD VARIABLE.....	H-195
DIN	SUBROUTINE.....	H-88
DINIT	SUBROUTINE.....	H-88
DISASM	SUBROUTINE.....	H-138
DISLINE	SUBROUTINE.....	H-139
DISNEXT	SUBROUTINE.....	H-140
DISPC	WORD VARIABLE.....	H-196
DISPREF	SUBROUTINE.....	H-174
DISROU	SUBROUTINE.....	H-141
DISTAB	TABLE.....	H-32
DLE	CONSTANT, VALUE = 010H.....	H-17
DOPOP	SUBROUTINE.....	H-39
DOPUSH	SUBROUTINE.....	H-40
DOUT	SUBROUTINE.....	H-88
DREG	CONSTANT, VALUE = 2.....	H-26
DREGMASK	CONSTANT, VALUE = 00111000B.....	H-16
DUMMYROU	SUBROUTINE.....	H-80
DYNIO	BYTE ARRAY, LENGTH = 3.....	H-197
EM	CONSTANT, VALUE = 019H.....	H-17
EMCALL	CONSTANT, VALUE = EMMEM+1.....	H-31
EMDCRM	CONSTANT, VALUE = EMCALL+1.....	H-31
EMDI	CONSTANT, VALUE = EMDCRM+1.....	H-31
EMDRM	CONSTANT, VALUE = EMLDAX+1.....	H-31
EMEI	CONSTANT, VALUE = EMDI+1.....	H-31
EMERR	CONSTANT, VALUE = EMLHLD+1.....	H-32
EMHLT	CONSTANT, VALUE = EMEI+1.....	H-31
EMIN	CONSTANT, VALUE = EMHLT+1.....	H-31
EMINFO	BYTE VARIABLE.....	H-196
EMINRM	CONSTANT, VALUE = EMIN+1.....	H-31
EMJMP	CONSTANT, VALUE = EMINRM+1.....	H-31
EMLDA	CONSTANT, VALUE = EMJMP+1.....	H-31
EMLDAX	CONSTANT, VALUE = EMLDA+1.....	H-31
EMLHLD	CONSTANT, VALUE = EMXTHL+1.....	H-32
EMMASK	CONSTANT, VALUE = 00011111B.....	H-16
EMMEM	CONSTANT, VALUE = EXECUTE+1.....	H-31
EMMSR	CONSTANT, VALUE = EMDRM+1.....	H-31
EMMVIM	CONSTANT, VALUE = EMMSR+1.....	H-32
EMOUT	CONSTANT, VALUE = EMMVIM+1.....	H-32
EMPCHL	CONSTANT, VALUE = EMOUT+1.....	H-32
EMPOP	CONSTANT, VALUE = EMPCHL+1.....	H-32
EMPUSH	CONSTANT, VALUE = EMPOP+1.....	H-32
EMRET	CONSTANT, VALUE = EMRIM+1.....	H-32
EMRIM	CONSTANT, VALUE = EMPUSH+1.....	H-32

EMRST	CONSTANT, VALUE = EMRET+1.....	H-32
EMSHLD	CONSTANT, VALUE = EMSPHL+1.....	H-32
EMSIM	CONSTANT, VALUE = EMRST+1.....	H-32
EMSPHL	CONSTANT, VALUE = EMSIM+1.....	H-32
EMSTA	CONSTANT, VALUE = EMSHLD+1.....	H-32
EMSTAX	CONSTANT, VALUE = EMSTA+1.....	H-32
EMUTAB	TABLE.....	H-31
EMXTHL	CONSTANT, VALUE = EMSTAX+1.....	H-32
ENABINT	BYTE VARIABLE.....	H-196
ENDMARK	BYTE VARIABLE.....	H-195
ENDMSG	CONSTANT, VALUE = NUL.....	H-17
ENQ	CONSTANT, VALUE = 005H.....	H-17
EOFMARK	CONSTANT, VALUE = CTRLZ.....	H-17
EOL	CONSTANT, VALUE = 0.....	H-17
EOT	CONSTANT, VALUE = 004H.....	H-17
EQUALS	CONSTANT, VALUE = 4.....	H-17
ERCALL	SUBROUTINE.....	H-44
ERDCRM	SUBROUTINE.....	H-45
ERDI	SUBROUTINE.....	H-45
ERDMMSG	MESSAGE.....	H-98
ERDRM	SUBROUTINE.....	H-46
EREI	SUBROUTINE.....	H-47
ERERR	SUBROUTINE.....	H-47
ERHLT	SUBROUTINE.....	H-47
ERIN	SUBROUTINE.....	H-48
ERINRM	SUBROUTINE.....	H-49
ERJMP	SUBROUTINE.....	H-50
ERLDA	SUBROUTINE.....	H-50
ERLDAX	SUBROUTINE.....	H-51
ERLHLD	SUBROUTINE.....	H-52
ERMBYTE	MESSAGE.....	H-154
ERMDEF	MESSAGE.....	H-158
ERMDIS	MESSAGE.....	H-141
ERMEM	SUBROUTINE.....	H-53
ERMGO	MESSAGE.....	H-165
ERMHELP	MESSAGE.....	H-108
ERMMAP	MESSAGE.....	H-114
ERMPORT	MESSAGE.....	H-169
ERMPRINT	MESSAGE.....	H-175
ERMREG	MESSAGE.....	H-134
ERMRESET	MESSAGE.....	H-127
ERMSR	SUBROUTINE.....	H-54
ERMSTEP	MESSAGE.....	H-185
ERMVIM	SUBROUTINE.....	H-54
EROUT	SUBROUTINE.....	H-55
ERPCHL	SUBROUTINE.....	H-56
ERPOP	SUBROUTINE.....	H-56
ERPUSH	SUBROUTINE.....	H-57
ERRET	SUBROUTINE.....	H-57
ERRFLAG	BYTE VARIABLE.....	H-195

ERRIM	SUBROUTINE.....	H-58
ERROR	SUBROUTINE.....	H-71
ERRST	SUBROUTINE.....	H-58
ERSHLD	SUBROUTINE.....	H-59
ERSIM	SUBROUTINE.....	H-60
ERSPHL	SUBROUTINE.....	H-60
ERSTA	SUBROUTINE.....	H-61
ERSTAX	SUBROUTINE.....	H-61
ERXTHL	SUBROUTINE.....	H-62
ESC	CONSTANT, VALUE = 01BH.....	H-17
ESCENTERED	CONSTANT, VALUE = 5.....	H-18
ESMSG	MESSAGE.....	H-74
ETB	CONSTANT, VALUE = 017H.....	H-17
ETX	CONSTANT, VALUE = 003H.....	H-16
EX16BMOD	CONSTANT, VALUE = 9.....	H-19
EX16BRD	CONSTANT, VALUE = 4.....	H-19
EX16BWR	CONSTANT, VALUE = 5.....	H-19
EX8BIN	CONSTANT, VALUE = 6.....	H-19
EX8BMOD	CONSTANT, VALUE = 8.....	H-19
EX8BOU	CONSTANT, VALUE = 7.....	H-19
EX8BRD	CONSTANT, VALUE = 2.....	H-19
EX8BWR	CONSTANT, VALUE = 3.....	H-19
EXEC	SUBROUTINE.....	H-120
EXECMD	SUBROUTINE.....	H-80
EXECUTE	CONSTANT, VALUE = 0.....	H-31
EXINTR	CONSTANT, VALUE = 10.....	H-19
EXITFLAG	BYTE VARIABLE.....	H-195
EXITROU	SUBROUTINE.....	H-83
EXNOREF	CONSTANT, VALUE = 1.....	H-19
FADR	SUBROUTINE.....	H-172
FALSE	CONSTANT, VALUE = NOT TRUE.....	H-16
FF	CONSTANT, VALUE = 00CH.....	H-17
FILL	SUBROUTINE.....	H-155
FIRSTOPR	CONSTANT, VALUE = 00001111B.....	H-16
FIRSTSYM	CONSTANT, VALUE = 7.....	H-17
FLAGS	BYTE VARIABLE.....	H-196
FS	CONSTANT, VALUE = 01CH.....	H-17
GETCMD	SUBROUTINE.....	H-83
GETINDIRECT	SUBROUTINE.....	H-41
GETINFO	SUBROUTINE.....	H-142
GETRA	SUBROUTINE.....	H-41
GETRANGE	SUBROUTINE.....	H-106
GETRPA	SUBROUTINE.....	H-42
GETSYM	SUBROUTINE.....	H-89
GO	SUBROUTINE.....	H-163
GOROU	SUBROUTINE.....	H-164
GRATAB	TABLE.....	H-42
GS	CONSTANT, VALUE = 01DH.....	H-17
GSTRADR	SUBROUTINE.....	H-142
GUAMSG	MESSAGE.....	H-74

GUARDACCESS	CONSTANT, VALUE = 4.....	H-18
GUARDED	CONSTANT, VALUE = RDERR OR WRERR.....	H-18
HELPROU	SUBROUTINE.....	H-108
HLPTAB	TABLE.....	H-109
HLTEMSG	MESSAGE.....	H-74
HLTENCOUNTER	CONSTANT, VALUE = 1.....	H-18
HOMECD	CONSTANT, VALUE = 'H'.....	H-17
HT	CONSTANT, VALUE = 009H.....	H-17
ILLEGALINST	CONSTANT, VALUE = 3.....	H-18
ILLMSG	MESSAGE.....	H-74
IMASK	BYTE VARIABLE.....	H-196
INCPD	SUBROUTINE.....	H-118
INDIRECT	SUBROUTINE.....	H-78
INSERR	MESSAGE.....	H-160
INSERT	SUBROUTINE.....	H-159
IOMAP	ABSOLUTE BYTE ARRAY, LENGTH = MAPLEN.....	H-197
K	CONSTANT, VALUE = 1024.....	H-16
KEYIN	SUBROUTINE.....	H-71
KEYVAL	BYTE VARIABLE.....	H-196
KITIN	SUBROUTINE.....	H-65
KITOUT	SUBROUTINE.....	H-65
KITTEST	SUBROUTINE.....	H-65
LABBUF	BYTE ARRAY, LENGTH = LABLEN.....	H-195
LABCMP	SUBROUTINE.....	H-91
LABLEN	CONSTANT, VALUE = 6.....	H-18
LABPRT	PROC.....	H-160
LASTCH	SUBROUTINE.....	H-91
LASTDIS	WORD VARIABLE.....	H-196
LF	CONSTANT, VALUE = 00AH.....	H-17
LINES	BYTE VARIABLE.....	H-196
LOOKUP	SUBROUTINE.....	H-92
MAPADR	WORD VARIABLE.....	H-197
MAPLEN	CONSTANT, VALUE = 256.....	H-18
MAPPRT	SUBROUTINE.....	H-112
MAPROU	SUBROUTINE.....	H-113
MAPS	ABSOLUTE STRUCTURE.....	H-197
MAPUPDATE	SUBROUTINE.....	H-115
MAPVAL	BYTE VARIABLE.....	H-197
MEMMAP	ABSOLUTE BYTE ARRAY, LENGTH = MAPLEN.....	H-197
MINUS	CONSTANT, VALUE = 6.....	H-17
MSB	CONSTANT, VALUE = 10000000B.....	H-16
MSGINT	MESSAGE.....	H-178
MULT	SUBROUTINE.....	H-93
MYSP	WORD VARIABLE.....	H-196
NAK	CONSTANT, VALUE = 015H.....	H-17
NEWSP	STRUCTURE.....	H-197
NEXECM	MESSAGE.....	H-181
NEXTCH	SUBROUTINE.....	H-93
NONE	CONSTANT, VALUE = 0.....	H-26
NOTEXEC	CONSTANT, VALUE = 0.....	H-19

NUL	CONSTANT, VALUE = 000H.....	H-16
NUMBER	CONSTANT, VALUE = 2.....	H-17
NUMROU	SUBROUTINE.....	H-84
OPCODE	BYTE VARIABLE.....	H-196
OPERAND	BYTE VARIABLE.....	H-196
OPRD16	CONSTANT, VALUE = 4.....	H-27
OPRD8	CONSTANT, VALUE = 3.....	H-27
OPRDREG	CONSTANT, VALUE = 2.....	H-27
OPRDSR	CONSTANT, VALUE = 9.....	H-27
OPRMASK	CONSTANT, VALUE = 00001111B.....	H-16
OPRNONE	CONSTANT, VALUE = 0.....	H-27
OPRRD8	CONSTANT, VALUE = 10.....	H-27
OPRRPA	CONSTANT, VALUE = 5.....	H-27
OPRRPD16	CONSTANT, VALUE = 8.....	H-27
OPRRPL	CONSTANT, VALUE = 6.....	H-27
OPRRPS	CONSTANT, VALUE = 7.....	H-27
OPRRST	CONSTANT, VALUE = 11.....	H-27
OPRSREG	CONSTANT, VALUE = 1.....	H-27
OPRTAB	TABLE.....	H-26
OUTMSG	SUBROUTINE.....	H-66
OUTSTR	SUBROUTINE.....	H-72
P16REF	SUBROUTINE.....	H-173
P8REF	SUBROUTINE.....	H-173
PACKTAB	TABLE.....	H-67
PHEXB	SUBROUTINE.....	H-72
PHEXW	SUBROUTINE.....	H-73
PIOREF	SUBROUTINE.....	H-174
PLABEL	SUBROUTINE.....	H-143
PLUS	CONSTANT, VALUE = 5.....	H-17
PMSG1	MESSAGE.....	H-168
PMSG2	MESSAGE.....	H-168
PNREG	SUBROUTINE.....	H-144
POPERAND	SUBROUTINE.....	H-145
PORTNUM	BYTE VARIABLE.....	H-197
PORTRD	SUBROUTINE.....	H-168
PORTROU	SUBROUTINE.....	H-169
PORTWR	SUBROUTINE.....	H-170
PRINTL	SUBROUTINE.....	H-147
PRINTROU	SUBROUTINE.....	H-175
PROERR	MESSAGE.....	H-170
PROMASK	CONSTANT, VALUE = 00000011B.....	H-18
PROMPT	SUBROUTINE.....	H-21
PRTABLE	SUBROUTINE.....	H-155
PRTADR	SUBROUTINE.....	H-147
PRTALL	SUBROUTINE.....	H-176
PRTBYTES	SUBROUTINE.....	H-148
PRTLIN	SUBROUTINE.....	H-177
PRTNUM	SUBROUTINE.....	H-178
PRTOPRS	SUBROUTINE.....	H-148
PRTREF	SUBROUTINE.....	H-179

PRTREG	SUBROUTINE.....	H-149
PRTYPE	SUBROUTINE.....	H-181
PSCOND	CONSTANT, VALUE = 82H.....	H-19
PSERROR	CONSTANT, VALUE = 85H.....	H-19
PSMTAB	TABLE.....	H-73
PSNUM	CONSTANT, VALUE = 80H.....	H-19
PSTATUS	SUBROUTINE.....	H-73
PSTION	CONSTANT, VALUE = 83H.....	H-19
PSTR01	MESSAGE.....	H-67
PSTR02	MESSAGE.....	H-67
PSTR03	MESSAGE.....	H-67
PSTR04	MESSAGE.....	H-67
PSTR05	MESSAGE.....	H-67
PSTR06	MESSAGE.....	H-67
PSTRIG	CONSTANT, VALUE = 81H.....	H-19
PSYNTAX	CONSTANT, VALUE = 84H.....	H-19
PTYPE	SUBROUTINE.....	H-116
PUTINDIRECT	SUBROUTINE.....	H-42
QCOMMAND	SUBROUTINE.....	H-110
QHNUM	SUBROUTINE.....	H-94
QLETTER	SUBROUTINE.....	H-95
QLLETTER	SUBROUTINE.....	H-95
QNUM	SUBROUTINE.....	H-96
RBEGIN	WORD VARIABLE.....	H-195
RDERR	CONSTANT, VALUE = (NOT READBIT) AND READBIT..	H-18
RDOK	CONSTANT, VALUE = READBIT.....	H-18
READBIT	CONSTANT, VALUE = 00000001B.....	H-18
READNU	SUBROUTINE.....	H-96
READONLY	CONSTANT, VALUE = RDOK OR WRERR.....	H-18
REGA	BYTE VARIABLE.....	H-196
REGADR	WORD VARIABLE.....	H-195
REGB	BYTE VARIABLE.....	H-196
REGBC	WORD VARIABLE.....	H-196
REGC	BYTE VARIABLE.....	H-196
REGD	BYTE VARIABLE.....	H-196
REGDE	WORD VARIABLE.....	H-196
REGDIS	SUBROUTINE.....	H-131
REGDMP	SUBROUTINE.....	H-131
REGE	BYTE VARIABLE.....	H-196
REGH	BYTE VARIABLE.....	H-196
REGHL	WORD VARIABLE.....	H-196
REGISTERS	STRUCTURE.....	H-196
REGL	BYTE VARIABLE.....	H-196
REGLOOKUP	SUBROUTINE.....	H-132
REGM	WORD VARIABLE.....	H-196
REGPC	WORD VARIABLE.....	H-196
REGPSW	WORD VARIABLE.....	H-196
REGROU	SUBROUTINE.....	H-133
REGSP	WORD VARIABLE.....	H-196
REGSTR	WORD VARIABLE.....	H-195

REGTAB	TABLE.....	H-132
REGTER	MESSAGE.....	H-134
REGTYPE	BYTE VARIABLE.....	H-195
REGUPDATE	SUBROUTINE.....	H-135
REND	WORD VARIABLE.....	H-195
RESERVED	SUBROUTINE.....	H-98
RESETROU	SUBROUTINE.....	H-127
RESMAP	SUBROUTINE.....	H-128
RESTAB	TABLE.....	H-100
RGEERR	MESSAGE.....	H-154
ROUND	SUBROUTINE.....	H-189
ROUTINE	BYTE ARRAY, LENGTH = 6.....	H-197
RPAMASK	CONSTANT, VALUE = 00110000B.....	H-16
RPARITH	CONSTANT, VALUE = 5.....	H-26
RPATAB	TABLE.....	H-146
RPLIMIT	CONSTANT, VALUE = 6.....	H-26
RPLMASK	CONSTANT, VALUE = 00010000B.....	H-16
RPLTAB	TABLE.....	H-146
RPSMASK	CONSTANT, VALUE = 00110000B.....	H-16
RPSTAB	TABLE.....	H-146
RPSTACK	CONSTANT, VALUE = 7.....	H-26
RS	CONSTANT, VALUE = 01EH.....	H-17
RST55V	CONSTANT, VALUE = 2CH.....	H-19
RST65V	CONSTANT, VALUE = 34H.....	H-19
RST75V	CONSTANT, VALUE = 3CH.....	H-19
RSTMASK	CONSTANT, VALUE = 00111000B.....	H-16
RSTNUM	CONSTANT, VALUE = 8.....	H-26
RTABLE	TABLE.....	H-149
RUNNING	CONSTANT, VALUE = 0.....	H-18
RXRDY	CONSTANT, VALUE = 00000010B.....	H-64
SCODE	CONSTANT, VALUE = 01000H.....	H-19
SDATA	CONSTANT, VALUE = 0E800H.....	H-19
SECOPR	CONSTANT, VALUE = 11110000B.....	H-16
SHIFTL	SUBROUTINE.....	H-103
SI	CONSTANT, VALUE = 00FH.....	H-17
SO	CONSTANT, VALUE = 00EH.....	H-17
SOH	CONSTANT, VALUE = 001H.....	H-16
SPACE	CONSTANT, VALUE = 020H.....	H-17
SPACES	SUBROUTINE.....	H-75
SREG	CONSTANT, VALUE = 1.....	H-26
SREGMASK	CONSTANT, VALUE = 00000111B.....	H-16
SSINIT	SUBROUTINE.....	H-121
SSTEP	SUBROUTINE.....	H-122
STABLE	CONSTANT, VALUE = 0E000H.....	H-18
STACKL	CONSTANT, VALUE = 100 * WORD.....	H-18
STATUS	BYTE VARIABLE.....	H-196
STDUMP	SUBROUTINE.....	H-160
STEP	SUBROUTINE.....	H-183
STEPCOUNT	WORD VARIABLE.....	H-196
STEPROU	SUBROUTINE.....	H-184

STRA	MESSAGE.....	H-100
STRACI	CONSTANT, VALUE = 3.....	H-27
STRADC	CONSTANT, VALUE = 2.....	H-27
STRADD	CONSTANT, VALUE = 0.....	H-27
STRADI	CONSTANT, VALUE = 1.....	H-27
STRADR	WORD VARIABLE.....	H-196
STRANA	CONSTANT, VALUE = 4.....	H-28
STRANI	CONSTANT, VALUE = 5.....	H-28
STRB	MESSAGE.....	H-100
STRC	MESSAGE.....	H-100
STRCALL	CONSTANT, VALUE = 6.....	H-28
STRCC	CONSTANT, VALUE = 11.....	H-28
STRCM	CONSTANT, VALUE = 10.....	H-28
STRCMA	CONSTANT, VALUE = 15.....	H-28
STRCMC	CONSTANT, VALUE = 16.....	H-28
STRCMP	CONSTANT, VALUE = 17.....	H-28
STRCNC	CONSTANT, VALUE = 12.....	H-28
STRCNZ	CONSTANT, VALUE = 8.....	H-28
STRCP	CONSTANT, VALUE = 9.....	H-28
STRCPE	CONSTANT, VALUE = 13.....	H-28
STRCPI	CONSTANT, VALUE = 18.....	H-28
STRCPO	CONSTANT, VALUE = 14.....	H-28
STRCZ	CONSTANT, VALUE = 7.....	H-28
STRD	MESSAGE.....	H-100
STRDAA	CONSTANT, VALUE = 19.....	H-28
STRDAD	CONSTANT, VALUE = 20.....	H-28
STRDCR	CONSTANT, VALUE = 21.....	H-28
STRDCX	CONSTANT, VALUE = 22.....	H-28
STRDI	CONSTANT, VALUE = 23.....	H-28
STRE	MESSAGE.....	H-100
STREI	CONSTANT, VALUE = 24.....	H-28
STRERR	CONSTANT, VALUE = 80.....	H-31
STRFLAGS	MESSAGE.....	H-100
STRH	MESSAGE.....	H-101
STRHLT	CONSTANT, VALUE = 25.....	H-28
STRIN	CONSTANT, VALUE = 26.....	H-28
STRING	CONSTANT, VALUE = 3.....	H-17
STRINR	CONSTANT, VALUE = 27.....	H-29
STRINX	CONSTANT, VALUE = 28.....	H-29
STRJC	CONSTANT, VALUE = 34.....	H-29
STRJM	CONSTANT, VALUE = 33.....	H-29
STRJMP	CONSTANT, VALUE = 29.....	H-29
STRJNC	CONSTANT, VALUE = 35.....	H-29
STRJNZ	CONSTANT, VALUE = 31.....	H-29
STRJP	CONSTANT, VALUE = 32.....	H-29
STRJPE	CONSTANT, VALUE = 36.....	H-29
STRJPO	CONSTANT, VALUE = 37.....	H-29
STRJZ	CONSTANT, VALUE = 30.....	H-29
STRL	MESSAGE.....	H-101
STRLDA	CONSTANT, VALUE = 38.....	H-29

STRLDX	CONSTANT, VALUE = 39.....	H-29
STRLHLD	CONSTANT, VALUE = 40.....	H-29
STRLXI	CONSTANT, VALUE = 41.....	H-29
STRM	MESSAGE.....	H-101
STRMASK	CONSTANT, VALUE = 01111111B.....	H-16
STRMOV	CONSTANT, VALUE = 42.....	H-29
STRMVI	CONSTANT, VALUE = 43.....	H-29
STRNOP	CONSTANT, VALUE = 44.....	H-29
STRORA	CONSTANT, VALUE = 45.....	H-29
STRORI	CONSTANT, VALUE = 46.....	H-29
STROUT	CONSTANT, VALUE = 47.....	H-29
STRPC	MESSAGE.....	H-102
STRPCHL	CONSTANT, VALUE = 48.....	H-29
STRPOP	CONSTANT, VALUE = 49.....	H-29
STRPUSH	CONSTANT, VALUE = 50.....	H-29
STRRAL	CONSTANT, VALUE = 51.....	H-30
STRRAR	CONSTANT, VALUE = 52.....	H-30
STRRC	CONSTANT, VALUE = 61.....	H-30
STRRET	CONSTANT, VALUE = 56.....	H-30
STRRIM	CONSTANT, VALUE = 55.....	H-30
STRRLC	CONSTANT, VALUE = 53.....	H-30
STRRM	CONSTANT, VALUE = 60.....	H-30
STRRNC	CONSTANT, VALUE = 62.....	H-30
STRRNZ	CONSTANT, VALUE = 58.....	H-30
STRRP	CONSTANT, VALUE = 59.....	H-30
STRRPE	CONSTANT, VALUE = 63.....	H-30
STRRPO	CONSTANT, VALUE = 64.....	H-30
STRRRC	CONSTANT, VALUE = 54.....	H-30
STRRST	CONSTANT, VALUE = 65.....	H-30
STRRZ	CONSTANT, VALUE = 57.....	H-30
STRSBB	CONSTANT, VALUE = 74.....	H-31
STRSBI	CONSTANT, VALUE = 75.....	H-31
STRSHLD	CONSTANT, VALUE = 68.....	H-30
STRSIM	CONSTANT, VALUE = 66.....	H-30
STRSP	MESSAGE.....	H-102
STRSPHL	CONSTANT, VALUE = 67.....	H-30
STRSTA	CONSTANT, VALUE = 69.....	H-30
STRSTC	CONSTANT, VALUE = 71.....	H-30
STRSTX	CONSTANT, VALUE = 70.....	H-30
STRSUB	CONSTANT, VALUE = 72.....	H-30
STRSUI	CONSTANT, VALUE = 73.....	H-30
STRTAB	TABLE.....	H-27
STRXCHG	CONSTANT, VALUE = 76.....	H-31
STRXRA	CONSTANT, VALUE = 78.....	H-31
STRXRI	CONSTANT, VALUE = 79.....	H-31
STRXTHL	CONSTANT, VALUE = 77.....	H-31
STX	CONSTANT, VALUE = 002H.....	H-16
SYMA	CONSTANT, VALUE = FIRSTSYM.....	H-100
SYMB	CONSTANT, VALUE = SYMBYTE+1.....	H-100
SYMBLEN	CONSTANT, VALUE = 8.....	H-18

SYMBOLIC	BYTE VARIABLE.....	H-195
SYMBUF	BYTE ARRAY, LENGTH = SYMBLEN.....	H-195
SYMBYTE	CONSTANT, VALUE = SYMA+1.....	H-100
SYMC	CONSTANT, VALUE = SYMCOUNT+1.....	H-100
SYMCOUNT	CONSTANT, VALUE = SYMB+1.....	H-100
SYMD	CONSTANT, VALUE = SYMDIS+1.....	H-100
SYMDEF	CONSTANT, VALUE = SYMC+1.....	H-100
SYMDIS	CONSTANT, VALUE = SYMDEF+1.....	H-100
SYME	CONSTANT, VALUE = SYMEXIT+1.....	H-100
SYMEXEC	CONSTANT, VALUE = SYMD+1.....	H-100
SYMEXIT	CONSTANT, VALUE = SYMEXEC+1.....	H-100
SYMFLAGS	CONSTANT, VALUE = SYME+1.....	H-100
SYMFOR	CONSTANT, VALUE = SYMFLAGS+1.....	H-100
SYMFROM	CONSTANT, VALUE = SYMFOR+1.....	H-101
SYMGO	CONSTANT, VALUE = SYMGUARDED+1.....	H-101
SYMGUARDED	CONSTANT, VALUE = SYMFROM+1.....	H-101
SYMH	CONSTANT, VALUE = SYMHELP+1.....	H-101
SYMHELP	CONSTANT, VALUE = SYMGO+1.....	H-101
SYMINPUT	CONSTANT, VALUE = SYMH+1.....	H-101
SYMIO	CONSTANT, VALUE = SYMINPUT+1.....	H-101
SYML	CONSTANT, VALUE = SYMLEN+1.....	H-101
SYMLEN	CONSTANT, VALUE = SYMIO+1.....	H-101
SYMM	CONSTANT, VALUE = SYMMEM+1.....	H-101
SYMMAP	CONSTANT, VALUE = SYML+1.....	H-101
SYMMEM	CONSTANT, VALUE = SYMMAP+1.....	H-101
SYMOFF	CONSTANT, VALUE = SYMM+1.....	H-101
SYMON	CONSTANT, VALUE = SYMOFF+1.....	H-102
SYMOUT	CONSTANT, VALUE = SYMON+1.....	H-102
SYMPC	CONSTANT, VALUE = SYMOUT+1.....	H-102
SYMPORT	CONSTANT, VALUE = SYMPC+1.....	H-102
SYMPRINT	CONSTANT, VALUE = SYMPORT+1.....	H-102
SYMR99	MESSAGE.....	H-187
SYMRD	CONSTANT, VALUE = SYMRDO+1.....	H-102
SYMRDO	CONSTANT, VALUE = SYMPRINT+1.....	H-102
SYMREG	CONSTANT, VALUE = SYMRD+1.....	H-102
SYMRESET	CONSTANT, VALUE = SYMREG+1.....	H-102
SYMROU	SUBROUTINE.....	H-186
SYMSP	CONSTANT, VALUE = SYMRESET+1.....	H-102
SYMSTEP	CONSTANT, VALUE = SYMSP+1.....	H-102
SYMSYM	CONSTANT, VALUE = SYMSTEP+1.....	H-102
SYMTILL	CONSTANT, VALUE = SYMSYM+1.....	H-102
SYMTO	CONSTANT, VALUE = SYMTILL+1.....	H-103
SYMTOLAB	SUBROUTINE.....	H-103
SYMTYPE	BYTE VARIABLE.....	H-195
SYMUSER	CONSTANT, VALUE = SYMTO+1.....	H-103
SYMVAL	WORD VARIABLE.....	H-195
SYMWRITE	CONSTANT, VALUE = SYMUSER+1.....	H-103
SYN	CONSTANT, VALUE = 016H.....	H-17
SYNERR	MESSAGE.....	H-81
TAB	CONSTANT, VALUE = 009H.....	H-17

TABOUT	SUBROUTINE.....	H-75
TBACK	WORD VARIABLE.....	H-197
TBCOUNT	WORD VARIABLE.....	H-197
TBELEN	BYTE VARIABLE.....	H-197
TBENTRY	STRUCTURE.....	H-197
TBFREE	WORD VARIABLE.....	H-197
TBFRONT	WORD VARIABLE.....	H-197
TBINIT	SUBROUTINE.....	H-189
TBINTR	SUBROUTINE.....	H-189
TBLEN	CONSTANT, VALUE = 512.....	H-18
TBMASK	CONSTANT, VALUE = NOT (TBLEN-1).....	H-18
TBPTR	WORD VARIABLE.....	H-197
TBUFFER	BYTE ARRAY, LENGTH = TBLEN.....	H-195
TBWRB	SUBROUTINE.....	H-190
TBWREX	SUBROUTINE.....	H-190
TBWRITE	SUBROUTINE.....	H-191
TBWRP	SUBROUTINE.....	H-192
TBWRW	SUBROUTINE.....	H-193
TERMCH	CONSTANT, VALUE = ESC.....	H-17
TRIG	BYTE VARIABLE.....	H-196
TRIG?	SUBROUTINE.....	H-165
TRIGCOND	CONSTANT, VALUE = 2.....	H-18
TRIGEX	CONSTANT, VALUE = 1.....	H-18
TRIGFOR	CONSTANT, VALUE = 0.....	H-18
TRIGIN	CONSTANT, VALUE = 2.....	H-18
TRIGMSG	MESSAGE.....	H-74
TRIGOUT	CONSTANT, VALUE = 3.....	H-18
TRIGRD	CONSTANT, VALUE = 4.....	H-18
TRIGWORD	WORD VARIABLE.....	H-196
TRIGWR	CONSTANT, VALUE = 5.....	H-18
TRUE	CONSTANT, VALUE = 0FFH.....	H-16
TXRDY	CONSTANT, VALUE = 00000001B.....	H-64
UNDEF	MESSAGE.....	H-92
UNKNOWN	CONSTANT, VALUE = 1.....	H-17
US	CONSTANT, VALUE = 01FH.....	H-17
USER	CONSTANT, VALUE = RDOK OR WROK.....	H-18
VALERR	SUBROUTINE.....	H-135
VALLOOK	SUBROUTINE.....	H-149
VALSTR	MESSAGE.....	H-85
VARTYPE	CONSTANT, VALUE = 10000000B.....	H-16
VT	CONSTANT, VALUE = 00BH.....	H-17
WAITMSG	MESSAGE.....	H-139
WORD	CONSTANT, VALUE = 2.....	H-16
WORDVAR	CONSTANT, VALUE = 10000000B.....	H-16
WRERR	CONSTANT, VALUE = NOT WRITEBIT AND WRITEBIT..	H-18
WRITEBIT	CONSTANT, VALUE = 00000010B.....	H-18
WROK	CONSTANT, VALUE = WRITEBIT.....	H-18

```

LOC   OBJ      LINE      SOURCE STATEMENT
1 $TITLE('UCT SABus symbolic debugger')
2 $PAGELENGTH(75)
3 ;
4 ;-----;
5 ;
6 ;      Symbolic debugger for use on the micro teaching Kit's at UCT.
7 ;
8 ;      Written by J.G.Jack 1985
9 ;
10 ;-----;
11 ;
12 ;      The following table lists the files that are combined to form the
13 ;      UCT SABus symbolic debugger (DEB).
14 ;
15 ;-----;
16 ;
17 ;      File Name:      Brief discription:
18 ;      -----
19 ;
20 ;      DEBMAIN.A85      - Main program.
21 ;      DEBCMD.A85      - Command handles.
22 ;
23 ;      DEBEQU.A85      - Equates.
24 ;      DEBDATA.A85     - Data area
25 ;
26 ;      DEBGETSY.A85     - Get symbol routine and support code.
27 ;      DEBRANGE.A85    - Get range routine.
28 ;      DEBTABLE.A85    - Debugger tables.
29 ;      DEBGEN.A85      - General routine.
30 ;      DEBCHECK.A85    - Memory/IO protection checking routines.
31 ;
32 ;      DEBBIO.A85      - Basis I/O routine.
33 ;      DEBOUTMSG       - Packed message output routine.
34 ;      DEBIO.A85       - More complex I/O
35 ;
36 ;      DEBSSTEP.A85     - Routine to execute one instruction
37 ;      DEBEMUL.A85     - Emulation routines.
38 ;      DEBTRACE.A85    - Trace buffer management routines
39 ;
40 ;      DEBRESET.A85     - RESET command routine.
41 ;      DEBREG.A85      - REGISTER command routine.
42 ;      DEBDEF.A85      - DEFINE command routine.
43 ;      DEBMAP.A85      - MAP command routine.
44 ;      DEBHELP.A85     - HELP command routine.
45 ;      DEBPORT.A85     - PORT command routine.
46 ;      DEBSTEP.A85     - STEP command routine.
47 ;      DEBSYM.A85      - SYMBOLIC command routine.
48 ;      DEBPRINT.A85    - PRINT command routine.
49 ;      DEBBYTE.A85     - BYTE command routine.
50 ;      DEBGO.A85       - GO command routine.
51 ;      DEBDIS.A85      - DISASM command routine.
52 ;
53 ;-----;
54 ;
55 ;      NAME      DEB
56 $EJECT

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 2

LOC	OBJ	LINE	SOURCE STATEMENT
		57 ;	-----
		58 ;	
		59 ;	Include all equates used in the debugger.
		60 ;	
		61	\$INCLUDE(DEBEQU.A85)
=		62 ;	-----
=		63 ;	
=		64 ;	Equates used by the UCT SABus debugger.
=		65 ;	
=		66 ;	The equates are divided into the following sections:
=		67 ;	
=		68 ;	A: Logical equates
=		69 ;	B: Storage size Equates (In terms of bytes)
=		70 ;	C: Word and bytes sizes in bits
=		71 ;	D: Masks
=		72 ;	D.1 General masks
=		73 ;	D.2 Instruction table field masks
=		74 ;	D.3 Register position masks
=		75 ;	D.4 Operand table masks
=		76 ;	D.5 Variable field masks and type declarations
=		77 ;	E: ASCII constants
=		78 ;	E.1 Non-printing ASCII character set
=		79 ;	E.2 Control characters
=		80 ;	E.3 Special characters
=		81 ;	E.4 Printable characters
=		82 ;	F: Symbol types
=		83 ;	G: VT52 escape codes
=		84 ;	H: MAP codes
=		85 ;	I: Possible STATUS conditions
=		86 ;	J: Possible trigger conditions
=		87 ;	K: Defaults
=		88 ;	L: Buffer sizes
=		89 ;	M: Main debugger stack length
=		90 ;	N: Memory map equates
=		91 ;	O: Trace buffer entry types
=		92 ;	P: Interrupt vectors
=		93 ;	Q: String equates : Used in packed strings
=		94 ;	
=		95 ;	-----
=		96	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		= 97 ;	-----;
		= 98 ;	
		= 99 ;	A: Logical equates.
		= 100 ;	
00FF		= 101 TRUE EQU OFFH	; Logical TRUE.
FF00		= 102 FALSE EQU NOT TRUE	; Logical FALSE.
		= 103 ;	
		= 104 ;	-----
		= 105 ;	
		= 106 ;	B: Storage size Equates (In terms of bytes).
		= 107 ;	
0001		= 108 BYTE EQU 1	; 1 Byte = 1 Byte.
0002		= 109 WORD EQU 2	; 1 Word = 2 Bytes.
0400		= 110 K EQU 1024	; 1 Kbytes = 1024 bytes.
		= 111 ;	
		= 112 ;	-----
		= 113 ;	
		= 114 ;	C: Word and bytes sizes in bits.
		= 115 ;	
0010		= 116 BITSPW EQU 16	; Bits per word = 16.
0008		= 117 BITSPB EQU 8	; Bits per byte = 8.
		= 118 ;	
		= 119 ;	-----
		= 120 ;	
		= 121 ;	D: Masks:
		= 122 ;	
		= 123 ;	D.1 General masks:
		= 124 ;	
001F		= 125 CTRLMASK EQU 00011111B	; Control mask.
0080		= 126 MSB EQU 10000000B	; Most significant bit
007F		= 127 ASCII EQU 01111111B	; Seven bit ASCII mask.
		= 128 ;	
		= 129 ;	D.2 Instruction table field masks:
		= 130 ;	
007F		= 131 STRMASK EQU 01111111B	; String number mask.
000F		= 132 OPRMASK EQU 00001111B	; Operand info field mask.
001F		= 133 EMMASK EQU 00011111B	; Emulation info field mask.
		= 134 ;	
		= 135 ;	D.3 Register position masks:
		= 136 ;	
0030		= 137 RPAMASK EQU 00110000B	; Register pair arithmetic mask.
0010		= 138 RPLMASK EQU 00010000B	; Register pair limited mask.
0030		= 139 RPSMASK EQU 00110000B	; Register pair stack mask.
0007		= 140 SREGMASK EQU 00000111B	; Source register mask.
0038		= 141 DREGMASK EQU 00111000B	; Destination register mask.
		= 142 ;	
0038		= 143 RSTMASK EQU 00111000B	; Restart number mask.
		= 144 ;	
		= 145 ;	D.4 Operand table masks:
		= 146 ;	
000F		= 147 FIRSTOPR EQU 00001111B	; First operand mask
00F0		= 148 SECOPR EQU 11110000B	; Second operand mask.
		= 149 ;	
		= 150 ;	D.5 Variable field masks and type declarations:
		= 151 ;	
0080		= 152 VARTYPE EQU 10000000B	; Variable type bit position
		= 153 ;	
0000		= 154 BYTEVAR EQU 00000000B	; Byte type
0080		= 155 WORDVAR EQU 10000000B	; Word type
		= 156 ;	
		= 157 ;	-----
		= 158 ;	
		= 159 ;	E: ASCII constants
		= 160 ;	
		= 161 ;	E.1 Non-printing ASCII character set:
		= 162 ;	
0000		= 163 NUL EQU 000H	; ASCII null character
0001		= 164 SOH EQU 001H	; ASCII Start of header character
0002		= 165 STX EQU 002H	; ASCII Start of transmission character
0003		= 166 ETX EQU 003H	; ASCII End of transmission character

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 4

LOC	OBJ	LINE	SOURCE STATEMENT
0004		= 167 EOT	EQU 004H ; ASCII End of text marker.
0005		= 168 ENQ	EQU 005H ; ASCII Enquire character
0006		= 169 ACK	EQU 006H ; ASCII Ack character
0007		= 170 BELL	EQU 007H ; ASCII Bell character.
0008		= 171 BS	EQU 008H ; ASCII Back space.
0009		= 172 HT	EQU 009H ; ASCII Horiz tab character.
0009		= 173 TAB	EQU 009H ; ASCII Horiz tab character.
000A		= 174 LF	EQU 00AH ; ASCII Line feed character.
000B		= 175 VT	EQU 00BH ; ASCII Vertical tab character
000C		= 176 FF	EQU 00CH ; ASCII Form Feed character.
000D		= 177 CR	EQU 00DH ; ASCII Carriage return character.
000E		= 178 SO	EQU 00EH ; ASCII Shift out character
000F		= 179 SI	EQU 00FH ; ASCII Shift in character
		= 180	;
0010		= 181 DLE	EQU 010H ; ASCII DLE character.
0011		= 182 DC1	EQU 011H ; ASCII Device control #1 (Xon)
0012		= 183 DC2	EQU 012H ; ASCII Device control #2
0013		= 184 DC3	EQU 013H ; ASCII Device control #3 (Xoff)
0014		= 185 DC4	EQU 014H ; ASCII Device control #4
0015		= 186 NAK	EQU 015H ; ASCII Negative ACK character
0016		= 187 SYN	EQU 016H ; ASCII Sync character
0017		= 188 ETB	EQU 017H ; ASCII End of transmission buffer.
0018		= 189 CAN	EQU 018H ; ASCII Cancel.
0019		= 190 EM	EQU 019H ; ASCII
001A		= 191 ASUB	EQU 01AH ; ASCII Submit character.
001B		= 192 ESC	EQU 01BH ; ASCII Escape character.
001C		= 193 FS	EQU 01CH ; ASCII Field separator.
001D		= 194 GS	EQU 01DH ; ASCII Group separator
001E		= 195 RS	EQU 01EH ; ASCII Record separator
001F		= 196 US	EQU 01FH ; ASCII Unit separator.
		= 197 ;	
		= 198 ;	E.2 Control characters:
		= 199 ;	
		= 200 ;	
0013		= 201 CTRLS	EQU CTRLMASK AND 'S' ; Control 'S'
0011		= 202 CTRLQ	EQU CTRLMASK AND 'Q' ; Control 'Q'
001A		= 203 CTRLZ	EQU CTRLMASK AND 'Z' ; Control 'Z'
		= 204 ;	
		= 205 ;	E.3 Special characters:
		= 206 ;	
001B		= 207 TERMCH	EQU ESC ; Debug termination character.
001A		= 208 EOFMARK	EQU CTRLZ ; End of file marker.
0000		= 209 ENDMSG	EQU NUL ; End of message marker.
		= 210 ;	
		= 211 ;	E.4 Printable characters:
		= 212 ;	
0020		= 213 SPACE	EQU 020H ; ASCII Space.
002C		= 214 COMMA	EQU ',' ; ASCII Comma
007F		= 215 DEL	EQU 07FH ; ASCII Delete character.
		= 216 ;	
		= 217 ;	-----
		= 218 ;	
		= 219 ;	F: Symbol types:
		= 220 ;	
0000		= 221 EOL	EQU 0 ; End of line.
0001		= 222 UNKNOWN	EQU 1 ; Unknown symbol.
0002		= 223 NUMBER	EQU 2 ; Number.
0003		= 224 STRING	EQU 3 ; ASCII string.
0004		= 225 EQUALS	EQU 4 ; Equals sign '='.
0005		= 226 PLUS	EQU 5 ; Plus sign '+'.
0006		= 227 MINUS	EQU 6 ; minus sign '-'.
		= 228 ;	
0007		= 229 FIRSTSYM	EQU 7 ; First symbol in the reserved word table.
		= 230 ;	
		= 231 ;	-----
		= 232 ;	
		= 233 ;	G: VT52 escape codes.
		= 234 ;	
0048		= 235 HOME	EQU 'H' ; Home the cursor
004A		= 236 CLRS	EQU 'J' ; Clear to end of screen

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 5

LOC	OBJ	LINE	SOURCE STATEMENT
004B		= 237	CLRLINE EQU 'K' ; Clear to end of line
		= 238 ;	
		= 239 ;	
		= 240 ;	
		= 241 ;	H: MAP codes.
		= 242 ;	
0003		= 243	PROMASK EQU 00000011B ; Position of the protection bits
		= 244 ;	
0001		= 245	READBIT EQU 00000001B ; Position of the Read permission bit
0002		= 246	WRITEBIT EQU 00000010B ; Position of the Write permission bit
		= 247 ;	
0001		= 248	RDOK EQU READBIT ; Read OK
0000		= 249	RDERR EQU (NOT READBIT) AND READBIT ; Read Error
0002		= 250	WROK EQU WRITEBIT ; Write OK
0000		= 251	WRERR EQU (NOT WRITEBIT) AND WRITEBIT ; Write Error
		= 252 ;	
0003		= 253	USER EQU RDOK OR WROK ; Can read or write in user mode
0000		= 254	GUARDED EQU RDERR OR WRERR ; Cannot read or write in guarded mode
0001		= 255	READONLY EQU RDOK OR WRERR ; Can not write in readonly mode
		= 256 ;	
		= 257 ;	
		= 258 ;	
		= 259 ;	I: Possible STATUS conditions:
		= 260 ;	
0000		= 261	RUNNING EQU 0 ; Program running
0001		= 262	HLTENCOUNTERED EQU 1 ; Program halted : Halt encountered.
0002		= 263	TRIGCONDITION EQU 2 ; Program halted : Trigger condition met.
0003		= 264	ILLEGALINST EQU 3 ; Program halted : Illegal instruction.
0004		= 265	GUARDACCESS EQU 4 ; Program halted : Guarded access.
0005		= 266	ESCENTERED EQU 5 ; Program halted : ESC typed by user.
		= 267 ;	
		= 268 ;	
		= 269 ;	
		= 270 ;	J: Possible trigger conditions:
		= 271 ;	
0000		= 272	TRIGFOR EQU 0 ; Forever
0001		= 273	TRIGEX EQU 1 ; Execute
0002		= 274	TRIGIN EQU 2 ; Input
0003		= 275	TRIGOUT EQU 3 ; Output
0004		= 276	TRIGRD EQU 4 ; Read
0005		= 277	TRIGWR EQU 5 ; Write
		= 278 ;	
		= 279 ;	
		= 280 ;	
		= 281 ;	K: Defaults:
		= 282 ;	
0001		= 283	DEFCOUNT EQU 1 ; Default number of steps used by the
		= 284	single step instructino.
		= 285 ;	
		= 286 ;	
		= 287 ;	
		= 288 ;	L: Buffer sizes.
		= 289 ;	
0008		= 290	SYMBLEN EQU 8 ; Symbol buffer length
0006		= 291	LABLEN EQU 6 ; Label buffer length.
0100		= 292	MAPLEN EQU 256 ; Map length.
0200		= 293	TBLEN EQU 512 ; Trace buffer length (Must be a power of 2)
FDFD		= 294	TBMASK EQU NOT TBLEN ; Mask used for wrap around
		= 295 ;	
		= 296 ;	M: Main debugger stack length
		= 297 ;	
00C8		= 298	STACKL EQU 100 * WORD ; Debugger main stack
		= 299 ;	
		= 300 ;	
		= 301 ;	
		= 302 ;	N: Memory map equates:
		= 303 ;	
E000		= 304	STABLE EQU 0E000H ; Start address of the Symbol table.
		= 305	;(Size = 256 * 8 = 2K)
		= 306 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 6

LOC	OBJ	LINE	SOURCE STATEMENT
1000		= 307	SCODE EQU 01000H ; Start address of the debugger
E800		= 308	SDATA EQU 0E800H ; Start address of the debugger data area
		= 309	; (Size +/- 256 bytes , includes the stack)
		= 310 ;	
		= 311 ;	-----
		= 312 ;	
		= 313 ;	O: Trace buffer entry types:
		= 314 ;	
0000		= 315	NOTEXEC EQU 0 ; Instruction not executed
0001		= 316	EXNOREF EQU 1 ; Executed - No memory references
0002		= 317	EX8BRD EQU 2 ; Executed - 8 bit read
0003		= 318	EX8BWR EQU 3 ; Executed - 8 bit write
0004		= 319	EX16BRD EQU 4 ; Executed - 16 bit read
0005		= 320	EX16BWR EQU 5 ; Executed - 16 bit write
0006		= 321	EX8BIN EQU 6 ; Executed - 8 bit input
0007		= 322	EX8BOUT EQU 7 ; Executed - 8 bit output
0008		= 323	EX8BMOD EQU 8 ; Executed - 8 bit modify
0009		= 324	EX16BMOD EQU 9 ; Executed - 16 bit modify
000A		= 325	EXINTR EQU 10 ; Executed - Interrupt
		= 326 ;	
		= 327 ;	-----
		= 328 ;	
		= 329 ;	P: Interrupt vectors.
		= 330 ;	
002C		= 331	RST55V EQU 2CH ; Address that the 8085 jumps to on receiving
		= 332	; an RST 5.5 interrupt.
0034		= 333	RST65V EQU 34H ; (same as above but for RST 6.5)
003C		= 334	RST75V EQU 3CH ; (same as above but for RST 7.5)
		= 335 ;	
		= 336 ;	-----
		= 337 ;	
		= 338 ;	Q: Packed string equates:
		= 339 ;	
0080		= 340	PSNUM EQU 80H ; Pack string byte for the string '<Number> '
0081		= 341	PSTRIG EQU 81H ; As above for the string '<TriggerCondition> '
0082		= 342	PSCOND EQU 82H ; As above for the string '<Condition> '
0083		= 343	PSTION EQU 83H ; As above for the string 'tion'
0084		= 344	PSYNTAX EQU 84H ; As above for the string 'yntax'
0085		= 345	PSERROR EQU 85H ; As above for the string 'error'
		= 346 ;	
		= 347 ;	END <DEBEQU.A85>
		348	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 7

LOC	OBJ	LINE	SOURCE STATEMENT
		349 ;	-----
		350 ;	
		351 ;	START OF THE MAIN PROGRAM
		352 ;	
		353 ;	-----
		354 ;	
1000		355	ORG SCODE ;
		356	;
1000		357	AACODE EQU \$; Start of code marker
		358	;
		359	DEB: ; Program DEB;
		360	;
1000	F3	361	DI ; DisableInterrupts;
		362	;
1001	31CBED	363	LXI SP,NEWSP ;
1004	CD1F10	364	CALL DEBINIT ; DEBInitialise;
		365	;
		366	NEXTCMD: ; Repeat
1007	CD1910	367	CALL PROMPT ; Prompt;
100A	CD3C1B	368	CALL GETCMD ; GetCommand;
100D	CDCF1A	369	CALL EXECMD ; ExecuteCommand;
1010	3A79EA	370	LDA EXITFLAG ;
1013	FE00	371	CPI FALSE ;
1015	CA0710	372	JZ NEXTCMD ; Until ExitFlag = TRUE
		373	DEBEND: ;
1018	76	374	HLT ; End.
		375 ;	
		376	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 8

LOC	OBJ	LINE	SOURCE STATEMENT
		377	;=====*
		378	;* *
		379	;* Function : PROMPT (Prompt user) *
		380	;* Inputs : None. *
		381	;* Outputs : None. *
		382	;* Destroys : A and flags. *
		383	;* Calls : KITOUT. *
		384	;* *
		385	;* Description : This routine prompts the user to enter a command. *
		386	;* *
		387	;=====*
		388	;
		389	PROMPT: ; Procedure Prompt;
1019	3E23	390	MVI A,'#' ;
101B	CDE218	391	CALL KITOUT ; KitOut('#');
		392	;
101E	C9	393	RET ; EndProc prompt;
		394	;
		395	;=====*
		396	;* *
		397	;* Function : DEBINIT (Debugger initialise). *
		398	;* Inputs : None. *
		399	;* Outputs : None. *
		400	;* Destroys : H, L, A and Flags. *
		401	;* Calls : TBINIT. *
		402	;* *
		403	;* Description : This routine initialises the memory variables. *
		404	;* *
		405	;=====*
		406	;
		407	DEBINIT: ; Procedure DebuggerInitialise;
101F	3E00	408	MVI A,FALSE ;
1021	3279EA	409	STA EXITFLAG ; ExitFlag := False;
1024	32A2EA	410	STA ENABINT ; Enable_interrupt := False;
		411	;
1027	3EFF	412	MVI A,TRUE ;
1029	327AEA	413	STA SYMBOLIC ; Symbolic := True;
		414	;
102C	210100	415	LXI H,DEFCOUNT ;
102F	22B0EA	416	SHLD STEPCOUNT ; StepCount := DefaultCount;
		417	;
1032	3E00	418	MVI A,0 ;
1034	32A1EA	419	STA IMASK ; InterruptMask := 0;
1037	3E10	420	MVI A,00010000B ;
1039	30	421	SIM ; ResetRST75;
		422	;
103A	CD352F	423	CALL TBINIT ; TraceBufferInitialise;
		424	;
103D	C9	425	RET ; EndProc Debuggerinitialise;
		426	;
		427	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 9

LOC	OBJ	LINE	SOURCE STATEMENT
		428 ;	-----
		429 ;	
		430 ;	I N C L U D E F I L E S :
		431 ;	
		432 ;	-----
		433 ;	
		434 ;	Memory and IO protection map check routines:
		435 ;	
		436 \$INCLUDE(DEBCHECK.A85)	
= 437 ;			-----;
= 438 ;			;
= 439 ;			Protection checking routines used by the UCT SABus debugger. ;
= 440 ;			;
= 441 ;			This module contains (in order) the following subroutines. ;
= 442 ;			;
= 443 ;			a) CHKM RD - Check memory read. ;
= 444 ;			b) CHKM WR - Check memory write. ;
= 445 ;			c) CHK PRD - Check port read. ;
= 446 ;			d) CHK PWR - Check port write. ;
= 447 ;			;
= 448 ;			-----;
= 449			\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 10

LOC	OBJ	LINE	SOURCE STATEMENT
		= 450 ;	=====*
		= 451 ;*	*
		= 452 ;*	Function : CHKMRD (Check memory read). *
		= 453 ;*	Inputs : HL : Address to check. *
		= 454 ;*	Outputs : Z = 1 : Access denied. *
		= 455 ;*	Z = 0 : Then Access allowed. *
		= 456 ;*	Destroys : A and Flags. *
		= 457 ;*	Calls : DCMP. *
		= 458 ;*	*
		= 459 ;*	Description : This routine performs two functions. The first is to *
		= 460 ;*	determine if this access is the trigger condition. For this to be *
		= 461 ;*	true the type of trigger (given by the memory variable TRIG) must *
		= 462 ;*	be a memory read (TRIGRD) and the trigger address (given by the *
		= 463 ;*	memory variable TRIGWORD) must be equal to the address being *
		= 464 ;*	accessed (i.e. the HL register pair). If these conditions are both *
		= 465 ;*	true then the debugger status is changed to TRIGGER_CONDITION to *
		= 466 ;*	indicate that the trigger condition has been encountered. *
		= 467 ;*	*
		= 468 ;*	The second function of this routine is to check and see if a read *
		= 469 ;*	from the memory location is allowed. It does this by checking the *
		= 470 ;*	readbit in the memory map. *
		= 471 ;*	*
		= 472 ;	=====*
		= 473 ;	
		= 474	CHKMRD: ; Procedure CheckMemoryRead(Address:Integer);
103E	E5	= 475	PUSH H ;
103F	D5	= 476	PUSH D ;
		= 477	;
1040	3AB3EA	= 478	LDA TRIG ;
1043	FE04	= 479	CPI TRIGRD ;
1045	C25910	= 480	JNZ CMRD10 ; If Trigger = MemoryRead AND
		= 481	;
1048	E5	= 482	PUSH H ;
1049	EB	= 483	XCHG ;
104A	2AB4EA	= 484	LHLD TRIGWORD ;
104D	CDBF1A	= 485	CALL DCMP ;
1050	E1	= 486	POP H ;
1051	C25910	= 487	JNZ CMRD10 ; TriggerWord = Address then
		= 488	;
1054	3E02	= 489	MVI A,TRIGCONDITION ;
1056	32B2EA	= 490	STA STATUS ; Status := Triggercondition;
		= 491	CMRD10: ;
1059	1600	= 492	MVI D,0 ;
105B	5C	= 493	MOV E,H ;
105C	2100EC	= 494	LXI H,MEMMAP ;
105F	19	= 495	DAD D ;
1060	7E	= 496	MOV A,M ; Protection := MemoryMap[HIGH Address];
		= 497	;
1061	E601	= 498	ANI READBIT ; Return(Protection AND ReadBit)
1063	D1	= 499	POP D ;
1064	E1	= 500	POP H ;
1065	C9	= 501	RET ; EndProc CheckMemoryRead;
		= 502 ;	
		= 503 ;	=====*
		= 504 ;*	*
		= 505 ;*	Function : CHKMWR (Check memory write). *
		= 506 ;*	Inputs : HL : Address to check: *
		= 507 ;*	Outputs : Z = 1 : Access denied. *
		= 508 ;*	Z = 0 : Then Access allowed. *
		= 509 ;*	Destroys : A and Flags. *
		= 510 ;*	Calls : DCMP. *
		= 511 ;*	*
		= 512 ;*	Description : This routine performs two functions. The first is to *
		= 513 ;*	determine if this access is the trigger condition. For this to be *
		= 514 ;*	true the type of trigger (given by the memory variable TRIG) must *
		= 515 ;*	be a memory write (TRIGWR) and the trigger address (given by the *
		= 516 ;*	memory variable TRIGWORD) must be equal to the address being *
		= 517 ;*	accessed (i.e. the HL register pair). If these conditions are both *
		= 518 ;*	true then the debugger status is changed to TRIGGER_CONDITION to *
		= 519 ;*	indicate that the trigger condition has been encountered. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 11

LOC	OBJ	LINE	SOURCE STATEMENT
		= 520 ;*	
		= 521 ;*	The second function of this routine is to check and see if a write
		= 522 ;*	to the memory location is allowed. It does this by checking the
		= 523 ;*	writebit in the memory map.
		= 524 ;*	
		= 525 ;=====*	
		= 526 ;	
		= 527	CHKMWR: ; Procedure CheckMemoryWrite(Address:Integer);
1066	E5	= 528	PUSH H ;
1067	D5	= 529	PUSH D ;
		= 530	;
1068	3AB3EA	= 531	LDA TRIG ;
106B	FE05	= 532	CPI TRIGWR ;
106D	C28110	= 533	JNZ CMWR10 ; If Trigger = MemoryWrite AND
		= 534	;
1070	E5	= 535	PUSH H ;
1071	EB	= 536	XCHG ;
1072	2AB4EA	= 537	LHLD TRIGWORD ;
1075	CDBF1A	= 538	CALL DCMP ;
1078	E1	= 539	POP H ;
1079	C28110	= 540	JNZ CMWR10 ; TriggerWord = Address then
		= 541	;
107C	3E02	= 542	MVI A,TRIGCONDITION ;
107E	32B2EA	= 543	STA STATUS ; Status := Triggercondition;
		= 544	CMWR10: ;
1081	1600	= 545	MVI D,0 ;
1083	5C	= 546	MOV E,H ;
1084	2100EC	= 547	LXI H,MEMMAP ;
1087	19	= 548	DAD D ;
1088	7E	= 549	MOV A,M ; Protection := MemoryMap[HIGH Address];
		= 550	;
1089	E602	= 551	ANI WRITEBIT ; Return(Protection AND WriteBit)
108B	D1	= 552	POP D ;
108C	E1	= 553	POP H ;
108D	C9	= 554	RET ; EndProc CheckMemoryWrite;
		= 555 ;	
		= 556 ;=====*	
		= 557 ;*	
		= 558 ;*	Function : CHKPRD (Check port read).
		= 559 ;*	Inputs : A : Port number to be checked.
		= 560 ;*	Outputs : Z = 1 : Access denied.
		= 561 ;*	Z = 0 : Then Access allowed.
		= 562 ;*	Destroys : A and flags.
		= 563 ;*	Calls : None.
		= 564 ;*	
		= 565 ;*	Description : This routine performs two functions. The first is to
		= 566 ;*	determine if this access is the trigger condition. For this to be
		= 567 ;*	true the type of trigger (given by the memory variable TRIG) must
		= 568 ;*	be a port input (TRIGIN) and the trigger address (given by the
		= 569 ;*	memory variable TRIGWORD) must be equal to the port number being
		= 570 ;*	accessed (i.e. the A register pair). If these conditions are both
		= 571 ;*	true then the debugger status is changed to TRIGGER_CONDITION to
		= 572 ;*	indicate that the trigger condition has been encountered.
		= 573 ;*	
		= 574 ;*	The second function of this routine is to check and see if an input
		= 575 ;*	to the IO location is allowed. It does this by checking the
		= 576 ;*	readbit in the IO map.
		= 577 ;*	
		= 578 ;=====*	
		= 579 ;	
		= 580	CHKPRD: ; Procedure CheckPortRead(PortNumber:Byte);
108E	E5	= 581	PUSH H ;
108F	D5	= 582	PUSH D ;
1090	1600	= 583	MVI D,0 ;
1092	5F	= 584	MOV E,A ;
		= 585	;
1093	3AB3EA	= 586	LDA TRIG ;
1096	FE02	= 587	CPI TRIGIN ;
1098	C2A710	= 588	JNZ CPDR10 ; If Trigger = OutputInput AND
		= 589	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 12

```

LOC  OBJ      LINE      SOURCE STATEMENT
109B  3AB4EA    = 590      LDA      TRIGWORD      ;
109E  BB        = 591      CMP      E              ;      TriggerWord = PortAddress then
109F  C2A710    = 592      JNZ      CPWD10      ;
          = 593              ;
10A2  3E02      = 594      MVI      A,TRIGCONDITION ;
10A4  32B2EA    = 595      STA      STATUS      ;      Status := TriggerCondition;
          = 596 CPWD10:    ;
10A7  2100EB    = 597      LXI      H,IOMAP      ;
10AA  19        = 598      DAD      D              ;
10AB  7E        = 599      MOV      A,M          ;      Protection := IOMap[PortNumber];
          = 600              ;
10AC  E601      = 601      ANI      READBIT      ;      Return(Protection AND ReadBit)
10AE  D1        = 602      POP      D              ;
10AF  E1        = 603      POP      H              ;
10B0  C9        = 604      RET                      ; EndProc CheckPortRead;
          = 605 ;
          = 606 ;=====
          = 607 ;*
          = 608 ;* Function : CHKPWR (Check port write). *
          = 609 ;* Inputs : A : Port number to be checked. *
          = 610 ;* Outputs : Z = 1 : Access denied. *
          = 611 ;*          Z = 0 : Then Access allowed. *
          = 612 ;* Destroys : A and Flags. *
          = 613 ;* Calls : None. *
          = 614 ;*
          = 615 ;* Description : This routine performs two functions. The first is to *
          = 616 ;* determine if this access is the trigger condition. For this to be *
          = 617 ;* true the type of trigger (given by the memory variable TRIG) must *
          = 618 ;* be a port output (TRIGOUT) and the trigger address (given by the *
          = 619 ;* memory variable TRIGWORD) must be equal to the port number being *
          = 620 ;* accessed (i.e. the A register pair). If these conditions are both *
          = 621 ;* true then the debugger status is changed to TRIGGER_CONDITION to *
          = 622 ;* indicate that the trigger condition has been encountered. *
          = 623 ;*
          = 624 ;* The second function of this routine is to check and see if an *
          = 625 ;* output to the IO location is allowed. It does this by checking the *
          = 626 ;* writebit in the IO map. *
          = 627 ;*
          = 628 ;=====
          = 629 ;
          = 630 CHKPWR:      ; Procedure CheckPortWrite(PortNumber:Byte);
10B1  E5        = 631      PUSH     H              ;
10B2  D5        = 632      PUSH     D              ;
10B3  1600      = 633      MVI      D,0          ;
10B5  5F        = 634      MOV      E,A          ;
          = 635              ;
10B6  3AB3EA    = 636      LDA      TRIG          ;
10B9  FE03      = 637      CPI      TRIGOUT      ;
10BB  C2CA10    = 638      JNZ      CPWR10      ;      If Trigger = PortOutput AND
          = 639              ;
10BE  3AB4EA    = 640      LDA      TRIGWORD     ;
10C1  BB        = 641      CMP      E              ;      TriggerWord = PortAddress then
10C2  C2CA10    = 642      JNZ      CPWR10      ;
          = 643              ;
10C5  3E02      = 644      MVI      A,TRIGCONDITION ;
10C7  32B2EA    = 645      STA      STATUS      ;      Status := TriggerCondition;
          = 646 CPWR10:    ;
10CA  2100EB    = 647      LXI      H,IOMAP      ;
10CD  19        = 648      DAD      D              ;
10CE  7E        = 649      MOV      A,M          ;      Protection := IOMap[PortNumber];
          = 650              ;
10CF  E602      = 651      ANI      WRITEBIT     ;      Return(Protection AND WriteBit)
10D1  D1        = 652      POP      D              ;
10D2  E1        = 653      POP      H              ;
10D3  C9        = 654      RET                      ; EndProc CheckPortWrite;
          = 655 ;
          = 656 ;      END      <DEBCHECK.A85>
          = 657 $EJECT

```


UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 13

```

LOC  OBJ      LINE      SOURCE STATEMENT
                                658 ;
                                659 ;      Debugger tables:
                                660 ;
                                661 $INCLUDE(DEBTABLE.A85)
= 662 ;-----;
= 663 ;
= 664 ;      UCT SABus Debugger Tables:
= 665 ;
= 666 ;      A: Operand and byte count table: (OPRTAB)
= 667 ;
= 668 ;      There is an entry in the table for every possible combination
= 669 ;      of operands. Each entry is two bytes (16 bits) long. The
= 670 ;      format is given below:
= 671 ;
= 672 ;      Byte1      Byte2      aaaaaaaa = 8 bit byte count.
= 673 ;      aaaaaaaa bbbbbbcc   bbbb    = Second operand.
= 674 ;      cccc          = First operand.
= 675 ;
= 676 ;      B: String table: (STRTAB)
= 677 ;
= 678 ;      There is an entry for every instruction mnemonic. The entries
= 679 ;      are not all the same length. The strings are terminated by
= 680 ;      the high bit being set in the last character.
= 681 ;
= 682 ;      C: Executing/Emulation information table: (EMUTAB)
= 683 ;
= 684 ;      This is a table of emulation routine address. Each entry is
= 685 ;      one word (16 bits) long.
= 686 ;
= 687 ;      D: Instruction table: (DISTAB)
= 688 ;
= 689 ;      This is the main table used by all sections of the debugger.
= 690 ;      There are 256 entries in this table, one entry for every
= 691 ;      possible opcode (i.e. 0H..0FFH). Each entry is 16 bits long
= 692 ;      and has the following format.
= 693 ;
= 694 ;      eeeee oooo sssssss   eeeee   = Execution/Emulation code.
= 695 ;                          oooo    = Operand code.
= 696 ;                          sssssss = mnemonic string number.
= 697 ;
= 698 ;-----;
= 699 ;
= 700 ;
= 701 ;
= 702 ;      TABLE A: Operand table.
= 703 ;
= 704 ;      Format summary:
= 705 ;
= 706 ;      Byte 1 : Number of bytes used by the instruction (1..3)
= 707 ;      Byte 2 : Two four bit operand fields
= 708 ;                Lower nibble = first operand (0 if none exists)
= 709 ;                Upper-nibble = second operand.
= 710 ;
= 711 ;-----;
= 712 ;
= 713 ;      Table of possible operands:
= 714 ;
0000 = 715 NONE EQU 0 ; None operand (e.g. NOP)
0001 = 716 SREG EQU 1 ; Source register (e.g. ADD B)
0002 = 717 DREG EQU 2 ; Destination register (e.g. MVI C,55)
0003 = 718 DATA8 EQU 3 ; Eight bit data (e.g. ADI 77)
0004 = 719 DATA16 EQU 4 ; Sixteen bit data (e.g. JMP 1234)
0005 = 720 RPARITH EQU 5 ; Register pair arithmetic (e.g. DAD SP)
0006 = 721 RPLIMIT EQU 6 ; Register pair limited (e.g. STAX B)
0007 = 722 RPSTACK EQU 7 ; Register pair stack (e.g. PUSH PSW)
0008 = 723 RSTNUM EQU 8 ; Restart number (e.g. RST 5)
= 724 ;
= 725 ;-----;
= 726 ;
= 727 OPRTAB: ; Start of operand table

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 14

LOC	OBJ	LINE	SOURCE STATEMENT
		= 728	
0000		= 729 OPRNONE EQU	0 ; No operand entry
10D4 01		= 730 DB	1 ; ByteCount : 1
10D5 00		= 731 DB	NONE + (NONE SHL 4) ; Operands : None
		= 732	
0001		= 733 OPRSREG EQU	1 ; Source register entry
10D6 01		= 734 DB	1 ; ByteCount : 1
10D7 01		= 735 DB	SREG + (NONE SHL 4) ; Operands : SourceRegister
		= 736	
0002		= 737 OPRDREG EQU	2 ; Destination register entry
10D8 01		= 738 DB	1 ; ByteCount : 1
10D9 02		= 739 DB	DREG + (NONE SHL 4) ; Operands : DestinationRegister
		= 740	
0003		= 741 OPRD8 EQU	3 ; 8 bit data entry
10DA 02		= 742 DB	2 ; ByteCount : 2
10DB 03		= 743 DB	DATA8 + (NONE SHL 4) ; Operands : Data8
		= 744	
0004		= 745 OPRD16 EQU	4 ; 16 bit data entry
10DC 03		= 746 DB	3 ; ByteCount : 3
10DD 04		= 747 DB	DATA16 + (NONE SHL 4) ; Operands : Data16
		= 748	
0005		= 749 OPRRPA EQU	5 ; Register pair arithmetic entry
10DE 01		= 750 DB	1 ; ByteCount : 1
10DF 05		= 751 DB	RPARITH + (NONE SHL 4) ; Operands : RegPairArith
		= 752	
0006		= 753 OPRRPL EQU	6 ; Register pair limited entry
10E0 01		= 754 DB	1 ; ByteCount : 1
10E1 06		= 755 DB	RPLIMIT + (NONE SHL 4) ; Operands : RegPairLimited
		= 756	
0007		= 757 OPRRPS EQU	7 ; Register pair stack entry
10E2 01		= 758 DB	1 ; ByteCount : 1
10E3 07		= 759 DB	RPSTACK + (NONE SHL 4) ; Operands : RegPairStack
		= 760	
0008		= 761 OPRRPD16 EQU	8 ; Register pair, 16 bit data entry
10E4 03		= 762 DB	3 ; ByteCount : 3
10E5 45		= 763 DB	RPARITH + (DATA16 SHL 4) ; Operands : RegPairArith,Data16
		= 764	
0009		= 765 OPRDSR EQU	9 ; Destination, Source register entry
10E6 01		= 766 DB	1 ; ByteCount : 1
10E7 12		= 767 DB	DREG + (SREG SHL 4) ; Operands : Destination, Source
		= 768	
000A		= 769 OPRRD8 EQU	10 ; Destination reg, 8 bit data entry
10E8 02		= 770 DB	2 ; ByteCount : 2
10E9 32		= 771 DB	DREG + (DATA8 SHL 4) ; Operands : Destination, Data8
		= 772	
000B		= 773 OPRRST EQU	11 ; Restart number entry
10EA 01		= 774 DB	1 ; ByteCount : 1
10EB 08		= 775 DB	RSTNUM + (NONE SHL 4) ; Operands : RSTNumber
		= 776 ;	
		= 777 ;	
		= 778 ;	
		= 779 ;	String table: Each string is terminated by a character with its top
		= 780 ;	bit set.
		= 781 ;	
		= 782 ;	
		= 783 ;	
		= 784 ;	
		= 785 STRTAB:	
		= 786 ;	
0000		= 787 STRADD EQU	0
10EC 4144C4		= 788 DB	'AD', 'D' + MSB
		= 789 ;	
0001		= 790 STRADI EQU	1
10EF 4144C9		= 791 DB	'AD', 'I' + MSB
		= 792 ;	
0002		= 793 STRADC EQU	2
10F2 4144C3		= 794 DB	'AD', 'C' + MSB
		= 795 ;	
0003		= 796 STRACI EQU	3
10F5 4143C9		= 797 DB	'AC', 'I' + MSB

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 15

LOC	OBJ	LINE	SOURCE STATEMENT
		= 798 ;	
0004		= 799 STRANA EQU	4
10F8	414EC1	= 800 DB	'AN', 'A' + MSB
		= 801 ;	
0005		= 802 STRANI EQU	5
10FB	414EC9	= 803 DB	'AN', 'I' + MSB
		= 804 ;	
0006		= 805 STRCALL EQU	6
10FE	43414CCC	= 806 DB	'CAL', 'L' + MSB
		= 807 ;	
0007		= 808 STRCZ EQU	7
1102	43DA	= 809 DB	'C', 'Z' + MSB
		= 810 ;	
0008		= 811 STRCNZ EQU	8
1104	434EDA	= 812 DB	'CN', 'Z' + MSB
		= 813 ;	
0009		= 814 STRCP EQU	9
1107	43D0	= 815 DB	'C', 'P' + MSB
		= 816 ;	
000A		= 817 STRCM EQU	10
1109	43CD	= 818 DB	'C', 'M' + MSB
		= 819 ;	
000B		= 820 STRCC EQU	11
110B	43C3	= 821 DB	'C', 'C' + MSB
		= 822 ;	
000C		= 823 STRCNC EQU	12
110D	434EC3	= 824 DB	'CN', 'C' + MSB
		= 825 ;	
000D		= 826 STRCPE EQU	13
1110	4350C5	= 827 DB	'CP', 'E' + MSB
		= 828 ;	
000E		= 829 STRCPO EQU	14
1113	4350CF	= 830 DB	'CP', 'O' + MSB
		= 831 ;	
000F		= 832 STRCMA EQU	15
1116	434DC1	= 833 DB	'CM', 'A' + MSB
		= 834 ;	
0010		= 835 STRCMC EQU	16
1119	434DC3	= 836 DB	'CM', 'C' + MSB
		= 837 ;	
0011		= 838 STRCMP EQU	17
111C	434DD0	= 839 DB	'CM', 'P' + MSB
		= 840 ;	
0012		= 841 STRCPI EQU	18
111F	4350C9	= 842 DB	'CP', 'I' + MSB
		= 843 ;	
0013		= 844 STRDAA EQU	19
1122	4441C1	= 845 DB	'DA', 'A' + MSB
		= 846 ;	
0014		= 847 STRDAD EQU	20
1125	4441C4	= 848 DB	'DA', 'D' + MSB
		= 849 ;	
0015		= 850 STRDCR EQU	21
1128	4443D2	= 851 DB	'DC', 'R' + MSB
		= 852 ;	
0016		= 853 STRDCX EQU	22
112B	4443D8	= 854 DB	'DC', 'X' + MSB
		= 855 ;	
0017		= 856 STRDI EQU	23
112E	44C9	= 857 DB	'D', 'I' + MSB
		= 858 ;	
0018		= 859 STREI EQU	24
1130	45C9	= 860 DB	'E', 'I' + MSB
		= 861 ;	
0019		= 862 STRHLT EQU	25
1132	484CD4	= 863 DB	'HL', 'T' + MSB
		= 864 ;	
001A		= 865 STRIN EQU	26
1135	49CE	= 866 DB	'I', 'N' + MSB
		= 867 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 16

LOC	OBJ	LINE	SOURCE STATEMENT
001B		= 868	STRINR EQU 27
1137	494ED2	= 869	DB 'IN', 'R' + MSB
		= 870 ;	
001C		= 871	STRINX EQU 28
113A	494ED8	= 872	DB 'IN', 'X' + MSB
		= 873 ;	
001D		= 874	STRJMP EQU 29
113D	4A4DD0	= 875	DB 'JM', 'P' + MSB
		= 876 ;	
001E		= 877	STRJZ EQU 30
1140	4ADA	= 878	DB 'J', 'Z' + MSB
		= 879 ;	
001F		= 880	STRJNZ EQU 31
1142	4A4EDA	= 881	DB 'JN', 'Z' + MSB
		= 882 ;	
0020		= 883	STRJP EQU 32
1145	4AD0	= 884	DB 'J', 'P' + MSB
		= 885 ;	
0021		= 886	STRJM EQU 33
1147	4ACD	= 887	DB 'J', 'M' + MSB
		= 888 ;	
0022		= 889	STRJC EQU 34
1149	4AC3	= 890	DB 'J', 'C' + MSB
		= 891 ;	
0023		= 892	STRJNC EQU 35
114B	4A4EC3	= 893	DB 'JN', 'C' + MSB
		= 894 ;	
0024		= 895	STRJPE EQU 36
114E	4A50C5	= 896	DB 'JP', 'E' + MSB
		= 897 ;	
0025		= 898	STRJPO EQU 37
1151	4A50CF	= 899	DB 'JP', 'O' + MSB
		= 900 ;	
0026		= 901	STRLDA EQU 38
1154	4C44C1	= 902	DB 'LD', 'A' + MSB
		= 903 ;	
0027		= 904	STRLDX EQU 39
1157	4C4441D8	= 905	DB 'LDA', 'X' + MSB
		= 906 ;	
0028		= 907	STRLHLD EQU 40
115B	4C484CC4	= 908	DB 'LHL', 'D' + MSB
		= 909 ;	
0029		= 910	STRLXI EQU 41
115F	4C58C9	= 911	DB 'LX', 'I' + MSB
		= 912 ;	
002A		= 913	STRMOV EQU 42
1162	4D4FD6	= 914	DB 'MO', 'V' + MSB
		= 915 ;	
002B		= 916	STRMVI EQU 43
1165	4D56C9	= 917	DB 'MV', 'I' + MSB
		= 918 ;	
002C		= 919	STRNOP EQU 44
1168	4E4FD0	= 920	DB 'NO', 'P' + MSB
		= 921 ;	
002D		= 922	STRORA EQU 45
116B	4F52C1	= 923	DB 'OR', 'A' + MSB
		= 924 ;	
002E		= 925	STRORI EQU 46
116E	4F52C9	= 926	DB 'OR', 'I' + MSB
		= 927 ;	
002F		= 928	STROUT EQU 47
1171	4F55D4	= 929	DB 'OU', 'T' + MSB
		= 930 ;	
0030		= 931	STRPCHL EQU 48
1174	504348CC	= 932	DB 'PCH', 'L' + MSB
		= 933 ;	
0031		= 934	STRPOP EQU 49
1178	504FD0	= 935	DB 'PO', 'P' + MSB
		= 936 ;	
0032		= 937	STRPUSH EQU 50

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 17

LOC	OBJ	LINE	SOURCE STATEMENT
117B	505553C8	= 938	DB 'PUS', 'H' + MSB
		= 939 ;	
0033		= 940 STRRAL	EQU 51
117F	5241CC	= 941	DB 'RA', 'L' + MSB
		= 942 ;	
0034		= 943 STRRAR	EQU 52
1182	5241D2	= 944	DB 'RA', 'R' + MSB
		= 945 ;	
0035		= 946 STRRLC	EQU 53
1185	524CC3	= 947	DB 'RL', 'C' + MSB
		= 948 ;	
0036		= 949 STRRRC	EQU 54
1188	5252C3	= 950	DB 'RR', 'C' + MSB
		= 951 ;	
0037		= 952 STRRIM	EQU 55
118B	5249CD	= 953	DB 'RI', 'M' + MSB
		= 954 ;	
0038		= 955 STRRET	EQU 56
118E	5245D4	= 956	DB 'RE', 'T' + MSB
		= 957 ;	
0039		= 958 STRRZ	EQU 57
1191	52DA	= 959	DB 'R', 'Z' + MSB
		= 960 ;	
003A		= 961 STRRNZ	EQU 58
1193	524EDA	= 962	DB 'RN', 'Z' + MSB
		= 963 ;	
003B		= 964 STRRP	EQU 59
1196	52D0	= 965	DB 'R', 'P' + MSB
		= 966 ;	
003C		= 967 STRRM	EQU 60
1198	52CD	= 968	DB 'R', 'M' + MSB
		= 969 ;	
003D		= 970 STRRC	EQU 61
119A	52C3	= 971	DB 'R', 'C' + MSB
		= 972 ;	
003E		= 973 STRRNC	EQU 62
119C	524EC3	= 974	DB 'RN', 'C' + MSB
		= 975 ;	
003F		= 976 STRRPE	EQU 63
119F	5250C5	= 977	DB 'RP', 'E' + MSB
		= 978 ;	
0040		= 979 STRRPO	EQU 64
11A2	5250CF	= 980	DB 'RP', 'O' + MSB
		= 981 ;	
0041		= 982 STRRST	EQU 65
11A5	5253D4	= 983	DB 'RS', 'T' + MSB
		= 984 ;	
0042		= 985 STRSIM	EQU 66
11A8	5349CD	= 986	DB 'SI', 'M' + MSB
		= 987 ;	
0043		= 988 STRSPHL	EQU 67
11AB	535048CC	= 989	DB 'SPH', 'L' + MSB
		= 990 ;	
0044		= 991 STRSHLD	EQU 68
11AF	53484CC4	= 992	DB 'SHL', 'D' + MSB
		= 993 ;	
0045		= 994 STRSTA	EQU 69
11B3	5354C1	= 995	DB 'ST', 'A' + MSB
		= 996 ;	
0046		= 997 STRSTX	EQU 70
11B6	535441D8	= 998	DB 'STA', 'X' + MSB
		= 999 ;	
0047		=1000 STRSTC	EQU 71
11BA	5354C3	=1001	DB 'ST', 'C' + MSB
		=1002 ;	
0048		=1003 STRSUB	EQU 72
11BD	5355C2	=1004	DB 'SU', 'B' + MSB
		=1005 ;	
0049		=1006 STRSUI	EQU 73
11C0	5355C9	=1007	DB 'SU', 'I' + MSB

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 18

LOC	OBJ	LINE	SOURCE STATEMENT
		=1008 ;	
004A		=1009 STRSBB EQU	74
11C3	5342C2	=1010 DB	'SB', 'B' + MSB
		=1011 ;	
004B		=1012 STRSBI EQU	75
11C6	5342C9	=1013 DB	'SB', 'I' + MSB
		=1014 ;	
004C		=1015 STRXCHG EQU	76
11C9	584348C7	=1016 DB	'XCH', 'G' + MSB
		=1017 ;	
004D		=1018 STRXTHL EQU	77
11CD	585448CC	=1019 DB	'XTH', 'L' + MSB
		=1020 ;	
004E		=1021 STRXRA EQU	78
11D1	5852C1	=1022 DB	'XR', 'A' + MSB
		=1023 ;	
004F		=1024 STRXRI EQU	79
11D4	5852C9	=1025 DB	'XR', 'I' + MSB
		=1026 ;	
0050		=1027 STRERR EQU	80
11D7	496E7661 6C69E4	=1028 DB	'Invalid', 'd' + MSB
		=1029 ;	
		=1030 ;	
		=1031 ;	
		=1032 ;	Address of emulation routines:
		=1033 ;	
		=1034 ;	
		=1035 ;	
		=1036 EMUTAB:	
		=1037 ;	
0000		=1038 EXECUTE EQU	0 ; Execute (Do not emulate)
		=1039	;
0001		=1040 EMMEM EQU	EXECUTE+1 ; Emulate ArithmeticInstruction M
11DE	9716	=1041 DW	ERMEM ;
		=1042	;
0002		=1043 EMCALL EQU	EMMEM+1 ; Emulate Calls
11E0	CE14	=1044 DW	ERCALL ;
		=1045	;
0003		=1046 EMDCRM EQU	EMCALL+1 ; Emulate DCR M
11E2	0215	=1047 DW	ERDCRM ;
		=1048	;
0004		=1049 EMDI EQU	EMDCRM+1 ; Emulate DI
11E4	2815	=1050 DW	ERDI ;
		=1051	;
0005		=1052 EMEI EQU	EMDI+1 ; Emulate EI
11E6	6715	=1053 DW	EREI ;
		=1054	;
0006		=1055 EMHLT EQU	EMEI+1 ; Emulate HLT
11E8	7B15	=1056 DW	ERHLT ;
		=1057	;
0007		=1058 EMIN EQU	EMHLT+1 ; Emulate IN Data8
11EA	8115	=1059 DW	ERIN ;
		=1060	;
0008		=1061 EMINRM EQU	EMIN+1 ; Emulate INR M
11EC	D115	=1062 DW	ERINRM ;
		=1063	;
0009		=1064 EMJMP EQU	EMINRM+1 ; Emulate JMPs
11EE	F715	=1065 DW	ERJMP ;
		=1066	;
000A		=1067 EMLDA EQU	EMJMP+1 ; Emulate LDA Data16
11F0	1C16	=1068 DW	ERLDA ;
		=1069	;
000B		=1070 EMLDAX EQU	EMLDA+1 ; Emulate LDAX D B
11F2	4016	=1071 DW	ERLDAX ;
		=1072	;
000C		=1073 EMDRM EQU	EMLDAX+1 ; Emulate MOV DestinationRegister,M
11F4	3915	=1074 DW	ERDRM ;
		=1075	;
000D		=1076 EMMSR EQU	EMDRM+1 ; Emulate MOV M,SourceRegister

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 19

LOC	OBJ	LINE	SOURCE STATEMENT
11F6	D216	=1077	DW ERMSR ;
		=1078	; ;
000E		=1079	EMMVIM EQU EMMSR+1 ; Emulate MVI M,Data8
11F8	0017	=1080	DW ERMVIM ;
		=1081	; ;
000F		=1082	EMOUT EQU EMMVIM+1 ; Emulate OUT Data8
11FA	2417	=1083	DW EROUT ;
		=1084	; ;
0010		=1085	EMPCHL EQU EMOUT+1 ; Emulate PCHL
11FC	5617	=1086	DW ERPCHL ;
		=1087	; ;
0011		=1088	EMPOP EQU EMPCHL+1 ; Emulate POP PSW B D H
11FE	6217	=1089	DW ERPOP ;
		=1090	; ;
0012		=1091	EMPUSH EQU EMPOP+1 ; Emulate PUSH PSW B D H
1200	7F17	=1092	DW ERPUSH ;
		=1093	; ;
0013		=1094	EMRIM EQU EMPUSH+1 ; Emulate RIM
1202	C517	=1095	DW ERRIM ;
		=1096	; ;
0014		=1097	EMRET EQU EMRIM+1 ; Emulate RETs
1204	9A17	=1098	DW ERRET ;
		=1099	; ;
0015		=1100	EMRST EQU EMRET+1 ; Emulate RST 0..7
1206	D417	=1101	DW ERRST ;
		=1102	; ;
0016		=1103	EMSIM EQU EMRST+1 ; Emulate SIM
1208	2418	=1104	DW ERSIM ;
		=1105	; ;
0017		=1106	EMSPHL EQU EMSIM+1 ; Emulate SPHL
120A	4718	=1107	DW ERSPHL ;
		=1108	; ;
0018		=1109	EMSHLD EQU EMSPHL+1 ; Emulate SHLD Data16
120C	F417	=1110	DW ERSHLD ;
		=1111	; ;
0019		=1112	EMSTA EQU EMSHLD+1 ; Emulate STA Data16
120E	5618	=1113	DW ERSTA ;
		=1114	; ;
001A		=1115	EMSTAX EQU EMSTA+1 ; Emulate STAX B D
1210	7A18	=1116	DW ERSTAX ;
		=1117	; ;
001B		=1118	EMXTHL EQU EMSTAX+1 ; Emulate XTHL
1212	A418	=1119	DW ERXTHL ;
		=1120	; ;
001C		=1121	EMLHLD EQU EMXTHL+1 ; Emulate LHLD Data16
1214	6A16	=1122	DW ERLHLD ;
		=1123	; ;
001D		=1124	EMERR EQU EMLHLD+1 ; Emulate illegal instructions.
1216	7515	=1125	DW ERERR ;
		=1126 ;	
		=1127 ;	
		=1128 ;	
		=1129 ;	Table D : Instruction table.
		=1130 ;	
		=1131	DISTAB:
		=1132 ;	
		=1133	INST00:
1218	2C00	=1134	DW STRNOP + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; NOP
121A	2904	=1135	DW STRLXI + (OPRRPD16 SHL 7) + (EXECUTE SHL 11) ; LXI B,????
121C	46D3	=1136	DW STRSTX + (OPRRPL SHL 7) + (EMSTAX SHL 11) ; STAX B
121E	9C02	=1137	DW STRINX + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; INX B
1220	1B01	=1138	DW STRINR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; INR B
1222	1501	=1139	DW STRDCR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; DCR B
1224	2B05	=1140	DW STRMVI + (OPRRD8 SHL 7) + (EXECUTE SHL 11) ; MVI B,??
1226	3500	=1141	DW STRRLC + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; RLC
		=1142	INST08:
1228	50E8	=1143	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
122A	9402	=1144	DW STRDAD + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; DAD B
122C	275B	=1145	DW STRLDX + (OPRRPL SHL 7) + (EMLDAX SHL 11) ; LDAX B
122E	9602	=1146	DW STRDCX + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; DCX B

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 20

LOC	OBJ	LINE	SOURCE STATEMENT
1230	1B01	=1147	DW STRINR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; INR C
1232	1501	=1148	DW STRDCR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; DCR C
1234	2B05	=1149	DW STRMVI + (OPRRD8 SHL 7) + (EXECUTE SHL 11) ; MVI C,??
1236	3600	=1150	DW STRRRC + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; RRC
		=1151	INST10:
1238	50E8	=1152	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
123A	2904	=1153	DW STRLXI + (OPRRPD16 SHL 7) + (EXECUTE SHL 11) ; LXI D,????
123C	46D3	=1154	DW STRSTX + (OPRRPL SHL 7) + (EMSTAX SHL 11) ; STAX D
123E	9C02	=1155	DW STRINX + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; INX D
1240	1B01	=1156	DW STRINR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; INR D
1242	1501	=1157	DW STRDCR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; DCR D
1244	2B05	=1158	DW STRMVI + (OPRRD8 SHL 7) + (EXECUTE SHL 11) ; MVI D,??
1246	3300	=1159	DW STRRAL + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; RAL
		=1160	INST18:
1248	50E8	=1161	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
124A	9402	=1162	DW STRDAD + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; DAD D
124C	275B	=1163	DW STRLDX + (OPRRPL SHL 7) + (EMLDAX SHL 11) ; LDAX D
124E	9602	=1164	DW STRDCX + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; DCX D
1250	1B01	=1165	DW STRINR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; INR E
1252	1501	=1166	DW STRDCR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; DCR E
1254	2B05	=1167	DW STRMVI + (OPRRD8 SHL 7) + (EXECUTE SHL 11) ; MVI E,??
1256	3400	=1168	DW STRRAR + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; RAR
		=1169	INST20:
1258	3798	=1170	DW STRRIM + (OPRNONE SHL 7) + (EMRIM SHL 11) ; RIM
125A	2904	=1171	DW STRLXI + (OPRRPD16 SHL 7) + (EXECUTE SHL 11) ; LXI H,????
125C	44C2	=1172	DW STRSHLD + (OPRD16 SHL 7) + (EMSHLD SHL 11) ; SHLD ????
125E	9C02	=1173	DW STRINX + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; INX H
1260	1B01	=1174	DW STRINR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; INR H
1262	1501	=1175	DW STRDCR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; DCR H
1264	2B05	=1176	DW STRMVI + (OPRRD8 SHL 7) + (EXECUTE SHL 11) ; MVI H,??
1266	1300	=1177	DW STRDAA + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; DAA
		=1178	INST28:
1268	50E8	=1179	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
126A	9402	=1180	DW STRDAD + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; DAD H
126C	28E2	=1181	DW STRLHLD + (OPRD16 SHL 7) + (EMLHLD SHL 11) ; LHLD ????
126E	9602	=1182	DW STRDCX + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; DCX H
1270	1B01	=1183	DW STRINR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; INR L
1272	1501	=1184	DW STRDCR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; DCR L
1274	2B05	=1185	DW STRMVI + (OPRRD8 SHL 7) + (EXECUTE SHL 11) ; MVI L,??
1276	0F00	=1186	DW STRCMA + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; CMA
		=1187	INST30:
1278	42B0	=1188	DW STRSIM + (OPRNONE SHL 7) + (EMSIM SHL 11) ; SIM
127A	2904	=1189	DW STRLXI + (OPRRPD16 SHL 7) + (EXECUTE SHL 11) ; LXI SP,????
127C	45CA	=1190	DW STRSTA + (OPRD16 SHL 7) + (EMSTA SHL 11) ; STA ????
127E	9C02	=1191	DW STRINX + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; INX SP
1280	1B41	=1192	DW STRINR + (OPRDREG SHL 7) + (EMINRM SHL 11) ; INR M
1282	1519	=1193	DW STRDCR + (OPRDREG SHL 7) + (EMDCRM SHL 11) ; DCR M
1284	2B75	=1194	DW STRMVI + (OPRRD8 SHL 7) + (EMMVIM SHL 11) ; MVI M,??
1286	4700	=1195	DW STRSTC + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; STC
		=1196	INST38:
1288	50E8	=1197	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
128A	9402	=1198	DW STRDAD + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; DAD SP
128C	2652	=1199	DW STRLDA + (OPRD16 SHL 7) + (EMLDA SHL 11) ; LDA ????
128E	9602	=1200	DW STRDCX + (OPRRPA SHL 7) + (EXECUTE SHL 11) ; DCX SP
1290	1B01	=1201	DW STRINR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; INR A
1292	1501	=1202	DW STRDCR + (OPRDREG SHL 7) + (EXECUTE SHL 11) ; DCR A
1294	2B05	=1203	DW STRMVI + (OPRRD8 SHL 7) + (EXECUTE SHL 11) ; MVI A,??
1296	1000	=1204	DW STRCMC + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; CMC
		=1205	INST40:
1298	AA04	=1206	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV B,B
129A	AA04	=1207	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV B,C
129C	AA04	=1208	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV B,D
129E	AA04	=1209	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV B,E
12A0	AA04	=1210	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV B,H
12A2	AA04	=1211	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV B,L
12A4	AA64	=1212	DW STRMOV + (OPRDSR SHL 7) + (EMDRM SHL 11) ; MOV B,M
12A6	AA04	=1213	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV B,A
		=1214	INST48:
12A8	AA04	=1215	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV C,B
12AA	AA04	=1216	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV C,C

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 21

LOC	OBJ	LINE	SOURCE STATEMENT
12AC	AA04	=1217	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV C,D
12AE	AA04	=1218	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV C,E
12B0	AA04	=1219	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV C,H
12B2	AA04	=1220	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV C,L
12B4	AA64	=1221	DW STRMOV + (OPRDSR SHL 7) + (EMDRM SHL 11) ; MOV C,M
12B6	AA04	=1222	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV C,A
		=1223	INST50:
12B8	AA04	=1224	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV D,B
12BA	AA04	=1225	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV D,C
12BC	AA04	=1226	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV D,D
12BE	AA04	=1227	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV D,E
12C0	AA04	=1228	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV D,H
12C2	AA04	=1229	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV D,L
12C4	AA64	=1230	DW STRMOV + (OPRDSR SHL 7) + (EMDRM SHL 11) ; MOV D,M
12C6	AA04	=1231	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV D,A
		=1232	INST58:
12C8	AA04	=1233	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV E,B
12CA	AA04	=1234	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV E,C
12CC	AA04	=1235	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV E,D
12CE	AA04	=1236	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV E,E
12D0	AA04	=1237	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV E,H
12D2	AA04	=1238	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV E,L
12D4	AA64	=1239	DW STRMOV + (OPRDSR SHL 7) + (EMDRM SHL 11) ; MOV E,M
12D6	AA04	=1240	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV E,A
		=1241	INST60:
12D8	AA04	=1242	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV H,B
12DA	AA04	=1243	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV H,C
12DC	AA04	=1244	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV H,D
12DE	AA04	=1245	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV H,E
12E0	AA04	=1246	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV H,H
12E2	AA04	=1247	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV H,L
12E4	AA64	=1248	DW STRMOV + (OPRDSR SHL 7) + (EMDRM SHL 11) ; MOV H,M
12E6	AA04	=1249	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV H,A
		=1250	INST68:
12E8	AA04	=1251	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV L,B
12EA	AA04	=1252	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV L,C
12EC	AA04	=1253	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV L,D
12EE	AA04	=1254	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV L,E
12F0	AA04	=1255	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV L,H
12F2	AA04	=1256	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV L,L
12F4	AA64	=1257	DW STRMOV + (OPRDSR SHL 7) + (EMDRM SHL 11) ; MOV L,M
12F6	AA04	=1258	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV L,A
		=1259	INST70:
12F8	AA6C	=1260	DW STRMOV + (OPRDSR SHL 7) + (EMMSR SHL 11) ; MOV M,B
12FA	AA6C	=1261	DW STRMOV + (OPRDSR SHL 7) + (EMMSR SHL 11) ; MOV M,C
12FC	AA6C	=1262	DW STRMOV + (OPRDSR SHL 7) + (EMMSR SHL 11) ; MOV M,D
12FE	AA6C	=1263	DW STRMOV + (OPRDSR SHL 7) + (EMMSR SHL 11) ; MOV M,E
1300	AA6C	=1264	DW STRMOV + (OPRDSR SHL 7) + (EMMSR SHL 11) ; MOV M,H
1302	AA6C	=1265	DW STRMOV + (OPRDSR SHL 7) + (EMMSR SHL 11) ; MOV M,L
1304	1930	=1266	DW STRHLT + (OPRNONE SHL 7) + (EMHLT SHL 11) ; HLT
1306	AA6C	=1267	DW STRMOV + (OPRDSR SHL 7) + (EMMSR SHL 11) ; MOV M,A
		=1268	INST78:
1308	AA04	=1269	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV A,B
130A	AA04	=1270	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV A,C
130C	AA04	=1271	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV A,D
130E	AA04	=1272	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV A,E
1310	AA04	=1273	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV A,H
1312	AA04	=1274	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV A,L
1314	AA64	=1275	DW STRMOV + (OPRDSR SHL 7) + (EMDRM SHL 11) ; MOV A,M
1316	AA04	=1276	DW STRMOV + (OPRDSR SHL 7) + (EXECUTE SHL 11) ; MOV A,A
		=1277	INST80:
1318	8000	=1278	DW STRADD + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADD B
131A	8000	=1279	DW STRADD + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADD C
131C	8000	=1280	DW STRADD + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADD D
131E	8000	=1281	DW STRADD + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADD E
1320	8000	=1282	DW STRADD + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADD H
1322	8000	=1283	DW STRADD + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADD L
1324	8008	=1284	DW STRADD + (OPRSREG SHL 7) + (EMMEM SHL 11) ; ADD M
1326	8000	=1285	DW STRADD + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADD A
		=1286	INST88:

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 22

LOC	OBJ	LINE	SOURCE STATEMENT
1328	8200	=1287	DW STRADC + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADC B
132A	8200	=1288	DW STRADC + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADC C
132C	8200	=1289	DW STRADC + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADC D
132E	8200	=1290	DW STRADC + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADC E
1330	8200	=1291	DW STRADC + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADC H
1332	8200	=1292	DW STRADC + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADC L
1334	8208	=1293	DW STRADC + (OPRSREG SHL 7) + (EMMEM SHL 11) ; ADC M
1336	8200	=1294	DW STRADC + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ADC A
		=1295	INST90:
1338	C800	=1296	DW STRSUB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SUB B
133A	C800	=1297	DW STRSUB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SUB C
133C	C800	=1298	DW STRSUB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SUB D
133E	C800	=1299	DW STRSUB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SUB E
1340	C800	=1300	DW STRSUB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SUB H
1342	C800	=1301	DW STRSUB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SUB L
1344	C808	=1302	DW STRSUB + (OPRSREG SHL 7) + (EMMEM SHL 11) ; SUB M
1346	C800	=1303	DW STRSUB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SUB A
		=1304	INST98:
1348	CA00	=1305	DW STRSBB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SBB B
134A	CA00	=1306	DW STRSBB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SBB C
134C	CA00	=1307	DW STRSBB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SBB D
134E	CA00	=1308	DW STRSBB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SBB E
1350	CA00	=1309	DW STRSBB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SBB H
1352	CA00	=1310	DW STRSBB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SBB L
1354	CA08	=1311	DW STRSBB + (OPRSREG SHL 7) + (EMMEM SHL 11) ; SBB M
1356	CA00	=1312	DW STRSBB + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; SBB A
		=1313	INSTA0:
1358	8400	=1314	DW STRANA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ANA B
135A	8400	=1315	DW STRANA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ANA C
135C	8400	=1316	DW STRANA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ANA D
135E	8400	=1317	DW STRANA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ANA E
1360	8400	=1318	DW STRANA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ANA H
1362	8400	=1319	DW STRANA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ANA L
1364	8408	=1320	DW STRANA + (OPRSREG SHL 7) + (EMMEM SHL 11) ; ANA M
1366	8400	=1321	DW STRANA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ANA A
		=1322	INSTA8:
1368	CE00	=1323	DW STRXRA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; XRA B
136A	CE00	=1324	DW STRXRA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; XRA C
136C	CE00	=1325	DW STRXRA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; XRA D
136E	CE00	=1326	DW STRXRA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; XRA E
1370	CE00	=1327	DW STRXRA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; XRA H
1372	CE00	=1328	DW STRXRA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; XRA L
1374	CE08	=1329	DW STRXRA + (OPRSREG SHL 7) + (EMMEM SHL 11) ; XRA M
1376	CE00	=1330	DW STRXRA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; XRA A
		=1331	INSTB0:
1378	AD00	=1332	DW STORA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ORA B
137A	AD00	=1333	DW STORA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ORA C
137C	AD00	=1334	DW STORA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ORA D
137E	AD00	=1335	DW STORA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ORA E
1380	AD00	=1336	DW STORA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ORA H
1382	AD00	=1337	DW STORA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ORA L
1384	AD08	=1338	DW STORA + (OPRSREG SHL 7) + (EMMEM SHL 11) ; ORA M
1386	AD00	=1339	DW STORA + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; ORA A
		=1340	INSTB8:
1388	9100	=1341	DW STRCMP + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; CMP B
138A	9100	=1342	DW STRCMP + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; CMP C
138C	9100	=1343	DW STRCMP + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; CMP D
138E	9100	=1344	DW STRCMP + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; CMP E
1390	9100	=1345	DW STRCMP + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; CMP H
1392	9100	=1346	DW STRCMP + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; CMP L
1394	9108	=1347	DW STRCMP + (OPRSREG SHL 7) + (EMMEM SHL 11) ; CMP M
1396	9100	=1348	DW STRCMP + (OPRSREG SHL 7) + (EXECUTE SHL 11) ; CMP A
		=1349	INSTC0:
1398	JAA0	=1350	DW STRRNZ + (OPRNONE SHL 7) + (EMRET SHL 11) ; RNZ
139A	B18B	=1351	DW STRPOP + (OPRRPS SHL 7) + (EMPOP SHL 11) ; POP B
139C	1F4A	=1352	DW STRJNZ + (OPRD16 SHL 7) + (EMJMP SHL 11) ; JNZ ????
139E	1D4A	=1353	DW STRJMP + (OPRD16 SHL 7) + (EMJMP SHL 11) ; JMP ????
13A0	0812	=1354	DW STRCNZ + (OPRD16 SHL 7) + (EMCALL SHL 11) ; CNZ ????
13A2	B293	=1355	DW STRPUSH + (OPRRPS SHL 7) + (EMPUSH SHL 11) ; PUSH B
13A4	8101	=1356	DW STRADI + (OPRD8 SHL 7) + (EXECUTE SHL 11) ; ADI ??

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 23

LOC	OBJ	LINE	SOURCE STATEMENT
13A6	C1AD	=1357	DW STRRST + (OPRRST SHL 7) + (EMRST SHL 11) ; RST 0
		=1358	INSTC8:
13A8	39A0	=1359	DW STRRZ + (OPRNONE SHL 7) + (EMRET SHL 11) ; RZ
13AA	38A0	=1360	DW STRRET + (OPRNONE SHL 7) + (EMRET SHL 11) ; RET
13AC	1E4A	=1361	DW STRJZ + (OPRD16 SHL 7) + (EMJMP SHL 11) ; JZ ????
13AE	50E8	=1362	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
13B0	0712	=1363	DW STRCZ + (OPRD16 SHL 7) + (EMCALL SHL 11) ; CZ ????
13B2	0612	=1364	DW STRCALL + (OPRD16 SHL 7) + (EMCALL SHL 11) ; CALL ????
13B4	8301	=1365	DW STRACI + (OPRD8 SHL 7) + (EXECUTE SHL 11) ; ACI ??
13B6	C1AD	=1366	DW STRRST + (OPRRST SHL 7) + (EMRST SHL 11) ; RST 1
		=1367	INSTD0:
13B8	3EA0	=1368	DW STRRNC + (OPRNONE SHL 7) + (EMRET SHL 11) ; RNC
13BA	B18B	=1369	DW STRPOP + (OPRRPS SHL 7) + (EMPOP SHL 11) ; POP D
13BC	234A	=1370	DW STRJNC + (OPRD16 SHL 7) + (EMJMP SHL 11) ; JNC ????
13BE	AF79	=1371	DW STRROUT + (OPRD8 SHL 7) + (EMOUT SHL 11) ; OUT ??
13C0	0C12	=1372	DW STRCNC + (OPRD16 SHL 7) + (EMCALL SHL 11) ; CNC ????
13C2	B293	=1373	DW STRPUSH + (OPRRPS SHL 7) + (EMPUSH SHL 11) ; PUSH D
13C4	C901	=1374	DW STRSUI + (OPRD8 SHL 7) + (EXECUTE SHL 11) ; SUI ??
13C6	C1AD	=1375	DW STRRST + (OPRRST SHL 7) + (EMRST SHL 11) ; RST 2
		=1376	INSTD8:
13C8	3DA0	=1377	DW STRRC + (OPRNONE SHL 7) + (EMRET SHL 11) ; RC
13CA	50E8	=1378	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
13CC	224A	=1379	DW STRJC + (OPRD16 SHL 7) + (EMJMP SHL 11) ; JC ????
13CE	9A39	=1380	DW STRIN + (OPRD8 SHL 7) + (EMIN SHL 11) ; IN ??
13D0	0B12	=1381	DW STRCC + (OPRD16 SHL 7) + (EMCALL SHL 11) ; CC ????
13D2	50E8	=1382	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
13D4	CB01	=1383	DW STRSBI + (OPRD8 SHL 7) + (EXECUTE SHL 11) ; SBI ??
13D6	C1AD	=1384	DW STRRST + (OPRRST SHL 7) + (EMRST SHL 11) ; RST 3
		=1385	INSTE0:
13D8	40A0	=1386	DW STRRPO + (OPRNONE SHL 7) + (EMRET SHL 11) ; RPO
13DA	B18B	=1387	DW STRPOP + (OPRRPS SHL 7) + (EMPOP SHL 11) ; POP H
13DC	254A	=1388	DW STRJPO + (OPRD16 SHL 7) + (EMJMP SHL 11) ; JPO ????
13DE	4DD8	=1389	DW STRXTHL + (OPRNONE SHL 7) + (EMXTHL SHL 11) ; XTHL
13E0	0E12	=1390	DW STRCPO + (OPRD16 SHL 7) + (EMCALL SHL 11) ; CPO ????
13E2	B293	=1391	DW STRPUSH + (OPRRPS SHL 7) + (EMPUSH SHL 11) ; PUSH H
13E4	8501	=1392	DW STRANI + (OPRD8 SHL 7) + (EXECUTE SHL 11) ; ANI ??
13E6	C1AD	=1393	DW STRRST + (OPRRST SHL 7) + (EMRST SHL 11) ; RST 4
		=1394	INSTE8:
13E8	3FA0	=1395	DW STRRPE + (OPRNONE SHL 7) + (EMRET SHL 11) ; RPE
13EA	3080	=1396	DW STRPCHL + (OPRNONE SHL 7) + (EMPCHL SHL 11) ; PCHL
13EC	244A	=1397	DW STRJPE + (OPRD16 SHL 7) + (EMJMP SHL 11) ; JPE ????
13EE	4C00	=1398	DW STRXCHG + (OPRNONE SHL 7) + (EXECUTE SHL 11) ; XCHG
13F0	0D12	=1399	DW STRCPE + (OPRD16 SHL 7) + (EMCALL SHL 11) ; CPE ????
13F2	50E8	=1400	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
13F4	CF01	=1401	DW STRXRI + (OPRD8 SHL 7) + (EXECUTE SHL 11) ; XRI ??
13F6	C1AD	=1402	DW STRRST + (OPRRST SHL 7) + (EMRST SHL 11) ; RST 5
		=1403	INSTF0:
13F8	3BA0	=1404	DW STRRP + (OPRNONE SHL 7) + (EMRET SHL 11) ; RP
13FA	B18B	=1405	DW STRPOP + (OPRRPS SHL 7) + (EMPOP SHL 11) ; POP PSW
13FC	204A	=1406	DW STRJJP + (OPRD16 SHL 7) + (EMJMP SHL 11) ; JP ????
13FE	1720	=1407	DW STRDI + (OPRNONE SHL 7) + (EMDI SHL 11) ; DI
1400	0912	=1408	DW STRCP + (OPRD16 SHL 7) + (EMCALL SHL 11) ; CP ????
1402	B293	=1409	DW STRPUSH + (OPRRPS SHL 7) + (EMPUSH SHL 11) ; PUSH PSW
1404	AE01	=1410	DW STRORI + (OPRD8 SHL 7) + (EXECUTE SHL 11) ; ORI ??
1406	C1AD	=1411	DW STRRST + (OPRRST SHL 7) + (EMRST SHL 11) ; RST 6
		=1412	INSTF8:
1408	3CA0	=1413	DW STRRM + (OPRNONE SHL 7) + (EMRET SHL 11) ; RM
140A	43B8	=1414	DW STRSPHL + (OPRNONE SHL 7) + (EMSPHL SHL 11) ; SPHL
140C	214A	=1415	DW STRJMP + (OPRD16 SHL 7) + (EMJMP SHL 11) ; JM ????
140E	1828	=1416	DW STREI + (OPRNONE SHL 7) + (EMEI SHL 11) ; EI
1410	0A12	=1417	DW STRCM + (OPRD16 SHL 7) + (EMCALL SHL 11) ; CM ????
1412	50E8	=1418	DW STRERR + (OPRNONE SHL 7) + (EMERR SHL 11) ; Invalid
1414	9201	=1419	DW STRCPI + (OPRD8 SHL 7) + (EXECUTE SHL 11) ; CPI ??
1416	C1AD	=1420	DW STRRST + (OPRRST SHL 7) + (EMRST SHL 11) ; RST 7
		=1421 ;	
		=1422 ;	END <DEBTABLE.A85>
		1423	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 24

LOC	OBJ	LINE	SOURCE STATEMENT
1424			;
1425			Emulation routines:
1426			;
1427			\$INCLUDE(DEBEMUL.A85)
=1428			-----;
=1429			;
=1430			Emulate routine used by the UCT SABus debugger.
=1431			;
=1432			This module contains (in order) the following subroutines.
=1433			;
=1434			a) D16FETCH - Data 16 fetch routine. (e.g. CALL ????)
=1435			b) D8FETCH - Data 8 fetch routine. (e.g. MVI A,??)
=1436			c) DECIDE - Decide on condition.
=1437			d) DOPOP - Perform POP.
=1438			e) DOPUSH - Perform PUSH.
=1439			f) GETINDIRECT - Fetch 16 bit value indirect via HL.
=1440			g) GETRA - Get address of register storage.
=1441			h) GETRPA - Get Address of register pair storage.
=1442			i) PUTINDIRECT - Put 16 bit indirect via HL register pair.
=1443			j) ERCALL - Emulate routine for CALL instruction.
=1444			;
=1445			k) ERDCRM - Emulate routine for DCR M instruction.
=1446			l) ERDI - Emulate routine for DI instruction.
=1447			m) ERDRM - Emulate routine for MOV Reg,M instruction.
=1448			n) EREI - Emulate routine for EI instruction.
=1449			o) ERERR - Emulate routine for illegal instructions
=1450			p) ERHLT - Emulate routine for HLT instruction.
=1451			q) ERIN - Emulate routine for IN ?? instruction.
=1452			r) ERINRM - Emulate routine for INR M instruction.
=1453			s) ERJMP - Emulate routine for JMP ???? instruction.
=1454			t) ERLDA - Emulate routine for LDA ???? instruction.
=1455			;
=1456			s) ERLDAX - Emulate routine for LDAX instruction.
=1457			v) ERLHLD - Emulate routine for LHLD ???? instruction.
=1458			w) ERMEM - Emulate routine for INST M instruction.
=1459			x) ERMSR - Emulate routine for MOV M,Reg instructions.
=1460			y) ERMVIM - Emulate routine for MVI M,?? instructions.
=1461			z) EROUT - Emulate routine for OUT ?? instructions.
=1462			aa) ERPCHL - Emulate routine for PCHL instruction.
=1463			bb) ERPOP - Emulate routine for POP instruction.
=1464			cc) ERPUSH - Emulate routine for PUSH instruction.
=1465			dd) ERRET - Emulate routine for RET instruction.
=1466			;
=1467			ee) ERRIM - Emulate routine for RIM instruction.
=1468			ff) ERRST - Emulate routine for RST ? instructions.
=1469			gg) ERSHLD - Emulate routine for SHLD ???? instruction.
=1470			hh) ERSIM - Emulate routine for SIM instruction.
=1471			ii) ERSPLH - Emulate routine for SPL instruction.
=1472			jj) ERSTA - Emulate routine for STA ???? instruction.
=1473			kk) ERSTAX - Emulate routine for STAX instruction.
=1474			ll) ERXTHL - Emulate routine for XTHL instruction.
=1475			;
=1476			-----;
=1477			\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 25

LOC	OBJ	LINE	SOURCE STATEMENT
		=1478 ;	=====*
		=1479 ;*	*
		=1480 ;*	Function : D16FETCH (Data 16 fetch).
		=1481 ;*	Inputs : (REGPC) : Current program counter.
		=1482 ;*	Outputs : HL : 16 bit operand.
		=1483 ;*	Destroys : A, H, L and flags.
		=1484 ;*	Calls : GETINDIRECT.
		=1485 ;*	*
		=1486 ;*	Description : This routine fetches the 16 bit operand following
		=1487 ;*	three byte instruction.
		=1488 ;*	*
		=1489 ;*	Note: The 8085 stores 16 bit words in reverse order, i.e. least
		=1490 ;*	significant byte first.
		=1491 ;*	*
		=1492 ;	=====*
		=1493 ;	
		=1494	D16FETCH: ; Procedure Data16Fetch;
1418	2A9DEA	=1495	LHLD REGPC ;
141B	23	=1496	INX H ;
141C	CD9414	=1497	CALL GETINDIRECT ;
		=1498	; Return((Memory[RegPC+1]) +
		=1499	; (Memory[RegPC+2] shl 8));
		=1500	; EndProc Data16Fetch;
141F	C9	=1501	RET ; EndProc Data16Fetch;
		=1502 ;	
		=1503 ;	=====*
		=1504 ;*	*
		=1505 ;*	Function : D8FETCH (Data 8 fetch).
		=1506 ;*	Inputs : (REGPC) : Current value of the program counter.
		=1507 ;*	Outputs : A : 8 bit operand.
		=1508 ;*	Destroys : A and flags.
		=1509 ;*	Calls : None.
		=1510 ;*	*
		=1511 ;*	Description : This routine fetches the operand following two byte
		=1512 ;*	instructions.
		=1513 ;*	*
		=1514 ;	=====*
		=1515 ;	
		=1516	D8FETCH: ; Procedure Data8Fetch;
1420	E5	=1517	PUSH H ;
1421	2A9DEA	=1518	LHLD REGPC ;
1424	23	=1519	INX H ;
1425	7E	=1520	MOV A,M ; Return(Memory[RegPC+1]);
1426	E1	=1521	POP H ;
1427	C9	=1522	RET ; EndProc Data8Fetch;
		=1523 ;	
		=1524 ;	=====*
		=1525 ;*	*
		=1526 ;*	Function : DECIDE (Decide if a condition is true).
		=1527 ;*	Inputs : A : Conditional instruction opcode.
		=1528 ;*	Outputs : A = TRUE : Condition is true.
		=1529 ;*	A = FALSE : Condition is false.
		=1530 ;*	Destroys : A and flags.
		=1531 ;*	Calls : DECROU.
		=1532 ;*	*
		=1533 ;*	Description : This routine uses a short dynamic routine to decide
		=1534 ;*	if the condition passed in the opcode is true or not. The process
		=1535 ;*	used to decide if the condition is true is explained below:
		=1536 ;*	*
		=1537 ;*	a) Convert the condition into a conditional return instruction.
		=1538 ;*	b) Create the following short routine (in RAM)
		=1539 ;*	*
		=1540 ;*	DecideRoutine: (Condition return created in section a)
		=1541 ;*	CMA
		=1542 ;*	RET
		=1543 ;*	*
		=1544 ;*	c) Restore the flags register.
		=1545 ;*	d) Set the A register equal to TRUE.
		=1546 ;*	e) Call the decideRoutine.
		=1547 ;*	*

H-39

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 27

```

LOC   OBJ       LINE      SOURCE STATEMENT
1459  CDE52F     =1618
                      =1619      CALL    TBWRW      ;
                      =1620                        ;      TraceBufferWriteWord(RegSP);
145C   5E        =1621      MOV     E,M          ;
145D   23        =1622      INX     H            ;      LSB := Memory[RegSP];
145E   56        =1623      MOV     D,M          ;
145F   23        =1624      INX     H            ;      MSB := Memory[RegSP+1];
1460  229FEA     =1625      SHLD    REGSP         ;
1463  EB         =1626      XCHG                     ;      RegSP := RegSP + 2;
                      =1627                        ;
                      =1628                        ;      Value := LSB + 256 * MSG;
1464  CDE52F     =1629      CALL    TBWRW      ;
1467  3E01       =1630      MVI     A,1          ;      TraceBufferWriteWord(Value);
1469  B7         =1631      ORA     A            ;
                      =1632                        ;      Status := OK
                      =1633      DOP010:         ;      End
                      =1634                        ;      Else
                      =1635                        ;      Status := Error;
146A  C9         =1636      RET                     ; EndProc DoPOP;
                      =1637 ;
                      =1638 ;=====
                      =1639 ;*
                      =1640 ;* Function : DOPUSH (Perform PUSH onto stack). *
                      =1641 ;* Inputs   : HL      : 16 bit word to push onto the stack. *
                      =1642 ;*          : (REGSP) : Current processor stack pointer. *
                      =1643 ;* Outputs  : Z = 1 : Protection error. *
                      =1644 ;*          : Z = 0 : Push performed correctly. *
                      =1645 ;* Destroys : A, D, E, H, L and flags. *
                      =1646 ;* Calls   : CHKMWR. *
                      =1647 ;*
                      =1648 ;* Description : This routine pushes a 16 bit value onto the user *
                      =1649 ;* stack. The memory protection map is checked. *
                      =1650 ;*
                      =1651 ;*
                      =1652 ;* A diagram of the stack is given below: *
                      =1653 ;*
                      =1654 ;* Higher address |-----| <- Initial SP *
                      =1655 ;* |-----| *
                      =1656 ;* | Most significant byte | *
                      =1657 ;* |-----| *
                      =1658 ;* | Least significant byte | <- New SP *
                      =1659 ;* |-----| *
                      =1660 ;* Lower address | *
                      =1661 ;*
                      =1662 ;=====
                      =1663 ;
                      =1664 DOPUSH: ; Procedure DoPUSH(DataWord:Integer);
146B  EB         =1665      XCHG                     ;
146C  2A9FEA     =1666      LHLD    REGSP         ;
146F  2B         =1667      DCX     H            ;
1470  CD6610     =1668      CALL    CHKMWR        ;
1473  CA9314     =1669      JZ      DOPU80         ;      If CheckMemoryWrite(RegSP-1) AND
                      =1670                        ;
1476  2B         =1671      DCX     H            ;
1477  CD6610     =1672      CALL    CHKMWR        ;
147A  CA9314     =1673      JZ      DOPU80         ;      CheckMemoryWrite(RegSP-2) then begin
                      =1674                        ;
147D  23         =1675      INX     H            ;
147E  72         =1676      MOV     M,D          ;      Memory[RegSP-1] := High dataword;
147F  2B         =1677      DCX     H            ;
1480  73         =1678      MOV     M,E          ;      Memory[RegSP-2] := Low dataword;
                      =1679                        ;
1481  3E05       =1680      MVI     A,EX16BWR      ;
1483  CD662F     =1681      CALL    TBWREX        ;      TraceBufferWriteExecuted
                      =1682                        ;      (16BitWrite);
                      =1683                        ;
1486  CDE52F     =1684      CALL    TBWRW      ;      TraceBufferWriteWord(RegSP);
                      =1685                        ;
1489  229FEA     =1686      SHLD    REGSP         ;      RegSP := RegSP - 2;
148C  EB         =1687      XCHG                     ;

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 28

LOC	OBJ	LINE	SOURCE STATEMENT
148D	CDE52F	=1688	CALL TBWRW ; TraceBufferWriteWord(DataWord);
1490	3E01	=1689	MVI A,1 ;
1492	B7	=1690	ORA A ; Status := OK;
		=1691	; End
		=1692	; Else
		=1693	DOPU80: ; Status := Error;
		=1694	; Return(Status)
		=1695	; Return(Status)
		=1696	; EndProc DoPUSH;
1493	C9	=1697	RET ; EndProc DoPUSH;
		=1698	;
		=1699	=====*
		=1700	;
		=1701	;* Function : GETINDIRECT (Get 16 bit data indirect) *
		=1702	;* Inputs : HL : Pointer word in memory. *
		=1703	;* Outputs : HL : Value read from memory. *
		=1704	;* Destroys : H, L and flags. *
		=1705	;* Calls : None. *
		=1706	;* *
		=1707	;* Description : This routine fetches a 16 bit value from memory. The *
		=1708	;* HL register pair contains the address on the first byte. *
		=1709	;* *
		=1710	;* Note: The 8085 stores word (16 bit values) in reverse order - i.e. *
		=1711	;* least significant byte followed by the most significant byte *
		=1712	;* *
		=1713	=====*
		=1714	;
		=1715	GETINDIRECT: ; Procedure GetIndirect(Ptr:Integer);
1494	D5	=1716	PUSH D ;
1495	5E	=1717	MOV E,M ; LSB := memory[Ptr];
1496	23	=1718	INX H ;
1497	56	=1719	MOV D,M ; MSB := memory[Ptr+1]
1498	EB	=1720	XCHG ;
		=1721	; Return(MSB*256+LSB);
1499	D1	=1722	POP D ;
149A	C9	=1723	RET ; EndProc GetIndirect;
		=1724	;
		=1725	=====*
		=1726	;
		=1727	;* Function : GETRA (Get register address). *
		=1728	;* Inputs : A : Register number (See below) *
		=1729	;* Outputs : HL : Value of the register storage. *
		=1730	;* Destroys : H, L and flags. *
		=1731	;* Calls : None. *
		=1732	;* *
		=1733	;* Description : This routine returns a pointer to the register *
		=1734	;* storage byte. The routine uses a lookup table and simply indexes *
		=1735	;* into the table by the amount given by the register number to *
		=1736	;* obtain the register storage address. *
		=1737	;* *
		=1738	=====*
		=1739	;
		=1740	A : Register number.
		=1741	;
		=1742	000 = B 001 = C 010 = D 011 = E
		=1743	100 = H 101 = L 110 = M (***) 111 = A
		=1744	;
		=1745	(***) This value cannot be used as M is a special case.
		=1746	;
		=1747	GETRA: ; Procedure GetRegisterAddress(Opcode:Byte);
149B	D5	=1748	PUSH D ;
149C	6F	=1749	MOV L,A ;
149D	2600	=1750	MVI H,0 ; Ptr := A
149F	29	=1751	DAD H ; Ptr := Ptr * 2
14A0	11A914	=1752	LXI D,GRATAB ;
14A3	19	=1753	DAD D ; Ptr := Ptr * 2 + GRATAB;
		=1754	;
14A4	CD9414	=1755	CALL GETINDIRECT ; RegAdr := GetIndirect(Ptr);
14A7	D1	=1756	POP D ;
14A8	C9	=1757	RET ; EndProc GetRegisterAddress;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SAbus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 29

LOC	OBJ	LINE	SOURCE STATEMENT
		=1758	;
		=1759	GRATAB: ;
14A9	96EA	=1760	DW REGB ; RegNum : 000
14AB	95EA	=1761	DW REGC ; RegNum : 001
14AD	98EA	=1762	DW REGD ; RegNum : 010
14AF	97EA	=1763	DW REGE ; RegNum : 011
14B1	9AEA	=1764	DW REGH ; RegNum : 100
14B3	99EA	=1765	DW REGL ; RegNum : 101
14B5	0000	=1766	DW 0 ; (***) Not used : 110
14B7	9CEA	=1767	DW REGA ; RegNum : 111
		=1768 ;	
		=1769 ;	=====*
		=1770 ;*	*
		=1771 ;*	Function : GETRPA (Get register pair address) *
		=1772 ;*	Inputs : A : Opcode containing a register pair specification. *
		=1773 ;*	Outputs : HL : Address of the word memory variable *
		=1774 ;*	Destroys : A, H, L and flags. *
		=1775 ;*	Calls : None. *
		=1776 ;*	*
		=1777 ;*	Description : This routine uses the fact that the register pairs *
		=1778 ;*	are declared in the order given below to obtain the register *
		=1779 ;*	address by simply adding a constant to twice the register number. *
		=1780 ;*	*
		=1781 ;	=====*
		=1782 ;	
		=1783 ;	Position of Register pair specification in Instruction:
		=1784 ;	
		=1785 ;	xxRRxxxx : Where RR = Register pair number
		=1786 ;	
		=1787 ;	RR = 00 : Register pair BC
		=1788 ;	RR = 01 : Register pair DE
		=1789 ;	RR = 10 : Register pair HL
		=1790 ;	RR = 11 : Register pair PSW
		=1791 ;	
		=1792	GETRPA: ; Procedure GetRegisterPairAddress(Opcode:Byte);
14B9	D5	=1793	PUSH D ;
14BA	E630	=1794	ANI 00110000B ;
14BC	0F	=1795	RRC ;
14BD	0F	=1796	RRC ;
14BE	0F	=1797	RRC ;
14BF	0F	=1798	RRC ; RegNum := (Opcode AND 00110000B) shr 4;
14C0	6F	=1799	MOV L,A ;
14C1	2600	=1800	MVI H,0 ;
14C3	29	=1801	DAD H ; RegOffset := RegNum * 2
14C4	1195EA	=1802	LXI D,REGISTERS ;
14C7	19	=1803	DAD D ; RegAddress := RegOffset + Registers;
14C8	D1	=1804	POP D ;
14C9	C9	=1805	RET ; EndProc GetRegisterPairAddress;
		=1806 ;	
		=1807 ;	=====*
		=1808 ;*	*
		=1809 ;*	Function : PUTINDIRECT (Put 16 bit value indirect). *
		=1810 ;*	Inputs : HL : Address of word storage. *
		=1811 ;*	DE : Value to be stored. *
		=1812 ;*	Outputs : None. *
		=1813 ;*	Destroys : H, L and flags. *
		=1814 ;*	Calls : None. *
		=1815 ;*	*
		=1816 ;*	Description : This routine stores the word passed in DE in the word *
		=1817 ;*	variable whose address is passed in the HL register pair. *
		=1818 ;*	*
		=1819 ;	=====*
		=1820 ;	
		=1821 ;	Adr : HL
		=1822 ;	Data : DE
		=1823 ;	
		=1824	PUTINDIRECT: ; Procedure PutIndirect(Adr, Data : Integer);
		=1825	;
14CA	73	=1826	MOV M,E ; Memory[Adr] := LOW Data;
14CB	23	=1827	INX H ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 30

LOC	OBJ	LINE	SOURCE STATEMENT
14CC	72	=1828	MOV M,D ; Memory[Adr+1] := HIGH Data;
		=1829	;
14CD	C9	=1830	RET ; EndProc PutIndirect;
		=1831 ;	
		=1832 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 31

LOC	OBJ	LINE	SOURCE STATEMENT
		=1833 ;	-----;
		=1834 ;	;
		=1835 ;	EMULATE ROUTINES:
		=1836 ;	;
		=1837 ;	-----;
		=1838 ;	;
		=1839 ;	=====*
		=1840 ;*	*
		=1841 ;*	Function : ERCALL (Emulate routine for the CALL instruction).
		=1842 ;*	Inputs : Instruction and register set in memory.
		=1843 ;*	Outputs : Updated register set.
		=1844 ;*	Destroys : A, H, L and flags.
		=1845 ;*	Calls : DECIDE, INPCP, ADDBC, D16FETCH and DOPUSH.
		=1846 ;*	*
		=1847 ;*	Description : This routine emulates the CALL instructions. The call
		=1848 ;*	is performed if the opcode is an unconditional call or the result
		=1849 ;*	of the decide routine. The decide routine uses the three bit
		=1850 ;*	condition contained in the conditional call instruction and the
		=1851 ;*	flags register and returns TRUE if the conditional is true and the
		=1852 ;*	call should therefore be executed.
		=1853 ;*	*
		=1854 ;*	To perform a call instruction the routine pushes the address of the
		=1855 ;*	instruction following the call instruction onto the stack. The
		=1856 ;*	program counter is then loaded with the value of the operand.
		=1857 ;*	*
		=1858 ;*	Memory reference type: 16 bit write. (Done by DOPUSH routine).
		=1859 ;*	*
		=1860 ;	=====*
		=1861 ;	;
		=1862 ;	This routine must emulate the following instructions:
		=1863 ;	;
		=1864 ;	CALL ???? ;
		=1865 ;	CNZ ???? CZ ???? CNC ???? CC ???? ;
		=1866 ;	CPO ???? CPE ???? CP ???? CM ???? ;
		=1867 ;	;
		=1868 ERCALL:	; Procedure EmulateCall;
14CE	3AA8EA	=1869 LDA	OPCODE ;
14D1	FECF	=1870 CPI	(CALL) ;
14D3	CAE714	=1871 JZ	ERC40 ; If (Opcode <> (CALL)) or
		=1872 ;	;
14D6	CD2814	=1873 CALL	DECIDE ; (Decide(Opcode)) then
14D9	CAE714	=1874 JZ	ERC40 ;
		=1875 ;	Begin
14DC	3E01	=1876 MVI	A,EXNOREF ;
14DE	CD662F	=1877 CALL	TBWREX ; TraceBufferWriteExecuted(NoRef);
		=1878 ;	;
14E1	CDD021	=1879 CALL	INPCP ; IncrementPC
14E4	C30115	=1880 JMP	ERC80 ; End
		=1881 ERC40:	; Else
		=1882 ;	Begin
14E7	2A9DEA	=1883 LHLD	REGPC ;
14EA	CDC621	=1884 CALL	ADDBC ; AddByteCount;
14ED	CD6B14	=1885 CALL	DOPUSH ; If DoPUSH(RegPC) then
14F0	CAFC14	=1886 JZ	ERC50 ;
		=1887 ;	;
14F3	CD1814	=1888 CALL	D16FETCH ;
14F6	229DEA	=1889 SHLD	REGPC ; RegPC := Data16Fetch;
14F9	C30115	=1890 JMP	ERC80 ;
		=1891 ERC50:	; Else
14FC	3E04	=1892 MVI	A,GUARDACCESS ;
14FE	32B2EA	=1893 STA	STATUS ; Status := GuardedAccess;
		=1894 ERC80:	; End
1501	C9	=1895 RET	; EndProc EmulateCall;
		=1896 ;	;
		=1897 ;	=====*
		=1898 ;*	*
		=1899 ;*	Function : ERDCRM (Emulate routine for the DCR M instruction).
		=1900 ;*	Inputs : None.
		=1901 ;*	Outputs : None.
		=1902 ;*	Destroys : A, H, L and flags.

LOC	OBJ	LINE	SOURCE STATEMENT
		=1903 ;*	Calls : CHKMWR, TBWREX, TBWRB and INCPC.
		=1904 ;*	
		=1905 ;*	Description : This routine emulates the 8 bit modify instruction
		=1906 ;*	DCR M. The routine checks that writes are allowed to the memory
		=1907 ;*	location whose address is given by the HL register pair.
		=1908 ;*	
		=1909 ;*	Note: The routine only checks for write permission as it is
		=1910 ;*	impossible to set a location to allow writes and not allow
		=1911 ;*	reads.
		=1912 ;*	
		=1913 ;*	Memory reference type: 8 bit modify.
		=1914 ;*	
		=1915 ;*	=====
		=1916 ;	
		=1917 ;	This routine must emulate the following instruction:
		=1918 ;	
		=1919 ;	DCR M
		=1920 ;	
		=1921 ERDCRM:	; Procedure EmulateDCR_M;
1502	2A99EA	=1922	LHLD REGM ;
1505	CD6610	=1923	CALL CHKMWR ;
1508	CA2215	=1924	JZ ERDM10 ; If CheckMemoryRead(RegM) then
		=1925	; Begin
150B	3E08	=1926	MVI A,EX8BMOD ;
150D	CD662F	=1927	CALL TBWREX ; TraceBufferWriteExecute
		=1928	; (8BitModify)
		=1929	;
1510	CDE52F	=1930	CALL TBWRW ; TraceBufferWriteWord(RegM);
1513	7E	=1931	MOV A,M ;
1514	CD572F	=1932	CALL TBWRB ; TraceBufferWrite(Memory[RegM]);
		=1933	;
1517	35	=1934	DCR M ; Memory[RegM] := Memory[RegM] - 1;
1518	7E	=1935	MOV A,M ;
1519	CD572F	=1936	CALL TBWRB ; TraceBufferWrite(Memory[RegM]);
		=1937	;
151C	CDD021	=1938	CALL INCPC ; IncrementPc;
151F	C32715	=1939	JMP ERDM20 ; End
		=1940 ERDM10:	; Else
1522	3E04	=1941	MVI A,GUARDACCESS ;
1524	32B2EA	=1942	STA STATUS ; Status := Guarded_Access;
		=1943 ERDM20:	;
1527	C9	=1944	RET ; EndProc EmulateDCR_M;
		=1945 ;	
		=1946 ;*	=====
		=1947 ;*	
		=1948 ;*	Function : ERDI (Emulate routine for the DI routine)
		=1949 ;*	Inputs : None.
		=1950 ;*	Outputs : None.
		=1951 ;*	Destroys : A and flags.
		=1952 ;*	Calls : TBWREX and INCPC.
		=1953 ;*	
		=1954 ;*	Description : This routine emulates the DI routine by clearing the
		=1955 ;*	interrupt enable bit in the interrupt mask.
		=1956 ;*	
		=1957 ;*	Memory reference type: No memory reference.
		=1958 ;*	
		=1959 ;*	=====
		=1960 ;	
		=1961 ;	This routine must emulate the following instruction:
		=1962 ;	
		=1963 ;	DI
		=1964 ;	
		=1965 ERDI:	; Procedure EmulateDI;
1528	3E01	=1966	MVI A,EXNOREF ;
152A	CD662F	=1967	CALL TBWREX ; TraceBufferWriteExecuted(NoRef);
152D	3AA1EA	=1968	LDA IMASK ;
1530	E6F7	=1969	ANI 11110111B ; InterruptMask := InterruptMask AND
		=1970	; 11110111B);
1532	32A1EA	=1971	STA IMASK ;
1535	CDD021	=1972	CALL INCPC ; IncrementPC;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 33

LOC	OBJ	LINE	SOURCE STATEMENT
		=1973	;
1538	C9	=1974	RET ; EndProc EmulateDI;
		=1975 ;	
		=1976 ;=====*	
		=1977 ;*	
		=1978 ;* Function : ERDRM (Emulate MOV Reg,M instructions)	*
		=1979 ;* Inputs : None.	*
		=1980 ;* Outputs : None.	*
		=1981 ;* Destroys : A, H, L and flags.	*
		=1982 ;* Calls : EXWREX, TBWRW, TBWRB, GETRA, CHKMRD and INPCP.	*
		=1983 ;*	*
		=1984 ;* Description : This routine emulates all the MOVE routines that have	*
		=1985 ;* M as their source register. The routine uses the	*
		=1986 ;* GetRegisterAddress routine to find the address of the register	*
		=1987 ;* storage address.	*
		=1988 ;*	*
		=1989 ;* Memory reference type: 8 bit read	*
		=1990 ;*	*
		=1991 ;=====*	
		=1992 ;	
		=1993 ; This routine must emulate the following instructions:	
		=1994 ;	
		=1995 ; MOV B,M MOV C,M MOV D,M MOV E,M	
		=1996 ; MOV H,M MOV L,M MOV A,M	
		=1997 ;	
		=1998 ERDRM: ; Procedure EmulateDestinationRegMemory;	
1539	2A99EA	=1999 LHLD REGM ;	
153C	CD3E10	=2000 CALL CHKMRD ;	
153F	CA6115	=2001 JZ ERDM80 ; If CheckMemoryRead(RegM) then	
		=2002 ; Begin	
1542	3E02	=2003 MVI A,EX8BRD ;	
1544	CD662F	=2004 CALL TBWREX ; TraceBufferWriteExecuted	
		=2005 ; (8BitRead);	
		=2006 ;	
1547	CDE52F	=2007 CALL TBWRW ; TraceBufferWriteWord(RegM);	
154A	7E	=2008 MOV A,M ;	
154B	CD572F	=2009 CALL TBWRB ; TraceBufferWriteByte	
		=2010 ; (Memory[RegM]);	
154E	47	=2011 MOV B,A ;	
		=2012 ;	
154F	3AA8EA	=2013 LDA OPCODE ;	
1552	E638	=2014 ANI 00111000B ;	
1554	0F	=2015 RRC ;	
1555	0F	=2016 RRC ;	
1556	0F	=2017 RRC ; RegNum := (Opcode AND 00111000B)	
		=2018 ; shr 3;	
		=2019 ;	
1557	CD9B14	=2020 CALL GETRA ; RegAdr := GetRegisterAddress(RegNum);	
		=2021 ;	
155A	70	=2022 MOV M,B ; Memory[RegAdr] := memory[RegM];	
		=2023 ;	
155B	CDD021	=2024 CALL INPCP ; IncrementPC;	
155E	C36615	=2025 JMP ERDM99 ; End	
		=2026 ;	
		=2027 ERDM80: ; Else	
1561	3E04	=2028 MVI A,GUARDACCESS ;	
1563	32B2EA	=2029 STA STATUS ; Status := GuardAccess;	
		=2030 ERDM99: ;	
1566	C9	=2031 RET ; EndProc EmulateDestinationRegMemory;	
		=2032 ;	
		=2033 ;=====*	
		=2034 ;*	*
		=2035 ;* Function : EREI (Emulate routine for the EI routine)	*
		=2036 ;* Inputs : None.	*
		=2037 ;* Outputs : None.	*
		=2038 ;* Destroys : A and flags.	*
		=2039 ;* Calls : TBWREX and INPCP.	*
		=2040 ;*	*
		=2041 ;* Description : This routine emulates the EI instruction by setting	*
		=2042 ;* the interrupt enable bit in the interrupt mask.	*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 34

LOC	OBJ	LINE	SOURCE STATEMENT
		=2043 ;*	
		=2044 ;*	Memory reference type: No memory reference.
		=2045 ;*	
		=2046 ;*	NOTE: As the effect of an EI instruction must only occur after the
		=2047 ;*	NEXT instruction, all the EREI routine does is set a flag
		=2048 ;*	that is polled in the SSTEP routine and will cause the
		=2049 ;*	interrupt enable bit to be set in the interrupt mask after
		=2050 ;*	the instruction following the EI.
		=2051 ;*	
		=2052 ;	=====
		=2053 ;	
		=2054 ;	This routine must emulate the following instruction:
		=2055 ;	
		=2056 ;	EI
		=2057 ;	
		=2058 EREI:	; Procedure EmulateEI;
1567	3E01	=2059	MVI A,EXNOREF ;
1569	CD662F	=2060	CALL TBWREX ; TraceBufferWriteExecuted(NoRef);
156C	3EFF	=2061	MVI A,TRUE ;
156E	32A2EA	=2062	STA ENABINT ; Enable_interrupt := True;
		=2063	
1571	CDD021	=2064	CALL INCPC ; IncrementPC;
		=2065	
1574	C9	=2066	RET ; EndProc EmulateEI;
		=2067 ;	
		=2068 ;	=====
		=2069 ;*	
		=2070 ;*	Function : ERERR (Emulate illegal instructions)
		=2071 ;*	Inputs : None.
		=2072 ;*	Outputs : None.
		=2073 ;*	Destroys : A.
		=2074 ;*	Calls : None.
		=2075 ;*	
		=2076 ;*	Description : This routine emulates illegal instructions by simply
		=2077 ;*	setting the status byte to 'ILLEGAL_INSTRUCTION'.
		=2078 ;*	
		=2079 ;*	Memory reference type: No memory reference.
		=2080 ;*	
		=2081 ;	=====
		=2082 ;	
		=2083 ;	This routine must handle all illegal opcodes.
		=2084 ;	
		=2085 ERERR:	; Procedure EmulateError;
1575	3E03	=2086	MVI A,ILLEGALINST ;
1577	32B2EA	=2087	STA STATUS ; Status := IllegalInstruction;
		=2088	
157A	C9	=2089	RET ; Endproc EmulateError;
		=2090 ;	
		=2091 ;	=====
		=2092 ;*	
		=2093 ;*	Function : ERHLT (Emulate HLT instruction).
		=2094 ;*	Inputs : None.
		=2095 ;*	Outputs : None.
		=2096 ;*	Destroys : A.
		=2097 ;*	Calls : None.
		=2098 ;*	
		=2099 ;*	Description : This routine emulates the HLT instruction by setting
		=2100 ;*	the status byte to 'HLT_ENCOUNTERED'.
		=2101 ;*	
		=2102 ;*	Memory reference type: No memory reference.
		=2103 ;*	
		=2104 ;	=====
		=2105 ;	
		=2106 ;	This routine must emulate the following instruction:
		=2107 ;	
		=2108 ;	HLT
		=2109 ;	
		=2110 ERHLT:	; Procedure EmulateHlt;
157B	3E01	=2111	MVI A,HLTENCODED ;
157D	32B2EA	=2112	STA STATUS ; Status := HltEncountered;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 35

LOC	OBJ	LINE	SOURCE STATEMENT
		=2113	;
1580	C9	=2114	RET ; EndProc EmulateHlt;
		=2115 ;	
		=2116 ;=====*	
		=2117 ;*	
		=2118 ;* Function : ERIN (Emulate IN instruction).	
		=2119 ;* Inputs : None.	
		=2120 ;* Outputs : None.	
		=2121 ;* Destroys : A, B, H, L and flags.	
		=2122 ;* Calls : D8FETCH, CHKPRD, TBWREX, TBWRB, INCPC and DYNIO.	
		=2123 ;*	
		=2124 ;* Description : This routine emulates the IN instruction. The routine	
		=2125 ;* checks that inputs are allowed from the particular port.	
		=2126 ;*	
		=2127 ;* To allow the user program access to the console a dummy console is	
		=2128 ;* maintained by the debugger. In order for this to word inputs from	
		=2129 ;* the data and status registers are trapped by the routine and	
		=2130 ;* handled separately. If an input from the data register is detected	
		=2131 ;* then the memory value KEYVAL is returned and the ReceiverReady bit	
		=2132 ;* in the dummy console status byte, CSTATUS, is cleared.	
		=2133 ;*	
		=2134 ;* Memory reference type: 8 bit input.	
		=2135 ;*	
		=2136 ;=====*	
		=2137 ;	
		=2138 ; This routine must emulate the following instruction:	
		=2139 ;	
		=2140 ; IN ????	
		=2141 ;	
		=2142 ERIN: ; Procedure EmulateIN;	
		=2143 ;	
1581	CD2014	=2144 CALL D8FETCH ; Data8 := FetchData8;	
1584	47	=2145 MOV B,A ;	
1585	CD8E10	=2146 CALL CHKPRD ;	
1588	CACB15	=2147 JZ ERIN10 ; If CheckPortRead(Data8) then	
		=2148 ; Begin	
158B	3E06	=2149 MVI A,EX8BIN ;	
158D	CD662F	=2150 CALL TBWREX ; TraceBufferWriteExecute(8BitInput);	
1590	78	=2151 MOV A,B ;	
1591	CD572F	=2152 CALL TBWRB ; TraceBufferWriteByte(Data8);	
		=2153 ;	
1594	FE00	=2154 CPI CODATA ;	
1596	C2A715	=2155 JNZ ERIN02 ; If Data8 = ConsoleData then	
		=2156 ; Begin	
1599	3AB6EA	=2157 LDA CSTATUS ;	
159C	E6FD	=2158 ANI NOT RXRDY ;	
159E	32B6EA	=2159 STA CSTATUS ; ConsoleStatus :=	
		=2160 ; ConsoleStatus AND	
		=2161 ; (NOT ReceiverReady);	
		=2162 ;	
15A1	3AB7EA	=2163 LDA KEYVAL ; ByteValue := KeyValue;	
		=2164 ;	
15A4	C3BF15	=2165 JMP ERIN08 ; End	
		=2166 ERIN02: ;	
15A7	FE01	=2167 CPI COSTAT ;	
15A9	C2B215	=2168 JNZ ERIN04 ; Elseif Data8 = ConsoleStatus then	
		=2169 ; Begin	
15AC	3AB6EA	=2170 LDA CSTATUS ; ByteValue := ConsoleStatus;	
15AF	C3BF15	=2171 JMP ERIN08 ; End	
		=2172 ERIN04: ;	
		=2173 ;	
		=2174 ; Else	
		=2175 ; Begin	
15B2	21CEEA	=2175 LXI H,DYNIO ;	
15B5	36DB	=2176 MVI M,(IN) ;	
15B7	23	=2177 INX H ;	
15B8	70	=2178 MOV M,B ;	
15B9	23	=2179 INX H ;	
15BA	36C9	=2180 MVI M,(RET) ;	
15BC	CDCEEA	=2181 CALL DYNIO ; ByteValue := InputFromPort	
		=2182 ; (Data8);	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 36

LOC	OBJ	LINE	SOURCE STATEMENT
		=2183	;
		=2184	;
		=2185	ERIN08: ; End;
15BF	329CEA	=2186	STA REGA ; AReg := ByteValue;
		=2187	;
15C2	CD572F	=2188	CALL TBWRB ; TraceBufferWriteByte(ByteValue);
		=2189	;
15C5	CDD021	=2190	CALL INCPC ; IncrementPc;
15C8	C3D015	=2191	JMP ERIN20 ; End
		=2192	ERIN10: ; Else
15CB	3E04	=2193	MVI A,GUARDACCESS ;
15CD	32B2EA	=2194	STA STATUS ; Status := Guarded_Access;
		=2195	ERIN20: ;
15D0	C9	=2196	RET ; EndProc EmulateIN;
		=2197	;
		=2198	*****
		=2199	;
		=2200	Function : ERINRM (Emulate INR M instruction) *
		=2201	Inputs : None. *
		=2202	Outputs : None. *
		=2203	Destroys : A, H, L and flags. *
		=2204	Calls : CHKMWR, TBWREX, TBWRW, TBWRB and INCPC. *
		=2205	;
		=2206	Description : This routine emulates the 8 bit modify instruction *
		=2207	INR M. The routine checks that writes are allowed to the memory *
		=2208	location whose address is given by the HL register pair. *
		=2209	;
		=2210	Note: The routine only checks for write permission as it is *
		=2211	impossible to set a location to allow writes and not allow *
		=2212	reads. *
		=2213	;
		=2214	Memory reference type: 8 bit modify. *
		=2215	;
		=2216	*****
		=2217	;
		=2218	This routine must emulate the following instruction:
		=2219	;
		=2220	INR M
		=2221	;
		=2222	ERINRM: ; Procedure EmulateINR_M;
15D1	2A99EA	=2223	LHLD REGM ;
15D4	CD6610	=2224	CALL CHKMWR ;
15D7	CAF115	=2225	JZ ERIM10 ; If CheckMemoryRead(RegM) then
		=2226	Begin
15DA	3E08	=2227	MVI A,EX8BMOD ;
15DC	CD662F	=2228	CALL TBWREX ; TraceBufferWriteExecute
		=2229	;
		=2230	;
15DF	CDE52F	=2231	CALL TBWRW ; TraceBufferWriteWord(RegM);
15E2	7E	=2232	MOV A,M ;
15E3	CD572F	=2233	CALL TBWRB ; TraceBufferWrite(Memory[RegM]);
		=2234	;
15E6	34	=2235	INR M ; Memory[RegM] := Memory[RegM] + 1;
15E7	7E	=2236	MOV A,M ;
15E8	CD572F	=2237	CALL TBWRB ; TraceBufferWrite(Memory[RegM]);
		=2238	;
15EB	CDD021	=2239	CALL INCPC ; IncrementPc;
15EE	C3F615	=2240	JMP ERIM20 ; End
		=2241	ERIM10: ; Else
15F1	3E04	=2242	MVI A,GUARDACCESS ;
15F3	32B2EA	=2243	STA STATUS ; Status := Guarded_Access;
		=2244	ERIM20: ;
15F6	C9	=2245	RET ; EndProc EmulateINR_M;
		=2246	;
		=2247	*****
		=2248	;
		=2249	Function : ERJMP (Emulate jump instructions). *
		=2250	Inputs : None. *
		=2251	Outputs : None. *
		=2252	Destroys : A, H, L and flags. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 37

LOC	OBJ	LINE	SOURCE STATEMENT
		=2253 ;*	Calls : TBWREX, D16FETCH, INCPC and DECIDE.
		=2254 ;*	
		=2255 ;*	Description : This routine emulates all jump, conditional and non
		=2256 ;*	conditional, instructions. To decide if a conditional jump, is to
		=2257 ;*	be performed the DECIDE routine is used. Emulating a jump is done
		=2258 ;*	by simply setting the program counter to the value of the 16 bit
		=2259 ;*	operand.
		=2260 ;*	
		=2261 ;*	Memory reference type: No memory reference.
		=2262 ;*	
		=2263 ;*	=====
		=2264 ;	
		=2265 ;	This routine must emulate the following instructions:
		=2266 ;	
		=2267 ;	JMP ???? ;
		=2268 ;	JNZ ???? JZ ???? JNC ???? JC ???? ;
		=2269 ;	JPO ???? JPE ???? JP ???? JM ???? ;
		=2270 ;	
		=2271 ERJMP:	; Procedure EmulateJump;
15F7	3E01	=2272	MVI A, EXNOREF ;
15F9	CD662F	=2273	CALL TBWREX ; TraceBufferWriteExecuted(NoRef);
		=2274 ;	
15FC	3AA8EA	=2275	LDA OPCODE ;
15FF	FEC3	=2276	CPI (JMP) ;
1601	CA1516	=2277	JZ ERJ40 ; If (Opcode <> (JMP)) or
		=2278 ;	
1604	CD2814	=2279	CALL DECIDE ; (Decide(Opcode)) then
1607	CA1516	=2280	JZ ERJ40 ;
		=2281 ;	Begin
160A	3E01	=2282	MVI A, EXNOREF ;
160C	CD662F	=2283	CALL TBWREX ; TraceBufferWriteExecuted(NoRef);
		=2284 ;	
160F	CDD021	=2285	CALL INCPC ; IncrementPC
1612	C31B16	=2286	JMP ERJ80 ; End
		=2287 ERJ40:	; Else
		=2288 ;	Begin
1615	CD1814	=2289	CALL D16FETCH ;
1618	229DEA	=2290	SHLD REGPC ; RegPC := Data16Fetch;
		=2291 ERJ80:	; End
161B	C9	=2292	RET ; EndProc EmulateJump;
		=2293 ;	
		=2294 ;	
		=2295 ;*	=====
		=2296 ;*	
		=2297 ;*	Function : ERLDA (Emulate routine for the LDA ???? instruction).
		=2298 ;*	Inputs : None.
		=2299 ;*	Outputs : None.
		=2300 ;*	Destroys : A, H, L and flags.
		=2301 ;*	Calls : D16FETCH, CHKMRD, TBWREX, TBWRW, TBWRB and INCPC.
		=2302 ;*	
		=2303 ;*	Description : This routine emulated the LDA ???? instruction. The
		=2304 ;*	memory address of the byte to be read, the value of the 16 bit
		=2305 ;*	operand, is checked for protection errors.
		=2306 ;*	
		=2307 ;*	Memory reference type: 8 bit read.
		=2308 ;*	
		=2309 ;*	=====
		=2310 ;	
		=2311 ;	This routine must emulate the following instruction:
		=2312 ;	
		=2313 ;	LDA ???? ;
		=2314 ;	
		=2315 ERLDA:	; Procedure EmulateLDA;
161C	CD1814	=2316	CALL D16FETCH ;
161F	CD3E10	=2317	CALL CHKMRD ;
1622	CA3A16	=2318	JZ ERLA10 ; If CheckMemoryRead(Data16) then
		=2319 ;	Begin
1625	3E02	=2320	MVI A, EX8BRD ;
1627	CD662F	=2321	CALL TBWREX ; TraceBufferWriteExecuted
		=2322	(8bitRead);

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 38

LOC	OBJ	LINE	SOURCE STATEMENT
		=2323	
162A	CDE52F	=2324	CALL TBWRW ; TraceBufferWriteWord(Data16);
162D	7E	=2325	MOV A,M ;
162E	329CEA	=2326	STA REGA ; Areg := Memory[Data16];
		=2327	
1631	CD572F	=2328	CALL TBWRB ; TraceBufferWriteByte(Areg);
		=2329	
1634	CDD021	=2330	CALL INCPC ; IncrementPc;
1637	C33F16	=2331	JMP ERLA20 ; End
		=2332	ERLA10: ; Else
163A	3E04	=2333	MVI A,GUARDACCESS ;
163C	32B2EA	=2334	STA STATUS ; Status := Guarded_Access;
		=2335	ERLA20: ;
163F	C9	=2336	RET ; EndProc EmulateLDA;
		=2337	
		=2338	=====*
		=2339	;* Function : ERLDAX (Emulate routine for LDAX B D instructions) *
		=2340	;* Inputs : None. *
		=2341	;* Outputs : None. *
		=2342	;* Destroys : A, H, L and flags. *
		=2343	;* Calls : GETRPA, GETINDIRECT, CHKMRD, TBWREX, TBWRW, INCPC *
		=2344	;* and TNWRB *
		=2345	;* *
		=2346	;* Description : This routine emulates LDAX B and LDAX D routines. *
		=2347	;* The contents of the register pair (DE or HL) is obtained by first *
		=2348	;* finding the storage address of the word and then doing an indirect *
		=2349	;* fetch via that address. The contents of the register pair are then *
		=2350	;* used as the address for the memory read. *
		=2351	;* Memory reference type: 8 bit read. *
		=2352	;* *
		=2353	;* *
		=2354	;* =====*
		=2355	
		=2356	
		=2357	This routine must emulate the following instructions:
		=2358	
		=2359	Inst: LDAX B LDAX D
		=2360	
		=2361	Opcode: 00001010 00011010
		=2362	
		=2363	ERLDAX: ; Procedure EmulateLDAX;
1640	3AA8EA	=2364	LDA OPCODE ;
1643	CDB914	=2365	CALL GETRPA ; RegAdr := GetRegisterPairAddress(Opcode);
1646	CD9414	=2366	CALL GETINDIRECT ; RegValue := GetIndirect(RegAdr);
1649	CD3E10	=2367	CALL CHKMRD ;
164C	CA6416	=2368	JZ ERLD10 ; If CheckMemoryRead(RegValue) then
		=2369	Begin
164F	3602	=2370	MVI M,EX8BRD ;
1651	CD662F	=2371	CALL TBWREX ; TraceBufferWriteExecuted
		=2372	(8bitRead);
		=2373	
1654	CDE52F	=2374	CALL TBWRW ; TraceBufferWriteWord(RegValue);
1657	7E	=2375	MOV A,M ;
1658	329CEA	=2376	STA REGA ; RegA := Memory[RegValue];
		=2377	
165B	CD572F	=2378	CALL TBWRB ; TraceBufferWriteByte(RegA);
165E	CDD021	=2379	CALL INCPC ; IncrementPC;
1661	C36916	=2380	JMP ERLD20 ; End
		=2381	ERLD10: ; Else
1664	3E04	=2382	MVI A,GUARDACCESS ;
1666	32B2EA	=2383	STA STATUS ; Status := GuardedAccess;
		=2384	ERLD20: ;
1669	C9	=2385	RET ; EndProc EmulateLDAX;
		=2386	
		=2387	=====*
		=2388	;* Function : ERLHLD (Emulate routine for the LHLD instruction) *
		=2389	;* Inputs : None. *
		=2390	;* Outputs : None. *
		=2391	;* Destroys : A, H, L and flags. *
		=2392	;* *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 39

LOC	OBJ	LINE	SOURCE STATEMENT
		=2393 ;*	Calls : D16FETCH, CHKMRD, GETINDIRECT, TBWREX, TBWRW and INCPC. *
		=2394 ;*	
		=2395 ;*	Description : This routine emulates the LHLH routine. The two bytes *
		=2396 ;*	starting at the address given as the 16 operand are checked to *
		=2397 ;*	ensure that memory reads are allowed. *
		=2398 ;*	
		=2399 ;*	Memory reference type: 16 bit read. *
		=2400 ;*	
		=2401 ;	=====*
		=2402 ;	
		=2403 ;	This routine must emulate the following instruction:
		=2404 ;	
		=2405 ;	LHLH ????
		=2406 ;	
		=2407	ERLHLH: ; Procedure EmulateLHLH;
166A	CD1814	=2408	CALL D16FETCH ;
166D	CD3E10	=2409	CALL CHKMRD ;
1670	CA9116	=2410	JZ ERLH10 ; If CheckMemoryRead(Data16) AND
		=2411	;
1673	23	=2412	INX H ;
1674	CD3E10	=2413	CALL CHKMRD ;
1677	CA9116	=2414	JZ ERLH10 ; CheckmemoryRead(Data16+1) then
		=2415	Begin
167A	3E04	=2416	MVI A,EX16BRD ;
167C	CD662F	=2417	CALL TBWREX ; TraceBufferWriteExecuted
		=2418	;
		=2419	;
167F	CDE52F	=2420	CALL TBWRW ; TraceBufferWriteWord(Data16);
1682	CD9414	=2421	CALL GETINDIRECT ;
1685	2299EA	=2422	SHLD REGM ; RegM := memory[Data16+1]*256 +
		=2423	memory[Data16];
		=2424	;
1688	CDE52F	=2425	CALL TBWRW ; TraceBufferWrite(RegM);
		=2426	;
168B	CDD021	=2427	CALL INCPC ; IncrementPc;
168E	C39616	=2428	JMP ERLH20 ; End
		=2429	ERLH10: ; Else
1691	3E04	=2430	MVI A,GUARDACCESS ;
1693	32B2EA	=2431	STA STATUS ; Status := Guarded_Access;
		=2432	ERLH20: ;
1696	C9	=2433	RET ; EndProc EmulateLHLH;
		=2434 ;	
		=2435 ;	=====*
		=2436 ;*	
		=2437 ;*	Function : ERMEM (Emulate Arithmetic instructions that have M *
		=2438 ;*	as their operand). *
		=2439 ;*	Inputs : None. *
		=2440 ;*	Outputs : None. *
		=2441 ;*	Destroys : A, B, H, L and flags. *
		=2442 ;*	Calls : CHKMRD, TBWREX, TBWRW, TBWRB, DECROU and INCPC. *
		=2443 ;*	
		=2444 ;*	Description : This routine emulates the eight arithmetic routines *
		=2445 ;*	the have M as their operand. The way the routine does this is by *
		=2446 ;*	creating a small dynamic routine which executes a different *
		=2447 ;*	version of the same instruction. The process is given in point *
		=2448 ;*	form below: *
		=2449 ;*	
		=2450 ;*	a) Convert the ArithmeticInstruction M instruction into the *
		=2451 ;*	ArithmeticInstruction B for. This can be done by ANDing the *
		=2452 ;*	opcode with the following mask: 11111000B. *
		=2453 ;*	
		=2454 ;*	b) Create the following routine: *
		=2455 ;*	
		=2456 ;*	Routine: ArithmeticInstruction B *
		=2457 ;*	RET *
		=2458 ;*	
		=2459 ;*	c) Restore the A register and flags. *
		=2460 ;*	d) Fetch the byte pointed to by M (the HL register pair) and put *
		=2461 ;*	it in the B register. *
		=2462 ;*	e) Call the dynamic routine. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 40

LOC	OBJ	LINE	SOURCE STATEMENT
		=2463 ;*	f) Save the A register and flags. *
		=2464 ;*	*
		=2465 ;*	Memory reference type: 8 bit read. *
		=2466 ;*	*
		=2467 ;	=====*
		=2468 ;	*
		=2469 ;	This routine must emulate the following instructions:
		=2470 ;	*
		=2471 ;	ADD M ADC M SUB M SBB M
		=2472 ;	ANA M XRA M ORA M CMP M
		=2473 ;	*
		=2474 ;	General format: 10iiii10
		=2475 ;	*
		=2476 ;	Where iii = Instruction.
		=2477 ;	*
		=2478	ERMEM: ; Procedure EmulateMemoryAccessInstructions;
1697	2A99EA	=2479	LHLD REGM ;
169A	CD3E10	=2480	CALL CHKMWD ;
169D	CACC16	=2481	JZ ERM80 ; If CheckMemoryread(RegM) then
		=2482	Begin
16A0	3E02	=2483	MVI A,EX8BRD ;
16A2	CD662F	=2484	CALL TBWREX ; TraceBufferWriteExecuted
		=2485	(8bitRead);
		=2486	;
16A5	CDE52F	=2487	CALL TBWRW ; TraceBufferWriteWord(RegM);
16A8	7E	=2488	MOV A,M ;
16A9	CD572F	=2489	CALL TBWRB ; TraceBufferWriteByte
		=2490	(Memory[RegM]);
		=2491	;
16AC	46	=2492	MOV B,M ; B := Memory[RegM]
16AD	3AA8EA	=2493	LDA OPCODE ;
16B0	E6F8	=2494	ANI 1111000B ; Opcode := Opcode and 1111000b
16B2	21D1EA	=2495	LXI H,DECROU ;
16B5	77	=2496	MOV M,A ; DecideRoutine[1] := Opcode;
16B6	23	=2497	INX H ;
16B7	36C9	=2498	MVI M,(RET) ; DecideRoutine[2] := RET
16B9	2A9BEA	=2499	LHLD REGPSW ;
16BC	E5	=2500	PUSH H ;
16BD	F1	=2501	POP PSW ;
16BE	CDD1EA	=2502	CALL DECROU ; PerformInstruction;
16C1	F5	=2503	PUSH PSW ;
16C2	E1	=2504	POP H ;
16C3	229BEA	=2505	SHLD REGPSW ;
16C6	CDD021	=2506	CALL INCPC ;
16C9	C3D116	=2507	JMP ERM99 ; End
		=2508	ERM80: ; Else
16CC	3E04	=2509	MVI A,GUARDACCESS ;
16CE	32B2EA	=2510	STA STATUS ; Status := GuardedAccess;
		=2511	ERM99: ;
16D1	C9	=2512	RET ; EndProc EmulateMemoryAccessInstructions;
		=2513 ;	*
		=2514 ;	=====*
		=2515 ;*	*
		=2516 ;*	Function : ERMSR (Emulate routine for MOV M,Register instructions) *
		=2517 ;*	Inputs : None. *
		=2518 ;*	Outputs : None. *
		=2519 ;*	Destroys : A, D, E, H, L and flags. *
		=2520 ;*	Calls : CHKMWR, TBWREX, TBWRW, TBWRB, GETRA and INCPC. *
		=2521 ;*	*
		=2522 ;*	Description : This routine emulates move routines that have the M *
		=2523 ;*	register as their destination register. The routine calculates the *
		=2524 ;*	storage address of the register variable given as the source *
		=2525 ;*	register by using the opcode and the GETRA routine. *
		=2526 ;*	*
		=2527 ;*	Memory reference type: 8 bit write. *
		=2528 ;*	*
		=2529 ;	=====*
		=2530 ;	*
		=2531 ;	This routine must emulate the following instructions:
		=2532 ;	*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 41

LOC	OBJ	LINE	SOURCE STATEMENT
		=2533 ;	MOV M,B MOV M,C MOV M,D MOV M,E
		=2534 ;	MOV M,H MOV M,L MOV M,A
		=2535 ;	
		=2536	ERMSR: ; Procedure EmulateMemorySourceRegister;
16D2	2A99EA	=2537	LHLD REGM ;
16D5	CD6610	=2538	CALL CHKMWR ;
16D8	CAFA16	=2539	JZ ERMS20 ; If CheckMemoryWrite(RegM) then
		=2540	Begin
16DB	3E03	=2541	MVI A,EX8BWR ;
16DD	CD662F	=2542	CALL TBWREX ; TraceBufferWriteExecuted
		=2543	(8BitWrite);
		=2544	;
16E0	CDE52F	=2545	CALL TBWRW ; TraceBufferWriteWord(RegM);
16E3	EB	=2546	XCHG ;
		=2547	;
16E4	3AA8EA	=2548	LDA OPCode ;
16E7	E638	=2549	ANI 00111000B ;
16E9	0F	=2550	RRC ;
16EA	0F	=2551	RRC ;
16EB	0F	=2552	RRC ; RegNum := (OpCode AND 00111000B)
		=2553	shr 3;
		=2554	;
16EC	CD9B14	=2555	CALL GETRA ; RegAdr := GetRegisterAddress(RegNum);
16EF	7E	=2556	MOV A,M ;
16F0	12	=2557	STAX D ; Memory[RegM] := Memory[RegAdr];
		=2558	;
16F1	CD572F	=2559	CALL TBWRB ; TraceBufferWriteByte(Memory[RegM]);
		=2560	;
16F4	CDD021	=2561	CALL INCPC ; IncrementPC;
16F7	C3FF16	=2562	JMP ERMS99 ; End
		=2563	;
		=2564	ERMS20: ; Else
16FA	3E04	=2565	MVI A, GUARDACCESS ;
16FC	32B2EA	=2566	STA STATUS ; Status := GuardAccess;
		=2567	ERMS99: ;
16FF	C9	=2568	RET ; EndProc EmulateMemorySourceRegister;
		=2569 ;	
		=2570 ;	=====*
		=2571 ;*	*
		=2572 ;*	* Function : ERMVIM (Emulate routine for the MVI M,?? instruction) *
		=2573 ;*	* Inputs : None. *
		=2574 ;*	* Outputs : None. *
		=2575 ;*	* Destroys : A, H, L and flags. *
		=2576 ;*	* Calls : CHKMWR, D8FETCH, INCPC, TBWREX, TBWRB and TBWRW. *
		=2577 ;*	*
		=2578 ;*	* Description : This routine emulates the MVI M,?? instruction. *
		=2579 ;*	*
		=2580 ;*	* Memory reference type: 8 bit Write *
		=2581 ;*	*
		=2582 ;	=====*
		=2583 ;	
		=2584 ;	This routine must emulate the following instruction:
		=2585 ;	
		=2586 ;	MVI M,??
		=2587 ;	
		=2588	ERMVIM: ; Procedure EmulateMVI_M;
1700	2A99EA	=2589	LHLD REGM ;
1703	CD6610	=2590	CALL CHKMWR ;
1706	CA1E17	=2591	JZ ERMM10 ; If CheckMemoryRead(RegM) then
		=2592	Begin
1709	3E03	=2593	MVI A,EX8BWR ;
170B	CD662F	=2594	CALL TBWREX ; TraceBufferWriteExecuted
		=2595	(8BitWrite);
		=2596	;
170E	CDE52F	=2597	CALL TBWRW ; TraceBufferWriteWord(RegM);
		=2598	;
1711	CD2014	=2599	CALL D8FETCH ; Data8 := Data8Fetch;
		=2600	;
1714	77	=2601	MOV M,A ; Memory[RegM] := Data8;
		=2602	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 42

LOC	OBJ	LINE	SOURCE STATEMENT
1715	CD572F	=2603	CALL TBWRB ; TraceBufferWriteByte(Data8);
		=2604	;
1718	CDD021	=2605	CALL INCPC ; IncrementPc;
171B	C32317	=2606	JMP ERM20 ; End
		=2607	ERM10: ; Else
171E	3E04	=2608	MVI A, GUARDACCESS ;
1720	32B2EA	=2609	STA STATUS ; Status := Guarded_Access;
		=2610	ERM20: ;
1723	C9	=2611	RET ; EndProc EmulateMVI_M;
		=2612	;
		=2613	=====*
		=2614	;
		=2615	;* Function : EROUT (Emulate routine for the OUT instruction) *
		=2616	;* Inputs : None. *
		=2617	;* Outputs : None. *
		=2618	;* Destroys : A, B, H, L and flags. *
		=2619	;* Calls : D8FETCH, CHKPWR, TBWREX, TBWRB, DYNIO and INCPC. *
		=2620	;* *
		=2621	;* Description : This routine emulates the OUT ?? instruction. As the *
		=2622	;* 8085 lacks an indirect output instruction a short dynamic routine *
		=2623	;* must be created to perform the output operation. *
		=2624	;* *
		=2625	;* Memory reference type: 8 bit output *
		=2626	;* *
		=2627	=====*
		=2628	;
		=2629	This routine must emulate the following instruction:
		=2630	;
		=2631	OUT ??
		=2632	;
		=2633	EROUT: ; Procedure EmulateOUT;
1724	CD2014	=2634	CALL D8FETCH ;
1727	47	=2635	MOV B,A ;
1728	CDB110	=2636	CALL CHKPWR ;
172B	CA5017	=2637	JZ EROU10 ; If CheckPortWrite(Data8) then
		=2638	;
		=2639	MVI A, EX8BOUT ; Begin
172E	3E07	=2640	CALL TBWREX ; TraceBufferWriteExecute(8BitOutput);
1730	CD662F	=2641	MOV A,B ;
1733	78	=2642	CALL TBWRB ; TraceBufferWriteByte(Data8);
1734	CD572F	=2643	;
1737	21CEEA	=2644	LXI H, DYNIO ;
173A	36D3	=2645	MVI M, (OUT) ;
173C	23	=2646	INX H ;
173D	70	=2647	MOV M, B ;
173E	23	=2648	INX H ;
173F	36C9	=2649	MVI M, (RET) ;
1741	3A9CEA	=2650	LDA REGA ;
1744	CDCEEA	=2651	CALL DYNIO ; OutputToPort(Data8, Areg);
		=2652	;
1747	CD572F	=2653	CALL TBWRB ; TraceBufferWriteByte(Areg);
		=2654	;
174A	CDD021	=2655	CALL INCPC ; IncrementPc;
174D	C35517	=2656	JMP EROU20 ; End
		=2657	EROU10: ; Else
1750	3E04	=2658	MVI A, GUARDACCESS ;
1752	32B2EA	=2659	STA STATUS ; Status := Guarded_Access;
		=2660	EROU20: ;
1755	C9	=2661	RET ; EndProc EmulateOUT;
		=2662	;
		=2663	=====*
		=2664	;
		=2665	;* Function : ERPCHL (Emulate routine for the PCHL instruction) *
		=2666	;* Inputs : None. *
		=2667	;* Outputs : None. *
		=2668	;* Destroys : H and L *
		=2669	;* Calls : TBWREX and INCPC. *
		=2670	;* *
		=2671	;* Description : This routine emulates the PCHL routine by copying the *
		=2672	;* contents of the HL register pair into the Program counter. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 43

```

LOC  OBJ      LINE      SOURCE STATEMENT
                                     *
=2673 ;*                                     *
=2674 ;* Memory reference type: No memory reference           *
=2675 ;*                                     *
=2676 ;=====*
=2677 ;
=2678 ;      This routine must emulate the following instruction:
=2679 ;
=2680 ;      PCHL
=2681 ;
=2682 ERPCHL:                                ; Procedure EmulatePCHL;
1756 3E01      =2683      MVI      A,EXNOREF                ;
1758 CD662F    =2684      CALL     TBWREX                    ; TraceBufferWriteExecuted(NoRef);
175B 2A99EA    =2685      LHLD     REGM                      ;
175E 229DEA    =2686      SHLD     REGPC                     ; RegPC := RegM
=2687 ;
1761 C9        =2688      RET                                ; EndProc EmulatePCHL;
=2689 ;
=2690 ;=====*
=2691 ;*                                     *
=2692 ;* Function : ERPOP (Emulate routine for the POP instruction) *
=2693 ;* Inputs   : None.                                           *
=2694 ;* Outputs  : None.                                           *
=2695 ;* Destroys : A, D, E, H, L and flags.                       *
=2696 ;* Calls    : DOPOP, GETRPA, PUTINDIRECT and INCPC.          *
=2697 ;*                                     *
=2698 ;* Description : This routine emulates all four POP routines. The
=2699 ;* pops a 16 bit value off the stack. It then writes this value to
=2700 ;* the storage address of the register pair. The storage address of
=2701 ;* the register pair is calculated by using the GETRPA routine and
=2702 ;* the opcode.
=2703 ;*
=2704 ;* Memory reference type: 16 bit read (Done by DOPOP routine)
=2705 ;*
=2706 ;=====*
=2707 ;
=2708 ;      This routine must emulate the following instructions:
=2709 ;
=2710 ;      Inst:      POP   PSW      POP   B      POP   D      POP   H
=2711 ;
=2712 ;      Opcode:    11110001      11000001      11010001      11100001
=2713 ;
=2714 ERPOP:                                ; Procedure EmulatePOP;
1762 CD4314    =2715      CALL     DOPOP                      ;
1765 CA7917    =2716      JZ       ERPO10                      ; If DoPOP Then
=2717 ;                               ; Begin
1768 E5        =2718      PUSH     H                          ;
1769 3AA8EA     =2719      LDA       OPCODE                    ;
176C CDB914     =2720      CALL     GETRPA                      ; RegAdr :=
=2721 ;                               ; GetRegisterPairAddress(Opcod);
176F D1        =2722      POP      D                          ;
1770 CDCA14     =2723      CALL     PUTINDIRECT                ; PutIndirect(TOS);
1773 CDD021     =2724      CALL     INCPC                      ; IncrementPC;
1776 C37E17     =2725      JMP      ERPO20                    ; End
=2726 ERPO10:                                ; Else
1779 3E04      =2727      MVI      A,GUARDACCESS              ;
177B 32B2EA     =2728      STA      STATUS                    ; Status := GuardedAccess;
=2729 ERPO20:                                ;
177E C9        =2730      RET                                ; EndProc EmulatePOP;
=2731 ;
=2732 ;=====*
=2733 ;*                                     *
=2734 ;* Function : ERPUSH (Emulate routine for the PUSH instructions) *
=2735 ;* Inputs   : None.                                           *
=2736 ;* Outputs  : None.                                           *
=2737 ;* Destroys : A, H, L and flags.                       *
=2738 ;* Calls    : GETRPA, GETINDIRECT, DOPUSH and INCPC.          *
=2739 ;*                                     *
=2740 ;* Description : This routine emulates all four PUSH routines. The
=2741 ;* value to be pushed is read from the address calculated by the
=2742 ;* GETRPA routine and the opcode.

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 44

LOC	OBJ	LINE	SOURCE STATEMENT
		=2743 ;*	
		=2744 ;*	Memory reference type: 16 bit write (Done by DOPUSH routine)
		=2745 ;*	
		=2746 ;=====*	
		=2747 ;	
		=2748 ;	This routine must emulate the following instructions:
		=2749 ;	
		=2750 ;	PUSH PSW PUSH B PUSH D PUSH H
		=2751 ;	
		=2752 ERPUSH:	; Procedure EmulatePUSH;
177F	3AA8EA	=2753	LDA OPCODE ;
1782	CDB914	=2754	CALL GETRPA ; RegAdr := GetRegisterPairAddress(Opcode);
1785	CD9414	=2755	CALL GETINDIRECT ; RegValue := GetIndirect(RegAdr);
1788	CD6B14	=2756	CALL DOPUSH ;
178B	CA9417	=2757	JZ ERPU10 ; If DoPUSH Then
		=2758 ;	
178E	CDD021	=2759	CALL INCPC ; IncrementPC
1791	C39917	=2760	JMP ERPU20 ; Else
		=2761 ERPU10:	
1794	3E04	=2762	MVI A, GUARDACCESS ;
1796	32B2EA	=2763	STA STATUS ; Status := GuardedAccess;
		=2764 ERPU20:	
1799	C9	=2765	RET ; EndProc EmulatePUSH;
		=2766 ;	
		=2767 ;=====*	
		=2768 ;*	
		=2769 ;*	Function : ERRET (Emulate routine for RET instructions)
		=2770 ;*	Inputs : None.
		=2771 ;*	Outputs : None.
		=2772 ;*	Destroys : A, H, L and flags.
		=2773 ;*	Calls : DECIDE, DOPOP and INCPC.
		=2774 ;*	
		=2775 ;*	Description : This routine emulates all the return instructions,
		=2776 ;*	conditional and unconditional. If the instruction is executed the
		=2777 ;*	value popped off the stack is written to the program counter.
		=2778 ;*	
		=2779 ;*	Memory reference type: 16 bit read (Done by DOPOP routine)
		=2780 ;*	
		=2781 ;=====*	
		=2782 ;	
		=2783 ;	This routine must emulate the following instructions:
		=2784 ;	
		=2785 ;	RET ???? ;
		=2786 ;	RNZ ???? RZ ???? RNC ???? RC ???? ;
		=2787 ;	RPO ???? RPE ???? RP ???? RM ???? ;
		=2788 ;	
		=2789 ERRET:	; Procedure EmulateRET;
179A	3AA8EA	=2790	LDA OPCODE ;
179D	FEC9	=2791	CPI (RET) ;
179F	CAB317	=2792	JZ ERRE40 ; If (Opcode <> (RET)) or
		=2793 ;	
17A2	CD2814	=2794	CALL DECIDE ; (Decide(Opcode)) then
17A5	CAB317	=2795	JZ ERRE40 ;
		=2796 ;	Begin
17A8	3E01	=2797	MVI A, EXNOREF ;
17AA	CD662F	=2798	CALL TBWREX ; TraceBufferWriteExecuted(NoRef);
		=2799 ;	
17AD	CDD021	=2800	CALL INCPC ; IncrementPC
17B0	C3C417	=2801	JMP ERRE80 ; End
		=2802 ERRE40:	Else
		=2803 ;	Begin
17B3	CD4314	=2804	CALL DOPOP ;
17B6	CABF17	=2805	JZ ERRE50 ; If DoPOP then
		=2806 ;	
17B9	229DEA	=2807	SHLD REGPC ; RegPC := TOS
17BC	C3C417	=2808	JMP ERRE80 ;
		=2809 ERRE50:	Else
17BF	3E04	=2810	MVI A, GUARDACCESS ;
17C1	32B2EA	=2811	STA STATUS ; Status := GuardedAccess;
		=2812 ERRE80:	End

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 45

```

LOC  OBJ      LINE      SOURCE STATEMENT
17C4  C9      =2813      RET                                ; EndProc EmulateRET;
=2814 ;
=2815 ;=====
=2816 ;*
=2817 ;* Function : ERRIM (Emulate routine for the RIM instruction)
=2818 ;* Inputs : None.
=2819 ;* Outputs : None.
=2820 ;* Destroys : A and flags.
=2821 ;* Calls : TBWREX and INCPC.
=2822 ;*
=2823 ;* Description : This routine read the interrupt mask into the A
=2824 ;* register.
=2825 ;*
=2826 ;* Memory reference type: No memory reference.
=2827 ;*
=2828 ;=====
=2829 ;
=2830 ;      This routine must emulate the following instruction:
=2831 ;
=2832 ;      RIM
=2833 ;
=2834 ERRIM:                                ; Procedure EmulateRIM;
17C5  3E01    =2835      MVI    A,EXNOREF                ;
17C7  CD662F  =2836      CALL   TBWREX                    ; TraceBufferWriteExecuted(NoRef);
17CA  3AA1EA  =2837      LDA    IMASK                      ;
17CD  329CEA  =2838      STA    REGA                        ; RegA := InterruptMask;
=2839 ;
17D0  CDD021  =2840      CALL   INCPC                        ; IncrementPC;
=2841 ;
17D3  C9      =2842      RET                                ; EndProc EmulateRIM;
=2843 ;
=2844 ;=====
=2845 ;*
=2846 ;* Function : ERRST (Emulate routine for the RST instructions)
=2847 ;* Inputs : None.
=2848 ;* Outputs : None.
=2849 ;* Destroys : A, H, L and flags.
=2850 ;* Calls : ADDBC and DOPUSH.
=2851 ;*
=2852 ;* Description : This routine emulates the eight Restart restart
=2853 ;* routines. The address of the call is calculated from the opcode.
=2854 ;*
=2855 ;* The following table lists the instructions and the call address.
=2856 ;*
=2857 ;*      Instruction      Opcode      Call address
=2858 ;*
=2859 ;*      RST 0            C7H          00H
=2860 ;*      RST 1            CFH          08H
=2861 ;*      RST 2            D7H          10H
=2862 ;*      RST 3            DFH          18H
=2863 ;*      RST 4            E7H          20H
=2864 ;*      RST 5            EFH          28H
=2865 ;*      RST 6            F7H          30H
=2866 ;*      RST 7            FFH          38H
=2867 ;*
=2868 ;* Memory reference type: 16 bit write (Done by DOPUSH routine)
=2869 ;*
=2870 ;=====
=2871 ;
=2872 ;      This routine must emulate the following instructions:
=2873 ;
=2874 ;      RST 0            RST 1            RST 2            RST 3
=2875 ;      RST 4            RST 5            RST 6            RST 7
=2876 ;
=2877 ERRST:                                ; Procedure EmulateRST;
17D4  2A9DEA  =2878      LHLD   REGPC                        ;
17D7  CDC621  =2879      CALL   ADDBC                        ; AddByteCount;
17DA  CD6B14  =2880      CALL   DOPUSH                      ;
17DD  CAEE17  =2881      JZ     ERRSTO                        ; If DoPUSH(REGPC) then
=2882 ; Begin

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 46

LOC	OBJ	LINE	SOURCE STATEMENT
17E0	3AA8EA	=2883	LDA OPCODE ;
17E3	E638	=2884	ANI 00111000B ;
17E5	6F	=2885	MOV L,A ;
17E6	2600	=2886	MVI H,0 ;
17E8	229DEA	=2887	SHLD REGPC ;
17EB	C3F317	=2888	JMP ERRST5 ;
		=2889	ERRST0: ;
17EE	3E04	=2890	MVI A,GUARDACCESS ;
17F0	32B2EA	=2891	STA STATUS ;
		=2892	ERRST5: ;
17F3	C9	=2893	RET ;
		=2894	;
		=2895	;
		=2896	;
		=2897	Function : ERSHLD (Emulate routine for SHLD routine)
		=2898	Inputs : None.
		=2899	Outputs : None.
		=2900	Destroys : A, D, E, H, L and flags.
		=2901	Calls : D16FETCH, CHKMWR, TBWREX, TBWRW, PUTINDIRECT and INCPC.
		=2902	;
		=2903	Description : This routine emulates the SHLD ??? routine. The
		=2904	value of the HL register pair is written to the address given by
		=2905	the 16 bit operand.
		=2906	;
		=2907	Memory reference type: 16 bit write.
		=2908	;
		=2909	;
		=2910	;
		=2911	This routine must emulate the following instruction:
		=2912	;
		=2913	SHLD ???
		=2914	;
		=2915	ERSHLD: ;
17F4	CD1814	=2916	CALL D16FETCH ;
17F7	CD6610	=2917	CALL CHKMWR ;
17FA	CA1E18	=2918	JZ ERSH10 ;
		=2919	;
17FD	23	=2920	INX H ;
17FE	CD6610	=2921	CALL CHKMWR ;
1801	CA1E18	=2922	JZ ERSH10 ;
		=2923	;
1804	2B	=2924	DCX H ;
1805	3E05	=2925	MVI A,EX16BWR ;
1807	CD662F	=2926	CALL TBWREX ;
		=2927	;
		=2928	;
180A	CDE52F	=2929	CALL TBWRW ;
180D	EB	=2930	XCHG ;
180E	2A99EA	=2931	LHLD REGM ;
1811	CDE52F	=2932	CALL TBWRW ;
1814	EB	=2933	XCHG ;
1815	CDCA14	=2934	CALL PUTINDIRECT ;
1818	CDD021	=2935	CALL INCPC ;
181B	C32318	=2936	JMP ERSH20 ;
		=2937	ERSH10: ;
181E	3E04	=2938	MVI A,GUARDACCESS ;
1820	32B2EA	=2939	STA STATUS ;
		=2940	ERSH20: ;
1823	C9	=2941	RET ;
		=2942	;
		=2943	;
		=2944	;
		=2945	Function : ERSIM (Emulate SIM instruction)
		=2946	Inputs : None.
		=2947	Outputs : None.
		=2948	Destroys : A, B and flags.
		=2949	Calls : TBWREX and INCPC.
		=2950	;
		=2951	Description : This routine emulates the SIM (Set interrupt mask)
		=2952	routine. The following functions are performed:

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 47

LOC	OBJ	LINE	SOURCE STATEMENT
		=2953 ;*	
		=2954 ;*	a) If D4 (RST 7.5 reset bit) is set then the RST 7.5 flip flop
		=2955 ;*	is reset by executing a SIM instruction.
		=2956 ;*	
		=2957 ;*	b) If D3 (Mask set bit) is set then the new interrupt mask
		=2958 ;*	(D2..D0) are copied to the inerrupt mask register (IMASK)
		=2959 ;*	
		=2960 ;*	Memory reference type: No memory reference.
		=2961 ;*	
		=2962 ;*	=====
		=2963 ;	
		=2964 ;	This routine must emulate the following instruction:
		=2965 ;	
		=2966 ;	SIM
		=2967 ;	
		=2968 ERSIM:	; Procedure EmulateSIM;
1824	3E01	=2969	MVI A,EXNOREF ;
1826	CD662F	=2970	CALL TBWREX ; TraceBufferWriteExecuted(NoRef);
		=2971	;
1829	3A9CEA	=2972	LDA REGA ;
182C	47	=2973	MOV B,A ;
182D	E610	=2974	ANI 00010000B ;
182F	30	=2975	SIM ; If RegA AND Reset75 then
		=2976	Reset75FlipFlop;
1830	78	=2977	MOV A,B ;
1831	E608	=2978	ANI 00001000B ;
1833	CA4318	=2979	JZ ERSIME ; If RegA AND SetInterruptMasks then
		=2980	;
1836	47	=2981	MOV B,A ;
1837	E607	=2982	ANI 00000111B ;
1839	47	=2983	MOV B,A ;
183A	3AA1EA	=2984	LDA IMASK ;
183D	E6F8	=2985	ANI 11111000B ;
183F	B0	=2986	ORA B ;
1840	32A1EA	=2987	STA IMASK ; InterruptMask := (RegA and MaskBits) OR
		=2988	(InterruptMask AND (NOT maskbits))
		=2989 ERSIME:	;
1843	CDD021	=2990	CALL INCPC ; IncrementPC;
		=2991	;
1846	C9	=2992	RET ; EndProc EmulateSIM;
		=2993 ;	
		=2994 ;*	=====
		=2995 ;*	
		=2996 ;*	Function : ERSPHL (Emulate routine for the SPHL instruction)
		=2997 ;*	Inputs : None.
		=2998 ;*	Outputs : None.
		=2999 ;*	Destroys : H and L
		=3000 ;*	Calls : INCPC.
		=3001 ;*	
		=3002 ;*	Description : This routine emulates the SPHL instruction by copying
		=3003 ;*	the HL register pair to the stack pointer.
		=3004 ;*	
		=3005 ;*	Memory reference type: No memory reference.
		=3006 ;*	
		=3007 ;*	=====
		=3008 ;	
		=3009 ;	This routine must emulate the following instruction:
		=3010 ;	
		=3011 ;	SPHL
		=3012 ;	
		=3013 ERSPHL:	; Procedure EmulateSPHL;
1847	2A99EA	=3014	LHLD REGM ;
184A	229FEA	=3015	SHLD REGSP ; RegSP := RegM;
		=3016	;
184D	3E01	=3017	MVI A,EXNOREF ;
184F	CD662F	=3018	CALL TBWREX ; TraceBufferWriteExecuted(NoRef);
		=3019	;
1852	CDD021	=3020	CALL INCPC ; IncrementPC;
		=3021	;
1855	C9	=3022	RET ; EndProc EmulateSPHL;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 48

LOC	OBJ	LINE	SOURCE STATEMENT
		=3023 ;	
		=3024 ;=====*	
		=3025 ;*	
		=3026 ;* Function : ERSTA (Emulate routine for the STA instruction)	*
		=3027 ;* Inputs : None.	*
		=3028 ;* Outputs : None.	*
		=3029 ;* Destroys : A, H, L and flags.	*
		=3030 ;* Calls : D16FETCH, CHKMWR, TBWREX, TBWRW, TBWRB and INPC.	*
		=3031 ;*	*
		=3032 ;* Description : This routine emulates the STA ???? instruction. The	*
		=3033 ;* A register is written to the address given by the 16 bit operand.	*
		=3034 ;*	*
		=3035 ;* Memory reference type: 8 bit write.	*
		=3036 ;*	*
		=3037 ;=====*	
		=3038 ;	
		=3039 ; This routine must emulate the following instruction:	
		=3040 ;	
		=3041 ; STA ????	
		=3042 ;	
		=3043 ERSTA: ; Procedure EmulateSTA;	
1856	CD1814	=3044 CALL D16FETCH ;	
1859	CD6610	=3045 CALL CHKMWR ;	
185C	CA7418	=3046 JZ ERSA10 ; If CheckMemoryRead(Data16) then	
		=3047 ; Begin	
185F	3E03	=3048 MVI A,EX8BWR ;	
1861	CD662F	=3049 CALL TBWREX ; TraceBufferWriteExecuted	
		=3050 ; (8BitWrite);	
		=3051 ;	
1864	CDE52F	=3052 CALL TBWRW ;	
1867	3A9CEA	=3053 LDA REGA ; TraceBufferWriteWord(RegM);	
		=3054 ;	
186A	77	=3055 MOV M,A ; Memory[Data16] := RegA;	
		=3056 ;	
186B	CD572F	=3057 CALL TBWRB ; TraceBufferWriteByte(RegA);	
186E	CDD021	=3058 CALL INCPC ; IncrementPc;	
1871	C37918	=3059 JMP ERSA20 ; End	
		=3060 ERSA10: ; Else	
1874	3E04	=3061 MVI A,GUARDACCESS ;	
1876	32B2EA	=3062 STA STATUS ; Status := Guarded_Access;	
		=3063 ERSA20: ;	
1879	C9	=3064 RET ; EndProc EmulateSTA;	
		=3065 ;	
		=3066 ;=====*	
		=3067 ;*	
		=3068 ;* Function : ERSTAX (Emulate routine for the STAX instruction)	*
		=3069 ;* Inputs : None.	*
		=3070 ;* Outputs : None.	*
		=3071 ;* Destroys : A, H, L and flags.	*
		=3072 ;* Calls : GETRPA, GETINDIRECT, CHKMWR, TBWREX, TBWRW and TBWRB	*
		=3073 ;* and INPC.	*
		=3074 ;*	*
		=3075 ;* Description : This routine emulates the STAX B and STAX D	*
		=3076 ;* instructions.	*
		=3077 ;*	*
		=3078 ;* Memory reference type: 8 bit write.	*
		=3079 ;*	*
		=3080 ;=====*	
		=3081 ;	
		=3082 ; This routine must emulate the following instructions:	
		=3083 ;	
		=3084 ; Inst STAX B STAX D	
		=3085 ;	
		=3086 ; Opcode: 00Q00010 00010010	
		=3087 ;	
		=3088 ERSTAX: ; Procedure EmulateSTAX;	
187A	3AA8EA	=3089 LDA OPCODE ;	
187D	CDB914	=3090 CALL GETRPA ; RegAdr := GetRegisterPairAddress(Opcode);	
1880	CD9414	=3091 CALL GETINDIRECT ; RegValue := GetIndirect(RegAdr)	
1883	CD6610	=3092 CALL CHKMWR ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 49

LOC	OBJ	LINE	SOURCE STATEMENT
1886	CA9E18	=3093	JZ ERST10 ; If CheckMemoryWrite(RegValue) then
		=3094	; Begin
1889	3E03	=3095	MVI A,EX8BWR ;
188B	CD662F	=3096	CALL TBWREX ; TraceBufferWriteExecuted
		=3097	; (8BitWrite)
		=3098	; ;
188E	CDE52F	=3099	CALL TBWRW ; TraceBufferWriteWord(RegM);
1891	3A9CEA	=3100	LDA REGA ;
1894	77	=3101	MOV M,A ; Memory[RegValue] := RegA;
		=3102	; ;
1895	CD572F	=3103	CALL TBWRB ; TraceBufferWrite(RegA);
1898	CDD021	=3104	CALL INCPD ; IncrementPC;
189B	C3A318	=3105	JMP ERST20 ; End
		=3106	ERST10: ; Else
189E	3E04	=3107	MVI A,GUARDACCESS ;
18A0	32B2EA	=3108	STA STATUS ; Status := GuardedAccess;
		=3109	ERST20: ;
18A3	C9	=3110	RET ; EndProc EmulateSTAX
		=3111	;
		=3112	;
		=3113	;
		=3114	Function : ERXTHL (Emulate XTHL instruction) *
		=3115	Inputs : None. *
		=3116	Outputs : None. *
		=3117	Destroys : A, D, E, H, L and flags. *
		=3118	Calls : INCPD, TBWREX, TBWRW and CHKMWR. *
		=3119	;
		=3120	Description : This routine emulates the XTHL instruction. The XTHL *
		=3121	instruction swaps the HL register pair and the word on the top of *
		=3122	stack. *
		=3123	;
		=3124	Memory reference type: 16 bit modify. *
		=3125	;
		=3126	;
		=3127	;
		=3128	This routine must emulate the following instruction:
		=3129	;
		=3130	XTHL
		=3131	;
		=3132	ERXTHL: ; Procedure EmulateXTHL;
18A4	2A9FEA	=3133	LHLD REGSP ;
18A7	CD6610	=3134	CALL CHKMWR ;
18AA	CAD918	=3135	JZ ERXT20 ; If (CheckMemoryWrite(RegSP)) AND
		=3136	; ;
18AD	23	=3137	INX H ;
18AE	CD6610	=3138	CALL CHKMWR ;
18B1	CAD918	=3139	JZ ERXT20 ; (CheckMemoryWrite(RegSP+1)) then
		=3140	; Begin
18B4	2B	=3141	DCX H ;
18B5	3E09	=3142	MVI A,EX16BMOD ;
18B7	CD662F	=3143	CALL TBWREX ; TraceBufferWriteExecuted
		=3144	; (16BitModify);
		=3145	; ;
18BA	CDE52F	=3146	CALL TBWRW ; TraceBufferWriteWord(RegSP);
18BD	EB	=3147	XCHG ;
18BE	2A99EA	=3148	LHLD REGM ; TraceBufferWriteWord(RegM);
18C1	CDE52F	=3149	CALL TBWRW ;
		=3150	; ;
18C4	EB	=3151	XCHG ;
18C5	7B	=3152	MOV A,E ; TempByte := Reg.L;
18C6	5E	=3153	MOV E,M ; Reg.L := Memory(RegSP);
18C7	77	=3154	MOV M,A ; Memory(RegSP) := TempByte;
18C8	23	=3155	INX H ;
18C9	7A	=3156	MOV A,D ; Temp := Reg.H;
18CA	56	=3157	MOV D,M ; Reg.H := Memory(RegSP+1);
18CB	77	=3158	MOV M,A ; Memory(RegSP+1) := TempByte;
18CC	EB	=3159	XCHG ;
		=3160	; ;
18CD	2299EA	=3161	SHLD REGM ;
18D0	CDE52F	=3162	CALL TBWRW ; TraceBufferWriteWord(RegM);

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 50

LOC	OBJ	LINE	SOURCE STATEMENT
18D3	CDD021	=3163	CALL INCPC ; IncrementPC;
18D6	C3DE18	=3164	JMP ERXT99 ; End
		=3165	ERXT20: ; Else
18D9	3E04	=3166	MVI A, GUARDACCESS ;
18DB	32B2EA	=3167	STA STATUS ; Status := GuardedAccess;
		=3168	ERXT99: ;
18DE	C9	=3169	RET ; EndProc EmulateXTHL;
		=3170	;
		=3171	END <DEBEMUL.A85>
		3172	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 51

LOC	OBJ	LINE	SOURCE STATEMENT
		3173 ;	
		3174 ;	Basic Input Output routines:
		3175 ;	
		3176 \$INCLUDE(DEBBIO.A85)	
		=3177 ;	-----;
		=3178 ;	;
		=3179 ;	DEBBIO module for the UCT SABus debugger
		=3180 ;	-----;
		=3181 ;	;
		=3182 ;	This module contains the basic I/O routines used in the SABUS 8085
		=3183 ;	debugger. These routines should be changed if the debugger is to
		=3184 ;	be run on another machine.
		=3185 ;	;
		=3186 ;	This module should contain the following routines:
		=3187 ;	;
		=3188 ;	a) KITIN - KIT character input routine.
		=3189 ;	b) KITOUT - KIT character output routine.
		=3190 ;	c) KITTEST - KIT console test routine.
		=3191 ;	;
		=3192 ;	-----;
		=3193 ;	
		=3194 ;	UCT SABus monitor equates used to perform the above functions:
		=3195 ;	
F013		=3196 CI EQU 0F013H	; Keyboard input routine.
F01C		=3197 CO EQU 0F01CH	; Screen output routine.
F00A		=3198 CITEST EQU 0F00AH	; Console input test routine.
		=3199 ;	
		=3200 ;	-----
		=3201 ;	
0002		=3202 RXRDY EQU 00000010B	; Receiver ready bit
0001		=3203 TXRDY EQU 00000001B	; Transmitter ready bit.
		=3204 ;	
0000		=3205 CODATA EQU 0	; I/O Address of the console data register
0001		=3206 COSTAT EQU 1	; I/O Address of the console status register.
		=3207 ;	
		=3208 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 52

LOC	OBJ	LINE	SOURCE STATEMENT
		=3209	=====*
		=3210	;
		=3211	Function : KITIN (Kit character input routine)
		=3212	Inputs : None.
		=3213	Outputs : A : Character read from the keyboard.
		=3214	Destroys : A and flags.
		=3215	Calls : CI.
		=3216	;
		=3217	Description : This routine must input the next character from the
		=3218	keyboard. If no character is ready then the routine must wait
		=3219	for a key to be typed. The routine must leave all but the A
		=3220	register and the flags untouched.
		=3221	;
		=3222	=====*
		=3223	;
		=3224	KITIN: ; Procedure KitInput : Byte;
		=3225	;
18DF	C313F0	=3226	JMP CI ; Return(ConsoleInput);
		=3227	;
		=3228	; EndProc KitInput;
		=3229	;
		=3230	=====*
		=3231	;
		=3232	Function : KITOUT (Kit character output routine).
		=3233	Inputs : A : Character to display on the console.
		=3234	Outputs : None.
		=3235	Destroys : None. (Can destroy A and flags).
		=3236	Calls : CO.
		=3237	;
		=3238	Description : This routine must display the ASCII character passed
		=3239	in the A register on the console screen. The routine should, if
		=3240	necessary, advance the cursor.
		=3241	;
		=3242	The routine should also implement flow control. This means that if
		=3243	a XOFF (Control-S) is typed by the user the KITOUT routine should
		=3244	wait until a XON (Control-Q) is typed before any further
		=3245	characters can be sent to the console screen.
		=3246	;
		=3247	=====*
		=3248	;
		=3249	KITOUT: ; Procedure KitOutput(Ch:Byte);
18E2	F5	=3250	PUSH PSW ;
18E3	CD5E19	=3251	CALL BREAK ; Break; {implement flow control}
18E6	F1	=3252	POP PSW ;
18E7	CD1CF0	=3253	CALL CO ; ConsoleOutput(Ch);
		=3254	;
18EA	C9	=3255	RET ; EndProc KitOutput;
		=3256	;
		=3257	=====*
		=3258	;
		=3259	Function : KITTEST (Kit console test routine).
		=3260	Inputs : None.
		=3261	Outputs : Z = 0 : Character is ready to fetched from the keyboard
		=3262	Z = 1 : No character ready.
		=3263	Destroys : A and Flags.
		=3264	Calls : CITEST.
		=3265	;
		=3266	Description : This routine must check to see if a character has
		=3267	been typed by the user and is ready to be feteched by the program.
		=3268	;
		=3269	=====*
		=3270	;
		=3271	KITTEST: ; Procedure KitTest : Boolean;
		=3272	;
18EB	C30AF0	=3273	JMP CITEST ; Return(ConsoleInputTest);
		=3274	;
		=3275	; EndProc KitTest;
		=3276	;
		=3277	END ;<DEBBIO.A85>
		3278	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 53

LOC	OBJ	LINE	SOURCE STATEMENT
		3279 ;	
		3280 ;	Packed message output routines
		3281 ;	
		3282 \$INCLUDE(DEBOUTMSG.A85)	
		=3283 ;	-----;
		=3284 ;	;
		=3285 ;	Message output module for the UCT SABus 8085 symbolic debugger
		=3286 ;	;
		=3287 ;	This module contains (in order) the following subroutines:
		=3288 ;	;
		=3289 ;	a) OUTMSG - This routine is used to output messages that have
		=3290 ;	been stored in a packed format on the screen.
		=3291 ;	;
		=3292 ;	-----;
		=3293 ;	
		=3294 ;	=====*
		=3295 ;*	*
		=3296 ;*	Function : OUTMSG (Output message)
		=3297 ;*	Inputs : HL : Pointer to start address of message.
		=3298 ;*	Outputs : None.
		=3299 ;*	Destroys : A, H, L and flags.
		=3300 ;*	Calls : KITOUT.
		=3301 ;*	*
		=3302 ;*	Description : This routine outputs a message to the KIT. The
		=3303 ;*	message is terminated by a <EndMsg> character.
		=3304 ;*	*
		=3305 ;	=====*
		=3306 ;	
		=3307 OUTMSG:	; Procedure OutMessage(Ptr);
		=3308	;
18EE 7E		=3309 MOV A,M	; Ch := Memory[Ptr];
18EF FE00		=3310 CPI ENDMMSG	;
18F1 CA2719		=3311 JZ OUTM99	; While Ch <> EndMsg do
		=3312	Begin
		=3313	;
18F4 E6E0		=3314 ANI 11100000B	; If (Ch and 11100000B)
18F6 FE80		=3315 CPI 10000000B	; = 10000000B then
18F8 C21319		=3316 JNZ OUTM03	;
		=3317	Begin
18FB D5		=3318 PUSH D	;
18FC E5		=3319 PUSH H	;
18FD 7E		=3320 MOV A,M	;
18FE E61F		=3321 ANI 00011111B	; Entry := Ch and 11111B
1900 87		=3322 ADD A	;
1901 6F		=3323 MOV L,A	;
1902 2600		=3324 MVI H,0	;
1904 115219		=3325 LXI D,PACKTAB	;
1907 19		=3326 DAD D	;
1908 CD9414		=3327 CALL GETINDIRECT	; StringAddress :=
		=3328	PackTable[Entry];
		=3329	;
190B CDEE18		=3330 CALL OUTMSG	; OutMessage(StringAddress);
190E E1		=3331 POP H	;
190F D1		=3332 POP D	;
1910 C32319		=3333 JMP OUTM10	; End
		=3334 OUTM03:	; Else
		=3335	;
		=3336	Begin
1913 7E		=3337 MOV A,M	;
1914 E67F		=3338 ANI ASCII	;
1916 CDE218		=3339 CALL KITOUT	; KitOutput(CH and ASCII);
1919 7E		=3340 MOV A,M	;
191A B7		=3341 ORA A	;
191B F22319		=3342 JP OUTM08	; If Ch < 0 then
		=3343	Begin
191E 3E20		=3344 MVI A,SPACE	;
1920 CDE218		=3345 CALL KITOUT	; KitOutput(Space);
		=3346 OUTM08:	End;
		=3347	;
		=3348 OUTM10:	End;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 54

LOC	OBJ	LINE	SOURCE STATEMENT
1923	23	=3349	INX H ; Ptr := Ptr + 1;
1924	C3EE18	=3350	JMP OUTMSG ;
		=3351	OUTM99: ; End;
		=3352	; ;
1927	C9	=3353	RET ; EndProc OutMessage;
		=3354	;
		=3355	=====
		=3356	;
		=3357	Tables of replacement strings and pointers to the strings:
		=3358	;
1928	3C4E554D 424552BE 00	=3359	PSTR01: DB '<NUMBER', '>'+MSB, EndMsg ; 80H
1931	3C547269 67676572	=3360	PSTR02: DB '<Trigger' ; 81H
1939	436F6E64 6983BE00	=3361	PSTR03: DB 'Condi', PSTION, '>'+MSB, EndMsg ; 82H
1941	74696F6E 00	=3362	PSTR04: DB 'tion', EndMsg ; 83H
1946	796E7461 7800	=3363	PSTR05: DB 'yntax', EndMsg ; 84H
194C	6572726F 7200	=3364	PSTR06: DB 'error', EndMsg ; 85H
		=3365	;
1952	2819	=3366	PACKTAB: DW PSTR01 ; Pointer to pack string # 1
1954	3119	=3367	DW PSTR02 ; Pointer to pack string # 2
1956	3919	=3368	DW PSTR03 ; Pointer to pack string # 3
1958	4119	=3369	DW PSTR04 ; Pointer to pack string # 4
195A	4619	=3370	DW PSTR05 ; Pointer to pack string # 5
195C	4C19	=3371	DW PSTR06 ; Pointer to pack string # 6
		=3372	;
		=3373	END <DEBOUTMSG.A85>
		3374	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 55

LOC	OBJ	LINE	SOURCE STATEMENT
		3375 ;	
		3376 ;	Input output routines:
		3377 ;	
		3378	\$INCLUDE(DEBIO.A85)
		=3379 ;	-----;
		=3380 ;	;
		=3381 ;	DEBIO module for the UCT SABus debugger
		=3382 ;	-----;
		=3383 ;	;
		=3384 ;	This module contains the I/O routines used in the SABUS 8085 debugger ;
		=3385 ;	;
		=3386 ;	This module contains (in order) the following routines:
		=3387 ;	;
		=3388 ;	a) BREAK - Implement Xon/Xoff flow control.
		=3389 ;	b) CONHEX - Convert to hexadecimal.
		=3390 ;	c) CMDERR - Report a command error. (print correct syntax)
		=3391 ;	d) CRLF - Start a new line on the display.
		=3392 ;	e) ERROR - Used to report an error to the console.
		=3393 ;	f) KEYIN - Fetch a key from the keyboard.
		=3394 ;	g) OUTSTR - Output a string. (*)
		=3395 ;	h) PHEXB - Print a byte as 2 hexadecimal digits.
		=3396 ;	i) PHEXW - Print a word as 4 hexadecimal digits.
		=3397 ;	j) PSTATUS - Print the status of the user program.
		=3398 ;	k) SPACES - Print n spaces.
		=3399 ;	l) TABOUT - Output a tab character to the console.
		=3400 ;	;
		=3401 ;	(*) A string and a message are defined as being different in this
		=3402 ;	program. A string is a series of ASCII characters and is
		=3403 ;	terminated by a byte with its most significant bit set.
		=3404 ;	;
		=3405 ;	A message on the other hand is a series of byte, some of them
		=3406 ;	ASCII characters. The setting of the top bit of a byte in a
		=3407 ;	message means that a space must following. If the byte is in
		=3408 ;	the range (80H..9FH), i.e. control codes with there top bits
		=3409 ;	set, then a special string is subsituted.
		=3410 ;	;
		=3411 ;	-----;
		=3412	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 56

LOC	OBJ	LINE	SOURCE STATEMENT
		=3413 ;	=====*
		=3414 ;*	*
		=3415 ;*	Function : BREAK (Implement XON/XOFF flow control) *
		=3416 ;*	Inputs : None. *
		=3417 ;*	Outputs : None. *
		=3418 ;*	Destroys : A and flags. *
		=3419 ;*	Calls : KITTEST and KITIN. *
		=3420 ;*	*
		=3421 ;*	Description : BREAK checks whether a character has been typed on *
		=3422 ;*	the console and takes the appropriate action. If it is a control-S *
		=3423 ;*	(XOFF) the routine waits until a control-Q (XON) is entered before *
		=3424 ;*	returning. Any other characters are lost. *
		=3425 ;*	*
		=3426 ;	=====*
		=3427 ;	
		=3428 BREAK:	; Procedure Break;
195E	CDEB18	=3429	CALL KITTEST ;
1961	CA7619	=3430	JZ BRK10 ; If KitInputTest then
		=3431	; Begin
1964	CDDF18	=3432	CALL KITIN ; Ch := KitInput;
1967	E67F	=3433	ANI ASCII ; Ch := Ch AND ASCII;
1969	FE13	=3434	CPI CTRLS ;
196B	C27619	=3435	JNZ BRK10 ; If Ch = XOFF then
		=3436 BRK05:	; Repeat
		=3437	; Repeat
		=3438	; Ch := KitInput;
196E	CDDF18	=3439	CALL KITIN ; Ch := KitInput;
1971	FE11	=3440	CPI CTRLQ ;
1973	C26E19	=3441	JNZ BRK05 ; Until Ch = XON;
		=3442 BRK10:	; End
		=3443	; EndProc Break;
1976	C9	=3444	RET ; EndProc Break;
		=3445 ;	
		=3446 ;	=====*
		=3447 ;*	*
		=3448 ;*	Function : CONHEX (Convert a nibble into a hexadecimal character). *
		=3449 ;*	Inputs : A : Nibble to be converted. (Value 0 to 15) *
		=3450 ;*	Outputs : A : Character (Value '0' to '9' and 'A' to 'F') *
		=3451 ;*	Destroys : A and flags. *
		=3452 ;*	Calls : None. *
		=3453 ;*	*
		=3454 ;*	Description : This routine is used when outputting numbers in *
		=3455 ;*	hexadecimal format. This routine is used to convert a nibble into *
		=3456 ;*	its hexadecimal equivalent. *
		=3457 ;*	*
		=3458 ;	=====*
		=3459 ;	
		=3460 CONHEX:	; Procedure ConvertHex(Nibble:Byte):Byte;
1977	FE0A	=3461	CPI 10 ;
1979	FA7E19	=3462	JM CONH05 ; If Nibble >= 10 then
		=3463	; Nibble := Nibble + 'A' - '9' - 1;
197C	C607	=3464	ADI 'A' - '9' - 1 ; Nibble := Nibble + 'A' - '9' - 1;
		=3465 CONH05:	; Return(Chr(Nibble)+'0');
197E	C630	=3466	ADI '0' ;
		=3467	; EndProc ConvertHex;
1980	C9	=3468	RET ; EndProc ConvertHex;
		=3469 ;	
		=3470 ;	=====*
		=3471 ;*	*
		=3472 ;*	Function : CMDERR (Command error routine) *
		=3473 ;*	Inputs : HL : Pointer to the syntax message for the command. *
		=3474 ;*	Outputs : None. *
		=3475 ;*	Destroys : H, L, A and flags. *
		=3476 ;*	Calls : KITOUT, CRLF and OUTMSG. *
		=3477 ;*	*
		=3478 ;*	Description : *
		=3479 ;*	*
		=3480 ;	=====*
		=3481 ;	
		=3482 CMDERR:	; Procedure CommandError(Ptr:Integer);

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 57

```

LOC  OBJ      LINE      SOURCE STATEMENT
1981  E5      =3483
          =3484      PUSH    H          ; TempPtr := Ptr;
          =3485
          =3486 CMDE05:          ; While
1982  7E      =3487      MOV     A,M      ;
1983  B7      =3488      ORA     A          ;
1984  FA8E19   =3489      JM      CMDE10     ; Memory[TempPtr] > 0 do
          =3490          ; Begin
1987  CDE218   =3491      CALL    KITOUT     ; KitOutput(Memory[TempPtr]);
198A  23      =3492      INX     H          ; TempPtr := TempPtr + 1;
198B  C38219   =3493      JMP     CMDE05     ; End;
          =3494 CMDE10:          ;
198E  E67F     =3495      ANI     ASCII     ;
1990  CDE218   =3496      CALL    KITOUT     ; KitOutput(Memory[TempPtr] AND ASCII);
          =3497          ;
1993  21A119   =3498      LXI     H,CMDERM   ;
1996  CDEE18   =3499      CALL    OUTMSG     ; OutMessage(CommandErrorMessage);
          =3500          ;
1999  E1       =3501      POP     H          ;
199A  CDEE18   =3502      CALL    OUTMSG     ; OutMessage(Ptr);
          =3503          ;
199D  CDB419   =3504      CALL    CRLF      ; NewLine;
          =3505          ;
19A0  C9       =3506      RET              ; EndProc CommandError;
          =3507 ;
          =3508 CMDERM:
          =3509 ;
          =3510 ; format:
          =3511 ;
          =3512 ; command syntax error
          =3513 ; Syntax:
          =3514 ;
          =3515 ; declaration:
          =3516 ;
19A1  20636F6D =3517      DB      ' command s', PSYNTAX, ' ', PSERROR
      6D616E64
      20738420
      85
19AE  0DOA     =3518      DB      CR, LF
19B0  53843A00 =3519      DB      'S', PSYNTAX, ': ', EndMsg
          =3520 ;
          =3521 ;=====
          =3522 ;*
          =3523 ;* Function : CRLF (New line routine) *
          =3524 ;* Inputs : None. *
          =3525 ;* Outputs : None. *
          =3526 ;* Destroys : None. *
          =3527 ;* Calls : KITOUT. *
          =3528 ;*
          =3529 ;* Description : This routine starts a new line on the console screen. *
          =3530 ;* It does this by sending the following two control codes to the *
          =3531 ;* console: *
          =3532 ;*
          =3533 ;* CR - Carriage return. *
          =3534 ;* LF - Line feed. *
          =3535 ;*
          =3536 ;=====
          =3537 ;
          =3538 CRLF:          ; Procedure CarriageReturnLineFeed;
19B4  F5       =3539      PUSH    PSW      ;
19B5  3E0D     =3540      MVI     A,CR      ;
19B7  CDE218   =3541      CALL    KITOUT     ; KitOutput(CR);
19BA  3E0A     =3542      MVI     A,LF      ;
19BC  CDE218   =3543      CALL    KITOUT     ; KitOutput(LF);
19BF  F1       =3544      POP     PSW      ;
19C0  C9       =3545      RET              ; EndProc CarriageReturnLineFeed;
          =3546 ;
          =3547 ;=====
          =3548 ;*
          =3549 ;* Function : ERROR (Print error message) *
```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 58

LOC	OBJ	LINE	SOURCE STATEMENT
		=3550 ;*	Inputs : HL : Pointer to error message. *
		=3551 ;*	Outputs : None. *
		=3552 ;*	Destroys : A, H, L and flags. *
		=3553 ;*	Calls : KITOUT and CRLF. *
		=3554 ;*	
		=3555 ;*	Description : This routine is used to report an error and set the *
		=3556 ;*	error flag. The start address of the error message is passed in *
		=3557 ;*	the HL register pair. The string is terminated by the zero byte. *
		=3558 ;*	
		=3559 ;	=====*
		=3560 ;	
		=3561 ERROR:	; Procedure Error(MessagePtr:Integer);
		=3562 ;	
19C1	3A78EA	=3563	LDA ERRFLAG ;
19C4	FE00	=3564	CPI FALSE ;
19C6	C2D419	=3565	JNZ ERR10 ; If ErrorFlag = False then
		=3566	Begin
19C9	CDEE18	=3567	CALL OUTMSG ; OutMessage(MessagePtr);
19CC	CDB419	=3568	CALL CRLF ; NewLine;
19CF	3EFF	=3569	MVI A,TRUE ;
19D1	3278EA	=3570	STA ERRFLAG ; ErrorFlag := True;
		=3571 ;	
		=3572 ERR10:	; End
19D4	C9	=3573	RET ; EndProc Error;
		=3574 ;	
		=3575 ;	=====*
		=3576 ;*	
		=3577 ;*	Function : KEYIN (Key input). *
		=3578 ;*	Inputs : None. *
		=3579 ;*	Outputs : A : Upper case character typed by the user. *
		=3580 ;*	Destroys : A and flags. *
		=3581 ;*	Calls : KITIN. *
		=3582 ;*	
		=3583 ;*	Description : This routine fetches a valid character from the *
		=3584 ;*	keyboard. If the character is a lower case character then it is *
		=3585 ;*	converted to upper case. The following characters are valid: *
		=3586 ;*	
		=3587 ;*	i) CR and DEL *
		=3588 ;*	ii) All printable characters. *
		=3589 ;*	
		=3590 ;	=====*
		=3591 ;	
		=3592 KEYIN:	; Procedure KeyInput : Byte;
		=3593 ;	Repeat
19D5	CDDF18	=3594	CALL KITIN ; Ch := KitInput;
19D8	FE61	=3595	CPI 'a' ;
19DA	FAE419	=3596	JM NOTLOW ; If (Ch >= 'a') and
		=3597 ;	
19DD	FE7B	=3598	CPI 'z'+1 ;
19DF	F2E419	=3599	JP NOTLOW ; (Ch < ('z'+1)) then
		=3600 ;	
19E2	D620	=3601	SUI 'a'-'A' ; Ch := Ch - ('a' - 'A')
		=3602 ;	
		=3603 NOTLOW:	
		=3604 ;	Until
19E4	FE0D	=3605	CPI CR ;
19E6	CAF119	=3606	JZ KEYI99 ; (Ch = CR) or
		=3607 ;	
19E9	FE20	=3608	CPI SPACE ;
19EB	F2F119	=3609	JP KEYI99 ; (Ch >= Space)
		=3610 ;	
19EE	C3D519	=3611	JMP KEYIN ;
		=3612 KEYI99:	; Return(Ch);
		=3613 ;	
19F1	C9	=3614	RET ; EndProc KeyInput;
		=3615 ;	
		=3616 ;	=====*
		=3617 ;*	
		=3618 ;*	Function : OUTSTR (Output string) *
		=3619 ;*	Inputs : HL : Pointer to start address of string. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 59

LOC	OBJ	LINE	SOURCE STATEMENT
		=3620 ;*	Outputs : None.
		=3621 ;*	Destroys : A, H, L and flags.
		=3622 ;*	Calls : KITOUT.
		=3623 ;*	
		=3624 ;*	Description : This routine outputs a string. The string is
		=3625 ;*	terminated by a character with its top bit set.
		=3626 ;*	
		=3627 ;	=====*
		=3628 ;	
		=3629	OUTSTR: ; Procedure OutString(Ptr);
19F2	7E	=3630	MOV A,M ;
19F3	B7	=3631	ORA A ;
19F4	FAFE19	=3632	JM OUTS05 ; While Memory[Ptr] > 0 do
		=3633	; Begin
		=3634	;
19F7	CDE218	=3635	CALL KITOUT ; KitOutput(Memory[Ptr]);
19FA	23	=3636	INX H ; Ptr := Ptr + 1;
19FB	C3F219	=3637	JMP OUTSTR ;
		=3638	OUTS05: ; End;
19FE	E67F	=3639	ANI ASCII ;
1A00	CDE218	=3640	CALL KITOUT ; KitOutput(Memory[Ptr] AND ASCII);
		=3641	;
1A03	C9	=3642	RET ; EndProc OutString;
		=3643 ;	
		=3644 ;	=====*
		=3645 ;*	
		=3646 ;*	Function : PHEXB (Print hexadecimal byte).
		=3647 ;*	Inputs : A : Byte to be printed.
		=3648 ;*	Outputs : None.
		=3649 ;*	Destroys : A and flags.
		=3650 ;*	Calls : CONHEX and KITOUT.
		=3651 ;*	
		=3652 ;*	Description : This routine prints an eight bit byte as two hex-
		=3653 ;*	adecimal digits. The upper nibble (four bit group) is first
		=3654 ;*	shifted into the lower nibble and converted to the equivalent
		=3655 ;*	character. The character is then printed. The process is repeated
		=3656 ;*	with the lower nibble.
		=3657 ;*	
		=3658 ;	=====*
		=3659 ;	
		=3660	PHEXB: ; Procedure PrintHexByte (ByteVal:Byte);
1A04	D5	=3661	PUSH D ;
		=3662	;
1A05	5F	=3663	MOV E,A ;
1A06	E6F0	=3664	ANI 11110000B ;
1A08	1F	=3665	RAR ;
1A09	1F	=3666	RAR ;
1A0A	1F	=3667	RAR ;
1A0B	1F	=3668	RAR ; Nibble := ((ByteVal AND 11110000B) shr 4);
		=3669	;
1A0C	CD7719	=3670	CALL CONHEX ; Ch := ConvertHex(Nibble);
		=3671	;
1A0F	CDE218	=3672	CALL KITOUT ; KitOutput(Ch);
		=3673	;
1A12	7B	=3674	MOV A,E ;
1A13	E60F	=3675	ANI 00001111B ; Nibble := ByteVal AND 00001111B
		=3676	;
1A15	CD7719	=3677	CALL CONHEX ; Ch := ConvertHex(Nibble);
		=3678	;
1A18	CDE218	=3679	CALL KITOUT ; KitOutput(Ch);
		=3680	;
1A1B	D1	=3681	POP D ;
1A1C	C9	=3682	RET ; EndProc PrintHexByte;
		=3683 ;	
		=3684 ;	=====*
		=3685 ;*	
		=3686 ;*	Function : PHEXW (Print hexadecimal word).
		=3687 ;*	Inputs : HL : Word to be printed.
		=3688 ;*	Outputs : None.
		=3689 ;*	Destroys : A and flags.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 61

LOC	OBJ	LINE	SOURCE STATEMENT
		=3760 ;	
		=3761 HLTEMSG:	
		=3762 ;	
		=3763 ; format:	
		=3764 ;	
		=3765 ; Hlt encountered	
		=3766 ;	
		=3767 ; declaration:	
		=3768 ;	
1A49	486CF465 6E636F75 6E746572 656400	=3769	DB 'Hl', 't'+MSB, 'encountered', EndMsg
		=3770 ;	
		=3771 TRIGMSG:	
		=3772 ;	
		=3773 ; format:	
		=3774 ;	
		=3775 ; Trigger condition met	
		=3776 ;	
		=3777 ; declaration:	
		=3778 ;	
1A58	54726967 6765F263 6F6E6469 83206D65 7400	=3779	DB 'Trigge', 'r'+MSB, 'condi', PSTION, ' met', EndMsg
		=3780 ;	
		=3781 ILLMSG:	
		=3782 ;	
		=3783 ; format:	
		=3784 ;	
		=3785 ; Illegal instruction	
		=3786 ;	
		=3787 ; declaration:	
		=3788 ;	
1A6A	496C6C65 6761EC69 6E737472 75638300	=3789	DB 'Illega', 'l'+MSB, 'instruc', PSTION, EndMsg
		=3790 ;	
		=3791 GUAMSG:	
		=3792 ;	
		=3793 ; format:	
		=3794 ;	
		=3795 ; Guarded access	
		=3796 ;	
		=3797 ; declaration:	
		=3798 ;	
1A7A	47756172 6465E461 63636573 7300	=3799	DB 'Guarde', 'd'+MSB, 'access', EndMsg
		=3800 ;	
		=3801 ESCMSG:	
		=3802 ;	
		=3803 ; format:	
		=3804 ;	
		=3805 ; Program terminated - <ESC> entered	
		=3806 ;	
		=3807 ; declaration:	
		=3808 ;	
1A88	50726F67 7261ED74 65726D69 6E617465 E4	=3809	DB 'Progra', 'm'+MSB, 'terminate', 'd'+MSB
1A99	AD3C4553 43BE656E 74657265	=3810	DB '-'+MSB, '<ESC', '>'+MSB, 'entered', EndMsg

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 62

LOC	OBJ	LINE	SOURCE STATEMENT
		6400	
		=3811 ;	
		=3812 ;=====*	
		=3813 ;*	
		=3814 ;* Function : SPACES (Print n spaces)	*
		=3815 ;* Inputs : A : Number of spaces to print.	*
		=3816 ;* Outputs : None.	*
		=3817 ;* Destroys : None.	*
		=3818 ;* Calls : KITOUT.	*
		=3819 ;*	*
		=3820 ;* Description : This routine prints n spaces on the console. The	*
		=3821 ;* number of spaces to print, n, is passed in the A register.	*
		=3822 ;*	*
		=3823 ;=====*	
		=3824 ;	
		=3825 SPACES:	; Procedure Spaces(Count:Byte);
1AA7	F5	=3826 PUSH PSW	;
		=3827	; For I := 1 to Count do
1AA8	3E20	=3828 MVI A, ' '	;
1AAA	CDE218	=3829 CALL KITOUT	; KitOutput(' ');
1AAD	F1	=3830 POP PSW	;
1AAE	3D	=3831 DCR A	;
1AAF	C2A71A	=3832 JNZ SPACES	;
		=3833	;
1AB2	C9	=3834 RET	; EndProc Spaces;
		=3835 ;	
		=3836 ;=====*	
		=3837 ;*	
		=3838 ;* Function : TABOUT (Horizontal tab output).	*
		=3839 ;* Inputs : None.	*
		=3840 ;* Outputs : None.	*
		=3841 ;* Destroys : A.	*
		=3842 ;* Calls : KITOUT.	*
		=3843 ;*	*
		=3844 ;* Description : This routine outputs a horizontal tab character to	*
		=3845 ;* the console. If the console or terminal device does not support	*
		=3846 ;* the tab character, this routine should output spaces to place the	*
		=3847 ;* cursor on the next tab position. Tab positions occur at every	*
		=3848 ;* multiple of eight characters.	*
		=3849 ;*	*
		=3850 ;=====*	
		=3851 ;	
		=3852 TABOUT:	; Procedure TabOutput;
1AB3	3E09	=3853 MVI A,TAB	;
1AB5	CDE218	=3854 CALL KITOUT	; KitOut(Tab);
		=3855	;
1AB8	C9	=3856 RET	; EndProc TabOutput;
		=3857 ;	
		=3858 ; END <DEBIO.A85>	
		3859 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 63

LOC	OBJ	LINE	SOURCE STATEMENT
		3860 ;	
		3861 ;	General routines:
		3862 ;	
		3863	\$INCLUDE(DEBGEN.A85)
		=3864 ;	-----;
		=3865 ;	;
		=3866 ;	General routines use in the UCT SABus debugger. ;
		=3867 ;	;
		=3868 ;	This module contains (in order) the following subroutines. ;
		=3869 ;	;
		=3870 ;	a) BCMP - compare the BC and HL register pairs. ;
		=3871 ;	b) DCMP - Compares the DE and HL register pairs. ;
		=3872 ;	c) INDIRECT - Does an indirect call via the HL register pair. ;
		=3873 ;	;
		=3874 ;	-----;
		=3875	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 64

LOC	OBJ	LINE	SOURCE STATEMENT
		=3876 ;	=====*
		=3877 ;*	*
		=3878 ;*	Function : BCMP (Compare BC and HL register pairs) *
		=3879 ;*	Inputs : BC and HL (The two 16 bit numbers to be compared) *
		=3880 ;*	Outputs : Z = 1 : If BC = HL *
		=3881 ;*	Z = 0 : If BC <> HL. *
		=3882 ;*	Destroys : A and flags. *
		=3883 ;*	Calls : None. *
		=3884 ;*	*
		=3885 ;*	Description : This routine compares the BC and HL register pairs. *
		=3886 ;*	Unlike the DCMP routine, which does a full comparison, this *
		=3887 ;*	routine only checks to see if they are equal or not. *
		=3888 ;*	*
		=3889 ;	=====*
		=3890 ;	
		=3891 BCMP:	; Procedure BCompare;
		=3892 ;	;
1AB9	78	=3893 MOV A,B	;
1ABA	BC	=3894 CMP H	; If Reg.B = Reg.H AND
1ABB	C0	=3895 RNZ	;
1ABC	79	=3896 MOV A,C	;
1ABD	BD	=3897 CMP L	; Reg.C = Reg.L then
		=3898 ;	;
		=3899 ;	Return(True)
		=3900 ;	Else
		=3901 ;	Return(False);
		=3902 ;	;
1ABE	C9	=3903 RET	; EndProc BCompare;
		=3904 ;	
		=3905 ;	=====*
		=3906 ;*	*
		=3907 ;*	Function : DCMP (Compare DE and HL register pairs) *
		=3908 ;*	Inputs : DE and HL to be compared. *
		=3909 ;*	Outputs : CY flag clear if HL >= DE *
		=3910 ;*	set if HL < DE *
		=3911 ;*	Z flag set if HL = DE *
		=3912 ;*	Calls : NONE *
		=3913 ;*	Destroys : Flags. *
		=3914 ;*	*
		=3915 ;*	Description : DCMP compares DE and HL (unsigned 16-bit comparison), *
		=3916 ;*	setting the carry and zero flags according to the result of *
		=3917 ;*	HL - DE. *
		=3918 ;*	*
		=3919 ;	=====*
		=3920 ;	
		=3921 DCMP:	; Procedure DE_HL_Compare;
1ABF	C5	=3922 PUSH B	;
1AC0	47	=3923 MOV B,A	; {Save A register}
1AC1	7D	=3924 MOV A,L	;
1AC2	93	=3925 SUB E	;
1AC3	4F	=3926 MOV C,A	; Reg.C := Reg.L - Reg.E; {Ignore Borrow}
1AC4	7C	=3927 MOV A,H	;
1AC5	9A	=3928 SBB D	; Reg.A := Reg.H - Reg.D; {With Borrow}
		=3929 ;	
		=3930 ;	Have now done a complete 16 bit subtraction. HL - DE
		=3931 ;	
1AC6	DACA1A	=3932 JC DCP05	; If NOT carry then
1AC9	B1	=3933 ORA C	; If (Reg.A = 0) AND (Reg.C=0) then
		=3934 ;	SetZeroFlag;
		=3935 DCP05:	;
1ACA	78	=3936 MOV A,B	;
1ACB	C1	=3937 POP B	;
1ACC	C9	=3938 RET	; EndProc DE_HL_Compare;
		=3939 ;	
		=3940 ;	=====*
		=3941 ;*	*
		=3942 ;*	Function : INDIRECT (Used to do indirect calls). *
		=3943 ;*	Inputs : HL : Address of the routine to call. *
		=3944 ;*	Outputs : None. *
		=3945 ;*	Destroys : Depends on the routine called. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 65

LOC	OBJ	LINE	SOURCE STATEMENT
		=3946 ;*	Calls : Depends on contents of HL register pair. *
		=3947 ;*	*
		=3948 ;*	Description : This routine is used to perform an indirect call via *
		=3949 ;*	the HL register pair. The routine is used in the following way. *
		=3950 ;*	*
		=3951 ;*	The address of the routine to be called is placed in the HL *
		=3952 ;*	register pair. This routine, INDIRECT, is then called. The routine *
		=3953 ;*	called must ends with the normal return instruction and return *
		=3954 ;*	control will be returned to the instruction following the *
		=3955 ;*	CALL INDIRECT instruction. *
		=3956 ;*	*
		=3957 ;	===== *
		=3958 ;	
		=3959 INDIRECT:	; Procedure IndirectCall(Address:Integer);
		=3960	;
1ACD	E9	=3961 PCHL	; Reg.PC := Reg.HL;
		=3962	;
		=3963	; EndProc IndirectCall;;
		=3964 ;	END <DEBGEN.A85>
		3965 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 66

LOC	OBJ	LINE	SOURCE STATEMENT
		3966 ;	
		3967 ;	Get and execute command routines:
		3968 ;	
		3969 \$INCLUDE(DEBCMD.A85)	
		=3970 ;	-----;
		=3971 ;	;
		=3972 ;	Command routines used by the UCT SABus debugger.
		=3973 ;	;
		=3974 ;	This module contains (in order) the following subroutines.
		=3975 ;	;
		=3976 ;	a) DUMMYROU - Dummy routine (does not do a lot)
		=3977 ;	b) EXECMD - Execute command.
		=3978 ;	c) EXITROU - Called if the 'EXIT' command is typed.
		=3979 ;	d) GETCMD - Get command.
		=3980 ;	e) NUMROU - Called if a number is typed.
		=3981 ;	;
		=3982 ;	-----;
		=3983 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 67

LOC	OBJ	LINE	SOURCE STATEMENT
		=3984 ;	=====*
		=3985 ;*	*
		=3986 ;*	Function : DUMMYROU (Dummy routine) *
		=3987 ;*	Inputs : None. *
		=3988 ;*	Outputs : None. *
		=3989 ;*	Destroys : None. *
		=3990 ;*	Calls : None. *
		=3991 ;*	*
		=3992 ;*	Description : As this routine inputs nothing, outputs nothing, *
		=3993 ;*	destroys nothing and calls nothings, the best description of this *
		=3994 ;*	routine is that it does NOTHING. *
		=3995 ;*	*
		=3996 ;	=====*
		=3997 ;	
		=3998 DUMMYROU:	; Procedure DummyRoutine;
		=3999	;
1ACE	C9	=4000 RET	; EndProc;
		=4001 ;	
		=4002 ;	=====*
		=4003 ;*	*
		=4004 ;*	Function : EXECMD (Execute command). *
		=4005 ;*	Inputs : Command line in buffer. *
		=4006 ;*	Outputs : None. *
		=4007 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=4008 ;*	Calls : GETSYM, BYTEROU, DEFROU, GOROU, HELPROU, MAPROU, *
		=4009 ;*	PORTROU, PRINTROU, REGROU, RESETROU, DISROU, *
		=4010 ;*	EXITROU, DUMMYROU, STEPROU, CRLF, OUTMSG, PHEXW *
		=4011 ;*	and ERROR. *
		=4012 ;*	*
		=4013 ;*	Description : This routine decides what type of command has been *
		=4014 ;*	entered by the user. It does this by reading the first symbol off *
		=4015 ;*	the command line and checking it against a list of valid commands. *
		=4016 ;*	*
		=4017 ;*	The following symbols are valid as the first symbol on the command *
		=4018 ;*	line. The action that must be taken is also given. *
		=4019 ;*	*
		=4020 ;*	Symbol Action *
		=4021 ;*	----- *
		=4022 ;*	BYTE Call the BYTE command handler routine. *
		=4023 ;*	DEFINE Call the DEFINE command handler routine. *
		=4024 ;*	GO Call the GO command handler routine. *
		=4025 ;*	HELP Call the HELP command handler routine. *
		=4026 ;*	MAP Call the MAP command handler routine. *
		=4027 ;*	PORT Call the PORT command handler routine. *
		=4028 ;*	PRINT Call the PRINT command handler routine. *
		=4029 ;*	REG Call the REG command handler routine. *
		=4030 ;*	RESET Call the RESET command handler routine. *
		=4031 ;*	DIS Call the DISASSEMBLY command handler routine. *
		=4032 ;*	STEP Call the STEP command handler routine. *
		=4033 ;*	SYMBOLIC Call the SYMBOLIC routine handler routine. *
		=4034 ;*	<Number> Display the value of the number in Hexadecimal and in *
		=4035 ;*	decimal. *
		=4036 ;*	EOL Ignore - This is a blank line. *
		=4037 ;*	*
		=4038 ;	=====*
		=4039 ;	
		=4040 EXECMD:	; Procedure ExecuteCommand;
1ACF	2128EA	=4041 LXI H,CMDLIN	;
1AD2	2226EA	=4042 SHLD CLPTR	; CommandLinePointer := 0;
1AD5	3E00	=4043 MVI A,FALSE	;
1AD7	3278EA	=4044 STA ERRFLAG	; ErrorFlag := FALSE;
		=4045	;
1ADA	CDFF1B	=4046 CALL GETSYM	; Sym := GetNextSymbol;
1ADD	47	=4047 MOV B,A	;
		=4048	;
1ADE	21081B	=4049 LXI H,CMDTAB	; Ptr := 0
		=4050	;
		=4051 EXE60:	; While
1AE1	7E	=4052 MOV A,M	;
1AE2	FEFF	=4053 CPI OFFH	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 68

LOC	OBJ	LINE	SOURCE STATEMENT
1AE4	CAFC1A	=4054	JZ EXE90 ; CommandTable[Ptr].Symbol <> OFFH AND
		=4055	;
1AE7	B8	=4056	CMP B ;
1AE8	CAF11A	=4057	JZ EXE70 ; CommandTable[Ptr].Symbol <> Sym do
		=4058	;
1AEB	23	=4059	INX H ;
1AEC	23	=4060	INX H ;
1AED	23	=4061	INX H ; Ptr := Ptr + 1;
1AEE	C3E11A	=4062	JMP EXE60 ;
		=4063	;
		=4064	EXE70: ; If CommandTable[Ptr].Symbol <> OFFH then
		=4065	Begin
1AF1	23	=4066	INX H ;
1AF2	5E	=4067	MOV E,M ;
1AF3	23	=4068	INX H ;
1AF4	56	=4069	MOV D,M ; SubroutineAdr :=
		=4070	CommandTable[Ptr].Address;
1AF5	EB	=4071	XCHG ;
1AF6	CDCD1A	=4072	CALL INDIRECT ; Indirect(SubroutineAdr);
1AF9	C3021B	=4073	JMP EXE99 ;
		=4074	;
		=4075	;
		=4076	EXE90: ; End
1AFC	21031B	=4077	LXI H,SYNERR ; Else
1AFF	CDC119	=4078	CALL ERROR ; Error(SyntaxError);
		=4079	;
		=4080	EXE99: ; End;
		=4081	;
1B02	C9	=4082	RET ; EndProc ExecuteCommand;
		=4083	;
		=4084	SYNERR:
		=4085	;
		=4086	format:
		=4087	;
		=4088	Syntax error
		=4089	;
		=4090	declaration
		=4091	;
1B03	53842085	=4092	DB 'S', PSYNTAX, ' ', PSEERROR, EndMsg
	00	=4093	;
		=4094	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 69

LOC	OBJ	LINE	SOURCE STATEMENT
		=4095 ;	-----
		=4096 ;	
		=4097	CMDTAB: ; Command table
		=4098 ;	
		=4099 ;	This table contains all the valid commands and the address of the
		=4100 ;	subroutine that must be called if they are entered as the first
		=4101 ;	symbol on a line.
		=4102 ;	
		=4103 ;	The format of the table is as follows:
		=4104 ;	
		=4105 ;	Command_record
		=4106 ;	Command_record
		=4107 ;	.
		=4108 ;	.
		=4109 ;	Command_record
		=4110 ;	OFFH ; End of table marker.
		=4111 ;	
		=4112 ;	The format of each command record is as follows:
		=4113 ;	
		=4114 ;	BYTE : Symbol number (e.g. SYMGO for the 'GO' command)
		=4115 ;	WORD : Address of the subroutine to be called if the
		=4116 ;	symbol is typed.
		=4117 ;	
		=4118 ;	
		=4119 ;	Entry for the 'BYTE' command:
1B08	08	=4120	DB SYMBYTE ; SymByte
1B09	1C29	=4121	DW BYTEROU ; ByteRoutine
		=4122 ;	
		=4123 ;	Entry for the 'DEFINE' command:
		=4124 ;	
1B0B	0C	=4125	DB SYMDEFINE ; SymDefine
1B0C	BD29	=4126	DW DEFROU ; DefineRoutine
		=4127 ;	
		=4128 ;	Entry for the 'EXIT' command:
		=4129 ;	
1B0E	10	=4130	DB SYMEXIT ; SymExit
1B0F	361B	=4131	DW EXITROU ; ExitRoutine
		=4132 ;	
		=4133 ;	Entry for the 'GO' command:
		=4134 ;	
1B11	16	=4135	DB SYMGO ; SymGo
1B12	A52A	=4136	DW GOROU ; GoRoutine
		=4137 ;	
		=4138 ;	Entry for the 'HELP' command:
		=4139 ;	
1B14	17	=4140	DB SYMHELP ; SymHelp
1B15	E01F	=4141	DW HELPROU ; HelpRoutine
		=4142 ;	
		=4143 ;	Entry for the 'MAP' command:
		=4144 ;	
1B17	1D	=4145	DB SYMMAP ; SymMap
1B18	C320	=4146	DW MAPROU ; MapRoutine
		=4147 ;	
		=4148 ;	Entry for the 'PORT' command:
		=4149 ;	
1B1A	24	=4150	DB SYMPORT ; SymPort
1B1B	C72B	=4151	DW PORTROU ; PortRoutine
		=4152 ;	
		=4153 ;	Entry for the 'PRINT' command:
		=4154 ;	
1B1D	25	=4155	DB SYMPRINT ; SymPrint
1B1E	922C	=4156	DW PRINTROU ; PrintRoutine
		=4157 ;	
		=4158 ;	Entry for the 'REG' command:
		=4159 ;	
1B20	28	=4160	DB SYMREG ; SymRegister
1B21	8D24	=4161	DW REGROU ; RegisterRoutine
		=4162 ;	
		=4163 ;	Entry for the 'RESET' command:
		=4164 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 70

LOC	OBJ	LINE	SOURCE STATEMENT
1B23	29	=4165	DB SYMRESET ; SymReset
1B24	8123	=4166	DW RESETROU ; ResetRoutine
		=4167	;
		=4168	;
		=4169	Entry for the 'STEP' command:
1B26	2B	=4170	DB SYMSTEP ; SymStep
1B27	952E	=4171	DW STEProu ; StepRoutine
		=4172	;
		=4173	;
		=4174	Entry for the 'SYMBOLIC' command:
1B29	2C	=4175	DB SYMSYM ; SymSymbolic
1B2A	EA2E	=4176	DW SYMrou ; SymRoutine
		=4177	;
		=4178	;
		=4179	Entry for the 'DISASM' command:
1B2C	0D	=4180	DB SYMDIS ; SymDisassemble
1B2D	2726	=4181	DW DISrou ; DisassembleRoutine
		=4182	;
		=4183	;
		=4184	Entry for numbers
1B2F	02	=4185	DB NUMBER ; Number
1B30	7A1B	=4186	DW NUMrou ; NumberRoutine;
		=4187	;
		=4188	;
		=4189	Entry for a blank line
1B32	00	=4190	DB EOL ; EndOfLine
1B33	CE1A	=4191	DW DUMMYrou ; DummyRoutine
		=4192	;
1B35	FF	=4193	DB OFFH ; End of table marker.
		=4194 ;	
		=4195 ;	=====*
		=4196 ;*	*
		=4197 ;*	Function : EXITROU (Exit Routine) *
		=4198 ;*	Inputs : None. *
		=4199 ;*	Outputs : Sets EXITFLAG. *
		=4200 ;*	Destroys : A. *
		=4201 ;*	Calls : None. *
		=4202 ;*	*
		=4203 ;*	Description : This routine, which is called when the 'EXIT' command *
		=4204 ;*	is typed, simply sets the EXITFLAG. *
		=4205 ;*	*
		=4206 ;	=====*
		=4207 ;	
		=4208	EXITROU: ; Procedure ExitRoutine;
1B36	3EFF	=4209	MVI A,TRUE ;
1B38	3279EA	=4210	STA EXITFLAG ; ExitFlag := TRUE;
		=4211	;
1B3B	C9	=4212	RET ; EndProc;
		=4213 ;	
		=4214 ;	=====*
		=4215 ;*	*
		=4216 ;*	Function : GETCMD (Get Command). *
		=4217 ;*	Inputs : None. *
		=4218 ;*	Outputs : None. *
		=4219 ;*	Destroys : A, B, H, L and flags. *
		=4220 ;*	Calls : KEYIN, CRLF and KITOUT. *
		=4221 ;*	*
		=4222 ;*	Description : This routine reads a command line from the keyboard *
		=4223 ;*	and places it in a command line buffer. The routine supports the *
		=4224 ;*	delete key which allows the last character typed to be removed. *
		=4225 ;*	*
		=4226 ;*	NOTE: Characters other than printable ASCII character, Returns and *
		=4227 ;*	the delete key are ignored. *
		=4228 ;*	*
		=4229 ;	=====*
		=4230 ;	
		=4231	GETCMD: ; Procedure GetCommand;
		=4232	;
1B3C	2128EA	=4233	LXI H,CMDLIN ; Ptr := CommandLine;
1B3F	0600	=4234	MVI B,0 ; CharCount := 0;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 71

LOC	OBJ	LINE	SOURCE STATEMENT
		=4235	;
		=4236	GETC05: ; Repeat
1B41	CDD519	=4237	CALL KEYIN ; Ch := KeyInput;
1B44	FE7F	=4238	CPI DEL ;
1B46	C2621B	=4239	JNZ GET10 ; If Ch = DEL then
		=4240	Begin ;
1B49	78	=4241	MOV A,B ;
1B4A	B7	=4242	ORA A ;
1B4B	CA411B	=4243	JZ GETC05 ; If CharCount <> 0 then
		=4244	Begin ;
1B4E	05	=4245	DCR B ; CharCount := CharCount - 1;
1B4F	2B	=4246	DCX H ; Ptr := Ptr - 1;
1B50	3E08	=4247	MVI A,BS ;
1B52	CDE218	=4248	CALL KITOUT ; KitOut(BackSpace);
1B55	3E20	=4249	MVI A,SPACE ;
1B57	CDE218	=4250	CALL KITOUT ; KitOut(Space);
1B5A	3E08	=4251	MVI A,BS ;
1B5C	CDE218	=4252	CALL KITOUT ; KitOut(BackSpace)
1B5F	C3411B	=4253	JMP GETC05 ; End
		=4254	GET10: ; End
		=4255	;
1B62	FE0D	=4256	CPI CR ;
1B64	C2701B	=4257	JNZ GET15 ; Else If Ch = CR then
		=4258	Begin ;
		=4259	;
1B67	361A	=4260	MVI M,EOFMARK ; CommandLine[Ptr] := EOFMARK;
1B69	CDB419	=4261	CALL CRLF ; NewLine;
1B6C	78	=4262	MOV A,B ;
1B6D	C3791B	=4263	JMP GETC20 ; End
		=4264	;
		=4265	GET15: ; Else
		=4266	Begin ;
1B70	77	=4267	MOV M,A ; CommandLine[Ptr] := Ch;
1B71	23	=4268	INX H ; Ptr := Ptr + 1;
1B72	04	=4269	INR B ; CharCount := CharCount + 1;
1B73	CDE218	=4270	CALL KITOUT ; KitOut(Ch);
1B76	C3411B	=4271	JMP GETC05 ; End;
		=4272	;
		=4273	GETC20: ; Until Ch = CR;
		=4274	;
1B79	C9	=4275	RET ; EndProc GetCommand;
		=4276	;
		=4277	*****
		=4278	;
		=4279	Function : NUMROU (Number routine) *
		=4280	Inputs : None. *
		=4281	Outputs : None. *
		=4282	Destroys : A, H, L and flags. *
		=4283	Calls : GETSYM, CRLF, OUTMSG, PHEXW and ERROR. *
		=4284	;
		=4285	Description : This routine is called if the first symbol read from *
		=4286	the command line is a number. It could be a symbol preceded by a *
		=4287	DOT, e.g. '.BUFFER', or it could be a number (in any base), e.g. *
		=4288	'01A7H' or 1234. This routine checks that it is the only symbol on *
		=4289	on the command line. *
		=4290	;
		=4291	The syntax for the 'NUMBER' command (it is not really a command, *
		=4292	but rather a way of determining the values of symbol and *
		=4293	converting numbers to hexadecimal) is: *
		=4294	;
		=4295	<Number> <Eol> *
		=4296	;
		=4297	*****
		=4298	;
		=4299	NUMROU: ; Procedure NumberRoutine;
		=4300	;
1B7A	CDFF1B	=4301	CALL GETSYM ; Sym := GetNextSymbol;
1B7D	FE00	=4302	CPI EOL ;
1B7F	C2941B	=4303	JNZ NUMR10 ; If Sym = EOL then
		=4304	Begin ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 72

LOC	OBJ	LINE	SOURCE STATEMENT
1B82	219B1B	=4305	LXI H,VALSTR ;
1B85	CDEE18	=4306	CALL OUTMSG ; OutMessage('Value = \$');
1B88	2A00EA	=4307	LHLD SYMVAL ;
1B8B	CD1D1A	=4308	CALL PHEXW ; PrintHexWord(SymValue);
		=4309	;
1B8E	CDB419	=4310	CALL CRLF ; NewLine
1B91	C39A1B	=4311	JMP NUMR99 ; End
		=4312	NUMR10: ; Else
1B94	21031B	=4313	LXI H,SYNERR ;
1B97	CDC119	=4314	CALL ERROR ; Error(SyntaxError)
		=4315	NUMR99: ;
1B9A	C9	=4316	RET ; EndProc;
		=4317	;
		=4318	VALSTR:
		=4319	;
		=4320	format:
		=4321	;
		=4322	Value = \$
		=4323	;
		=4324	declaration:
		=4325	;
1B9B	56616C75	=4326	DB 'Valu', 'e'+MSB, '='+MSB, '\$', EndMsg
	E5BD2400		
		=4327	;
		=4328	END <DEBCMD.A85>
		4329	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 73

LOC	OBJ	LINE	SOURCE STATEMENT
		4330 ;	
		4331 ;	Get symbol routine:
		4332 ;	
		4333	\$INCLUDE(DEBGETSYM.A85)
		=4334 ;	-----;
		=4335 ;	
		=4336 ;	DEBGETSYM module for the UCT SABus debugger
		=4337 ;	-----;
		=4338 ;	
		=4339 ;	This module contains (in order) the following subroutines:
		=4340 ;	
		=4341 ;	a) COPYSYM - Copy a symbol from the input buffer to the
		=4342 ;	symbol buffer.
		=4343 ;	b) DIN - Digit In, Places a digit in the digit buffer.
		=4344 ;	c) DINIT - Digit buffer initialize.
		=4345 ;	d) DOUT - Digit Out, Removes a digit from the buffer. The
		=4346 ;	digit buffer is a LAST-IN FIRST-OUT buffer.
		=4347 ;	e) GETSYM - Fetch the next symbol from the command line.
		=4348 ;	f) LABCMP - Compare the label in the label buffer with the
		=4349 ;	one in the symbol table.
		=4350 ;	g) LASTCH - Return the last character fetched to the input
		=4351 ;	stream.
		=4352 ;	h) LOOKUP - Lookup a symbol in the symbol table.
		=4353 ;	i) MULT - Multiply two 16 bit number to give a 16 bit
		=4354 ;	result. Note: overflow is ignored.
		=4355 ;	j) NEXTCH - Fetch the next character from the command line.
		=4356 ;	k) QHNUM - Check for Hexadecimal Number.
		=4357 ;	l) QLETTER - Check for letter ('A'..'Z' and 'a'..'z')
		=4358 ;	m) QLLETTER - Check for label letter (Letter, 0..9 and '_')
		=4359 ;	n) QNUM - Check for number ('0'..'9')
		=4360 ;	o) READNU - Read a number, of any base from the input line.
		=4361 ;	p) RESERVED - Check the reserved word table.
		=4362 ;	q) SHIFTL - Shift the HL register pair one bit left.
		=4363 ;	r) SYMTOLAB - Copy the entry in the symbol buffer to the label
		=4364 ;	buffer.
		=4365 ;	
		=4366 ;	-----;
		=4367	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=4368 ;	=====*
		=4369 ;*	*
		=4370 ;*	Function : COPYSYM (Copy symbol) *
		=4371 ;*	Inputs : A = First character *
		=4372 ;*	B = Length of buffer *
		=4373 ;*	HL = Start address of buffer *
		=4374 ;*	Outputs : The symbol in the buffer. *
		=4375 ;*	Destroys : A, B, C, H, L and Flags. *
		=4376 ;*	Calls : NEXTCH, LASTCH and QLETTER *
		=4377 ;*	*
		=4378 ;*	Description : This routine copies a symbol from the input line to *
		=4379 ;*	buffer specified. The maximum length of the symbol is also given. *
		=4380 ;*	If the length of the symbol on the input line exceeds this maximum *
		=4381 ;*	length then the extra characters are ignored. *
		=4382 ;*	*
		=4383 ;	=====*
		=4384 ;	
		=4385 COPYSYM:	; Procedure CopySymbol(BufferAdr:Integer;
		=4386	; BufferLen:Byte;
		=4387	; FirstChar:Byte);
1BA3	E5	=4388	PUSH H
1BA4	C5	=4389	PUSH B
		=4390 COPS05:	; For I := 1 to BufferLen do
		=4391	MVI M,SPACE ; Memory[BufferAdr+I-1] := Space;
1BA7	23	=4392	INX H
1BA8	05	=4393	DCR B
1BA9	C2A51B	=4394	JNZ COPS05
		=4395	;
1BAC	C1	=4396	POP B ; CharCount := BufferLen;
1BAD	E1	=4397	POP H ; BufPtr := BufferAdr;
		=4398	;
1BAE	77	=4399	MOV M,A ; Memory[BufPtr] := FirstChar;
1BAF	23	=4400	INX H ; BufPtr := BufPtr + 1;
1BB0	05	=4401	DCR B ; CharCount := CharCount - 1;
		=4402 COPS10:	; Repeat
1BB1	CD1B1D	=4403	CALL NEXTCH ; Ch := NextCh;
1BB4	4F	=4404	MOV C,A ;
1BB5	CD601D	=4405	CALL QLETTER ; If ?LabelLetter(CH) then
1BB8	C2C71B	=4406	JNZ COPS20 ; Begin
		=4407	;
1BBB	78	=4408	MOV A,B ;
1BBC	FE00	=4409	CPI 0 ; If CharCount <> 0 then
1BBE	CAB11B	=4410	JZ COPS10 ; Begin
		=4411	;
1BC1	05	=4412	DCR B ; CharCount := CharCount-1;
1BC2	71	=4413	MOV M,C ; Memory[BufPtr] := Ch;
1BC3	23	=4414	INX H ; BufPtr := BufPtr + 1;
		=4415	;
		=4416	End;
1BC4	C3B11B	=4417	JMP COPS10 ; Until Not ?Letter(CH);
		=4418 COPS20:	;
1BC7	CDB71C	=4419	CALL LASTCH ; LastCh;
1BCA	C9	=4420	RET ; EndProc CopySymbol;
		=4421 ;	
		=4422 ;	=====*
		=4423 ;*	*
		=4424 ;*	Function : DIN (Digit Input). *
		=4425 ;*	Inputs : A : Digit to be put into the digit buffer. *
		=4426 ;*	Outputs : None. *
		=4427 ;*	Destroys : H, L, A and Flags. *
		=4428 ;*	Calls : None. *
		=4429 ;*	*
		=4430 ;*	Description : This routine puts a digit into the digit buffer. It *
		=4431 ;*	also updates the Digit buffer input pointer and the digit counter. *
		=4432 ;*	*
		=4433 ;*	Note: The digit buffer is a LAST-IN FIRST-OUT buffer used during *
		=4434 ;*	the evaluation of numbers. *
		=4435 ;*	*
		=4436 ;	=====*
		=4437 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 75

LOC	OBJ	LINE	SOURCE STATEMENT
		=4438	DIN: ; Procedure DigitIn(Dig:Byte);
1BCB	2A12EA	=4439	LHLD DIGIPTR ;
1BCE	77	=4440	MOV M,A ; Memory[DigitBufferInputPointer] := Dig;
1BCF	23	=4441	INX H ;
1BD0	2212EA	=4442	SHLD DIGIPTR ; DigitBufferInputPointer :=
		=4443	; DigitBufferPointer + 1;
1BD3	3A11EA	=4444	LDA DIGCOUNT ;
1BD6	3C	=4445	INR A ;
1BD7	3211EA	=4446	STA DIGCOUNT ; DigitCounter := DigitCounter + 1;
		=4447	; EndProc DigitIn;
1BDA	C9	=4448	RET ; EndProc DigitIn;
		=4449	;
		=4450	=====*
		=4451	;
		=4452	Function : DINIT (Digit buffer initialise). *
		=4453	Inputs : None. *
		=4454	Outputs : None. *
		=4455	Destroys : H, L and A. *
		=4456	Calls : None. *
		=4457	;
		=4458	Description : This routine initialises the LAST-IN FIRST-OUT buffer *
		=4459	used by the number evaluation routine. *
		=4460	;
		=4461	=====*
		=4462	;
		=4463	DINIT: ; Procedure DigitInitialise;
1BDB	2116EA	=4464	LXI H,DIGBUF ;
1BDE	2212EA	=4465	SHLD DIGIPTR ; DigitBufferInputPointer := DigitBuffer;
1BE1	2214EA	=4466	SHLD DIGOPTR ; DigitBufferOutputPointer := DigitBuffer;
1BE4	3E00	=4467	MVI A,0 ;
1BE6	3211EA	=4468	STA DIGCOUNT ; DigitCount := 0;
		=4469	;
1BE9	C9	=4470	RET ; EndProc DigitInitialise;
		=4471	;
		=4472	=====*
		=4473	;
		=4474	Function : DOUT (Digit Out). *
		=4475	Inputs : None. *
		=4476	Outputs : S = 1 : No more digits in the buffers. *
		=4477	S = 0 : Digits in the buffer. *
		=4478	A : The value of the digit fetched from the buffer. *
		=4479	Destroys : A and Flags. *
		=4480	Calls : None. *
		=4481	;
		=4482	Description : This routine fetches characters from the LAST-IN *
		=4483	FIRST-OUT digit buffer. If there are no more digits in the buffer *
		=4484	then a negative number is returned. *
		=4485	;
		=4486	=====*
		=4487	;
		=4488	DOUT: ; Procedure DigitOut;
1BEA	E5	=4489	PUSH H ;
1BEB	3A11EA	=4490	LDA DIGCOUNT ;
1BEE	3D	=4491	DCR A ;
1BEF	3211EA	=4492	STA DIGCOUNT ; DigitCount := DigitCount - 1;
1BF2	FAFD1B	=4493	JM DOUT05 ; If DigitCount >= 0 then
		=4494	;
		=4495	; Begin
1BF5	2A14EA	=4496	LHLD DIGOPTR ;
1BF8	7E	=4497	MOV A,M ; Digit :=
		=4498	memory[DigitBufferOutputPointer];
1BF9	23	=4499	INX H ;
1BFA	2214EA	=4500	SHLD DIGOPTR ; DigitBufferOutputPointer :=
		=4501	DigitBufferOutputPointer + 1;
		=4502	;
		=4503	DOUT05: ; End;
1BFD	E1	=4504	POP H ;
1BFE	C9	=4505	RET ; EndProc DigitOut;
		=4506	;
		=4507	=====*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 76

LOC	OBJ	LINE	SOURCE STATEMENT
		=4508 ;*	
		=4509 ;*	Function : GETSYM (Get the next symbol of the command line)
		=4510 ;*	Inputs : None.
		=4511 ;*	Outputs : A : Symbol type. (Also stored in the memory variable
		=4512 ;*	SymType.
		=4513 ;*	
		=4514 ;*	If the symbol type is a number then its value will be stored in the
		=4515 ;*	memory variable SymValue.
		=4516 ;*	
		=4517 ;*	If the symbol type is a string then the string is contained in the
		=4518 ;*	label buffer.
		=4519 ;*	
		=4520 ;*	Destroys : A and Flags.
		=4521 ;*	Calls : NEXTCH, QLETTE, COPYSYM, RESERVED, SYMTOLAB, QNUM,
		=4522 ;*	READNU and LOOKUP.
		=4523 ;*	
		=4524 ;*	Description : This routine fetches the next Symbol from the command
		=4525 ;*	line. A symbol can fall into a number of different classes. These
		=4526 ;*	are listed below:
		=4527 ;*	
		=4528 ;*	End_of_line : If there are no more character on the command line
		=4529 ;*	then this symbol type is returned.
		=4530 ;*	
		=4531 ;*	Number : This can either be a numerical entry or a symbol that has
		=4532 ;*	been looked-up in the symbol table. (e.g. 21H or .FRED)
		=4533 ;*	
		=4534 ;*	String : This is a string of characters that is not recognised as
		=4535 ;*	being one of the reserved words. (e.g. FRED)
		=4536 ;*	
		=4537 ;*	Reserved word : This is a string of characters that is recognised
		=4538 ;*	as one of the reserved words. (e.g. BYTE)
		=4539 ;*	
		=4540 ;*	A special character : these are single characters that have
		=4541 ;*	special meanings. (e.g. '+' or '-')
		=4542 ;*	
		=4543 ;*	Note: Space compression is performed by this routine.
		=4544 ;*	
		=4545 ;*	=====
		=4546 ;*	
		=4547	GETSYM: ; Procedure GetSym;
1BFF	C5	=4548	PUSH B ;
1C00	D5	=4549	PUSH D ;
1C01	E5	=4550	PUSH H ;
		=4551	GETS01: ; Repeat
1C02	CD1B1D	=4552	CALL NEXTCH ; Ch := GetNextCharacter;
1C05	FE20	=4553	CPI SPACE ;
1C07	CA021C	=4554	JZ GETS01 ; Until Ch <> Space;
		=4555	;
1C0A	47	=4556	MOV B,A ;
1C0B	FE1A	=4557	CPI EOFMARK ; If Ch = EndOfFile then
1C0D	C2151C	=4558	JNZ GETS02 ;
		=4559	;
		=4560	MVI A,EOL ; SymType := EndOfLine
1C10	3E00	=4561	JMP GETS99 ; End;
1C12	C3961C	=4562	GETS02: ;
		=4563	CALL QLETTE ; ElseIf Letter(Ch) then
1C15	CD4D1D	=4564	JNZ GETS04 ;
1C18	C2341C	=4565	;
		=4566	MOV A,B ;
1C1B	78	=4567	LXI H,SYMBUF ;
1C1C	2103EA	=4568	MVI B,SYMBLEN ;
1C1F	0608	=4569	CALL COPYSYM ; CopySymbol(SymBuf, SymBufferLen, Ch);
1C21	CDA31B	=4570	CALL RESERVED ; SymType:= CheckReservedWordTable;
1C24	CD2D1E	=4571	CPI UNKNOWN ; If SymType = Unknown then
1C27	FE01	=4572	JNZ GETS03 ;
1C29	C2311C	=4573	;
		=4574	CALL SYMTOLAB ; Begin
1C2C	CD6B1F	=4575	MVI A,STRING ; CopySymbolToLabel;
1C2F	3E03	=4576	GETS03: ; SymType = String
		=4577	JMP GETS99 ; End;
1C31	C3961C	=4577	JMP GETS99 ; End

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 77

LOC	OBJ	LINE	SOURCE STATEMENT
		=4578	GETS04: ;
1C34	78	=4579	MOV A,B ;
1C35	CD7B1D	=4580	CALL QNUM ; ElseIf Number(Ch) then
1C38	FA501C	=4581	JM GETS05 ;
		=4582	; Begin
1C3B	CD8E1D	=4583	CALL READNU ;
1C3E	2200EA	=4584	SHLD SYMVAL ; SymVal := ReadNumber:
		=4585	;
1C41	3A78EA	=4586	LDA ERRFLAG ;
1C44	FE00	=4587	CPI FALSE ;
1C46	3E02	=4588	MVI A,NUMBER ; SymType := Number;
		=4589	;
1C48	CA4D1C	=4590	JZ GETSNE ; If ErrorFlags <> FALSE then
		=4591	;
1C4B	3E01	=4592	MVI A,UNKNOWN ; SymType := Unknown;
		=4593	GETSNE: ;
1C4D	C3961C	=4594	JMP GETS99 ; end
		=4595	GETS05: ;
1C50	78	=4596	MOV A,B ;
1C51	FE2E	=4597	CPI '.' ; ElseIf Ch := '.' then
1C53	C2761C	=4598	JNZ GETS10 ;
		=4599	; Begin
1C56	CD1B1D	=4600	CALL NEXTCH ; Ch := GetNextCharacter;
1C59	210BEA	=4601	LXI H,LABBUF ;
1C5C	0606	=4602	MVI B,LABLEN ;
1C5E	CDA31B	=4603	CALL COPYSYM ; CopySymbol(LabBuf, LabLen, Ch);
		=4604	;
1C61	CDCA1C	=4605	CALL LOOKUP ;
1C64	2200EA	=4606	SHLD SYMVAL ; SymVal := Lookup;
		=4607	;
1C67	3A78EA	=4608	LDA ERRFLAG ;
1C6A	FE00	=4609	CPI FALSE ;
1C6C	3E02	=4610	MVI A,NUMBER ; SymType := Number
		=4611	;
1C6E	CA731C	=4612	JZ GETS07 ; If ErrorFlag then
		=4613	;
1C71	3E01	=4614	MVI A,UNKNOWN ; SymType := Unknown;
		=4615	GETS07: ;
1C73	C3961C	=4616	JMP GETS99 ; end
		=4617	GETS10: ;
1C76	FE3D	=4618	CPI '=' ; ElseIf Ch := '=' then
1C78	C2801C	=4619	JNZ GETS20 ;
		=4620	; Begin
1C7B	3E04	=4621	MVI A,EQUALS ; SymType := Number
1C7D	C3961C	=4622	JMP GETS99 ; end
		=4623	GETS20: ;
1C80	FE2D	=4624	CPI '-' ; ElseIf Ch := '-' then
1C82	C28A1C	=4625	JNZ GETS30 ;
		=4626	; Begin
1C85	3E06	=4627	MVI A,MINUS ; SymType := Minus
1C87	C3961C	=4628	JMP GETS99 ; End
		=4629	GETS30: ;
1C8A	FE2B	=4630	CPI '+' ; ElseIf Ch := '+' then
1C8C	C2941C	=4631	JNZ GETS40 ;
		=4632	; Begin
1C8F	3E05	=4633	MVI A,PLUS ; SymType := Plus
1C91	C3961C	=4634	JMP GETS99 ; End
		=4635	GETS40: ;
		=4636	; Else
1C94	3E01	=4637	MVI A,UNKNOWN ; SymType := Unknown;
		=4638	GETS99: ;
1C96	3202EA	=4639	STA SYMTYPE ;
1C99	E1	=4640	POP H ;
1C9A	D1	=4641	POP D ;
1C9B	C1	=4642	POP B ;
1C9C	C9	=4643	RET ; Endproc GetSym;
		=4644	;
		=4645	;=====*
		=4646	;*
		=4647	* Function : LABCMP (Label compare). *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 78

LOC	OBJ	LINE	SOURCE STATEMENT
		=4648 ;*	Inputs : HL : Pointer to entry in the symbol table. *
		=4649 ;*	Contents of the label buffer. *
		=4650 ;*	Outputs : Z = 1 : If the entry in the symbol table is equal to *
		=4651 ;*	the symbol in the label buffer, else Z = 0. *
		=4652 ;*	Destroys : A and Flags. *
		=4653 ;*	Calls : None. *
		=4654 ;*	*
		=4655 ;*	Description : This routine compares two strings. The first is the *
		=4656 ;*	string contained in the label buffer (Start address: LABBUF and *
		=4657 ;*	Length: LABLEN). The second is the string field of the current *
		=4658 ;*	entry in the symbol table. The pointer to this entry is passed in *
		=4659 ;*	the HL register pair. *
		=4660 ;*	*
		=4661 ;*	NOTE: Some of the characters in the second string, the symbol table *
		=4662 ;*	entry, may have their top bit, D7, set and should therefore *
		=4663 ;*	be ANDed with the ASCII mask 01111111B before they are used. *
		=4664 ;*	*
		=4665 ;*	===== *
		=4666 ;	*
		=4667	LABCMP: ; Procedure LabelCompare;
1C9D	C5	=4668	PUSH B ;
1C9E	D5	=4669	PUSH D ;
1C9F	E5	=4670	PUSH H ;
1CA0	0606	=4671	MVI B,LABLEN ; Ptr := LabelLength;
1CA2	110BEA	=4672	LXI D,LABBUF ;
1CA5	EB	=4673	XCHG ;
		=4674	LCMP05: ; Repeat
1CA6	1A	=4675	LDAX D ;
1CA7	E67F	=4676	ANI ASCII ; LabCh := LabelBuffer[Ptr] AND ASCII;
1CA9	BE	=4677	CMP M ; SymCh := SymbolEntry[Ptr];
1CAA	C2B31C	=4678	JNZ LCMP10 ;
		=4679	;
1CAD	23	=4680	INX H ;
1CAE	13	=4681	INX D ;
1CAF	05	=4682	DCR B ; Ptr := Ptr - 1;
		=4683	;
1CB0	C2A61C	=4684	JNZ LCMP05 ; Until (Ptr = 0) or (LabCh <> SymCh)
		=4685	;
		=4686	LCMP10: ; If Ptr = 0 then return(True) else
1CB3	E1	=4687	POP H ; return(False);
1CB4	D1	=4688	POP D ;
1CB5	C1	=4689	POP B ;
1CB6	C9	=4690	RET ; EndProc LabelCompare;
		=4691 ;	*
		=4692 ;*	===== *
		=4693 ;*	*
		=4694 ;*	Function : LASTCH (Last character) *
		=4695 ;*	Inputs : None. *
		=4696 ;*	Outputs : None. *
		=4697 ;*	Destroys : A and Flags. *
		=4698 ;*	Calls : None. *
		=4699 ;*	*
		=4700 ;*	Description : This routine effectively returns the last character *
		=4701 ;*	fetched to the input stream and refetches the previous character. *
		=4702 ;*	This is done simply by decrementing the input pointer. The only *
		=4703 ;*	problem occurs if the input pointer has reached the last character *
		=4704 ;*	in the line. In this case the pointer is not decremented. *
		=4705 ;*	*
		=4706 ;*	===== *
		=4707 ;	*
		=4708	LASTCH: ; Procedure LastCh;
1CB7	E5	=4709	PUSH H ;
1CB8	2A26EA	=4710	LHLD CLPTR ;
1CBB	7E	=4711	MOV A,M ;
1CBC	FE1A	=4712	CPI EOFMARK ;
1CBE	CAC21C	=4713	JZ ONLY1 ; If CommandLine[CommandLinePtr] <> EOF then
		=4714	;
1CC1	2B	=4715	DCX H ; CommandLinePtr := CommandLinePtr - 1;
		=4716	ONLY1: ;
1CC2	2B	=4717	DCX H ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 79

LOC	OBJ	LINE	SOURCE STATEMENT
1CC3	7E	=4718	MOV A,M ; Ch := CommandLine[CommandLinePtr-1];
1CC4	23	=4719	INX H ;
1CC5	2226EA	=4720	SHLD CLPTR ;
1CC8	E1	=4721	POP H ;
1CC9	C9	=4722	RET ; EndProc LastCh;
		=4723 ;	
		=4724 ;	
		=4725 ;*	
		=4726 ;*	Function : Lookup (Lookup a symbol in the symbol table). *
		=4727 ;*	Inputs : Contents of the label buffer. *
		=4728 ;*	Outputs : ErrFlag = False : If symbol is found. *
		=4729 ;*	Errflag = True : If the symbol does no exist. *
		=4730 ;*	HL = Value of symbol. *
		=4731 ;*	
		=4732 ;*	Destroys : A, B, D, E, H, L and Flags. *
		=4733 ;*	Calls : LABCMP. *
		=4734 ;*	
		=4735 ;*	Description : This routine tries to lookup the symbol contained in *
		=4736 ;*	label buffer in the symbol table. Although the table was created *
		=4737 ;*	using a hashing technique, a linear search is used. If the symbol *
		=4738 ;*	is the label buffer matches any of the entries in the symbol table *
		=4739 ;*	then its value is fetched and returned in the HL register pair. *
		=4740 ;*	
		=4741 ;*	The format of the symbol table is described below: *
		=4742 ;*	
		=4743 ;*	The symbol table is an array of 256 eight bytes entries. Each *
		=4744 ;*	entry has two fields. These are the string field and the value *
		=4745 ;*	field. The string field is six characters long, left justified and *
		=4746 ;*	space padded. This field contains the symbol. The second field, a *
		=4747 ;*	16 bit word, contains the value of the symbol. The start address *
		=4748 ;*	of the table is given by the symbol STABLE. *
		=4749 ;*	
		=4750 ;	
		=4751 ;	
		=4752	LOOKUP: ; Procedure Lookup;
1CCA	110800	=4753	LXI D,LABLEN + WORD ;
1CCD	2100E0	=4754	LXI H,STABLE ;
1CD0	0600	=4755	MVI B,0 ; EntryPtr := 256;
		=4756	LUP05: ;
1CD2	CD9D1C	=4757	CALL LABCMP ; While (SymbolTable[EntryPtr].String <>
1CD5	CAE61C	=4758	JZ LUP10 ; LabelBuffer) and (EntryPtr <> 0) do
		=4759	;
1CD8	19	=4760	DAD D ;
1CD9	05	=4761	DCR B ; EntryPtr := EntryPtr - 1;
1CDA	C2D21C	=4762	JNZ LUP05 ;
		=4763	;
1CDD	21EF1C	=4764	LXI H,UNDEF ; If EntryPtr = 0 then
1CE0	CDC119	=4765	CALL ERROR ; ErrorMessage('Undefined symbol')
1CE3	C3EE1C	=4766	JMP LUP15 ;
		=4767	LUP10: ; Else
1CE6	110600	=4768	LXI D,LABLEN ;
1CE9	19	=4769	DAD D ;
1CEA	5E	=4770	MOV E,M ;
1CEB	23	=4771	INX H ;
1CEC	56	=4772	MOV D,M ;
1CED	EB	=4773	XCHG ; Return(SymbolTable[EntryPtr].Value);
		=4774	LUP15: ;
1CEE	C9	=4775	RET ; EndProc Lookup;
		=4776 ;	
		=4777	UNDEF: ;
		=4778 ;	
		=4779 ;	format: ;
		=4780 ;	
		=4781 ;	Undefined symbol ;
		=4782 ;	
		=4783 ;	declaration: ;
		=4784 ;	
1CEF	556E6465 66696E65 E473796D	=4785	DB 'Undefine', 'd'+MSB, 'symbol', EndMsg

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 80

LOC	OBJ	LINE	SOURCE STATEMENT
		626F6C00	
		=4786 ;	
		=4787 ;=====*	
		=4788 ;*	
		=4789 ;* Function : MULT (Multiply).	
		=4790 ;* Inputs : HL and DE are the two 16 bit numbers to be multiplied.	
		=4791 ;* Outputs : HL = 16 bit result.	
		=4792 ;* Destroys : H, L, D, E, A and Flags.	
		=4793 ;* Calls : SHIFTL.	
		=4794 ;*	
		=4795 ;* Description : This routine multiplies the two 16 bit numbers passed	
		=4796 ;* in the HL and DE register pairs. The 16 bit result is returned in	
		=4797 ;* the HL register pair. Overflow is ignored. The routine used the	
		=4798 ;* standard shift and add method as used in long multiplication.	
		=4799 ;*	
		=4800 ;=====*	
		=4801 ;	
		=4802 ; HL : Number1	
		=4803 ; DE : Number2	
		=4804 ;	
		=4805 MULT:	; Procedure Mult(Number1, Number2 : Integer);
1CFF	C5	=4806	PUSH B ;
1D00	44	=4807	MOV B,H ;
1D01	4D	=4808	MOV C,L ;
		=4809	;
1D02	3E10	=4810	MVI A,BITSPW ;
		=4811	;
1D04	210000	=4812	LXI H,0000H ; Result := 0;
		=4813	;
		=4814 NXTBIT:	; For I := 1 to BitsPerWord do
		=4815	Begin
1D07	F5	=4816	PUSH PSW ;
1D08	CD621F	=4817	CALL SHIFTL ; Result := Result SHL 1;
1D0B	EB	=4818	XCHG ;
1D0C	CD621F	=4819	CALL SHIFTL ; Number2 := Number2 SHL 1;
1D0F	EB	=4820	XCHG ;
1D10	D2141D	=4821	JNC NOADD ; If CarryOutOfLastShift then
		=4822	;
1D13	09	=4823	DAD B ; Result := Result + Number1;
		=4824 NOADD:	;
1D14	F1	=4825	POP PSW ;
1D15	3D	=4826	DCR A ;
1D16	C2071D	=4827	JNZ NXTBIT ; End;
		=4828	;
1D19	C1	=4829	POP B ;
		=4830	Return(Result)
		=4831	;
1D1A	C9	=4832	RET ; EndProc Mult;
		=4833 ;	
		=4834 ;=====*	
		=4835 ;*	
		=4836 ;* Function : NEXTCH (Next Character).	
		=4837 ;* Inputs : None.	
		=4838 ;* Outputs : A : Character fetched from the input line. If there are	
		=4839 ;* no more characters on the input line the end of	
		=4840 ;* file character EOFMARK is returned.	
		=4841 ;* Destroys : A and Flags.	
		=4842 ;* Calls : None.	
		=4843 ;*	
		=4844 ;* Description : This routine fetches the next character from the	
		=4845 ;* command line and increments the pointer to the next character to	
		=4846 ;* be fetched. If the character fetched is the end of file marker	
		=4847 ;* then the pointer is not incremented and all future calls to the	
		=4848 ;* routine will continue to return the EOF mark.	
		=4849 ;*	
		=4850 ;=====*	
		=4851 ;	
		=4852 NEXTCH:	; Procedure NextCh : Byte;
1D1B	E5	=4853	PUSH H ;
1D1C	2A26EA	=4854	LHLD CLPTR ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 81

LOC	OBJ	LINE	SOURCE STATEMENT
1D1F	7E	=4855	MOV A,M ; Ch := Memory[CommandLinePointer];
1D20	FE1A	=4856	CPI EOFMARK ;
1D22	CA291D	=4857	JZ NOINC ; If Ch <> EOFMARK then
		=4858	;
1D25	23	=4859	INX H ;
1D26	2226EA	=4860	SHLD CLPTR ; CommandLinePointer :=
		=4861	; CommandLinePointer - 1;
		=4862	NOINC: ;
1D29	E1	=4863	POP H ; Return(Ch);
		=4864	;
1D2A	C9	=4865	RET ; EndProc NextCh;
		=4866	;
		=4867	=====*
		=4868	;
		=4869	;* Function : QHNUM (? Hexadecimal number) *
		=4870	;* Inputs : A = Digit to be checked. *
		=4871	;* Outputs : A = value of digit, if the character is a valid Hex *
		=4872	digit (i.e. 0 to 9 and A to F). *
		=4873	;* If the character is outside these ranges then a *
		=4874	negative value is returned. *
		=4875	;* Destroys : Acc and Flags. *
		=4876	;* Calls : None. *
		=4877	;* *
		=4878	;* Description: This routine is used by the READNU routine to check *
		=4879	that the characters is reads are valid digits. *
		=4880	;* *
		=4881	=====*
		=4882	;
		=4883	QHNUM: ; Procedure ?HexNumber(Ch:Byte);
		=4884	;
		=4885	Check that the character is in the range '0' to 'F'
		=4886	;
1D2B	FE30	=4887	CPI '0' ;
1D2D	FA491D	=4888	JM ERRN ; If Char < '0' then not a number.
		=4889	;
1D30	FE47	=4890	CPI 'F'+1 ;
1D32	F2491D	=4891	JP ERRN ; If char >= 'F'+1 then not a number.
		=4892	;
		=4893	Now check to see if it is a Letter (A to F) or
		=4894	a Limited number (0 to 9).
		=4895	;
		=4896	;
1D35	FE3A	=4897	CPI '9'+1 ;
1D37	FA441D	=4898	JM NUM ; If Character < '9'+1 then it's a Limited num.
		=4899	;
1D3A	FE41	=4900	CPI 'A' ;
1D3C	FA491D	=4901	JM ERRN ; If Character < 'A' then it's not a number.
		=4902	;
		=4903	LETTER: ; Else it's in the range 'A' to 'F'
1D3F	D637	=4904	SUI 'A'-10 ;
1D41	C34C1D	=4905	JMP QNUMEX ; => value = Character - 'A' + 10
		=4906	;
		=4907	NUM: ; Character in the range '0' to '9'
1D44	D630	=4908	SUI '0' ;
1D46	C34C1D	=4909	JMP QNUMEX ; => value = character - '0'
		=4910	;
		=4911	ERRN: ; Character not in range
1D49	3EFF	=4912	MVI A,0FFH ;
1D4B	B7	=4913	ORA A ; => value = -1
		=4914	QNUMEX: ;
1D4C	C9	=4915	RET ; EndProc ?HexNumber;
		=4916	;
		=4917	=====*
		=4918	;
		=4919	;* Function : QLETTER (?Letter) *
		=4920	;* Inputs : A : Character to be checked. *
		=4921	;* Outputs : Z = 1 : Character is a letter. *
		=4922	;* Z = 0 : The character is not in the range 'A'..'Z' *
		=4923	;* Destroys : A and Flags. *
		=4924	;* Calls : None. *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 82

LOC	OBJ	LINE	SOURCE STATEMENT
		=4925 ;*	
		=4926 ;*	Description : This routine checks that the character passed in the
		=4927 ;*	A register is in the range 'A' to 'Z'. Only uppercase characters
		=4928 ;*	are checked as all characters are converted to uppercase in the
		=4929 ;*	command line input routine.
		=4930 ;*	
		=4931 ;	=====
		=4932 ;	
		=4933	QLETTER: ; Procedure ?Letter(Ch:Byte) : Boolean;
1D4D	FE41	=4934	CPI 'A' ;
1D4F	FA5C1D	=4935	JM QLET05 ; If Ch >= 'A' AND
		=4936	;
1D52	FE5B	=4937	CPI 'Z'+1 ;
1D54	F25C1D	=4938	JP QLET05 ; Ch < 'Z'+1 Then
		=4939	;
1D57	3E00	=4940	MVI A,0 ; ReturnValue := True
1D59	C35E1D	=4941	JMP QLET10 ;
		=4942	QLET05: ; Else
1D5C	3E01	=4943	MVI A,1 ; ReturnValue := False;
		=4944	QLET10: ;
1D5E	B7	=4945	ORA A ; Return(ReturnValue);
		=4946	;
1D5F	C9	=4947	RET ; EndProc ?Letter;
		=4948 ;	
		=4949 ;	=====
		=4950 ;*	
		=4951 ;*	Function : QLETTER (? label letter)
		=4952 ;*	Inputs : A : Character to be checked.
		=4953 ;*	Outputs : Z = 1 : Character is a label letter.
		=4954 ;*	Z = 0 : The character is not in a label letter.
		=4955 ;*	Destroys : A and Flags.
		=4956 ;*	Calls : QLETTER and QNUM.
		=4957 ;*	
		=4958 ;*	Description : This routine checks that the character passed in the
		=4959 ;*	A register is a valid label character. Only uppercase characters
		=4960 ;*	are checked as all characters are converted to uppercase in the
		=4961 ;*	command line input routine.
		=4962 ;*	
		=4963 ;	=====
		=4964 ;	
		=4965	QLETTER: ; Procedure ?LabelLetter(Ch:Byte) : Boolean;
1D60	C5	=4966	PUSH B ;
1D61	47	=4967	MOV B,A ;
1D62	CD4D1D	=4968	CALL QLETTER ;
1D65	CA791D	=4969	JZ QLETE ; If (?Letter(Ch)) OR
		=4970	;
1D68	78	=4971	MOV A,B ;
1D69	CD7B1D	=4972	CALL QNUM ;
1D6C	FA731D	=4973	JM QLETC ; (?Number(Ch)) OR
		=4974	;
1D6F	AF	=4975	XRA A ;
1D70	C3791D	=4976	JMP QLETE ;
		=4977	QLETC: ;
1D73	78	=4978	MOV A,B ;
1D74	FE5F	=4979	CPI '_' ;
1D76	CA791D	=4980	JZ QLETE ; (Ch='_') then ?LabelLetter := TRUE
		=4981	;
		=4982	;
		=4983	Else
		=4984	QLETE: ;
		=4985	POP B ;
1D79	C1	=4986	RET ; EndProc ?LabelLetter;
1D7A	C9	=4987 ;	
		=4988 ;	=====
		=4989 ;*	
		=4990 ;*	Function : QNUM (? decimal digit)
		=4991 ;*	Inputs : A = Digit to be checked.
		=4992 ;*	Outputs : A = value of digit, if the character is a valid Hex
		=4993 ;*	digit (i.e. 0 to 9)
		=4994 ;*	If the character is outside these ranges then a

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 83

LOC	OBJ	LINE	SOURCE STATEMENT
		=4995 ;*	negative value is returned.
		=4996 ;*	Destroys : Acc and Flags.
		=4997 ;*	Calls : None.
		=4998 ;*	
		=4999 ;*	Description: This routine is used by the GETSYM routine to check
		=5000 ;*	that the characters is reads are valid digits.
		=5001 ;*	
		=5002 ;	=====
		=5003 ;	
		=5004 QNUM:	; Procedure Qnum;
		=5005 ;	
		=5006 ;	Check that the character is in the range '0' to '9'
		=5007 ;	
1D7B	FE30	=5008	CPI '0' ;
1D7D	FA8A1D	=5009	JM QNUM05 ; If Char >= '0' AND
		=5010	;
1D80	FE3A	=5011	CPI '9'+1 ;
1D82	F28A1D	=5012	JP QNUM05 ; Char < '9' then
		=5013	;
1D85	D630	=5014	SUI '0' ;
1D87	C38C1D	=5015	JMP QNUM10 ; Value := Char - Ord('0');
		=5016	;
		=5017 QNUM05:	; Else
1D8A	3EFF	=5018	MVI A,-1 ; Value := -1;
		=5019 QNUM10:	;
1D8C	B7	=5020	ORA A ;
1D8D	C9	=5021	RET ; Endproc QNum;
		=5022 ;	
		=5023 ;	=====
		=5024 ;*	
		=5025 ;*	Function : READ Number
		=5026 ;*	Inputs : A = The first digit of the number to be read.
		=5027 ;*	Outputs : CY = 1 : Error ecountered while reading number.
		=5028 ;*	CY = 0 : No error encountered.
		=5029 ;*	HL = Number (If no error was encountered.
		=5030 ;*	Destroys : A, B, C, D, E, H, L and Flags.
		=5031 ;*	Calls : DINIT, QNUM, DIN, MULT and DOUT.
		=5032 ;*	
		=5033 ;*	Description: This routine reads a number from the input command
		=5034 ;*	line. The value is returned in the HL register pair.
		=5035 ;*	
		=5036 ;*	NOTE: The routine makes use of a digit buffer.
		=5037 ;*	
		=5038 ;*	The number formats supported are as follows:
		=5039 ;*	
		=5040 ;*	i) Decimal (Default base) eg 1234
		=5041 ;*	ii) Decimal (Format two) eg 1234D
		=5042 ;*	iii) Octal eg 3344O
		=5043 ;*	iv) Octal (Format two) eg 3455Q
		=5044 ;*	v) Hexidecimal eg 0A7B3H
		=5045 ;*	vi) Binary eg 1010B
		=5046 ;*	vii) Binary (Format two) eg 0110Y
		=5047 ;*	
		=5048 ;	=====
		=5049 ;	
		=5050 READNU:	; Proc ReadNumber(FirstDigit:Byte);
1D8E	F5	=5051	PUSH PSW ;
1D8F	CDD81B	=5052	CALL DINIT ; DigitBufferInitialise;
1D92	F1	=5053	POP PSW ;
1D93	CDCB1B	=5054	CALL DIN ; DigitIn(FirstDigit);
1D96	CD1B1D	=5055	CALL NEXTCH ; Ch := NextCh;
		=5056 RNUM05:	;
1D99	47	=5057	MOV B,A ;
1D9A	CD2B1D	=5058	CALL QHNUM ;
1D9D	FAA91D	=5059	JM RNUM10 ; While ?hexNumber(Ch) do
		=5060	Begin
1DA0	CDCB1B	=5061	CALL DIN ; DigitIn(Digit);
1DA3	CD1B1D	=5062	CALL NEXTCH ; Ch := NextCh;
1DA6	C3991D	=5063	JMP RNUM05 ; End;
		=5064 ;	

LOC	OBJ	LINE	SOURCE STATEMENT
		=5065 ;	-----
		=5066 ;	
		=5067 ;	All the digits have been pushed into the digit LIFO
		=5068 ;	
		=5069 RNUM10:	; Case Ch of
1DA9	78	=5070	MOV A,B ;
1DAA	FE48	=5071	CPI 'H' ;
1DAC	C2B41D	=5072	JNZ RNUM15 ; 'H' : Base := 16;
		=5073	;
1DAF	3E10	=5074	MVI A,16 ;
1DB1	C3F91D	=5075	JMP BASEF ;
		=5076 RNUM15:	;
1DB4	FE4F	=5077	CPI 'O' ;
1DB6	C2BE1D	=5078	JNZ RNUM20 ; 'O' : Base := 8;
		=5079	;
1DB9	3E08	=5080	MVI A,8 ;
1DBB	C3F91D	=5081	JMP BASEF ;
		=5082 RNUM20:	;
1DBE	FE51	=5083	CPI 'Q' ;
1DC0	C2C81D	=5084	JNZ RNUM25 ; 'Q' : Base := 8;
		=5085	;
1DC3	3E08	=5086	MVI A,8 ;
1DC5	C3F91D	=5087	JMP BASEF ;
		=5088 RNUM25:	;
1DC8	FE59	=5089	CPI 'Y' ;
1DCA	C2D21D	=5090	JNZ RNUM30 ; 'Y' : Base := 2;
		=5091	;
1DCD	3E02	=5092	MVI A,2 ;
1DCF	C3F91D	=5093	JMP BASEF ;
		=5094 RNUM30:	; Else
		=5095	;
1DD2	CDB71C	=5096	CALL LASTCH ; Ch := LastCh;
		=5097	;
		=5098	; Case Ch of
1DD5	FE42	=5099	CPI 'B' ;
1DD7	C2E61D	=5100	JNZ RNUM35 ; 'B' : Begin
		=5101	;
1DDA	3E02	=5102	MVI A,2 ; Base := 2;
1DDC	3A11EA	=5103	LDA DIGCOUNT ;
1DDF	3D	=5104	DCR A ;
1DE0	3211EA	=5105	STA DIGCOUNT ; DigitCount :=
		=5106	; DigitCount - 1;
1DE3	C3F91D	=5107	JMP BASEF ; End;
		=5108 RNUM35:	;
1DE6	FE44	=5109	CPI 'D' ; 'D' : Begin
1DE8	C2F71D	=5110	JNZ RNUM40 ;
		=5111	;
1DEB	3E02	=5112	MVI A,2 ; Base := 10;
1DED	3A11EA	=5113	LDA DIGCOUNT ;
1DF0	3D	=5114	DCR A ;
1DF1	3211EA	=5115	STA DIGCOUNT ; DigitCount :=
		=5116	; DigitCount - 1;
1DF4	C3F91D	=5117	JMP BASEF ; End;
		=5118 RNUM40:	;
		=5119	; Else
1DF7	3E0A	=5120	MVI A,10 ; Base := 10;
		=5121	;
		=5122	; End Case;
		=5123	; End Case;
		=5124 ;	-----
		=5125 ;	
		=5126 ;	
		=5127 ;	The base of the number is in the A register and the memory variable
		=5128 ;	DIGCOUNT contains the number of digits in the number.
		=5129 ;	
		=5130 BASEF:	;
1DF9	4F	=5131	MOV C,A ;
1DFA	210000	=5132	LXI H,0000H ; Result := 0;
1DFD	CDEA1B	=5133	CALL DOUT ; Digit := DigitOut;
		=5134 NEXTDI:	; For I := 1 to Digitcount do

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 85

```

LOC   OBJ       LINE      SOURCE STATEMENT
=5135
1E00  B9        =5136      CMP      C          ;      Begin
1E01  D2191E    =5137      JNC      ERRDIG      ;
=5138      ;      If digit > base then error.
1E04  5F        =5139      MOV      E,A      ;
1E05  1600      =5140      MVI      D,0      ;
1E07  D5        =5141      PUSH     D          ;
1E08  1600      =5142      MVI      D,0      ;
1E0A  59        =5143      MOV      E,C      ;
1E0B  CDF1C     =5144      CALL     MULT      ;      Result := Result * base;
1E0E  D1        =5145      POP      D          ;
1E0F  19        =5146      DAD      D          ;      Result := Result + Digit;
1E10  CDEA1B    =5147      CALL     DOUT      ;      Digit := DigitOut;
1E13  F2001E    =5148      JP       NEXTDI     ;      End;
=5149
1E16  C31F1E    =5150      JMP      RDNUME      ;
=5151
=5152 ERRDIG:    =5152      ;      If Error then
1E19  21201E    =5153      LXI      H,ERDMSG      ;
1E1C  CDC119    =5154      CALL     ERROR      ;      ErrorMessage('Illegal digit');
=5155 RDNUME:    =5155      ;
1E1F  C9        =5156      RET          ; EndProc ReadNumber;
=5157 ;
=5158 ERDMSG:    =5158      ;
=5159 ;
=5160 ; format:
=5161 ;
=5162 ;      Illegal digit
=5163 ;
=5164 ; declaration:
=5165 ;
1E20  496C6C65  =5166      DB      'Illega', '1'+MSB, 'digit', EndMsg
      6761EC64
      69676974
      00
=5167 ;
=5168 ;=====
=5169 ;*
=5170 ;* Function : RESERVED (Check reserved word table).
=5171 ;* Inputs : String in the symbol buffer.
=5172 ;* Outputs : A = Symbol number.
=5173 ;*
=5174 ;* NOTE: If the symbol in the symbol buffer is not found in the
=5175 ;* reserved word table then the symbol type UNKNOWN is returned
=5176 ;*
=5177 ;* Destroys : H, L, A and Flags.
=5178 ;* Calls : None.
=5179 ;*
=5180 ;* Description : This routine checks to see if the string in the
=5181 ;* symbol buffer is one of the reserved words. If it is then the
=5182 ;* symbol type is returned in the A register. If it is not found then
=5183 ;* the special symbol type UNKNOWN is returned.
=5184 ;*
=5185 ;=====
=5186 ;
=5187 RESERVED:    ; Procedure Reserved;
1E2D  C5        =5188      PUSH     B          ;
1E2E  D5        =5189      PUSH     D          ;
1E2F  0E06      =5190      MVI      C,FIRSTSYM-1 ;      SymbolNumber := FirstSymbol - 1;
1E31  216E1E    =5191      LXI      H,RESTAB      ;      ResPtr := ReservedWordTable;
=5192      ;      Found := False;
=5193      ;
=5194 NRESW:      ;      Repeat
=5195      ;
1E34  0C        =5196      INR      C          ;      SymbolNumber := SymbolNumber + 1;
1E35  1103EA    =5197      LXI      D,SYMBUF      ;      SymPtr := SymbolBuffer;
1E38  46        =5198      MOV      B,M          ;      SigChars := Memory[ResPtr];
1E39  78        =5199      MOV      A,B          ;
1E3A  B7        =5200      ORA      A          ;
1E3B  CA651E    =5201      JZ       RESEND      ;      If SigChars <> 0 then begin

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 86

LOC	OBJ	LINE	SOURCE STATEMENT
		=5202	;
1E3E	23	=5203	INX H
		=5204	NRESWC:
1E3F	EB	=5205	XCHG
1E40	1A	=5206	LDAX D
1E41	BE	=5207	CMP M
1E42	EB	=5208	XCHG
		=5209	;
		=5210	;
		=5211	;
		=5212	;
1E43	C25C1E	=5213	JNZ SKIP
1E46	23	=5214	INX H
1E47	13	=5215	INX D
1E48	05	=5216	DCR B
1E49	C23F1E	=5217	JNZ NRESWC
		=5218	;
		=5219	; All the significant characters are equal. Now check the rest.
		=5220	;
		=5221	CHECK:
		=5222	;
1E4C	1A	=5223	LDAX D
1E4D	FE20	=5224	CPI SPACE
1E4F	CA6A1E	=5225	JZ FOUND
		=5226	;
1E52	1A	=5227	LDAX D
1E53	BE	=5228	CMP M
1E54	C25C1E	=5229	JNZ SKIP
		=5230	;
1E57	23	=5231	INX H
1E58	13	=5232	INX D
		=5233	;
1E59	C34C1E	=5234	JMP CHECK
		=5235	SKIP:
		=5236	;
		=5237	;
1E5C	7E	=5238	MOV A,M
1E5D	23	=5239	INX H
1E5E	B7	=5240	ORA A
1E5F	C25C1E	=5241	JNZ SKIP
		=5242	;
		=5243	;
1E62	C3341E	=5244	JMP NRESW
		=5245	;
		=5246	RESEND:
1E65	3E01	=5247	MVI A,UNKNOWN
1E67	C36B1E	=5248	JMP RESRET
		=5249	FOUND:
1E6A	79	=5250	MOV A,C
		=5251	RESRET:
1E6B	D1	=5252	POP D
1E6C	C1	=5253	POP B
		=5254	;
		=5255	;
1E6D	C9	=5256	RET
		=5257	;
		=5258	;
		=5259	;
		=5260	; The following table of reserved words has the following format:
		=5261	;
		=5262	; a: Significant characters : BYTE : This value indicates the number of
		=5263	characters that MUST match before the symbol is recognised.
		=5264	;
		=5265	; b: The character string.
		=5266	;
		=5267	; c: The last character of the string is a NUL
		=5268	;
		=5269	; The table is terminated by an entry with a significant character
		=5270	field of zero.
		=5271	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 87

LOC	OBJ	LINE	SOURCE STATEMENT
		=5272	RESTAB:
		=5273	;
0007		=5274	SYMA EQU FIRSTSYM ; SYMA entry
		=5275	;
1E6E	01	=5276	DB 1 ; Significant characters : 1
1E6F	41	=5277	STRA: DB 'A' ; String : 'A'
1E70	00	=5278	DB NUL ; Terminator
		=5279	;
0008		=5280	SYMBYTE EQU SYMA+1 ; SYMBYTE entry
1E71	02	=5281	DB 2 ; Significant character : 2
1E72	42595445	=5282	DB 'BYTE' ; String : 'BYTE'
1E76	00	=5283	DB NUL ; Terminator
		=5284	;
0009		=5285	SYMB EQU SYMBYTE+1 ; SYMB entry
1E77	01	=5286	DB 1 ; Significant characters : 1
1E78	42	=5287	STRB: DB 'B' ; String : 'B'
1E79	00	=5288	DB NUL ; Terminator
		=5289	;
000A		=5290	SYMCOUNT EQU SYMB+1 ; SYMCOUNT entry
1E7A	02	=5291	DB 2 ; Significant characters : 2
1E7B	434F554E	=5292	DB 'COUNT' ; String 'COUNT'
	54		
1E80	00	=5293	DB NUL ; Terminator
		=5294	;
000B		=5295	SYMC EQU SYMCOUNT+1 ; SYMC entry
1E81	01	=5296	DB 1 ; Significant characters : 1
1E82	43	=5297	STRC: DB 'C' ; String 'C'
1E83	00	=5298	DB NUL ; Terminator
		=5299	;
000C		=5300	SYMDEF EQU SYMC+1 ; SYMDEF entry
1E84	03	=5301	DB 3 ; Significant characters : 3
1E85	44454649	=5302	DB 'DEFINE' ; String 'DEFINE'
	4E45		
1E8B	00	=5303	DB NUL ; Terminator
		=5304	;
000D		=5305	SYMDIS EQU SYMDEF+1 ; SYMDIS entry
1E8C	02	=5306	DB 2 ; Significant characters : 2
1E8D	44495341	=5307	DB 'DISASM' ; String 'DISASM'
	534D		
1E93	00	=5308	DB NUL ; Terminator
		=5309	;
000E		=5310	SYMD EQU SYMDIS+1 ; SYMD entry
1E94	01	=5311	DB 1 ; Significant characters : 1
1E95	44	=5312	STRD: DB 'D' ; String 'D'
1E96	00	=5313	DB NUL ; Terminator
		=5314	;
000F		=5315	SYMEXEC EQU SYMD+1 ; SYMEXEC entry
1E97	03	=5316	DB 3 ; Significant characters : 3
1E98	454543	=5317	DB 'EXEC' ; String 'EXEC'
1E9C	00	=5318	DB NUL ; Terminator
		=5319	;
0010		=5320	SYMEXIT EQU SYMEXEC+1 ; SYMEXIT entry
1E9D		=5321	DB 2 ; Significant characters : 2
1E9E	5584954	=5322	DB 'EXIT' ; String 'EXIT'
1EA2	00	=5323	DB NUL ; Terminator
		=5324	;
0011		=5325	SYME EQU SYMEXIT+1 ; SYME entry
1EA	01	=5326	DB 1 ; Significant characters : 1
1E	45	=5327	STRE: DB 'E' ; String 'E'
1E	00	=5328	DB NUL ; Terminator
		=5329	;
1E		=5330	SYMFLAGS EQU SYME+1 ; SYMFLAGS entry
EA6	02	=5331	DB 2 ; Significant characters : 2
1EA7	464C4147	=5332	STRFLAGS: DB 'FLAGS' ; String 'FLAGS'
	53		
1EAC	00	=5333	DB NUL ; Terminator
		=5334	;
0013		=5335	SYMFOR EQU SYMFLAGS+1 ; SYMFOR entry
1EAD	02	=5336	DB 2 ; Significant characters : 2
1EAE	464F5245	=5337	DB 'FOREVER' ; String 'FOREVER'

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 89

LOC	OBJ	LINE	SOURCE STATEMENT
1EF9	4F4646	=5403	DB 'OFF' ; String 'OFF'
1EFC	00	=5404	DB NUL ; Terminator
		=5405	
0021		=5406	SYMON EQU SYMOFF+1 ; SYMON entry
1EFD	02	=5407	DB 2 ; Significant characters : 2
1EFE	4F4E	=5408	DB 'ON' ; String 'ON'
1F00	00	=5409	DB NUL ; Terminator
		=5410	
0022		=5411	SYMOUT EQU SYMON+1 ; SYMOUT entry
1F01	01	=5412	DB 1 ; Significant characters : 1
1F02	4F555450	=5413	DB 'OUTPUT' ; String 'OUTPUT'
	5554		
1F08	00	=5414	DB NUL ; Terminator
		=5415	
0023		=5416	SYMPC EQU SYMOUT+1 ; SYMPC entry
1F09	02	=5417	DB 2 ; Significant characters : 2
1F0A	5043	=5418	DB 'PC' ; String 'PC'
1F0C	00	=5419	DB NUL ; Terminator
		=5420	
0024		=5421	SYMPORT EQU SYMPC+1 ; SYMPORT entry
1F0D	02	=5422	DB 2 ; Significant characters : 2
1F0E	504F5254	=5423	DB 'PORT' ; String 'PORT'
1F12	00	=5424	DB NUL ; Terminator
		=5425	
0025		=5426	SYMPRINT EQU SYMPORT+1 ; SYMPRINT entry
1F13	01	=5427	DB 1 ; Significant characters : 1
1F14	5052494E	=5428	DB 'PRINT' ; String 'PRINT'
	54		
1F19	00	=5429	DB NUL ; Terminator
		=5430	
0026		=5431	SYMRDO EQU SYMPRINT+1 ; SYMRDO entry
1F1A	05	=5432	DB 5 ; Significant characters : 5
1F1B	52454144	=5433	DB 'READONLY' ; String 'READONLY'
	4F4E4C59		
1F23	00	=5434	DB NUL ; Terminator
		=5435	
0027		=5436	SYMRD EQU SYMRDO+1 ; SYMRD entry
1F24	01	=5437	DB 1 ; Significant characters : 1
1F25	52454144	=5438	DB 'READ' ; String 'READ'
1F29	00	=5439	DB NUL ; Terminator
		=5440	
0028		=5441	SYMREG EQU SYMRD+1 ; SYMREG entry
1F2A	03	=5442	DB 3 ; Significant characters : 3
1F2B	524547	=5443	DB 'REG' ; String 'REG'
1F2E	00	=5444	DB NUL ; Terminator
		=5445	
0029		=5446	SYMRESET EQU SYMREG+1 ; SYMRESET entry
1F2F	03	=5447	DB 3 ; Significant characters : 3
1F30	52455345	=5448	DB 'RESET' ; String 'RESET'
	54		
1F35	00	=5449	DB NUL ; Terminator
		=5450	
002A		=5451	SYMSP EQU SYMRESET+1 ; SYMSP entry
1F36	02	=5452	DB 2 ; Significant characters : 2
1F37	5350	=5453	DB 'SP' ; String 'SP'
1F39	00	=5454	DB NUL ; Terminator
		=5455	
002B		=5456	SYMSTEP EQU SYMSP+1 ; SYMSTEP entry
1F3A	02	=5457	DB 2 ; Significant characters : 2
1F3B	53544550	=5458	DB 'STEP' ; String 'STEP'
1F3F	00	=5459	DB NUL ; Terminator
		=5460	
002C		=5461	SYMSYM EQU SYMSTEP+1 ; SYMSYM entry
1F40	02	=5462	DB 2 ; Significant characters : 2
1F41	53594D42	=5463	DB 'SYMBOLIC' ; String 'SYMBOLIC'
	4F4C4943		
1F49	00	=5464	DB NUL ; Terminator
		=5465	
002D		=5466	SYMTILL EQU SYMSYM+1 ; SYMTILL entry
1F4A	02	=5467	DB 2 ; Significant characters : 2

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 91

LOC	OBJ	LINE	SOURCE STATEMENT
		=5537	STL05: ; For Ptr := 1 to LabelLength do
1F76	7E	=5538	MOV A,M ;
1F77	12	=5539	STAX D ; LabelBuffer[Ptr] :=
1F78	23	=5540	INX H ; SymbolBuffer[Ptr];
1F79	13	=5541	INX D ;
1F7A	05	=5542	DCR B ;
1F7B	C2761F	=5543	JNZ STL05 ;
		=5544	;
1F7E	C1	=5545	POP B ;
1F7F	D1	=5546	POP D ;
1F80	E1	=5547	POP H ;
1F81	C9	=5548	RET ; EndProc SymbolToLabel;
		=5549 ;	
		=5550 ;	END ;<DEBGETSYM.A85>
		5551	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 92

LOC	OBJ	LINE	SOURCE STATEMENT
		5552 ;	
		5553 ;	Get range routine:
		5554 ;	
		5555 \$INCLUDE(DEBRANGE.A85)	
		=5556 ;=====*	
		=5557 ;*	
		=5558 ;*	Function : GETRANGE (Get range from the command line)
		=5559 ;*	Inputs : (SYMVAL) this first number in the range.
		=5560 ;*	Outputs : Z = 1 : Range fetched without any errors.
		=5561 ;*	Z = 0 : If an error occurred.
		=5562 ;*	(RBEGIN) Range start value.
		=5563 ;*	(REND) Range end value.
		=5564 ;*	(SYMTYPE) the terminating symbol.
		=5565 ;*	Destroys : A, B, C, D, E, H, L and Flags.
		=5566 ;*	Calls : GETSYM.
		=5567 ;*	
		=5568 ;*	Description : This routine fetches a numerical range from the
		=5569 ;*	command line. The first symbol, which is always a number has
		=5570 ;*	already been fetched and its value placed in the memory variable
		=5571 ;*	SYMVAL. This is transferred to the range variable RBEGIN (Range
		=5572 ;*	begin).
		=5573 ;*	The following are valid range formats:
		=5574 ;*	
		=5575 ;*	Type 1 <Number> TO <Number> <NextSymbol>
		=5576 ;*	Type 2 <Number> LENGTH <Number> <NextSymbol>
		=5577 ;*	Type 3 <Number> <NextSymbol>
		=5578 ;*	
		=5579 ;*	If the symbol following the first <Number> is not 'TO' or 'LENGTH'
		=5580 ;*	then the range is assumed to be of type 3 and the start and end
		=5581 ;*	values are simply set equal to the first number.
		=5582 ;*	
		=5583 ;*	If the symbol following the first number is equal to 'TO' then
		=5584 ;*	another symbol is fetched from the command line. If this symbol is
		=5585 ;*	not a number then an error status is returned. If the symbol is a
		=5586 ;*	number then the range end value is set equal to this second number.
		=5587 ;*	
		=5588 ;*	If the symbol following the first number is equal to 'LENGTH' then
		=5589 ;*	another symbol is fetched from the command line. If this symbol is
		=5590 ;*	not a number then an error status is returned. If the symbol is a
		=5591 ;*	number then the range end value is set equal to the sum of this
		=5592 ;*	second number and range start value (given by the first number.)
		=5593 ;*	
		=5594 ;*	If either of these last two cases arises then another symbol is
		=5595 ;*	automatically fetched to remain consistent with type 3, where
		=5596 ;*	second symbol fetched is a terminating symbol.
		=5597 ;*	
		=5598 ;=====*	
		=5599 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 93

LOC	OBJ	LINE	SOURCE STATEMENT
		=5600	GETRANGE:
1F82	2A00EA	=5601	LHLD SYMVAL
1F85	227BEA	=5602	SHLD RBEGIN
1F88	CDFF1B	=5603	CALL GETSYM
1F8B	FE2E	=5604	CPI SYMTO
1F8D	C2AB1F	=5605	JNZ GETR20
		=5606	
1F90	CDFF1B	=5607	CALL GETSYM
1F93	FE02	=5608	CPI NUMBER
1F95	C2A61F	=5609	JNZ GETR05
		=5610	
1F98	2A00EA	=5611	LHLD SYMVAL
1F9B	227DEA	=5612	SHLD REND
1F9E	CDFF1B	=5613	CALL GETSYM
1FA1	3E00	=5614	MVI A,0
1FA3	C3A81F	=5615	JMP GETR10
		=5616	GETR05:
1FA6	3E01	=5617	MVI A,1
		=5618	GETR10:
1FA8	C3DE1F	=5619	JMP GETR99
		=5620	GETR20:
1FAB	FE1B	=5621	CPI SYMLEN
1FAD	C2D61F	=5622	JNZ GETR40
		=5623	
1FB0	CDFF1B	=5624	CALL GETSYM
1FB3	FE02	=5625	CPI NUMBER
1FB5	C2D11F	=5626	JNZ GETR25
		=5627	
1FB8	2A00EA	=5628	LHLD SYMVAL
1FBB	7C	=5629	MOV A,H
1FBC	B5	=5630	ORA L
1FBD	CAC11F	=5631	JZ GETR23
		=5632	
1FC0	2B	=5633	DCX H
		=5634	GETR23:
		=5635	
1FC1	EB	=5636	XCHG
1FC2	2A7BEA	=5637	LHLD RBEGIN
1FC5	19	=5638	DAD D
1FC6	227DEA	=5639	SHLD REND
		=5640	
1FC9	CDFF1B	=5641	CALL GETSYM
1FCC	3E00	=5642	MVI A,0
1FCE	C3D31F	=5643	JMP GETR30
		=5644	GETR25:
1FD1	3E01	=5645	MVI A,1
		=5646	GETR30:
1FD3	C3DE1F	=5647	JMP GETR99
		=5648	GETR40:
		=5649	
1FD6	2A7BEA	=5650	LHLD RBEGIN
1FD9	227DEA	=5651	SHLD REND
1FDC	3E00	=5652	MVI A,0
		=5653	
		=5654	GETR99:
1FDE	B7	=5655	ORA A
1FDF	C9	=5656	RET
		=5657	
		=5658	END <DEBRANGE.A85>
		5659	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 94

LOC	OBJ	LINE	SOURCE STATEMENT
		5660 ;	
		5661 ;	HELP command routine:
		5662 ;	
		5663 \$INCLUDE(DEBHELP.A85)	
		=5664 ;	-----;
		=5665 ;	;
		=5666 ;	HELP command routines used by the UCT SABus debugger. ;
		=5667 ;	;
		=5668 ;	This module contains (in order) the following subroutines. ;
		=5669 ;	;
		=5670 ;	a) HELPROU - HELP command routine. ;
		=5671 ;	b) QCOMMAND - ? command routine. ;
		=5672 ;	;
		=5673 ;	-----;
		=5674 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 95

LOC	OBJ	LINE	SOURCE STATEMENT
		=5675 ;	=====*
		=5676 ;*	*
		=5677 ;*	Function : HELPROU (Help routine). *
		=5678 ;*	Inputs : None. *
		=5679 ;*	Outputs : None. *
		=5680 ;*	Destroys : H, L, A and flags. *
		=5681 ;*	Calls : CMDERR, GETSYM, QCOMMAND, CRLF, GETINDIRECT and OUTMSG. *
		=5682 ;*	*
		=5683 ;*	Description : This routine handles the debugger HELP command. It *
		=5684 ;*	does this by printing the command syntax for each of the 11 *
		=5685 ;*	commands. A table listing the start address of each syntax *
		=5686 ;*	message is given at the end of the routine. *
		=5687 ;*	*
		=5688 ;*	Syntax: HELP [<Command>] <EOL> *
		=5689 ;*	*
		=5690 ;	=====*
		=5691 ;	
		=5692 HELPROU:	; Procedure HelpRoutine;
1FE0	CDFF1B	=5693 CALL GETSYM	; Sym := GetNextSymbol;
1FE3	FE00	=5694 CPI EOL	; If sym <> EndOfLine
1FE5	CA0820	=5695 JZ HLP05	;
		=5696	; Begin
1FE8	CD6020	=5697 CALL QCOMMAND	;
1FEB	CAFF1F	=5698 JZ HLP02	; If ?Command(Sym) then
		=5699	; Begin
1FEE	CDFF1B	=5700 CALL GETSYM	; Sym := GetNextSymbol;
1FF1	FE00	=5701 CPI EOL	;
1FF3	C2FF1F	=5702 JNZ HLP02	; If Sym = EOL then
		=5703	; Begin
1FF6	CDEE18	=5704 CALL OUTMSG	; OutMessage
		=5705	; (CommandSyntaxPtr);
1FF9	CDB419	=5706 CALL CRLF	; NewLine
		=5707	; End;
1FFC	C30520	=5708 JMP HLP04	; End
		=5709 HLP02:	; Else
1FFF	212920	=5710 LXI H,ERMHELP	;
2002	CD8119	=5711 CALL CMDERR	; CommandError
		=5712 HLP04:	; (HelpCommandErrorMessage);
2005	C32820	=5713 JMP HLP99	; End
		=5714 HLP05:	; Else
		=5715	; Begin
2008	213920	=5716 LXI H,CLRSCN	;
200B	CDEE18	=5717 CALL OUTMSG	; OutMessage(ClearScreen)
		=5718	;
200E	213E20	=5719 LXI H,HLP02TAB	; TabPtr := HelpTable;
		=5720 HLP10:	;
2011	7E	=5721 MOV A,M	;
2012	B7	=5722 ORA A	;
2013	CA2820	=5723 JZ HLP15	; While Memory[TabPtr] <> 0 do
		=5724	; Begin
2016	E5	=5725 PUSH H	;
2017	23	=5726 INX H	;
2018	CD9414	=5727 CALL GETINDIRECT	; MessageAdr := GetIndirect
		=5728	; (TabPtr+1);
201B	CDEE18	=5729 CALL OUTMSG	; OutMessage(MessageAdr)
201E	CDB419	=5730 CALL CRLF	; NewLine;
		=5731	;
2021	E1	=5732 POP H	;
2022	23	=5733 INX H	;
2023	23	=5734 INX H	;
2024	23	=5735 INX H	; TabPtr := TabPtr + 3;
2025	C31120	=5736 JMP HLP10	; End
		=5737 HLP15:	;
		=5738	;
		=5739 HLP99:	; End;
2028	C9	=5740 RET	; Endproc HelpRoutine;
		=5741 ;	
		=5742 ERMHELP:	
		=5743 ;	
		=5744 ; format:	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 96

LOC	OBJ	LINE	SOURCE STATEMENT
		=5745 ;	
		=5746 ;	HELP [<Command>]'
		=5747 ;	
		=5748 ;	declaration:
		=5749 ;	
2029	48454CD0 DB3C436F 6D6D616E 64BE5D00	=5750	DB 'HEL', 'P'+MSB, '['+MSB, '<Command', '>'+MSB, ']', EndMsg
		=5751 ;	
2039	1B481B4A 00	=5752	CLRSCN: DB ESC, HOMEC, ESC, CLRS, EndMsg
		=5753 ;	
		=5754 ;	-----
		=5755 ;	
		=5756 ;	Help command message lookup table. This table contains a three
		=5757 ;	byte entry for each possible command. The format of the entry
		=5758 ;	is as follows:
		=5759 ;	
		=5760 ;	Byte 1 : Symbol type (e.g. SYMGO)
		=5761 ;	Byte 2, Byte 3 : Address of the syntax message for the command.
		=5762 ;	
		=5763 ;	The table is terminated with an entry whose first byte is zero.
		=5764 ;	
		=5765	HLPTAB: ; Start address of table of pointers to
		=5766	; command syntax messages.
		=5767	;
203E	29	=5768	DB SYMRESET ; RESET symbol
203F	BD23	=5769	DW ERMRESET ; Address of the RESET command syntax message.
		=5770	;
2041	28	=5771	DB SYMREG ; REG symbol
2042	F424	=5772	DW ERMREG ; Address of the REG command syntax message.
		=5773	;
2044	0C	=5774	DB SYMDEF ; DEF symbol
2045	F829	=5775	DW ERMDEF ; Address of the DEF command syntax message.
		=5776	;
2047	1D	=5777	DB SYMMAP ; MAP symbol
2048	3021	=5778	DW ERM MAP ; Address of the MAP command syntax message.
		=5779	;
204A	17	=5780	DB SYMHELP ; HELP symbol
204B	2920	=5781	DW ERMHELP ; Address of the HELP command syntax message.
		=5782	;
204D	24	=5783	DB SYMPORT ; PORT symbol
204E	052C	=5784	DW ERMPORT ; Address of the PORT command syntax message.
		=5785	;
2050	2B	=5786	DB SYMSTEP ; STEP symbol
2051	D62E	=5787	DW ERMSTEP ; Address of the STEP command syntax message.
		=5788	;
2053	25	=5789	DB SYMPRINT ; PRINT symbol
2054	CA2C	=5790	DW ERMPRINT ; Address of the PRINT command syntax message.
		=5791	;
2056	08	=5792	DB SYMBYTE ; BYTE symbol
2057	7529	=5793	DW ERM BYTE ; Address of the BYTE command syntax message.
		=5794	;
2059	16	=5795	DB SYMGO ; GO symbol
205A	1D2B	=5796	DW ERMGO ; Address of the GO command syntax message.
		=5797	;
205C	0D	=5798	DB SYMDIS ; DIS symbol
205D	4F26	=5799	DW ERMDIS ; Address of the DIS command syntax message.
		=5800	;
205F	00	=5801	DB 0 ; Zero byte termination.
		=5802 ;	
		=5803 ;	=====*
		=5804 ;*	*
		=5805 ;*	Function : QCOMMAND (Command ?)
		=5806 ;*	Inputs : A : Symbol to be checked.
		=5807 ;*	Outputs : Z = 1 : Symbol is not a command.
		=5808 ;*	Z = 0 : Symbol is a command
		=5809 ;*	HL : Pointer to command syntax message.
		=5810 ;*	Destroys : H, L, A, B and flags.
			*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 97

LOC	OBJ	LINE	SOURCE STATEMENT
		=5811 ;*	Calls : GETINDIRECT.
		=5812 ;*	
		=5813 ;*	Description : This routine searches the help table to see is the
		=5814 ;*	symbol passed in the A register is a valid command and has a
		=5815 ;*	valid help response.
		=5816 ;*	
		=5817 ;	=====*
		=5818 ;	
		=5819	QCOMMAND: ; Procedure ?Command(Sym:Byte);
		=5820	;
2060	47	=5821	MOV B,A ;
2061	213E20	=5822	LXI H,HLPTAB ; Ptr := HelpTable;
		=5823	;
		=5824	QCMD05: ; While
2064	7E	=5825	MOV A,M ;
2065	B7	=5826	ORA A ;
2066	CA7A20	=5827	JZ QCMD99 ; (Memory[Ptr] <> 0) AND
		=5828	;
2069	B8	=5829	CMP B ;
206A	CA7320	=5830	JZ QCMD10 ; (Memory[Ptr] <> Symbol) do
		=5831	;
206D	23	=5832	INX H ;
206E	23	=5833	INX H ;
206F	23	=5834	INX H ; Ptr := Ptr + 3;
2070	C36420	=5835	JMP QCMD05 ;
		=5836	QCMD10: ; If Memory[Ptr] = Sym then
2073	23	=5837	INX H ;
2074	CD9414	=5838	CALL GETINDIRECT ; MessageAddress := GetIndirect(Ptr+1);
2077	3E01	=5839	MVI A,1 ;
2079	B7	=5840	ORA A ;
		=5841	QCMD99: ;
207A	C9	=5842	RET ; EndProc ?Command;
		=5843 ;	
		=5844 ;	END <DEBHELP.A85>
		5845	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 98

LOC	OBJ	LINE	SOURCE STATEMENT
		5846 ;	
		5847 ;	MAP command routine:
		5848 ;	
		5849 \$INCLUDE(DEBMAP.A85)	
		=5850 ;	-----;
		=5851 ;	;
		=5852 ;	MAP command routines for the UCT SABus debugger. ;
		=5853 ;	;
		=5854 ;	This module contains (in order) the following subroutines. ;
		=5855 ;	;
		=5856 ;	a) ADJUST - Adjust range. ;
		=5857 ;	b) MAPPRT - Print the map. ;
		=5858 ;	c) MAPROU - MAP command routine. ;
		=5859 ;	d) MAPUPDATE - Update the map. ;
		=5860 ;	e) PTYPE - Print access type. ;
		=5861 ;	;
		=5862 ;	-----;
		=5863 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 99

LOC	OBJ	LINE	SOURCE STATEMENT
		=5864 ;	=====*
		=5865 ;*	*
		=5866 ;*	Function : ADJUST (Adjust range values) *
		=5867 ;*	Inputs : (MAPADR) : Address of map to be updated. *
		=5868 ;*	(RBEGIN) : Range begin value. *
		=5869 ;*	(REND) : Range end value. *
		=5870 ;*	Outputs : New range values in memory. *
		=5871 ;*	Destroys : H, L, D, E, A and Flags. *
		=5872 ;*	Calls : DCMP. *
		=5873 ;*	*
		=5874 ;*	Description : This range adjusts the range values so that the same *
		=5875 ;*	update map routine can be used for both the IO map and the MEMORY *
		=5876 ;*	map. The problem arises because each entry in the IO map *
		=5877 ;*	corresponds to ONE IO location and each entry in the MEMORY map *
		=5878 ;*	corresponds to 256 bytes in the memory map. *
		=5879 ;*	*
		=5880 ;*	To allow the same update routine to be used the IO locations must *
		=5881 ;*	multiplied by 256 and then the update routine can assume that *
		=5882 ;*	each entry corresponds to 256 bytes in the MEMORY map or 256 *
		=5883 ;*	locations in the IO map. *
		=5884 ;*	*
		=5885 ;	=====*
		=5886 ;	
		=5887 ADJUST:	; Procedure Adjust;
207B	2A01ED	=5888 LHL D MAPADR	;
207E	1100EB	=5889 LXI D, IOMAP	;
2081	CDBF1A	=5890 CALL DCMP	;
2084	C29920	=5891 JNZ ADJ05	; If MapAddress = IOMap then
		=5892	; Begin
2087	2A7BEA	=5893 LHL RBEGIN	;
208A	65	=5894 MOV H, L	;
208B	2E00	=5895 MVI L, 0	;
208D	227BEA	=5896 SHLD RBEGIN	; RangeBegin := RangeBegin * 256;
		=5897	;
2090	2A7DEA	=5898 LHL REND	;
2093	65	=5899 MOV H, L	;
2094	2E00	=5900 MVI L, 0	;
2096	227DEA	=5901 SHLD REND	; RangeEnd := RangeEnd * 256
		=5902	;
		=5903 ADJ05:	; End;
2099	C9	=5904 RET	; EndProc Adjust;
		=5905 ;	
		=5906 ;	=====*
		=5907 ;*	*
		=5908 ;*	Function : MAPPRT (Map Print). *
		=5909 ;*	Inputs : (MAPADR) : Start address of the map to print. *
		=5910 ;*	Outputs : None. *
		=5911 ;*	Destroys : A, B, C, D, H, L and flags. *
		=5912 ;*	Calls : PHEXB, KITOUT, CRLF and PTYPE. *
		=5913 ;*	*
		=5914 ;*	Description : This routine prints either the MEMORY map or the IO *
		=5915 ;*	map. Both maps are printed in the same format. Each of the 256 *
		=5916 ;*	entries is printed in the following format: *
		=5917 ;*	*
		=5918 ;*	HH:T Where HH is a two hexadecimal digit number and *
		=5919 ;*	T is the protection type. *
		=5920 ;*	*
		=5921 ;*	See the PTYPE routine for information of the possible values of T. *
		=5922 ;*	*
		=5923 ;*	If the map is an IO map then the number HH represents the port *
		=5924 ;*	number. If the map is a MEMORY map then the number represents the *
		=5925 ;*	page number. *
		=5926 ;*	*
		=5927 ;	=====*
		=5928 ;	
		=5929 MAPPRT:	; Procedure MapPrint;
		=5930	;
209A	2A01ED	=5931 LHL D MAPADR	; MapPtr := MapAddress;
209D	1600	=5932 MVI D, 0	; EntryNumber := 0;
209F	0610	=5933 MVI B, 16	; For i := 1 to 16 do

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 100

LOC	OBJ	LINE	SOURCE STATEMENT
		=5934	MAPP05: ; Begin
20A1	0E10	=5935	MVI C,16 ; For j := 1 to 16 do
		=5936	MAPP10: ; Begin
20A3	7A	=5937	MOV A,D ;
20A4	CD041A	=5938	CALL PHEXB ; PrintHexByte(EntryNumber);
20A7	3E3A	=5939	MVI A,': ' ;
20A9	CDE218	=5940	CALL KITOUT ; KitOut(':');
20AC	7E	=5941	MOV A,M ;
20AD	CDA221	=5942	CALL PTYPE ; PrintType(Memory[MapPtr]);
20B0	3E20	=5943	MVI A, ' ' ;
20B2	CDE218	=5944	CALL KITOUT ; KitOut(' ');
		=5945	;
20B5	23	=5946	INX H ; MapPtr := MapPtr + 1;
		=5947	;
20B6	14	=5948	INR D ; EntryNumber := EntryNumber + 1;
20B7	0D	=5949	DCR C ;
20B8	C2A320	=5950	JNZ MAPP10 ; End
		=5951	;
20BB	CDB419	=5952	CALL CRLF ; NewLine
20BE	05	=5953	DCR B ;
20BF	C2A120	=5954	JNZ MAPP05 ; End;
		=5955	;
20C2	C9	=5956	RET ; EndProc MapPrint;
		=5957	;
		=5958	=====*
		=5959	;*
		=5960	;* Function : MAPROU (Map command routine). *
		=5961	;* Inputs : None. *
		=5962	;* Outputs : None. *
		=5963	;* Destroys : A, B, C, D, E, H, L and flags. *
		=5964	;* Calls : GETSYM, MAPPRT, GETRANGE, MAPUPDATE and CMDERR. *
		=5965	;*
		=5966	;* Description : This routine handles the MAP command. This allows *
		=5967	;* the contents of the maps to be displayed or modified. If the map *
		=5968	;* type is not specified (IO or MEMORY) then the MEMORY map is *
		=5969	;* assumed. The BNF syntax for the MAP command is given below: *
		=5970	;*
		=5971	;* Syntax: MAP [MEMORY:IO] [<Number> [TO LENGTH <Number>] = <ProType>] *
		=5972	;*
		=5973	;* <ProType> = USER READONLY GUARDED *
		=5974	;*
		=5975	;* Some sample commands are their meanings: *
		=5976	;*
		=5977	;* MAP - Display the MEMORY map. *
		=5978	;* MAP IO - Display the IO map. *
		=5979	;* MAP MEMORY 0 = USER - Set the protection of page 0 in the *
		=5980	;* MEMORY map to USER. *
		=5981	;* MAP 0H to 0FH = READONLY - Set the protection of pages 0 to 15 *
		=5982	;* in the MEMORY map to READONLY. *
		=5983	;* MAP 10 LENGTH 20 = GUARDED - Set the protection of pages 10 to 30 *
		=5984	;* in the MEMORY map to GUARDED. *
		=5985	;*
		=5986	=====*
		=5987	;
		=5988	MAPROU: ; Procedure MapRoutine;
20C3	2100EC	=5989	LXI H, MEMMAP ;
20C6	CDF1B	=5990	CALL GETSYM ; Sym := GetNextSymbol;
20C9	FE1E	=5991	CPI SYMMEM ;
20CB	CAD620	=5992	JZ MAPR10 ; If (Sym = MEMORY) or (Sym = IO) then
		=5993	; Begin
20CE	FE1A	=5994	CPI SYMIO ;
20D0	C2D920	=5995	JNZ MAPR15 ; If Sym = IO then
20D3	2100EB	=5996	LXI H, IOMAP ; MapAddress := IOMap
		=5997	MAPR10: ; Else
		=5998	; MapAddress := MemoryMap;
		=5999	;
20D6	CDF1B	=6000	CALL GETSYM ; Sym := GetNextSymbol;
		=6001	;
		=6002	; End
		=6003	MAPR15: ; Else

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 101

LOC	OBJ	LINE	SOURCE STATEMENT
		=6004	
20D9	2201ED	=6005	SHLD MAPADR
20DC	FE00	=6006	CPI EOL
20DE	C2E720	=6007	JNZ MAPR20
		=6008	
20E1	CD9A20	=6009	CALL MAPPR20
20E4	C32F21	=6010	JMP MAPR99
		=6011	MAPR20:
		=6012	
20E7	FE02	=6013	CPI NUMBER
20E9	C22921	=6014	JNZ MAPR80
		=6015	
20EC	CD821F	=6016	CALL GETRANGE
20EF	C22921	=6017	JNZ MAPR80
		=6018	
20F2	3A02EA	=6019	LDA SYMTYPE
20F5	FE04	=6020	CPI EQUALS
20F7	C22921	=6021	JNZ MAPR80
		=6022	
20FA	CDFF1B	=6023	CALL GETSYM
		=6024	
		=6025	
		=6026	
20FD	FE2F	=6027	CPI SYMUSER
20FF	C20721	=6028	JNZ MAPR25
		=6029	
2102	3E03	=6030	MVI A,USER
2104	C31821	=6031	JMP MAPR40
		=6032	MAPR25:
2107	FE15	=6033	CPI SYMGUARD
2109	C21121	=6034	JNZ MAPR30
		=6035	
210C	3E00	=6036	MVI A,GUARDED
210E	C31821	=6037	JMP MAPR40
		=6038	MAPR30:
2111	FE26	=6039	CPI SYMRDO
2113	C22921	=6040	JNZ MAPR80
		=6041	
2116	3E01	=6042	MVI A,READONLY
		=6043	
		=6044	MAPR40:
2118	3200ED	=6045	STA MAPVAL
211B	CDFF1B	=6046	CALL GETSYM
211E	FE00	=6047	CPI EOL
2120	C22921	=6048	JNZ MAPR80
		=6049	
2123	CD7A21	=6050	CALL MAPUPDATE
		=6051	
		=6052	
		=6053	
2126	C32F21	=6054	JMP MAPR99
		=6055	MAPR80:
2129	213021	=6056	LXI H,ERMMAP
212C	CD8119	=6057	CALL CMDERR
		=6058	MAPR99:
212F	C9	=6059	RET
		=6060	
		=6061	ERMMAP:
		=6062	
		=6063	format:
		=6064	
		=6065	MAP [MEMORY IO] [<Number> [TO LENGTH <Number>] = <ProType>]
		=6066	<ProType> = USER READONLY GUARDED
		=6067	
		=6068	declaration:
		=6069	
2130	4D41D0DB 4D454D4F 52597C49 CFDD	=6070	DB 'MA', 'P'+MSB, '['+MSB, 'MEMORY I', 'O'+MSB, ']' +MSB

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 102

LOC	OBJ	LINE	SOURCE STATEMENT
213E	DB80DB54 4F7C4C45 4E4754C8	=6071	DB '['+MSB, PSNUM, '['+MSB, 'TO LENGT', 'H'+MSB
214A	80DDBD3C 50726F54 797065BE 5D	=6072	DB PSNUM, ']' +MSB, '='+MSB, '<ProType', '>' +MSB, ']'
2157	0D0A09	=6073	DB CR, LF, TAB
215A	3C50726F 54797065 BEBD5553 45527C52 4541444F 4E4C597C 47554152 444544	=6074	DB '<ProType', '>' +MSB, '='+MSB, 'USER READONLY GUARDED'
2179	00	=6075	DB EndMsg
		=6076 ;	
		=6077 ;=====*	
		=6078 ;*	
		=6079 ;* Function : MAPUPDATE (Map update).	*
		=6080 ;* Inputs : (MAPVAL) : New protection value.	*
		=6081 ;* (MAPADR) : Start address of the map to update.	*
		=6082 ;* (RBEGIN) : Range begin.	*
		=6083 ;* (REND) : Range end.	*
		=6084 ;* Outputs : Updated map.	*
		=6085 ;* Destroys : A, B, C, D, E, H, L and flags.	*
		=6086 ;* Calls : ADJUST and DCMPL.	*
		=6087 ;*	*
		=6088 ;* Description : This routine updates the map pointed to by the memory	*
		=6089 ;* variable MAPADR. The range in the map given by the two range	*
		=6090 ;* variables RBEGIN and REND is updated to the new protection value	*
		=6091 ;* given by the memory variable MAPVAL.	*
		=6092 ;*	*
		=6093 ;=====*	
		=6094 ;	
		=6095 MAPUPDATE: ; Procedure MapUpdate;	
		=6096 ;	
217A	CD7B20	=6097	CALL ADJUST ; AdjustRangeValues;
217D	3A00ED	=6098	LDA MAPVAL ;
2180	47	=6099	MOV B,A ;
2181	3A02ED	=6100	LDA MAPADR+1 ;
2184	57	=6101	MOV D,A ;
2185	67	=6102	MOV H,A ;
2186	3A7EEA	=6103	LDA REND+1 ;
2189	5F	=6104	MOV E,A ;
218A	3A7CEA	=6105	LDA RBEGIN+1 ;
218D	6F	=6106	MOV L,A ; MapPtr := RangeStart;
218E	CDBF1A	=6107	CALL DCMPL ;
2191	DA9521	=6108	JC MAPU05 ;
2194	EB	=6109	XCHG ;
		=6110 MAPU05: ; While MapPtr < RangeEnd do	
2195	CDBF1A	=6111	CALL DCMPL ;
2198	CAA021	=6112	JZ MAPU10 ;
		=6113 ; Begin	
219B	70	=6114	MOV M,B ; Memory[MapPtr] := MapValue;
219C	23	=6115	INX H ; MapPtr := MapPtr + 1;
219D	C39521	=6116	JMP MAPU05 ; End;
		=6117 MAPU10: ;	
		=6118 ;	
21A0	70	=6119	MOV M,B ; Memory[MapPtr] := MapValue;
		=6120 ;	
21A1	C9	=6121	RET ; Endproc MapUpdate;
		=6122 ;	
		=6123 ;=====*	
		=6124 ;*	
		=6125 ;* Function : PTYPE (Print the protection type).	*
		=6126 ;* Inputs : A : Protection byte.	*
		=6127 ;* Outputs : None.	*
		=6128 ;* Destroys : A and flags.	*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 103

LOC	OBJ	LINE	SOURCE STATEMENT
		=6129 ;*	Calls : KITOUT.
		=6130 ;*	
		=6131 ;*	Description : This routine prints the letter corresponding to each
		=6132 ;*	of the three protection types. The protection type and its letter
		=6133 ;*	are given below:
		=6134 ;*	
		=6135 ;*	USER - 'U' - This Memory/IO can be read and written.
		=6136 ;*	READONLY - 'R' - This Memory/IO can only be read from.
		=6137 ;*	GUARDED - 'G' - No access to this Memory/IO is allowed.
		=6138 ;*	
		=6139 ;*	If the byte passed is not one of the above then a question mark is
		=6140 ;*	printed. If this occurs then it means that the map has been
		=6141 ;*	corrupted by the user program.
		=6142 ;*	
		=6143 ;	=====
		=6144 ;	
		=6145 PTYPE:	; Procedure PrintType(TypeByte:Byte);
		=6146	;
		=6147	; Case TypeByte of
21A2 FE03		=6148 CPI USER	;
21A4 C2AC21		=6149 JNZ PT05	; USER :
		=6150	;
21A7 3E55		=6151 MVI A,'U'	; Ch := 'U';
21A9 C3C221		=6152 JMP PT20	;
		=6153 PT05:	;
21AC FE01		=6154 CPI READONLY	;
21AE C2B621		=6155 JNZ PT10	; READONLY :
		=6156	;
21B1 3E52		=6157 MVI A,'R'	; Ch := 'R';
21B3 C3C221		=6158 JMP PT20	;
		=6159 PT10:	;
21B6 FE00		=6160 CPI GUARDED	;
21B8 C2C021		=6161 JNZ PT15	; GUARDED :
		=6162	;
21BB 3E47		=6163 MVI A,'G'	; Ch := 'G'
21BD C3C221		=6164 JMP PT20	;
		=6165 PT15:	; Else
21C0 3E3F		=6166 MVI A,'?'	; Ch := '?';
		=6167 PT20:	; End Case;
		=6168	;
21C2 CDE218		=6169 CALL KITOUT	; KitOut(Ch);
		=6170	;
21C5 C9		=6171 RET	; EndProc PrintType;
		=6172 ;	
		=6173 ;	END <DEBMAP.A85>
		6174 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 104

LOC	OBJ	LINE	SOURCE STATEMENT
		6175 ;	
		6176 ;	STEP command routine:
		6177 ;	
		6178	\$INCLUDE(DEBSSTEP.A85)
		=6179 ;	-----;
		=6180 ;	;
		=6181 ;	Single step routines use by the UCT SABus debugger. ;
		=6182 ;	;
		=6183 ;	This module contains (in order) the following subroutines. ;
		=6184 ;	;
		=6185 ;	a) ADDBC - Add byte count to the HL register pair. ;
		=6186 ;	b) CHKKEY - Check the keyboard. ;
		=6187 ;	c) COPYBYTES - Copy instruction into execution buffer. ;
		=6188 ;	d) EXEC - Execute the instruction. ;
		=6189 ;	e) INCPC - Increment program counter by the byte count. ;
		=6190 ;	f) SSINIT - Single step initialise. ;
		=6191 ;	g) SSTEP - Single step the next instruction. ;
		=6192 ;	;
		=6193 ;	-----;
		=6194	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		=6195 ;	=====*
		=6196 ;*	*
		=6197 ;*	Function : ADDBC (Add byte count) *
		=6198 ;*	Inputs : HL : Pointer to the current instruction. *
		=6199 ;*	Outputs : HL : Pointer to the next instruction. *
		=6200 ;*	Destroys : A, H, L and flags. *
		=6201 ;*	Calls : None. *
		=6202 ;*	*
		=6203 ;*	Description : This routine adds the byte count to the address of *
		=6204 ;*	the current instruction in order to obtain the address of the *
		=6205 ;*	next instruction. *
		=6206 ;*	*
		=6207 ;	=====*
		=6208 ;	
		=6209 ADDBC:	; Procedure AddByteCount : Word;
21C6	D5	=6210	PUSH D ;
21C7	3AA3EA	=6211	LDA BYTECOUNT ;
21CA	5F	=6212	MOV E,A ;
21CB	1600	=6213	MVI D,0 ;
21CD	19	=6214	DAD D ; Return(Reg.HL + ByteCount);
21CE	D1	=6215	POP D ;
21CF	C9	=6216	RET ; EndProc AddByteCount;
		=6217 ;	
		=6218 ;	=====*
		=6219 ;*	*
		=6220 ;*	Function : INPC (Increment Program Counter) *
		=6221 ;*	Inputs : None. *
		=6222 ;*	Outputs : None. *
		=6223 ;*	Destroys : A, H, L and flags. *
		=6224 ;*	Calls : ADDBC. *
		=6225 ;*	*
		=6226 ;*	Description : This routine updates the program counter by adding *
		=6227 ;*	the byte count to the program counter. *
		=6228 ;*	*
		=6229 ;	=====*
		=6230 ;	
		=6231 INPC:	; Procedure IncrementProgramCounter;
21D0	2A9DEA	=6232	LHLD REGPC ;
21D3	CDC621	=6233	CALL ADDBC ; RegPC := AddByteCount(RegPC);
21D6	229DEA	=6234	SHLD REGPC ;
21D9	C9	=6235	RET ; EndProc IncrementProgramCounter;
		=6236 ;	
		=6237 ;	=====*
		=6238 ;*	*
		=6239 ;*	Function : CHKKEY (Check keyboard) *
		=6240 ;*	Inputs : None. *
		=6241 ;*	Outputs : None. *
		=6242 ;*	Destroys : A, B, C and flags. *
		=6243 ;*	Calls : None. *
		=6244 ;*	*
		=6245 ;*	Description : This routine is used to maintain the dummy console. *
		=6246 ;*	It is called just before each instruction is to be executed or *
		=6247 ;*	emulated. The routine checks the console status. If the status *
		=6248 ;*	indicates that a character is ready to be fetched then the *
		=6249 ;*	character is fetched and checked against the termination character *
		=6250 ;*	TERMCH, If it is not equal to the termination character then the *
		=6251 ;*	value of the character is stored in the KEYVAL variable and the *
		=6252 ;*	dummy status byte is updated to indicate that a character is *
		=6253 ;*	ready to be fetched. If the character is equal to the termination *
		=6254 ;*	character then the status byte is changed to indicate that an *
		=6255 ;*	ESC (or any other termination character) has been typed. *
		=6256 ;*	*
		=6257 ;	=====*
		=6258 ;	
		=6259 CHKKEY:	; Procedure CheckKeyboard;
		=6260	;
21DA	DB01	=6261	IN COSTAT ;
21DC	4F	=6262	MOV C,A ; NewStatus := InputFromPort(CoStatusPort);
21DD	E602	=6263	ANI RXRDY ;
21DF	C2F221	=6264	JNZ CHKK10 ; If NOT ReceiverReady(Console) then

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 106

LOC	OBJ	LINE	SOURCE STATEMENT
		=6265	
21E2	3AB6EA	=6266	LDA CSTATUS ; Begin
21E5	E602	=6267	ANI RXRDY ; RxReady := CoStatus AND RxRDY;
21E7	47	=6268	MOV B,A ;
21E8	79	=6269	MOV A,C ;
21E9	E6FD	=6270	ANI NOT RXRDY ;
21EB	B0	=6271	ORA B ;
21EC	32B6EA	=6272	STA CSTATUS ; CoStatus := (NewStatus AND
		=6273	(NOT RxRDY)) OR
		=6274	RxReady;
21EF	C31022	=6275	JMP CHKK20 ; End
		=6276	CHKK10: ; Else
		=6277	Begin
21F2	79	=6278	MOV A,C ;
21F3	32B6EA	=6279	STA CSTATUS ; CoStatus := NewStatus;
		=6280	
21F6	DB00	=6281	IN CODATA ; CoData := InputFromPort(CoDataPort);
21F8	FE1B	=6282	CPI TERMCH ;
21FA	C20D22	=6283	JNZ CHKK15 ; If CoData = TermCh then
		=6284	Begin
21FD	3AB6EA	=6285	LDA CSTATUS ;
2200	E6FD	=6286	ANI NOT RXRDY ;
2202	32B6EA	=6287	STA CSTATUS ; CoStatus := CoStatus AND
		=6288	(NOT RxRdy);
2205	3E05	=6289	MVI A,ESCENTERED ;
2207	32B2EA	=6290	STA STATUS ; Status := ESCEntered;
220A	C31022	=6291	JMP CHKK17 ; End
		=6292	CHKK15: ; Else
220D	32B7EA	=6293	STA KEYVAL ; KeyValue := CoData;
		=6294	CHKK17: ;
		=6295	CHKK20: ; End;
		=6296	
2210	C9	=6297	RET ; EndProc CheckKeyboard;
		=6298	;
		=6299	=====*
		=6300	;
		=6301	Function : COPYBYTES (Copy bytes) *
		=6302	Inputs : A : Number of bytes to copy. *
		=6303	HL : Source address. *
		=6304	DE : Destination address. *
		=6305	Outputs : None. *
		=6306	Destroys : A, B, D, E, H, L and flags. *
		=6307	Calls : None. *
		=6308	;
		=6309	Description : This routine copies n bytes from the source address *
		=6310	to the destination address. The routine also writes two NOPs to *
		=6311	the second and third bytes of the destination range. *
		=6312	;
		=6313	=====*
		=6314	;
		=6315	COPYBYTES: ; Procedure CopyBytes(ByteCount:Byte;
		=6316	ProgramCounter:Word;
		=6317	Routine:Word);
2211	47	=6318	MOV B,A ;
2212	3E00	=6319	MVI A,(NOP) ;
2214	32D2EA	=6320	STA ROUTINE+1 ; Routine[1] := (NOP);
2217	32D3EA	=6321	STA ROUTINE+2 ; Routine[2] := (NOP);
		=6322	;
		=6323	COPB05: ; For I := 0 to ByteCount-1 do
		=6324	begin
221A	7E	=6325	MOV A,M ;
221B	12	=6326	STAX D ; Routine[i] :=
		=6327	Memory[ProgramCounter+1];
221C	13	=6328	INX D ;
221D	23	=6329	INX H ;
221E	05	=6330	DCR B ;
221F	C21A22	=6331	JNZ COPB05 ; End;
		=6332	;
2222	C9	=6333	RET ; EndProc CopyBytes;
		=6334	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 107

LOC	OBJ	LINE	SOURCE STATEMENT
		=6335	*****
		=6336	;
		=6337	Function : EXEC (Execute instruction)
		=6338	Inputs : None.
		=6339	Outputs : None.
		=6340	Destroys : A, B, C, D, E, H, L and flags.
		=6341	Calls : None.
		=6342	;
		=6343	Description : This routine executes the instruction in the
		=6344	execution buffer. The routine restores the user registers. Then
		=6345	jumps to the instruction in the execution buffer. The execution
		=6346	buffer ends with a jump to the RETURN label.
		=6347	;
		=6348	*****
		=6349	;
		=6350	EXEC: ; Procedure Execute;
2223	2A95EA	=6351	LHLD REGBC ;
2226	44	=6352	MOV B,H ;
2227	4D	=6353	MOV C,L ; RestoreBC;
		=6354	;
2228	2A97EA	=6355	LHLD REGDE ;
222B	54	=6356	MOV D,H ;
222C	5D	=6357	MOV E,L ; RestoreDE;
		=6358	;
222D	210000	=6359	LXI H,0 ;
2230	39	=6360	DAD SP ;
2231	22AEEA	=6361	SHLD MYSP ; MyStackPointer := SP;
		=6362	;
2234	2A9BEA	=6363	LHLD REGPSW ;
2237	E5	=6364	PUSH H ;
2238	F1	=6365	POP PSW ; RestorePSW;
		=6366	;
2239	2A9FEA	=6367	LHLD REGSP ;
223C	F9	=6368	SPHL ; RestoreSP;
		=6369	;
223D	2A99EA	=6370	LHLD REGM ; RestoreHL;
		=6371	;
2240	C3D1EA	=6372	JMP ROUTINE ; PerfromInstruction;
		=6373	RETURN: ;
		=6374	;
2243	2299EA	=6375	SHLD REGM ; SaveHL;
		=6376	;
2246	F5	=6377	PUSH PSW ;
2247	E1	=6378	POP H ;
2248	229BEA	=6379	SHLD REGPSW ; SavePSW;
		=6380	;
224B	210000	=6381	LXI H,0 ;
224E	39	=6382	DAD SP ;
224F	229FEA	=6383	SHLD REGSP ; SaveSP;
		=6384	;
2252	2AAEEA	=6385	LHLD MYSP ;
2255	F9	=6386	SPHL ; SP := MyStackPointer;
		=6387	;
2256	62	=6388	MOV H,D ;
2257	6B	=6389	MOV L,E ;
2258	2297EA	=6390	SHLD REGDE ; SaveDE;
		=6391	;
225B	60	=6392	MOV H,B ;
225C	69	=6393	MOV L,C ;
225D	2295EA	=6394	SHLD REGBC ; SaveBC;
		=6395	;
2260	C9	=6396	RET ; EndProc Execute;
		=6397	;
		=6398	*****
		=6399	;
		=6400	Function : SSINIT (Single step initialise)
		=6401	Inputs : None.
		=6402	Outputs : None.
		=6403	Destroys : A, H and L.
		=6404	Calls : None.

LOC	OBJ	LINE	SOURCE STATEMENT
		=6405 ;*	
		=6406 ;*	Description : This routine initialises the execution buffer and
		=6407 ;*	perpares for a single step instruction.
		=6408 ;*	
		=6409 ;*	The format of the execution buffer is given below:
		=6410 ;*	
		=6411 ;*	Byte 0 : Opcode of the instruction to be executed.
		=6412 ;*	Byte 1 : First byte of the operand.
		=6413 ;*	Byte 2 : Second byte of the operand.
		=6414 ;*	Byte 3 : (JMP) opcode
		=6415 ;*	Byte 4 : Least significant byte of the address of the
		=6416 ;*	RETURN label.
		=6417 ;*	Byte 5 : Most significant byte of the address of the
		=6418 ;*	RETURN label.
		=6419 ;*	
		=6420 ;*	Note: If the instruction has less then three bytes then bytes 1 and
		=6421 ;*	2 are set to the opcode for an NOP instruction.
		=6422 ;*	
		=6423 ;*	The dummy console status is set to zero (i.e. No character ready)
		=6424 ;*	
		=6425 ;	=====
		=6426 ;	
		=6427	SSINIT: ; Procedure SingleStepInitialise;
2261	3E00	=6428	MVI A,0 ;
2263	32B6EA	=6429	STA CSTATUS ; ConsoleStatus := 0;
2266	3EC3	=6430	MVI A,(JMP) ;
2268	32D4EA	=6431	STA ROUTINE+3 ; Routine[3] := (JMP);
226B	214322	=6432	LXI H,RETURN ;
226E	22D5EA	=6433	SHLD ROUTINE+4 ; Routine[4] := LOW RETURN;
		=6434	; Routine[5] := HIGH RETURN;
		=6435	;
2271	C9	=6436	RET ; EndProc SingleStepInitialise;
		=6437 ;	
		=6438 ;	=====
		=6439 ;*	
		=6440 ;*	Function : SSTEP (Single step)
		=6441 ;*	Inputs : None.
		=6442 ;*	Outputs : None.
		=6443 ;*	Destroys : A, B, C, D, E, H, L and flags.
		=6444 ;*	Calls : CHKKEY, TBWROP, CHKMRD, DCMPC, COPYBYTES, INCPC, TBWREX,
		=6445 ;*	EXEC, INDIRECT, GETINDIRECT and TBWRITE.
		=6446 ;*	
		=6447 ;*	Description : This routine Executes/Emulates the next instruction.
		=6448 ;*	The routine performs the following functions:
		=6449 ;*	
		=6450 ;*	a) Check the keyboard and update the dummy console status byte.
		=6451 ;*	
		=6452 ;*	b) Using the tables calculate the following information:
		=6453 ;*	i) BYTECOUNT. (Number of bytes in the instruction)
		=6454 ;*	ii) EMINFO. (Emulation information)
		=6455 ;*	
		=6456 ;*	c) Check that read operations are allowed from the locations
		=6457 ;*	containing the instruction byte(s).
		=6458 ;*	
		=6459 ;*	d) Check if the trigger condition has been met.
		=6460 ;*	
		=6461 ;*	e) If the emulation byte indicates that the instruction can be
		=6462 ;*	executed directly (i.e. EMINFO = EXECUTED) then the
		=6463 ;*	instruction is copied into the execution buffer and the EXEC
		=6464 ;*	routine called.
		=6465 ;*	
		=6466 ;*	f) If the Emulation information byte indicates that the
		=6467 ;*	instruction cannot be executed then the correct emulation
		=6468 ;*	routine is called.
		=6469 ;*	
		=6470 ;*	g) The final function of the routine is to write the contents of
		=6471 ;*	trace buffer entry to the trace buffer.
		=6472 ;*	
		=6473 ;	=====
		=6474 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 109

LOC	OBJ	LINE	SOURCE STATEMENT
		=6475	SSTEP: ; Procedure SingleStep;
		=6476	;
2272	CDDA21	=6477	CALL CHKKEY ; CheckKeyBoard;
		=6478	;
2275	20	=6479	RIM ;
2276	E670	=6480	ANI 01110000B ;
2278	47	=6481	MOV B,A ;
2279	3AA1EA	=6482	LDA IMASK ;
227C	E68F	=6483	ANI 10001111B ; {Update IMASK}
227E	B0	=6484	ORA B ;
227F	47	=6485	MOV B,A ;
2280	32A1EA	=6486	STA IMASK ; Imask := (RIM and Pending_bits) or
		=6487	;(Imask and not pending_bits);
		=6488 ;	
		=6489 ;	Check to see if there are any interrupts pending:
		=6490 ;	
		=6491 ;	By reading the interrupt mask of the 8085 (using the RIM instruction)
		=6492 ;	and checking this against the interrupt mask bits, one can determine
		=6493 ;	which interrupts would, during the normal operation of the 8085,
		=6494 ;	be serviced.
		=6495 ;	
		=6496 ;	Position of the interrupt mask and pending bits in the interrupt register:
		=6497 ;	
		=6498 ;	D7 D6 D5 D4 D3 D2 D1 D0
		=6499 ;	-----
		=6500 ;	x P75 P65 P55 ENA M75 M65 M55
		=6501 ;	-----
		=6502 ;	
		=6503 ;	Key: P75, P65 and P55 are the interrupt pending bits ('1'=Pending)
		=6504 ;	M75, M65 and M55 are the interrupt mask bits ('0'=enabled)
		=6505 ;	ENA = Interrupt enable/disable flag ('1'=enabled)
		=6506 ;	
		=6507 ;	
2283	E608	=6508	ANI 00001000B ;
2285	CAD422	=6509	JZ NOINTR ; If InterruptMask AND InterruptEnable <> 0
		=6510	; then begin
2288	78	=6511	MOV A,B ;
2289	E670	=6512	ANI 01110000B ;
228B	0F	=6513	RRC ;
228C	0F	=6514	RRC ;
228D	0F	=6515	RRC ;
228E	0F	=6516	RRC ;
228F	4F	=6517	MOV C,A ; Reg.C := (RIM() and 01110000B) shr 4;
2290	78	=6518	MOV A,B ;
2291	2F	=6519	CMA ;
2292	E607	=6520	ANI 00000111B ;
2294	A1	=6521	ANA C ; Reg.A := NOT (InterruptMask and 111B)
		=6522	; AND Reg.C
		=6523 ;	
		=6524 ;	I have now calculated if interrupts are enabled (ENA='1'),
		=6525 ;	which interrupts are enabled and which interrupts are not masked.
		=6526 ;	
		=6527 ;	i.e. the accumulator now contains the following information:
		=6528 ;	
		=6529 ;	D7 D6 D5 D4 D3 D2 D1 D0
		=6530 ;	-----
		=6531 ;	0 0 0 0 0 A75 A65 A55
		=6532 ;	-----
		=6533 ;	
		=6534 ;	Where A75 (active RST 7.5) = ENA.M75.P75 (i.e. interrupt must ber serviced)
		=6535 ;	
		=6536 ;	Where A65 (active RST 7.5) = ENA.M65.P65 (i.e. interrupt must ber serviced)
		=6537 ;	
		=6538 ;	Where A55 (active RST 7.5) = ENA.M55.P55 (i.e. interrupt must ber serviced)
		=6539 ;	
		=6540 ;	
		=6541 ;	If all three bits are zero then there are no interrupts that can occur.
		=6542 ;	
		=6543	;
2295	CAD422	=6544	JZ NOINTR ; If Reg.A = 0 <> 0 then

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 110

LOC	OBJ	LINE	SOURCE STATEMENT
		=6545	;
		=6546	;
2298	47	=6547	MOV B,A
		=6548	;
2299	E604	=6549	ANI 00000100B
229B	CAA722	=6550	JZ NOTR75
		=6551	;
229E	3E10	=6552	MVI A,00010000B
22A0	30	=6553	SIM
22A1	213C00	=6554	LXI H,RST75V
22A4	C3B622	=6555	JMP DOINTR
		=6556	NOTR75:
22A7	78	=6557	MOV A,B
22A8	E602	=6558	ANI 00000010B
22AA	CAB322	=6559	JZ NOTR65
		=6560	;
22AD	213400	=6561	LXI H,RST65V
22B0	C3B622	=6562	JMP DOINTR
		=6563	NOTR65:
		=6564	;
		=6565	;
22B3	212C00	=6566	LXI H,RST55V
		=6567	;
		=6568	;
		=6569	;
		=6570	; I have now identified which interrupt will occur. (its vector is loaded into
		=6571	; the HL register pair.
		=6572	;
		=6573	; NOTE: the order in which the bits are checked determines the relative
		=6574	priorities of the three interrupts.
		=6575	;
		=6576	DOINTR:
22B6	E5	=6577	PUSH H
22B7	CD4B2F	=6578	CALL TBINTR
22BA	2A9DEA	=6579	LHLD REGPC
22BD	CD6B14	=6580	CALL DOPUSH
		=6581	;
22C0	3E0A	=6582	MVI A,EXINTR
22C2	32B9EA	=6583	STA TBENTRY+1
		=6584	;
		=6585	;
22C5	E1	=6586	POP H
22C6	229DEA	=6587	SHLD REGPC
		=6588	;
		=6589	;
22C9	3AA1EA	=6590	LDA IMASK
22CC	E6F7	=6591	ANI 11110111B
22CE	32A1EA	=6592	STA IMASK
22D1	C37D23	=6593	JMP STEP99
		=6594	;
		=6595	NOINTR:
22D4	3AA2EA	=6596	LDA ENABINT
22D7	FEFF	=6597	CPI TRUE
22D9	C2E922	=6598	JNZ STEP01
		=6599	;
		=6600	;
22DC	3AA1EA	=6600	LDA IMASK
22DF	F608	=6601	ORI 00001000B
22E1	32A1EA	=6602	STA IMASK
22E4	3E00	=6603	MVI A,FALSE
22E6	32A2EA	=6604	STA ENABINT
		=6605	;
		=6606	;
		=6607	STEP01:
22E9	2A9DEA	=6608	LHLD REGPC
22EC	7E	=6609	MOV A,M
22ED	32A8EA	=6610	STA OPCODE
		=6611	;
22F0	6F	=6612	MOV L,A
22F1	2600	=6613	MVI H,0
22F3	29	=6614	DAD H

Begin
Case Reg.A of
100B : { RST 7.5 } Begin
Reset_75_Floplop;
DoIntr(RST75_vector);
End;
010B : { RST 6.5 } Begin
DoIntr(RST65_vector);
Else
DoIntr(RST55_vector);
End; {Case}
=6570 ; I have now identified which interrupt will occur. (its vector is loaded into
=6571 ; the HL register pair.
=6572 ;
=6573 ; NOTE: the order in which the bits are checked determines the relative
=6574 ; priorities of the three interrupts.
=6575 ;
=6576 DOINTR:
TraceBufferInterrupt;
DoPush(ProgramCounter);
TraceBuffer.Type
:= ExecuteInterrupt;
ProgramCounter
:= InterruptVector;
Disable_Interrupts;
End;
If Enable_Interrupt_flag then
Begin
Imask := Imask or IntEnable;
Enable_interrupt_flag := FALSE;
End
Opcode := memory[RegPC];
Ptr:= Opcode * 2;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 111

LOC	OBJ	LINE	SOURCE STATEMENT
		=6615	
22F4	111812	=6616	LXI D,DISTAB
22F7	19	=6617	DAD D
		=6618	
22F8	5E	=6619	MOV E,M
22F9	23	=6620	INX H
22FA	56	=6621	MOV D,M
		=6622	
22FB	7A	=6623	MOV A,D
22FC	0F	=6624	RRC
22FD	0F	=6625	RRC
22FE	0F	=6626	RRC
22FF	E61F	=6627	ANI EMMASK
2301	32A5EA	=6628	STA EMINFO
		=6629	
2304	EB	=6630	XCHG
2305	29	=6631	DAD H
2306	7C	=6632	MOV A,H
2307	E60F	=6633	ANI OPRMASK
		=6634	
2309	6F	=6635	MOV L,A
230A	2600	=6636	MVI H,0
230C	29	=6637	DAD H
230D	11D410	=6638	LXI D,OPRTAB
2310	19	=6639	DAD D
2311	7E	=6640	MOV A,M
2312	32A3EA	=6641	STA BYTECOUNT
		=6642	
2315	2A9DEA	=6643	LHLD REGPC
2318	CDCA2F	=6644	CALL TBWROP
		=6645	
231B	3AA3EA	=6646	LDA BYTECOUNT
231E	47	=6647	MOV B,A
231F	2AB4EA	=6648	LHLD TRIGWORD
2322	EB	=6649	XCHG
2323	2A9DEA	=6650	LHLD REGPC
		=6651	STEP02:
		=6652	
2326	CD3E10	=6653	CALL CHKMRD
2329	CA7823	=6654	JZ STEP80
		=6655	
232C	3AB3EA	=6656	LDA TRIG
232F	FE01	=6657	CPI TRIGEX
2331	C23F23	=6658	JNZ STEP03
		=6659	
2334	CDBF1A	=6660	CALL DCOMP
2337	C23F23	=6661	JNZ STEP03
		=6662	
233A	3E02	=6663	MVI A,TRIGCONDITION
233C	32B2EA	=6664	STA STATUS
		=6665	STEP03:
		=6666	
		=6667	
233F	23	=6668	INX H
2340	05	=6669	DCR B
2341	C22623	=6670	JNZ STEP02
		=6671	
2344	3AA5EA	=6672	LDA EMINFO
2347	FE00	=6673	CPI EXECUTE
2349	C26623	=6674	JNZ STEP05
		=6675	
234C	2A9DEA	=6676	LHLD REGPC
234F	11D1EA	=6677	LXI D,ROUTINE
2352	3AA3EA	=6678	LDA BYTECOUNT
2355	CD1122	=6679	CALL COPYBYTES
		=6680	
		=6681	
2358	CD2322	=6682	CALL EXEC
235B	CDD021	=6683	CALL INCP
235E	3E01	=6684	MVI A,EXNOREF

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 112

LOC	OBJ	LINE	SOURCE STATEMENT
2360	CD662F	=6685	CALL TBWREX ; TraceBufferWriteExecute(NoRef);
		=6686	;
2363	C37D23	=6687	JMP STEP99 ; End
		=6688 STEP05:	;
		=6689	;
2366	11DE11	=6690	LXI D,EMUTAB ; Begin
2369	3D	=6691	DCR A ;
236A	6F	=6692	MOV L,A ;
236B	2600	=6693	MVI H,0 ;
236D	29	=6694	DAD H ;
236E	19	=6695	DAD D ;
		=6696	;
236F	CD9414	=6697	CALL GETINDIRECT ; EmulateRoutine := GetIndirect
		=6698	;
		=6699	;
2372	CDCD1A	=6700	CALL INDIRECT ; CallIndirect(EmulateRoutine);
		=6701	;
2375	C37D23	=6702	JMP STEP99 ; End;
		=6703 STEP80:	;
2378	3E04	=6704	MVI A,GUARDACCESS ;
237A	32B2EA	=6705	STA STATUS ;
		=6706 STEP99:	;
237D	CD712F	=6707	CALL TBWRITE ; TraceBufferWrite;
		=6708	;
2380	C9	=6709	RET ; EndProc SingleStep;
		=6710 ;	;
		=6711 ;	END <DEBSSTEP.A85>
		6712 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 113

LOC	OBJ	LINE	SOURCE STATEMENT
		6713 ;	
		6714 ;	RESET command routine:
		6715 ;	
		6716	\$INCLUDE(DEBRESET.A85)
		=6717 ;	-----;
		=6718 ;	;
		=6719 ;	Reset command routines used by the UCT SABus debugger ;
		=6720 ;	;
		=6721 ;	This module contains (in order) the following subroutines: ;
		=6722 ;	;
		=6723 ;	a) RESETROU - RESET command routine. ;
		=6724 ;	b) RESMAP - Reset map routine. ;
		=6725 ;	;
		=6726 ;	-----;
		=6727	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 114

LOC	OBJ	LINE	SOURCE STATEMENT
		=6728 ;	=====*
		=6729 ;*	*
		=6730 ;*	Function : RESETROU (Reset routine)
		=6731 ;*	Inputs : None.
		=6732 ;*	Outputs : None.
		=6733 ;*	Destroys : A, B, C, D, E, H, L and flags.
		=6734 ;*	Calls : GETSYM, RESMAP and CMDERR.
		=6735 ;*	*
		=6736 ;*	Description : This routine handles the RESET command. If the first
		=6737 ;*	symbol fetched from the command line is the RESET symbol then this
		=6738 ;*	routine is called. The syntax of the RESET command is given below:
		=6739 ;*	*
		=6740 ;*	BNF Syntax: RESET [MEMORY IO]
		=6741 ;*	*
		=6742 ;*	Possible commands and their meanings:
		=6743 ;*	*
		=6744 ;*	RESET - Reset both maps (MEMORY and IO) to guarded access.
		=6745 ;*	RESET MEMORY - Reset the MEMORY map to guarded access.
		=6746 ;*	RESET IO - Reset the IO map to guarded access.
		=6747 ;*	*
		=6748 ;	=====*
		=6749 ;	
		=6750	RESETROU: ; Procedure ResetMap;
2381	CDFF1B	=6751	CALL GETSYM ; Sym := GetNextSymbol;
2384	FE00	=6752	CPI EOL ;
2386	CAA723	=6753	JZ RM20 ; If Symbol <> EOL then
		=6754	; Begin
2389	2100EC	=6755	LXI H, MEMMAP ;
238C	FE1E	=6756	CPI SYMMEM ;
238E	CA9923	=6757	JZ RM05 ; If Sym = MEMORY or
		=6758	;
2391	2100EB	=6759	LXI H, IOMAP ;
2394	FE1A	=6760	CPI SYMIO ; Sym = IO
2396	C2B623	=6761	JNZ RM50 ;
		=6762	RM05: ; Begin
2399	CDFF1B	=6763	CALL GETSYM ; Sym := GetNextSymbol;
239C	FE00	=6764	CPI EOL ; If Sym = EOL then
239E	C2B623	=6765	JNZ RM50 ;
		=6766	;
23A1	CDCC23	=6767	CALL RESMAP ; ResetMap;
23A4	C3BC23	=6768	JMP RM99 ; End
		=6769	RM20: ; Else
		=6770	; Begin
23A7	2100EC	=6771	LXI H, MEMMAP ;
23AA	CDCC23	=6772	CALL RESMAP ; ResetMemoryMap;
23AD	2100EB	=6773	LXI H, IOMAP ;
23B0	CDCC23	=6774	CALL RESMAP ; ResetIOMap;
23B3	C3BC23	=6775	JMP RM99 ; End;
		=6776	RM50: ;
23B6	21BD23	=6777	LXI H, ERMRESET ;
23B9	CD8119	=6778	CALL CMDERR ;
		=6779	RM99: ;
23BC	C9	=6780	RET ; Endproc ResetMap;
		=6781 ;	
		=6782	ERMRESET:
		=6783 ;	
		=6784 ;	format:
		=6785 ;	
		=6786 ;	DB RESET IO MEMORY
		=6787 ;	
		=6788 ;	declaration:
		=6789 ;	
23BD	52455345 D4494F7C 4D454D4F 525900	=6790	DB 'RESE', 'T'+MSB, 'IO MEMORY', EndMsg
		=6791 ;	
		=6792 ;	=====*
		=6793 ;*	*
		=6794 ;*	Function : RESMAP (Reset Map).

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 115

LOC	OBJ	LINE	SOURCE STATEMENT
		=6795 ;*	Inputs : HL = Start address of the map to be cleared. *
		=6796 ;*	Outputs : None. *
		=6797 ;*	Destroys : B, H, L and flags. *
		=6798 ;*	Calls : None. *
		=6799 ;*	
		=6800 ;*	Description : This routine clears the map pointed to by the HL *
		=6801 ;*	register pair. All the map entries are reset to the guarded access *
		=6802 ;*	state. *
		=6803 ;*	
		=6804 ;*	=====*
		=6805 ;	
		=6806 RESMAP:	; Procedure ResetMap(MapAddress:Integer);
23CC	0600	=6807 MVI B,0	;
		=6808 RESM05:	; For I := 0 to MapLength - 1 do
23CE	3600	=6809 MVI M,GUARDED	;
23D0	23	=6810 INX H	; Memory[MapAddress+I] := Guarded;
23D1	05	=6811 DCR B	;
23D2	C2CE23	=6812 JNZ RESM05	;
		=6813	;
23D5	C9	=6814 RET	; EndProc ResetMap;
		=6815 ;	
		=6816 ;	END <DEBRESET.A85>
		6817 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 116

LOC	OBJ	LINE	SOURCE STATEMENT
		6818 ;	
		6819 ;	REG command routine:
		6820 ;	
		6821 \$INCLUDE(DEBREG.A85)	
		=6822 ;	-----;
		=6823 ;	;
		=6824 ;	;
		=6825 ;	Register command routines of the UCT SABus debugger.
		=6826 ;	;
		=6827 ;	;
		=6828 ;	This module contains (in order) the following subroutines.
		=6829 ;	;
		=6830 ;	a) COPYENTRY - Copy information from one of the entries in the
		=6831 ;	register table into some memory variables.
		=6832 ;	b) REGDIS - Display the register whose information is given
		=6833 ;	byte the memory variables REGSTR, REGADR and
		=6834 ;	REGTYPE.
		=6835 ;	c) REGDMP - Dump the contents of ALL the registers.
		=6836 ;	d) REGLOOKUP - Lookup a symbol in the register table to
		=6837 ;	determine if a symbol is a register name or not.
		=6838 ;	e) REGROU - REG command routine.
		=6839 ;	f) REGUPDATE - Update a register.
		=6840 ;	;
		=6841 ;	-----;
		=6842 \$EJECT	

LOC	OBJ	LINE	SOURCE STATEMENT
		=6843 ;	=====*
		=6844 ;*	*
		=6845 ;*	Function : COPYENTRY (Copy register table entry). *
		=6846 ;*	Inputs : HL : Address of a register table entry. *
		=6847 ;*	Outputs : (REGADR) : Address of register storage. *
		=6848 ;*	(REGSTR) : Address of register string. *
		=6849 ;*	(REGTYPE) : Register type (BYTE or WORD). *
		=6850 ;*	Destroys : A and flags. *
		=6851 ;*	Calls : None. *
		=6852 ;*	*
		=6853 ;*	Description : This routine copies the information contained in a *
		=6854 ;*	single register table entry into the three memory variables: *
		=6855 ;*	*
		=6856 ;*	REGSTR : Start address of the register string. *
		=6857 ;*	REGADR : Start address of the register storage. *
		=6858 ;*	REGTYPE : Register type (BYTE - 8 bits or WORD - 16 bits). *
		=6859 ;*	*
		=6860 ;	=====*
		=6861 ;	
		=6862	COPYENTRY: ; Procedure CopyEntry(Ptr:Integer);
23D6	E5	=6863	PUSH H ;
23D7	D5	=6864	PUSH D ;
23D8	7E	=6865	MOV A,M ;
23D9	E680	=6866	ANI VARTYPE ;
23DB	3290EA	=6867	STA REGTYPE ; RegType := Memory[Ptr] AND VarType;
		=6868	;
23DE	23	=6869	INX H ;
23DF	5E	=6870	MOV E,M ;
23E0	23	=6871	INX H ;
23E1	56	=6872	MOV D,M ;
23E2	23	=6873	INX H ;
23E3	EB	=6874	XCHG ;
23E4	2291EA	=6875	SHLD REGADR ; RegAddress := Memory[Ptr+1] +
		=6876	; Memory[Ptr+2] * 256;
23E7	EB	=6877	XCHG ;
23E8	5E	=6878	MOV E,M ;
23E9	23	=6879	INX H ;
23EA	56	=6880	MOV D,M ;
23EB	23	=6881	INX H ;
23EC	EB	=6882	XCHG ;
23ED	2293EA	=6883	SHLD REGSTR ; RegString := Memory[Ptr+3] +
		=6884	; Memory[Ptr+4] * 256;
23F0	D1	=6885	POP D ;
23F1	E1	=6886	POP H ;
23F2	C9	=6887	RET ; EndProc CopyEntry;
		=6888 ;	
		=6889 ;	=====*
		=6890 ;*	*
		=6891 ;*	Function : REGDIS (Register Display). *
		=6892 ;*	Inputs : (REGSTR) : Address of register string. *
		=6893 ;*	(REGADR) : Address of register storage. *
		=6894 ;*	(REGTYPE) : Register type (BYTE or WORD). *
		=6895 ;*	Outputs : None. *
		=6896 ;*	Destroys : A and flags. *
		=6897 ;*	Calls : OUTMSG, KITOUT, PHEXB and PHEXW. *
		=6898 ;*	*
		=6899 ;*	Description : This routine displays the contents of a single *
		=6900 ;*	register on the screen. All the information regarding the register *
		=6901 ;*	is contained in the three memory variables given above. *
		=6902 ;*	*
		=6903 ;*	The format of the display is as follows: *
		=6904 ;*	*
		=6905 ;*	<RegisterString>:<RegisterValue><Space>. *
		=6906 ;*	*
		=6907 ;*	Examples of both 8 and 16 bit registers: *
		=6908 ;*	*
		=6909 ;*	8 bit register: A:2C *
		=6910 ;*	16 bit register: SP:1234 *
		=6911 ;*	*
		=6912 ;*	Note: A space is also output so that this routine can be used *

LOC	OBJ	LINE	SOURCE STATEMENT
		=6913 ;*	by the register dump routine.
		=6914 ;*	
		=6915 ;	=====*
		=6916 ;	
		=6917 REGDIS:	; Procedure RegisterDisplay;
23F3	E5	=6918	PUSH H ;
23F4	D5	=6919	PUSH D ;
23F5	2A93EA	=6920	LHLD REGSTR ;
23F8	CDEE18	=6921	CALL OUTMSG ; OutMessage(RegisterString);
		=6922	;
23FB	3E3A	=6923	MVI A,':' ;
23FD	CDE218	=6924	CALL KITOUT ; KitOut(':');
		=6925	;
2400	2A91EA	=6926	LHLD REGADR ;
2403	3A90EA	=6927	LDA REGTYPE ;
2406	FE00	=6928	CPI BYTEVAR ;
2408	C21224	=6929	JNZ REGD05 ; If RegisterType = ByteVariable then
		=6930	Begin
240B	7E	=6931	MOV A,M ;
240C	CD041A	=6932	CALL PHEXB ; PrintHexByte(Memory[RegisterAdr]);
		=6933	;
240F	C31924	=6934	JMP REGD10 ; End
		=6935 REGD05:	Else
		=6936	Begin
2412	5E	=6937	MOV E,M ;
2413	23	=6938	INX H ;
2414	56	=6939	MOV D,M ;
2415	EB	=6940	XCHG ;
2416	CD1D1A	=6941	CALL PHEXW ; PrintHexWord(Memory[RegisterAdr] +
		=6942	Memory[RegisterAdr+1] * 256);
		=6943	End;
		=6944 REGD10:	;
2419	3E20	=6945	MVI A, ' ' ;
241B	CDE218	=6946	CALL KITOUT ; KitOut(' ');
241E	D1	=6947	POP D ;
241F	E1	=6948	POP H ;
2420	C9	=6949	RET ; Endproc RegisterDisplay;
		=6950 ;	
		=6951 ;	=====*
		=6952 ;*	
		=6953 ;*	Function : REGDMP (Register dump)
		=6954 ;*	Inputs : None.
		=6955 ;*	Outputs : None.
		=6956 ;*	Destroys : A, B, C, D, E, H, L and flags.
		=6957 ;*	Calls : COPYENTRY and REGDIS.
		=6958 ;*	
		=6959 ;*	Description : This routine dumps the contents of ALL the registers.
		=6960 ;*	it does this by simply using every entry in the register table and
		=6961 ;*	calling the register display routine - REGDIS.
		=6962 ;*	
		=6963 ;	=====*
		=6964 ;	
		=6965 REGDMP:	; Procedure RegisterDump;
		=6966	;
2421	215524	=6967	LXI H,REGTAB ; Ptr := RegisterTable;
2424	110500	=6968	LXI D,BYTE+2*WORD ;
		=6969 RDMP05:	;
2427	7E	=6970	MOV A,M ;
2428	B7	=6971	ORA A ;
2429	CA3624	=6972	JZ RDMP10 ; While Memory[Ptr] <> 0 do
		=6973	Begin
242C	CDD623	=6974	CALL COPYENTRY ; CopyEntry;
242F	CDF323	=6975	CALL REGDIS ; RegisterDisplay;
2432	19	=6976	DAD D ; Ptr := Ptr + Length(RegTabEntry)
2433	C32724	=6977	JMP RDMP05 ; End;
		=6978 RDMP10:	;
2436	C9	=6979	RET ; EndProc RegisterDump;
		=6980 ;	
		=6981 ;	=====*
		=6982 ;*	

LOC	OBJ	LINE	SOURCE STATEMENT
		=6983 ;*	Function : REGLOOKUP (Register lookup)
		=6984 ;*	Inputs : A : Symbol.
		=6985 ;*	Outputs : Z = 0 : Symbol is not a register type.
		=6986 ;*	Z = 1 : The symbol is a register type.
		=6987 ;*	(REGSTR) : Address of register string.
		=6988 ;*	(REGADR) : Address of register storage.
		=6989 ;*	(REGTYPE) : Register type (BYTE or WORD).
		=6990 ;*	Destroys : A, B, C, D, E, H, L and flags.
		=6991 ;*	Calls : COPYENTRY.
		=6992 ;*	
		=6993 ;*	Description : This routine looks through a table to determine
		=6994 ;*	if the symbol passed to it, is a register symbol.
		=6995 ;*	
		=6996 ;*	If the symbol is a register symbol then information regarding the
		=6997 ;*	symbol is copied into memory variables for used by other routines.
		=6998 ;*	
		=6999 ;*	The table used is given below and its format is discussed.
		=7000 ;*	
		=7001 ;	=====
		=7002 ;	
		=7003	REGLOOKUP: ; Procedure RegisterLookup(Symbol:Byte);
2437	47	=7004	MOV B,A ;
2438	215524	=7005	LXI H,REGTAB ; RegTabPtr := RegisterTable;
243B	110500	=7006	LXI D,BYTE+WORD+WORD;
		=7007	REGL05: ;
243E	7E	=7008	MOV A,M ;
243F	FE00	=7009	CPI 0 ;
2441	CA5424	=7010	JZ REGL99 ; While Memory[RegTabPtr] <> 0 AND
		=7011	; ;
2444	E67F	=7012	ANI NOT VARTYPE ;
2446	B8	=7013	CMP B ;
2447	CA4E24	=7014	JZ REGL10 ; Memory[RegTabPtr].Sym <> Symbol do
		=7015	; ;
244A	19	=7016	DAD D ; RegTabPtr := RegTabPtr +
244B	C33E24	=7017	JMP REGL05 ; Length(RegTableEntry);
		=7018	; ;
		=7019	REGL10: ; If Memory[RegTabPtr] <> 0 then
		=7020	; Begin
244E	CDD623	=7021	CALL COPYENTRY ; CopyEntry;
2451	3E01	=7022	MVI A,1 ; Status := OK
2453	B7	=7023	ORA A ; End
		=7024	REGL99: ; Else
		=7025	; Status := Error;
		=7026	; Return(Status);
2454	C9	=7027	RET ; EndProc RegisterLookup;
		=7028 ;	
		=7029 ;	-----
		=7030 ;	
		=7031 ;	Register table format:
		=7032 ;	
		=7033 ;	Byte 0 - wsssssss w = 0 if register is a word variable
		=7034 ;	sssssss = Symbol type.
		=7035 ;	
		=7036 ;	Byte 1,2 - Address of word in data area.
		=7037 ;	
		=7038 ;	Byte 3,4 - Address of register name, in string format.
		=7039 ;	
		=7040 ;	Note: The table is terminated by an entry with a zero first byte.
		=7041 ;	
		=7042	REGTAB: ; Start of register table.
		=7043	; ;
2455	07	=7044	DB SYMA OR BYTEVAR ; A register
2456	9CEA	=7045	DW REGA ;
2458	6F1E	=7046	DW STRA ;
		=7047	; ;
245A	09	=7048	DB SYMB OR BYTEVAR ; B register
245B	96EA	=7049	DW REGB ;
245D	781E	=7050	DW STRB ;
		=7051	; ;
245F	0B	=7052	DB SYMC OR BYTEVAR ; C register

LOC	OBJ	LINE	SOURCE STATEMENT
2460	95EA	=7053	DW REGC ;
2462	821E	=7054	DW STRC ;
		=7055	;
2464	0E	=7056	DB SYMD OR BYTEVAR ; D register
2465	98EA	=7057	DW REGD ;
2467	951E	=7058	DW STRD ;
		=7059	;
2469	11	=7060	DB SYME OR BYTEVAR ; E register
246A	97EA	=7061	DW REGE ;
246C	A41E	=7062	DW STRE ;
		=7063	;
246E	12	=7064	DB SYMFLAGS OR BYTEVAR ; Flags register
246F	9BEA	=7065	DW FLAGS ;
2471	A71E	=7066	DW STRFLAGS ;
		=7067	;
2473	18	=7068	DB SYMH OR BYTEVAR ; H register
2474	9AEA	=7069	DW REGH ;
2476	D01E	=7070	DW STRH ;
		=7071	;
2478	1C	=7072	DB SYML OR BYTEVAR ; L register
2479	99EA	=7073	DW REGL ;
247B	E61E	=7074	DW STRL ;
		=7075	;
247D	9F	=7076	DB SYMM OR WORDVAR ; HL register pair
247E	99EA	=7077	DW REGM ;
2480	F61E	=7078	DW STRM ;
		=7079	;
2482	A3	=7080	DB SYMPC OR WORDVAR ; Program counter
2483	9DEA	=7081	DW REGPC ;
2485	0A1F	=7082	DW STRPC ;
		=7083	;
2487	AA	=7084	DB SYMSP OR WORDVAR ; Stack pointer
2488	9FEA	=7085	DW REGSP ;
248A	371F	=7086	DW STRSP ;
		=7087	;
248C	00	=7088	DB 0 ; Table termination.
		=7089 ;	
		=7090 ;	=====*
		=7091 ;*	*
		=7092 ;*	Function : REGROU (REG command routine). *
		=7093 ;*	Inputs : None. *
		=7094 ;*	Outputs : None. *
		=7095 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=7096 ;*	Calls : GETSYM, REGDMP, ERROR, CMDERR, REGLOOKUP, REGDIS *
		=7097 ;*	and REGUPDATE. *
		=7098 ;*	*
		=7099 ;*	Description : This routine handles the REG command. The syntax for *
		=7100 ;*	the command is given below: *
		=7101 ;*	*
		=7102 ;*	BNF syntax: REG [RegName [= Number]] *
		=7103 ;*	*
		=7104 ;*	RegName = A B C D E H L M SP PC Flags *
		=7105 ;*	*
		=7106 ;*	Some example of the REG command (with a brief explanation) are *
		=7107 ;*	given below: *
		=7108 ;*	*
		=7109 ;*	REG - Display the values of ALL registers. *
		=7110 ;*	REG SP - Display the value of the stack pointer. *
		=7111 ;*	REG PC = 100H - Set the value of the program counter equal to 100H *
		=7112 ;*	*
		=7113 ;	=====*
		=7114 ;	
		=7115 REGROU:	; Procedure RegisterRoutine;
		=7116	;
248D	CDFF1B	=7117	CALL GETSYM ; Sym := GetNextSymbol;
2490	FE00	=7118	CPI EOL ;
2492	C29E24	=7119	JNZ REGR05 ; If Sym = EOL then
		=7120	Begin
2495	CD2124	=7121	CALL REGDMP ; RegisterDump;
2498	CDB419	=7122	CALL CRLF ; Newline

LOC	OBJ	LINE	SOURCE STATEMENT
249B	C3DF24	=7123	JMP REGR99 ; End
		=7124	REGRO5: ; Else
		=7125	; Begin
249E	CD3724	=7126	CALL REGLOOKUP ;
24A1	CAD024	=7127	JZ REGR80 ; If RegisterLookup(Sym) then begin
		=7128	; Sym := GetNextSymbol;
24A4	CDFF1B	=7129	CALL GETSYM ;
24A7	FE00	=7130	CPI EOL ;
24A9	C2B524	=7131	JNZ REGR10 ; If Sym = EOL then begin
		=7132	; RegisterDisplay;
24AC	CDFF323	=7133	CALL REGDIS ; Newline;
24AF	CDB419	=7134	CALL CRLF ;
24B2	C3DF24	=7135	JMP REGR99 ; End
		=7136	REGRO10: ; Else
24B5	FE04	=7137	CPI EQUALS ;
24B7	C2D924	=7138	JNZ REGR90 ; If Sym = EQUALS then begin
		=7139	; Sym := GetNextSymbol;
24BA	CDFF1B	=7140	CALL GETSYM ;
24BD	FE02	=7141	CPI NUMBER ;
24BF	C2D924	=7142	JNZ REGR90 ; If Sym = Number then begin
		=7143	; Sym := GetNextSymbol;
24C2	CDFF1B	=7144	CALL GETSYM ;
24C5	FE00	=7145	CPI EOL ;
24C7	C2D924	=7146	JNZ REGR90 ; If Sym = EOL then
		=7147	; RegisterUpdate;
24CA	CD2E25	=7148	CALL REGUPDATE ;
24CD	C3DF24	=7149	JMP REGR99 ;
		=7150	; End Else
		=7151	; Error;
		=7152	REGRO80: ; End Else
24D0	21E024	=7153	LXI H, REGTER ;
24D3	CDC119	=7154	CALL ERROR ; ErrorMessage(RegisterError);
24D6	C3DF24	=7155	JMP REGR99 ;
		=7156	REGRO90: ;
24D9	21F424	=7157	LXI H, ERMREG ;
24DC	CD8119	=7158	CALL CMDERR ;
		=7159	REGRO99: ;
24DF	C9	=7160	RET ; EndProc RegisterRoutine;
		=7161	;
		=7162	REGTER: ;
		=7163	;
		=7164	format: ;
		=7165	;
		=7166	Invalid register type ;
		=7167	;
		=7168	declaration: ;
		=7169	;
24E0	496E7661	=7170	DB 'Invali', 'd'+MSB, 'registe', 'r'+MSB, 'type', EndMsg
	6C69E472		
	65676973		
	7465F274		
	79706500		
		=7171	;
		=7172	ERMREG: ;
		=7173	;
		=7174	format: ;
		=7175	;
		=7176	REG [<RegName> [= <Number>]]
		=7177	<RegName> = A B C D E H L M SP PC FLAGS
		=7178	;
		=7179	declaration: ;
		=7180	;
24F4	5245C7DB	=7181	DB 'RE', 'G'+MSB, '['+MSB, 'RegName', '>'+MSB
	5265674E		
	616D65BE		
2500	DBBD80DD	=7182	DB '['+MSB, '='+MSB, PSNUM, ']+MSB, ']'
	5D		
2505	OD0A09	=7183	DB CR, LF, TAB
2508	3C526567	=7184	DB '<RegName', '>'+MSB, '='+MSB
	4E616D65		

LOC	OBJ	LINE	SOURCE STATEMENT
	BEBD		
2512	417C427C	=7185	DB 'A B C D E H L M SP PC FLAGS', EndMsg
	437C447C		
	457C487C		
	4C7C4D7C		
	53507C50		
	437C464C		
	41475300		
		=7186 ;	
		=7187 ;=====*	
		=7188 ;*	
		=7189 ;* Function : REGUPDATE (Register update).	
		=7190 ;* Inputs : (SYMBOL) : New register value.	
		=7191 ;* (REGADR) : Address of the register storage.	
		=7192 ;* (REGTYPE) : Register type (BYTE or WORD).	
		=7193 ;* Outputs : None.	
		=7194 ;* Destroys : A, B, C, D, E, H, L and flags.	
		=7195 ;* Calls : ERROR.	
		=7196 ;*	
		=7197 ;* Description : This routine updates the value of a register stored	
		=7198 ;* in memory. The new value of the register is given by the value of	
		=7199 ;* the last number read (i.e. SYMBOL). The register type and address	
		=7200 ;* is obtained from the memory variables REGADR and REGTYPE.	
		=7201 ;*	
		=7202 ;* Note: A range check is performed when writing to 8 bit registers.	
		=7203 ;*	
		=7204 ;* i.e. the command 'REG A = 9999' will cause an error as the A	
		=7205 ;* register is an 8 bit register and the value 9999 cannot be	
		=7206 ;* stored in it.	
		=7207 ;*	
		=7208 ;=====*	
		=7209 ;	
		=7210 REGUPDATE:	; Procedure RegisterUpdate;
252E	2A00EA	=7211 LHL	SYMBOL ;
2531	EB	=7212 XCHG	; ;
2532	2A91EA	=7213 LHL	REGADR ;
2535	3A90EA	=7214 LDA	REGTYPE ;
2538	FE00	=7215 CPI	BYTEVAR ;
253A	C24F25	=7216 JNZ	REGU20 ; If RegType = ByteVariable then
		=7217 ;	; Begin
253D	7A	=7218 MOV	A,D ;
253E	B7	=7219 ORA	A ;
253F	C24625	=7220 JNZ	REGU05 ; If High Symbol = 0 then
		=7221 ;	; Memory(RegisterAddress):=low Symbol;
2542	73	=7222 MOV	M,E ;
2543	C34C25	=7223 JMP	REGU10 ;
		=7224 REGU05:	; Else
2546	215325	=7225 LXI	H, VALERR ;
2549	CDC119	=7226 CALL	ERROR ; Error(ValueRangeError);
		=7227 REGU10:	; ;
254C	C35225	=7228 JMP	REGU99 ; End
		=7229 REGU20:	; Else
		=7230 ;	; Begin
254F	73	=7231 MOV	M,E ; Memory(RegisterAddress):=Low Symbol;
2550	23	=7232 INX	H ;
2551	72	=7233 MOV	M,D ; Memory(RegisterAddress+1):=High Symbol;
		=7234 ;	; End;
		=7235 REGU99:	; ;
2552	C9	=7236 RET	; EndProc RegisterUpdate;
		=7237 ;	
		=7238 VALERR:	
		=7239 ;	
		=7240 ; format:	
		=7241 ;	
		=7242 ; 8 bit value expected	
		=7243 ;	
		=7244 ; declaration:	
		=7245 ;	
2553	B86269F4	=7246 DB	'8'+MSB, 'bi', 't'+MSB, 'valu', 'e'+MSB, 'expected', EndMsg
	76616C75		

DEB LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 123

LOC	OBJ	LINE	SOURCE STATEMENT
	E5657870		
	65637465		
	6400		
		=7247 ;	
		=7248 ;	END <DEBREG.A85>
		7249 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 124

LOC	OBJ	LINE	SOURCE STATEMENT
		7250 ;	
		7251 ;	DISASM command routine:
		7252 ;	
		7253	\$INCLUDE(DEBDIS.A85)
		=7254 ;	-----;
		=7255 ;	;
		=7256 ;	Disassembler routines used by the UCT SABus debugger. ;
		=7257 ;	;
		=7258 ;	This module contains (in order) the following subroutines. ;
		=7259 ;	;
		=7260 ;	a) DECLINE - Decrement line count. ;
		=7261 ;	b) DISASM - Disassemble range. ;
		=7262 ;	c) DISLINE - Disassemble a single line. ;
		=7263 ;	d) DISNEXT - Disassemble next twenty lines. ;
		=7264 ;	e) DISROU - DISASM command routine. ;
		=7265 ;	f) GETINFO - Get instruction information. ;
		=7266 ;	g) GSTRADR - Get string address. ;
		=7267 ;	h) PLABEL - Print labels. ;
		=7268 ;	i) PNREG - Print Nth register. ;
		=7269 ;	j) POPERAND - Print operand. ;
		=7270 ;	k) PRINTL - Print label. ;
		=7271 ;	l) PRTADR - Print address. ;
		=7272 ;	m) PRTBYTES - Print bytes. ;
		=7273 ;	n) PRTOPRS - Print operands. ;
		=7274 ;	o) PRTREG - Print register. ;
		=7275 ;	p) VALLOOK - Value lookup. ;
		=7276 ;	;
		=7277 ;	-----;
		=7278	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 125

LOC	OBJ	LINE	SOURCE STATEMENT
		=7279 ;	=====*
		=7280 ;*	*
		=7281 ;*	Function : DECLINES (Decrement line counter). *
		=7282 ;*	Inputs : (LINES) : Number of lines left on the screen. *
		=7283 ;*	Outputs : (LINES) : Number of lines left on the screen. *
		=7284 ;*	Destroys : A and flags. *
		=7285 ;*	Calls : None. *
		=7286 ;*	*
		=7287 ;*	Description : This routine decrements the number of lines left on *
		=7288 ;*	the screen. *
		=7289 ;*	*
		=7290 ;	=====*
		=7291 ;	
		=7292	DECLINES: ; Procedure DecrementLines;
2565	3AADEA	=7293	LDA LINES ;
2568	3D	=7294	DCR A ; Lines := Lines - 1;
2569	32ADEA	=7295	STA LINES ;
256C	C9	=7296	RET ; EndProc DecrementLines;
		=7297 ;	
		=7298 ;	=====*
		=7299 ;*	*
		=7300 ;*	Function : DISASM (Disassemble range). *
		=7301 ;*	Inputs : (RBEGIN) : Start address of the range to disassemble. *
		=7302 ;*	(REND) : End address of the range. *
		=7303 ;*	Outputs : None. *
		=7304 ;*	Destroys : A, D, E, H, L and flags. *
		=7305 ;*	Calls : DISLINE, ADDBC, OUTMSG, DCMP, CRLF and KITIN. *
		=7306 ;*	*
		=7307 ;*	Description : This routine disassembles the range given by the two *
		=7308 ;*	memory variables RBEGIN and REND. RBEGIN contains the start *
		=7309 ;*	address and REND the end address. The disassembly is stopped every *
		=7310 ;*	twenty lines to allow the user to read what is on the screen *
		=7311 ;*	before continueing. *
		=7312 ;*	*
		=7313 ;	=====*
		=7314 ;	
		=7315	DISASM: ; Procedure Disassemble;
		=7316	;
256D	2A7DEA	=7317	LHLD REND ; EndPtr := RangeEnd;
2570	EB	=7318	XCHG ;
2571	2A7BEA	=7319	LHLD RBEGIN ; Ptr := RangeBegin
		=7320	;
		=7321	;
2574	3E14	=7322	MVI A,20 ;
2576	32ADEA	=7323	STA LINES ; Lines := 20;
		=7324	;
		=7325	DISA05: ; Repeat
		=7326	;
2579	CDD725	=7327	CALL DISLINE ; DiassembleLine(Ptr);
		=7328	;
257C	CDC621	=7329	CALL ADDBC ; AddByteCount;
		=7330	;
257F	3AADEA	=7331	LDA LINES ;
2582	B7	=7332	ORA A ;
2583	F2A625	=7333	JP DISA15 ; If Line >= 0 then
		=7334	Begin
2586	3E14	=7335	MVI A,20 ;
2588	32ADEA	=7336	STA LINES ; Lines := 20;
258B	E5	=7337	PUSH H ;
258C	21B025	=7338	LXI H, WAITMSG ;
258F	CDEE18	=7339	CALL OUTMSG ; OutputMessage(WaitMessage);
2592	E1	=7340	POP H ;
2593	CDDF18	=7341	CALL KITIN ; Ch := KitInput;
2596	FE1B	=7342	CPI ESC ;
2598	CCB419	=7343	CZ CRLF ; If Ch = ESC then
		=7344	Begin
		=7345	;
		=7346	Newline;
259B	CAAC25	=7347	ExitProcedure
		=7348	End;
			;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 126

LOC	OBJ	LINE	SOURCE STATEMENT
259E	E5	=7349	PUSH H ;
259F	21D325	=7350	LXI H,CLRMSG ;
25A2	CDEE18	=7351	CALL OUTMSG ; OutMessage(ClearMessage);
25A5	E1	=7352	POP H ;
		=7353	DISA15: ; End;
25A6	CDBF1A	=7354	CALL DCMF ;
25A9	DA7925	=7355	JC DISA05 ; Until Ptr > EndPtr;
		=7356	DISA20: ;
25AC	22ABEA	=7357	SHLD LASTDIS ; LastDisassebly := Ptr;
		=7358	; /
25AF	C9	=7359	RET ; End Disassemble;
		=7360	;
		=7361	WAITMSG:
		=7362	;
		=7363	format:
		=7364	;
		=7365	Press any key to continue or <ESC> to exit
		=7366	;
		=7367	declaration:
		=7368	;
25B0	50726573 F3616EF9 6B65F9	=7369	DB 'Pres', 's'+MSB, 'an', 'y'+MSB, 'ke', 'y'+MSB
25BB	74EF636F 6E74696E 75E56FF2	=7370	DB 't', 'o'+MSB, 'contin', 'e'+MSB, 'o', 'r'+MSB
25C7	3C455343 BE74EF65 78697400	=7371	DB '<ESC', '>'+MSB, 't', 'o'+MSB, 'exi', 't', EndMsg
		=7372	;
		=7373	CLRMSG:
		=7374	;
		=7375	format:
		=7376	;
		=7377	Control sequence to clear the current line (using VT52 codes)
		=7378	;
		=7379	declaration:
		=7380	;
25D3	0D1B4B00	=7381	DB CR, ESC, CLRLINE, EndMsg
		=7382	;
		=7383	=====*
		=7384	;
		=7385	* Function : DISLINE (Disassemble line) *
		=7386	* Inputs : HL = Program counter. *
		=7387	* Outputs : None. *
		=7388	* Destroys : A and flags. *
		=7389	* Calls : PLABEL, GETINFO, PRTADR, PRTBYTES, OUTSTR, PRTOPRS *
		=7390	* DECLINES and CRLF. *
		=7391	;
		=7392	* Description : This routine disassembles a single instruction *
		=7393	* starting at the address passed in the HL register pair. *
		=7394	;
		=7395	* If the disassembly is done in symbolic format then any values in *
		=7396	* the symbol table that are equal to the program counter are *
		=7397	* displayed as labels. *
		=7398	;
		=7399	* All the information regarding the instruction is looked up in the *
		=7400	* main instruction table using the GETINFO routine. The outputs of *
		=7401	* this routine are a series of memory variables. More information *
		=7402	* about these memory variables can be obtained by looking at the *
		=7403	* header on the GETINFO routine. *
		=7404	;
		=7405	=====*
		=7406	;
		=7407	DISLINE: ; Procedure DisassembleLine(PC:Int);
		=7408	;
25D7	E5	=7409	PUSH H ;
25D8	D5	=7410	PUSH D ;
25D9	C5	=7411	PUSH B ;
25DA	22A9EA	=7412	SHLD DISPC ; DisassemblyPC := PC;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 127

LOC	OBJ	LINE	SOURCE STATEMENT
25DD	3A7AEA	=7413	LDA SYMBOLIC ;
25E0	FEFF	=7414	CPI TRUE ; If SymbolicDisassembly then
25E2	CCB226	=7415	CZ PLABEL ; Printlabel;
25E5	7E	=7416	MOV A,M ;
25E6	32A8EA	=7417	STA OPCODE ; Opcode := memory(PC);
25E9	CD6226	=7418	CALL GETINFO ; GetInfo(Opcod);
25EC	CD0528	=7419	CALL PRTADR ; PrintAddress(PC);
25EF	3AA3EA	=7420	LDA BYTECOUNT ;
25F2	CD0E28	=7421	CALL PRTBYTES ; PrintBytes(ByteCount);
25F5	2AA6EA	=7422	LHLD STRADR ;
25F8	CDF219	=7423	CALL OUTSTR ; OutputMessage(StringAddress);
25FB	CDB31A	=7424	CALL TABOUT ;
25FE	3AA4EA	=7425	LDA OPERAND ;
2601	CD3028	=7426	CALL PRTOPRS ; PrintOperands(Operands, PC)
2604	CD6525	=7427	CALL DECLINES ; DecrementLines;
2607	CDB419	=7428	CALL CRLF ; NewLine;
260A	C1	=7429	POP B ;
260B	D1	=7430	POP D ;
260C	E1	=7431	POP H ;
260D	C9	=7432	RET ; EndProc DisassembleLine;
		=7433 ;	
		=7434 ;	=====*
		=7435 ;	*
		=7436 ;	* Function : DISNEXT (Disassemble the next range). *
		=7437 ;	* Inputs : (LASTDIS) : Start address of the disassembly. *
		=7438 ;	* Outputs : None. *
		=7439 ;	* Destroys : A, H, L and flags. *
		=7440 ;	* Calls : DISLINE and ADDBC. *
		=7441 ;	*
		=7442 ;	* Description : This routine disassembles a range of memory starting *
		=7443 ;	* at the address given by the memory variable LASTDIS. The *
		=7444 ;	* disassembly continues until 20 lines have been displayed. These *
		=7445 ;	* lines may be code lines or they may be instructions. *
		=7446 ;	*
		=7447 ;	=====*
		=7448 ;	
		=7449	DISNEXT: ; Procedure DisassembleNext;
260E	3E14	=7450	MVI A,20 ;
2610	32ADEA	=7451	STA LINES ; Lines := 20;
		=7452	; DisPc := LastDis;
2613	2AABEA	=7453	LHLD LASTDIS ;
		=7454	; Repeat
		=7455	DISN05: ;
		=7456	; DisassembleLine(DisPc)
2616	CDD725	=7457	CALL DISLINE ; DisPc := DisPc + ByteCount;
2619	CDC621	=7458	CALL ADDBC ;
		=7459	; Until Line < 0;
261C	3AADEA	=7460	LDA LINES ;
261F	B7	=7461	ORA A ;
2620	F21626	=7462	JP DISN05 ; LastDis := DisPc;
		=7463	; EndProc DisassembleNext;
2623	22ABEA	=7464	SHLD LASTDIS ;
		=7465	; EndProc DisassembleNext;
2626	C9	=7466	RET ; EndProc DisassembleNext;
		=7467 ;	
		=7468 ;	=====*
		=7469 ;	*
		=7470 ;	* Function : DISROU (Disassembly routine) *
		=7471 ;	* Inputs : None. *
		=7472 ;	* Outputs : None. *
		=7473 ;	* Destroys : All. *
		=7474 ;	* Calls : GETSYM, DISNEXT, CMDERR and DISASM. *
		=7475 ;	*
		=7476 ;	* Description : This routine handles the disassembly routine. This *
		=7477 ;	* routine allows the user to display, in disassembled format, a *
		=7478 ;	* range of memory. The BNF syntax of the DISASM command is given *
		=7479 ;	* below: *
		=7480 ;	*
		=7481 ;	* Syntax: DISASM [<Number>] [TO LENGTH <Number>] <EOL> *
		=7482 ;	*

LOC	OBJ	LINE	SOURCE STATEMENT
		=7483 ;*	Some examples of the DISASM command and their meanings are given *
		=7484 ;*	below: *
		=7485 ;*	*
		=7486 ;*	DIS - Continue disassembling where you left off during *
		=7487 ;*	the last disassembly command. The routine *
		=7488 ;*	will disassemble the next 20 lines. *
		=7489 ;*	DIS 100 - Disassemble the instruction in memory at 100. *
		=7490 ;*	DIS 0 TO 20 - Disassemble the range 0 to 20. *
		=7491 ;*	DIS 10 LENGTH 20 - Disassemble the 20 bytes starting at address 10. *
		=7492 ;*	*
		=7493 ;	===== *
		=7494 ;	*
		=7495	DISROU: ; Procedure DisassemblyRoutine;
		=7496	;
2627	CDFF1B	=7497	CALL GETSYM ; Sym := GetNextSymbol;
262A	FE00	=7498	CPI EOL ; If Sym = EOL then
262C	C23526	=7499	JNZ DISR05 ;
		=7500	;
262F	CD0E26	=7501	CALL DISNEXT ; DisassembleNextRange;
2632	C34E26	=7502	JMP DISR99 ;
		=7503	DISR05: ;
2635	FE02	=7504	CPI NUMBER ; ElseIf Sym = Number then
2637	C24826	=7505	JNZ DISR90 ;
		=7506	;
263A	CD821F	=7507	CALL GETRANGE ; If NOT GetRangeError and
263D	FE00	=7508	CPI EOL ; Sym = EOL then
263F	C24826	=7509	JNZ DISR90 ;
		=7510	;
2642	CD6D25	=7511	CALL DISASM ; DiassembleRange;
2645	C34E26	=7512	JMP DISR99 ;
		=7513	DISR90: ; Else
2648	214F26	=7514	LXI H,ERMDIS ;
264B	CD8119	=7515	CALL CMDERR ; CommandError(DisassemblyCommandError);
		=7516	DISR99: ;
264E	C9	=7517	RET ; EndProc DisassemblyRoutine;
		=7518 ;	;
		=7519	ERMDIS:
		=7520 ;	;
		=7521 ;	format:
		=7522 ;	;
		=7523 ;	DIS [<Number>] [TO LENGTH <Number>]
		=7524 ;	;
		=7525 ;	declaration:
		=7526 ;	;
264F	4449D3DB	=7527	DB 'DI', 'S'+MSB, '['+MSB, PSNUM, ']+MSB
	80DD		
2655	DB544F7C	=7528	DB '['+MSB, 'TO LENGT', 'H'+MSB, PSNUM, ']', EndMsg
	4C454E47		
	54C8805D		
	00		
		=7529 ;	;
		=7530 ;	===== *
		=7531 ;*	*
		=7532 ;*	Function : GETINFO (Get instruction information). *
		=7533 ;*	Inputs : A : Opcode to be looked up. *
		=7534 ;*	Outputs : The following memory variables. *
		=7535 ;*	*
		=7536 ;*	(STRADR) : Address of the mnemonic string. *
		=7537 ;*	(EMINFO) : Exectute/Emulate information field. *
		=7538 ;*	(BYTECOUNT) : Byte count. *
		=7539 ;*	(OPERAND) : Operand byte (two nibbles). *
		=7540 ;*	*
		=7541 ;*	Destroys : A and flags. *
		=7542 ;*	Calls : GSTRADR. *
		=7543 ;*	*
		=7544 ;*	Description : This routine uses the byte passed in the A register *
		=7545 ;*	to do a full lookup for the instnction. For table format *
		=7546 ;*	information see 'DEBTABLE.A85' file. *
		=7547 ;*	*
		=7548 ;	===== *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 129

LOC	OBJ	LINE	SOURCE STATEMENT
		=7549 ;	
		=7550 GETINFO:	; Procedure GetInfo(Opcode:Byte);
2662	E5	=7551	PUSH H ;
2663	D5	=7552	PUSH D ;
2664	111812	=7553	LXI D,DISTAB ;
2667	2600	=7554	MVI H,0 ;
2669	6F	=7555	MOV L,A ;
266A	29	=7556	DAD H ;
266B	19	=7557	DAD D ;
266C	5E	=7558	MOV E,M ;
266D	23	=7559	INX H ;
266E	56	=7560	MOV D,M ; Reg.DE := DisTable[Opcode*2]
266F	7B	=7561	MOV A,E ;
2670	E67F	=7562	ANI STRMASK ; StringNumber := Reg.DE AND STRMASK;
2672	CD9A26	=7563	CALL GSTRADR ;
2675	22A6EA	=7564	SHLD STRADR ; StringAddress :=
		=7565	GetStringAddress(StringNumber);
2678	7A	=7566	MOV A,D ;
2679	0F	=7567	RRC ;
267A	0F	=7568	RRC ;
267B	0F	=7569	RRC ;
267C	E61F	=7570	ANI EMMASK ;
267E	32A5EA	=7571	STA EMINFO ; EmulateInfo := (Reg.DE SHR 11) AND EMMASK;
2681	EB	=7572	XCHG ;
2682	29	=7573	DAD H ;
2683	7C	=7574	MOV A,H ;
2684	E60F	=7575	ANI OPRMASK ; OperandType := (Reg.DE SHR 7) AND OPRMASK;
2686	11D410	=7576	LXI D,OPRTAB ;
2689	2600	=7577	MVI H,0 ;
268B	6F	=7578	MOV L,A ;
268C	29	=7579	DAD H ;
268D	19	=7580	DAD D ;
268E	7E	=7581	MOV A,M ;
268F	32A3EA	=7582	STA BYTECOUNT ; ByteCount := OperandTable[OperandType*2];
2692	23	=7583	INX H ;
2693	7E	=7584	MOV A,M ;
2694	32A4EA	=7585	STA OPERAND ; Operand := OperandTable[OperandType*2+1];
2697	D1	=7586	POP D ;
2698	E1	=7587	POP H ;
2699	C9	=7588	RET ; EndProc GetInfo;
		=7589 ;	
		=7590 ;=====*	
		=7591 ;*	
		=7592 ;* Function : GSTRADR (Get string address).	*
		=7593 ;* Inputs : A : String number.	*
		=7594 ;* Outputs : HL : Start address of the n th string in the string	*
		=7595 ;* table. (n is passed in the A register).	*
		=7596 ;* Destroys : H, L, A and flags.	*
		=7597 ;* Calls : None.	*
		=7598 ;*	*
		=7599 ;* Description : This routine skips thru the string table looking for	*
		=7600 ;* the nth entry in the string table. When it finds the nth entry in	*
		=7601 ;* the table, its start address is returned in the HL register pair.	*
		=7602 ;*	*
		=7603 ;* NOTE: Each entry in the string table is terminated by a character	*
		=7604 ;* with its top bit set.	*
		=7605 ;*	*
		=7606 ;=====*	
		=7607 ;	
		=7608 GSTRADR:	; Procedure GetStringAddress(StringNum:Byte);
		=7609	;
269A	21EC10	=7610	LXI H,STRTAB ; Ptr := StringTable;
		=7611	GSA10: ;
269D	B7	=7612	ORA A ;
269E	CAB126	=7613	JZ GSA20 ; While StringNum <> 0 do
		=7614	;
		=7615	Begin
26A1	F5	=7616	GSA15: PUSH PSW ;
		=7617	MOV A,M ;
26A2	7E	=7618	ORA A ;
26A3	B7		

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 130

LOC	OBJ	LINE	SOURCE STATEMENT
26A4	FAAB26	=7619	JM GSA18 ; While Memory[Ptr] > 0 do
		=7620	;
26A7	23	=7621	INX H ; Ptr := Ptr + 1;
26A8	C3A226	=7622	JMP GSA15 ;
		=7623	GSA18: ;
26AB	23	=7624	INX H ; Ptr := Ptr + 1;
26AC	F1	=7625	POP PSW ;
26AD	3D	=7626	DCR A ; StringNum := StringNum - 1;
		=7627	;
26AE	C39D26	=7628	JMP GSA10 ; End;
		=7629	GSA20: ;
		=7630	;
		=7631	Return(Ptr);
26B1	C9	=7632	RET ; EndProc GetStringAddress;
		=7633	;
		=7634	=====*
		=7635	;* *
		=7636	;* Function : PLABEL (Print labels). *
		=7637	;* Inputs : HL : Current disassembly program counter). *
		=7638	;* Outputs : None. *
		=7639	;* Destroys : A and flags. *
		=7640	;* Calls : DCMF, TABOUT, PHEXW, CRLF, DELINES, KITOUT and PRINTL. *
		=7641	;* *
		=7642	;* Description : This routine prints labels if the current program *
		=7643	;* counter is equal to the value of any of the symbols in the symbol *
		=7644	;* table. *
		=7645	;* *
		=7646	=====*
		=7647	;
		=7648	PLABEL: ; Procedure PrintLabelField(PC:Integer);
26B2	C5	=7649	PUSH B ;
26B3	D5	=7650	PUSH D ;
26B4	E5	=7651	PUSH H ;
26B5	EB	=7652	XCHG ;
26B6	2100E0	=7653	LXI H,STABLE ; Ptr := SymbolTable;
26B9	0600	=7654	MVI B,0 ;
		=7655	PLAB05: ; For I := 1 to 256 do begin
26BB	7E	=7656	MOV A,M ;
26BC	B7	=7657	ORA A ;
26BD	CAF126	=7658	JZ PLAB10 ; If Memory[Ptr] <> 0 then
		=7659	Begin
26C0	E5	=7660	PUSH H ;
26C1	D5	=7661	PUSH D ;
26C2	110600	=7662	LXI D,LABLEN ;
26C5	19	=7663	DAD D ; TempPtr := Ptr + Offset(Value);
26C6	5E	=7664	MOV E,M ;
26C7	23	=7665	INX H ;
26C8	56	=7666	MOV D,M ;
26C9	EB	=7667	XCHG ;
		=7668	;
26CA	D1	=7669	POP D ;
26CB	CDBF1A	=7670	CALL DCMF ;
26CE	E1	=7671	POP H ;
26CF	C2F126	=7672	JNZ PLAB10 ; If SymValue = PC then
		=7673	Begin
26D2	E5	=7674	PUSH H ;
26D3	2AA9EA	=7675	LHLD DISPC ;
26D6	CD1D1A	=7676	CALL PHEXW ; PrintHexWord(PC);
26D9	CDB31A	=7677	CALL TABOUT ; TabOutput;
26DC	CDB31A	=7678	CALL TABOUT ; TabOutput;
26DF	CDB31A	=7679	CALL TABOUT ; TabOutput;
26E2	E1	=7680	POP H ;
		=7681	;
26E3	CDF327	=7682	CALL PRINTL ; PrintLabel;
26E6	3E3A	=7683	MVI A,':' ;
26E8	CDE218	=7684	CALL KITOUT ; KitOut(':');
26EB	CDB419	=7685	CALL CRLF ; Newline;
26EE	CD6525	=7686	CALL DECLINES ; Declinecount;
		=7687	;
		=7688	PLAB10: ; End;
			End

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 131

LOC	OBJ	LINE	SOURCE STATEMENT
26F1	C5	=7689	PUSH B ;
26F2	010800	=7690	LXI B, LABLEN+WORD ;
26F5	09	=7691	DAD B ; Ptr := Ptr + LengthSymbolTableEntry;
26F6	C1	=7692	POP B ;
26F7	05	=7693	DCR B ;
26F8	C2BB26	=7694	JNZ PLAB05 ; End;
		=7695	;
26FB	E1	=7696	POP H ;
26FC	D1	=7697	POP D ;
26FD	C1	=7698	POP B ;
26FE	C9	=7699	RET ; EndProc PrintLabelField;
		=7700 ;	
		=7701 ;	=====*
		=7702 ;	*
		=7703 ;	* Function : PNREG (Print Nth register). *
		=7704 ;	* Inputs : HL : Pointer to a table of register strings. *
		=7705 ;	* A : The register number (0..3) *
		=7706 ;	* Outputs : None. *
		=7707 ;	* Destroys : A, H, L and flags. *
		=7708 ;	* Calls : OUTSTR. *
		=7709 ;	*
		=7710 ;	* Description : This routine outputs the n string in the table *
		=7711 ;	* pointed to by the HL register pair. *
		=7712 ;	*
		=7713 ;	=====*
		=7714 ;	
		=7715	PNREG: ; Procedure PrintNthRegister (RegNum:Integer;
		=7716	; Ptr:Integer);
		=7717	;
26FF	B7	=7718	ORA A ; While RegNum <> 0 do
2700	CA1327	=7719	JZ PNR20 ;
		=7720	; Begin
2703	3D	=7721	DCR A ; RegNum := RegNum - 1;
2704	F5	=7722	PUSH PSW ;
		=7723	PNR10: ;
2705	7E	=7724	MOV A,M ;
2706	B7	=7725	ORA A ; While memory[Ptr] > 0 do
2707	FA0E27	=7726	JM PNR15 ;
		=7727	; Begin
270A	23	=7728	INX H ; Ptr := Ptr + 1
270B	C30527	=7729	JMP PNR10 ; End;
		=7730	PNR15: ;
270E	23	=7731	INX H ; Ptr := Ptr + 1;
270F	F1	=7732	POP PSW ;
2710	C3FF26	=7733	JMP PNREG ; End;
		=7734	PNR20: ;
2713	CDF219	=7735	CALL OUTSTR ; PrintMessage(Ptr)
		=7736	;
2716	C9	=7737	RET ; EndProc PrintNthRegister;
		=7738 ;	
		=7739 ;	=====*
		=7740 ;	*
		=7741 ;	* Function : POPERAND (Print a single operand). *
		=7742 ;	* Inputs : A : Operand type (Possible values: 0..7) *
		=7743 ;	* Outputs : None. *
		=7744 ;	* Destroys : A, D, E, H, L and flags. *
		=7745 ;	* Calls : PRTREG, VALLOOK, KITOUT, RHEXB, PRINTL, PHEXW and PNREG *
		=7746 ;	*
		=7747 ;	* Description : This routine prints a single operand. There are 8 *
		=7748 ;	* possible operands. These are: *
		=7749 ;	*
		=7750 ;	* SREG - Source register - Instruction = xxSSSxxx *
		=7751 ;	* DREG - Destination register - Instruction = xxxxxDDD *
		=7752 ;	* DATA8 - 8 bit data - Given by the byte following the opcode. *
		=7753 ;	* DATA16 - 16 bit data - given by the two bytes followings the *
		=7754 ;	* opcode. *
		=7755 ;	* RPARITH - Register pair arithmetic - xxRPxxxx *
		=7756 ;	* RPLIMIT - Register pair limited - xxxRxxxx *
		=7757 ;	* RPSTACK - Register pair Stack - xxRPxxxx *
		=7758 ;	* RSTNUM - Restart number xxRRRxxx *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 132

LOC	OBJ	LINE	SOURCE STATEMENT
		=7759	;*
		=7760	;-----*
		=7761	;
		=7762	POPERAND: ; Procedure PrintOperand(Opr:Areg);
		=7763	;
2717	FE01	=7764	CPI SREG ; Case operand of
2719	C22727	=7765	JNZ POPR05 ; SourceRegister :
		=7766	;
271C	3AA8EA	=7767	LDA OPCODE ;
271F	E607	=7768	ANI SREGMASK ;
2721	CD4E28	=7769	CALL PRTREG ; PrintRegister(Opcod and
2724	C3E527	=7770	JMP POPR99 ; SourceRegisterMask);
		=7771	POPR05: ;
2727	FE02	=7772	CPI DREG ;
2729	C23A27	=7773	JNZ POPR10 ; DestinationRegister :
		=7774	;
272C	3AA8EA	=7775	LDA OPCODE ;
272F	E638	=7776	ANI DREGMASK ;
2731	0F	=7777	RRC ;
2732	0F	=7778	RRC ;
2733	0F	=7779	RRC ; PrintRegister((Opcod and
2734	CD4E28	=7780	CALL PRTREG ; DestinationRegisterMask) shr 3);
2737	C3E527	=7781	JMP POPR99 ;
		=7782	POPR10: ;
273A	FE03	=7783	CPI DATA8 ;
273C	C26327	=7784	JNZ POPR15 ; Data8 :
		=7785	Begin
273F	2AA9EA	=7786	LHLD DISPC ;
2742	23	=7787	INX H ;
2743	6E	=7788	MOV L,M ;
2744	1600	=7789	MVI D,0 ; Data := memory[PC+1];
2746	CD6628	=7790	CALL VALLOOK ; If lookupValue(Data) then
2749	CA5227	=7791	JZ POPR12 ;
		=7792	;
274C	CDF327	=7793	CALL PRINTL ; PrintLabel
274F	C36027	=7794	JMP POPR14 ; Else
		=7795	POPR12: ; Begin
2752	3E30	=7796	MVI A,'0' ;
2754	CDE218	=7797	CALL KITOUT ; KitOut('0');
2757	7D	=7798	MOV A,L ;
2758	CD041A	=7799	CALL PHEXB ; PrintHexByte(Low Data);
275B	3E48	=7800	MVI A,'H' ;
275D	CDE218	=7801	CALL KITOUT ; KitOut('H');
		=7802	POPR14: ; End;
2760	C3E527	=7803	JMP POPR99 ; End;
		=7804	POPR15: ;
2763	FE04	=7805	CPI DATA16 ; Data16 :
2765	C28B27	=7806	JNZ POPR20 ;
		=7807	Begin
2768	2AA9EA	=7808	LHLD DISPC ;
276B	23	=7809	INX H ;
276C	CD9414	=7810	CALL GETINDIRECT ; data := GetIndirect(PC+1);
276F	CD6628	=7811	CALL VALLOOK ; If LookupValue(data) then
2772	CA7B27	=7812	JZ POPR17 ;
		=7813	;
2775	CDF327	=7814	CALL PRINTL ; PrintLabel
2778	C38827	=7815	JMP POPR19 ;
		=7816	POPR17: ; Else
		=7817	Begin
277B	3E30	=7818	MVI A,'0' ;
277D	CDE218	=7819	CALL KITOUT ; KitOut('0');
2780	CD1D1A	=7820	CALL PHEXW ; PrintHexWord(data);
2783	3E48	=7821	MVI A,'H' ;
2785	CDE218	=7822	CALL KITOUT ; KitOut('H');
		=7823	POPR19: ; End;
2788	C3E527	=7824	JMP POPR99 ; End;
		=7825	POPR20: ;
278B	FE05	=7826	CPI RPARITH ; RegisterPairArithmetic :
278D	C2A227	=7827	JNZ POPR25 ;
		=7828	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 133

LOC	OBJ	LINE	SOURCE STATEMENT
2790	3AA8EA	=7829	LDA OPCODE ;
2793	E630	=7830	ANI RPAMASK ;
2795	OF	=7831	RRC ;
2796	OF	=7832	RRC ;
2797	OF	=7833	RRC ;
2798	OF	=7834	RRC ;
2799	21E627	=7835	LXI H,RPATAB ; PrintRegister(RegisterPairArith,
279C	CDFF26	=7836	CALL PNREG ; (Opcode and RPAMask) shr 4));
279F	C3E527	=7837	JMP POPR99 ;
		=7838	POPR25: ;
27A2	FE06	=7839	CPI RPLIMIT ; RegisterPairlimited :
27A4	C2B927	=7840	JNZ POPR30 ;
		=7841	;
27A7	3AA8EA	=7842	LDA OPCODE ;
27AA	E610	=7843	ANI RPLMASK ;
27AC	OF	=7844	RRC ;
27AD	OF	=7845	RRC ;
27AE	OF	=7846	RRC ;
27AF	OF	=7847	RRC ;
27B0	21E627	=7848	LXI H,RPLTAB ; PrintRegister(RegisterPairLimited,
27B3	CDFF26	=7849	CALL PNREG ; (Opcode and RPLMask) shr 4));
27B6	C3E527	=7850	JMP POPR99 ;
		=7851	POPR30: ;
27B9	FE07	=7852	CPI RPSTACK ; RegisterPairStack :
27BB	C2D027	=7853	JNZ POPR35 ;
		=7854	;
27BE	3AA8EA	=7855	LDA OPCODE ;
27C1	E630	=7856	ANI RPSMASK ;
27C3	OF	=7857	RRC ;
27C4	OF	=7858	RRC ;
27C5	OF	=7859	RRC ;
27C6	OF	=7860	RRC ;
27C7	21ED27	=7861	LXI H,RPSTAB ; PrintRegister(RegisterPairStack,
27CA	CDFF26	=7862	CALL PNREG ; (Opcode and RPSMask) shr 4));
27CD	C3E527	=7863	JMP POPR99 ;
		=7864	POPR35: ;
27D0	FE08	=7865	CPI RSTNUM ; RestartNumber :
27D2	C2E527	=7866	JNZ POPR40 ;
		=7867	;
27D5	3AA8EA	=7868	LDA OPCODE ;
27D8	E638	=7869	ANI RSTMASK ;
27DA	OF	=7870	RRC ;
27DB	OF	=7871	RRC ;
27DC	OF	=7872	RRC ;
27DD	C630	=7873	ADI '0' ; KitOut(((Opcode and RSTmask) shr 3) +
27DF	CDE218	=7874	CALL KITOUT ; '0'));
27E2	C3E527	=7875	JMP POPR99 ;
		=7876	POPR40: ;
		=7877	POPR99: ; End Case;
		=7878	;
27E5	C9	=7879	RET ; EndProc PrintOperand;
		=7880	;
		=7881	-----
		=7882	;
		=7883	Register Pair tables:
		=7884	;
27E6	C2	=7885	RPATAB: DB 'B' + MSB ; Register pair = 00B, Register = 'B'
27E7	C4	=7886	DB 'D' + MSB ; Register pair = 01B, Register = 'D'
27E8	C8	=7887	DB 'H' + MSB ; Register pair = 10B, Register = 'H'
27E9	53D0	=7888	DB 'S', 'P' + MSB ; Register pair = 11B, Register = 'SP'
		=7889	;
27EB	C2	=7890	RPLTAB: DB 'B' + MSB ; Register pair limited = 0, Register = 'B'
27EC	C4	=7891	DB 'D' + MSB ; Register pair limited = 1, Register = 'D'
		=7892	;
27ED	C2	=7893	RPSTAB: DB 'B' + MSB ; Register pair stack = 00B, Register = 'B'
27EE	C4	=7894	DB 'D' + MSB ; Register pair stack = 01B, Register = 'D'
27EF	C8	=7895	DB 'H' + MSB ; Register pair stack = 10B, Register = 'H'
27F0	5053D7	=7896	DB 'PS', 'W' + MSB ; Register pair stack = 11B, Register = 'PSW'
		=7897	;
		=7898	=====*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 134

LOC	OBJ	LINE	SOURCE STATEMENT
		=7899 ;*	
		=7900 ;*	Function : PRINTL (Print label - with space compression).
		=7901 ;*	Inputs : HL : Start address of the label to print.
		=7902 ;*	Outputs : None.
		=7903 ;*	Destroys : A and flags.
		=7904 ;*	Calls : KITOUT.
		=7905 ;*	
		=7906 ;*	Description : This routine outputs a label to the console. The
		=7907 ;*	difference between this routine and the other label output routine
		=7908 ;*	is that this one does space compression, i.e. only non space
		=7909 ;*	characters are printed.
		=7910 ;*	
		=7911 ;	=====
		=7912 ;	
		=7913 PRINTL:	; Procedure PrintLabel (LabelPtr:Integer);
27F3	E5	=7914	PUSH H ;
27F4	C5	=7915	PUSH B ;
27F5	0606	=7916	MVI B,LABLEN ;
		=7917	PL05: ; For I := 0 to LabelLength-1 do
		=7918	; Begin
27F7	7E	=7919	MOV A,M ;
27F8	FE20	=7920	CPI SPACE ; If Memory[LabelPtr+I] <> space then
27FA	C4E218	=7921	CNZ KITOUT ; KitOut(Memory[LabelPtr+I]);
27FD	23	=7922	INX H ;
27FE	05	=7923	DCR B ;
27FF	C2F727	=7924	JNZ PL05 ; End;
		=7925	;
2802	C1	=7926	POP B ;
2803	E1	=7927	POP H ;
2804	C9	=7928	RET ; EndProc PrintLabel;
		=7929 ;	
		=7930 ;	=====
		=7931 ;*	
		=7932 ;*	Function : PRTADR (Print address).
		=7933 ;*	Inputs : HL : Address to be printed.
		=7934 ;*	Outputs : None.
		=7935 ;*	Destroys : H, L, A and flags.
		=7936 ;*	Calls : PHEXW and SPACES.
		=7937 ;*	
		=7938 ;*	Description : This routine outputs the current program counter as
		=7939 ;*	four hexadecimal digits.
		=7940 ;*	
		=7941 ;	=====
		=7942 ;	
		=7943 PRTADR:	; Procedure PrintAddress(PC:Integer);
		=7944	;
2805	CD1D1A	=7945	CALL PHEXW ; PrintHexWord(PC);
2808	3E02	=7946	MVI A,2 ;
280A	CDA71A	=7947	CALL SPACES ; Spaces(2);
		=7948	;
280D	C9	=7949	RET ; EndProc PrintAddress;
		=7950 ;	
		=7951 ;	=====
		=7952 ;*	
		=7953 ;*	Function : PRTBYTES (Print bytes)
		=7954 ;*	Inputs : HL = Program count.
		=7955 ;*	A = Number of bytes to display.
		=7956 ;*	Outputs : None.
		=7957 ;*	Destroys : B, A and flags.
		=7958 ;*	Calls : PHEXB, KITOUT and TABOUT.
		=7959 ;*	
		=7960 ;*	Description : This routine outputs n bytes starting at the address
		=7961 ;*	passed in the HL register pair. The number of bytes to print is
		=7962 ;*	passed in the A register. The opcode is printed first and
		=7963 ;*	followed by a spaces. As an example a three byte instruction would
		=7964 ;*	be printed as follows:
		=7965 ;*	
		=7966 ;*	OO DDDD - Where OO = Opcode and DDDD = two data bytes.
		=7967 ;*	
		=7968 ;	=====

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 135

LOC	OBJ	LINE	SOURCE STATEMENT
		=7969 ;	
		=7970 PRTBYTES:	; Procedure PrintBytes (PC:Integer;
		=7971	Count:Byte)
280E	E5	=7972	PUSH H ;
280F	47	=7973	MOV B,A ;
2810	7E	=7974	MOV A,M ;
2811	CD041A	=7975	CALL PHEXB ; PrintHexByte(Memory[PC]);
2814	3E20	=7976	MVI A,SPACE ;
2816	CDE218	=7977	CALL KITOUT ; KitOutput(Space);
		=7978 PRTB05:	;
2819	05	=7979	DCR B ; Count := Count - 1;
		=7980	;
281A	CA2528	=7981	JZ PRTB10 ; While Count <> 0 do
		=7982	Begin ;
281D	23	=7983	INX H ; PC := PC + 1;
281E	7E	=7984	MOV A,M ;
281F	CD041A	=7985	CALL PHEXB ; PrintHexByte(Memory[PC]);
		=7986	Count := Count - 1
2822	C31928	=7987	JMP PRTB05 ; End;
		=7988 PRTB10:	;
2825	CDB31A	=7989	CALL TABOUT ; TabOutput;
2828	CDB31A	=7990	CALL TABOUT ; TabOutput;
282B	CDB31A	=7991	CALL TABOUT ; TabOutput;
282E	E1	=7992	POP H ;
282F	C9	=7993	RET ; EndProc PrintBytes;
		=7994 ;	
		=7995 ;=====*	
		=7996 ;*	
		=7997 ;* Function : PRTOPRS (Print operands). *	
		=7998 ;* Inputs : A : Operand byte : Lower nibble = First operand. *	
		=7999 ;* Upper nibble = Second operand. *	
		=8000 ;* Outputs : None. *	
		=8001 ;* Destroys : A, B and flags. *	
		=8002 ;* Calls : POPERAND and KITOUT. *	
		=8003 ;*	
		=8004 ;* Description : This routine outputs the operand field or fields of *	
		=8005 ;* an assembler instruction. If there are no operands the routine *	
		=8006 ;* returns without printing anything. If there is one operand then *	
		=8007 ;* it is printed. If there are two operands then the routine prints *	
		=8008 ;* them separated by a comma. *	
		=8009 ;*	
		=8010 ;=====*	
		=8011 ;	
		=8012 PRTOPRS:	; Procedure PrintOperands(OprByte:Byte);
2830	47	=8013	MOV B,A ;
2831	E60F	=8014	ANI FIRSTOPR ;
2833	CA4D28	=8015	JZ PRT020 ; If (OprByte AND FirstOperand) <> 0 then
		=8016	Begin ;
		=8017	;
2836	CD1727	=8018	CALL POPERAND ; PrintOperand(OprByte AND
		=8019	FirstOperand);
2839	78	=8020	MOV A,B ;
283A	E6F0	=8021	ANI SECOPR ;
283C	CA4D28	=8022	JZ PRT020 ; If (OprByte AND SecondOperand) <> 0
		=8023	Then begin ;
283F	F5	=8024	PUSH PSW ;
2840	3E2C	=8025	MVI A,COMMA ;
2842	CDE218	=8026	CALL KITOUT ; KitOutput(Comma);
2845	F1	=8027	POP PSW ;
2846	0F	=8028	RRC ;
2847	0F	=8029	RRC ;
2848	0F	=8030	RRC ;
2849	0F	=8031	RRC ;
284A	CD1727	=8032	CALL POPERAND ; PrintOperand((OprByte AND
		=8033	SecondOperand) SHR 4));
		=8034	;
		=8035	End;
		=8036 PRT020:	End;
		=8037	;
284D	C9	=8038	RET ; EndProc PrintOperands;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 136

LOC	OBJ	LINE	SOURCE STATEMENT
		=8039 ;	
		=8040 ;=====*	
		=8041 ;*	
		=8042 ;* Function : PRTREG (Print register).	
		=8043 ;* Inputs : A : Register number (0..7)	
		=8044 ;* Outputs : None.	
		=8045 ;* Destroys : A and flags.	
		=8046 ;* Calls : KITOUT.	
		=8047 ;*	
		=8048 ;* Description : This routine outputs the single character that	
		=8049 ;* corresponds to the register number.	
		=8050 ;*	
		=8051 ;=====*	
		=8052 ;	
		=8053 PRTREG:	; Procedure PrintNthRegister(RegNum:Byte);
284E	E5	=8054	PUSH H ;
284F	D5	=8055	PUSH D ;
2850	215E28	=8056	LXI H,RTABLE ;
2853	5F	=8057	MOV E,A ;
2854	1600	=8058	MVI D,0 ;
2856	19	=8059	DAD D ;
2857	7E	=8060	MOV A,M ;
2858	CDE218	=8061	CALL KITOUT ; KitOutput(Memory[RegisterTable+
		=8062	; RegNum]);
285B	D1	=8063	POP D ;
285C	E1	=8064	POP H ;
285D	C9	=8065	RET ; EndProc PrintNthRegister;
		=8066 ;	
		=8067 ;-----	
		=8068 ;	
		=8069 ; Register table:	
		=8070 ;	
		=8071 RTABLE:	; Start address of register table.
		=8072	;
285E	42	=8073	DB 'B' ; RegNum = 000B : Register B
285F	43	=8074	DB 'C' ; RegNum = 001B : Register C
2860	44	=8075	DB 'D' ; RegNum = 010B : Register D
2861	45	=8076	DB 'E' ; RegNum = 011B : Register E
2862	48	=8077	DB 'H' ; RegNum = 100B : Register H
2863	4C	=8078	DB 'L' ; RegNum = 101B : Register L
2864	4D	=8079	DB 'M' ; RegNum = 110B : Register M
2865	41	=8080	DB 'A' ; RegNum = 111B : Register A
		=8081 ;	
		=8082 ;=====*	
		=8083 ;*	
		=8084 ;* Function : VALLOOK (Value lookup).	
		=8085 ;* Inputs : HL : Value.	
		=8086 ;* Outputs : Z = 1 : Value not found in symbol table.	
		=8087 ;* Z = 0 : Value found in the symbol table.	
		=8088 ;* HL : pointer to label string.	
		=8089 ;* Destroys : A and flags.	
		=8090 ;* Calls : DCMP.	
		=8091 ;*	
		=8092 ;* Description : This routine searches thru the symbol table looking	
		=8093 ;* for a symbol whose value is equal to the value passed in the HL	
		=8094 ;* register pair.	
		=8095 ;*	
		=8096 ;* Note: If the symbolic disassembly flag <SYMBOLIC> is clear then	
		=8097 ;* this routine will always return non zero.	
		=8098 ;*	
		=8099 ;=====*	
		=8100 ;	
		=8101 VALLOOK:	; Procedure ValueLookup(Value:Integer);
		=8102	;
2866	3A7AEA	=8103	LDA SYMBOLIC ;
2869	FE00	=8104	CPI FALSE ;
286B	CAA228	=8105	JZ VALLEX ; If SymbolicDisassembly then begin
		=8106	;
286E	C5	=8107	PUSH B ;
286F	D5	=8108	PUSH D ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 137

LOC	OBJ	LINE	SOURCE STATEMENT
2870	E5	=8109	PUSH H ;
2871	EB	=8110	XCHG ;
2872	2100E0	=8111	LXI H,STABLE ; Ptr := SymbolTable;
2875	0600	=8112	MVI B,0 ;
		=8113	VALL05: ; For I := 1 to 256 do
		=8114	; Begin
2877	7E	=8115	MOV A,M ;
2878	B7	=8116	ORA A ;
2879	CA9528	=8117	JZ VALL10 ; If Memory[Ptr] <> 0 do
		=8118	; Begin
287C	E5	=8119	PUSH H ;
287D	D5	=8120	PUSH D ;
287E	110600	=8121	LXI D,LABLEN ;
2881	19	=8122	DAD D ;
		=8123	; TempPtr :=
		=8124	; Ptr+offset(value);
2882	5E	=8124	MOV E,M ;
2883	23	=8125	INX H ;
2884	56	=8126	MOV D,M ; SymValue :=
		=8127	; Memory[TempPtr] +
		=8128	; Memory[TempPtr+1] * 256;
2885	EB	=8129	XCHG ;
2886	D1	=8130	POP D ;
2887	CDBF1A	=8131	CALL DCMP ;
288A	E1	=8132	POP H ;
288B	C29528	=8133	JNZ VALL10 ; If SymValue = Value then
		=8134	; Begin
288E	E3	=8135	XTHL ; Return(Ptr);
288F	3E01	=8136	MVI A,1 ;
2891	B7	=8137	ORA A ;
2892	C39F28	=8138	JMP VALL99 ; ExitProcedure;
		=8139	; End;
		=8140	VALL10: ; End;
2895	C5	=8141	PUSH B ;
2896	010800	=8142	LXI B,LABLEN+WORD ;
2899	09	=8143	DAD B ; Ptr := Ptr +
		=8144	; LengthSymbolTableEntry;
289A	C1	=8145	POP B ;
289B	05	=8146	DCR B ;
289C	C27728	=8147	JNZ VALL05 ;
		=8148	; End;
		=8149	VALL99: ;
289F	E1	=8150	POP H ;
28A0	D1	=8151	POP D ;
28A1	C1	=8152	POP B ;
		=8153	VALLEX: ; End;
		=8154	;
28A2	C9	=8155	RET ; EndProcedure ValueLookup;
		=8156 ;	
		=8157 ;	END <DEBDIS.A85>
		8158	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 138

LOC	OBJ	LINE	SOURCE STATEMENT
		8159 ;	
		8160 ;	BYTE command routine:
		8161 ;	
		8162	\$INCLUDE(DEBBYTE.A85)
		=8163 ;	-----;
		=8164 ;	;
		=8165 ;	BYTE command routines used by the UCT SABus debugger. ;
		=8166 ;	;
		=8167 ;	This module contains (in order) the following subroutines: ;
		=8168 ;	;
		=8169 ;	a) BYTEDUMP - Display routine. ;
		=8170 ;	b) BYTEROU - BYTE command routine. ;
		=8171 ;	c) FILL - Write routine. ;
		=8172 ;	d) PRTABLE - Convert to printable character routine. ;
		=8173 ;	;
		=8174 ;	-----;
		=8175	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
=8176			;=====*
=8177			;*
=8178			;* Function : BYTEDUMP (Byte display routine).*
=8179			;* Inputs : (RBEGIN) : Range start address.*
=8180			;* (REND) : Range end address.*
=8181			;* Outputs : None.*
=8182			;* Destroys : A, B, C, D, E, H, L and flags.*
=8183			;* Calls : DCMP, PHEXW, PHEXB, OUTMSG, PRTABLE, SPACES and CRLF.*
=8184			;*
=8185			;* Description : This routine display a range of bytes in the following*
=8186			;* format.*
=8187			;*
=8188			;* 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF*
=8189			;*
=8190			;* NNN0 BB BB BB BB BB BB BB BB BB BB BB BB BB BB
=8191			;* NNN0 BB BB BB BB BB BB
=8192			;*
=8193			;* Where: NNN0 : Start address of a 16 byte range.*
=8194			;* BB : Two hexadecimal digits.*
=8195			;* . : Printable ASCII characters.*
=8196			;*
=8197			;=====*
=8198			;*
=8199			; Variables: HL = Ptr
=8200			; DE = EndValue
=8201			;*
=8202			BYTEDUMP: ; Procedure ByteDump;
=8203			;*
28A3	2A7DEA	=8204	LHLD REND ;
28A6	7D	=8205	MOV A,L ;
28A7	E6F0	=8206	ANI 11110000B ;
28A9	6F	=8207	MOV L,A ; EndValue := (RangeEnd AND OFFFOH) + 16;
28AA	111000	=8208	LXI D,00010000B ;
28AD	19	=8209	DAD D ;
28AE	EB	=8210	XCHG ;
28AF	2A7BEA	=8211	LHLD RBEGIN ;
28B2	7D	=8212	MOV A,L ;
28B3	E6F0	=8213	ANI 11110000B ;
28B5	6F	=8214	MOV L,A ; Ptr := (RangeBegin AND OFFFOH);
		=8215	BYTD05: ;
28B6	017FEA	=8216	LXI B,ASCBUF ; ASCIIptr := ASCIIBuffer;
28B9	CDBF1A	=8217	CALL DCMP ;
28BC	CA6229	=8218	JZ BYTR99 ; While Ptr <= EndValue do
		=8219	; Begin
		=8220	;*
28BF	CD1D1A	=8221	CALL PHEXW ; PrintHexWord(Ptr);
28C2	D5	=8222	PUSH D ;
		=8223	;*
28C3	3E02	=8224	MVI A,2 ;
28C5	CDA71A	=8225	CALL SPACES ; Spaces(2);
		=8226	;*
		=8227	BYTD10: ; Repeat
28C8	EB	=8228	XCHG ;
28C9	2A7BEA	=8229	LHLD RBEGIN ;
28CC	EB	=8230	XCHG ;
28CD	CDBF1A	=8231	CALL DCMP ;
28D0	DADE28	=8232	JC BYTD20 ; If (Ptr < RangeBegin) or
		=8233	;*
28D3	EB	=8234	XCHG ;
28D4	2A7DEA	=8235	LHLD REND ;
28D7	CDBF1A	=8236	CALL DCMP ;
28DA	EB	=8237	XCHG ;
28DB	D2E928	=8238	JNC BYTD30 ; (Ptr > RangeEnd) then
		=8239	BYTD20: ;
		=8240	;*
28DE	3E20	=8241	MVI A,SPACE ; Begin
28E0	02	=8242	STAX B ;
		=8243	; Memory[ASCIIPtr] :=
			Space;
28E1	3E02	=8244	MVI A,2 ;
28E3	CDA71A	=8245	CALL SPACES ; Spaces(2);

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 140

LOC	OBJ	LINE	SOURCE STATEMENT
28E6	C3F228	=8246	JMP BYTD35 ; End
		=8247	BYTD30: ; Else
		=8248	; Begin
28E9	7E	=8249	MOV A,M ;
28EA	CDAE29	=8250	CALL PRTABLE ;
28ED	02	=8251	STAX B ; Memory[ASCIIPtr] :=
		=8252	; Printable(Memory[Ptr]);
28EE	7E	=8253	MOV A,M ;
28EF	CD041A	=8254	CALL PHEXB ; PrintHexByte(Memory[Ptr]);
		=8255	BYTD35: ; End;
		=8256	;
28F2	23	=8257	INX H ; Ptr := Ptr + 1;
		=8258	;
28F3	03	=8259	INX B ; ASCIIptr := ASCIIptr + 1;
		=8260	;
28F4	7D	=8261	MOV A,L ;
28F5	E603	=8262	ANI 11B ; If Ptr mod 4 = 0 then
28F7	3E02	=8263	MVI A,2 ;
28F9	CAFE28	=8264	JZ BYTD40 ; Spaces(2)
28FC	3E01	=8265	MVI A,1 ;
		=8266	BYTD40: ; Else
		=8267	;
28FE	CDA71A	=8268	CALL SPACES ; Spaces(1);
		=8269	;
2901	7D	=8270	MOV A,L ;
2902	E60F	=8271	ANI 00001111B ;
2904	C2C828	=8272	JNZ BYTD10 ; Until (Ptr mod 16) = 0;
		=8273	;
2907	3E00	=8274	MVI A,0 ;
2909	328FEA	=8275	STA ENDMARK ; EndMark = 0;
		=8276	;
290C	E5	=8277	PUSH H ;
290D	217FEA	=8278	LXI H,ASCBUF ;
2910	CDEE18	=8279	CALL OUTMSG ; OutMessage(ASCIIBuffer);
2913	E1	=8280	POP H ;
2914	CDB419	=8281	CALL CRLF ; NewLine;
		=8282	;
2917	D1	=8283	POP D ;
2918	C3B628	=8284	JMP BYTD05 ;
		=8285	BYTD99: ; End;
		=8286	;
291B	C9	=8287	RET ; EndProc ByteDump;
		=8288	;
		=8289	=====*
		=8290	;*
		=8291	;* Function : BYTEROU (BYTE command routine). *
		=8292	;* Inputs : None. *
		=8293	;* Outputs : None. *
		=8294	;* Destroys : A, B, C, D, E, H, L and flags. *
		=8295	;* Calls : GETSYM, CMDERR, GETRANGE, ERROR, BYTEDUMP and FILL. *
		=8296	;*
		=8297	;* Description : This routine handles the BYTE command. This command *
		=8298	;* allows the user to display or modify the contents of memory. *
		=8299	;* The BNF syntax of the BYTE command is as follows: *
		=8300	;*
		=8301	;* Syntax: BYTE <Number> [LENGTH TO <Number>] [=.<Number>] <EOL> *
		=8302	;*
		=8303	;* Some examples of the BYTE command and their meanings are given *
		=8304	;* below: *
		=8305	;*
		=8306	;* BYTE 0 - Display the value of the byte at address 0 *
		=8307	;* BYTE 0 TO OFFH - Display the values of the bytes in the *
		=8308	;* range 0 to FF hexadecimal. *
		=8309	;* BYTE 0 = 10 - Write 10 to location 0. *
		=8310	;* BYTE 0 LEN 10 = 0 - Write 0 to the range 0 to 10. *
		=8311	;*
		=8312	;* Note: a) When writes are performed the protection map is checked. *
		=8313	;* b) When writes are performed no verification is done. *
		=8314	;* c) Displays are done in both hexadecimal and ASCII. *
		=8315	;*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 141

LOC	OBJ	LINE	SOURCE STATEMENT
		=8316 ;	=====*
		=8317 ;	
		=8318 BYTEROU:	; Procedure ByteRoutine;
		=8319	;
291C	CDFF1B	=8320	CALL GETSYM ; Sym := GetNextSymbol;
291F	FE02	=8321	CPI NUMBER ;
2921	C25C29	=8322	JNZ BYTR80 ; If Sym = Number then
		=8323	;
2924	CD821F	=8324	CALL GETRANGE ;
2927	CA3329	=8325	JZ BYTR10 ; If GetRangeError then
		=8326	;
292A	216329	=8327	LXI H,RGEERR ;
292D	CDC119	=8328	CALL ERROR ; Error(RangeError)
2930	C36229	=8329	JMP BYTR99 ;
		=8330 BYTR10:	; Else
		=8331	Begin
2933	3A02EA	=8332	LDA SYMTYPE ;
2936	FE00	=8333	CPI EOL ;
2938	C24129	=8334	JNZ BYTR15 ; If Sym = EndOfLine then
		=8335	;
293B	CDA328	=8336	CALL BYTEDUMP ; DumpBytes
293E	C36229	=8337	JMP BYTR99 ;
		=8338 BYTR15:	; Else
2941	FE04	=8339	CPI EQUALS ;
2943	C25C29	=8340	JNZ BYTR80 ; If Sym = EQUALS then begin
		=8341	;
2946	CDFF1B	=8342	CALL GETSYM ; Sym := GetNextSymbol;
2949	FE02	=8343	CPI NUMBER ;
294B	C25C29	=8344	JNZ BYTR80 ; If Sym = NUMBER then begin
		=8345	;
294E	CDFF1B	=8346	CALL GETSYM ; Sym = GetNextSymbol;
2951	FE00	=8347	CPI EOL ;
2953	C25C29	=8348	JNZ BYTR80 ; if Sym = EOL then
		=8349	;
2956	CD8B29	=8350	CALL FILL ; Fill;
2959	C36229	=8351	JMP BYTR99 ;
		=8352	;
		=8353 BYTR80:	; Else
295C	217529	=8354	LXI H,ERMBYTE ;
295F	CD8119	=8355	CALL CMDERR ; CommandError(ByteError);
		=8356 BYTR99:	;
2962	C9	=8357	RET ; EndProc ByteRoutine;
		=8358 ;	
		=8359 RGEERR:	
		=8360 ;	
		=8361 ; format:	
		=8362 ;	
		=8363 ; Range specification error	
		=8364 ;	
		=8365 ; declaration:	
		=8366 ;	
2963	52616E67	=8367	DB 'Rang', 'e'+MSB, 'specifica', PSTION, ' ', PSERROR, EndMsg
	E5737065		
	63696669		
	63618320		
	8500		
		=8368 ;	
		=8369 ERMBYTE:	
		=8370 ;	
		=8371 ; format:	
		=8372 ;	
		=8373 ; BYTE <Number> [LENGTH TO <Number>] [= <Number>]	
		=8374 ;	
		=8375 ; declaration:	
		=8376 ;	
2975	425954C5	=8377	DB 'BYT', 'E'+MSB, PSNUM, '['+MSB, 'LENGTH T', 'O'+MSB, PSNUM
	80DB4C45		
	4E475448		
	7C54CF80		
2985	DDDBBD80	=8378	DB '['+MSB, '['+MSB, '='+MSB, PSNUM, ']', EndMsg

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 142

LOC	OBJ	LINE	SOURCE STATEMENT
	5D00	=8379 ;	
		=8380 ;=====*	
		=8381 ;*	
		=8382 ;* Function : FILL (Fill range of memory with a given value)	
		=8383 ;* Inputs : (RBEGIN) : Start Address of the range to fill.	
		=8384 ;* (REND) : End Address of the range.	
		=8385 ;* (SYMVALUE) : Value to write to memory.	
		=8386 ;* Outputs : None.	
		=8387 ;* Destroys : A, B, D, E, H, L and flags.	
		=8388 ;* Calls : DCMF and CHKMWR.	
		=8389 ;*	
		=8390 ;* Description : This routine fill the ranges of memory given by the	
		=8391 ;* memory variables RBEGIN and REND with the value in the another	
		=8392 ;* variable SYMVALUE. The routine also checks that writes are allowed	
		=8393 ;* to the memory address being written to. If a protection error is	
		=8394 ;* detected then the writing process stops and an error message is	
		=8395 ;* printed.	
		=8396 ;*	
		=8397 ;=====*	
		=8398 ;	
		=8399 FILL: ; Procedure Fill;	
298B	3A00EA	=8400 LDA SYMVALUE ;	
298E	47	=8401 MOV B,A ;	
298F	2A7DEA	=8402 LHLD REND ;	
2992	EB	=8403 XCHG ;	
2993	2A7BEA	=8404 LHLD RBEGIN ; Ptr := RangeBegin;	
		=8405 ;	
		=8406 FILL05: ; Repeat	
2996	CD6610	=8407 CALL CHKMWR ;	
2999	CAA729	=8408 JZ FILL80 ; If CheckMemoryWrite(Ptr) then	
		=8409 ;	
299C	70	=8410 MOV M,B ; Memory[Ptr] := SymbolValue;	
299D	CDBF1A	=8411 CALL DCMF ;	
29A0	D2AD29	=8412 JNC FILL99 ;	
		=8413 ;	
29A3	23	=8414 INX H ;	
29A4	C39629	=8415 JMP FILL05 ; Until (NOT CheckMemoryWrite(Ptr)) or	
		=8416 ; (Ptr >= RangeEnd);	
		=8417 FILL80: ;	
		=8418 ; If NOT CheckMemoryWrite(ptr)	
29A7	21352C	=8419 LXI H,PROERR ;	
29AA	CDC119	=8420 CALL ERROR ; Error(ProtectionError);	
		=8421 FILL99: ;	
29AD	C9	=8422 RET ; EndProc Fill;	
		=8423 ;	
		=8424 ;=====*	
		=8425 ;*	
		=8426 ;* Function : PRTABLE (Printable)	
		=8427 ;* Inputs : A : Eight bit value.	
		=8428 ;* Outputs : A : Printable ASCII character.	
		=8429 ;* Destroys : A and flags.	
		=8430 ;* Calls : None.	
		=8431 ;*	
		=8432 ;* Description : This routine converts any eight bit byte into a	
		=8433 ;* printable ASCII character. If the input character is not printable	
		=8434 ;* then an ASCII dot is returned.	
		=8435 ;*	
		=8436 ;=====*	
		=8437 ;	
		=8438 PRTABLE: ; Function Printable (Char) : Char;	
		=8439 ;	
29AE	E67F	=8440 ANI ASCII ; Ch := Ch and ASCII;	
29B0	FE7F	=8441 CPI DEL ;	
29B2	CABA29	=8442 JZ PRTA10 ; If (Ch = DEL) OR	
		=8443 ;	
29B5	FE20	=8444 CPI SPACE ;	
29B7	F2BC29	=8445 JP PRTA20 ; (Ch < Space) then	
		=8446 PRTA10: ;	
29BA	3E2E	=8447 MVI A,'.' ; Ch := '.';	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 143

LOC	OBJ	LINE	SOURCE STATEMENT
		=8448	PRTA20: ;
		=8449	; Return(Ch)
		=8450	;
29BC	C9	=8451	RET ; EndProc Printable;
		=8452	;
		=8453	; END <DEBBYTE.A85>
		=8454	
		=8455	
		8456	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 144

LOC	OBJ	LINE	SOURCE STATEMENT
		8457 ;	
		8458 ;	DEFINE command routine:
		8459 ;	
		8460	\$INCLUDE(DEBDEF.A85)
		=8461 ;	-----;
		=8462 ;	;
		=8463 ;	Definition module for the UCT SABus debugger. ;
		=8464 ;	;
		=8465 ;	This module contains (in order) the following subroutines. ;
		=8466 ;	;
		=8467 ;	a) DEFROU - DEF command routine. ;
		=8468 ;	b) INSERT - Insert a definition into the symbol table. ;
		=8469 ;	c) LABPRT - Label print routine. ;
		=8470 ;	d) STDUMP - Symbol table dump. ;
		=8471 ;	;
		=8472 ;	-----;
		=8473	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 145

LOC	OBJ	LINE	SOURCE STATEMENT
		=8474 ;	=====*
		=8475 ;*	*
		=8476 ;*	Function : DEFROU (Define routine). *
		=8477 ;*	Inputs : None. *
		=8478 ;*	Outputs : None. *
		=8479 ;*	Destroys : A, B, C, D, E, H, L and Flags. *
		=8480 ;*	Calls : GETSYM, INSERT, CRLF, STDUMP and CMDERR. *
		=8481 ;*	*
		=8482 ;*	Description : This routine handles the 'DEF' command. This command *
		=8483 ;*	allows a symbol to be defined or a symbol table dump to be *
		=8484 ;*	displayed. The BNF syntax for the DEF command is given below: *
		=8485 ;*	*
		=8486 ;*	Syntax: DEF [<String> = <Number>] <EOL> *
		=8487 ;*	*
		=8488 ;*	Some examples of the DEF command and their meanings: *
		=8489 ;*	*
		=8490 ;*	DEF - Display the values of all the symbols in the *
		=8491 ;*	symbol table. (i.e. A symbol table dump). *
		=8492 ;*	*
		=8493 ;*	DEF FRED = 1234H - Assign the value 1234H to the symbol FRED. *
		=8494 ;*	*
		=8495 ;*	Note: When symbols are added to the symbol table, they are simply *
		=8496 ;*	placed in the first available slot. (THEY ARE NOT HASHED). *
		=8497 ;*	*
		=8498 ;	=====*
		=8499 ;	
		=8500 DEFROU:	; Procedure DefineRoutine;
		=8501	;
29BD	CDFF1B	=8502 CALL GETSYM	; Sym := GetNextSymbol;
29C0	FE00	=8503 CPI EOL	;
29C2	C2CE29	=8504 JNZ DEFR05	; If Sym = EOL then
		=8505	Begin
29C5	CD582A	=8506 CALL STDUMP	; SymbolTableDump;
29C8	CDB419	=8507 CALL CRLF	; NewLine;
29CB	C3F729	=8508 JMP DEFR99	; End
		=8509 DEFR05:	; Else
29CE	FE03	=8510 CPI STRING	;
29D0	C2F129	=8511 JNZ DEFR90	; If Sym = STRING then begin
		=8512	;
29D3	CDFF1B	=8513 CALL GETSYM	; Sym := GetNextSymbol;
29D6	FE04	=8514 CPI EQUALS	;
29D8	C2F129	=8515 JNZ DEFR90	; If Sym = EQUALS then begin
		=8516	;
29DB	CDFF1B	=8517 CALL GETSYM	; Sym := GetNextSymbol;
29DE	FE02	=8518 CPI NUMBER	;
29E0	C2F129	=8519 JNZ DEFR90	; If Sym = NUMBER then begin
		=8520	;
29E3	CDFF1B	=8521 CALL GETSYM	; Sym := GetNextSymbol;
29E6	FE00	=8522 CPI EOL	;
29E8	C2F129	=8523 JNZ DEFR90	; If Sym = EOL then
		=8524	;
29EB	CD082A	=8525 CALL INSERT	; Insert;
29EE	C3F729	=8526 JMP DEFR99	; End
		=8527	; End
		=8528	; End
		=8529 DEFR90:	; Else
29F1	21F829	=8530 LXI H,ERMDEF	;
29F4	CD8119	=8531 CALL CMDERR	; CommandError(DefineError);
		=8532 DEFR99:	; End;
		=8533	;
29F7	C9	=8534 RET	; EndProc DefineRoutine;
		=8535 ;	
		=8536 ERMDEF:	
		=8537 ;	
		=8538 ; format:	
		=8539 ;	
		=8540 ; DEF [<String> = <Number>]	
		=8541 ;	
		=8542 ; declaration:	
		=8543 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 146

LOC	OBJ	LINE	SOURCE STATEMENT
29F8	4445C6DB	=8544	DB 'DE', 'F'+MSB, '['+MSB, '<String', '>'+MSB
	3C537472		
	696E67BE		
2A04	BD805D00	=8545	DB '='+MSB, PSNUM, ']', EndMsg
		=8546 ;	
		=8547 ;=====*	
		=8548 ;*	
		=8549 ;*	Function : INSERT (Insert a definition into the symbol table). *
		=8550 ;*	Inputs : (LABBUF) : Symbol to be defined. *
		=8551 ;*	(SYMVAL) : Value of the symbol to be defined. *
		=8552 ;*	Outputs : None. *
		=8553 ;*	Destroys : A, B, D, E, H, I and Flags. *
		=8554 ;*	Calls : ERROR *
		=8555 ;*	
		=8556 ;*	Description : This routine is used to add a symbol definition to *
		=8557 ;*	the symbol table. The two variables, the string and the value, are *
		=8558 ;*	stored in the following two memory variables, LABBUF and SYMVAL. *
		=8559 ;*	
		=8560 ;*	It should be noted that although the symbol table was generated as *
		=8561 ;*	hash table. The new symbol is simply inserted into the table at *
		=8562 ;*	the first free slot in the table. To determine if a slot is free *
		=8563 ;*	or not the first byte of the entry is checked. If it is zero then *
		=8564 ;*	the 8 byte entry that following is free. *
		=8565 ;*	
		=8566 ;*	If no free slot can be found then the symbol table is full and an *
		=8567 ;*	error message to this effect is printed. *
		=8568 ;*	
		=8569 ;=====*	
		=8570 ;	
		=8571	INSERT: ; Procedure Insert;
2A08	2100E0	=8572	LXI H,STABLE ;
2A0B	110800	=8573	LXI D,LABLEN+WORD ;
2A0E	0600	=8574	MVI B,0 ; i := LengthSymbolTable;
		=8575	; ;
		=8576	INS05: ; While (Table[i].String[1] = 0) AND
		=8577	; (I <> 0) do
2A10	7E	=8578	MOV A,M ;
2A11	B7	=8579	ORA A ;
2A12	CA232A	=8580	JZ INS20 ;
		=8581	; ;
2A15	19	=8582	DAD D ;
2A16	05	=8583	DCR B ; I := I - 1;
2A17	C2102A	=8584	JNZ INS05 ;
		=8585	; If I = 0 then
		=8586	; ;
2A1A	213A2A	=8587	LXI H,INSERR ;
2A1D	CDC119	=8588	CALL ERROR ; ErrorMessage('Symbol table full');
2A20	C3392A	=8589	JMP INS99 ;
		=8590	INS20: ; Else
		=8591	; Begin
2A23	0606	=8592	MVI B,LABLEN ;
2A25	110BEA	=8593	LXI D,LABBUF ;
		=8594	INS25: ; For I := J to LabelLength do
2A28	1A	=8595	LDAX D ;
2A29	77	=8596	MOV M,A ; Table[i].String[j] :=
		=8597	; LabelBuffer[j];
2A2A	23	=8598	INX H ;
2A2B	13	=8599	INX D ;
2A2C	05	=8600	DCR B ;
2A2D	C2282A	=8601	JNZ INS25 ;
		=8602	; ;
2A30	3A00EA	=8603	LDA SYMVAL ;
2A33	77	=8604	MOV M,A ;
2A34	23	=8605	INX H ;
2A35	3A01EA	=8606	LDA SYMVAL+1 ;
2A38	77	=8607	MOV M,A ; Table[i].Value := SymbolValue;
		=8608	; ;
		=8609	INS99: ; End;
2A39	C9	=8610	RET ; EndProc Insert;
		=8611 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 147

LOC	OBJ	LINE	SOURCE STATEMENT
		=8612	INSERR:
		=8613	;
		=8614	; format:
		=8615	;
		=8616	Symbol table full
		=8617	;
		=8618	declaration:
		=8619	;
2A3A	53796D62 6FEC7461 626CE566 756C6C00	=8620	DB 'Symbo', 'l'+MSB, 'tabl', 'e'+MSB, 'full', EndMsg
		=8621	;
		=8622	=====*
		=8623	;
		=8624	* Function : LABPRT (Label Print). *
		=8625	* Inputs : HL : Pointer to six byte string. *
		=8626	* Outputs : HL : Points to the byte following the string. *
		=8627	* Destroys : H, L, A and Flags. *
		=8628	* Calls : KITOUT. *
		=8629	;
		=8630	* Description : This routine prints the contents of the six character *
		=8631	* string buffer pointed to by the HL register pair. Characters are *
		=8632	* output via the KITOUT routine. *
		=8633	;
		=8634	=====*
		=8635	;
		=8636	LABPRT: ; Procedure LabelPrint(LabelPointer:Integer);
2A4A	C5	=8637	PUSH B ;
2A4B	0606	=8638	MVI B, LABLEN ;
		=8639	LABP05: ; For I := 1 to LabelLength Do
2A4D	7E	=8640	MOV A, M ;
2A4E	CDE218	=8641	CALL KITOUT ; KitOut(Memory[LabelPointer+I-1]);
2A51	23	=8642	INX H ;
2A52	05	=8643	DCR B ;
2A53	C24D2A	=8644	JNZ LABP05 ;
		=8645	;
2A56	C1	=8646	POP B ;
2A57	C9	=8647	RET ; EndProc LabelPrint;
		=8648	;
		=8649	=====*
		=8650	;
		=8651	* Function : STDUMP (Symbol Table Dump). *
		=8652	* Inputs : None. *
		=8653	* Outputs : None. *
		=8654	* Destroys : A, B, C, D, E, H, L and Flags. *
		=8655	* Calls : LABPRT, SPACES, PHEXW and CRLF. *
		=8656	;
		=8657	* Description : This routine prints every entry in the symbol table. *
		=8658	* The entries are printed in the following format; *
		=8659	;
		=8660	* CCCCCC HHHH - Where CCCCCC = Six character string. *
		=8661	* HHHH = 16 bit hexadecimal number. *
		=8662	;
		=8663	* Six such entries are printed per line. The routine simply checks *
		=8664	* every entry in the symbol table to see if it is in use. It does *
		=8665	* this by checking the first byte in the eight byte entry. If the *
		=8666	* first byte is zero then the entry is not in use. If the byte *
		=8667	* is not zero then the entry is in use and it's string and value *
		=8668	* are then printed in the format given above. *
		=8669	;
		=8670	=====*
		=8671	;
		=8672	STDUMP: ; Procedure SymbolTableDump;
		=8673	;
2A58	2100E0	=8674	LXI H, STABLE ;
2A5B	110800	=8675	LXI D, LABLEN+WORD ;
2A5E	0600	=8676	MVI B, 0 ;
2A60	0E06	=8677	MVI C, 6 ; SymbolsLeftOnLine := 6;
		=8678	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 148

LOC	OBJ	LINE	SOURCE STATEMENT
		=8679	STD05:
		=8680	
2A62	7E	=8681	MOV A,M
2A63	B7	=8682	ORA A
2A64	CA882A	=8683	JZ STD40
		=8684	
2A67	E5	=8685	PUSH H
2A68	D5	=8686	PUSH D
2A69	CD4A2A	=8687	CALL LABPRT
2A6C	3E01	=8688	MVI A,1
2A6E	CDA71A	=8689	CALL SPACES
2A71	5E	=8690	MOV E,M
2A72	23	=8691	INX H
2A73	56	=8692	MOV D,M
2A74	EB	=8693	XCHG
2A75	CD1D1A	=8694	CALL PHEXW
2A78	3E02	=8695	MVI A,2
2A7A	CDA71A	=8696	CALL SPACES
2A7D	D1	=8697	POP D
2A7E	E1	=8698	POP H
		=8699	
2A7F	OD	=8700	DCR C
		=8701	
		=8702	
2A80	C2882A	=8703	JNZ STD40
		=8704	
2A83	CDB419	=8705	CALL CRLF
2A86	OE06	=8706	MVI C,6
		=8707	
		=8708	STD40:
		=8709	
2A88	19	=8710	DAD D
2A89	05	=8711	DCR B
2A8A	C2622A	=8712	JNZ STD05
		=8713	
2A8D	C9	=8714	RET
		=8715	
		=8716	END <DEBDEF.A85>
		8717	\$EJECT

```

; For I := 0 to SymbolTableLength do
;   Begin
;       If Table[i].String[1] <> 0 then
;         Begin
;             PrintLabel(Table[i].string);
;             Spaces(1);
;             PrintHexWord(Table[i].value);
;             Spaces(2);
;         End
;       End
;   End
; SymbolsLeftOnLine :=
;   SymbolsLeftOnLine - 1;
; If SymbolsLeftOnLine = 0 then
;   Begin
;       NewLine;
;       SymbolsLeftOnLine := 6;
;   End
; End;
; EndProc SymbolTableDump;

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 149

LOC	OBJ	LINE	SOURCE STATEMENT
		8718 ;	
		8719 ;	GO command routine:
		8720 ;	
		8721 \$INCLUDE(DEBGO.A85)	
		=8722 ;	-----;
		=8723 ;	;
		=8724 ;	GO command routines used by the UCT SABus debugger. ;
		=8725 ;	;
		=8726 ;	This module contains (in order) the following subroutines. ;
		=8727 ;	;
		=8728 ;	a) GO - Routine that runs the program. ;
		=8729 ;	b) GOROU - GO command routine. ;
		=8730 ;	c) TRIG? - Trigger condition ? ;
		=8731 ;	;
		=8732 ;	-----;
		=8733 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 150

LOC	OBJ	LINE	SOURCE STATEMENT
		=8734 ;	=====*
		=8735 ;*	*
		=8736 ;*	Function : GO (Start the program). *
		=8737 ;*	Inputs : None. *
		=8738 ;*	Outputs : None. *
		=8739 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=8740 ;*	Calls : SSINIT, SSTEP and PSTATUS. *
		=8741 ;*	*
		=8742 ;*	Description : This routine runs a program by calling the single *
		=8743 ;*	step routine until something causes the program to be halted. Once *
		=8744 ;*	the program is stopped (STATUS<>RUNNING) the cause of the halt is *
		=8745 ;*	displayed by the print status routine. *
		=8746 ;*	*
		=8747 ;	=====*
		=8748 ;	
		=8749 GO:	; Procedure GO;
		=8750	;
2A8E	CD6122	=8751	CALL SSINIT ; SingleStepInitialise;
2A91	3E00	=8752	MVI A,RUNNING ;
2A93	32B2EA	=8753	STA STATUS ; Status := RUNNING;
		=8754	;
		=8755 GOR05:	; Repeat
		=8756	;
2A96	CD7222	=8757	CALL SSTEP ; SingleStep;
2A99	3AB2EA	=8758	LDA STATUS ;
2A9C	FE00	=8759	CPI RUNNING ;
2A9E	CA962A	=8760	JZ GOR05 ; Until (Status <> Running)
		=8761	;
2AA1	CD261A	=8762	CALL PSTATUS ; PrintStatus;
		=8763	;
2AA4	C9	=8764	RET ; EndProc GO;
		=8765 ;	
		=8766 ;	=====*
		=8767 ;*	*
		=8768 ;*	Function : GOROU (GO command routine) *
		=8769 ;*	Inputs : None. *
		=8770 ;*	Outputs : None. *
		=8771 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=8772 ;*	Calls : GETSYM, TRIG?, CMDERR and GO. *
		=8773 ;*	*
		=8774 ;*	Description : This routine handles the GO command. This command *
		=8775 ;*	allows the user to start his program from a particular point *
		=8776 ;*	and specify a termination condition. The BNF syntax of the GO *
		=8777 ;*	Command is as follows. *
		=8778 ;*	*
		=8779 ;*	Syntax: GO [FROM <Number>] [FOREVER TILL <Number> <Cond>] <EOL> *
		=8780 ;*	*
		=8781 ;*	Where <Cond> is one of the following termination conditions: *
		=8782 ;*	*
		=8783 ;*	<Cond> = EXECUTED INPUT OUTPUT READ WRITE *
		=8784 ;*	*
		=8785 ;*	Some sample GO commands and their meanings are given below: *
		=8786 ;*	*
		=8787 ;*	GO - Continue from the current value of the *
		=8788 ;*	program counter. The termination *
		=8789 ;*	condition is unchanged. *
		=8790 ;*	GO FROM 0 FOREVER - Start program from location 0. The *
		=8791 ;*	previous trigger condition is cancelled. *
		=8792 ;*	GO TILL 0 INPUT - Continue the program from the current *
		=8793 ;*	value of the program counter. The *
		=8794 ;*	termination condition is set to an input *
		=8795 ;*	from I/O port 0. *
		=8796 ;*	GO FROM 0 TILL 10 READ - Start the program from location 0. The *
		=8797 ;*	trigger condition is set to a read from *
		=8798 ;*	memory location 10. *
		=8799 ;*	*
		=8800 ;*	Note: If the FROM <Number> clause is present then the trace buffer *
		=8801 ;*	is initialised. *
		=8802 ;*	*
		=8803 ;	=====*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 151

LOC	OBJ	LINE	SOURCE	STATEMENT
		=8804		;
		=8805	GOROU:	; Procedure GoRoutine;
		=8806		;
2AA5	CDFF1B	=8807	CALL	GETSYM ; Sym := GetNextSymbol;
2AA8	FE14	=8808	CPI	SYMFROM ;
2AAA	C2C72A	=8809	JNZ	GORO10 ; If Sym = SymFROM then begin
		=8810		;
2AAD	CDFF1B	=8811	CALL	GETSYM ; Sym := GetNextSymbol;
2AB0	FE02	=8812	CPI	NUMBER ;
2AB2	C2C42A	=8813	JNZ	GORO05 ; If Sym = Number then begin
		=8814		;
2AB5	2A00EA	=8815	LHLD	SYMVAL ;
2AB8	229DEA	=8816	SHLD	REGPC ; ProgramCounter := SymValue;
2ABB	CD352F	=8817	CALL	TBINIT ; TraceBufferInitialise;
2ABE	CDFF1B	=8818	CALL	GETSYM ; Sym := GetNextSymbol;
2AC1	C3C72A	=8819	JMP	GORO10 ;
		=8820	GORO05:	End Else
2AC4	C3162B	=8821	JMP	GORO80 ; ExitWithError;
		=8822		End;
		=8823	GORO10:	;
2AC7	FE2D	=8824	CPI	SYMTILL ;
2AC9	C2FE2A	=8825	JNZ	GORO20 ; If Sym = SymTILL then begin
		=8826		;
2ACC	CDFF1B	=8827	CALL	GETSYM ; Sym := GetNextSymbol;
2ACF	FE02	=8828	CPI	NUMBER ;
2AD1	C2FB2A	=8829	JNZ	GORO15 ; If Sym = Number then begin
		=8830		;
2AD4	2A00EA	=8831	LHLD	SYMVAL ;
2AD7	22B4EA	=8832	SHLD	TRIGWORD ; TrigWord := SymValue;
		=8833		;
2ADA	CDFF1B	=8834	CALL	GETSYM ; Sym := GetNextSymbol;
2ADD	FE00	=8835	CPI	EOL ;
2ADF	C2EC2A	=8836	JNZ	GORO12 ; If Sym = EOL then
		=8837		;
2AE2	3E01	=8838	MVI	A,TRIGEX ;
2AE4	32B3EA	=8839	STA	TRIG ; Trig := TriggerExecuted;
2AE7	3E00	=8840	MVI	A,EOL ;
2AE9	C3FE2A	=8841	JMP	GORO20 ;
		=8842	GORO12:	;
2AEC	CD662B	=8843	CALL	TRIG? ;
2AEF	C2162B	=8844	JNZ	GORO80 ; ElseIf Trigger?(Sym) then
		=8845		;
2AF2	32B3EA	=8846	STA	TRIG ; Trig := TriggerLookup(Sym);
		=8847		Else
		=8848		ExitWithError;
		=8849		;
2AF5	CDFF1B	=8850	CALL	GETSYM ; Sym := GetNextSymbol;
		=8851		;
2AF8	C3FE2A	=8852	JMP	GORO20 ; End
		=8853	GORO15:	End Else
2AFB	C3162B	=8854	JMP	GORO80 ; ExitWithError;
		=8855	GORO20:	;
2AFE	FE13	=8856	CPI	SYMFOR ;
2B00	C20B2B	=8857	JNZ	GORO30 ; If Sym = SymFOREVER begin
		=8858		;
2B03	3E00	=8859	MVI	A,TRIGFOR ;
2B05	32B3EA	=8860	STA	TRIG ; Trig := TriggerForever;
2B08	CDFF1B	=8861	CALL	GETSYM ; Sym := GetNextSymbol;
		=8862		;
		=8863	GORO30:	End;
2B0B	FE00	=8864	CPI	EOL ;
2B0D	C2162B	=8865	JNZ	GORO80 ; If Sym = EOL then
		=8866		;
2B10	CD8E2A	=8867	CALL	GO ; Go
2B13	C31C2B	=8868	JMP	GORO99 ; Else
		=8869	GORO80:	;
2B16	211D2B	=8870	LXI	H,ERMGO ; ExitWithError;
2B19	CD8119	=8871	CALL	CMDERR ;
		=8872	GORO99:	;
2B1C	C9	=8873	RET	; EndProc GoRoutine;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 152

LOC	OBJ	LINE	SOURCE STATEMENT
		=8874 ;	
		=8875 ERMGO:	
		=8876 ;	
		=8877 ; Format.	
		=8878 ;	
		=8879 ; GO [FROM <Number>] [FOREVER <TriggerCondition>]	
		=8880 ; <TriggerCondition> = TILL <Number> [<Condition>]	
		=8881 ; <Condition> = EXECUTED INPUT OUTPUT READ WRITE	
		=8882 ;	
		=8883 ; Declaration.	
		=8884 ;	
2B1D	47CFDB46	=8885	DB 'G', 'O'+MSB, '['+MSB, 'FRO', 'M'+MSB, PSNUM, '']'+MSB
	524FCD80		
	DD		
2B26	DB464F52	=8886	DB '['+MSB, 'FOREVER ', PSTRIG, '']'
	45564552		
	7C815D		
2B31	OD0A09	=8887	DB CR, LF, TAB
		=8888	
2B34	81BD5449	=8889	DB PSTRIG, '='+MSB, 'TIL', 'L'+MSB, PSNUM, '['+MSB
	4CCC80DB		
2B3C	3C825D	=8890	DB '<', PSCOND, '']'
2B3F	OD0A09	=8891	DB CR, LF, TAB
		=8892	
2B42	3C82BD45	=8893	DB '<', PSCOND, '='+MSB, 'EXECUTED INPUT OUTPUT READ WRITE'
	58454355		
	5445447C		
	494E5055		
	547C4F55		
	54505554		
	7C524541		
	447C5752		
	495445		
		=8894	
2B65	00	=8895	DB EndMsg
		=8896 ;	
		=8897 ;=====*	
		=8898 ;*	
		=8899 ;* Function : TRIG? (Trigger condition ?).	
		=8900 ;* Inputs : A : Symbol to be checked.	
		=8901 ;* Outputs : Z = 1 : Symbol is a trigger condition.	
		=8902 ;* Z = 0 : Symbol is not a trigger condition.	
		=8903 ;* If the symbol is a trigger condition then the trigger	
		=8904 ;* type is returned in the A register.	
		=8905 ;* Destroys : A and flags.	
		=8906 ;* Calls : None.	
		=8907 ;*	
		=8908 ;* Description : This routine checks to see if the symbol passed to	
		=8909 ;* the routine is a trigger condition. The following are valid	
		=8910 ;* trigger condition and the trigger type.	
		=8911 ;*	
		=8912 ;* String Symbol Trigger type	
		=8913 ;*	
		=8914 ;* 'EXECUTE' SYMEXEC TRIGEX	
		=8915 ;* 'INPUT' SYMINPUT TRIGIN	
		=8916 ;* 'OUTPUT' SYMOUT TRIGOUT	
		=8917 ;* 'READ' SYMRD TRIGRD	
		=8918 ;* 'WRITE' SYMWRITE TRIGWR	
		=8919 ;*	
		=8920 ;=====*	
		=8921 ;	
		=8922 TRIG?: ; Procedure Trigger?(Sym:Symbol):Boolean;	
		=8923 ;	
		=8924 ; Case Sym of	
2B66	FE0F	=8925 CPI SYMEXEC	
2B68	C2702B	=8926 JNZ TRIG?0	
		=8927 ; SymEXEC :	
		=8928 MVI A,TRIGEX	
2B6B	3E01	=8928 MVI A,TRIGEX	
2B6D	C3952B	=8929 JMP TRIG?9	
		=8930 TRIG?0: ; TrigCondition := TrigExecute;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 153

LOC	OBJ	LINE	SOURCE STATEMENT
2B70	FE19	=8931	CPI SYMINPUT ;
2B72	C27A2B	=8932	JNZ TRIG71 ; SymINPUT :
		=8933	;
2B75	3E02	=8934	MVI A,TRIGIN ; TrigCondition := TrigInput;
2B77	C3952B	=8935	JMP TRIG79 ;
		=8936	TRIG71: ;
2B7A	FE22	=8937	CPI SYMOUT ;
2B7C	C2842B	=8938	JNZ TRIG72 ; SymOUTPUT :
		=8939	;
2B7F	3E03	=8940	MVI A,TRIGOUT ; TrigCondition := TrigOutput;
2B81	C3952B	=8941	JMP TRIG79 ;
		=8942	TRIG72: ;
2B84	FE27	=8943	CPI SYMRD ;
2B86	C28E2B	=8944	JNZ TRIG73 ; SymREAD :
		=8945	;
2B89	3E04	=8946	MVI A,TRIGRD ; TrigCondition := TrigRead;
2B8B	C3952B	=8947	JMP TRIG79 ;
		=8948	TRIG73: ;
2B8E	FE30	=8949	CPI SYMWRITE ;
2B90	C2952B	=8950	JNZ TRIG79 ; SymWRITE :
		=8951	;
2B93	3E05	=8952	MVI A,TRIGWR ; TrigCondition := TrigWritten;
		=8953	TRIG79: ;
		=8954	;
		=8955	;
		=8956	;
		=8957	;
2B95	C9	=8958	RET ; EndProc Trigger?;
		=8959	;
		=8960	END <DEBGO.A85>
		8961	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 154

LOC	OBJ	LINE	SOURCE STATEMENT
		8962 ;	
		8963 ;	PORT command routine:
		8964 ;	
		8965 \$INCLUDE(DEBPORT.A85)	
		=8966 ;	-----;
		=8967 ;	;
		=8968 ;	PORT command routines used by the UCT SABus debugger. ;
		=8969 ;	;
		=8970 ;	This module contains (in order) the following subroutines. ;
		=8971 ;	;
		=8972 ;	a) PORTRD - Port read. ;
		=8973 ;	b) PORTROU - PORT command routine. ;
		=8974 ;	c) PORTWR - Port write. ;
		=8975 ;	;
		=8976 ;	-----;
		=8977 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 155

LOC	OBJ	LINE	SOURCE STATEMENT
		=8978 ;	=====*
		=8979 ;*	*
		=8980 ;*	Function : PORTRD (Port read).
		=8981 ;*	Inputs : (PORTNUM) : Port number to be read.
		=8982 ;*	Outputs : None.
		=8983 ;*	Destroys : H, L, A and flags.
		=8984 ;*	Calls : OUTMSG, PHEXB, DYNIO and CRLF.
		=8985 ;*	*
		=8986 ;*	Description : This routine does an I/O read from the port address
		=8987 ;*	given by the memory variable PORTNUM and displays the result in
		=8988 ;*	the following format:
		=8989 ;*	*
		=8990 ;*	Port PP = DD Where PP = Port number (in hexadecimal).
		=8991 ;*	and DD = Data read from the port (also in
		=8992 ;*	hexadecimal)
		=8993 ;*	*
		=8994 ;*	The 8085 does not support indirect port access and the program must
		=8995 ;*	therefore construct it's own indirect port access routine. It does
		=8996 ;*	this by writing the following three bytes into RAM and then
		=8997 ;*	calling them as a subroutine.
		=8998 ;*	*
		=8999 ;*	Byte 1 : Opcode for an IN instruction.
		=9000 ;*	Byte 2 : The port number obtained from the memory variable PORTNUM
		=9001 ;*	Byte 3 : Opcode for a RET instruction.
		=9002 ;*	*
		=9003 ;*	On returning from this routine the A register will contain the value
		=9004 ;*	read from the port address given on the command line.
		=9005 ;*	*
		=9006 ;	=====*
		=9007 ;	
		=9008	PORTRD: ; Procedure PortRead;
2B96	21BF2B	=9009	LXI H,PMSG1 ;
2B99	CDEE18	=9010	CALL OUTMSG ; PrintMessage(PortMessage1);
2B9C	3ACDEA	=9011	LDA PORTNUM ;
2B9F	CD041A	=9012	CALL PHEXB ; PrintHexByte(PortNumber);
2BA2	21C42B	=9013	LXI H,PMSG2 ;
2BA5	CDEE18	=9014	CALL OUTMSG ; PrintMessage(PortMessage2);
2BA8	3ACDEA	=9015	LDA PORTNUM ;
2BAB	21CEEAA	=9016	LXI H,DYNIO ;
2BAE	36DB	=9017	MVI M,(IN) ;
2BB0	23	=9018	INX H ;
2BB1	77	=9019	MOV M,A ;
2BB2	23	=9020	INX H ;
2BB3	36C9	=9021	MVI M,(RET) ; CreateInputRoutine;
2BB5	CDCEEAA	=9022	CALL DYNIO ;
2BB8	CD041A	=9023	CALL PHEXB ; PrintHexByte(InputRoutine);
2BBB	CDB419	=9024	CALL CRLF ; Newline;
		=9025	;
2BBE	C9	=9026	RET ; EndProc PortRead;
		=9027 ;	
		=9028	PMSG1:
		=9029 ;	
		=9030 ;	format:
		=9031 ;	
		=9032 ;	Port
		=9033 ;	
		=9034 ;	declaration:
		=9035 ;	
2BBF	506F72F4 00	=9036	DB 'Por', 't'+MSB, EndMsg
		=9037 ;	
		=9038	PMSG2:
		=9039 ;	
		=9040 ;	format:
		=9041 ;	
		=9042 ;	=
		=9043 ;	
		=9044 ;	declaration:
		=9045 ;	
2BC4	20BD00	=9046	DB ' ', '='+MSB, EndMsg

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 156

LOC	OBJ	LINE	SOURCE STATEMENT
		=9047 ;	
		=9048 ;=====*	
		=9049 ;*	
		=9050 ;* Function : PORTROU (PORT command routine).	
		=9051 ;* Inputs : None.	
		=9052 ;* Outputs : None.	
		=9053 ;* Destroys : A, H, L and flags.	
		=9054 ;* Calls : GETSYM, CMDERR, PORTRD and PORTWR.	
		=9055 ;*	
		=9056 ;* Description : This routine handles the PORT command. This command	
		=9057 ;* allows SINGLE port reads or writes to be performed. The BNF syntax	
		=9058 ;* for the PORT command is given below:	
		=9059 ;*	
		=9060 ;* Syntax: PORT <Number> [= <Number>] <EOL>	
		=9061 ;*	
		=9062 ;* Some sample commands and their meanings:	
		=9063 ;*	
		=9064 ;* PORT 00 - Does an input from port 0 and displays the	
		=9065 ;* result.	
		=9066 ;*	
		=9067 ;* PORT 12H = 0FFH - Write 0FFH to port 12H.	
		=9068 ;*	
		=9069 ;=====*	
		=9070 ;	
		=9071 PORTROU: ; Procedure PortRoutine;	
		=9072 ;	
2BC7	CDFF1B	=9073 CALL GETSYM ; Sym := GetNextSymbol;	
2BCA	FE02	=9074 CPI NUMBER ;	
2BCC	C2FE2B	=9075 JNZ PORT90 ; If Sym = Number then	
		=9076 ; Begin	
2BCF	3A00EA	=9077 LDA SYMVAL ;	
2BD2	32CDEA	=9078 STA PORTNUM ; PortNumber := LOW SymValue;	
		=9079 ;	
2BD5	CDFF1B	=9080 CALL GETSYM ; Sym := GetNextSymbol;	
2BD8	FE00	=9081 CPI EOL ;	
2BDA	C2E32B	=9082 JNZ PORT10 ; If Sym = EOL then	
		=9083 ;	
2BDD	CD962B	=9084 CALL PORTRD ; DisplayRead	
2BE0	C3042C	=9085 JMP PORT99 ;	
		=9086 PORT10: ; Else	
2BE3	FE04	=9087 CPI EQUALS ; If Sym = Equals then begin	
2BE5	C2FE2B	=9088 JNZ PORT90 ;	
		=9089 ;	
2BE8	CDFF1B	=9090 CALL GETSYM ; Sym := GetNextSymbol;	
2BEB	FE02	=9091 CPI NUMBER ;	
2BED	C2FE2B	=9092 JNZ PORT90 ; If Sym = Number then begin	
		=9093 ;	
2BF0	CDFF1B	=9094 CALL GETSYM ; Sym := GetNextSymbol	
2BF3	FE00	=9095 CPI EOL ;	
2BF5	C2FE2B	=9096 JNZ PORT90 ; If Sym = EOL then	
		=9097 ;	
2BF8	CD0F2C	=9098 CALL PORTWR ; PortWrite;	
2BFB	C3042C	=9099 JMP PORT99 ; End	
		=9100 ; End	
		=9101 ;	
		=9102 PORT90: ; Else	
2BFE	21052C	=9103 LXI H, ERMPORT ;	
2C01	CD8119	=9104 CALL CMDERR ; CommandError(PortError);	
		=9105 PORT99: ;	
2C04	C9	=9106 RET ; EndProc PortRoutine;	
		=9107 ;	
		=9108 ERMPORT:	
		=9109 ;	
		=9110 ; format:	
		=9111 ;	
		=9112 ; PORT <Number> [= <Number>]	
		=9113 ;	
		=9114 ; declaration:	
		=9115 ;	
2C05	504F52D4	=9116 DB 'POR', 'T'+MSB, PSNUM, '['+MSB, '='+MSB, PSNUM, ']', EndMsg	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 157

LOC	OBJ	LINE	SOURCE STATEMENT
	80DBBD80		
	5D00		
		=9117 ;	
		=9118 ;=====*	
		=9119 ;*	
		=9120 ;* Function : PORTWR (Port write).	*
		=9121 ;* Inputs : (PORTNUM) : Port number to write to.	*
		=9122 ;* (SYMVAL) : Value to write to the port.	*
		=9123 ;* Outputs : None.	*
		=9124 ;* Destroys : A, H, L and flags.	*
		=9125 ;* Calls : DYNIO and ERROR.	*
		=9126 ;*	*
		=9127 ;* Description : This routine performs an IO write to the port	*
		=9128 ;* port location given by the memory variable PORTNUM. The data	*
		=9129 ;* output is given by the memory variable SYMVAL.	*
		=9130 ;*	*
		=9131 ;* As with the PORTRD routine a dynamic output routine must be	*
		=9132 ;* constructed to perform the output. The three bytes that are written	*
		=9133 ;* are given below. Before the routine is called the value to be	*
		=9134 ;* output is loaded into the A register.	*
		=9135 ;*	*
		=9136 ;* Byte 1 : Opcode for an OUT instruction.	*
		=9137 ;* Byte 2 : The port number obtained from the memory variable PORTNUM	*
		=9138 ;* Byte 3 : Opcode for a RET instruction.	*
		=9139 ;*	*
		=9140 ;* Note: The protection is checked before the value is output. If	*
		=9141 ;* the protection is set to USER then the write is performed	*
		=9142 ;* otherwise an error message is printed.	*
		=9143 ;*	*
		=9144 ;=====*	
		=9145 ;	
		=9146 PORTWR:	; Procedure PortWrite;
2C0F	3ACDEA	=9147 LDA PORTNUM	;
2C12	CDB110	=9148 CALL CHKPWR	;
2C15	CA2E2C	=9149 JZ POWR05	; If CheckPortWrite(PortNum) then
		=9150 ;	;
		=9151 ;	Begin
2C18	21CEEA	=9152 LXI H,DYNIO	;
2C1B	36D3	=9153 MVI M,(OUT)	;
2C1D	23	=9154 INX H	;
2C1E	3ACDEA	=9155 LDA PORTNUM	;
2C21	77	=9156 MOV M,A	;
2C22	23	=9157 INX H	;
2C23	36C9	=9158 MVI M,(RET)	; CreateOutInstruction;
2C25	3A00EA	=9159 LDA SYMVAL	;
2C28	CDCEEA	=9160 CALL DYNIO	; Output(LOW SymbolValue);
2C2B	C3342C	=9161 JMP POWR99	; End
		=9162 ;	;
		=9163 POWR05:	; Else
2C2E	21352C	=9164 LXI H,PROERR	;
2C31	CDC119	=9165 CALL ERROR	; Error(ProtectionError);
		=9166 POWR99:	;
		=9167 ;	;
2C34	C9	=9168 RET	; EndProc PortWrite;
		=9169 ;	;
		=9170 PROERR:	;
		=9171 ;	;
		=9172 ; format:	;
		=9173 ;	;
		=9174 ; Protection error	;
		=9175 ;	;
		=9176 ; declaration:	;
		=9177 ;	;
2C35	50726F74 65638320 8500	=9178 DB 'Protec', PSTION, ' ', PSERROR, EndMsg	
		=9179 ;	;
		=9180 ; END <DEBPORT.A85>	
		9181 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 158

LOC	OBJ	LINE	SOURCE STATEMENT
		9182 ;	
		9183 ;	PRINT command routine:
		9184 ;	
		9185 \$INCLUDE(DEBPRINT.A85)	
		=9186 ;	-----;
		=9187 ;	;
		=9188 ;	DEBPRINT module for the UCT SABus debugger ;
		=9189 ;	-----;
		=9190 ;	;
		=9191 ;	This module contains the trace buffer print routines for the UCT ;
		=9192 ;	SABus 8085 symbolic debugger. ;
		=9193 ;	;
		=9194 ;	This module contains (in order) the following routines: ;
		=9195 ;	;
		=9196 ;	a) ADDLEN - Add the record length to the buffer pointer. ;
		=9197 ;	i.e. set pointer to point to next record. ;
		=9198 ;	b) FADR - Fetch address. ;
		=9199 ;	c) P16REF - Print 16 bit reference. ;
		=9200 ;	d) P8REF - Print 8 bit reference. ;
		=9201 ;	e) P10REF - Print I/O Reference. ;
		=9202 ;	f) DISPREF - Print Reference. ;
		=9203 ;	g) PRINTROU - PRINT command handler. ;
		=9204 ;	h) PRTALL - This routine prints the entire trace buffer. ;
		=9205 ;	i) PRTLINE - This routine print one trace buffer entry. ;
		=9206 ;	j) PRTNUM - This routine prints a number of trace buffer ;
		=9207 ;	entries on the screen (Calls PRTLINE). ;
		=9208 ;	k) PRTREF - This routine prints any memory or I/O ;
		=9209 ;	references make by the instruction. ;
		=9210 ;	l) PRTYPE - This routine prints the access type. ;
		=9211 ;	;
		=9212 ;	-----;
		=9213 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 159

LOC	OBJ	LINE	SOURCE STATEMENT
		=9214 ;	=====*
		=9215 ;*	*
		=9216 ;*	Function : ADDLEN (Add length) *
		=9217 ;*	Inputs : HL : Trace buffer pointer (pointing to start of record) *
		=9218 ;*	Outputs : HL : Trace buffer pointer (pointing to next record) *
		=9219 ;*	Destroys : H, L, A and flags. *
		=9220 ;*	Calls : ROUND. *
		=9221 ;*	*
		=9222 ;*	Description : This routine increments the trace buffer pointer to *
		=9223 ;*	point to the start of the next record in the trace buffer. It does *
		=9224 ;*	this by adding the length of the current record (the first byte *
		=9225 ;*	of the record) to the start address of the current record. Wrap- *
		=9226 ;*	around is then implemented using the ROUND routine. *
		=9227 ;*	*
		=9228 ;	=====*
		=9229 ;	
		=9230 ADDLEN:	; Procedure AddRecordLength(Ptr:Word);
2C3F	C5	=9231	PUSH B ;
2C40	4E	=9232	MOV C,M ;
2C41	0600	=9233	MVI B,0 ;
2C43	09	=9234	DAD B ;
2C44	CD2C2F	=9235	CALL ROUND ; Ptr := Round(Ptr + Memory[Ptr]);
2C47	C1	=9236	POP B ;
2C48	C9	=9237	RET ; EndProc AddRecordLength;
		=9238 ;	
		=9239 ;	=====*
		=9240 ;*	*
		=9241 ;*	Function : FADR (Fetch address) *
		=9242 ;*	Inputs : HL : Pointer to address in trace buffer. *
		=9243 ;*	Outputs : HL : Pointer to opcode field in trace buffer. *
		=9244 ;*	DE : Address of instruction (from trace buffer) *
		=9245 ;*	Destroys : H, L, D, E and flags. *
		=9246 ;*	Calls : GETINDIRECT. *
		=9247 ;*	*
		=9248 ;*	Description : This routine fetches the address of the instruction *
		=9249 ;*	in the trace buffer and increments the pointer to point to the *
		=9250 ;*	opcode field. *
		=9251 ;*	*
		=9252 ;	=====*
		=9253 ;	
		=9254 FADR:	; Function FetchAddress(Ptr:Word) : Word;
2C49	54	=9255	MOV D,H ;
2C4A	5D	=9256	MOV E,L ;
2C4B	CD9414	=9257	CALL GETINDIRECT ; Address := GetIndirect(Ptr);
2C4E	EB	=9258	XCHG ;
2C4F	23	=9259	INX H ;
2C50	23	=9260	INX H ; Ptr := Ptr + 1;
		=9261	; Return(Address)
		=9262	; EndProc FetchAddress;
2C51	C9	=9263	RET ; EndProc FetchAddress;
		=9264	
		=9265 ;	
		=9266 ;	=====*
		=9267 ;*	*
		=9268 ;*	Function : P16REF (Print 16 bit reference) *
		=9269 ;*	Inputs : HL : Pointer to address field in trace buffer. *
		=9270 ;*	B : operation code. ('R' or 'W') *
		=9271 ;*	Outputs : HL : Incremented pointer. *
		=9272 ;*	Destroys : H, L, D, E, A and flags. *
		=9273 ;*	Calls : FADR and DISPREF. *
		=9274 ;*	*
		=9275 ;*	Description : This routine displays the information about a 16 *
		=9276 ;*	bit access as two eight bit accesses in the following format: *
		=9277 ;*	*
		=9278 ;*	XXXX-O-AA YYYY-O-BB *
		=9279 ;*	*
		=9280 ;*	XXXX = Address given in the address field of the trace buffer record *
		=9281 ;*	YYYY = XXXX + 1 *
		=9282 ;*	O = operation performed 'R' = Read or 'W' = Write. *
		=9283 ;*	AA = Low byte of the word read/written *

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 160

LOC	OBJ	LINE	SOURCE STATEMENT
		=9284 ;*	BB = High byte of the word read/written *
		=9285 ;*	
		=9286 ;*	Note: The operation code, in the B register, is simply passed to *
		=9287 ;*	the DISPREF routine and is not examined by this routine. *
		=9288 ;*	
		=9289 ;=====*	
		=9290 ;	
		=9291 P16REF:	; Procedure Print16BitReference (Ptr:Word;
		=9292	; RefType:Char);
		=9293	
2C52	CD492C	=9294	CALL FADR ; Address := FetchAddress(Ptr);
		=9295	; Ptr := Ptr + 2; {Skip address}
		=9296	
2C55	7E	=9297	MOV A,M ; LowData := TraceBufferEntry(Ptr);
		=9298	
2C56	CD7A2C	=9299	CALL DISPREF ; DisplayReference(Address,LowData,RefType);
		=9300	
2C59	13	=9301	INX D ; Address := Address + 1;
2C5A	23	=9302	INX H ; Ptr := Ptr + 1;
		=9303	
2C5B	7E	=9304	MOV A,M ; HighData := TraceBufferEntry(Ptr);
		=9305	
2C5C	CD7A2C	=9306	CALL DISPREF ; DisplayReference(Address,HighData,Reftype);
		=9307	
2C5F	C9	=9308	RET ; EndProc;
		=9309 ;	
		=9310 ;=====*	
		=9311 ;*	
		=9312 ;*	Function : P8REF (Print 8 bit reference) *
		=9313 ;*	Inputs : HL : Pointer to address field in trace buffer. *
		=9314 ;*	B : operation code. ('R' or 'W') *
		=9315 ;*	Outputs : HL : Incremented pointer. *
		=9316 ;*	Destroys : H, L, D, E, A and flags. *
		=9317 ;*	Calls : FADR and DISPREF. *
		=9318 ;*	
		=9319 ;*	Description : This routine displays the information about an 8 *
		=9320 ;*	bit access. The reference is displayed in the following format: *
		=9321 ;*	
		=9322 ;*	XXXX-O-AA *
		=9323 ;*	
		=9324 ;*	XXXX = Address given in the address field of the trace buffer record*
		=9325 ;*	O = operation performed 'R' = Read or 'W' = Write. *
		=9326 ;*	AA = value of the data byte read/written *
		=9327 ;*	
		=9328 ;*	Note: The operation code, in the B register, is simply passed to *
		=9329 ;*	the DISPREF routine and is not examined by this routine. *
		=9330 ;*	
		=9331 ;=====*	
		=9332 ;	
		=9333 P8REF:	; Procedure Print8BitRef(Ptr:Word;
		=9334	; RefType:Char);
		=9335	
2C60	CD492C	=9336	CALL FADR ; Address := FetchAddress(Ptr);
		=9337	; Ptr := Ptr + 2;
		=9338	
2C63	7E	=9339	MOV A,M ; DataValue := TraceBufferEntry(ptr);
		=9340	
2C64	CD7A2C	=9341	CALL DISPREF ; DisplayReference(Address, RefType, Data);
		=9342	
2C67	C9	=9343	RET ; EndProc Print8BitRef;
		=9344 ;	
		=9345 ;=====*	
		=9346 ;*	
		=9347 ;*	Function : PIOREF (Print I/O Reference) *
		=9348 ;*	Inputs : HL : Pointer to trace buffer. *
		=9349 ;*	B : reference type ('I'=Input or 'O'=Output) *
		=9350 ;*	Outputs : Incremented pointer. *
		=9351 ;*	Destroys : A, H, L and flags. *
		=9352 ;*	Calls : SPACES, PHEXB and PRTYPE. *
		=9353 ;*	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 161

LOC	OBJ	LINE	SOURCE STATEMENT
		=9354 ;*	Description : This routine prints the information from an I/O
		=9355 ;*	reference. The following format is generated.
		=9356 ;*	
		=9357 ;*	' AA-O-DD' Note the two leading spaces.
		=9358 ;*	
		=9359 ;*	AA = Port address being accessed
		=9360 ;*	O = Operation being performed ('O' or 'I')
		=9361 ;*	DD = Data byte input or output.
		=9362 ;*	
		=9363 ;	=====
		=9364 ;	
		=9365	PIOREF: ; Procedure PrintIOReference(Ptr:Word;
		=9366	RefType:Char);
2C68	3E02	=9367	MVI A,2 ;
2C6A	CDA71A	=9368	CALL SPACES ; PrintSpaces(2);
		=9369	;
2C6D	7E	=9370	MOV A,M ; PortAddress := TraceBufferEntry(ptr);
		=9371	;
2C6E	CD041A	=9372	CALL PHEXB ; PrintHexByte(PortAddress);
		=9373	;
2C71	CD4F2E	=9374	CALL PRTYPE ; PrintRefType(RefType);
		=9375	;
2C74	23	=9376	INX H ; Ptr := Ptr + 1;
2C75	7E	=9377	MOV A,M ; Data := TraceBufferEntry(Ptr);
		=9378	;
2C76	CD041A	=9379	CALL PHEXB ; PrintHexByte(Data);
		=9380	;
2C79	C9	=9381	RET ; EndProc;
		=9382 ;	
		=9383 ;	=====
		=9384 ;*	
		=9385 ;*	Function : DISPREF (Display reference)
		=9386 ;*	Inputs : DE : Address being accessed.
		=9387 ;*	B : Reference type ('R'=read or 'W'=Written)
		=9388 ;*	A : Value being read or written.
		=9389 ;*	Outputs : None.
		=9390 ;*	Destroys : A and flags.
		=9391 ;*	Calls : SPACES, PHEXB, PHEXW and PRTYPE.
		=9392 ;*	
		=9393 ;*	Description : This routine outputs a single memory access in the
		=9394 ;*	following format:
		=9395 ;*	
		=9396 ;*	AAAA-O-DD
		=9397 ;*	
		=9398 ;*	AAAA = Memory address being accessed.
		=9399 ;*	O = Type of operation ('R' or 'W')
		=9400 ;*	DD = Data value read or written.
		=9401 ;*	
		=9402 ;	=====
		=9403 ;	
		=9404	DISPREF: ; Procedure DisplayReference
		=9405	(Address:Word,
		=9406	RefType:Char,
		=9407	Value:Byte);
		=9408	;
2C7A	C5	=9409	PUSH B ;
2C7B	D5	=9410	PUSH D ;
2C7C	E5	=9411	PUSH H ;
2C7D	4F	=9412	MOV C,A ;
2C7E	EB	=9413	XCHG ;
2C7F	CD1D1A	=9414	CALL PHEXW ; PrintHexWord(Address);
		=9415	;
2C82	CD4F2E	=9416	CALL PRTYPE ; PrintType(RefType);
2C85	79	=9417	MOV A,C ;
2C86	CD041A	=9418	CALL PHEXB ; PrintHexByte(Value)
2C89	3E03	=9419	MVI A,3 ;
2C8B	CDA71A	=9420	CALL SPACES ; PrintSpaces(3)
2C8E	E1	=9421	POP H ;
2C8F	D1	=9422	POP D ;
2C90	C1	=9423	POP B ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 162

LOC	OBJ	LINE	SOURCE STATEMENT
2C91	C9	=9424	RET ; EndProc DisplayReference;
		=9425 ;	
		=9426 ;=====*	
		=9427 ;*	
		=9428 ;*	Function : PRINTROU (PRINT command handler routine)
		=9429 ;*	Inputs : Command line (first symbol = PRINT
		=9430 ;*	Outputs : None.
		=9431 ;*	Destroys : A, B, C, D, E, H, L and flags.
		=9432 ;*	Calls : GETSYM, PRTALL, PRNUM and CMDERR.
		=9433 ;*	
		=9434 ;*	Description : This routine handles the 'PRINT' command. This
		=9435 ;*	command is used to display the contents of the trace buffer. There
		=9436 ;*	are two basic forms of the command. These are: display all the
		=9437 ;*	entries in the trace buffer (Command = 'PRINT<EOL>') and display
		=9438 ;*	the latest n entries in the buffer (Command = 'PRINT - n <EOL>').
		=9439 ;*	
		=9440 ;*	The syntax of the command is therefore:
		=9441 ;*	
		=9442 ;*	PRINT [- <Number>] <Eol>
		=9443 ;*	
		=9444 ;*	NOTES: a) If the number specified is greater than the number of
		=9445 ;*	entries in the trace buffer then ALL the entries are
		=9446 ;*	displayed.
		=9447 ;*	
		=9448 ;*	b) If the number specified is zero then NO entries are
		=9449 ;*	displayed.
		=9450 ;*	
		=9451 ;=====*	
		=9452 ;	
		=9453	PRINTROU: ; Procedure PrintRoutine;
		=9454	;
2C92	CDFF1B	=9455	CALL GETSYM ; Sym := GetNextSymbol;
2C95	FE00	=9456	CPI EOL ;
2C97	C2A02C	=9457	JNZ PRTR10 ; If Sym = EOL then
		=9458	Begin
2C9A	CDD42C	=9459	CALL PRTALL ; PrintALL;
2C9D	C3C92C	=9460	JMP PRTR99 ; End
		=9461	PRTR10: ; Else
2CA0	FE06	=9462	CPI MINUS ;
2CA2	C2C32C	=9463	JNZ PRTR80 ; If Sym = Minus then begin
		=9464	;
2CA5	CDFF1B	=9465	CALL GETSYM ; Sym := GetNextSymbol;
2CA8	FE02	=9466	CPI NUMBER ;
2CAA	C2C32C	=9467	JNZ PRTR80 ; If Sym = Number then
		=9468	Begin
2CAD	CDFF1B	=9469	CALL GETSYM ; Sym := GetNextSymbol;
2CB0	FE00	=9470	CPI EOL ;
2CB2	C2C32C	=9471	JNZ PRTR80 ; If Sym = EOL then
		=9472	;
2CB5	2A00EA	=9473	LHLD SYMVAL ;
2CB8	7D	=9474	MOV A,L ;
2CB9	B4	=9475	ORA H ;
2CBA	CAC92C	=9476	JZ PRTR99 ; If Symval <> 0
		=9477	;
2CBD	CD6D2D	=9478	CALL PRNUM ; PrintNumberOfEntries;
2CC0	C3C92C	=9479	JMP PRTR99 ; Else
		=9480	PRTR80: ;
2CC3	21CA2C	=9481	LXI H,ERMPRINT ;
2CC6	CD8119	=9482	CALL CMDERR ; CommandError(PrintCommand);
		=9483	PRTR99: ;
2CC9	C9	=9484	RET ; EndProc PrintRoutine;
		=9485 ;	
		=9486	ERMPRINT:
		=9487 ;	
		=9488 ;	format:
		=9489 ;	
		=9490 ;	PRINT [- <Number>]
		=9491 ;	
		=9492 ;	declaration:
		=9493 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 163

LOC	OBJ	LINE	SOURCE STATEMENT
2CCA	5052494E D4DBAD80 5D00	=9494	DB 'PRIN', 'T'+MSB, '['+MSB, '-' +MSB, PSNUM, ']', EndMsg
		=9495 ;	
		=9496 ;=====*	
		=9497 ;*	
		=9498 ;* Function : PRTALL (Print All).	*
		=9499 ;* Inputs : Trace buffer and pointers.	*
		=9500 ;* Outputs : None.	*
		=9501 ;* Destroys : A, B, C, D, E, H, L and flags.	*
		=9502 ;* Calls : PRTLINE and ADDLEN.	*
		=9503 ;*	*
		=9504 ;* Description : This routine displays the contents of the trace	*
		=9505 ;* buffer. The number of entries is given by the TBCOUNT variable	*
		=9506 ;* and the pointer to the oldest record is given by the TBBACK	*
		=9507 ;* variable. The PRTLINE routine is used to do the actual printing.	*
		=9508 ;*	*
		=9509 ;=====*	
		=9510 ;	
		=9511 PRTALL:	; Procedure PrintAll;
		=9512 ;	
2CD4	2ACBEA	=9513 LHL	TBCOUNT ; Count := TraceBufferEntryCount;
2CD7	EB	=9514 XCHG	; ;
2CD8	2AC9EA	=9515 LHL	TBBACK ; Ptr := TraceBufferBackPointer;
		=9516 ;	
		=9517 PRTA05:	; While Count <> 0 Do
2CDB	7A	=9518 MOV	A,D ;
2CDC	B3	=9519 ORA	E ;
2CDD	CAEA2C	=9520 JZ	PRTA99 ;
		=9521 ;	Begin
		=9522 ;	
2CE0	CDEB2C	=9523 CALL	PRTLINE ; PrintLine(Ptr);
2CE3	CD3F2C	=9524 CALL	ADDLEN ; AddRecordLength;
2CE6	1B	=9525 DCX	D ; Count := Count - 1;
		=9526 ;	
2CE7	C3DB2C	=9527 JMP	PRTA05 ; End;
		=9528 PRTA99:	; ;
		=9529 ;	
2CEA	C9	=9530 RET	; EndProc PrintAll;
		=9531 ;	
		=9532 ;=====*	
		=9533 ;*	
		=9534 ;* Function : PRTLINE (Print line)	*
		=9535 ;* Inputs : HL : Pointer to record in the trace buffer.	*
		=9536 ;* Outputs : None.	*
		=9537 ;* Destroys : A, B, C and flags.	*
		=9538 ;* Calls : ROUND, GETINDIRECT, PHEXW, TABOUT, GETINFO, OUTSTR	*
		=9539 ;* PRTOPRS, ADDBC, PRTREF, P16REF and CRLF.	*
		=9540 ;*	*
		=9541 ;* Description : This routine displays a record from the trace buffer	*
		=9542 ;* as a single line on the screen.	*
		=9543 ;*	*
		=9544 ;* The record is first copied into the TraceBufferEntry buffer so	*
		=9545 ;* that it can be easily be accessed.	*
		=9546 ;*	*
		=9547 ;* The following information is then displayed in the following form:	*
		=9548 ;*	*
		=9549 ;* AAAA OPCODE OPERANDS BBBB-O-DD (*)	*
		=9550 ;*	*
		=9551 ;* AAAA = Address of the instruction. (not present for interrupts)	*
		=9552 ;* OPCODE and OPERANDS are simply the disassembled instruction.	*
		=9553 ;*	*
		=9554 ;* (*) There is one of these record for every memory or I/O Access	*
		=9555 ;* made by the instruction. This can vary from none to four.	*
		=9556 ;*	*
		=9557 ;* e.g. MOV A,B would have no memory references.	*
		=9558 ;* LDA 100 would have one memory read.	*
		=9559 ;* SHLD 100 would have two memory writes.	*
		=9560 ;* XTHL would have two memory reads and	*
		=9561 ;* two memory writes.	*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 164

LOC	OBJ	LINE	SOURCE STATEMENT
		=9562	;
		=9563	;* NOTE: If an interrupt is recognised there is no instruction but
		=9564	there are two memory writes (pushing the return address).
		=9565	;
		=9566	=====
		=9567	;
		=9568	PRTLINE: ; Procedure PrintLine(Ptr:Word);
2CEB	E5	=9569	PUSH H ;
2CEC	D5	=9570	PUSH D ;
		=9571	;
2CED	11B8EA	=9572	LXI D,TBENTRY ;
2CFO	46	=9573	MOV B,M ; RecordLength := TraceBuffer[Ptr];
		=9574	;
		=9575	PRTL02: ; For I := 1 to RecordLength do
		=9576	Begin
2CF1	7E	=9577	MOV A,M ;
2CF2	12	=9578	STAX D ; TraceBufferEntry[I] :=
		=9579	TraceBuffer[Ptr];
2CF3	23	=9580	INX H ;
2CF4	CD2C2F	=9581	CALL ROUND ; Ptr := Round(Ptr+1);
2CF7	13	=9582	INX D ;
2CF8	05	=9583	DCR B ;
2CF9	C2F12C	=9584	JNZ PRTL02 ; End;
		=9585	;
		=9586	PRTL03: ;
2CFC	3AB8EA	=9587	LDA TBENTRY ;
2CFF	FE06	=9588	CPI 6 ;
2D01	C20C2D	=9589	JNZ PRTL04 ;
		=9590	;
2D04	3AB9EA	=9591	LDA TBENTRY+1 ;
2D07	FE0A	=9592	CPI EXINTR ;
2D09	CA472D	=9593	JZ PRTL05 ;
		=9594	PRTL04: ; If Execution_type <> Interrupt then
		=9595	;
2D0C	21B9EA	=9596	LXI H,TBENTRY+1 ;
		=9597	;
2D0F	CD9414	=9598	CALL GETINDIRECT ; Address := GetIndirect(Ptr+1);
2D12	CD1D1A	=9599	CALL PHEXW ; PrintHexWord(Address);
		=9600	;
2D15	CDB31A	=9601	CALL TABOUT ; TabOutput;
		=9602	;
2D18	21BBEA	=9603	LXI H,TBENTRY+3 ;
2D1B	22A9EA	=9604	SHLD DISPC ;
		=9605	;
2D1E	E5	=9606	PUSH H ; (* PUSH ptr + 3 *)
		=9607	;
2D1F	7E	=9608	MOV A,M ;
2D20	32A8EA	=9609	STA OPCODE ; Opcode := memory[Ptr+3];
		=9610	;
2D23	CD6226	=9611	CALL GETINFO ; GetInfo(opcode);
		=9612	;
2D26	2AA6EA	=9613	LHLD STRADR ;
2D29	CDF219	=9614	CALL OUTSTR ; OutPutString(StringAddress);
		=9615	;
2D2C	CDB31A	=9616	CALL TABOUT ; TabOutput;
		=9617	;
2D2F	3AA4EA	=9618	LDA OPERAND ;
2D32	CD3028	=9619	CALL PRTOPRS ; PrintOperands(Operand);
		=9620	;
2D35	CDB31A	=9621	CALL TABOUT ; TabOutput;
2D38	CDB31A	=9622	CALL TABOUT ; TabOutput;
		=9623	;
2D3B	E1	=9624	POP H ; (* POP ptr + 3 *)
		=9625	;
2D3C	CDC621	=9626	CALL ADDBC ; Ptr := Ptr + 3 + ByteCount;
		=9627	;
2D3F	7E	=9628	MOV A,M ; ExecutionType := memory[Ptr];
2D40	23	=9629	INX H ;
		=9630	;
2D41	CD9F2D	=9631	CALL PRTREF ; PrintMemoryReferences

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 165

LOC	OBJ	LINE	SOURCE STATEMENT
		=9632	
2D44	C35E2D	=9633	JMP PRTL50 ; (ExecutionType, Ptr+1);
		=9634	PRTL05: ; Else
		=9635	; Begin
2D47	21642D	=9636	LXI H,MSGINT ;
2D4A	CDF219	=9637	CALL OUTSTR ;
2D4D	CDB31A	=9638	CALL TABOUT ; TabOutput;
2D50	CDB31A	=9639	CALL TABOUT ; TabOutput;
2D53	CDB31A	=9640	CALL TABOUT ; TabOutput;
2D56	21BAEA	=9641	LXI H,TBENTRY+2 ;
2D59	0657	=9642	MVI B,'W' ;
2D5B	CD522C	=9643	CALL P16REF ; Print16BitRef('W');
		=9644	; End;
		=9645	; End;
		=9646	PRTL50: ;
		=9647	; NewLine;
2D5E	CDB419	=9648	CALL CRLF ;
		=9649	; NewLine;
2D61	D1	=9650	POP D ;
2D62	E1	=9651	POP H ;
2D63	C9	=9652	RET ; EndProc PrintLine;
		=9653	;
2D64	496E7465	=9654	MSGINT: DB 'Interrup', 't'+MSB ;
	72727570		
	F4		
		=9655	;
		=9656	*****
		=9657	;
		=9658	Function : PRTNUM (Print a number of trace buffer entries) *
		=9659	Inputs : (SYMVAL) : The number of trace buffer entries to print. *
		=9660	Trace buffer and pointers. *
		=9661	Outputs : None. *
		=9662	Destroys : A, B, C, D, E, H, L and flags. *
		=9663	Calls : DCMP, PRTALL, ADDLEN, BCMP and PRTLINE. *
		=9664	;
		=9665	Description : This routine prints a number of entries from the *
		=9666	buffer. This routine is used with the "PRINT - n" version of the *
		=9667	print command, where n is the number of entries to be printed. *
		=9668	;
		=9669	This routine checks to see if there are n entries in the trace *
		=9670	buffer. If there are less than n entries in the trace buffer then *
		=9671	the command reverts to its first form "PRINT", and the PRTALL *
		=9672	routine is called. *
		=9673	;
		=9674	The actual printing of the records is done by the PRTLINE routine. *
		=9675	;
		=9676	*****
		=9677	;
		=9678	PRTNUM: ; Procedure PrintNumberOfEntries;
2D6D	2ACBEA	=9679	LHLD TBCOUNT ;
2D70	44	=9680	MOV B,H ;
2D71	4D	=9681	MOV C,L ; TempCount := TraceBufferCount;
2D72	EB	=9682	XCHG ;
2D73	2A00EA	=9683	LHLD SYMVAL ;
2D76	CDBF1A	=9684	CALL DCMP ;
2D79	DA822D	=9685	JC PRTN10 ; If SymVal >= TraceBufferCount then
		=9686	; PrintAll
2D7C	CDD42C	=9687	CALL PRTALL ;
2D7F	C39E2D	=9688	JMP PRTN20 ;
		=9689	PRTN10: ; Else
		=9690	; Begin
2D82	EB	=9691	XCHG ;
2D83	2AC9EA	=9692	LHLD TBBACK ; Ptr := TraceBufferBackPointer;
		=9693	PRTN12: ;
		=9694	; Repeat
		=9695	; Repeat
2D86	CD3F2C	=9696	CALL ADDLEN ; Ptr := AddLen(Ptr);
		=9697	; TempCount := TempCount - 1;
2D89	0B	=9698	DCX B ;
2D8A	EB	=9699	XCHG ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 166

LOC	OBJ	LINE	SOURCE STATEMENT
2D8B	CDB91A	=9700	CALL BCMP ;
2D8E	EB	=9701	XCHG ;
2D8F	C2862D	=9702	JNZ PRTN12 ; Until TempCount = SymVal;
		=9703	; ;
		=9704	PRTN15: ; Repeat
		=9705	; ;
2D92	CDEB2C	=9706	CALL PRTLINE ; PrintLine(Ptr);
		=9707	; ;
2D95	CD3F2C	=9708	CALL ADDLEN ; Ptr := AddLen(Ptr);
		=9709	; ;
2D98	1B	=9710	DCX D ; TempCount := TempCount - 1;
2D99	7A	=9711	MOV A,D ;
2D9A	B3	=9712	ORA E ;
2D9B	C2922D	=9713	JNZ PRTN15 ; Until TempCount = 0;
		=9714	; ;
		=9715	; End;
		=9716	PRTN20: ;
2D9E	C9	=9717	RET ; EndProc PrintNumberOfEntries;
		=9718	; ;
		=9719	=====*
		=9720	;*
		=9721	;* Function : PRTREF (Print Memory and I/O references) *
		=9722	;* Inputs : A : Reference type. *
		=9723	;* Outputs : None. *
		=9724	;* Destroys : A, H, L and flags. *
		=9725	;* Calls : OUTMSG, P8REF, P16REF, P10REF, DISPREF and FADR. *
		=9726	;* ;*
		=9727	;* Description : This routine prints the memory and I/O references *
		=9728	;* make by an instruction. The reference type is examined and the *
		=9729	;* correct information printed. *
		=9730	;* ;*
		=9731	;* The possible reference types are as follows: *
		=9732	;* ;*
		=9733	;* ReferenceType Action. *
		=9734	;* ;*
		=9735	;* NOTEEXEC - Output message ('Instruction not executed') *
		=9736	;* EXNORE - Output nothing. *
		=9737	;* EX8BRD - Single 8 bit read. Output: AAAA-R-DD *
		=9738	;* EX8BWR - Single 8 bit write. Output: AAAA-W-DD *
		=9739	;* EX16BRD - 16 bit read. Output: AAAA-R-DD BBBB-R-EE *
		=9740	;* EX16BWR - 16 bit write. Output: AAAA-W-DD BBBB-W-EE *
		=9741	;* EX8BIN - 8 bit input. Output: PP-I-DD *
		=9742	;* EX8BOUT - 8 bit output. Output: PP-O-DD *
		=9743	;* EX8BMOD - 8 Bit modify. Output: AAAA-R-DD AAAA-W-FF *
		=9744	;* EX16BMOD - 16 bit modify. Output: *
		=9745	;* ;*
		=9746	;* AAAA-R-DD AAAA-W-FF BBBB-R-EE BBBB-W-GG *
		=9747	;* ;*
		=9748	;* AAAA = memory address being accessed (in the case of 16 bit *
		=9749	;* accesses this is the first address (address of LSB) *
		=9750	;* BBBB = AAAA + 1 (address of MSB) *
		=9751	;* DD = Byte accessed (LSB in the case of 16 bit accesses) *
		=9752	;* EE = Most significant byte of a 16 bit access. *
		=9753	;* FF = Modified value written during an 8 bit modify operation *
		=9754	;* (LSB during a 16 bit modify) *
		=9755	;* GG = Most significant byte written during a 16 bit modify. *
		=9756	;* PP = Port address accessed. *
		=9757	;* ;*
		=9758	=====*
		=9759	; ;
		=9760	PRTREF: ; Procedure PrintMemoryReferences
		=9761	; (ReferenceType:Byte);
		=9762	; ;
2D9F	FE00	=9763	CPI NOTEEXEC ; Case ReferenceType of
		=9764	; ;
2DA1	C2AD2D	=9765	JNZ PREF10 ; NotExecuted :
		=9766	; ;
2DA4	21432E	=9767	LXI H,NEXECM ;
2DA7	CDEE18	=9768	CALL OUTMSG ; OutputMessage(NotExecutedMessage);
2DAA	C3422E	=9769	JMP PREF99 ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 167

LOC	OBJ	LINE	SOURCE STATEMENT
		=9770	PREF10:
2DAD	FE01	=9771	CPI EXNOREF
2DAF	C2B52D	=9772	JNZ PREF20
		=9773	
2DB2	C3422E	=9774	JMP PREF99
		=9775	PREF20:
2DB5	FE02	=9776	CPI EX8BRD
2DB7	C2C22D	=9777	JNZ PREF30
		=9778	
2DBA	0652	=9779	MVI B,'R'
2DBC	CD602C	=9780	CALL P8REF
2DBF	C3422E	=9781	JMP PREF99
		=9782	PREF30:
2DC2	FE03	=9783	CPI EX8BWR
2DC4	C2CF2D	=9784	JNZ PREF40
		=9785	
2DC7	0657	=9786	MVI B,'W'
2DC9	CD602C	=9787	CALL P8REF
2DCC	C3422E	=9788	JMP PREF99
		=9789	PREF40:
2DCF	FE04	=9790	CPI EX16BRD
2DD1	C2DC2D	=9791	JNZ PREF50
		=9792	
2DD4	0652	=9793	MVI B,'R'
2DD6	CD522C	=9794	CALL P16REF
2DD9	C3422E	=9795	JMP PREF99
		=9796	PREF50:
2DDC	FE05	=9797	CPI EX16BWR
2DDE	C2E92D	=9798	JNZ PREF60
		=9799	
2DE1	0657	=9800	MVI B,'W'
2DE3	CD522C	=9801	CALL P16REF
2DE6	C3422E	=9802	JMP PREF99
		=9803	PREF60:
2DE9	FE06	=9804	CPI EX8BIN
2DEB	C2F62D	=9805	JNZ PREF70
		=9806	
2DEE	0649	=9807	MVI B,'I'
2DF0	CD682C	=9808	CALL PIOREF
2DF3	C3422E	=9809	JMP PREF99
		=9810	PREF70:
2DF6	FE07	=9811	CPI EX8BOUT
2DF8	C2032E	=9812	JNZ PREF80
		=9813	
2DFB	064F	=9814	MVI B,'O'
2DFD	CD682C	=9815	CALL PIOREF
2E00	C3422E	=9816	JMP PREF99
		=9817	PREF80:
2E03	FE08	=9818	CPI EX8BMOD
2E05	C21B2E	=9819	JNZ PREF90
		=9820	
		=9821	
2E08	CD492C	=9822	CALL FADR
2E0B	0652	=9823	MVI B,'R'
2E0D	7E	=9824	MOV A,M
2E0E	CD7A2C	=9825	CALL DISPREF
2E11	23	=9826	INX H
2E12	0657	=9827	MVI B,'W'
2E14	7E	=9828	MOV A,M
2E15	CD7A2C	=9829	CALL DISPREF
		=9830	
2E18	C3422E	=9831	JMP PREF99
		=9832	PREF90:
2E1B	FE09	=9833	CPI EX16BMOD
2E1D	C2422E	=9834	JNZ PREF99
		=9835	
		=9836	
2E20	CD492C	=9837	CALL FADR
2E23	4E	=9838	MOV C,M
2E24	23	=9839	INX H

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 168

LOC	OBJ	LINE	SOURCE STATEMENT
2E25	23	=9840	INX H ;
2E26	7E	=9841	MOV A,M ;
2E27	0652	=9842	MVI B,'R' ;
2E29	CD7A2C	=9843	CALL DISPREF ; DisplayRef(Adr,'R',LSBReadVal);
2E2C	79	=9844	MOV A,C ;
2E2D	0657	=9845	MVI B,'W' ;
2E2F	CD7A2C	=9846	CALL DISPREF ; DisplayRef(Adr,'W',LSBWriteVal);
		=9847	;
2E32	23	=9848	INX H ;
2E33	13	=9849	INX D ;
2E34	7E	=9850	MOV A,M ;
2E35	0652	=9851	MVI B,'R' ;
2E37	CD7A2C	=9852	CALL DISPREF ; DisplayRef(Adr,'R',MSBReadVal);
2E3A	2B	=9853	DCX H ;
2E3B	2B	=9854	DCX H ;
2E3C	7E	=9855	MOV A,M ;
2E3D	0657	=9856	MVI B,'W' ;
2E3F	CD7A2C	=9857	CALL DISPREF ; DisplayRef(Adr,'W',MSBWriteVal);
		=9858	;
		=9859	;
		=9860	; End;
		=9861	PREF99: ; End Case;
		=9862	;
2E42	C9	=9863	RET ; EndProc PrintMemoryReferences;
		=9864	;
		=9865	NEXECM:
		=9866	;
		=9867	; format:
		=9868	;
		=9869	; Not executed
		=9870	;
		=9871	; declaration:
		=9872	;
2E43	4E6FF465 78656375 74656400	=9873	DB 'No', 't'+MSB, 'executed', EndMsg
		=9874	;
		=9875	=====*
		=9876	;*
		=9877	;* Function : PRTYPE (Print type) *
		=9878	;* Inputs : B : Access type : 'R' = Read, 'W' = Write *
		=9879	;* 'I' = Input, 'O' = Output. *
		=9880	;* Outputs : None. *
		=9881	;* Destroys : A and flags. *
		=9882	;* Calls : KITOUT. *
		=9883	;*
		=9884	;* Description : This routine prints the access type. *
		=9885	;*
		=9886	=====*
		=9887	;
		=9888	PRTYPE: ; Procedure PrintRefType (RefCh : Char)
2E4F	3E2D	=9889	MVI A,'-' ;
2E51	CDE218	=9890	CALL KITOUT ; KitOutput('-');
2E54	78	=9891	MOV A,B ;
2E55	CDE218	=9892	CALL KITOUT ; KitOutput(RefCh);
2E58	3E2D	=9893	MVI A,'-' ;
2E5A	CDE218	=9894	CALL KITOUT ; KitOutput('-');
		=9895	;
2E5D	C9	=9896	RET ; EndProc PrintRefType;
		=9897	;
		=9898	END <DEBPRINT.A85>
		9899	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 169

LOC	OBJ	LINE	SOURCE STATEMENT
		9900 ;	
		9901 ;	STEP command routine:
		9902 ;	
		9903	\$INCLUDE(DEBSTEP.A85)
		=9904 ;	-----;
		=9905 ;	;
		=9906 ;	STEP command routines used by the UCT SABus debugger. ;
		=9907 ;	;
		=9908 ;	This module contains (in order) the following subroutines. ;
		=9909 ;	;
		=9910 ;	a) STEP - Single step routine. ;
		=9911 ;	b) STEProu - STEP command routine. ;
		=9912 ;	;
		=9913 ;	-----;
		=9914	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 170

LOC	OBJ	LINE	SOURCE STATEMENT
		=9915 ;	=====*
		=9916 ;*	*
		=9917 ;*	Function : STEP (Step routine). *
		=9918 ;*	Inputs : (STEPCOUNT) : Number of instructions to execute. *
		=9919 ;*	(REGPC) : Address of first instruction to execute. *
		=9920 ;*	Outputs : None. *
		=9921 ;*	Destroys : H, L, A and flags. *
		=9922 ;*	Calls : SSINIT, SSTEP and PSTATUS. *
		=9923 ;*	*
		=9924 ;*	Description : This routine performs the single step operation. The *
		=9925 ;*	parameters, the start address and the number of instructions to *
		=9926 ;*	execute, are setup by the main step routine (STEP). This routine *
		=9927 ;*	uses the Single step routine (SSTEP) to execute a single *
		=9928 ;*	instruction. This routine is called until the counter reaches zero *
		=9929 ;*	or some other error is detected. *
		=9930 ;*	*
		=9931 ;	=====*
		=9932 ;	
		=9933 STEP:	; Procedure.Step;
		=9934	;
2E5E	CD6122	=9935 CALL SSINIT	; SingleStepInitialise;
		=9936	;
2E61	2AB0EA	=9937 LHLD STEPCOUNT	; Count := StepCounter;
2E64	3E00	=9938 MVI A,RUNNING	;
2E66	32B2EA	=9939 STA STATUS	; Status := RUNNING;
		=9940	;
		=9941 STPR05:	; Repeat
2E69	E5	=9942 PUSH H	;
		=9943	;
2E6A	CD2124	=9944 CALL REGDMP	; DumpRegisters;
2E6D	CDB419	=9945 CALL CRLF	; NewLine;
2E70	2A9DEA	=9946 LHLD REGPC	;
2E73	CDD725	=9947 CALL DISLINE	; DisassembleLine(Reg.PC);
		=9948	;
2E76	CD7222	=9949 CALL SSTEP	; SingleStep;
		=9950	;
2E79	CD2124	=9951 CALL REGDMP	; DumpRegisters;
2E7C	CDB419	=9952 CALL CRLF	; NewLine;
2E7F	CDB419	=9953 CALL CRLF	; NewLine;
		=9954	;
2E82	E1	=9955 POP H	;
2E83	2B	=9956 DCX H	; Count := Count - 1;
		=9957	;
2E84	3AB2EA	=9958 LDA STATUS	;
2E87	FE00	=9959 CPI RUNNING	;
2E89	C2912E	=9960 JNZ STPR10	; Until (Status <> Running) or
		=9961	;
2E8C	7C	=9962 MOV A,H	;
2E8D	B5	=9963 ORA L	;
2E8E	C2692E	=9964 JNZ STPR05	; (Count = 0);
		=9965	;
		=9966 STPR10:	;
2E91	CD261A	=9967 CALL PSTATUS	; PrintStatus;
		=9968	;
2E94	C9	=9969 RET	; EndProc Step;
		=9970 ;	
		=9971 ;	=====*
		=9972 ;*	*
		=9973 ;*	Function : STEPROU (STEP command routine) *
		=9974 ;*	Inputs : None. *
		=9975 ;*	Outputs : None. *
		=9976 ;*	Destroys : A, B, C, D, E, H, L and flags. *
		=9977 ;*	Calls : GETSYM, CMDERR and STEP. *
		=9978 ;*	*
		=9979 ;*	Description : This routine handles the STEP command. This command *
		=9980 ;*	allows the user to single step thru a number of instructions. *
		=9981 ;*	The BNF syntax of the STEP command is given below. *
		=9982 ;*	*
		=9983 ;*	Syntax: STEP [FROM <Number>] [COUNT <Number>] <EOL> *
		=9984 ;*	*

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 171

LOC	OBJ	LINE	SOURCE STATEMENT
		=9985 ;*	Defaults: If no start address (FROM clause) then the instruction
		=9986 ;*	pointed to by the program counter will be this first
		=9987 ;*	instruction executed.
		=9988 ;*	
		=9989 ;*	If no step count is given the previous step count
		=9990 ;*	specified will be used as the step count.
		=9991 ;*	
		=9992 ;*	Some sample STEP commands and their meanings:
		=9993 ;*	
		=9994 ;*	STEP - Start single step from the current value
		=9995 ;*	of the program counter. The number of
		=9996 ;*	steps is given by the previous count.
		=9997 ;*	
		=9998 ;*	STEP FROM 0 - Start single step at address 0. As above
		=9999 ;*	count is given by the previous count.
		=0000 ;*	
		=0001 ;*	STEP COUNT 20 - Start single step from the current program
		=0002 ;*	counter. Execute the next 20 instructions.
		=0003 ;*	
		=0004 ;*	STEP FROM 0 COUNT 10 - Execute the 10 instructions starting from
		=0005 ;*	address 0.
		=0006 ;*	
		=0007 ;*	Note: A single step command will also terminate if an ESC is typed,
		=0008 ;*	an illegal instruction is found or a guarded access is
		=0009 ;*	detected.
		=0010 ;*	
		=0011 ;*	=====
		=0012 ;	
		=0013	STEPROUT: ; Procedure StepRoutine;
2E95	CDF1B	=0014	CALL GETSYM ; Sym := GetNextSymbol;
		=0015	STP05: ;
2E98	FE14	=0016	CPI SYMFROM ;
2E9A	C2AE2E	=0017	JNZ STP10 ; If Sym = SymFROM then begin
		=0018	;
2E9D	CDF1B	=0019	CALL GETSYM ; Sym := GetNextSymbol;
2EA0	FE02	=0020	CPI NUMBER ;
2EA2	C2CF2E	=0021	JNZ STP80 ; If Sym = Number then
		=0022	Begin
2EA5	2A00EA	=0023	LHLD SYMVAL ;
2EA8	229DEA	=0024	SHLD REGPC ; ProgramCounter := SymValue;
2EAB	CDF1B	=0025	CALL GETSYM ; Sym := GetNextSymbol;
		=0026	End
		=0027	Else
		=0028	ErrorExit;
		=0029	STP10: ; End;
2EAE	FE0A	=0030	CPI SYMCOUNT ;
2EB0	C2C42E	=0031	JNZ STP20 ; If Sym = SymCOUNT then begin
		=0032	;
2EB3	CDF1B	=0033	CALL GETSYM ; Sym := GetNextSymbol;
2EB6	FE02	=0034	CPI NUMBER ;
2EB8	C2CF2E	=0035	JNZ STP80 ; If Sym = Number then
		=0036	Begin
2EBB	2A00EA	=0037	LHLD SYMVAL ;
2EBE	22B0EA	=0038	SHLD STEPCOUNT ; StepCount := SymValue;
2EC1	CDF1B	=0039	CALL GETSYM ; Sym := GetNextSymbol
		=0040	End
		=0041	Else
		=0042	ErrorExit;
		=0043	STP20: ; End
2EC4	FE00	=0044	CPI EOL ;
2EC6	C2CF2E	=0045	JNZ STP80 ; If Sym = EOL then
		=0046	;
2EC9	CD5E2E	=0047	CALL STEP ; Step
2ECC	C3D52E	=0048	JMP STP99 ;
		=0049	STP80: ; Else
2ECF	21D62E	=0050	LXI H,ERMSTEP ;
2ED2	CD8119	=0051	CALL CMDERR ; CommandError(StepCommandError);
		=0052	STP99: ;
2ED5	C9	=0053	RET ; EndProc StepRoutine;
		=0054 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 172

LOC	OBJ	LINE	SOURCE STATEMENT
		=0055	ERMSTEP:
		=0056	;
		=0057	; format:
		=0058	;
		=0059	STEP [FROM <Number>] [COUNT <Number>]'
		=0060	;
		=0061	; declaration:
		=0062	;
2ED6	535445D0 DB46524F CD80	=0063	DB 'STE', 'P'+MSB, '['+MSB, 'FRO', 'M'+MSB, PSNUM
2EE0	DDDB434F 554ED480 5D00	=0064	DB ']' +MSB, '['+MSB, 'COUN', 'T'+MSB, PSNUM, ']', EndMag
		=0065	;
		=0066	END <DEBSTEP.A85>
		10067	SEJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 173

LOC	OBJ	LINE	SOURCE STATEMENT
		10068 ;	
		10069 ;	SYMBOLIC command routine:
		10070 ;	
		10071	\$INCLUDE(DEBSYM.A85)
		=0072 ;	-----;
		=0073 ;	
		=0074 ;	SYMBOLIC command routines used by the UCT SABus debugger.
		=0075 ;	
		=0076 ;	-----;
		=0077 ;	
		=0078 ;	=====*
		=0079 ;*	*
		=0080 ;*	Function : SYMROU (SYMBOLIC command handler routine) *
		=0081 ;*	Inputs : Command line (first symbol = SYMBOLIC) *
		=0082 ;*	Outputs : None. *
		=0083 ;*	Destroys : A, B, H, L and flags. *
		=0084 ;*	Calls : GETSYM and CMDERR. *
		=0085 ;*	*
		=0086 ;*	Description : This routine handles the 'SYMBOLIC' command. This *
		=0087 ;*	command allows symbolic disassembly to be turned ON and OFF by *
		=0088 ;*	the user. *
		=0089 ;*	*
		=0090 ;*	If SYMBOLIC is ON a section of program would be displayed as *
		=0091 ;*	follows: *
		=0092 ;*	*
		=0093 ;*	1000 31 FFFF LXI SP,NEWSP *
		=0094 ;*	1003 LOOP: *
		=0095 ;*	1003 C3 0210 JMP LOOP *
		=0096 ;*	*
		=0097 ;*	If SYMBOLIC were OFF then the same piece of code would be *
		=0098 ;*	displayed as follows: *
		=0099 ;*	*
		=0100 ;*	1000 31 FFFF LXI SP,OFFFFH *
		=0101 ;*	1003 C3 0210 JMP 01003H *
		=0102 ;*	*
		=0103 ;*	BNF Syntax: SYMBOLIC = ON OFF *
		=0104 ;*	*
		=0105 ;*	NOTE: The SYMBOLIC command cannot be used to display the state of *
		=0106 ;*	the SYMBOLIC disassembly flag. i.e. 'SYM<Eol>' will generate *
		=0107 ;*	a SYMBOLIC command syntax error. *
		=0108 ;*	*
		=0109 ;	=====*
		=0110 ;	
		=0111	SYMROU: ; Procedure SymbolicRoutine;
		=0112	; ;
2EEA	CDF1B	=0113	CALL GETSYM ; Sym := GetNextSymbol;
2EED	FE04	=0114	CPI EQUALS ;
2EEF	C2152F	=0115	JNZ SYMR80 ; If Sym = Equals then begin
		=0116	; ;
2EF2	CDF1B	=0117	CALL GETSYM ; Sym := GetNextSymbol;
2EF5	FE21	=0118	CPI SYMON ;
2EF7	CA042F	=0119	JZ SYMR20 ; If Sym in [SymON, SymOFF] then
		=0120	; ;
2EFA	FE20	=0121	CPI SYMOFF ;
2EFC	C2152F	=0122	JNZ SYMR80 ;
		=0123	; If Sym = SymOFF then
2EFF	0600	=0124	MVI B,FALSE ;
2F01	C3062F	=0125	JMP SYMR30 ; Symbolic := False
		=0126	SYMR20: ;
2F04	06FF	=0127	MVI B,TRUE ; Else
		=0128	SYMR30: ;
2F06	CDF1B	=0129	CALL GETSYM ; Symbolic := True;
2F09	FE00	=0130	CPI EOL ;
2F0B	C2152F	=0131	JNZ SYMR80 ;
		=0132	; ;
2F0E	78	=0133	MOV A,B ;
2F0F	327AEA	=0134	STA SYMBOLIC ;
2F12	C31B2F	=0135	JMP SYMR99 ; End
		=0136	SYMR80: ; Else
2F15	211C2F	=0137	LXI H,ERMSYM ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 174

LOC	OBJ	LINE	SOURCE STATEMENT
2F18	CD8119	=0138	CALL CMDERR ; CommandError(Symbolic);
		=0139 SYMR99:	;
2F1B	C9	=0140	RET ; Endproc SymbolicRoutine;
		=0141 ;	
		=0142 ERMSYM:	
		=0143 ;	
		=0144 ;	format:
		=0145 ;	
		=0146 ;	SYMBOLIC = ON OFF
		=0147 ;	
		=0148 ;	declaration:
		=0149 ;	
2F1C	53594D42	=0150	DB 'SYMBOLI', 'C'+MSB, '='+MSB, 'ON OFF', EndMsg
	4F4C49C3		
	BD4F4E7C		
	4F464600		
		=0151 ;	
		=0152 ;	END <DEBSYM.A85>
		10153	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 175

LOC	OBJ	LINE	SOURCE STATEMENT
		10154 ;	
		10155 ;	TRACE command routine:
		10156 ;	
		10157	\$INCLUDE(DEBTRACE.A85)
		=0158 ;	-----;
		=0159 ;	;
		=0160 ;	Trace buffer module of the UCT SABus debugger. ;
		=0161 ;	;
		=0162 ;	This module contains (in order) the following subroutines. ;
		=0163 ;	;
		=0164 ;	a) ROUND - Perform buffer pointer wrap-around. ;
		=0165 ;	b) TBINIT - Trace buffer - Initialise. ;
		=0166 ;	c) TBWRB - Trace buffer - Write byte. ;
		=0167 ;	d) TBWREX - Trace buffer - Write executed. ;
		=0168 ;	e) TBWRITE - Trace buffer - Write entry to trace buffer. ;
		=0169 ;	f) TBWROP - Trace buffer - Write opcode and operands. ;
		=0170 ;	g) TBWRW - Trace buffer - Write word. ;
		=0171 ;	;
		=0172 ;	-----;
		=0173	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 176

LOC	OBJ	LINE	SOURCE STATEMENT
		=0174 ;	=====*
		=0175 ;*	*
		=0176 ;*	Function : ROUND (Implement wrap-around) *
		=0177 ;*	Inputs : HL : Trace buffer pointer. *
		=0178 ;*	Outputs : HL : New trace buffer pointer (with wrap-around) *
		=0179 ;*	Destroys : A, H, L and flags. *
		=0180 ;*	Calls : None. *
		=0181 ;*	*
		=0182 ;*	Description : The length of the trace buffer is a power of two and *
		=0183 ;*	wrap-around can therefore be implemented by ANDing the pointer *
		=0184 ;*	with one less than the length of the buffer. *
		=0185 ;*	*
		=0186 ;	=====*
		=0187 ;	*
		=0188	ROUND: ; Function Round(Ptr:Word):Word;
2F2C	7C	=0189	MOV A,H ;
2F2D	E6FD	=0190	ANI HIGH TBMASK ;
2F2F	67	=0191	MOV H,A ; Round.High := Ptr and HIGH TraceBufferMask;
2F30	7D	=0192	MOV A,L ;
2F31	E6FF	=0193	ANI LOW TBMASK ;
2F33	6F	=0194	MOV L,A ; Round.Low := Ptr and LOW TraceBufferMask;
		=0195	;
2F34	C9	=0196	RET ; EndProc;
		=0197 ;	*
		=0198 ;	=====*
		=0199 ;*	*
		=0200 ;*	Function : TBINIT (Trace buffer initialise) *
		=0201 ;*	Inputs : None. *
		=0202 ;*	Outputs : None. (initialised trace buffer) *
		=0203 ;*	Destroys : H, L, A and flags. *
		=0204 ;*	Calls : None. *
		=0205 ;*	*
		=0206 ;*	Description : This routine initialises the trace buffer by: *
		=0207 ;*	*
		=0208 ;*	a) Resetting the free space to the size of the buffer. *
		=0209 ;*	b) Resetting the entry count to zero. *
		=0210 ;*	c) Resetting the trace buffer back and front pointers to the start *
		=0211 ;*	of the buffer. *
		=0212 ;*	*
		=0213 ;	=====*
		=0214 ;	*
		=0215	TBINIT: ; Procedure TraceBufferInitialise;
2F35	210002	=0216	LXI H,TBLEN ;
2F38	22C5EA	=0217	SHLD TBFREE ; TraceBufferFree := TraceBufferLength;
2F3B	210000	=0218	LXI H,0 ;
2F3E	22CBEA	=0219	SHLD TBCOUNT ; TraceBufferEntryCount := 0;
2F41	2100E8	=0220	LXI H,TBUFFER ;
2F44	22C7EA	=0221	SHLD TBFRONT ; TraceBufferFront := Adr(TraceBuffer);
		=0222	;
2F47	22C9EA	=0223	SHLD TBBACK ; TraceBufferBack := Adr(TraceBuffer);
		=0224	;
2F4A	C9	=0225	RET ; EndProc TraceBufferInitialise;
		=0226 ;	*
		=0227 ;	=====*
		=0228 ;*	*
		=0229 ;*	Function : TBINTR (Trace buffer interrupt) *
		=0230 ;*	Inputs : None. *
		=0231 ;*	Outputs : None. *
		=0232 ;*	Destroys : H, L, A and flags. *
		=0233 ;*	Calls : None. *
		=0234 ;*	*
		=0235 ;*	Description : This routine writes the initial information in *
		=0236 ;*	readiness for the execution information from an interrupt. *
		=0237 ;*	*
		=0238 ;*	NOTE: There is no opcode. *
		=0239 ;*	*
		=0240 ;	=====*
		=0241 ;	*
		=0242	TBINTR: ; Procedure TraceBufferInterrupt;
2F4B	21B8EA	=0243	LXI H,TBENTRY ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 177

LOC	OBJ	LINE	SOURCE STATEMENT
2F4E	3602	=0244	MVI M,2 ; TraceBufferEntry.Length := 2;
2F50	23	=0245	INX H ;
2F51	3600	=0246	MVI M,NOTEXEC ; TraceBufferEntry.ExType := NotExecuted;
2F53	22C3EA	=0247	SHLD TBPTR ;
2F56	C9	=0248	RET ; EndProc TraceBufferInterrupt;
		=0249 ;	
		=0250 ;	
		=0251 ;*	
		=0252 ;*	Function : TBWRB (Trace buffer write byte)
		=0253 ;*	Inputs : A : Byte to be written to the trace buffer.
		=0254 ;*	Outputs : None.
		=0255 ;*	Destroys : H, L, A and FLAGS.
		=0256 ;*	Calls : None.
		=0257 ;*	
		=0258 ;*	Description : This routine writes a single 8 bit byte to the trace
		=0259 ;*	buffer entry, in readiness for being written to the main trace
		=0260 ;*	buffer at the end of the instruction execution.
		=0261 ;*	
		=0262 ;*	NOTE: The trace buffer entry length is also updated.
		=0263 ;*	
		=0264 ;	
		=0265 ;	
		=0266	TBWRB: ; TraceBufferWriteByte (DataByte:Integer);
2F57	E5	=0267	PUSH H ;
2F58	2AC3EA	=0268	LHLD TBPTR ;
2F5B	77	=0269	MOV M,A ; Memory[TraceBufferPtr] := DataByte;
2F5C	23	=0270	INX H ;
2F5D	22C3EA	=0271	SHLD TBPTR ; TraceBufferPtr := TraceBufferPtr + 1;
		=0272 ;	
2F60	21B8EA	=0273	LXI H,TBELEN ;
2F63	34	=0274	INR M ; TraceBufferEntryLength :=
		=0275 ;	TraceBufferEntryLength + 1;
2F64	E1	=0276	POP H ;
2F65	C9	=0277	RET ; EndProc TraceBufferWriteByte;
		=0278 ;	
		=0279 ;	
		=0280 ;*	
		=0281 ;*	Function : TBWREX (Trace buffer write execute information)
		=0282 ;*	Inputs : A : Execute type.
		=0283 ;*	Outputs : None.
		=0284 ;*	Destroys : Flags.
		=0285 ;*	Calls : None.
		=0286 ;*	
		=0287 ;*	Description : This routine sets the execute type of the instruction
		=0288 ;*	in the trace buffer.
		=0289 ;*	
		=0290 ;	
		=0291 ;	
		=0292	TBWREX: ; Procedure TraceBufferWriteExecute
		=0293 ;	(ExecuteType:Byte);
2F66	E5	=0294	PUSH H ;
2F67	2AC3EA	=0295	LHLD TBPTR ;
2F6A	77	=0296	MOV M,A ; Memory[TraceBufferPtr] := ExecuteType;
2F6B	23	=0297	INX H ;
2F6C	22C3EA	=0298	SHLD TBPTR ; TraceBufferPointer := TraceBufferPointer
		=0299 ;	+ 1;
2F6F	E1	=0300	POP H ;
2F70	C9	=0301	RET ; EndProc TraceBufferWriteExecute;
		=0302 ;	
		=0303 ;	
		=0304 ;*	
		=0305 ;*	Function : TBWRITE (Trace buffer write)
		=0306 ;*	Inputs : A trace buffer in memory.
		=0307 ;*	Outputs : None.
		=0308 ;*	Destroys : A, B, C, D, E, H, L and flags.
		=0309 ;*	Calls : DCMP and ROUND.
		=0310 ;*	
		=0311 ;*	Description : This routine writes the trace buffer entry, which is
		=0312 ;*	build up during the instruction into the main trace buffer.
		=0313 ;*	If there is not enough space free in the trace buffer then old

LOC	OBJ	LINE	SOURCE STATEMENT
		=0314 ;*	entries are deleted untill there is enough space. *
		=0315 ;*	*
		=0316 ;	=====
		=0317 ;	
		=0318	TBWRITE: ; Procedure TraceBufferWrite;
2F71	3AB8EA	=0319	LDA TBELEN ;
2F74	5F	=0320	MOV E,A ;
2F75	1600	=0321	MVI D,0 ;
		=0322	TBWR10:
2F77	2AC5EA	=0323	LHLD TBFFREE ;
2F7A	CDBF1A	=0324	CALL DCMP ;
2F7D	D2A02F	=0325	JNC TBWR20 ; While TraceBufferFree < EntryLength do
		=0326	;
		=0327	PUSH D ; Begin
		=0328	;
2F81	2AC9EA	=0329	LHLD TBBACK ;
2F84	5E	=0330	MOV E,M ;
2F85	1600	=0331	MVI D,0 ;
2F87	19	=0332	DAD D ;
2F88	CD2C2F	=0333	CALL ROUND ;
2F8B	22C9EA	=0334	SHLD TBBACK ; TraceBufferBackPointer :=
		=0335	Round(TraceBufferBackPointer +
		=0336	BackEntryLength);
		=0337	;
2F8E	2AC5EA	=0338	LHLD TBFFREE ;
2F91	19	=0339	DAD D ;
2F92	22C5EA	=0340	SHLD TBFFREE ; TraceBufferFree :=
		=0341	TraceBufferFree + BackEntryLength;
		=0342	;
2F95	2ACBEA	=0343	LHLD TBFCOUNT ;
2F98	2B	=0344	DCX H ;
2F99	22CBEA	=0345	SHLD TBFCOUNT ; TraceBufferCount :=
		=0346	TraceBufferCount - 1;
		=0347	;
2F9C	D1	=0348	POP D ;
2F9D	C3772F	=0349	JMP TBWR10 ; End;
		=0350	TBWR20:
		=0351	;
2FA0	11B8EA	=0352	LXI D,TBENTRY ; EntryPtr := TraceBufferEntry;
2FA3	2AC5EA	=0353	LHLD TBFFREE ;
2FA6	44	=0354	MOV B,H ;
2FA7	4D	=0355	MOV C,L ;
2FA8	2AC7EA	=0356	LHLD TBFRONT ; TraceBufferPtr := TraceBufferFrontPointer;
2FAB	1A	=0357	LDAX D ;
		=0358	TBWR30:
2FAC	F5	=0359	PUSH PSW ; For I := 1 to EntryLength do
		=0360	;
		=0361	LDAX D ; Begin
		=0362	;
2FAE	77	=0363	MOV M,A ; Memory[TraceBufferPtr] := TempByte;
		=0364	;
2FAF	13	=0365	INX D ; EntryPtr := EntryPtr + 1;
2FB0	23	=0366	INX H ;
2FB1	CD2C2F	=0367	CALL ROUND ; TraceBufferPtr := Round
		=0368	(TraceBufferPtr + 1);
		=0369	;
2FB4	0B	=0370	DCX B ; TraceBufferFree := TraceBufferFree
		=0371	- 1;
		=0372	;
2FB5	F1	=0373	POP PSW ;
2FB6	3D	=0374	DCR A ;
2FB7	C2AC2F	=0375	JNZ TBWR30 ; End;
		=0376	;
2FBA	22C7EA	=0377	SHLD TBFRONT ;
		=0378	;
2FBD	60	=0379	MOV H,B ;
2FBE	69	=0380	MOV L,C ;
2FBF	22C5EA	=0381	SHLD TBFFREE ;
		=0382	;
2FC2	2ACBEA	=0383	LHLD TBFCOUNT ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 179

LOC	OBJ	LINE	SOURCE STATEMENT
2FC5	23	=0384	INX H ;
2FC6	22CBEA	=0385	SHLD TBCOUNT ; TraceBufferEntryCounter :=
		=0386	; TraceBufferEntryCounter + 1;
		=0387	; EndProc TraceBufferWrite;
2FC9	C9	=0388	RET ; EndProc TraceBufferWrite;
		=0389	;
		=0390	=====*
		=0391	;
		=0392	Function : TBWROP (Trace buffer write opcode and operands) *
		=0393	Inputs : HL : Program counter. *
		=0394	A : Byte count for the instruction. *
		=0395	Outputs : None. *
		=0396	Destroys : A, B, D, E, H, L and flags. *
		=0397	Calls : None. *
		=0398	;
		=0399	Description : This routine writes the initial information for *
		=0400	normal instructions to the trace buffer entry. The information *
		=0401	that is written is as follows: *
		=0402	;
		=0403	Offset Meaning Possible values *
		=0404	;
		=0405	TraceBufferEntry+0 : Length of entry : 0..11 *
		=0406	TraceBufferEntry+1 : PC (low) : 0..255 *
		=0407	TraceBufferEntry+2 : PC (high) : 0..255 *
		=0408	;
		=0409	TraceBufferEntry+3 : Opcode : 0..255 *
		=0410	TraceBufferEntry+4 : Operand : 0..255 (*) *
		=0411	TraceBufferEntry+5 : Operand : 0..255 (*) *
		=0412	;
		=0413	TraceBufferEntry+? : Execution Type : 0..11 *
		=0414	;
		=0415	NOTE: The execution type is initially set to NotExecuted. *
		=0416	;
		=0417	(*) One or both of these bytes may be missing (instruction dependant) *
		=0418	;
		=0419	=====*
		=0420	;
		=0421	TBWROP: ; Procedure TraceBufferWriteOpcode(Pc:Integer;
		=0422	; ByteCount:Byte);
2FCA	47	=0423	MOV B,A ;
2FCB	C604	=0424	ADI 4 ;
2FCD	EB	=0425	XCHG ;
		=0426	;
2FCE	21B8EA	=0427	LXI H,TBENTRY ; Ptr := TraceBufferEntry;
		=0428	;
2FD1	77	=0429	MOV M,A ; TraceBufferEntry[Ptr] := ByteCount + 4;
		=0430	;
2FD2	23	=0431	INX H ; Ptr := Ptr + 1;
		=0432	;
2FD3	73	=0433	MOV M,E ; TraceBufferEntry[Ptr] := LOW Pc;
		=0434	;
2FD4	23	=0435	INX H ; Ptr := Ptr + 1;
		=0436	;
2FD5	72	=0437	MOV M,D ; TraceBufferEntry[Ptr] := HIGH Pc;
		=0438	;
2FD6	23	=0439	INX H ; Ptr := Ptr + 1;
		=0440	;
		=0441	TBW005: ; For I := 1 to ByteCount do begin
2FD7	1A	=0442	LDAX D ;
2FD8	77	=0443	MOV M,A ; TraceBufferEntry[Ptr] :=
		=0444	Memory[Pc+I-1];
2FD9	13	=0445	INX D ;
2FDA	23	=0446	INX H ; Ptr := Ptr + 1;
2FDB	05	=0447	DCR B ;
2FDC	C2D72F	=0448	JNZ TBW005 ; End;
		=0449	;
2FDF	3600	=0450	MVI M,NOTEEXEC ; TraceBufferEntry[I] := NotExecuted;
		=0451	;
2FE1	22C3EA	=0452	SHLD TBPTR ; TraceBufferPointer := Ptr;
		=0453	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 180

LOC	OBJ	LINE	SOURCE STATEMENT
2FE4	C9	=0454	RET ; EndProc TraceBufferWriteOpcode;
		=0455 ;	
		=0456 ;=====*	
		=0457 ;*	
		=0458 ;* Function : TBWRW (Trace Buffer Write Word)	*
		=0459 ;* Inputs : HL : Word to write to the trace buffer)z	*
		=0460 ;* Outputs : None.	*
		=0461 ;* Destroys : A and flags.	*
		=0462 ;* Calls : None.	*
		=0463 ;*	*
		=0464 ;* Description : This routine write a word, in reverse order, into the *	*
		=0465 ;* trace buffer. The trace buffer entry length is also updated.	*
		=0466 ;*	*
		=0467 ;=====*	
		=0468 ;	
		=0469 TBWRW:	; TraceBufferWriteWord (DataWord:Integer);
2FE5	E5	=0470	PUSH H ;
2FE6	D5	=0471	PUSH D ;
2FE7	EB	=0472	XCHG ;
2FE8	2AC3EA	=0473	LHLD TBPTR ;
2FEB	73	=0474	MOV M,E ; Memory[TraceBufferPtr] := LOW DataWord;
		=0475	; ;
2FEC	23	=0476	INX H ; TraceBufferPtr := TraceBufferPtr + 1;
		=0477	; ;
2FED	72	=0478	MOV M,D ; Memory[TraceBufferPtr] := HIGH DataWord;
		=0479	; ;
2FEE	23	=0480	INX H ; TraceBufferPtr := TraceBufferPtr + 1;
2FEF	22C3EA	=0481	SHLD TBPTR ;
2FF2	21B8EA	=0482	LXI H,TBELEN ;
2FF5	34	=0483	INR M ;
2FF6	34	=0484	INR M ; TraceBufferLength := TraceBufferLength
		=0485	; + 2;
2FF7	D1	=0486	POP D ;
2FF8	E1	=0487	POP H ;
2FF9	C9	=0488	RET ; EndProc TraceBufferWriteWord;
		=0489 ;	
		=0490 ;	END <DEBTRACE.A85>
		10491	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
 UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 181

LOC	OBJ	LINE	SOURCE STATEMENT
		10492 ;	
2FFA		10493	ZZCODE EQU \$
		10494 ;	
		10495 ;	Debugger data area:
		10496 ;	
		10497	\$INCLUDE(DEBDATA.A85)
		=0498 ;	-----;
		=0499 ;	;
		=0500 ;	Data area used by the UCT SABus debugger. ;
		=0501 ;	;
		=0502 ;	The data area is divided into the following sections: ;
		=0503 ;	;
		=0504 ;	A: Trace buffer ;
		=0505 ;	B: Symbol variables ;
		=0506 ;	C: Number evaluation variables ;
		=0507 ;	D: Command line buffer and pointer ;
		=0508 ;	E: Flags ;
		=0509 ;	F: Range variables ;
		=0510 ;	G: ASCII output buffer and termination byte ;
		=0511 ;	H: REG command variables ;
		=0512 ;	I: CPU registers ;
		=0513 ;	J: Instruction information variables ;
		=0514 ;	K: Disassembly variables ;
		=0515 ;	L: Emulation variables and trigger condition variables ;
		=0516 ;	M: Console DATA and STATUS registers ;
		=0517 ;	N: Trace buffer data ;
		=0518 ;	O: Dynamic I/O routine ;
		=0519 ;	P: Dynamic direct execution routine ;
		=0520 ;	Q: Maps and other map variables ;
		=0521 ;	R: Debugger stack ;
		=0522 ;	;
		=0523 ;	-----;
		=0524	\$EJECT

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 182

LOC	OBJ	LINE	SOURCE STATEMENT
E800		=0525	ORG SDATA
		=0526 ;	
E800		=0527 AADATA EQU \$	
		=0528 ;	
		=0529 ;	
		=0530 ;	
		=0531 ;	A: Trace buffer:
		=0532 ;	
E800		=0533 TBUFFER: DS TBLN	; Trace buffer
		=0534 ;	
		=0535 ;	
		=0536 ;	
		=0537 ;	B: Symbol variables:
		=0538 ;	
EA00		=0539 SYMVAL: DS WORD	; Symbol value
EA02		=0540 SYMTYPE: DS BYTE	; Symbol type
EA03		=0541 SYMBUF: DS SYMBLEN	; Symbol buffer
EA0B		=0542 LABBUF: DS LABLEN	; Label buffer
		=0543 ;	
		=0544 ;	
		=0545 ;	
		=0546 ;	C: Number evaluation variables.
		=0547 ;	
EA11		=0548 DIGCOUNT: DS BYTE	; Digit counter
EA12		=0549 DIGIPTR: DS WORD	; Digit buffer input pointer
EA14		=0550 DIGOPTR: DS WORD	; Digit buffer output pointer
EA16		=0551 DIGBUF: DS 16	; Digit buffer
		=0552 ;	
		=0553 ;	
		=0554 ;	
		=0555 ;	D: Command line buffer and pointer.
		=0556 ;	
EA26		=0557 CLPTR: DS WORD	; Command line pointer
EA28		=0558 CMDLINE: DS 80	; Command line
		=0559 ;	
		=0560 ;	
		=0561 ;	
		=0562 ;	E: Flags:
		=0563 ;	
EA78		=0564 ERRFLAG: DS BYTE	; Error flag
EA79		=0565 EXITFLAG: DS BYTE	; Exit flag
EA7A		=0566 SYMBOLIC: DS BYTE	; Symbolic disassembly flag
		=0567 ;	
		=0568 ;	
		=0569 ;	
		=0570 ;	F: Range variables:
		=0571 ;	
EA7B		=0572 RBEGIN: DS WORD	; Range begin
EA7D		=0573 REND: DS WORD	; Range end
		=0574 ;	
		=0575 ;	
		=0576 ;	
		=0577 ;	G: ASCII output buffer and termination byte.
		=0578 ;	
EA7F		=0579 ASCBUF: DS 16*BYTE	; ASCII buffer
EA8F		=0580 ENDMARK: DS BYTE	; End of ASCII buffer marker
		=0581 ;	
		=0582 ;	
		=0583 ;	
		=0584 ;	H: REG command variables:
		=0585 ;	
EA90		=0586 REGTYPE: DS BYTE	; Register type (WORD or BYTE)
EA91		=0587 REGADR: DS WORD	; Register storage address
EA93		=0588 REGSTR: DS WORD	; Register string address
		=0589 ;	
		=0590 ;	
		=0591 ;	
		=0592 ;	I: CPU registers:
		=0593 ;	
		=0594 ;	The order of the following Registers is important.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 183

LOC	OBJ	LINE	SOURCE STATEMENT
		=0595 ;	
		=0596 ;	BC, DE, HL, PSW
		=0597 ;	
		=0598 REGISTERS:	; Registers
		=0599	;
		=0600 REGBC:	; BC register pair
		=0601	;
EA95		=0602 REGC:	DS BYTE ; C register
EA96		=0603 REGB:	DS BYTE ; B register
		=0604	;
		=0605 REGDE:	; DE register pair
		=0606	;
EA97		=0607 REGE:	DS BYTE ; E register
EA98		=0608 REGD:	DS BYTE ; D register
		=0609	;
		=0610 REGM:	; M register pair
		=0611 REGHL:	; HL register pair
		=0612	;
EA99		=0613 REGL:	DS BYTE ; L register
EA9A		=0614 REGH:	DS BYTE ; H register
		=0615	;
		=0616 REGPSW:	; Processor Status Word
		=0617	;
EA9B		=0618 FLAGS:	DS BYTE ; Flags register
EA9C		=0619 REGA:	DS BYTE ; A register
		=0620	;
EA9D		=0621 REGPC:	DS WORD ; Program counter
EA9F		=0622 REGSP:	DS WORD ; Stack pointer
		=0623	;
EAA1		=0624 IMASK:	DS BYTE ; Interrupt mask
EAA2		=0625 ENABINT:	DS BYTE ; Enable interrupt flag
		=0626 ;	
		=0627 ;	-----
		=0628 ;	
		=0629 ;	J: Instruction information variables.
		=0630 ;	
EAA3		=0631 BYTECOUNT:	DS BYTE ; Number of bytes in the instruction
EAA4		=0632 OPERAND:	DS BYTE ; Operand information
EAA5		=0633 EMINFO:	DS BYTE ; Emulation/Execution information
EAA6		=0634 STRADR:	DS WORD ; Address of instruction string
EAA8		=0635 OPCODE:	DS BYTE ; Opcode
		=0636 ;	
		=0637 ;	-----
		=0638 ;	
		=0639 ;	K: Disassembly variables:
		=0640 ;	
EAA9		=0641 DISPC:	DS WORD ; Disassembly Program counter
EABAB		=0642 LASTDIS:	DS WORD ; Last instruction disassembled
EABAD		=0643 LINES:	DS BYTE ; Number of lines on the screen
		=0644 ;	
		=0645 ;	-----
		=0646 ;	
		=0647 ;	L: Emulation variables and trigger condition variables.
		=0648 ;	
EAAE		=0649 MYSP:	DS WORD ; Debugger Stack pointer
EAB0		=0650 STEPCOUNT:	DS WORD ; Single step count
EAB2		=0651 STATUS:	DS BYTE ; Debugger status
EAB3		=0652 TRIG:	DS BYTE ; Trigger type
EAB4		=0653 TRIGWORD:	DS WORD ; Trigger word
		=0654 ;	
		=0655 ;	-----
		=0656 ;	
		=0657 ;	M: Console DATA and STATUS registers.
		=0658 ;	
EAB6		=0659 CSTATUS:	DS BYTE ; Console STATUS
EAB7		=0660 KEYVAL:	DS BYTE ; Key value (Console data)
		=0661 ;	
		=0662 ;	-----
		=0663 ;	
		=0664 ;	N: Trace buffer data:

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB

PAGE 184

```

LOC  OBJ      LINE      SOURCE STATEMENT
                                =0665 ;
                                =0666 TBENTRY:                ; Start of trace buffer entry
                                =0667                          ;
EAB8   =0668 TBELEN:    DS    BYTE                ; Trace buffer length byte
EAB9   =0669           DS    10                    ; Trace buffer
                                =0670 ;
EAC3   =0671 TBPTR:     DS    WORD                ; Trace buffer pointer
EAC5   =0672 TBFREE:    DS    WORD                ; Trace buffer free space
EAC7   =0673 TBFRTONT:  DS    WORD                ; Trace buffer front pointer
EAC9   =0674 TBBACK:    DS    WORD                ; Trace buffer back pointer
EACB   =0675 TBCOUNT:   DS    WORD                ; Trace buffer entry counter
                                =0676 ;
                                =0677 ;-----
                                =0678 ;
                                =0679 ;      O: Dynamic I/O routine.
                                =0680 ;
EACD   =0681 PORTNUM:   DS    BYTE                ; Port number
                                =0682 ;
EACE   =0683 DYNIO:     DS    3                    ; Dynamic I/O routine
                                =0684 ;
                                =0685 ;-----
                                =0686 ;
                                =0687 ;      P: Dynamic direct execution routine.
                                =0688 ;
                                =0689 DECROU:                ; Decide routine and direct execution routine
EAD1   =0690 ROUTINE:   DS    6                    ;
                                =0691 ;
                                =0692 ;-----
                                =0693 ;
                                =0694 ;      Q: Maps and other map variables:
                                =0695 ;
EB00   =0696           ORG    ( (HIGH $) + 1) SHL 8
                                =0697 MAPS:
                                =0698 ;
EB00   =0699 IOMAP:     DS    MAPLEN                ; IO map
EC00   =0700 MEMMAP:    DS    MAPLEN                ; Memory map
                                =0701 ;
ED00   =0702 MAPVAL:    DS    BYTE                ; Map value
ED01   =0703 MAPADR:    DS    WORD                ; Map address
                                =0704 ;
                                =0705 ;-----
                                =0706 ;
                                =0707 ;      R: Debugger stack:
                                =0708 ;
ED03   =0709           DS    STACKL                ; Stack space
                                =0710 NEWSP:                ; Initial stack pointer
                                =0711 ;
EDCB   =0712 ZZDATA     EQU    $
                                =0713 ;
                                =0714 ;      END <DEBDATA.A85>
                                10715 $LIST
                                10716 ;
1000   10717           END      DEB

```

0 error(s) detected

User symbols

```

AACODE 1000 A  AADATA E800 A  ACK    0006 A  ADDBC  21C6 A  ADDLEN 2C3F A  ADJ05  2099 A  ADJUST 207B A
ASCBUF EA7F A  ASCII  007F A  ASUB   001A A  BASEF  1DF9 A  BCMP   1AB9 A  BELL   0007 A  BITSPB 0008 A
BITSPW 0010 A  BREAK  195E A  BRK05  196E A  BRK10  1976 A  BS     0008 A  BYTD05 28B6 A  BYTD10 28C8 A
BYTD20 28DE A  BYTD30 28E9 A  BYTD35 28F2 A  BYTD40 28FE A  BYTD99 291B A  BYTE    0001 A  BYTECO EAA3 A
BYTEDU 28A3 A  BYTERO 291C A  BYTEVA 0000 A  BYTR10 2933 A  BYTR15 2941 A  BYTR80 295C A  BYTR99 2962 A
CAN     0018 A  CHECK  1E4C A  CHKK10 21F2 A  CHKK15 220D A  CHKK17 2210 A  CHKK20 2210 A  CHKKEY 21DA A
CHKMRD 103E A  CHKMR  1066 A  CHKPRD 108E A  CHKPWR 10B1 A  CI     F013 A  CITEST F00A A  CLPTR  EA26 A
CLRLIN 004B A  CLMSG  25D3 A  CLRS   004A A  CLRSCN 2039 A  CMDE05 1982 A  CMDE10 198E A  CMDERM 19A1 A
CMDERR 1981 A  CMDLIN EA28 A  CMTAB  1B08 A  CMRD10 1059 A  CMWR10 1081 A  CO     F01C A  CODATA 0000 A
CONMA  002C A  CONH05 197E A  CONHEX 1977 A  COPB05 221A A  COPS10 1BB1 A  COPS20 1BC7 A
COPYBY 2211 A  COPYEN 23D6 A  COPYSY 1BA3 A  COSTAT 0001 A  CPRD10 10A7 A  CPWR10 10CA A  CR     000D A

```

DEB LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus symbolic debugger

13:46:17 14-APR-88

DEB PAGE 185

[illegible]

H-199

APPENDIX I

DIAG LISTING

The following pages contain an index of the symbols used in the diagnostics source listing. In order to keep the index reasonably short, the local symbols, those defined and referenced within a subroutine, are not listed.

As well as the symbol name and page reference, a symbol type is also given. The possible symbol types are:

- a) CONSTANT (in this case a value is also given)
- b) SUBROUTINE
- c) BYTE ARRAY (in this case the length is also given)
- f) TABLE
- g) MESSAGE

NOTE: The page reference given in the index below is the page on which the symbol is first defined.

This appendix also contains the command file that should be invoked in order to assemble, link and locate the UCT SABus system diagnostics. The command file was written to run under RSX 11M version 4.0 or higher.

<u>SYMBOL</u>	<u>DESCRIPTION</u>	<u>PAGE REFERENCE</u>
?EPROM	SUBROUTINE.....	I-62
?ESCSP	SUBROUTINE.....	I-14
ACK	CONSTANT, VALUE = 006H.....	I-11
ACTM	MESSAGE.....	I-27
AOCMD	CONSTANT, VALUE = 10001011B.....	I-36
AOMSG	MESSAGE.....	I-40
ASCBIN	CONSTANT, VALUE = 0F031H.....	I-77
ASCBUF	BYTE ARRAY, LENGTH = 16.....	I-63
ASCEOT	BYTE VARIABLE.....	I-63
ASUB	CONSTANT, VALUE = 01AH.....	I-11
BELL	CONSTANT, VALUE = 007H.....	I-11
BOCMD	CONSTANT, VALUE = 10011001B.....	I-36
BOMSG	MESSAGE.....	I-40
BOTCX	BYTE VARIABLE.....	I-75
BOTCY	BYTE VARIABLE.....	I-76

BREAK	CONSTANT, VALUE = 0F025H.....	I-77
BS	CONSTANT, VALUE = 008H.....	I-11
BUFWRT	SUBROUTINE.....	I-62
BYTE	CONSTANT, VALUE = 1.....	I-22
CAN	CONSTANT, VALUE = 018H.....	I-11
CI	CONSTANT, VALUE = 0F013H.....	I-77
CITEST	CONSTANT, VALUE = 0F00AH.....	I-77
CLRL	SUBROUTINE.....	I-16
CLRLC	CONSTANT, VALUE = 'K'.....	I-12
CLRS	SUBROUTINE.....	I-15
CLRSC	CONSTANT, VALUE = 'J'.....	I-12
CLRTOS	SUBROUTINE.....	I-16
CMAx	BYTE VARIABLE.....	I-76
CMIN	BYTE VARIABLE.....	I-76
CO	CONSTANT, VALUE = 0F01CH.....	I-77
COCMD	CONSTANT, VALUE = 10010010B.....	I-36
CODATA	CONSTANT, VALUE = 0.....	I-77
COMSG	MESSAGE.....	I-40
COSTAT	CONSTANT, VALUE = 1.....	I-77
COTEST	SUBROUTINE.....	I-77
CR	CONSTANT, VALUE = 00DH.....	I-11
CRLF	SUBROUTINE.....	I-15
CTRLMASK	CONSTANT, VALUE = 00011111B.....	I-11
CTRLQ	CONSTANT, VALUE = CTRLMASK AND 'Q'.....	I-11
CTRLS	CONSTANT, VALUE = CTRLMASK AND 'S'.....	I-11
CTRLZ	CONSTANT, VALUE = CTRLMASK AND 'Z'.....	I-11
DATA5	CONSTANT, VALUE = 00B SHL 2.....	I-35
DATA6	CONSTANT, VALUE = 01B SHL 2.....	I-35
DATA7	CONSTANT, VALUE = 10B SHL 2.....	I-35
DATA8	CONSTANT, VALUE = 11B SHL 2.....	I-35
DB110	CONSTANT, VALUE = 873.....	I-35
DB1200	CONSTANT, VALUE = 80.....	I-35
DB1800	CONSTANT, VALUE = 53.....	I-35
DB2400	CONSTANT, VALUE = 40.....	I-35
DB300	CONSTANT, VALUE = 320.....	I-35
DB3600	CONSTANT, VALUE = 27.....	I-35
DB4800	CONSTANT, VALUE = 20.....	I-35
DB600	CONSTANT, VALUE = 160.....	I-35
DB7200	CONSTANT, VALUE = 13.....	I-35
DB9600	CONSTANT, VALUE = 10.....	I-36
DC1	CONSTANT, VALUE = 011H.....	I-11
DC2	CONSTANT, VALUE = 012H.....	I-11
DC3	CONSTANT, VALUE = 013H.....	I-11
DC4	CONSTANT, VALUE = 014H.....	I-11
DCMP	CONSTANT, VALUE = 0F043H.....	I-77
DCMPI	CONSTANT, VALUE = 0F046H.....	I-77
DCMSG	MESSAGE.....	I-74
DCURP	SUBROUTINE.....	I-67
DEBLK	SUBROUTINE.....	I-61
DEL	CONSTANT, VALUE = 07FH.....	I-12

DEL75	BYTE VARIABLE.....	I-27
DELAY	SUBROUTINE.....	I-15
DIAG	MAIN PROGRAM.....	I-12
DIV1	CONSTANT, VALUE = 01B SHL 0.....	I-35
DIV16	CONSTANT, VALUE = 10B SHL 0.....	I-35
DIV64	CONSTANT, VALUE = 11B SHL 0.....	I-35
DLE	CONSTANT, VALUE = 010H.....	I-11
DNCMD	CONSTANT, VALUE = 'B'.....	I-12
DRAW	SUBROUTINE.....	I-68
DSR	CONSTANT, VALUE = 10000000B.....	I-35
DSREM	MESSAGE.....	I-50
DTR	CONSTANT, VALUE = 00000010B.....	I-35
DYNINIT	SUBROUTINE.....	I-40
ECHO	CONSTANT, VALUE = 0F034H.....	I-77
EM	CONSTANT, VALUE = 019H.....	I-11
EMPTY	SUBROUTINE.....	I-63
ENMSG	MESSAGE.....	I-60
ENQ	CONSTANT, VALUE = 005H.....	I-11
EOT	CONSTANT, VALUE = 004H.....	I-11
EOTABLE	CONSTANT, VALUE = 0.....	I-12
EPDATA	CONSTANT, VALUE = MSCARD+0.....	I-34
EPNUM	CONSTANT, VALUE = MSCARD+3.....	I-34
ERESET	CONSTANT, VALUE = 00010000B.....	I-35
ERMASK	CONSTANT, VALUE = 00111000B.....	I-35
ERRREP	SUBROUTINE.....	I-38
ESC	CONSTANT, VALUE = 01BH.....	I-11
ETB	CONSTANT, VALUE = 017H.....	I-11
ETC1	CONSTANT, VALUE = 038H.....	I-34
ETC2	CONSTANT, VALUE = 0CBH.....	I-34
ETX	CONSTANT, VALUE = 003H.....	I-11
EVENPARITY	CONSTANT, VALUE = 11B SHL 4.....	I-34
FALSE	CONSTANT, VALUE = 0.....	I-22
FDCMP	SUBROUTINE.....	I-14
FF	CONSTANT, VALUE = 00CH.....	I-11
FRMERR	CONSTANT, VALUE = 00100000B.....	I-35
FS	CONSTANT, VALUE = 01CH.....	I-11
GET1NM	CONSTANT, VALUE = 0F055H.....	I-77
GET2NM	CONSTANT, VALUE = 0F058H.....	I-77
GET3NM	CONSTANT, VALUE = 0F05BH.....	I-77
GETCM	CONSTANT, VALUE = 0F04FH.....	I-77
GETECHO	CONSTANT, VALUE = 0F02BH.....	I-77
GETHX	CONSTANT, VALUE = 0F052H.....	I-77
GETUC	CONSTANT, VALUE = 0F028H.....	I-77
GETWD	CONSTANT, VALUE = 0F02EH.....	I-77
GOTO	SUBROUTINE.....	I-25
GOTOC	CONSTANT, VALUE = 'Y'.....	I-12
GS	CONSTANT, VALUE = 01DH.....	I-11
HEXH	MESSAGE.....	I-61
HOME	SUBROUTINE.....	I-16
HOMECL	CONSTANT, VALUE = 'H'.....	I-12

HT	CONSTANT, VALUE = 009H.....	I-11
HUNTM	CONSTANT, VALUE = 10000000B.....	I-35
ICOLP	CONSTANT, VALUE = 57.....	I-27
ICOML	MESSAGE.....	I-27
IFLAGS	BYTE ARRAY, LENGTH = 3.....	I-27
IGOTO	SUBROUTINE.....	I-25
IGOTOM	MESSAGE.....	I-27
IN1	BYTE VARIABLE.....	I-41
IN2	BYTE VARIABLE.....	I-41
INDIRECT	SUBROUTINE.....	I-14
INP1	BYTE VARIABLE.....	I-41
INP2	BYTE VARIABLE.....	I-41
INT75	SUBROUTINE.....	I-24
INVSTOPBIT	CONSTANT, VALUE = 00B SHL 6.....	I-34
IOBASE	CONSTANT, VALUE = 0.....	I-34
IPOSDAT	TABLE.....	I-27
IPTTEST	SUBROUTINE.....	I-38
IRESET	CONSTANT, VALUE = 01000000B.....	I-35
IRMSG	MESSAGE.....	I-26
IROWP	CONSTANT, VALUE = 13.....	I-27
ITEST	SUBROUTINE.....	I-23
K	CONSTANT, VALUE = 1024.....	I-22
LF	CONSTANT, VALUE = 00AH.....	I-11
LFCMD	CONSTANT, VALUE = 'D'.....	I-12
LMAX	BYTE VARIABLE.....	I-76
LMIN	BYTE VARIABLE.....	I-76
LSBYTE	CONSTANT, VALUE = MSCARD+1.....	I-34
MAINMENU	MESSAGE.....	I-17
MAXX	CONSTANT, VALUE = 79.....	I-12
MAXY	CONSTANT, VALUE = 23.....	I-12
MCTAB	TABLE.....	I-20
MINX	CONSTANT, VALUE = 0.....	I-12
MINY	CONSTANT, VALUE = 0.....	I-12
MSBYTE	CONSTANT, VALUE = MSCARD+2.....	I-34
MSCARD	CONSTANT, VALUE = 20H.....	I-34
MSCM1	MESSAGE.....	I-59
MSCM2	MESSAGE.....	I-59
MSCMSG	MESSAGE.....	I-58
MSCTAB	TABLE.....	I-63
MSCTM1	MESSAGE.....	I-57
MTDISP	SUBROUTINE.....	I-31
MTERM	MESSAGE.....	I-32
MTEST	SUBROUTINE.....	I-30
MTGOTO	MESSAGE.....	I-32
MTMSG	MESSAGE.....	I-31
NAK	CONSTANT, VALUE = 015H.....	I-11
NAMEA	CONSTANT, VALUE = 3.....	I-34
NEXTC	SUBROUTINE.....	I-67
NMOUT	CONSTANT, VALUE = 0F03AH.....	I-77
NOACTM	MESSAGE.....	I-27

NOPARITY	CONSTANT, VALUE = 00B SHL 4.....	I-34
NUL	CONSTANT, VALUE = 000H.....	I-11
ODDPARITY	CONSTANT, VALUE = 01B SHL 4.....	I-34
OFFSET	CONSTANT, VALUE = 20H.....	I-12
OKMSG	MESSAGE.....	I-52
ONEANDAHSB	CONSTANT, VALUE = 10B SHL 6.....	I-34
ONESTOPBIT	CONSTANT, VALUE = 01B SHL 6.....	I-34
OUNIT	SUBROUTINE.....	I-50
OUMODE	CONSTANT, VALUE = 04EH.....	I-35
OUT1	BYTE VARIABLE.....	I-41
OUTMSG	CONSTANT, VALUE = 0F03DH.....	I-77
OUTP	BYTE VARIABLE.....	I-41
OVRRERR	CONSTANT, VALUE = 00010000B.....	I-35
PARERR	CONSTANT, VALUE = 00001000B.....	I-35
PATMSG	MESSAGE.....	I-41
PATPRT	SUBROUTINE.....	I-39
PDELAY	SUBROUTINE.....	I-39
PORTA	CONSTANT, VALUE = PPIBASE+0.....	I-34
PORTB	CONSTANT, VALUE = PPIBASE+1.....	I-34
PORTC	CONSTANT, VALUE = PPIBASE+2.....	I-34
PPIBASE	CONSTANT, VALUE = IOBASE+0CH.....	I-34
PPICONT	CONSTANT, VALUE = PPIBASE+3.....	I-34
PRTAT	SUBROUTINE.....	I-71
PRTNAME	SUBROUTINE.....	I-62
PRVAL	CONSTANT, VALUE = 0F037H.....	I-77
PTAB	SUBROUTINE.....	I-14
PTEMSG	MESSAGE.....	I-39
PTEST	SUBROUTINE.....	I-37
PTMSG	MESSAGE.....	I-40
PTOK	MESSAGE.....	I-39
PVTOUT	SUBROUTINE.....	I-71
QUOTE	CONSTANT, VALUE = 027H.....	I-12
READB	SUBROUTINE.....	I-62
RETIN1	BYTE VARIABLE.....	I-41
RETIN2	BYTE VARIABLE.....	I-41
RETOUT	BYTE VARIABLE.....	I-41
REVLf	CONSTANT, VALUE = 'I'.....	I-12
RS	CONSTANT, VALUE = 01EH.....	I-11
RST05	SUBROUTINE.....	I-47
RST10	SUBROUTINE.....	I-47
RST55A	CONSTANT, VALUE = 2CH.....	I-34
RST65A	CONSTANT, VALUE = 34H.....	I-34
RST75A	CONSTANT, VALUE = 3CH.....	I-34
RTCMD	CONSTANT, VALUE = 'C'.....	I-12
RTEST	SUBROUTINE.....	I-46
RTM01	MESSAGE.....	I-51
RTM10	MESSAGE.....	I-51
RTM15	MESSAGE.....	I-51
RTM20	MESSAGE.....	I-52
RTS	CONSTANT, VALUE = 00100000B.....	I-35

RXENAB	CONSTANT, VALUE = 00000100B.....	I-35
RXRDY	CONSTANT, VALUE = 00000010B.....	I-35
SBREAK	CONSTANT, VALUE = 00001000B.....	I-35
SCROLL	SUBROUTINE.....	I-67
SDMSG2	MESSAGE.....	I-60
SDTEST	SUBROUTINE.....	I-59
SEARCH	SUBROUTINE.....	I-14
SFTEST	MESSAGE.....	I-58
SI	CONSTANT, VALUE = 00FH.....	I-11
SO	CONSTANT, VALUE = 00EH.....	I-11
SOH	CONSTANT, VALUE = 001H.....	I-11
SPACE	CONSTANT, VALUE = 020H.....	I-12
STARTC	BYTE VARIABLE.....	I-75
STEST	MESSAGE.....	I-57
STX	CONSTANT, VALUE = 002H.....	I-11
SYN	CONSTANT, VALUE = 016H.....	I-11
SYNCMODE	CONSTANT, VALUE = 00B SHL 0.....	I-35
SYNDET	CONSTANT, VALUE = 01000000B.....	I-35
T0DATA	CONSTANT, VALUE = TBASE+0.....	I-34
T1DATA	CONSTANT, VALUE = TBASE+1.....	I-34
T2DATA	CONSTANT, VALUE = TBASE+2.....	I-34
TAB	CONSTANT, VALUE = 009H.....	I-11
TBASE	CONSTANT, VALUE = IOBASE+08H.....	I-34
TCONT	CONSTANT, VALUE = TBASE+3.....	I-34
TESTB	SUBROUTINE.....	I-31
TIM01M	MESSAGE.....	I-49
TIM10M	MESSAGE.....	I-49
TIMER0	CONSTANT, VALUE = 0 SHL 6.....	I-35
TIMER1	CONSTANT, VALUE = 1 SHL 6.....	I-35
TIMER2	CONSTANT, VALUE = 2 SHL 6.....	I-35
TMODE	CONSTANT, VALUE = 00110110B.....	I-35
TMODE0	CONSTANT, VALUE = TMODE+TIMER0.....	I-35
TMODE1	CONSTANT, VALUE = TMODE+TIMER1.....	I-35
TMODE2	CONSTANT, VALUE = TMODE+TIMER2.....	I-35
TOPCX	BYTE VARIABLE.....	I-75
TOPCY	BYTE VARIABLE.....	I-75
TRANSP	CONSTANT, VALUE = 0F049H.....	I-77
TRUE	CONSTANT, VALUE = (NOT FALSE) AND 0FFH.....	I-22
TWOSTOPBIT	CONSTANT, VALUE = 11B SHL 6.....	I-34
TXEMP	CONSTANT, VALUE = 00000100B.....	I-35
TXENAB	CONSTANT, VALUE = 00000001B.....	I-35
TXRDY	CONSTANT, VALUE = 00000001B.....	I-35
TXRDY	CONSTANT, VALUE = 00000001B.....	I-77
U0BASE	CONSTANT, VALUE = IOBASE+02H.....	I-34
U0CONT	CONSTANT, VALUE = U0STAT.....	I-34
U0DATA	CONSTANT, VALUE = U0BASE+0.....	I-34
U0STAT	CONSTANT, VALUE = U0BASE+1.....	I-34
U0TEST	CONSTANT, VALUE = 0F00DH.....	I-77
U1BASE	CONSTANT, VALUE = IOBASE+04H.....	I-34
U1CONT	CONSTANT, VALUE = U1STAT.....	I-34

UIBASE	CONSTANT, VALUE = UIBASE+0.....	I-34
UISTAT	CONSTANT, VALUE = UIBASE+1.....	I-34
UITEST	CONSTANT, VALUE = 0F010H.....	I-77
UCMD	CONSTANT, VALUE = 015H.....	I-35
UDUMMY	CONSTANT, VALUE = 02H.....	I-34
UERR0M	MESSAGE.....	I-49
UERR1M	MESSAGE.....	I-49
UI0	CONSTANT, VALUE = 0F016H.....	I-77
UI0WT	SUBROUTINE.....	I-51
UI1	CONSTANT, VALUE = 0F019H.....	I-77
UI1WT	SUBROUTINE.....	I-51
UINIT	SUBROUTINE.....	I-50
UMODE	CONSTANT, VALUE = 07EH.....	I-35
UO0	CONSTANT, VALUE = 0F01FH.....	I-77
UO1	CONSTANT, VALUE = 0F022H.....	I-77
UPCMD	CONSTANT, VALUE = 'A'.....	I-12
UPDATE	SUBROUTINE.....	I-25
URESET	CONSTANT, VALUE = 40H.....	I-34
US	CONSTANT, VALUE = 01FH.....	I-11
VALDL	CONSTANT, VALUE = 0F04CH.....	I-77
VER01M	MESSAGE.....	I-49
VER10M	MESSAGE.....	I-49
VINIT	SUBROUTINE.....	I-66
VT	CONSTANT, VALUE = 00BH.....	I-11
VTCHAR	BYTE VARIABLE.....	I-75
VTCLRS	SUBROUTINE.....	I-72
VTCLRF	SUBROUTINE.....	I-72
VTEST	SUBROUTINE.....	I-66
VTMSG1	MESSAGE.....	I-73
VTMSG2	MESSAGE.....	I-74
VTO0	MESSAGE.....	I-74
VTO1	MESSAGE.....	I-74
VTOC	MESSAGE.....	I-74
VTOUT	BYTE ARRAY, LENGTH = 4.....	I-75
VTOUTMSG	SUBROUTINE.....	I-72
VTOUTR	BYTE VARIABLE.....	I-75
VTSET0	SUBROUTINE.....	I-72
VTSET1	SUBROUTINE.....	I-73
VTSETC	SUBROUTINE.....	I-72
VTTAB	TABLE.....	I-75
VTU	MESSAGE.....	I-74
WAIT	SUBROUTINE.....	I-15
WAITMSG	MESSAGE.....	I-17
WORD	CONSTANT, VALUE = 2.....	I-22
YPOS	BYTE VARIABLE.....	I-75

```
.ENABLE SUBSTITUTION
.DISABLE LOWERCASE
;
.NOOK:
.ASKS ACTION (A)ssemble, (L)ink/locate or (B)oth?
.IF ACTION = "A" .GOTO ACTOK
.IF ACTION = "L" .GOTO ACTOK
.IF ACTION = "B" .GOTO ACTOK
.GOTO NOOK
.ACTOK:
.IF ACTION = "L" .GOTO LINKON
;
; This command file assembles, links and locates the SABus kit diagnostics.
;
; Enter the letters corresponding to the modules to be assembled
; <return> assembles all modules.
;
; D - DIAG      I - DIAGINTER  M - DIAGMEM      P - DIAGPAR
; R - DIAGRS232 S - DIAGMSC    V - DIAGVDU
;
.RETRY2:
.ASKS [1.:7.: "DIMPRSV"] MODS Modules to assemble?
.SETF DIAG
.SETF INTER
.SETF MEM
.SETF PAR
.SETF RS232
.SETF MSC
.SETF VDU
.SETN MODCT <STRLEN>
.GMODS:
.SETS TMOD MODS[MODCT:MODCT]
.TEST "DIMPRSV" TMOD
.IF <STRLEN> <> 0 .GOTO MODOK
;
; Don't know module "'TMOD'"
;
.GOTO RETRY2
.MODOK:
.IF TMOD = "D" .SETT DIAG
.IF TMOD = "I" .SETT INTER
.IF TMOD = "M" .SETT MEM
.IF TMOD = "P" .SETT PAR
.IF TMOD = "R" .SETT RS232
.IF TMOD = "S" .SETT MSC
.IF TMOD = "V" .SETT VDU
.DEC MODCT
.IF MODCT <> 0 .GOTO GMODS
.;
.ASK FULIST Full assembler listing
.IFT FULIST .SETS SW "/FU/SY"
.IFF FULIST .SETS SW "/-FU/-SY"
.;
.;      Start assembly of selected modules:
.;
.IFF DIAG .GOTO NODIAG
;
;      Assemble the main module DIAG.A85
;
PIP DIAG.O85;*/DE/NM
AS5 DIAG/-DB,DIAG'SW'=DIAG
```

```

.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NODIAG:
.IFF INTER .GOTO NINTER
;
;   Assemble the interrupt test routines DIAGINTER.A85
;
PIP DIAGINTER.O85;*/DE/NM
AS5 DIAGINTER/-DB,DIAGINTER'SW'=DIAGINTER
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NINTER:
.IFF MEM .GOTO NOMEM
;
;   Assemble the memory test module DIAGMEM.A85
;
PIP DIAGMEM.O85;*/DE/NM
AS5 DIAGMEM/-DB,DIAGMEM'SW'=DIAGMEM
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NOMEM:
.IFF PAR .GOTO NOPAR
;
;   Assemble the parallel test module DIAGPAR.A85
;
PIP DIAGPAR.O85;*/DE/NM
AS5 DIAGPAR/-DB,DIAGPAR'SW'=DIAGPAR
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NOPAR:
.IFF RS232 .GOTO NRS232
;
;   Assemble the RS-232 test routine DIAGRS232.A85
;
PIP DIAGRS232.O85;*/DE/NM
AS5 DIAGRS232/-DB,DIAGRS232'SW'=DIAGRS232
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NRS232:
.IFF MSC .GOTO NOMSC
;
;   Assemble the mass storage card test routine DIAGMSC.A85
;
PIP DIAGMSC.O85;*/DE/NM
AS5 DIAGMSC/DB,DIAGMSC'SW'=DIAGMSC
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NOMSC:
.IFF VDU .GOTO NOVDU
;
;   Assemble the VDU test routines DIAGVDU.A85
;
PIP DIAGVDU.O85;*/DE/NM
AS5 DIAGVDU/-DB,DIAGVDU'SW'=DIAGVDU
.IF <EXSTAT> <> 1 .GOTO AFAIL
.NOVDU:
;
;   Link all modules and system library.
;
.LINKON:
.;

```

```

.OPEN DDD.CMD
.ENABLE DATA
DIAG.REL,DIAG=DIAG
DIAGRS232,DIAGMSC,DIAGVDU,DIAGINTER,DIAGPAR,DIAGMEM
SABMON
.DISABLE DATA
.CLOSE DDD.CMD
LK5 @DDD
.IF <EXSTAT> <> 1 .GOTO LFAIL
PIP DDD.CMD;*/DE/NM
;
;       Locate diagnostics
;
.OPEN DDD.CMD
.ENABLE DATA
DIAG.ABS,DIAG=DIAG.REL
PURGE
ORDER=CODE,DATA,STACK
LIMIT=0:EFFF
CODE=1000
//
.DISABLE DATA
.CLOSE DDD.CMD
LC5 @DDD.CMD
.IF <EXSTAT> <> 1 .GOTO LCFAIL
PIP DDD.CMD;*/DE/NM
;
;       Convert absolute object file to downloadable hex file
;
OH5 DIAG.H85=DIAG.ABS
.;
.;       Clean up by removing absolute object and relocatable object.
.;
PIP DIAG.REL;*,DIAG.ABS;*/DE/NM
.;
.EXIT
.;
.;-----
.;
.;       Error exit points
.;
.AFAIL:
;
; Assembly failed.
;
.EXIT
.;
.LFAIL:
;
; link failed.
;
.EXIT
.;
.LCFAIL:
;
; Locate failed.
;
.EXIT

```

```

LOC   OBJ       LINE      SOURCE STATEMENT
                                1 $TITLE('UCT SABus KITS diagnostics')
                                2 $PAGELENGTH(75)
                                3 ;
                                4 ;-----;
                                5 ;
                                6 ;                      SABus diagnostics.
                                7 ;                      -----
                                8 ;
                                9 ;          The following files are part of the SABus diagnostics.
                                10 ;
                                11 ;      1) DIAG.A85      - This file (the main program)
                                12 ;      ii) DIAGRS232.A85 - Serial port test routines
                                13 ;      iii) DIAGPAR.A85 - Parallel port test routine
                                14 ;      iv) DIAGMEM.A85  - Memory test routine
                                15 ;      v) DIAGINTER.A85 - Interrupt test routine
                                16 ;      vi) DIAGMSC.A85  - Mass storage card test routine
                                17 ;      vii) DIAGVDU.A85 - Display test routines
                                18 ;
                                19 ;-----;
                                20 ;
                                21          NAME      DIAG
                                22 ;
                                23 $INCLUDE(ASCII.EQU)
                                24 ;
                                25 ;-----;
                                26 ;
001F   = 27 CTRLMASK EQU      00011111B      ; Control mask.
                                28 ;
                                29 ;          ASCII characters:
                                30 ;
0000   = 31 NUL      EQU      000H          ; ASCII null character
0001   = 32 SOH      EQU      001H          ; ASCII Start of header character
0002   = 33 STX      EQU      002H          ; ASCII Start of transmission character
0003   = 34 ETX      EQU      003H          ; ASCII End of transmission character
0004   = 35 EOT      EQU      004H          ; ASCII End of text marker.
0005   = 36 ENQ      EQU      005H          ; ASCII Enquire character
0006   = 37 ACK      EQU      006H          ; ASCII Ack character
0007   = 38 BELL     EQU      007H          ; ASCII Bell character.
0008   = 39 BS       EQU      008H          ; ASCII Back space.
0009   = 40 HT       EQU      009H          ; ASCII Horiz tab character.
0009   = 41 TAB      EQU      009H          ; ASCII Horiz tab character.
000A   = 42 LF       EQU      00AH          ; ASCII Line feed character.
000B   = 43 VT       EQU      00BH          ; ASCII Vertical tab character
000C   = 44 FF       EQU      00CH          ; ASCII Form Feed character.
000D   = 45 CR       EQU      00DH          ; ASCII Carriage return character.
000E   = 46 SO       EQU      00EH          ; ASCII Shift out character
000F   = 47 SI       EQU      00FH          ; ASCII Shift in character
                                48 ;
0010   = 49 DLE      EQU      010H          ; ASCII DLE character.
0011   = 50 DC1      EQU      011H          ; ASCII Device control #1 (Xon)
0012   = 51 DC2      EQU      012H          ; ASCII Device control #2
0013   = 52 DC3      EQU      013H          ; ASCII Device control #3 (Xoff)
0014   = 53 DC4      EQU      014H          ; ASCII Device control #4
0015   = 54 NAK      EQU      015H          ; ASCII Negative ACK character
0016   = 55 SYN      EQU      016H          ; ASCII Sync character
0017   = 56 ETB      EQU      017H          ; ASCII End of transmission buffer.
0018   = 57 CAN      EQU      018H          ; ASCII Cancel.
0019   = 58 EM       EQU      019H          ; ASCII
001A   = 59 ASUB     EQU      01AH          ; ASCII Submit character.
001B   = 60 ESC      EQU      01BH          ; ASCII Escape character.
001C   = 61 FS       EQU      01CH          ; ASCII Field separator.
001D   = 62 GS       EQU      01DH          ; ASCII Group separator
001E   = 63 RS       EQU      01EH          ; ASCII Record separator
001F   = 64 US       EQU      01FH          ; ASCII Unit separator.
                                65 ;
                                66 ;          Control characters:
                                67 ;
0013   = 68 CTRLS    EQU      CTRLMASK AND 'S'      ; Control 'S'
0011   = 69 CTRLQ    EQU      CTRLMASK AND 'Q'      ; Control 'Q'
001A   = 70 CTRLZ    EQU      CTRLMASK AND 'Z'      ; Control 'Z'

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITS diagnostics

14:21:43 20-APR-88

DIAG PAGE 2

LOC	OBJ	LINE	SOURCE STATEMENT
		= 71 ;	
		= 72 ;	Some more ASCII characters:
		= 73 ;	
0020		= 74 SPACE EQU 020H	; ASCII Space.
0027		= 75 QUOTE EQU 027H	; Single quote (')
007F		= 76 DEL EQU 07FH	; ASCII Delete character.
		= 77 ;	
		= 78 SINCLUDE(VT52.EQU)	
		= 79 ;	
		= 80 ;	-----
		= 81 ;	
		= 82 ;	VT52 ESC characters
		= 83 ;	
0048		= 84 HOME EQU 'H'	; Home cursor command
004A		= 85 CLRSC EQU 'J'	; Clear screen command
0059		= 86 GOTOC EQU 'Y'	; Direct cursor position command
004B		= 87 CLRLC EQU 'K'	; Clear line command
0049		= 88 REVLF EQU 'I'	; Reverse line feed
0020		= 89 OFFSET EQU 20H	; Direct cursor position offset
		= 90 ;	
0041		= 91 UPCMD EQU 'A'	; Up cursor command
0042		= 92 DNCMD EQU 'B'	; Down cursor command
0044		= 93 LFCMD EQU 'D'	; Left cursor command
0043		= 94 RTCMD EQU 'C'	; Right cursor command
		= 95 ;	
0000		= 96 MINX EQU 0	; Min x value (Col)
004F		= 97 MAXX EQU 79	; Max x value (Col)
0000		= 98 MINY EQU 0	; Min y value (Row)
0017		= 99 MAXY EQU 23	; Max y value (Row)
		= 100 ;	
		= 101 ;	
		= 102 ;	-----
		= 103 ;	
		= 104 ;	Local equates:
		= 105 ;	
0000		= 106 EOTABLE EQU 0	; End of table marker
		= 107 ;	
		= 108 ;	-----
		= 109 ;	
		= 110 EXTRN OUTMSG, GETUC, CO, CI, CITEST	
		= 111 ;	
		= 112 ;	-----
		= 113 ;	
		= 114 EXTRN ITEST, MTEST, VINIT, VTEST, STEST, RTEST, PTEST	
		= 115 ;	
		= 116 ;	-----
		= 117 ;	
		= 118 PUBLIC INDIRECT, ?ESCSP, FDCMP, PTAB, SEARCH, DELAY, WAIT, CLRS	
		= 119 PUBLIC CRLF, HOME	
		= 120 ;	
		= 121 ;	-----
		= 122 ;	
0064		= 123 STKLN 100	
		= 124 ;	
		= 125 ;	-----
		= 126 ;	
		= 127 CSEG	; Begin (* Start of main program *)
		= 128 ;	
		= 129 DIAG:	
0000 F3		= 130 DI	; Disable_interrupts;
0001 310000 S		= 131 LXI SP,STACK	; SP := Stack;
		= 132 ;	
0004 CD0000 E		= 133 CALL VINIT	; Initialize_video;
		= 134 MLOOP:	; Repeat
0007 CD8100 C		= 135 CALL CLRS	; Clear_screen;
000A 21C700 C		= 136 LXI H,MAINMENU	
000D CD0000 E		= 137 CALL OUTMSG	; Writeln(MainMenu);
		= 138 VALIDC:	; Repeat
0010 CD0000 E		= 139 CALL GETUC	; A := Get_upper_case_character;
0013 21D002 C		= 140 LXI H,MCTAB	

DIAG LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITS diagnostics

14:21:43 20-APR-88

DIAG PAGE 3

LOC	OBJ	LINE	SOURCE STATEMENT
0016	CD3A00	C 141	CALL SEARCH ; Search_Table(A);
0019	C21000	C 142	JNZ VALIDC ; Until A in Table_of_valid_options
		143 ;	
		144 ;	Search returns the address of the service routine in DE
		145 ;	
001C	CD8100	C 146	CALL CLRS ; Clear_screen;
001F	EB	147	XCHG ; Swap(HL,DE);
0020	CD2700	C 148	CALL INDIRECT ; Call_indirect(HL);
0023	C30700	C 149	JMP MLOOP ; Until false;
		150 EXIT:	
		151	HLT ; End.
0026	76	152 ;	
		153 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITs diagnostics

14:21:43 20-APR-88

DIAG PAGE 4

LOC	OBJ	LINE	SOURCE STATEMENT
		154 ;	=====
		155 ;	
		156 ;	
		157 ;	Subroutines.
		158 ;	
		159 ;	
		160 ;	-----
		161 ;	
0027	E9	162	INDIRECT: PCHL ; Indirect call routine
		163 ;	
		164 ;	-----
		165 ;	
		166 ;	Print a TAB character on the display.
		167 ;	
		168	PTAB: ; Procedure PrintTab;
		169	; Begin
0028	3E09	170	MVI A,TAB ;
002A	CD0000	171	CALL CO ; Console_out(TAB)
002D	C9	172	RET ; End;
		173 ;	
		174 ;	-----
		175 ;	
		176 ;	If the input character (In the acc) is a space or an Esc then the
		177 ;	Zero flag will be set before returning.
		178 ;	
		179	?ESCSP: ; Function ?ESCSP(Ch:Char) : Boolean;
		180	; Begin
002E	FE1B	181	CPI ESC ; If Ch = ESC then
0030	C8	182	RZ ; ?ESCSP := True
		183	; else
0031	FE20	184	CPI SPACE ; If Ch = Space then ?ESCSP := True
		185	; Else ?ESCSP := False;
0033	C9	186	RET ; End;
		187 ;	
		188 ;	-----
		189 ;	
		190 ;	Compare the two 16 bit word DE and HL. If they are equal then the
		191 ;	Zero flag will be set before returning.
		192 ;	
		193	FDCMP: ;
0034	7C	194	MOV A,H ; Fetch MSByte of the 1st number
0035	BA	195	CMP D ; Compare with the MSByte of the 1st number
0036	C0	196	RNZ ; Return with zero clear if not equal
		197	;
0037	7D	198	MOV A,L ; Fetch LSByte of the 2nd number
0038	BB	199	CMP E ; Compare with the LSByte of the 2nd number
0039	C9	200	RET ; Return with the zero flag set if equal.
		201 ;	
		202 ;	-----
		203 ;	
		204 ;	Search a table for a given character.
		205 ;	
		206 ;	Inputs: A = Character
		207 ;	HL = Start address of the table
		208 ;	
		209 ;	Outputs: Z = 1, If the character is found in the table.
		210 ;	DE = Address of the service routine
		211 ;	
		212 ;	Z = 0, If the character is not found
		213 ;	
		214	SEARCH: ; Start of the <SEARCH> routine
003A	47	215	MOV B,A ; Save character typed.
		216	SRCH05: ;
003B	7E	217	MOV A,M ; Fetch command character from table
003C	FE00	218	CPI EOTABLE ; End of table ?
003E	CA5400	219	JZ SRCH15 ; Yes - Then error exit
		220	;
0041	B8	221	CMP B ; Is it the same as the character typed ?
0042	CA4B00	222	JZ SRCH10 ; Yes - Then fetch pointer
		223	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITS diagnostics

14:21:43 20-APR-88

DIAG PAGE 5

LOC	OBJ	LINE	SOURCE STATEMENT	
0045	23	224	INX	H ;
0046	23	225	INX	H ;
0047	23	226	INX	H ; Inc pointer to next entry in command table.
0048	C33B00	C 227	JMP	SRCH05 ;
		228		;
		229	SRCH10:	; Come here if match found.
004B	23	230	INX	H ; Inc pointer past character.
004C	5E	231	MOV	E,M ; Fetch LSByte.
004D	23	232	INX	H ;
004E	56	233	MOV	D,M ; Fetch MSByte.
004F	3E00	234	MVI	A,0 ;
0051	C35600	C 235	JMP	SRCH20 ; Exit with zero flag set.
		236		;
		237	SRCH15:	;
0054	3EFF	238	MVI	A,OFFH ;
		239	SRCH20:	;
0056	B7	240	ORA	A ;
0057	C9	241	RET	;
		242		;
		243		;
		244		-----
		245		;
		246		Delay routine
		247		;
		248	DELAY:	; Start of the <DELAY> routine
0058	F5	249	PUSH	PSW ; Save Acc and flags
0059	E5	250	PUSH	H ; Save HL
005A	21E803	251	LXI	H,1000 ; HL (Delay counter) = Delay constant
		252	DEL05:	;
005D	2B	253	DCX	H ; Dec delay counter
005E	7D	254	MOV	A,L ;
005F	B7	255	ORA	A ; Delay counter = zero ?
0060	C25D00	C 256	JNZ	DEL05 ; No - then loop
		257		;
0063	E1	258	POP	H ; Restore HL
0064	F1	259	POP	PSW ; Restore Acc and flags
0065	C9	260	RET	; Return from the <DELAY> routine
		261		;
		262		-----
		263		;
		264		Carriage return/ line feed routine (Starts a new line on the screen)
		265		;
		266	CRLF:	; Start of the <CRLF> routine
0066	3E0D	267	MVI	A,CR ; Move ASCII constant for a CR to the Acc
0068	CD0000	E 268	CALL	CO ; Output via CO routine
006B	3E0A	269	MVI	A,LF ; Move ASCII constant for a LF to the Acc
006D	CD0000	E 270	CALL	CO ; Output via CO routine
0070	C9	271	RET	; Return from the <CRLF> routine
		272		;
		273		-----
		274		;
		275		Wait for a key to be typed in the screen (prompt printed)
		276		;
		277	WAIT:	;
0071	21A900	C 278	LXI	H,WAITMSG ;
0074	CD0000	E 279	CALL	OUTMSG ;
0077	CD0000	E 280	CALL	CITEST ;
007A	C40000	E 281	CNZ	CI ; Clear USART.
		282	WAIT05:	;
007D	CD0000	E 283	CALL	CI ;
0080	C9	284	RET	;
		285		;
		286		-----
		287		;
		288		Clear the screen
		289		;
		290	CLRS:	; Start of the <CLRS> routine
0081	CD9300	C 291	CALL	HOME ; Home the cursor
0084	CD9E00	C 292	CALL	CLRTOS ; Clear to the end of screen
0087	C9	293	RET	; Return from the <CLRS> routine

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITS diagnostics

14:21:43 20-APR-88

DIAG PAGE 6

LOC	OBJ	LINE	SOURCE STATEMENT
		294 ;	
		295 ;-----	
		296 ;	
		297 ;	Clear line
		298 ;	
		299 CLRL:	; Start of the <CLRL> routine
0088	3E1B	300	MVI A,ESC ;
008A	CD0000	E 301	CALL CO ; Print an <ESC> character
008D	3E4B	302	MVI A,CLRLC ;
008F	CD0000	E 303	CALL CO ; Print the VT52 clear line command
0092	C9	304	RET ; Return from the <CLRL> routine
		305 ;	
		306 ;-----	
		307 ;	
		308 ;	Home the cursor
		309 ;	
		310 HOME:	; Start of <HOME> routine
0093	3E1B	311	MVI A,ESC ;
0095	CD0000	E 312	CALL CO ; Print an <ESC> character
0098	3E48	313	MVI A,HOMEC ;
009A	CD0000	E 314	CALL CO ; Print the VT52 home cursor command
009D	C9	315	RET ; Return from the <HOME> routine
		316 ;	
		317 ;-----	
		318 ;	
		319 ;	Clear to end of screen
		320 ;	
		321 CLRTOS:	; Start of <CLRTOS> routine
009E	3E1B	322	MVI A,ESC ;
00A0	CD0000	E 323	CALL CO ; Print an <ESC> character
00A3	3E4A	324	MVI A,CLRSC ;
00A5	CD0000	E 325	CALL CO ; Print the VT52 clear to end of screen command
00A8	C9	326	RET ; Return from the <CLRTOS> routine
		327 ;	
		328 SEJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITs diagnostics

14:21:43 20-APR-88

DIAG PAGE 7

LOC	OBJ	LINE	SOURCE STATEMENT
		329	;=====
		330	;
		331	General message data.
		332	;
		333	WAITMSG:
00A9	1B593720	334	DB ESC,GOTOC,23+OFFSET,OFFSET
00AD	50726573	335	DB 'Press any key to continue'
	7320616E		
	79206B65		
	7920746F		
	20636F6E		
	74696E75		
	65		
00C6	00	336	DB 0
		337	;
		338	MAINMENU:
		339	;
00C7	0D0A	340	DB CR,LF
00C9	0D0A0920	341	DB CR,LF,TAB,' SABus Diagnostics'
	20202020		
	20202020		
	20202020		
	20202020		
	20205341		
	42757320		
	44696167		
	6E6F7374		
	696373		
00F0	0D0A0920	342	DB CR,LF,TAB,' -----'
	20202020		
	20202020		
	20202020		
	20202020		
	20202D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D		
0117	0D0A	343	DB CR,LF
0119	0D0A	344	DB CR,LF
011B	0D0A0920	345	DB CR,LF,TAB,' The following options are available:'
	20202020		
	20202020		
	20205468		
	6520666F		
	6C6C6F77		
	696E6720		
	6F707469		
	6F6E7320		
	61726520		
	61766169		
	6C61626C		
	653A		
014D	0D0A	346	DB CR,LF
014F	0D0A	347	DB CR,LF
0151	0D0A0920	348	DB CR,LF,TAB,' Test the interrupts (I)'
	20202020		
	20202020		
	20202020		
	20202054		
	65737420		
	74686520		
	696E7465		
	72727570		
	74732020		
	20202020		
	284929		
0180	0D0A0920	349	DB CR,LF,TAB,' Test memory (M)'
	20202020		
	20202020		

DIAG LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITs diagnostics

14:21:43 20-APR-88

DIAG

PAGE 8

LOC	OBJ	LINE	SOURCE STATEMENT
	20202020		
	20202054		
	65737420		
	6D656D6F		
	72792020		
	20202020		
	20202020		
	20202020		
	284D29		
01AF	OD0A0920	350	DB CR,LF,TAB,' Test the VDU (V)'
	20202020		
	20202020		
	20202020		
	20202054		
	65737420		
	74686520		
	56445520		
	20202020		
	20202020		
	20202020		
	285629		
01DE	OD0A0920	351	DB CR,LF,TAB,' Test Mass storage card (S)'
	20202020		
	20202020		
	20202020		
	20202054		
	65737420		
	4D617373		
	2073746F		
	72616765		
	20636172		
	64202020		
	285329		
020D	OD0A0920	352	DB CR,LF,TAB,' Test I/O card (RS232) (R)'
	20202020		
	20202020		
	20202020		
	20202054		
	65737420		
	492F4F20		
	63617264		
	20285253		
	32333229		
	20202020		
	285229		
023C	OD0A0920	353	DB CR,LF,TAB,' Test I/O card (Para) (P)'
	20202020		
	20202020		
	20202020		
	20202054		
	65737420		
	492F4F20		
	63617264		
	20285061		
	72612920		
	20202020		
	285029		
026B	OD0A	354	DB CR,LF
026D	OD0A0920	355	DB CR,LF,TAB,' Exit from test mode (E)'
	20202020		
	20202020		
	20202020		
	20202045		
	78697420		
	66726F6D		
	20746573		
	74206D6F		
	64652020		
	20202020		
	284529		

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITS diagnostics

14:21:43 20-APR-88

DIAG PAGE 9

LOC	OBJ	LINE	SOURCE STATEMENT
029C	OD0A	356	DB CR,LF
029E	OD0A0920	357	DB CR,LF,TAB,'
	20202020		Please enter one of the options >',0
	20202020		
	20202020		
	506C6561		
	73652065		
	6E746572		
	206F6E65		
	206F6620		
	74686520		
	6F707469		
	6F6E7320		
	3E00		
		358 ;	
		359 \$EJECT	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITS diagnostics

14:21:43 20-APR-88

DIAG PAGE 10

LOC	OBJ	LINE	SOURCE STATEMENT
		360 ;	-----
		361 ;	
		362 ;	C O M M A N D T A B L E S
		363 ;	-----
		364 ;	
		365 ;	The format of this table is as follows:
		366 ;	
		367 ;	Command_record
		368 ;	Command_record
		369 ;	.
		370 ;	.
		371 ;	Command_record
		372 ;	0 ; Zero marks the end of the table
		373 ;	.
		374 ;	The format of each three byte command record is as follows:
		375 ;	
		376 ;	Byte 0 : ASCII value for the command (e.g. 'I'=Interrupt Test)
		377 ;	
		378 ;	Byte 1 and byte 2 : Address of the subroutine to be called
		379 ;	if the command letter (given in the
		380 ;	previous byte is typed.
		381 ;	
		382 MCTAB:	
02D0	49	383	DB 'I' ; Interrupt test character
02D1	0000	E 384	DW ITEST ; Address of the interrupt test routine
		385	;
02D3	4D	386	DB 'M' ; Memory test character
02D4	0000	E 387	DW MTEST ; Address of the memory test routine
		388	;
02D6	56	389	DB 'V' ; VDU test character
02D7	0000	E 390	DW VTEST ; Address of the VDU test routine
		391	;
02D9	53	392	DB 'S' ; Mass storage card test character
02DA	0000	E 393	DW STEST ; Address of mass storage card test routine
		394	;
02DC	52	395	DB 'R' ; Serial test character
02DD	0000	E 396	DW RTEST ; Address of the serial test routine
		397	;
02DF	50	398	DB 'P' ; Parallel test character
02E0	0000	E 399	DW PTEST ; Address of the parallel port test routine
		400	;
02E2	45	401	DB 'E' ; Exit character
02E3	2600	C 402	DW EXIT ; Exit point of the program
		403	;
02E5	00	404	DB EOTABLE ; End of command table marker.
		405	;
		406	;
0000		407	END DIAG ; End of program.

0 error(s) detected

External symbols

CI	CITEST	CO	GETUC	ITEST	MTEST	OUTMSG	PTEST	RTEST	STEST	VINIT	VTEST

Public symbols

?ESCSP	CLRS	CRLF	DELAY	FDCMP	HONE	INDIRE	PTAB	SEARCH	WAIT

User symbols

?ESCSP	002E C	ACK	0006 A	ASUB	001A A	BELL	0007 A	BS	0008 A	CAN	0018 A	CI	0009 E
CITEST	0008 E	CLRL	0088 C	CLRLC	004B A	CLRS	0081 C	CLRSC	004A A	CLRTOS	009E C	CO	0007 E
CR	000D A	CRLF	0066 C	CTRLMA	001F A	CTRLQ	0011 A	CTRLS	0013 A	CTRLZ	001A A	DC1	0011 A
DC2	0012 A	DC3	0013 A	DC4	0014 A	DEL	007F A	DELO5	005D C	DELAY	0058 C	DIAG	0000 C
DLE	0010 A	DNCHD	0042 A	EN	0019 A	ENQ	0005 A	EOT	0004 A	EOTABL	0000 A	ESC	001B A
ETB	0017 A	ETX	0003 A	EXIT	0026 C	FDCMP	0034 C	FF	000C A	FS	001C A	GETUC	0001 E

DIAG LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0
UCT SABus KITs diagnostics

14:21:43 20-APR-88

DIAG PAGE 11

LOC	OBJ	LINE	SOURCE STATEMENT
GOTOC	0059 A	GS 001D A	HOME 0093 C HOMEC 0048 A HT 0009 A INDIRE 0027 C ITEST 000B E
LF	000A A	LFCMD 0044 A	MAINME 00C7 C MAXX 004F A MAXY 0017 A MCTAB 02D0 C MINX 0000 A
MINY	0000 A	MLOOP 0007 C	MTEST 0005 E NAK 0015 A NUL 0000 A OFFSET 0020 A OUTMSG 0004 E
PTAB	0028 C	PTEST 0002 E	QUOTE 0027 A REVLF 0049 A RS 001E A RTCMD 0043 A RTEST 0000 E
SEARCH	003A C	SI 000F A	SO 000E A SOM 0001 A SPACE 0020 A SRCH05 003B C SRCH10 004B C
SRCH15	0054 C	SRCH20 0056 C	STEST 0003 E STX 0002 A SYN 0016 A TAB 0009 A UPCND 0041 A
US	001F A	VALIDC 0010 C	VINIT 0006 E VT 000B A VTEST 000A E WAIT 0071 C WAIT05 007D C
WAITMS	00A9 C		

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$TITLE('Interrupt test routines for the UCT SABus diagnostics')
		2	\$PAGELENGTH(75)
		3	;
		4	;=====
		5	;
		6	A: Interrupt test. (File = DIAGINTER.A85)
		7	;
		8	;=====
		9	;
		10	NAME DINTER
		11	;
		12	Include files:
		13	;
		14	\$INCLUDE(COMEQU.EQU)
		= 15	;
		= 16	;-----
		= 17	;
		= 18	Common equates:
		= 19	;
0000		= 20	FALSE EQU 0 ; Logical FALSE.
00FF		= 21	TRUE EQU (NOT FALSE) AND OFFH ; Logical TRUE.
		= 22	;
		= 23	Storage size Equates (In terms of bytes).
		= 24	;
0002		= 25	WORD EQU 2 ; 1 Word = 2 Bytes
0001		= 26	BYTE EQU 1 ; 1 Byte = 1 Byte.
0400		= 27	K EQU 1024 ; 1kbytes = 1024 bytes
		= 28	;
		29	\$INCLUDE(ASCII.EQU)
		= 30	;
		= 31	;-----
		= 32	;
001F		= 33	CTRLMASK EQU 00011111B ; Control mask.
		= 34	;
		= 35	ASCII characters:
		= 36	;
0000		= 37	NUL EQU 000H ; ASCII null character
0001		= 38	SOH EQU 001H ; ASCII Start of header character
0002		= 39	STX EQU 002H ; ASCII Start of transmission character
0003		= 40	ETX EQU 003H ; ASCII End of transmission character
0004		= 41	EOT EQU 004H ; ASCII End of text marker.
0005		= 42	ENQ EQU 005H ; ASCII Enquire character
0006		= 43	ACK EQU 006H ; ASCII Ack character
0007		= 44	BELL EQU 007H ; ASCII Bell character.
0008		= 45	BS EQU 008H ; ASCII Back space.
0009		= 46	HT EQU 009H ; ASCII Horiz tab character.
0009		= 47	TAB EQU 009H ; ASCII Horiz tab character.
000A		= 48	LF EQU 00AH ; ASCII Line feed character.
000B		= 49	VT EQU 00BH ; ASCII Vertical tab character
000C		= 50	FF EQU 00CH ; ASCII Form Feed character.
000D		= 51	CR EQU 00DH ; ASCII Carriage return character.
000E		= 52	SO EQU 00EH ; ASCII Shift out character
000F		= 53	SI EQU 00FH ; ASCII Shift in character
		= 54	;
0010		= 55	DLE EQU 010H ; ASCII DLE character.
0011		= 56	DC1 EQU 011H ; ASCII Device control #1 (Xon)
0012		= 57	DC2 EQU 012H ; ASCII Device control #2
0013		= 58	DC3 EQU 013H ; ASCII Device control #3 (Xoff)
0014		= 59	DC4 EQU 014H ; ASCII Device control #4
0015		= 60	NAK EQU 015H ; ASCII Negative ACK character
0016		= 61	SYN EQU 016H ; ASCII Sync character
0017		= 62	ETB EQU 017H ; ASCII End of transmission buffer.
0018		= 63	CAN EQU 018H ; ASCII Cancel.
0019		= 64	EM EQU 019H ; ASCII
001A		= 65	ASUB EQU 01AH ; ASCII Submit character.
001B		= 66	ESC EQU 01BH ; ASCII Escape character.
001C		= 67	FS EQU 01CH ; ASCII Field separator.
001D		= 68	GS EQU 01DH ; ASCII Group separator
001E		= 69	RS EQU 01EH ; ASCII Record separator
001F		= 70	US EQU 01FH ; ASCII Unit separator.

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:22:11 20-APR-88

DINTER PAGE 2

Interrupt test routines for the UCT SABUS diagnostics

```

LOC   OBJ       LINE      SOURCE STATEMENT
      = 71 ;
      = 72 ;           Control characters:
      = 73 ;
0013   = 74 CTRLS EQU      CTRLMASK AND 'S'           ; Contorl 'S'
0011   = 75 CTRLQ EQU      CTRLMASK AND 'Q'           ; Control 'Q'
001A   = 76 CTRLZ EQU      CTRLMASK AND 'Z'           ; Control 'Z'
      = 77 ;
      = 78 ;           Some more ASCII characters:
      = 79 ;
0020   = 80 SPACE EQU      020H                       ; ASCII Space.
0027   = 81 QUOTE EQU      027H                       ; Single quote (')
007F   = 82 DEL EQU       07FH                       ; ASCII Delete character.
      = 83 ;
      84 $INCLUDE(VT52.EQU)
      = 85 ;
      = 86 ;-----
      = 87 ;
      = 88 ;           VT52 ESC characters
      = 89 ;
0048   = 90 HOMEC EQU      'H'           ; Home cursor command
004A   = 91 CLRSC EQU      'J'           ; Clear screen command
0059   = 92 GOTOC EQU      'Y'           ; Direct cursor position command
004B   = 93 CLRLC EQU      'K'           ; Clear line command
0049   = 94 REVLF EQU      'I'           ; Reverse line feed
0020   = 95 OFFSET EQU     20H           ; Direct cursor position offset
      = 96 ;
0041   = 97 UPCMD EQU      'A'           ; Up cursor command
0042   = 98 DNCMD EQU      'B'           ; Down cursor command
0044   = 99 LFCMD EQU      'D'           ; Left cursor command
0043   = 100 RTCMD EQU     'C'           ; Right cursor command
      = 101 ;
0000   = 102 MINX EQU      0             ; Min x value (Col)
004F   = 103 MAXX EQU      79           ; Max x value (Col)
0000   = 104 MINY EQU      0             ; Min y value (Row)
0017   = 105 MAXY EQU      23           ; Max y value (Row)
      = 106 ;
      107 ;
      108 ;-----
      109 ;
      110 ;           Publics:
      111 ;
      112 PUBLIC ITEST, GOTO
      113 ;
      114 ;-----
      115 ;
      116 ;           Externals:
      117 ;
      118 EXTRN DELAY, ?ESCSP           ; FROM DIAG.A85
      119 EXTRN OUTMSG, CITEST, CI, CO ; FROM SABMON.A85
      120 ;
      121 ;-----
      122 ;
      123 CSEG                        ; Start of code segment
      124 ;
      125 ;-----
      126 ;
      127 ;           Main interrupt test routine:
      128 ;
      129 ITEST:                      ; Procedure TestInterrupts;
0000 21C300 C 130 LXI H,IRMSG           ;
0003 CD0000 E 131 CALL OUTMSG           ; OutMessage(InterruptTestMessage);
0006 3E10 132 MVI A,00010000B           ;
0008 30 133 SIM                         ; Reset_75_flip_flop;
0009 3E00 134 MVI A,FALSE               ;
000B 320000 D 135 STA IFLAGS+0           ; InterruptFlags[0] := False {RST 5.5}
000E 320100 D 136 STA IFLAGS+1           ; InterruptFlags[1] := False {RST 6.5}
0011 320200 D 137 STA IFLAGS+2           ; InterruptFlags[2] := False {RST 7.5}
0014 3E00 138 MVI A,0                   ;
0016 320300 D 139 STA DEL75             ; Delay75 := 0;
      140 IT05:                      ;

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:22:11 20-APR-88

DINTER PAGE 3

Interrupt test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		141	
0019	21DF01	C 142	LXI H,IGOTOM ; Repeat
001C	CD0000	E 143	CALL OUTMSG ; OutMessage(InterruptGotoMessage);
001F	210000	D 144	LXI H,IFLAGS ;
0022	11E401	C 145	LXI D,IPOSDAT ;
0025	20	146	RIM ; PendingInterrupts := RIM;
0026	47	147	MOV B,A ;
0027	E610	148	ANI 00010000B ;
0029	CD7700	C 149	CALL UPDATE ; Update(PendingInterrupts AND 00010000B,
		150	InterruptPositionData+0,
		151	InterruptFlags+0);
002C	13	152	INX D ;
002D	13	153	INX D ;
002E	23	154	INX H ;
002F	78	155	MOV A,B ;
0030	E620	156	ANI 00100000B ;
0032	CD7700	C 157	CALL UPDATE ; Update(PendingInterrupts AND 00100000B,
		158	InterruptPositionData+2,
		159	InterruptFlags+1);
0035	13	160	INX D ;
0036	13	161	INX D ;
0037	23	162	INX H ;
0038	78	163	MOV A,B ;
0039	CD4F00	C 164	CALL INT75 ; Int75(PendingInterrupts AND 01100000B,
		165	InterruptPositionData+4,
		166	InterruptFlags+2);
		167	;
003C	CD0000	E 168	CALL DELAY ; Delay;
		169	;
003F	CD0000	E 170	CALL CITEST ;
0042	CA1900	C 171	JZ IT05 ; Until ConsoleInputTest = TRUE and
		172	;
0045	CD0000	E 173	CALL CI ;
0048	CD0000	E 174	CALL ?ESCSP ; ConsoleInput in [ESC, Space];
004B	C21900	C 175	JNZ IT05 ;
		176	;
004E	C9	177	RET ; EndProc;
		178	;
		179	*****
		180	;
		181	INT75: ; Procedure Interrupt75(InterruptPositionData;
		182	InterruptFlags:Word);
004F	3A0300	D 183	LDA DEL75 ;
0052	FE00	184	CPI 0 ;
0054	C26800	C 185	JNZ INT75A ; If Delay75 = 0 then
		186	Begin
		187	;
0057	78	187	MOV A,B ;
0058	E640	188	ANI 01000000B ;
005A	CA7300	C 189	JZ NO75 ;
		190	;
005D	CD7700	C 191	CALL UPDATE ; Update(PendingInterrupts AND 10000B,
		192	InterruptPositionData,
		193	InterruptFlags);
0060	3EFF	194	MVI A,OFFH ;
0062	320300	D 195	STA DEL75 ; Delay75 := 255;
		196	;
0065	C37600	C 197	JMP I75RET ; End
		198	INT75A: ; Else
		199	Begin
0068	3D	200	DCR A ;
0069	320300	D 201	STA DEL75 ; Delay75 := Delay75 - 1;
006C	C27600	C 202	JNZ I75RET ; If Delay75 = 0 then
		203	;
006F	AF	204	XRA A ;
0070	CD7700	C 205	CALL UPDATE ; Update(0, InterruptPositionData,
		206	InterruptFlags);
		207	NO75: ;
0073	3E10	208	MVI A,00010000B ;
0075	30	209	SIM ; Reset_75_flip_flop;
		210	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:22:11 20-APR-88

DINTER PAGE 4

Interrupt test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		211	I75RET: ; End;
0076	C9	212	RET ; EndProc;
		213	;
		214	;-----
		215	;
		216	; If Z=1 on entry then write interrupt not active
		217	; If Z=0 on entry then write Interrupt active
		218	;
		219	UPDATE: ; Procedure Update(InterruptPending,
		220	; InterruptPositionData,
		221	; InterruptFlags);
		222	;
0077	CA9000	C 223	JZ NOPEND ; If NOT InterruptPending then
		224	Begin
007A	7E	225	MOV A,M ;
007B	FEFF	226	CPI TRUE ;
007D	CA8D00	C 227	JZ UPDA05 ; If InterruptFlags[0] <> TRUE then
		228	Begin
		229	;
0080	36FF	230	MVI M,TRUE ; InterruptFlags[0] := TRUE;
0082	E5	231	PUSH H ;
0083	CDA400	C 232	CALL IGOTO ; IndirectGoto(
		233	InterruptPositionData);
0086	21EA01	C 234	LXI H,ACTM ;
0089	CD0000	E 235	CALL OUTMSG ; OutMessage(ActiveMessage)
008C	E1	236	POP H ;
		237	UPDA05: ; End;
008D	C3A300	C 238	JMP UPRET ; End
		239	NOPEND: ; Else
		240	Begin
0090	7E	241	MOV A,M ;
0091	FE00	242	CPI FALSE ;
0093	CAA300	C 243	JZ UPRET ; If InterruptFlags[0] = TRUE
		244	Begin
		245	;
0096	3600	246	MVI M,FALSE ; InterruptFlags[0] := FALSE;
0098	E5	247	PUSH H ;
0099	CDA400	C 248	CALL IGOTO ; IndirectGoto(
		249	InterruptPositionData);
009C	210202	C 250	LXI H,NOACTM ;
009F	CD0000	E 251	CALL OUTMSG ; OutMessage(NotActiveMessage)
00A2	E1	252	POP H ;
		253	;
		254	UPRET: ; End;
00A3	C9	255	RET ; EndProc;
		256	;
		257	;-----
		258	;
		259	; PosPtr = DE : Pointer to the ROW and COLUMN values to goto.
		260	;
		261	IGOTO: ; Procedure IndirectGoto(PosPtr:Word)
00A4	C5	262	PUSH B ;
00A5	1A	263	LDAX D ;
00A6	47	264	MOV B,A ; Col := Memory(Ptr);
00A7	13	265	INX D ;
00A8	1A	266	LDAX D ; Row := Memory(Ptr+1);
00A9	1B	267	DCX D ;
00AA	CDAF00	C 268	CALL GOTO ; Goto(Row, Col);
00AD	C1	269	POP B ;
00AE	C9	270	RET ; EndProc;
		271	;
		272	;-----
		273	;
		274	; A = Row
		275	; B = Col
		276	;
		277	GOTO: ; Procedure Goto(Row, Col:Byte);
00AF	F5	278	PUSH PSW ;
00B0	3E1B	279	MVI A,ESC ; ConsoleOut(ESC);
00B2	CD0000	E 280	CALL CO ;

Interrupt test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
00B5	3E59	281	MVI A,GOTOC ; ConsoleOut(GotoCommand);
00B7	CD0000	282	CALL CO ;
00BA	F1	283	POP PSW ; ConsoleOut(Row);
00BB	CD0000	284	CALL CO ;
00BE	78	285	MOV A,B ; ConsoleOut(Col);
00BF	CD0000	286	CALL CO ;
00C2	C9	287	RET ; EndProc;
		288 ;	
		289 ;	
		290 ;	
		291 ;	Interrupts test message data.
		292 ;	
00C3	0D0A	293	IRMSG: DB CR, LF
00C5	0D0A	294	DB CR, LF
00C7	0909	295	DB TAB, TAB
00C9	52535420	296	DB 'RST 5.5, 6.5 and 7.5 interrupt test routines'
	352E352C		
	20362E35		
	20616E64		
	20372E35		
	20696E74		
	65727275		
	70742074		
	65737420		
	726F7574		
	696E6573		
00F5	0D0A	297	DB CR, LF
00F7	0909	298	DB TAB,TAB
00F9	2D2D2D2D	299	DB '-----'
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
0125	0D0A	300	DB CR, LF
0127	0D0A	301	DB CR, LF
0129	0D0A	302	DB CR, LF
012B	0909	303	DB TAB, TAB
012D	20202020	304	DB ' RST 5.5 < > <Not active>'
	20525354		
	20352E35		
	20202020		
	3C202020		
	2020203E		
	2020203C		
	4E6F7420		
	61637469		
	76653E		
0154	0D0A	305	DB CR, LF
0156	0D0A	306	DB CR, LF
0158	0909	307	DB TAB, TAB
015A	20202020	308	DB ' RST 6.5 < > <Not active>'
	20525354		
	20362E35		
	20202020		
	3C202020		
	2020203E		
	2020203C		
	4E6F7420		
	61637469		
	76653E		
0181	0D0A	309	DB CR, LF
0183	0D0A	310	DB CR, LF
0185	0909	311	DB TAB, TAB
0187	20202020	312	DB ' RST 7.5 < > <Not active>'

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:22:11 20-APR-88
Interrupt test routines for the UCT SABus diagnostics

DINTER PAGE 6

LOC	OBJ	LINE	SOURCE STATEMENT
			20525354
			20372E35
			20202020
			3C202020
			2020203E
			2020203C
			4E6F7420
			61637469
			76653E
01AE	OD0A	313	DB CR, LF
01B0	OD0A	314	DB CR, LF
		315	ICOML:
01B2	OD0A	316	DB CR, LF
01B4	0909	317	DB TAB, TAB
01B6	2020203C	318	DB ' <ESC> or <space> returns to main menu',0
			4553433E
			206F7220
			3C737061
			63653E20
			72657475
			726E7320
			746F206D
			61696E20
			6D656E75
			00
		319 ;	
		320 ;	The following formula may seem complicated. It calculates the column
		321 ;	position for the cursor while the interrupts are being tested.
		322 ;	
		323 ;	\$-ICOML (The number of bytes generated by the last line of assembler)
		324 ;	-5 (Subtract the 5 control bytes in the line 2*TAB, CR, LF and 0)
		325 ;	+2*8 (Add 2 TAB characters)
		326 ;	+1 (Move to one character past the "u")
		327 ;	
0039		328 ICOLP EQU	\$-ICOML-5+2*8+1
000D		329 IROWP EQU	13
		330 ;	
01DF	1B592D59	331 IGOTOM: DB	ESC,GOTOC,IROWP+OFFSET,ICOLP+OFFSET,0
	00		
		332 ;	
01E4	4026	333 IPOS DAT: DB	32+OFFSET,6+OFFSET
01E6	4028	334 DB	32+OFFSET,8+OFFSET
01E8	402A	335 DB	32+OFFSET,10+OFFSET
		336	
01EA	3C416374	337 ACTM: DB	'<Active> < >',0
	6976653E		
	2020203C		
	20202020		
	20202020		
	20203E00		
0202	3C202020	338 NOACTM: DB	'< > <Not active>',0
	2020203E		
	2020203C		
	4E6F7420		
	61637469		
	76653E00		
		339 ;	
		340 ;	Interrupt test data area.
		341 ;	
		342	DSEG
		343 ;	
0000		344 IFLAGS: DS	3*BYTE ;
0003		345 DEL75: DS	BYTE ;
		346 ;	
		347 ;	End of <DIAGINTER.A85> module
		348 ;	
		349	END

0 error(s) detected

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:22:11 20-APR-88
Interrupt test routines for the UCT SABus diagnostics

DINTER PAGE 7

LOC OBJ LINE SOURCE STATEMENT

External symbols

?ESCSF CI CITEST CO DELAY OUTMSG

Public symbols

GOTO ITEST

User symbols

?ESCSF 0005 E	ACK 0006 A	ACTM 01EA C	ASUB 001A A	BELL 0007 A	BS 0008 A	BYTE 0001 A
CAN 0018 A	CI 0003 E	CITEST 0002 E	CLRLC 004B A	CLRSC 004A A	CO 0004 E	CR 000D A
CTRLMA 001F A	CTRLQ 0011 A	CTRLS 0013 A	CTRLZ 001A A	DC1 0011 A	DC2 0012 A	DC3 0013 A
DC4 0014 A	DEL 007F A	DEL75 0003 D	DELAY 0001 E	DLE 0010 A	DNCMD 0042 A	EM 0019 A
ENQ 0005 A	EOT 0004 A	ESC 001B A	ETB 0017 A	ETX 0003 A	FALSE 0000 A	FF 000C A
FS 001C A	GOTO 00AF C	GOTOC 0059 A	GS 001D A	HOMEC 0048 A	HT 0009 A	I75RET 0076 C
ICOLP 0039 A	ICOML 01B2 C	IPLAGS 0000 D	IGOTO 00A4 C	IGOTOM 01DF C	INT75 004F C	INT75A 0068 C
IPOSDA 01B4 C	IRMSG 00C3 C	IROWP 000D A	IT05 0019 C	ITEST 0000 C	K 0400 A	LF 000A A
LPCMD 0044 A	MAXX 004F A	MAXY 0017 A	MINX 0000 A	MINY 0000 A	NAK 0015 A	NO75 0073 C
NOACTM 0202 C	NOPEND 0090 C	NUL 0000 A	OFFSET 0020 A	OUTMSG 0000 E	QUOTE 0027 A	REVLf 0049 A
RS 001E A	RTCMD 0043 A	SI 000F A	SO 000E A	SON 0001 A	SPACE 0020 A	STX 0002 A
SYN 0016 A	TAB 0009 A	TRUE 00FF A	UPCMD 0041 A	UPDA05 008D C	UPDATE 0077 C	UPRET 00A3 C
US 001F A	VT 000B A	WORD 0002 A				


```

LOC   OBJ       LINE      SOURCE STATEMENT
                                1 $TITLE('Memory test routines for the UCT SABus KIT diagnostics')
                                2 $PAGELENGTH(75)
                                3 ;
                                4 ;*****
                                5 ;
                                6 ;           B:      Memory test.      (file=DIAGMEM.A85)
                                7 ;
                                8 ;*****
                                9 ;
                                10          NAME      MEM
                                11 ;
                                12 $INCLUDE(ASCII.EQU)
                                13 ;
                                14 ;-----
                                15 ;
001F  = 16 CTRLMASK EQU      00011111B      ; Control mask.
                                17 ;
                                18 ;           ASCII characters:
                                19 ;
0000  = 20 NUL      EQU      000H           ; ASCII null character
0001  = 21 SOH      EQU      001H           ; ASCII Start of header character
0002  = 22 STX      EQU      002H           ; ASCII Start of transmission character
0003  = 23 ETX      EQU      003H           ; ASCII End of transmission character
0004  = 24 EOT      EQU      004H           ; ASCII End of text marker.
0005  = 25 ENQ      EQU      005H           ; ASCII Enquire character
0006  = 26 ACK      EQU      006H           ; ASCII Ack character
0007  = 27 BELL     EQU      007H           ; ASCII Bell character.
0008  = 28 BS       EQU      008H           ; ASCII Back space.
0009  = 29 HT       EQU      009H           ; ASCII Horiz tab character.
0009  = 30 TAB      EQU      009H           ; ASCII Horiz tab character.
000A  = 31 LF       EQU      00AH           ; ASCII Line feed character.
000B  = 32 VT       EQU      00BH           ; ASCII Vertical tab character
000C  = 33 FF       EQU      00CH           ; ASCII Form Feed character.
000D  = 34 CR       EQU      00DH           ; ASCII Carriage return character.
000E  = 35 SO       EQU      00EH           ; ASCII Shift out character
000F  = 36 SI       EQU      00FH           ; ASCII Shift in character
                                37 ;
0010  = 38 DLE      EQU      010H           ; ASCII DLE character.
0011  = 39 DC1      EQU      011H           ; ASCII Device control #1 (Xon)
0012  = 40 DC2      EQU      012H           ; ASCII Device control #2
0013  = 41 DC3      EQU      013H           ; ASCII Device control #3 (Xoff)
0014  = 42 DC4      EQU      014H           ; ASCII Device control #4
0015  = 43 NAK      EQU      015H           ; ASCII Negative ACK character
0016  = 44 SYN      EQU      016H           ; ASCII Sync character
0017  = 45 ETB      EQU      017H           ; ASCII End of transmission buffer.
0018  = 46 CAN      EQU      018H           ; ASCII Cancel.
0019  = 47 EM       EQU      019H           ; ASCII
001A  = 48 ASUB     EQU      01AH           ; ASCII Submit character.
001B  = 49 ESC      EQU      01BH           ; ASCII Escape character.
001C  = 50 FS       EQU      01CH           ; ASCII Field separator.
001D  = 51 GS       EQU      01DH           ; ASCII Group separator
001E  = 52 RS       EQU      01EH           ; ASCII Record separator
001F  = 53 US       EQU      01FH           ; ASCII Unit separator.
                                54 ;
                                55 ;           Control characters:
                                56 ;
0013  = 57 CTRLS    EQU      CTRLMASK AND 'S'      ; Control 'S'
0011  = 58 CTRLQ    EQU      CTRLMASK AND 'Q'      ; Control 'Q'
001A  = 59 CTRLZ    EQU      CTRLMASK AND 'Z'      ; Control 'Z'
                                60 ;
                                61 ;           Some more ASCII characters:
                                62 ;
0020  = 63 SPACE    EQU      020H           ; ASCII Space.
0027  = 64 QUOTE     EQU      027H           ; Single quote (')
007F  = 65 DEL      EQU      07FH           ; ASCII Delete character.
                                66 ;
                                67 $INCLUDE(VT52.EQU)
                                68 ;
                                69 ;-----
                                70 ;

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:22:31 20-APR-88

MEM

PAGE 2

Memory test routines for the UCT SABus KIT diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		= 71 ;	VT52 ESC characters
		= 72 ;	
0048		= 73 HOMECL EQU 'H'	; Home cursor command
004A		= 74 CLRSC EQU 'J'	; Clear screen command
0059		= 75 GOTOC EQU 'Y'	; Direct cursor position command
004B		= 76 CLRCL EQU 'K'	; Clear line command
0049		= 77 REVLFF EQU 'I'	; Reverse line feed
0020		= 78 OFFSET EQU 20H	; Direct cursor position offset
		= 79 ;	
0041		= 80 UPCMD EQU 'A'	; Up cursor command
0042		= 81 DNCMD EQU 'B'	; Down cursor command
0044		= 82 LFCMD EQU 'D'	; Left cursor command
0043		= 83 RTCMD EQU 'C'	; Right cursor command
		= 84 ;	
0000		= 85 MINX EQU 0	; Min x value (Col)
004F		= 86 MAXX EQU 79	; Max x value (Col)
0000		= 87 MINY EQU 0	; Min y value (Row)
0017		= 88 MAXY EQU 23	; Max y value (Row)
		= 89 ;	
		= 90 ;	
		= 91 ;	
		= 92 ;	
		93 EXTRN OUTMSG,CI,CITEST,NMOUT	
		94 EXTRN FDCMP,WAIT,?ESCSP	
		= 95 ;	
		= 96 ;	
		= 97 ;	
		98 PUBLIC MTEST	
		= 99 ;	
		= 100 ;	
		= 101 ;	
		102 ; NOTE: The special symbol MEMORY is the address of the start of the	
		103 ; rest of memory.	
		104 ;	
		105 ;	
		= 106 ;	
		107 ;	
		108 CSEG	
		109 ;	
		110 ; MAIN memory test routine:	
		111 ;	
		112 MTEST:	; Procedure MemoryTest;
0000	215800	C 113 LXI H,MTMSG	;
0003	CD0000	E 114 CALL OUTMSG	; OutMsg(MemoryTestMessage);
		115 MT05:	;
0006	210000	M 116 LXI H,MEMORY	; Ptr := MEMORY;
0009	1100E0	117 LXI D,0E000H	; EndPtr := 0E000H;
		118 ;	
		119 MT10:	; Loop While
000C	CD0000	E 120 CALL CITEST	;
000F	CA1B00	C 121 JZ MT15	; ConsoleInputTest = False AND
		122 ;	
0012	CD0000	E 123 CALL CI	;
0015	CD0000	E 124 CALL ?ESCSP	;
0018	CA3700	C 125 JZ MT25	; Ch not in [ESC, Space] do
		126 ;	
		127 MT15:	; Begin
		128 ;	
001B	CD4700	C 129 CALL MTDISP	; DisplayAddress(Ptr);
001E	CD3800	C 130 CALL TESTB	; Status := TestByte(Ptr);
0021	CA3000	C 131 JZ MT20	;
		132 ;	
		133 ;	If Status = Error then
		134 ;	Begin
0024	21D700	C 134 LXI H,MTERRM	;
0027	CD0000	E 135 CALL OUTMSG	; OutMessage(MemoryError);
002A	CD0000	E 136 CALL WAIT	; Wait;
002D	C33700	C 137 JMP MT25	; End;
		138 MT20:	;
0030	23	139 INX H	; Ptr := Ptr + 1;
0031	CD0000	E 140 CALL FDCMP	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:22:31 20-APR-88
Memory test routines for the UCT SABus KIT diagnostics

MEM PAGE 3

PAGE 3

LOC	OBJ	LINE	SOURCE STATEMENT
0034	C20C00	141	JNZ MT10 ; Until (Ptr = EndPtr) or (Status=Error)
		142	
		143	MT25:
0037	C9	144	RET ; EndProc;
		145	
		146	
		147	
		148	TESTB: ; Procedure TestByte(Ptr:Word)
0038	0608	149	MVI B,8
003A	3E80	150	MVI A,10000000B ; Mask = 10000000B
		151	
		152	TB05: ; For I := 1 to 8 do
		153	Begin
003C	77	154	MOV M,A ; Memory(Ptr) := Mask
003D	BE	155	CMP M ; If Memory(Ptr) <> Mask then EXIT;
003E	C24600	156	JNZ TB10
		157	
0041	0F	158	RRC ; Mask := Mask SHR 1;
0042	05	159	DCR B
0043	C23C00	160	JNZ TB05 ; End;
		161	TB10:
0046	C9	162	RET ; EndProc;
		163	
		164	
		165	
		166	MTDISP: ; Procedure MemoryTestDisplay(Ptr:Word);
0047	E5	167	PUSH H
0048	21D200	168	LXI H,MTGOTO
004B	CD0000	169	CALL OUTMSG ; OutMessage(MemoryTestGotoMessage);
004E	E1	170	POP H
004F	7C	171	MOV A,H
0050	CD0000	172	CALL NMOUT ; NumberOut(HIGH Ptr);
0053	7D	173	MOV A,L
0054	CD0000	174	CALL NMOUT ; NumberOut(LOW Ptr);
		175	
0057	C9	176	RET ; EndProc;
		177	
		178	
		179	
		180	Memory test messages.
		181	
0058	0D0A	182	MTMSG: DB CR,LF
005A	0D0A0909	183	DB CR,LF,TAB,TAB,TAB,TAB,' Memory test'
	0909204D		
	656D6F72		
	79207465		
	7374		
006C	0D0A0909	184	DB CR,LF,TAB,TAB,TAB,TAB,' Memory test'
	0909202D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D		
007E	0D0A	185	DB CR,LF
0080	0D0A	186	DB CR,LF
0082	0D0A0909	187	DB CR,LF,TAB,TAB,TAB,' Testing byte at location :
	09546573		
	74696E67		
	20627974		
	65206174		
	206C6F63		
	6174696F		
	6E203A20		
00A2	0D0A	188	DB CR,LF
00A4	0D0A	189	DB CR,LF
00A6	0D0A0909	190	DB CR,LF,TAB,TAB,' <ESC> or <space> to return to main menu'
	3C455343		
	3E206F72		
	203C7370		
	6163653E		
	20746F20		

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:22:43 20-APR-88

PAR PAGE 3

Parallel port test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		= 141 ;	
0000		= 142 DATA5 EQU 00B SHL 2	; 5 data bits
0004		= 143 DATA6 EQU 01B SHL 2	; 6 data bits
0008		= 144 DATA7 EQU 10B SHL 2	; 7 data bits
000C		= 145 DATA8 EQU 11B SHL 2	; 8 data bits
		= 146 ;	
0000		= 147 SYNCMODE EQU 00B SHL 0	; Synchronous mode
0001		= 148 DIV1 EQU 01B SHL 0	; Clock x1
0002		= 149 DIV16 EQU 10B SHL 0	; Clock x16
0003		= 150 DIV64 EQU 11B SHL 0	; Clock x64
		= 151 ;	
		= 152 ;	USART mode word.
		= 153 ;	
007E		= 154 UMODE EQU ONESTOPBIT + EVENPARITY + DATA8 + DIV16	
004E		= 155 OUMODE EQU ONESTOPBIT + NOPARITY + DATA8 + DIV16	
		= 156 ;	
		= 157 ;	USART command register format
		= 158 ;	
0001		= 159 TXENAB EQU 00000001B	; Transmitter enable bit (Command)
0002		= 160 DTR EQU 00000010B	; Position of the DTR bit (Command)
0004		= 161 RXENAB EQU 00000100B	; Receiver enable bit (Command)
0008		= 162 SBREAK EQU 00001000B	; Send break (Command)
0010		= 163 ERESET EQU 00010000B	; Error reset bit (Command)
0020		= 164 RTS EQU 00100000B	; Request to send bit (Command)
0040		= 165 IRESET EQU 01000000B	; Internal reset bit (Command)
0080		= 166 HUNTM EQU 10000000B	; Enter hunt mode (Command)
		= 167 ;	
		= 168 ;	USART command word:
		= 169 ;	
0015		= 170 UCMD EQU TXENAB + RXENAB + ERESET	
		= 171 ;	
		= 172 ;	USART Status register format:
		= 173 ;	
0001		= 174 TXRDY EQU 00000001B	; Transmitter ready (Status)
0002		= 175 RXRDY EQU 00000010B	; Receiver ready (Status)
0004		= 176 TXEMP EQU 00000100B	; Transmitter empty (Status)
0008		= 177 PARERR EQU 00001000B	; Parity error flag (Status)
0010		= 178 OVRERR EQU 00010000B	; Overrun error flag (Status)
0020		= 179 FRMERR EQU 00100000B	; Framing error flag (Status)
0040		= 180 SYNDET EQU 01000000B	; Sync detect (Status)
0080		= 181 DSR EQU 10000000B	; DSR bit (Status)
		= 182 ;	
0038		= 183 ERMASK EQU 00111000B	; Combination error mask
		= 184 ;	
		= 185 ;	Timer commands and division factor.
		= 186 ;	
0000		= 187 TIMER0 EQU 0 SHL 6	; Address timer 0
0040		= 188 TIMER1 EQU 1 SHL 6	; Address timer 1
0080		= 189 TIMER2 EQU 2 SHL 6	; Address timer 2
		= 190 ;	
0036		= 191 TMODE EQU 00110110B	; Basic mode word =
		= 192	; (Read and write LSB then MSB,
		= 193	; Mode 3,
		= 194	; Count in binary).
		= 195	
0036		= 196 TMODE0 EQU TMODE+TIMER0	; Mode word for timer 0
0076		= 197 TMODE1 EQU TMODE+TIMER1	; Mode word for timer 1
00B6		= 198 TMODE2 EQU TMODE+TIMER2	; Mode word for timer 2
		= 199 ;	
		= 200 ;	Baud rate division factors for the USART, XTAL = 6.144 MHz
		= 201 ;	
0369		= 202 DB110 EQU 873	; Division ratio for 110 baud
0140		= 203 DB300 EQU 320	; Division ratio for 300 baud
00A0		= 204 DB600 EQU 160	; Division ratio for 600 baud
0050		= 205 DB1200 EQU 80	; Division ratio for 1200 baud
0035		= 206 DB1800 EQU 53	; Division ratio for 1800 baud
0028		= 207 DB2400 EQU 40	; Division ratio for 2400 aud
001B		= 208 DB3600 EQU 27	; Division ratio for 3600 baud
0014		= 209 DB4800 EQU 20	; Division ratio for 4800 baud
000D		= 210 DB7200 EQU 13	; Division ratio for 7200 baud

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:22:43 20-APR-88

PAR

PAGE 4

Parallel port test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
000A		= 211 DB9600 EQU 10	; Division ratio for 9600 baud
		= 212 ;	
		= 213 ;	PPI commands:
		= 214 ;	
008B		= 215 AOCMD EQU 10001011B	; A = Mode 0 output
		= 216	; B = Mode 0 input
		= 217	; C (upper/lower) = Mode 0 input
		= 218 ;	
0099		= 219 BOCMD EQU 10011001B	; A = Mode 0 input
		= 220	; B = Mode 0 output
		= 221	; C (upper/lower) = Mode 0 input
		= 222 ;	
0092		= 223 COCMD EQU 10010010B	; A = Mode 0 input
		= 224	; B = Mode 0 input
		= 225	; C (upper/lower) = Mode 0 output
		= 226 ;	
		= 227 ;	END HARD
		228 \$INCLUDE(COMEQU.EQU)	
		= 229 ;	
		= 230 ;	-----
		= 231 ;	
		= 232 ;	Common equates:
		= 233 ;	
0000		= 234 FALSE EQU 0	; Logical FALSE.
00FF		= 235 TRUE EQU (NOT FALSE) AND OFFH	; Logical TRUE.
		= 236 ;	
		= 237 ;	Storage size Equates (In terms of bytes).
		= 238 ;	
0002		= 239 WORD EQU 2	; 1 Word = 2 Bytes
0001		= 240 BYTE EQU 1	; 1 Byte = 1 Byte.
0400		= 241 K EQU 1024	; 1kbytes = 1024 bytes
		= 242 ;	
		243 \$INCLUDE(VT52.EQU)	
		= 244 ;	
		= 245 ;	-----
		= 246 ;	
		= 247 ;	VT52 ESC characters
		= 248 ;	
0048		= 249 HOMEC EQU 'H'	; Home cursor command
004A		= 250 CLRSC EQU 'J'	; Clear screen command
0059		= 251 GOTOC EQU 'Y'	; Direct cursor position command
004B		= 252 CLRLC EQU 'K'	; Clear line command
0049		= 253 REVLF EQU 'I'	; Reverse line feed
0020		= 254 OFFSET EQU 20H	; Direct cursor position offset
		= 255 ;	
0041		= 256 UPCMD EQU 'A'	; Up cursor command
0042		= 257 DNCMD EQU 'B'	; Down cursor command
0044		= 258 LFCMD EQU 'D'	; Left cursor command
0043		= 259 RTCMD EQU 'C'	; Right cursor command
		= 260 ;	
0000		= 261 MINX EQU 0	; Min x value (Col)
004F		= 262 MAXX EQU 79	; Max x value (Col)
0000		= 263 MINY EQU 0	; Min y value (Row)
0017		= 264 MAXY EQU 23	; Max y value (Row)
		= 265 ;	
		= 266 ;	
		= 267 ;	-----
		= 268 ;	
		269 PUBLIC PTEST	
		= 270 ;	
		= 271 ;	-----
		= 272 ;	
		273 EXTRN WAIT, GOTO, DELAY	
		274 EXTRN OUTMSG, CO	
		= 275 ;	
		= 276 ;	-----
		= 277 ;	
		278 CSEG	
		= 279 ;	
		= 280 ;	-----

Parallel port test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		281 ;	
		282 ;	Main parallel port test routine:
		283 ;	
		284 ;	This test configures each port in turn as an output, while setting
		285 ;	the remaining two ports to be inputs:
		286 ;	
0000	210C01	C 287	PTEST: ; Procedure ParallelTest;
0003	CD0000	E 288	LXI H,PTMSG ;
		289	CALL OUTMSG ; OutMessage(ParTestMessage);
		290 ;	
		291 ;	-----
		292 ;	
		293 ;	Testing with A as output, B and C input.
		294 ;	
0006	213101	C 295	LXI H,AOMSG ;
0009	CD0000	E 296	CALL OUTMSG ; OutMessage(PortAOutPutMessage);
		297 ;	
		298 ;	Setup registers A,B and C for the 1st test (A=out, B=in and C=in).
		299 ;	
000C	3E8B	300	MVI A,AOCMD ;
000E	D30F	301	OUT PPICONT ; PPIControlPort := PortAOutput;
		302	;
0010	3E0C	303	MVI A,PORTA ;
0012	010D0E	304	LXI B,PORTB + PORTC*100H ;
0015	1622	305	MVI D,2+OFFSET ;
0017	CD4600	C 306	CALL IPTEST ; ImplementTest(PORTA, PORTB, PORTC, 2);
		307 ;	
		308 ;	-----
		309 ;	
		310 ;	Testing with B as output, A and C input.
		311 ;	
001A	214C01	C 312	LXI H,BOMSG ;
001D	CD0000	E 313	CALL OUTMSG ; OutMessage(PortBOutPutMessage);
		314 ;	
		315 ;	Setup registers A,B and C for the 2nd test (A=in, B=out and C=in).
		316 ;	
0020	3E99	317	MVI A,BOCMD ;
0022	D30F	318	OUT PPICONT ; PPIControlPort := PortBOutput;
		319	;
0024	010C0E	320	LXI B,PORTA + PORTC*100H ;
0027	3E0D	321	MVI A,PORTB ;
0029	1624	322	MVI D,4+OFFSET ;
002B	CD4600	C 323	CALL IPTEST ; ImplementTest(PORTB, PORTA, PORTC, 4);
		324 ;	
		325 ;	-----
		326 ;	
		327 ;	Testing with C as output, A and B input.
		328 ;	
002E	216701	C 329	LXI H,COMSG ;
0031	CD0000	E 330	CALL OUTMSG ; OutMessage(PortCOutPutMessage);
		331 ;	
		332 ;	Setup registers A,B and C for the 3rd test (A=in, B=in and C=out).
		333 ;	
0034	3E92	334	MVI A,COCMD ;
0036	D30F	335	OUT PPICONT ; PPIControlPort := PortCOutput;
		336	;
0038	010D0C	337	LXI B,PORTB + PORTA*100H ;
003B	3E0E	338	MVI A,PORTC ;
003D	1626	339	MVI D,6+OFFSET ;
003F	CD4600	C 340	CALL IPTEST ; ImplementTest(PORTC, PORTA, PORTB, 6);
		341 ;	
		342 ;	-----
		343 ;	
0042	CD0000	E 344	CALL WAIT ; WaitForResponse;
0045	C9	345	RET ; EndProc;
		346 ;	
		347 ;	=====
		348 ;	
		349 ;	Implement test:
		350 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:22:43 20-APR-88 PAR PAGE 6
Parallel port test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		351 ;	Inputs: A = Output port address.
		352 ;	B = Input port address.
		353 ;	C = Input port address.
		354 ;	D = Line number.
		355 ;	
		356 ;	Outputs: None (Prints a message to indicate result)
		357 ;	
		358 IPTEST:	; Procedure ImplementTest(OutPort,
		359	; InPort1, InPort2, Line : Byte);
		360	;
0046	.CDE800	C 361	CALL DYNINIT ; DynamicInitialise;
0049	218201	C 362	LXI H,PATMSG ;
004C	CD0000	E 363	CALL OUTMSG ; OutMessage(PatternMessage);
		364	;
004F	1E00	365	MVI E,0 ; Error := 0;
		366	;
0051	3E01	367	MVI A,00000001B ; Pattern := 00000001B
		368	;
		369 IPT05:	; Repeat
0053	47	370	MOV B,A ;
0054	CDCC00	C 371	CALL PATPRT ; PrintPattern;
0057	CD0000	D 372	CALL OUT1 ; OutputIndirect(OutPort);
005A	CD0000	C 373	CALL PDELAY ; Delay;
005D	CD0300	D 374	CALL IN1 ; A := InputIndirect(InPort1);
0060	B8	375	CMP B ;
0061	CA6700	C 376	JZ IPT10 ; If A <> Pattern then
		377	; Begin
0064	7B	378	MOV A,E ;
0065	B0	379	ORA B ; Error := Error OR pattern;
0066	5F	380	MOV E,A ;
		381 IPT10:	; End;
		382	;
0067	CD0600	D 383	CALL IN2 ; A := InputIndirect(InPort2)
006A	B8	384	CMP B ;
006B	CA7100	C 385	JZ IPT15 ; If A <> Pattern then
		386	; Begin
006E	7B	387	MOV A,E ;
006F	B0	388	ORA B ; Error := Error OR Pattern;
0070	5F	389	MOV E,A ;
		390 IPT15:	; End;
0071	78	391	MOV A,B ;
0072	07	392	RLC ; Pattern := Pattern SHL 1;
0073	D25300	C 393	JNC IPT05 ; Until Carry;
		394	;
0076	CD7A00	C 395	CALL ERRREP ; ErrorReport(Line, Error);
0079	C9	396	RET ; EndProc;
		397 ;	
		398 ;=====	
		399 ;	
		400 ; D : Line	
		401 ; E : Error	
		402 ;	
		403 ERRREP:	; Procedure ErrorReport(Line, Error : Byte)
007A	0648	404	MVI B,PATPOS+OFFSET ;
007C	7A	405	MOV A,D ;
007D	CD0000	E 406	CALL GOTO ; Goto(PatternPosition, Line);
		407	;
0080	0608	408	MVI B,8 ;
		409 EREP05:	; For I := 1 to 8 do
		410	; Begin
0082	7B	411	MOV A,E ;
0083	07	412	RLC ;
0084	5F	413	MOV E,A ; Error := Error SHL 1;
		414	;
0085	3E2D	415	MVI A,'-' ;
0087	D28C00	C 416	JNC EREP10 ; If Carry then (Carry = Error)
008A	3E2A	417	MVI A,'*' ; Ch := '*'
		418 EREP10:	; Else
		419	; Ch := '-';
		420	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:22:43 20-APR-88

PAR

PAGE 7

Parallel port test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
008C	CD0000	E 421	CALL CO ; ConsoleOut(Ch);
008F	05	422	DCR B ;
0090	C28200	C 423	JNZ EREP05 ; End;
		424	;
0093	7B	425	MOV A,E ;
0094	B7	426	ORA A ;
0095	C29E00	C 427	JNZ EREP15 ; If Error = 0 then
		428	;
0098	21A500	C 429	LXI H,PTOK ; Message := 'Port OK'
009B	C3A100	C 430	JMP EREP20 ; Else
		431 EREP15:	;
009E	21B100	C 432	LXI H,PTEMSG ; Message := 'Port error'
		433 EREP20:	;
00A1	CD0000	E 434	CALL OUTMSG ; OutMessage(Message);
		435	;
00A4	C9	436	RET ; EndProc;
		437 ;	
00A5	20506F72	438 PTOK: DB	' Port OK',CR,LF,LF,0
	74204F4B		
	0D0A0A00		
00B1	20506F72	439 PTEMSG: DB	' Port error',CR,LF,LF,0
	74206572		
	726F720D		
	0A0A00		
		440 ;	
		441 ;=====	
		442 ;	
		443 PDELAY:	; Procedure PortDelay;
00C0	F5	444	PUSH PSW ;
00C1	3E32	445	MVI A,50 ;
		446 DEL05:	; For I := 1 to 50 do
00C3	CD0000	E 447	CALL DELAY ; Delay;
00C6	3D	448	DCR A ;
00C7	C2C300	C 449	JNZ DEL05 ;
00CA	F1	450	POP PSW ;
00CB	C9	451	RET ; EndProc;
		452 ;	
		453 ;=====	
		454 ;	
		455 ; D : Line	
		456 ; E : Error	
		457 ;	
		458 PATPRT:	; Procedure PrintPattern(Line, Error);
00CC	F5	459	PUSH PSW ;
00CD	C5	460	PUSH B ;
00CE	4F	461	MOV C,A ;
00CF	0648	462	MVI B,PATPOS+OFFSET ;
00D1	7A	463	MOV A,D ;
00D2	CD0000	E 464	CALL GOTO ; Goto(PatternPosition, Line);
00D5	0608	465	MVI B,8 ; For := 1 to 8 do
		466 PPRTC:	; Begin
00D7	79	467	MOV A,C ;
00D8	07	468	RLC ;
00D9	4F	469	MOV C,A ; Error := Error SHL 1;
00DA	3E30	470	MVI A,'0' ;
00DC	CE00	471	ACI 0 ; Ch := '0' + Carry;
		472	;
00DE	CD0000	E 473	CALL CO ; ConsoleOut(Ch);
00E1	05	474	DCR B ;
00E2	C2D700	C 475	JNZ PPRTC ; End;
		476	;
00E5	C1	477	POP B ;
00E6	F1	478	POP PSW ;
00E7	C9	479	RET ; EndProc;
		480 ;	
		481 ;=====	
		482 ;	
		483 ; Dynamic initialization of Input/Output routines:	
		484 ;	
		485 ; Inputs: A = Address of the output port.	

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:22:43 20-APR-88 PAR PAGE 9
Parallel port test routines for the UCT SABus diagnostics

```

LOC   OBJ       LINE      SOURCE STATEMENT
      616E6420
      4220696E
      70757473
      202000
0182  54657374    533 PATMSG: DB      'Test pattern: ',0
      20706174
      7465726E
      3A2000
      534 ;
0028  535 PATPOS EQU      $-COMSG-2
      536 ;
      537 ;-----
      538 ;
      539 ;      Parallel data area.
      540 ;
      541      DSEG
      542 ;
0000  543 OUT1: DS      BYTE          ; (OUT)
0001  544 OUTP: DS      BYTE          ; Address of output port
0002  545 RETOUT: DS     BYTE          ; (RET)
      546 ;
0003  547 IN1:  DS      BYTE          ; (IN)
0004  548 INP1:  DS      BYTE          ; Address of input port
0005  549 RETIN1: DS     BYTE          ; (RET)
      550 ;
0006  551 IN2:  DS      BYTE          ; (IN)
0007  552 INP2:  DS      BYTE          ; Address of 2nd input port
0008  553 RETIN2: DS     BYTE          ; (RET)
      554 ;
      555 ;      End of <DIAGPAR.A85> module
      556 ;
      557      END

```

0 error(s) detected

External symbols

CO DELAY GOTO OUTMSG WAIT

Public symbols

PTEST

User symbols

ACK	0006 A	AOCMD	008B A	AOMSG	0131 C	ASUB	001A A	BELL	0007 A	BOCMD	0099 A	BOMSG	014C C
BS	0008 A	BYTE	0001 A	CAN	0018 A	CLRLC	004B A	CLRSC	004A A	CO	0004 E	COCMD	0092 A
COMSG	0167 C	CR	000D A	CTRLMA	001F A	CTRLQ	0011 A	CTRLS	0013 A	CTRLZ	001A A	DATA5	0000 A
DATA6	0004 A	DATA7	0008 A	DATAB	000C A	DB110	0369 A	DB1200	0050 A	DB1800	0035 A	DB2400	0028 A
DB300	0140 A	DB3600	001B A	DB4800	0014 A	DB600	00A0 A	DB7200	000D A	DB9600	000A A	DC1	0011 A
DC2	0012 A	DC3	0013 A	DC4	0014 A	DEL	007F A	DEL05	00C3 C	DELAY	0003 E	DIV1	0001 A
DIV16	0002 A	DIV64	0003 A	DLE	0010 A	DNCMD	0042 A	DSR	0080 A	DTR	0002 A	DYNINI	0028 C
EN	0019 A	ENQ	0005 A	EOT	0004 A	EPDATA	0020 A	EPNUM	0023 A	EREPO5	0082 C	EREPI0	008C C
EREPI5	009E C	EREPI20	00A1 C	ERESET	0010 A	ERMASK	0038 A	ERRREP	007A C	ESC	001B A	ETB	0017 A
ETC1	0038 A	ETC2	00CB A	ETX	0003 A	EVENPA	0030 A	FALSE	0000 A	FF	000C A	FRMERR	0020 A
FS	001C A	GOTO	0000 E	GOTOC	0059 A	GS	001D A	HOMEC	0048 A	HT	0009 A	HUNTM	0080 A
IN1	0003 D	IN2	0006 D	INP1	0004 D	INP2	0007 D	INVSTO	0000 A	IOBASE	0000 A	IPT05	0053 C
IPT10	0067 C	IPT15	0071 C	IPTEST	0046 C	IRESET	0040 A	K	0400 A	LF	000A A	LFCND	0044 A
LSBYTE	0021 A	MAXX	004F A	MAXY	0017 A	MIX	0000 A	MINY	0000 A	MSBYTE	0022 A	MSCARD	0020 A
NAK	0015 A	NAMEA	0003 A	NOPARI	0000 A	NUL	0000 A	ODDPAR	0010 A	OFFSET	0020 A	ONEAND	0080 A
ONESTO	0040 A	OUNODE	004E A	OUT1	0000 D	OUTMSG	0002 E	OUTP	0001 D	OVRERR	0010 A	PARERR	0008 A
PATMSG	0182 C	PATPOS	0028 A	PATPRT	00CC C	PDELAY	00C0 C	PORTA	000C A	PORTB	000D A	PORTC	000E A
PPIBAS	000C A	PPICON	000F A	PPRTC	00D7 C	PTMSG	00B1 C	PTEST	0000 C	PTMSG	010C C	PTOK	00A5 C
QUOTE	0027 A	RETIN1	0005 D	RETIN2	0008 D	RETOUT	0002 D	REVL	0049 A	RS	001E A	RST55A	002C A
RST65A	0034 A	RST75A	003C A	RTCMD	0043 A	RTS	0020 A	RXENAB	0004 A	RXRDY	0002 A	SBREAK	0008 A
SI	000F A	SO	000E A	SOH	0001 A	SPACE	0020 A	STX	0002 A	SYN	0016 A	SYNMO	0000 A
SYNDET	0040 A	TODATA	0008 A	TIDATA	0009 A	T2DATA	000A A	TAB	0009 A	TBASE	0008 A	TCONT	000B A

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:22:43 20-APR-88 PAR PAGE 10
 Parallel port test routines for the UCT SABus diagnostics :

LOC	OBJ	LINE	SOURCE STATEMENT
TIMER0 0000 A	TIMER1 0040 A	TIMER2 0080 A	TMODE 0036 A TMODE0 0036 A TMODE1 0076 A TMODE2 00B6 A
TRUE 00FF A	TWOSTO 00C0 A	TXEMP 0004 A	TXENAB 0001 A TXRDY 0001 A UOBASE 0002 A UOCONT 0003 A
UODATA 0002 A	UOSTAT 0003 A	UIBASE 0004 A	UICONT 0005 A UI1DATA 0004 A UI1STAT 0005 A UCMD 0015 A
UDUMMY 0002 A	UMODE 007E A	UPCMD 0041 A	URESET 0040 A US 001F A VT 000B A WAIT 0001 E
WORD 0002 A			

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$TITLE('RS232 port test procedures for the UCT SABus diagnostics')
		2	\$PAGELENGTH(75)
		3	;
		4	*****
		5	;
		6	E: Serial I/O card test. (file=DIAGRS232.A85)
		7	;
		8	*****
		9	;
		10	NAME RS232
		11	;
		12	\$INCLUDE(HARD.EQU) ; Include hardware equates
		13	;
		14	-----
		15	;
		16	Hardware equates for the UCT SABus kits:
		17	;
		18	8085 interrupt vectors:
		19	;
002C		20	RST55A EQU 2CH ; Restart 5.5 address
0034		21	RST65A EQU 34H ; Restart 6.5 address
003C		22	RST75A EQU 3CH ; Restart 7.5 address
		23	;
		24	Mass storage equates.
		25	;
0020		26	MSCARD EQU 20H ; Mass storage card - base address
		27	;
0020		28	EPDATA EQU MSCARD+0 ; MSC : Data register
0021		29	LSBYTE EQU MSCARD+1 ; MSC : LOW byte of EPROM address
0022		30	MSBYTE EQU MSCARD+2 ; MSC : HIGH byte of EPROM address
0023		31	EPNUM EQU MSCARD+3 ; MSC : Eprom number register
		32	;
0038		33	ETC1 EQU 038H ; EPROM There Code, first byte.
00CB		34	ETC2 EQU 0CBH ; EPROM There Code, second byte.
		35	;
0003		36	NAMEA EQU 3 ; File Name offset
		37	;
0000		38	IOBASE EQU 0 ; I/O Card base address
		39	;
0002		40	UOBASE EQU IOBASE+02H ; USART0 base address (8251)
0004		41	UIBASE EQU IOBASE+04H ; USART1 base address (8251)
0008		42	TBASE EQU IOBASE+08H ; Timer Base address (8253)
000C		43	PPIBASE EQU IOBASE+0CH ; PPI base address (8255)
		44	;
		45	----- U0 registers. -----
		46	;
0002		47	UODATA EQU UOBASE+0 ; USART 0 : Data register
0003		48	UOSTAT EQU UOBASE+1 ; USART 0 : Status register
0003		49	UOCONT EQU UOSTAT ; USART 0 : Control register
		50	;
		51	----- U1 registers. -----
		52	;
0004		53	U1DATA EQU UIBASE+0 ; USART 1 : Data register
0005		54	U1STAT EQU UIBASE+1 ; USART 1 : Status register
0005		55	U1CONT EQU U1STAT ; USART 1 : Control register
		56	;
		57	----- Timer registers. -----
		58	;
0008		59	TODATA EQU TBASE+0 ; TIMER : Counter 0 data register
0009		60	T1DATA EQU TBASE+1 ; TIMER : Counter 1 data register
000A		61	T2DATA EQU TBASE+2 ; TIMER : Counter 2 data register
000B		62	TCONT EQU TBASE+3 ; TIMER : Control register
		63	;
		64	----- PPI registers -----
		65	;
000C		66	PORTA EQU PPIBASE+0 ; PPI : PORT A
000D		67	PORTB EQU PPIBASE+1 ; PPI : PORT B
000E		68	PORTC EQU PPIBASE+2 ; PPI : PORT C
000F		69	PPICONT EQU PPIBASE+3 ; PPI : Control/Status register
		70	;

LOC	OBJ	LINE	SOURCE STATEMENT
		= 71 ;	-----
		= 72 ;	
		= 73 ;	I/O equates, timer/USART control/Mode words.
		= 74 ;	
0002		= 75 UDUMMY EQU 02H	; USART dummy mode/control word
0040		= 76 URESET EQU 40H	; USART reset control word
		= 77 ;	
0000		= 78 INVSTOPBIT EQU 00B SHL 6	; Invalid stop bit spec
0040		= 79 ONESTOPBIT EQU 01B SHL 6	; One stop bit
0080		= 80 ONEANDAHALFSTOPBIT EQU 10B SHL 6	; One and a half stop bits
00C0		= 81 TWOSTOPBIT EQU 11B SHL 6	; Two stop bits
		= 82 ;	
0000		= 83 NOPARITY EQU 00B SHL 4	; No parity
0010		= 84 ODDPARITY EQU 01B SHL 4	; Odd parity
0030		= 85 EVENPARITY EQU 11B SHL 4	; Even parity
		= 86 ;	
0000		= 87 DATA5 EQU 00B SHL 2	; 5 data bits
0004		= 88 DATA6 EQU 01B SHL 2	; 6 data bits
0008		= 89 DATA7 EQU 10B SHL 2	; 7 data bits
000C		= 90 DATA8 EQU 11B SHL 2	; 8 data bits
		= 91 ;	
0000		= 92 SYNCMODE EQU 00B SHL 0	; Synchronous mode
0001		= 93 DIV1 EQU 01B SHL 0	; Clock x1
0002		= 94 DIV16 EQU 10B SHL 0	; Clock x16
0003		= 95 DIV64 EQU 11B SHL 0	; Clock x64
		= 96 ;	
		= 97 ;	USART mode word.
		= 98 ;	
007E		= 99 UMODE EQU ONESTOPBIT + EVENPARITY + DATA8 + DIV16	
004E		= 100 OUMODE EQU ONESTOPBIT + NOPARITY + DATA8 + DIV16	
		= 101 ;	
		= 102 ;	USART command register format
		= 103 ;	
0001		= 104 TXENAB EQU 00000001B	; Transmitter enable bit (Command)
0002		= 105 DTR EQU 00000010B	; Position of the DTR bit (Command)
0004		= 106 RXENAB EQU 00000100B	; Receiver enable bit (Command)
0008		= 107 SBREAK EQU 00001000B	; Send break (Command)
0010		= 108 ERESET EQU 00010000B	; Error reset bit (Command)
0020		= 109 RTS EQU 00100000B	; Request to send bit (Command)
0040		= 110 IRESET EQU 01000000B	; Internal reset bit (Command)
0080		= 111 HUNTM EQU 10000000B	; Enter hunt mode (Command)
		= 112 ;	
		= 113 ;	USART command word:
		= 114 ;	
0015		= 115 UCMD EQU TXENAB + RXENAB + ERESET	
		= 116 ;	
		= 117 ;	USART Status register format:
		= 118 ;	
0001		= 119 TXRDY EQU 00000001B	; Transmitter ready (Status)
0002		= 120 RXRDY EQU 00000010B	; Receiver ready (Status)
0004		= 121 TXEMP EQU 00000100B	; Transmitter empty (Status)
0008		= 122 PARERR EQU 00001000B	; Parity error flag (Status)
0010		= 123 OVRERR EQU 00010000B	; Overrun error flag (Status)
0020		= 124 FRMERR EQU 00100000B	; Framing error flag (Status)
0040		= 125 SYNDET EQU 01000000B	; Sync detect (Status)
0080		= 126 DSR EQU 10000000B	; DSR bit (Status)
		= 127 ;	
0038		= 128 ERMASK EQU 00111000B	; Combination error mask
		= 129 ;	
		= 130 ;	Timer commands and division factor.
		= 131 ;	
0000		= 132 TIMERO EQU 0 SHL 6	; Address timer 0
0040		= 133 TIMER1 EQU 1 SHL 6	; Address timer 1
0080		= 134 TIMER2 EQU 2 SHL 6	; Address timer 2
		= 135 ;	
0036		= 136 TMODE EQU 00110110B	; Basic mode word =
		= 137	; (Read and write LSB then MSB,
		= 138	; Mode 3,
		= 139	; Count in binary).
		= 140	;

LOC	OBJ	LINE	SOURCE STATEMENT
0036		= 141 TMODE0	EQU TMODE+TIMER0 ; Mode word for timer 0
0076		= 142 TMODE1	EQU TMODE+TIMER1 ; Mode word for timer 1
00B6		= 143 TMODE2	EQU TMODE+TIMER2 ; Mode word for timer 2
		= 144 ;	
		= 145 ;	Baud rate division factors for the USART, XTAL = 6.144 MHz
		= 146 ;	
0369		= 147 DB110	EQU 873 ; Division ratio for 110 baud
0140		= 148 DB300	EQU 320 ; Division ratio for 300 baud
00A0		= 149 DB600	EQU 160 ; Division ratio for 600 baud
0050		= 150 DB1200	EQU 80 ; Division ratio for 1200 baud
0035		= 151 DB1800	EQU 53 ; Division ratio for 1800 baud
0028		= 152 DB2400	EQU 40 ; Division ratio for 2400 aud
001B		= 153 DB3600	EQU 27 ; Division ratio for 3600 baud
0014		= 154 DB4800	EQU 20 ; Division ratio for 4800 baud
000D		= 155 DB7200	EQU 13 ; Division ratio for 7200 baud
000A		= 156 DB9600	EQU 10 ; Division ratio for 9600 baud
		= 157 ;	
		= 158 ;	PPI commands:
		= 159 ;	
008B		= 160 AOCMD	EQU 10001011B ; A = Mode 0 output
		= 161	; B = Mode 0 input
		= 162	; C (upper/lower) = Mode 0 input
		= 163 ;	
0099		= 164 BOCMD	EQU 10011001B ; A = Mode 0 input
		= 165	; B = Mode 0 output
		= 166	; C (upper/lower) = Mode 0 input
		= 167 ;	
0092		= 168 COCMD	EQU 10010010B ; A = Mode 0 input
		= 169	; B = Mode 0 input
		= 170	; C (upper/lower) = Mode 0 output
		= 171 ;	
		= 172 ;	END HARD
		173 \$INCLUDE(ASCII.EQU)	; Include ASCII equates
		= 174 ;	
		= 175 ;	-----
		= 176 ;	
001F		= 177 CTRLMASK	EQU 00011111B ; Control mask.
		= 178 ;	
		= 179 ;	ASCII characters:
		= 180 ;	
0000		= 181 NUL	EQU 000H ; ASCII null character
0001		= 182 SOH	EQU 001H ; ASCII Start of header character
0002		= 183 STX	EQU 002H ; ASCII Start of transmission character
0003		= 184 ETX	EQU 003H ; ASCII End of transmission character
0004		= 185 EOT	EQU 004H ; ASCII End of text marker.
0005		= 186 ENQ	EQU 005H ; ASCII Enquire character
0006		= 187 ACK	EQU 006H ; ASCII Ack character
0007		= 188 BELL	EQU 007H ; ASCII Bell character.
0008		= 189 BS	EQU 008H ; ASCII Back space.
0009		= 190 HT	EQU 009H ; ASCII Horiz tab character.
0009		= 191 TAB	EQU 009H ; ASCII Horiz tab character.
000A		= 192 LF	EQU 00AH ; ASCII Line feed character.
000B		= 193 VT	EQU 00BH ; ASCII Vertical tab character
000C		= 194 FF	EQU 00CH ; ASCII Form Feed character.
000D		= 195 CR	EQU 00DH ; ASCII Carriage return character.
000E		= 196 SO	EQU 00EH ; ASCII Shift out character
000F		= 197 SI	EQU 00FH ; ASCII Shift in character
		= 198 ;	
0010		= 199 DLE	EQU 010H ; ASCII DLE character.
0011		= 200 DC1	EQU 011H ; ASCII Device control #1 (Xon)
0012		= 201 DC2	EQU 012H ; ASCII Device control #2
0013		= 202 DC3	EQU 013H ; ASCII Device control #3 (Xoff)
0014		= 203 DC4	EQU 014H ; ASCII Device control #4
0015		= 204 NAK	EQU 015H ; ASCII Negative ACK character
0016		= 205 SYN	EQU 016H ; ASCII Sync character
0017		= 206 ETB	EQU 017H ; ASCII End of transmission buffer.
0018		= 207 CAN	EQU 018H ; ASCII Cancel.
0019		= 208 EM	EQU 019H ; ASCII
001A		= 209 ASUB	EQU 01AH ; ASCII Submit character.
001B		= 210 ESC	EQU 01BH ; ASCII Escape character.

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:23:13 20-APR-88

RS232 PAGE 4

RS232 port test procedures for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
001C		= 211 FS	EQU 01CH ; ASCII Field separator.
001D		= 212 GS	EQU 01DH ; ASCII Group separator
001E		= 213 RS	EQU 01EH ; ASCII Record separator
001F		= 214 US	EQU 01FH ; ASCII Unit separator.
		= 215 ;	
		= 216 ;	Control characters:
		= 217 ;	
0013		= 218 CTRLS	EQU CTRLMASK AND 'S' ; Control 'S'
0011		= 219 CTRLQ	EQU CTRLMASK AND 'Q' ; Control 'Q'
001A		= 220 CTRLZ	EQU CTRLMASK AND 'Z' ; Control 'Z'
		= 221 ;	
		= 222 ;	Some more ASCII characters:
		= 223 ;	
0020		= 224 SPACE	EQU 020H ; ASCII Space.
0027		= 225 QUOTE	EQU 027H ; Single quote (')
007F		= 226 DEL	EQU 07FH ; ASCII Delete character.
		= 227 ;	
		= 228 \$INCLUDE(VT52.EQU)	; Include VT52 command
		= 229 ;	
		= 230 ;	-----
		= 231 ;	
		= 232 ;	VT52 ESC characters
		= 233 ;	
0048		= 234 HOMEC	EQU 'H' ; Home cursor command
004A		= 235 CLRSC	EQU 'J' ; Clear screen command
0059		= 236 GOTOC	EQU 'Y' ; Direct cursor position command
004B		= 237 CLRLC	EQU 'K' ; Clear line command
0049		= 238 REVLf	EQU 'I' ; Reverse line feed
0020		= 239 OFFSET	EQU 20H ; Direct cursor position offset
		= 240 ;	
0041		= 241 UPCMD	EQU 'A' ; Up cursor command
0042		= 242 DNCMD	EQU 'B' ; Down cursor command
0044		= 243 LFCMD	EQU 'D' ; Left cursor command
0043		= 244 RTCMD	EQU 'C' ; Right cursor command
		= 245 ;	
0000		= 246 MINX	EQU 0 ; Min x value (Col)
004F		= 247 MAXX	EQU 79 ; Max x value (Col)
0000		= 248 MINY	EQU 0 ; Min y value (Row)
0017		= 249 MAXY	EQU 23 ; Max y value (Row)
		= 250 ;	
		= 251 \$INCLUDE(COMEQU.EQU)	; Include common equates
		= 252 ;	
		= 253 ;	-----
		= 254 ;	
		= 255 ;	Common equates:
		= 256 ;	
0000		= 257 FALSE	EQU 0 ; Logical FALSE.
00FF		= 258 TRUE	EQU (NOT FALSE) AND OFFH ; Logical TRUE.
		= 259 ;	
		= 260 ;	Storage size Equates (In terms of bytes).
		= 261 ;	
0002		= 262 WORD	EQU 2 ; 1 Word = 2 Bytes
0001		= 263 BYTE	EQU 1 ; 1 Byte = 1 Byte.
0400		= 264 K	EQU 1024 ; 1kbytes = 1024 bytes
		= 265 ;	
		= 266 ;	
		= 267 ;	-----
		= 268 ;	
		= 269	PUBLIC RTEST ; Make test routine entry point public
		= 270 ;	
		= 271 ;	-----
		= 272 ;	
		= 273	EXTRN CLRS, WAIT, DELAY
		= 274	EXTRN OUTMSG, UOO, UO1, UIO, UI1, UOTEST, U1TEST
		= 275 ;	
		= 276 ;	-----
		= 277 ;	
		= 278	CSEG ; Procedure RS232_Test;
		= 279	;
		= 280 RTEST:	;

DIAG LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:23:13 20-APR-88 RS232 PAGE 5
RS232 port test procedures for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
0000	CD0000	E 281	CALL CLRS ; ClearScreen;
		282 ;	
		283 ;	OutMessage('Testing transmission from USART0 to USART1')
		284 ;	
0003	211B02	C 285	LXI H,RTM01 ;
0006	CD0000	E 286	CALL OUTMSG ;
		287 ;	
0009	CDB301	C 288	CALL UINIT ; USART_Initialise;
		289 ;	
		290 ;	-----
		291 ;	
		292 ;	Test transmission from USART 0 to USART 1.
		293 ;	
000C	3E00	294	MVI A,0 ; Ch := 0;
		295	RST05: ;
000E	47	296	MOV B,A ;
000F	CD0000	E 297	CALL U00 ; U0_Output(Ch);
0012	CD0B02	C 298	CALL UI1WT ; U1_Wait_With_Timeout;
		299 ;	
0015	CAB300	C 300	JZ TIM01 ; IF TimeOut Then ExitWithError;
		301 ;	
0018	CD0000	E 302	CALL UI1 ; Tch := U1_Input;
001B	B8	303	CMP B ; If Tch <> Ch then ExitWithError;
001C	C20301	C 304	JNZ VERO1 ;
		305 ;	
001F	DB05	306	IN U1STAT ;
0021	E638	307	ANI ERMASK ;
0023	C25101	C 308	JNZ UERR1 ; If (USART1_Status AND error mask) <> 0
		309 ;	Then ExitWithError;
		310 ;	
0026	78	311	MOV A,B ;
0027	3C	312	INR A ; Ch := (Ch + 1) MOD 256;
		313 ;	
0028	C20E00	C 314	JNZ RST05 ; Until Ch = 0;
		315 ;	
002B	21D302	C 316	LXI H,OKMSG ;
002E	CD0000	E 317	CALL OUTMSG ; OutMessage('OK');
		318 ;	
		319 ;	-----
		320 ;	
		321 ;	Test transmission from USART 0 to USART 1.
		322 ;	
		323 ;	
		324 ;	OutMessage('Testing transmission from USART1 to USART0')
		325 ;	
0031	214F02	C 326	LXI H,RTM10 ;
0034	CD0000	E 327	CALL OUTMSG ;
0037	3E00	328	MVI A,0 ; Ch := 0;
		329	RST10: ; Repeat
0039	47	330	MOV B,A ;
003A	CD0000	E 331	CALL U01 ; U1_Output(Ch);
003D	CDFB01	C 332	CALL UIOWT ; U0_Wait_with_timeout;
0040	CADB00	C 333	JZ TIM10 ; If timeOut then ExitWithError;
		334 ;	
0043	CD0000	E 335	CALL UI0 ; Tch := U1_Input;
0046	B8	336	CMP B ; If Tch <> Ch then ExitWithError;
0047	C22A01	C 337	JNZ VERO1 ;
		338 ;	
004A	DB03	339	IN U0STAT ;
004C	E638	340	ANI ERMASK ;
004E	C27301	C 341	JNZ UERR0 ; If (USART0_Status AND error mask) <> 0
		342 ;	Then ExitWithError;
0051	78	343	MOV A,B ;
0052	3C	344	INR A ; Ch := (Ch + 1) MOD 256;
		345 ;	
0053	C23900	C 346	JNZ RST10 ; Until Ch := 0;
		347 ;	
0056	21D302	C 348	LXI H,OKMSG ;
0059	CD0000	E 349	CALL OUTMSG ; OutMessage('OK');
		350 ;	

LOC	OBJ	LINE	SOURCE STATEMENT
		351 ;	-----
		352 ;	
		353 ;	Test hand shake lines DTR(0) and DSR(1).
		354 ;	
		355 ;	OutMessage('Testing DSR (USART1) to DTR (USART0)');
		356 ;	
005C	218302	C 357	LXI H,RTM15 ;
005F	CD0000	E 358	CALL OUTMSG ;
		359	;
0062	3E15	360	MVI A,UCMD ;
0064	D303	361	OUT UOCONT ; Out(UO_Control, USART_Command);
		362	;
0066	CD0000	E 363	CALL DELAY ; Delay;
		364	;
0069	DB05	365	IN U1STAT ; Status := In(U1_Status)
006B	E680	366	ANI DSR ;
006D	C29501	C 367	JNZ DSRERR ; If Status AND DSR <> 0 then
		368	ExitWithError;
		369	;
0070	3E17	370	MVI A,UCMD + DTR ;
0072	D303	371	OUT UOCONT ; Out(UO_Control, USART_Command+DSR);
		372	;
0074	CD0000	E 373	CALL DELAY ; Delay;
		374	;
0077	DB05	375	IN U1STAT ; Status := In(U1_Status);
0079	E680	376	ANI DSR ;
007B	CA9501	C 377	JZ DSRERR ; If Status AND DSR = 0 then
		378	ExitWithError;
007E	21D302	C 379	LXI H,OKMSG ;
0081	CD0000	E 380	CALL OUTMSG ; OutMessage('OK');
		381 ;	
		382 ;	-----
		383 ;	
		384 ;	Test hand shake lines DTR(1) and DSR(0).
		385 ;	
		386 ;	OutMessage('Testing DSR (USART0) to DTR (USART1)');
		387 ;	
0084	21AB02	C 388	LXI H,RTM20 ;
0087	CD0000	E 389	CALL OUTMSG ;
		390	;
008A	3E15	391	MVI A,UCMD ;
008C	D305	392	OUT U1CONT ; Out(U1_control, USART_Command);
		393	;
008E	CD0000	E 394	CALL DELAY ; Delay;
		395	;
0091	DB03	396	IN U0STAT ; Status := In(U0_status);
0093	E680	397	ANI DSR ;
0095	C29501	C 398	JNZ DSRERR ; If Status AND DSR <> 0 then
		399	ExitWithError;
		400	;
0098	3E17	401	MVI A,UCMD + DTR ;
009A	D305	402	OUT U1CONT ; Out(U1_control, USART_Command + DTR);
		403	;
009C	CD0000	E 404	CALL DELAY ; Delay;
		405	;
009F	DB03	406	IN U0STAT ; Status := In(U0_status);
00A1	E680	407	ANI DSR ;
00A3	CA9501	C 408	JZ DSRERR ; If Status AND DST = 0 then
		409	ExitWithError;
00A6	21D302	C 410	LXI H,OKMSG ;
00A9	CD0000	E 411	CALL OUTMSG ; OutMessage('OK');
		412	;
00AC	CDE801	C 413	CALL OUINIT ; ResetUSART;
		414	;
00AF	CD0000	E 415	CALL WAIT ; WaitForResponse;
		416	;
00B2	C9	417	RET ; EndProc;
		418 ;	
		419 ;	-----
		420 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:23:13 20-APR-88

RS232 PAGE 7

RS232 port test procedures for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		421 ;	Error exit points.
		422 ;	
		423 TIM01:	
00B3	21B900 C	424 LXI	H,TIM01M ; Message := TimeOutMessage01;
00B6	C3AF01 C	425 JMP	RTERRE ;
00B9	54696D65	426 TIM01M: DB	'Timeout error USART0 to USART1'
	6F757420		
	6572726F		
	72205553		
	41525430		
	20746F20		
	55534152		
	5431		
00D7	0D0A0A00	427 DB	CR,LF,LF,0 ;
		428 ;	
		429 TIM10:	
00DB	21E100 C	430 LXI	H,TIM10M ; Message := TimeOutMessage10;
00DE	C3AF01 C	431 JMP	RTERRE ;
00E1	54696D65	432 TIM10M: DB	'Timeout error USART1 to USART0'
	6F757420		
	6572726F		
	72205553		
	41525431		
	20746F20		
	55534152		
	5430		
00FF	0D0A0A00	433 DB	CR,LF,LF,0
		434 ;	
		435 VER01:	
0103	210901 C	436 LXI	H,VER01M ; Message := VerifyErrorMessage01;
0106	C3AF01 C	437 JMP	RTERRE ;
0109	56657269	438 VER01M: DB	'Verify error USART0 to USART1'
	66792065		
	72726F72		
	20555341		
	52543020		
	746F2055		
	53415254		
	31		
0126	0D0A0A00	439 DB	CR,LF,LF,0
		440 ;	
		441 VER10:	
012A	213001 C	442 LXI	H,VER10M ; Message := VerifyErrorMessage10;
012D	C3AF01 C	443 JMP	RTERRE ;
0130	56657269	444 VER10M: DB	'Verify error USART1 to USART0'
	66792065		
	72726F72		
	20555341		
	52543120		
	746F2055		
	53415254		
	30		
014D	0D0A0A00	445 DB	CR,LF,LF,0
		446 ;	
		447 UERR1:	
0151	215701 C	448 LXI	H,UERR1M ; Message := USART1_Hardware_error;
0154	C3AF01 C	449 JMP	RTERRE ;
0157	48617264	450 UERR1M: DB	'Hardware error on USART1'
	77617265		
	20657272		
	6F72206F		
	6E205553		
	41525431		
016F	0D0A0A00	451 DB	CR,LF,LF,0
		452 ;	
		453 UERR0:	
0173	217901 C	454 LXI	H,UERR0M ; Message := USART0_HardWare_error;
0176	C3AF01 C	455 JMP	RTERRE ;
0179	48617264	456 UERR0M: DB	'Hardware error on USART0'
	77617265		

LOC	OBJ	LINE	SOURCE STATEMENT
	20657272		
	6F72206F		
	6E205553		
	41525430		
0191	OD0A0A00	457	DB CR,LF,LF,0
		458	;
		459	DSRERR: ;
0195	219B01 C	460	LXI H,DSREM ; Message := DSR_DTR_Error_message;
0198	C3AF01 C	461	JMP RTERRE ;
019B	44545220	462	DSREM: DB 'DTR to DSR error',CR,LF,LF,0
	746F2044		
	53522065		
	72726F72		
	OD0A0A00		
		463	;
		464	RTERRE: ;
Q1AF	CD0000 E	465	CALL OUTMSG ; OutMessage(Message)
01B2	C9	466	RET ;
		467	;
		468	-----
		469	;
		470	UINIT: ; Procedure USART_Initialise;
01B3	3E36	471	MVI A,TMODE0 ;
01B5	D30B	472	OUT TCONT ; TimerControl := TimerModeWord0;
01B7	3E0A	473	MVI A,LOW DB9600 ;
01B9	D308	474	OUT TODATA ; Timer0 := LOW DivisionRatio9600;
01BB	3E00	475	MVI A,HIGH DB9600 ;
01BD	D308	476	OUT TODATA ; Timer0 := HIGH DivisionRatio9600;
		477	;
01BF	3E76	478	MVI A,TMODE1 ;
01C1	D30B	479	OUT TCONT ; TimerControl := TimerModeWord1;
01C3	3E0A	480	MVI A,LOW DB9600 ;
01C5	D309	481	OUT T1DATA ; Timer1 := LOW DivisionRatio9600;
01C7	3E00	482	MVI A,HIGH DB9600 ;
01C9	D309	483	OUT T1DATA ; Timer1 := HIGH DivisionRatio9600;
		484	;
01CB	3E02	485	MVI A,UDUMMY ;
01CD	D303	486	OUT UOCONT ; USART0_Control := Dummy;
01CF	D305	487	OUT U1CONT ; USART1_Control := Dummy;
01D1	D303	488	OUT UOCONT ; USART0_Control := Dummy;
01D3	D305	489	OUT U1CONT ; USART1_Control := Dummy;
		490	;
01D5	3E40	491	MVI A,URESET ;
01D7	D303	492	OUT UOCONT ; USART0_Control := Reset;
01D9	D305	493	OUT U1CONT ; USART1_Control := Reset;
		494	;
01DB	3E7E	495	MVI A,UMODE ;
01DD	D303	496	OUT UOCONT ; USART0_Control := USART_Mode_Word;
01DF	D305	497	OUT U1CONT ; USART1_Control := USART_Mode_Word;
		498	;
01E1	3E15	499	MVI A,UCMD ;
01E3	D303	500	OUT UOCONT ; USART0_Control := USART_Command_Word;
01E5	D305	501	OUT U1CONT ; USART1_Control := USART_Command_Word;
		502	;
01E7	C9	503	RET ; EndProc;
		504	;
		505	-----
		506	;
		507	Reset USART to their oroginal settings:
		508	;
		509	OUNIT: ; Procedure Original_USART_initialise;
01E8	3E40	510	MVI A,URESET ;
01EA	D303	511	OUT UOCONT ; USART0_Control := Reset;
01EC	D305	512	OUT U1CONT ; USART1_Control := Reset;
		513	;
01EE	3E4E	514	MVI A,OU MODE ;
01F0	D303	515	OUT UOCONT ; USART0_Control := Orig_USART_Mode_Word;
01F2	D305	516	OUT U1CONT ; USART1_Control := Orig_USART_Mode_Word;
		517	;
01F4	3E15	518	MVI A,UCMD ;

LOC	OBJ	LINE	SOURCE STATEMENT
01F6	D303	519	OUT UOCONT ; USART0_Control := USART_Command_Word;
01F8	D305	520	OUT UICONT ; USART1_Control := USART_Command_Word;
		521	;
01FA	C9	522	RET ; EndProc;
		523	;
		524	-----
		525	;
		526	UIOWT: ; Procedure
		527	;
01FB	21FFFF	528	LXI H,OFFFHH ; TimeOutCount := OFFFHH;
		529	UIOWTL: ; Repeat
		530	;
01FE	CD0000	E 531	CALL UOTEST ; CharReady := USART0_Test;
0201	C20A02	C 532	JNZ UIOWTE ;
0204	2B	533	DCX H ; TimeOutCount := TimeOutCount - 1;
0205	7D	534	MOV A,L ;
0206	B4	535	ORA H ;
0207	C2FE01	C 536	JNZ UIOWTL ; Until TimeOutCount = 0 or CharReady;
		537	UIOWTE: ;
020A	C9	538	RET ; EndProc;
		539	;
		540	-----
		541	;
		542	UI1WT: ; Procedure
		543	;
020B	21FFFF	544	LXI H,OFFFHH ; TimeOutCount := OFFFHH;
		545	UI1WTL: ; Repeat
		546	;
020E	CD0000	E 547	CALL UI1TEST ; CharReady := USART1_Test;
0211	C21A02	C 548	JNZ UI1WTE ;
0214	2B	549	DCX H ; TimeOutCount := TimeOutCount - 1;
0215	7D	550	MOV A,L ;
0216	B4	551	ORA H ;
0217	C20E02	C 552	JNZ UI1WTL ; Until TimeOutCount = 0 or CharReady;
		553	UI1WTE: ;
021A	C9	554	RET ; EndProc;
		555	;
		556	-----
		557	;
		558	RS232 Message data.
		559	;
021B	54657374	560	RTM01: DB 'Testing transmission from USART0 to USART1
	696E6720		
	7472616E		
	736D6973		
	73696F6E		
	2066726F		
	6D205553		
	41525430		
	20746F20		
	55534152		
	5431202E		
	2E2E2E2E		
024B	0D0A0A00	561	DB CR,LF,LF,0
		562	;
024F	54657374	563	RTM10: DB 'Testing transmission from USART1 TO USART0
	696E6720		
	7472616E		
	736D6973		
	73696F6E		
	2066726F		
	6D205553		
	41525431		
	20544F20		
	55534152		
	5430202E		
	2E2E2E2E		
027F	0D0A0A00	564	DB CR,LF,LF,0
		565	;
0283	54657374	566	RTM15: DB 'Testing DSR (USART1) to DTR (USART0)'

DIAG LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:23:13 20-APR-88
RS232 port test procedures for the UCT SABus diagnostics

RS232 PAGE 10

LOC	OBJ	LINE	SOURCE STATEMENT
	696E6720		
	44535220		
	28555341		
	52543129		
	20746F20		
	44545220		
	28555341		
	52543029		
02A7	0D0A0A00	567	DB CR,LF,LF,0
		568 ;	
02AB	54657374	569	RTM20: DB 'Testing DSR (USART0) to DTR (USART1)'
	696E6720		
	44535220		
	28555341		
	52543029		
	20746F20		
	44545220		
	28555341		
	52543129		
02CF	0D0A0A00	570	DB CR,LF,LF,0
		571 ;	
02D3	4F4B0DOA 0A00	572	OKMSG: DB 'OK',CR,LF,LF,0
		573 ;	
		574 ;	End of <DIAGRS232.A85> module
		575 ;	
		576	END

0 error(s) detected

External symbols

CLRS	DELAY	OUTMSG	UOTEST	U1TEST	UI0	UI1	U00	U01	WAIT

Public symbols

RTEST

User symbols

ACK 0006 A	AOCMD 008B A	ASUB 001A A	BELL 0007 A	BOCMD 0099 A	BS 0008 A	BYTE 0001 A
CAN 0018 A	CLRLC 004B A	CLRS 0004 E	CLRSC 004A A	COCMD 0092 A	CR 000D A	CTRLMA 001F A
CTRLQ 0011 A	CTRLS 0013 A	CTRLZ 001A A	DATA5 0000 A	DATA6 0004 A	DATA7 0008 A	DATA8 000C A
DB110 0369 A	DB1200 0050 A	DB1800 0035 A	DB2400 0028 A	DB300 0140 A	DB3600 001B A	DB4800 0014 A
DB600 00A0 A	DB7200 000D A	DB9600 000A A	DC1 0011 A	DC2 0012 A	DC3 0013 A	DC4 0014 A
DEL 007F A	DELAY 0002 E	DIV1 0001 A	DIV16 0002 A	DIV64 0003 A	DLE 0010 A	DNCMD 0042 A
DSR 0080 A	DSREM 019B C	DSRERR 0195 C	DTR 0002 A	EM 0019 A	ENQ 0005 A	EOT 0004 A
EPDATA 0020 A	EPNUM 0023 A	ERESET 0010 A	ERMASK 0038 A	ESC 001B A	ETB 0017 A	ETC1 0038 A
ETC2 00CB A	ETX 0003 A	EVENPA 0030 A	FALSE 0000 A	FF 000C A	FRMERR 0020 A	FS 001C A
GOTOC 0059 A	GS 001D A	HOMEC 0048 A	HT 0009 A	HUNTM 0080 A	INVSTO 0000 A	IOBASE 0000 A
IRESET 0040 A	K 0400 A	LF 000A A	LFCMD 0044 A	LSBYTE 0021 A	MAXX 004F A	MAXY 0017 A
MINX 0000 A	MINY 0000 A	MSBYTE 0022 A	MSCARD 0020 A	NAK 0015 A	NAMEA 0003 A	NOPARI 0000 A
NUL 0000 A	ODDPAR 0010 A	OFFSET 0020 A	OKMSG 02D3 C	ONEAND 0080 A	ONESTO 0040 A	QUINIT 01E8 C
OUMODE 004E A	OUTMSG 0007 E	OVRERR 0010 A	PARERR 0008 A	PORTA 000C A	PORTB 000D A	PORTC 000E A
PPIBAS 000C A	PPICON 000F A	QUOTE 0027 A	REVLf 0049 A	RS 001E A	RST05 000E C	RST10 0039 C
RST55A 002C A	RST65A 0034 A	RST75A 003C A	RTCMD 0043 A	RTERRE 01AF C	RTEST 0000 C	RTM01 021B C
RTM10 024F C	RTM15 0283 C	RTM20 02AB C	RTS 0020 A	RXENAB 0004 A	RXRDRY 0002 A	SBREAK 0008 A
SI 000F A	SO 000E A	SOH 0001 A	SPACE 0020 A	STX 0002 A	SYN 0016 A	SYNCR0 0000 A
SYNDET 0040 A	TODATA 0008 A	TIDATA 0009 A	T2DATA 000A A	TAB 0009 A	TBASE 0008 A	TCONT 000B A
TIM01 00B3 C	TIM01M 00B9 C	TIM10 00DB C	TIM10M 00E1 C	TIMER0 0000 A	TIMER1 0040 A	TIMER2 0080 A
TMODE 0036 A	TMODE0 0036 A	TMODE1 0076 A	TMODE2 00B6 A	TRUE 00FF A	TWOSTO 00C0 A	TXEMP 0004 A
TXENAB 0001 A	TXRDRY 0001 A	U0BASE 0002 A	UOCONT 0003 A	UODATA 0002 A	U0STAT 0003 A	UOTEST 0003 E
U1BASE 0004 A	U1CONT 0005 A	UIDATA 0004 A	U1STAT 0005 A	U1TEST 0005 E	UCMD 0015 A	UDUMMY 0002 A
UERR0 0173 C	UERR0M 0179 C	UERR1 0151 C	UERR1M 0157 C	UI0 0008 E	UIOWT 01FB C	UIOWTE 020A C
UIOWTL 01FE C	UI1 0001 E	UI1WT 020B C	UI1WTE 021A C	UI1WTL 020E C	UINIT 01B3 C	UMODE 007E A
U00 0000 E	U01 0006 E	UPCMD 0041 A	URESET 0040 A	US 001F A	VER01 0103 C	VER01M 0109 C
VER10 012A C	VER10M 0130 C	VT 000B A	WAIT 0009 E	WORD 0002 A		

```

LOC   OBJ       LINE      SOURCE STATEMENT
      1 $TITLE('Mass Storage Card test routines for the UCT SABus diagnostics')
      2 $PAGELENGTH(75)
      3 ;
      4 ;*****
      5 ;
      6 ;           D:      Mass storage card test      (file=DIAGMCS.A85)
      7 ;
      8 ;*****
      9 ;
     10           NAME      MSC
     11 ;
     12 $INCLUDE(ASCII.EQU)
= 13 ;
= 14 ;-----
= 15 ;
001F  = 16 CTRLMASK EQU      00011111B      ; Control mask.
= 17 ;
= 18 ;           ASCII characters:
= 19 ;
0000  = 20 NUL      EQU      000H           ; ASCII null character
0001  = 21 SOH      EQU      001H           ; ASCII Start of header character
0002  = 22 STX      EQU      002H           ; ASCII Start of transmission character
0003  = 23 ETX      EQU      003H           ; ASCII End of transmission character
0004  = 24 EOT      EQU      004H           ; ASCII End of text marker.
0005  = 25 ENQ      EQU      005H           ; ASCII Enquire character
0006  = 26 ACK      EQU      006H           ; ASCII Ack character
0007  = 27 BELL     EQU      007H           ; ASCII Bell character.
0008  = 28 BS       EQU      008H           ; ASCII Back space.
0009  = 29 HT       EQU      009H           ; ASCII Horiz tab character.
0009  = 30 TAB      EQU      009H           ; ASCII Horiz tab character.
000A  = 31 LF       EQU      00AH           ; ASCII Line feed character.
000B  = 32 VT       EQU      00BH           ; ASCII Vertical tab character
000C  = 33 FF       EQU      00CH           ; ASCII Form Feed character.
000D  = 34 CR       EQU      00DH           ; ASCII Carriage return character.
000E  = 35 SO       EQU      00EH           ; ASCII Shift out character
000F  = 36 SI       EQU      00FH           ; ASCII Shift in character
= 37 ;
0010  = 38 DLE      EQU      010H           ; ASCII DLE character.
0011  = 39 DC1      EQU      011H           ; ASCII Device control #1 (Xon)
0012  = 40 DC2      EQU      012H           ; ASCII Device control #2
0013  = 41 DC3      EQU      013H           ; ASCII Device control #3 (Xoff)
0014  = 42 DC4      EQU      014H           ; ASCII Device control #4
0015  = 43 NAK      EQU      015H           ; ASCII Negative ACK character
0016  = 44 SYN      EQU      016H           ; ASCII Sync character
0017  = 45 ETB      EQU      017H           ; ASCII End of transmission buffer.
0018  = 46 CAN      EQU      018H           ; ASCII Cancel.
0019  = 47 EM       EQU      019H           ; ASCII
001A  = 48 ASUB     EQU      01AH           ; ASCII Submit character.
001B  = 49 ESC      EQU      01BH           ; ASCII Escape character.
001C  = 50 FS       EQU      01CH           ; ASCII Field separator.
001D  = 51 GS       EQU      01DH           ; ASCII Group separator
001E  = 52 RS       EQU      01EH           ; ASCII Record separator
001F  = 53 US       EQU      01FH           ; ASCII Unit separator.
= 54 ;
= 55 ;           Control characters:
= 56 ;
0013  = 57 CTRLS    EQU      CTRLMASK AND 'S'      ; Control 'S'
0011  = 58 CTRLQ    EQU      CTRLMASK AND 'Q'      ; Control 'Q'
001A  = 59 CTRLZ    EQU      CTRLMASK AND 'Z'      ; Control 'Z'
= 60 ;
= 61 ;           Some more ASCII characters:
= 62 ;
0020  = 63 SPACE    EQU      020H           ; ASCII Space.
0027  = 64 QUOTE     EQU      027H           ; Single quote (')
007F  = 65 DEL      EQU      07FH           ; ASCII Delete character.
= 66 ;
= 67 $INCLUDE(VT52.EQU)
= 68 ;
= 69 ;-----
= 70 ;

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:23:45 20-APR-88

MSC

PAGE 2

Mass Storage Card test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		= 71 ;	VT52 ESC characters
		= 72 ;	
0048		= 73 HOMEC EQU 'H'	; Home cursor command
004A		= 74 CLRSC EQU 'J'	; Clear screen command
0059		= 75 GOTOC EQU 'Y'	; Direct cursor position command
004B		= 76 CLRLC EQU 'K'	; Clear line command
0049		= 77 REVLF EQU 'I'	; Reverse line feed
0020		= 78 OFFSET EQU 20H	; Direct cursor position offset
		= 79 ;	
0041		= 80 UPCMD EQU 'A'	; Up cursor command
0042		= 81 DNCMD EQU 'B'	; Down cursor command
0044		= 82 LFCMD EQU 'D'	; Left cursor command
0043		= 83 RTCMD EQU 'C'	; Right cursor command
		= 84 ;	
0000		= 85 MINX EQU 0	; Min x value (Col)
004F		= 86 MAXX EQU 79	; Max x value (Col)
0000		= 87 MINY EQU 0	; Min y value (Row)
0017		= 88 MAXY EQU 23	; Max y value (Row)
		= 89 ;	
		90 \$INCLUDE(COMEQU.EQU)	
		= 91 ;	
		= 92 ;	-----
		= 93 ;	
		= 94 ;	Common equates:
		= 95 ;	
0000		= 96 FALSE EQU 0	; Logical FALSE.
00FF		= 97 TRUE EQU (NOT FALSE) AND OFFH	; Logical TRUE.
		= 98 ;	
		= 99 ;	Storage size Equates (In terms of bytes).
		= 100 ;	
0002		= 101 WORD EQU 2	; 1 Word = 2 Bytes
0001		= 102 BYTE EQU 1	; 1 Byte = 1 Byte.
0400		= 103 K EQU 1024	; 1kbytes = 1024 bytes
		= 104 ;	
		105 \$INCLUDE(HARD.EQU)	
		= 106 ;	
		= 107 ;	-----
		= 108 ;	
		= 109 ;	Hardware equates for the UCT SABus kits:
		= 110 ;	
		= 111 ;	8085 interrupt vectors:
		= 112 ;	
002C		= 113 RST55A EQU 2CH	; Restart 5.5 address
0034		= 114 RST65A EQU 34H	; Restart 6.5 address
003C		= 115 RST75A EQU 3CH	; Restart 7.5 address
		= 116 ;	
		= 117 ;	Mass storage equates.
		= 118 ;	
0020		= 119 MSCARD EQU 20H	; Mass storage card - base address
		= 120 ;	
0020		= 121 EPDATA EQU MSCARD+0	; MSC : Data register
0021		= 122 LSBYTE EQU MSCARD+1	; MSC : LOW byte of EPROM address
0022		= 123 MSBYTE EQU MSCARD+2	; MSC : HIGH byte of EPROM address
0023		= 124 EPNUM EQU MSCARD+3	; MSC : Eprom number register
		= 125 ;	
0038		= 126 ETC1 EQU 038H	; EPROM There Code, first byte.
00CB		= 127 ETC2 EQU 0CBH	; EPROM There Code, second byte.
		= 128 ;	
0003		= 129 NAMEA EQU 3	; File Name offset
		= 130 ;	
0000		= 131 IOBASE EQU 0	; I/O Card base address
		= 132 ;	
0002		= 133 UOBASE EQU IOBASE+02H	; USART0 base address (8251)
0004		= 134 UIBASE EQU IOBASE+04H	; USART1 base address (8251)
0008		= 135 TBASE EQU IOBASE+08H	; Timer Base address (8253)
000C		= 136 PPIBASE EQU IOBASE+0CH	; PPI base address (8255)
		= 137 ;	
		= 138 ;	----- U0 registers. -----
		= 139 ;	
0002		= 140 UODATA EQU UOBASE+0	; USART 0 : Data register

Mass Storage Card test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
0003		= 141	U0STAT EQU UOBASE+1 ; USART 0 : Status register
0003		= 142	U0CONT EQU U0STAT ; USART 0 : Control register
		= 143 ;	
		= 144 ;	----- U1 registers. -----
		= 145 ;	
0004		= 146	U1DATA EQU U1BASE+0 ; USART 1 : Data register
0005		= 147	U1STAT EQU U1BASE+1 ; USART 1 : Status register
0005		= 148	U1CONT EQU U1STAT ; USART 1 : Control register
		= 149 ;	
		= 150 ;	----- Timer registers. -----
		= 151 ;	
0008		= 152	T0DATA EQU TBASE+0 ; TIMER : Counter 0 data register
0009		= 153	T1DATA EQU TBASE+1 ; TIMER : Counter 1 data register
000A		= 154	T2DATA EQU TBASE+2 ; TIMER : Counter 2 data register
000B		= 155	TCONT EQU TBASE+3 ; TIMER : Control register
		= 156 ;	
		= 157 ;	----- PPI registers -----
		= 158 ;	
000C		= 159	PORTA EQU PPIBASE+0 ; PPI : PORT A
000D		= 160	PORTB EQU PPIBASE+1 ; PPI : PORT B
000E		= 161	PORTC EQU PPIBASE+2 ; PPI : PORT C
000F		= 162	PPICONT EQU PPIBASE+3 ; PPI : Control/Status register
		= 163 ;	
		= 164 ;	-----
		= 165 ;	
		= 166 ;	I/O equates, timer/USART control/Mode words.
		= 167 ;	
0002		= 168	UDUMMY EQU 02H ; USART dummy mode/control word
0040		= 169	URESET EQU 40H ; USART reset control word
		= 170 ;	
0000		= 171	INVESTOPBIT EQU 00B SHL 6 ; Invalid stop bit spec
0040		= 172	ONESTOPBIT EQU 01B SHL 6 ; One stop bit
0080		= 173	ONEANDAHALFSTOPBIT EQU 10B SHL 6 ; One and a half stop bits
00C0		= 174	TWOSTOPBIT EQU 11B SHL 6 ; Two stop bits
		= 175 ;	
0000		= 176	NOPARITY EQU 00B SHL 4 ; No parity
0010		= 177	ODDPARITY EQU 01B SHL 4 ; Odd parity
0030		= 178	EVENPARITY EQU 11B SHL 4 ; Even parity
		= 179 ;	
0000		= 180	DATA5 EQU 00B SHL 2 ; 5 data bits
0004		= 181	DATA6 EQU 01B SHL 2 ; 6 data bits
0008		= 182	DATA7 EQU 10B SHL 2 ; 7 data bits
000C		= 183	DATA8 EQU 11B SHL 2 ; 8 data bits
		= 184 ;	
0000		= 185	SYNCHMODE EQU 00B SHL 0 ; Synchronous mode
0001		= 186	DIV1 EQU 01B SHL 0 ; Clock x1
0002		= 187	DIV16 EQU 10B SHL 0 ; Clock x16
0003		= 188	DIV64 EQU 11B SHL 0 ; Clock x64
		= 189 ;	
		= 190 ;	USART mode word.
		= 191 ;	
007E		= 192	UMODE EQU ONESTOPBIT + EVENPARITY + DATA8 + DIV16
004E		= 193	OMODE EQU ONESTOPBIT + NOPARITY + DATA8 + DIV16
		= 194 ;	
		= 195 ;	USART command register format
		= 196 ;	
0001		= 197	TXENAB EQU 00000001B ; Transmitter enable bit (Command)
0002		= 198	DTR EQU 00000010B ; Position of the DTR bit (Command)
0004		= 199	RXENAB EQU 00000100B ; Receiver enable bit (Command)
0008		= 200	SBREAK EQU 00001000B ; Send break (Command)
0010		= 201	ERESET EQU 00010000B ; Error reset bit (Command)
0020		= 202	RTS EQU 00100000B ; Request to send bit (Command)
0040		= 203	IRESET EQU 01000000B ; Internal reset bit (Command)
0080		= 204	HUNTM EQU 10000000B ; Enter hunt mode (Command)
		= 205 ;	
		= 206 ;	USART command word:
		= 207 ;	
0015		= 208	UCMD EQU TXENAB + RXENAB + ERESET
		= 209 ;	
		= 210 ;	USART Status register format:

LOC	OBJ	LINE	SOURCE STATEMENT
		= 211 ;	
0001		= 212 TXRDY EQU 00000001B	; Transmitter ready (Status)
0002		= 213 RXRDY EQU 00000010B	; Receiver ready (Status)
0004		= 214 TXEMP EQU 00000100B	; Transmitter empty (Status)
0008		= 215 PARERR EQU 00001000B	; Parity error flag (Status)
0010		= 216 OVRERR EQU 00010000B	; Overrun error flag (Status)
0020		= 217 FRMERR EQU 00100000B	; Framing error flag (Status)
0040		= 218 SYNDET EQU 01000000B	; Sync detect (Status)
0080		= 219 DSR EQU 10000000B	; DSR bit (Status)
		= 220 ;	
0038		= 221 ERMASK EQU 00111000B	; Combination error mask
		= 222 ;	
		= 223 ;	Timer commands and division factor.
		= 224 ;	
0000		= 225 TIMERO EQU 0 SHL 6	; Address timer 0
0040		= 226 TIMER1 EQU 1 SHL 6	; Address timer 1
0080		= 227 TIMER2 EQU 2 SHL 6	; Address timer 2
		= 228 ;	
0036		= 229 TMODE EQU 00110110B	; Basic mode word =
		= 230	; (Read and write LSB then MSB,
		= 231	; Mode 3,
		= 232	; Count in binary).
		= 233	;
0036		= 234 TMODE0 EQU TMODE+TIMERO	; Mode word for timer 0
0076		= 235 TMODE1 EQU TMODE+TIMER1	; Mode word for timer 1
00B6		= 236 TMODE2 EQU TMODE+TIMER2	; Mode word for timer 2
		= 237 ;	
		= 238 ;	Baud rate division factors for the USART, XTAL = 6.144 MHz
		= 239 ;	
0369		= 240 DB110 EQU 873	; Division ratio for 110 baud
0140		= 241 DB300 EQU 320	; Division ratio for 300 baud
00A0		= 242 DB600 EQU 160	; Division ratio for 600 baud
0050		= 243 DB1200 EQU 80	; Division ratio for 1200 baud
0035		= 244 DB1800 EQU 53	; Division ratio for 1800 baud
0028		= 245 DB2400 EQU 40	; Division ratio for 2400 aud
001B		= 246 DB3600 EQU 27	; Division ratio for 3600 baud
0014		= 247 DB4800 EQU 20	; Division ratio for 4800 baud
000D		= 248 DB7200 EQU 13	; Division ratio for 7200 baud
000A		= 249 DB9600 EQU 10	; Division ratio for 9600 baud
		= 250 ;	
		= 251 ;	PPI commands:
		= 252 ;	
008B		= 253 AOCMD EQU 10001011B	; A = Mode 0 output
		= 254	; B = Mode 0 input
		= 255	; C (upper/lower) = Mode 0 input
		= 256 ;	
0099		= 257 BOCMD EQU 10011001B	; A = Mode 0 input
		= 258	; B = Mode 0 output
		= 259	; C (upper/lower) = Mode 0 input
		= 260 ;	
0092		= 261 COCMD EQU 10010010B	; A = Mode 0 input
		= 262	; B = Mode 0 input
		= 263	; C (upper/lower) = Mode 0 output
		= 264 ;	
		= 265 ;	END HARD
		= 266 ;	
		= 267 ;	-----
		= 268 ;	
		= 269	PUBLIC STEST
		= 270 ;	
		= 271 ;	-----
		= 272 ;	
		= 273	EXTRN CLRS, ?ESCSP, SEARCH, INDIRECT, PTAB, CRLF, HOME, WAIT
		= 274	EXTRN OUTMSG, GETUC, ASCBIN, NMOUT, CO, PRVAL
		= 275 ;	
		= 276 ;	=====
		= 277 ;	
		= 278 ;	Main test routine for the MASS STORAGE CARD.
		= 279 ;	
		= 280	CSEG ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:23:45 20-APR-88

MSC

PAGE 5

Mass Storage Card test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		281	;
		282	STEST: ; Procedure MassStorageTest;
		283	;
0000	CD0000	E 284	CALL CLRS ; ClearScreen;
0003	212300	C 285	LXI H,MSCTM1 ;
0006	CD0000	E 286	CALL OUTMSG ; OutMessage(MassStorageCardMessage1);
		287	MSCT05: ; Repeat
		288	;
0009	CD0000	E 289	CALL GETUC ; Ch := GetUpperCase;
000C	CD0000	E 290	CALL 7ESCSF ;
000F	CA2200	C 291	JZ MSCT99 ; If Ch NOT IN [' ', ESC] then
		292	Begin
0012	21B003	C 293	LXI H,MSCTAB ;
0015	CD0000	E 294	CALL SEARCH ; Found := Search(MSCTable, Ch);
0018	C20900	C 295	JNZ MSCT05 ; If Found then
		296	;
001B	EB	E 297	XCHG ;
001C	CD0000	E 298	CALL INDIRECT ; CallIndirect(Addr_Of_Routine);
		299	;
001F	C30000	C 300	JMP STEST ; End;
		301	MSCT99: ; Until Ch in [' ', ESC];
		302	;
0022	C9	C 303	RET ; EndProc;
		304	;
		305	MSCTM1:
0023	0A0A0A0A	C 306	DB LF,LF,LF,LF,LF
	0A		
0028	09092020	C 307	DB TAB,TAB,' Mass storage card test options:'
	20202020		
	2020204D		
	61737320		
	73746F72		
	61676520		
	63617264		
	20746573		
	74206F70		
	74696F6E		
	733A		
0052	0D0A0A	C 308	DB CR,LF,LF
0055	09092020	C 309	DB TAB,TAB,' Dump an EPROM in hexadecimal (D)'
	20202020		
	20202044		
	756D7020		
	616E2045		
	50524F4D		
	20696E20		
	68657861		
	64656369		
	6D616C20		
	20284429		
0081	0D0A0A	C 310	DB CR,LF,LF
0084	09092020	C 311	DB TAB,TAB,' Display the file in the mess storage card (F)'
	20446973		
	706C6179		
	20746865		
	2066696C		
	6520696E		
	20746865		
	206D6573		
	73207374		
	6F726167		
	65206361		
	72642028		
	4629		
00B6	0D0A0A0A	C 312	DB CR,LF,LF,LF
00BA	09092020	C 313	DB TAB,TAB,' Enter option ?>', 0
	20202020		
	20202020		
	20202020		
	2020456E		

LOC	OBJ	LINE	SOURCE STATEMENT
	74657220		
	6F707469		
	6F6E203F		
	3E00		
		314 ;	
		315 ;=====	
		316 ;	
		317 ;	Storage file test (Prints file names).
		318 ;	
		319 SFTEST:	; Procedure StorageFileTest;
		320	;
00DC	CD0000	E 321	CALL CLRS ; ClearScreen;
00DF	212601	C 322	LXI H, MSCMSG ;
00E2	CD0000	E 323	CALL OUTMSG ; OutMessage(MassStorageCardMessage);
		324	;
00E5	0600	325	MVI B, 0 ; EepromNumber := 0;
00E7	0E00	326	MVI C, 0 ; EepromCount := 0;
		327	;
		328 MSC05:	; Repeat
		329	;
00E9	CD0000	E 330	CALL PTAB ; PrintTab;
00EC	CD0000	E 331	CALL PTAB ; PrintTab;
00EF	CD6203	C 332	CALL ?EPROM ;
00F2	C2FC00	C 333	JNZ MSC10 ; If EepromPresent(EepromNumber) then
		334	Begin
00F5	0C	335	INR C ; EepromCount := EepromCount + 1;
00F6	CD8003	C 336	CALL PRTNAME ; PrintName;
00F9	C3FF00	C 337	JMP MSC15 ; End
		338 MSC10:	; Else
		339	Begin
00FC	CD9E03	C 340	CALL EMPTY ; PrintEmpty;
		341 MSC15:	; End;
00FF	04	342	INR B ; EepromNumber := EepromNumber + 1;
		343	;
0100	78	344	MOV A, B ;
0101	FE10	345	CPI 16 ;
0103	C2E900	C 346	JNZ MSC05 ; Until EepromNumber = 16;
		347	;
0106	CD0000	E 348	CALL CRLF ; NewLine;
		349	;
0109	79	350	MOV A, C ;
010A	FE00	351	CPI 0 ;
010C	CA1C01	C 352	JZ MSC20 ; If EepromCount <> 0 then
		353	Begin
010F	79	354	MOV A, C ;
0110	CD0000	E 355	CALL PRVAL ; PrintValue(EepromCount);
0113	21AA01	C 356	LXI H, MSCM1 ;
0116	CD0000	E 357	CALL OUTMSG ; OutMessage('EPROMs have been found')
0119	C32201	C 358	JMP MSC25 ; End
		359 MSC20:	; Else
011C	21C201	C 360	LXI H, MSCM2 ;
011F	CD0000	E 361	CALL OUTMSG ; OutMessage(NotFoundMessage);
		362 MSC25:	;
0122	CD0000	E 363	CALL WAIT ; WaitForResponse;
0125	C9	364	RET ; EndProc;
		365 ;	
		366 MSCMSG:	
0126	0D0A09	367	DB CR, LF, TAB
0129	54686520	368	DB 'The following EPROM(S) can be read from the Mass storage card'
	666F6C6C		
	6F77696E		
	67204550		
	524F4D28		
	53292063		
	616E2062		
	65207265		
	61642066		
	726F6D20		
	74686520		
	4D617373		

Mass Storage Card test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
	2073746F		
	72616765		
	20636172		
	64		
0166	0D0A09	369	DB CR,LF,TAB
0169	2D2D2D2D	370	DB '-----'
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D		
01A6	0D0A0A	371	DB CR,LF,LF
01A9	00	372	DB 0
		373 ;	
01AA	20455052	374 MSCM1: DB	' EPROMs have been found',0
	4F4D7320		
	68617665		
	20626565		
	6E20666F		
	756E6400		
01C2	4D617373	375 MSCM2: DB	'Mass storage card appears to be empty or not present',0
	2073746F		
	72616765		
	20636172		
	64206170		
	70656172		
	7320746F		
	20626520		
	656D7074		
	79206F72		
	206E6F74		
	20707265		
	73656E74		
	00		
		376 ;	
		377 ;=====	
		378 ;	
		379 SDTEST:	; Procedure StorageDumpTest;
01F7	3E00	380 MVI A,0	;
01F9	321000 D	381 STA ASCEOT	; ASCIIBuffer[ASCII_EOT] := EOT;
		382	;
01FC	CD0000 E	383 CALL CLRS	; ClearScreen;
01FF	215302 C	384 LXI H,ENMSG	;
0202	CD0000 E	385 CALL OUTMSG	; OutMessage('Enter EPROM number (0..F)?>');
		386	;
		387 SD05:	; Repeat
0205	CD0000 E	388 CALL GETUC	;
0208	47	389 MOV B,A	; Ch := GetUpperCase;
0209	CD0000 E	390 CALL ASCBIN	; EpromNumber := ASCII_to_Binary(Ch);
		391	;
020C	D20502 C	392 JNC SD05	; Until Ch in ['0'..'9', 'A'..'F'];
		393	;
020F	78	394 MOV A,B	;
0210	CD0000 E	395 CALL CO	; ConsoleOut(Ch);
0213	CD0000 E	396 CALL ASCBIN	;
0216	D323	397 OUT EPNUM	; Out(EpromSelect, EpromNumber);
		398	;
0218	CD0000 E	399 CALL CLRS	; ClearScreen;
		400 ;	
		401 ; The EPROM has been selected, now data can be read using 'READB'	

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:23:45 20-APR-88 MSC PAGE 8
Mass Storage Card test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		402 ;	
021B	210000	403	LXI H,0 ; Ptr := 0;
		404 SD07:	; Repeat
021E	CDBD02 C	405	CALL DEBLK ; DisplayBlock(Ptr);
0221	E5	406	PUSH H ;
0222	217202 C	407	LXI H,SDMSG2 ;
0225	CD0000 E	408	CALL OUTMSG ; OutMessage(StorageDumpMessage2);
0228	E1	409	POP H ;
		410 SD10:	;
0229	CD0000 E	411	CALL GETUC ; Ch := GetUpperCase;
022C	CD0000 E	412	CALL ?ESCSP ;
022F	CA5202 C	413	JZ SD99 ;
		414	;
0232	FE4E	415	CPI 'N' ;
0234	C23E02 C	416	JNZ SDNN ; If Ch = 'N' then
		417	; Begin
0237	110001	418	LXI D,100H ;
023A	19	419	DAD D ; Ptr := Ptr + 100H
023B	C31E02 C	420	JMP SD07 ; End
		421	;
		422 SDNN:	;
023E	FE50	423	CPI 'P' ;
0240	C24A02 C	424	JNZ SDNP ; ElseIf Ch = 'P' then
		425	; Begin
0243	1100FF	426	LXI D,-100H ;
0246	19	427	DAD D ; Ptr := Ptr - 100H;
0247	C31E02 C	428	JMP SD07 ; End
		429	;
		430 SDNP:	;
024A	FE52	431	CPI 'R' ; ElseIf Ch = 'R' then
024C	CAF701 C	432	JZ SDTEST ; RestartRoutine;
		433	;
024F	C32902 C	434	JMP SD10 ; Until Ch in [ESC, ' '];
		435	;
		436 SD99:	;
0252	C9	437	RET ; EndProc
		438 ;	
0253	0909456E	439	ENMSG: DB TAB,TAB,'Enter EPROM number (0..F) ?>', 0
	74657220		
	4550524F		
	4D206E75		
	6D626572		
	2028302E		
	2E462920		
	3F3E00		
		440 ;	
0272	OD0A0A	441	SDMSG2: DB CR,LF,LF
0275	4553432F	442	DB 'ESC/SPACE to exit, P for previous block, N for next block,'
	53504143		
	4520746F		
	20657869		
	742C2050		
	20666F72		
	20707265		
	76696F75		
	7320626C		
	6F636B2C		
	204E2066		
	6F72206E		
	65787420		
	626C6F63		
	6B2C		
02AF	20522074	443	DB ' R to restart', 0
	6F207265		
	73746172		
	7400		
		444 ;	
		445 ;	
		446 ;	
		447 ;	HL : Ptr

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:23:45 20-APR-88

MSC

PAGE 9

Mass Storage Card test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		448 ;	
		449 DEBLK:	; Procedure DisplayBlock(Ptr:Word);
		450	
02BD	CD0000	E 451	CALL HOME ; HomeCursor;
02C0	E5	452	PUSH H ;
02C1	210A03	C 453	LXI H,HEXH ;
02C4	CD0000	E 454	CALL OUTMSG ; OutMessage(HexDumpHeader);
02C7	E1	455	POP H ;
02C8	E5	456	PUSH H ;
02C9	C5	457	PUSH B ;
02CA	D5	458	PUSH D ;
02CB	0610	459	MVI B,16 ; For I := 1 to 16 do
		460 DEB05:	Begin
02CD	7C	461	MOV A,H ;
02CE	CD0000	E 462	CALL NMOUT ; NumberOutput(HIGH Ptr);
02D1	7D	463	MOV A,L ;
02D2	CD0000	E 464	CALL NMOUT ; NumberOutput(LOW Ptr);
02D5	3E20	465	MVI A,SPACE ;
02D7	CD0000	E 466	CALL CO ; ConsoleOut(' ');
02DA	CD0000	E 467	CALL CO ; ConsoleOut(' ');
02DD	110000	D 468	LXI D,ASCBUF ; ASCIIBufferPtr := ASCIIBuffer;
02E0	0E10	469	MVI C,16 ; For J := 1 to 16 do
		470 DEB10:	Begin
		471	
02E2	CD7603	C 472	CALL READB ; B := ReadByte(Ptr);
02E5	CD4F03	C 473	CALL BUFWRIT ; BufferWrite(B);
02E8	CD0000	E 474	CALL NMOUT ; NumberOutput(B);
02EB	3E20	475	MVI A,SPACE ;
02ED	CD0000	E 476	CALL CO ; ConsoleOut(' ');
02F0	0D	477	DCR C ;
02F1	C2E202	C 478	JNZ DEB10 ; End;
		479	
02F4	E5	480	PUSH H ;
02F5	210000	D 481	LXI H,ASCBUF ;
02F8	CD0000	E 482	CALL OUTMSG ; OutMessage(ASCIIBuffer);
02FB	E1	483	POP H ;
		484	
02FC	CD0000	E 485	CALL CRLF ; NewLine;
02FF	05	486	DCR B ;
0300	C2CD02	C 487	JNZ DEB05 ; End;
		488	
0303	CD0000	E 489	CALL CRLF ; NewLine;
		490	
0306	D1	491	POP D ;
0307	C1	492	POP B ;
0308	E1	493	POP H ;
0309	C9	494	RET ; EndProc;
		495 ;	
		496 ;	AAAA BB BB BB BB BB BB BB BB BB BB BB BB BB BB ...SAD...
		497 ;	
030A	20414452	498 HEXH: DB ' ADR 0 1 2 3 4 5 6 7 8 9 A B C D E F ASCII'	
	20202030		
	20203120		
	20322020		
	33202034		
	20203520		
	20362020		
	37202038		
	20203920		
	20412020		
	42202043		
	20204420		
	20452020		
	46202020		
	20202020		
	41534349		
	49		
034B	0D0A0A00	499 DB CR,LF,LF,0	
		500 ;	
		501 ;	

LOC	OBJ	LINE	SOURCE STATEMENT
		502 ;	
		503 ;	
		504 ;	
		505 ;	DE : ASCIIBufferPtr
		506 ;	A : ByteToWrite
		507 ;	
		508 BUFWRT:	; Procedure BufferWrite(ASCIIBufferPtr:Word;
		509	ByteToWrite:Byte);
034F	F5	510	PUSH PSW ;
0350	E67F	511	ANI 01111111B ; B := ByteToWrite AND 01111111B
		512	;
0352	FE7F	513	CPI DEL ;
0354	CA5C03	514	JZ PUTDOT ; If (B = DEL) OR
		515	;
0357	FE20	516	CPI SPACE ;
0359	F25E03	517	JP BOK ; (B < Space) then
		518 PUTDOT:	;
035C	3E2E	519	MVI A, '.' ; B := '.';
		520 BOK:	;
035E	12	521	STAX D ; ASCIIBuffer[ASCIIBufferPtr] := B;
035F	13	522	INX D ; ASCIIBufferPtr := ASCIIBufferPtr + 1;
0360	F1	523	POP PSW ;
0361	C9	524	RET ; EndProc;
		525 ;	
		526 ;	
		527 ;	
		528 ;	B : EpromNumber
		529 ;	
		530 ?EPROM:	; Function ?Eprom(EpromNumber:Byte):Boolean;
0362	78	531	MOV A,B ;
0363	D323	532	OUT EPNUM ; Out(EpromSelect, EpromNumber);
0365	210000	533	LXI H,0 ;
0368	CD7603	534	CALL READB ; B := ReadByte(0);
036B	FE38	535	CPI ETC1 ;
036D	C27503	536	JNZ ?EPRET ; If B = ETC1 then
		537	Begin
0370	CD7603	538	CALL READB ; B := ReadByte(1);
0373	FECB	539	CPI ETC2 ; If B = ETC2 then
		540	Return(TRUE)
		541	Else
		542	Return(FALSE)
		543	End
		544 ?EPRET:	Else
		545	return(FALSE)
0375	C9	546	RET ; EndProc;
		547 ;	
		548 ;	
		549 ;	
		550 ;	HL : Address
		551 ;	
		552 READB:	; Function ReadByte (Var Address:Word):Byte;
0376	7C	553	MOV A,H ;
0377	D322	554	OUT MSBYTE ; Out(AddressMostSigByte, HIGH Address);
0379	7D	555	MOV A,L ;
037A	D321	556	OUT LSBYTE ; Out(AddressLeastSigByte, LOW Address);
		557	;
037C	DB20	558	IN EPDATA ; ReadByte := In(EpromData);
037E	23	559	INX H ; Address := Address + 1;
037F	C9	560	RET ; EndProc;
		561 ;	
		562 ;	
		563 ;	
		564 ;	B : EpromNumber
		565 ;	
		566 PRTNAME:	; Procedure PrintName(EpromNumber:Byte);
0380	C5	567	PUSH B ;
0381	78	568	MOV A,B ;
0382	CD0000	569	CALL PRVAL ; PrintValue(EpromNumber);
0385	3E20	570	MVI A,SPACE ;
0387	CD0000	571	CALL CO ; ConsoleOutput(' ');

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:23:45 20-APR-88

MSC

PAGE 11

Mass Storage Card test routines for the UCT SABus diagnostics

```

LOC   OBJ       LINE   SOURCE STATEMENT
038A  210300     572     LXI   H,NAMEA      ;   Ptr := NameAddress;
038D  060A       573     MVI   B,10         ;   For I := 1 to 10 do
                                574 PRTN05:      ;       Begin
038F  CD7603    C  575     CALL  READB      ;       B := ReadByte(Ptr);
0392  CD0000    E  576     CALL  CO         ;       ConsoleOutput(B);
0395  05        577     DCR    B           ;
0396  C28F03    C  578     JNZ   PRTN05      ;       End;
                                579           ;
0399  C1        580     POP    B           ;
039A  CD0000    E  581     CALL  CRLF       ;       NewLine;
                                582           ;
039D  C9        583     RET                ; EndProc;
                                584 ;
                                585 ;-----
                                586 ;
                                587 ;       B : EpromNumber
                                588 ;
                                589 EMPTY:      ; Procedure Entry(EpromNumber:Byte);
039E  78        590     MOV    A,B         ;
039F  CD0000    E  591     CALL  PRVAL      ;       PrintValue(EpromNumber);
03A2  3E20      592     MVI   A,SPACE      ;
03A4  CD0000    E  593     CALL  CO         ;       ConsoleOutput(' ');
03A7  3E2D      594     MVI   A,'-'       ;
03A9  CD0000    E  595     CALL  CO         ;       ConsoleOutput('-');
03AC  CD0000    E  596     CALL  CRLF       ;       NewLine;
                                597           ;
03AF  C9        598     RET                ; EndProc;
                                599 ;
                                600 ;-----
                                601 ;
                                602 MSCTAB:      ; Mass storage card test command table
                                603           ;
03B0  46        604     DB     'F'         ;   Command letter for the FILES command
03B1  DC00      C  605     DW     SFTEST      ;   Address of the file routine
                                606           ;
03B3  44        607     DB     'D'         ;   Command letter for the DUMP command
03B4  F701      C  608     DW     SDTEST      ;   Address of the dump routine
                                609           ;
03B6  00        610     DB     0           ; End of table marker
                                611 ;
                                612 ;#####
                                613 ;
                                614 DSEG
                                615 ;
0000  616 ASCBUF: DS     16           ; 16 byte ASCII buffer
0010  617 ASCBOT: DS     BYTE        ; End of buffer marker (Initialised to zero
                                618           ; by the program).
                                619 ;
                                620 ;       End of <DIAGMSC.A85> module
                                621 ;
                                622 END

```

0 error(s) detected

External symbols

```

?ESCSP  ASCBIN  CLRS   CO      CRLF  GETUC  HOME   INDIRE  NMOUT  OUTMSG  PRVAL  PTAB   SEARCH
WAIT

```

Public symbols

STEST

User symbols

```

?EPRET 0375 C   ?EPROM 0362 C   ?ESCSP 0006 E   ACK   0006 A   AOCMD 008B A   ASCBIN 000C E   ASCBUF 0000 D
ASCEOT 0010 D   ASUB   001A A   BELL   0007 A   BOCMD 0099 A   BOK    035E C   BS     0008 A   BUFVRT 034F C

```

DIAG LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:23:45 20-APR-88

MSC

PAGE 12

Mass Storage Card test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
BYTE	0001 A	CAN 0018 A	CLRLC 004B A
CR	000D A	CRLF 0000 E	CTRLMA 001F A
DATA6	0004 A	DATA7 0008 A	DATA8 000C A
DB300	0140 A	DB3600 001B A	DB4800 0014 A
DC2	0012 A	DC3 0013 A	DC4 0014 A
DIV1	0001 A	DIV16 0002 A	DIV64 0003 A
EM	0019 A	EMPTY 039E C	ENMSG 0253 C
ERESET	0010 A	ERNASK 0038 A	ESC 001B A
EVENPA	0030 A	FALSE 0000 A	FF 000C A
GS	001D A	HEXH 030A C	HOME 000D E
INVSTO	0000 A	IOBASE 0000 A	IRESET 0040 A
MAXX	004F A	MAXY 0017 A	MINX 0000 A
MSC15	00FF C	MSC20 011C C	MSC25 0122 C
MSCT05	0009 C	MSCT99 0022 C	MSCTAB 03B0 C
NOPARI	0000 A	NUL 0000 A	ODDPAR 0010 A
OUTMSG	0007 E	OVRERR 0010 A	PARERR 0008 A
PPICON	000F A	PRTN05 038F C	PRTNAM 0380 C
READB	0376 C	REVLf 0049 A	RS 001E A
RTS	0020 A	RKENAB 0004 A	RXRdy 0002 A
SD99	0252 C	SDMSG2 0272 C	SDNN 023E C
SI	000F A	SO 000E A	SOH 0001 A
SYNCMO	0000 A	SYNDET 0040 A	T0DATA 0008 A
TCONT	000B A	TINERO 0000 A	TIMER1 0040 A
TMODE2	00B6 A	TRUE 00FF A	TWOSTO 00C0 A
UOCONT	0003 A	UODATA 0002 A	UOSTAT 0003 A
UCMD	0015 A	UDUMMY 0002 A	UMODE 007E A
WAIT	0009 E	WORD 0002 A	
			CLRS 0002 E
			CLRSC 004A A
			CO 0008 E
			COCMD 0092 A
			CTRLQ 0011 A
			CTRLS 0013 A
			CTRLZ 001A A
			DATA5 0000 A
			DB1200 0050 A
			DB1800 0035 A
			DB2400 0028 A
			DB600 00A0 A
			DB7200 000D A
			DB9600 000A A
			DC1 0011 A
			DEB05 02CD C
			DEB10 02E2 C
			DEBLK 02BD C
			DEL 007F A
			DLE 0010 A
			DNCMD 0042 A
			DSR 0080 A
			DTR 0002 A
			EOT 0004 A
			EPDATA 0020 A
			EPNUM 0023 A
			ETC1 0038 A
			ETC2 00CB A
			ETX 0003 A
			FS 001C A
			GETUC 000A E
			GOTOC 0059 A
			HT 0009 A
			HUNTM 0080 A
			INDIRE 0003 E
			K 0400 A
			LF 000A A
			LFCMD 0044 A
			LSBYTE 0021 A
			MSBYTE 0022 A
			MSC05 00E9 C
			MSC10 00FC C
			MSCM1 01AA C
			MSCM2 01C2 C
			MSCMSG 0126 C
			NAK 0015 A
			NAMEA 0003 A
			NNOUT 0004 E
			ONEAND 0080 A
			ONESTO 0040 A
			OUMODE 004E A
			PORTA 000C A
			PORTB 000D A
			PORTC 000E A
			PPIBAS 000C A
			PRVAL 0001 E
			PTAB 000B E
			PUTDOT 035C C
			QUOTE 0027 A
			RST55A 002C A
			RST65A 0034 A
			RST75A 003C A
			RTCMD 0043 A
			SD05 0205 C
			SD07 021E C
			SD10 0229 C
			SDNP 024A C
			SDTEST 01F7 C
			SEARCH 0005 E
			SFTEST 00DC C
			SPACE 0020 A
			STEST 0000 C
			STX 0002 A
			SYN 0016 A
			T2DATA 000A A
			TAB 0009 A
			TBASE 0008 A
			TMODE 0036 A
			TMODE0 0036 A
			TMODE1 0076 A
			TXEMP 0004 A
			TXENAB 0001 A
			TXRDY 0001 A
			U0BASE 0004 A
			U1CONT 0005 A
			U1DATA 0004 A
			U1STAT 0005 A
			UPCMD 0041 A
			URESET 0040 A
			US 001F A
			VT 000B A


```

LOC  OBJ      LINE      SOURCE STATEMENT
                                1 $TITLE('VDU test routines for the UCT SABus diagnostics')
                                2 $PAGELENGTH(75)
                                3 ;
                                4 ;*****
                                5 ;
                                6 ;      C:      VDU test.      (file=DIAGVDU.A85)
                                7 ;
                                8 ;*****
                                9 ;
                                10      NAME      VDU
                                11 ;
                                12 $INCLUDE(COMEQU.A85)
= 13 ;      NAME      COMEQU
= 14 ;-----
= 15 ;
= 16 ;      PUBLIC  FALSE, TRUE
= 17 ;      PUBLIC  BYTE, WORD, K
= 18 ;
= 19 ;-----
= 20 ;
= 21 ;      Logical equates.
= 22 ;
0000  = 23 FALSE  EQU      0      ; Logical FALSE.
00FF  = 24 TRUE   EQU      (NOT FALSE) AND OFFH ; Logical TRUE.
= 25 ;
= 26 ;      Storage size Equates ( In terms of bytes ).
= 27 ;
0002  = 28 WORD   EQU      2      ; 1 Word = 2 Bytes
0001  = 29 BYTE   EQU      1      ; 1 Byte = 1 Byte.
0400  = 30 K      EQU      1024    ; 1kbytes = 1024 bytes
= 31 ;
= 32 ;      END
                                33 $INCLUDE(ASCII.A85)
= 34 ;      NAME      ASCII
= 35 ;-----
= 36 ;
= 37 ;      PUBLIC  NULL, EOT, BELL, BS, TAB, LF, CR, FF
= 38 ;      PUBLIC  CTRLS, CTRLQ, CTRLZ, ESC, SPACE, QUOTE, DEL
= 39 ;
= 40 ;-----
= 41 ;      ASCII constants.
= 42 ;
0000  = 43 NULL   EQU      000H    ; Null
0004  = 44 EOT    EQU      004H    ; End of text marker.
0007  = 45 BELL   EQU      007H    ; Bell.
0008  = 46 BS     EQU      008H    ; Back space.
0009  = 47 TAB    EQU      009H    ; Horiz Tab.
000A  = 48 LF     EQU      00AH    ; Line feed.
000D  = 49 CR     EQU      00DH    ; Carriage return.
000C  = 50 FF     EQU      00CH    ; Form Feed.
0013  = 51 CTRLS  EQU      013H    ; Control 'S'
0011  = 52 CTRLQ  EQU      011H    ; Control 'Q'
001A  = 53 CTRLZ  EQU      01AH    ; Control 'Z'
001B  = 54 ESC    EQU      01BH    ; ESC
0020  = 55 SPACE  EQU      020H    ; ASCII Space.
0027  = 56 QUOTE  EQU      027H    ; Single quote (')
007F  = 57 DEL    EQU      07FH    ; ASCII Delete character.
= 58 ;
= 59 ;      END
                                60 $INCLUDE(VT52.A85)
= 61 ;      NAME      VT52
= 62 ;
= 63 ;-----
= 64 ;
= 65 ;      PUBLIC  HOME, CLRSC, GOTOC, CLRLC, REVLF, OFFSET
= 66 ;      PUBLIC  UPCMD, DNCMD, LFCMD, RTCMD
= 67 ;      PUBLIC  MINX, MINY, MAXX, MAXY
= 68 ;
= 69 ;-----
= 70 ;

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:24:20 20-APR-88

VDU

PAGE 2

VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		= 71 ;	VT52 ESC characters
		= 72 ;	
0048		= 73 HOMEC EQU 'H'	; Home cursor command
004A		= 74 CLRSC EQU 'J'	; Clear screen command
0059		= 75 GOTOC EQU 'Y'	; Direct cursor position command
004B		= 76 CLRLC EQU 'K'	; Clear line command
0049		= 77 REVLF EQU 'I'	; Reverse line feed
0020		= 78 OFFSET EQU 20H	; Direct cursor position offset
		= 79 ;	
0041		= 80 UPCMD EQU 'A'	; Up cursor command
0042		= 81 DNCMD EQU 'B'	; Down cursor command
0044		= 82 LFCMD EQU 'D'	; Left cursor command
0043		= 83 RTCMD EQU 'C'	; Right cursor command
		= 84 ;	
0000		= 85 MINX EQU 0	; Min x value (Col)
004F		= 86 MAXX EQU 79	; Max x value (Col)
0000		= 87 MINY EQU 0	; Min y value (Row)
0017		= 88 MAXY EQU 23	; Max y value (Row)
		= 89 ;	
		= 90 ;	END
		91 ;	
		92 ;	-----
		93 ;	
		94	EXTRN CLRS, INDIRECT, ?ESCSP, SEARCH
		95	EXTRN OUTMSG, GETUC, CI, CTEST, CO, UO0, UO1
		96 ;	
		97 ;	-----
		98 ;	
		99	PUBLIC VTEST, VINIT
		100 ;	
		101 ;	-----
		102 ;	
		103 ;	Start of main VDU test routine (VTEST)
		104 ;	
		105	CSEG ;
		106 VTEST:	; Procedure VTEST;
		107	; Loop;
		108	; Clear_screen;
0000	CD0000	E 109	CALL CLRS ;
0003	21A702	C 110	LXI H,VTMSG1 ;
0006	CD0000	E 111	CALL OUTMSG ; Display_message(Video_test_message1);
0009	CD0C02	C 112	CALL PVTOUT ; Display_current_output_stream;
000C	211604	C 113	LXI H,VTMSG2 ;
000F	CD0000	E 114	CALL OUTMSG ; Display_message(Video_test_message2);
		115	; Repeat
		116 VT05:	; A := Fetch_upper_case_character;
0012	CD0000	E 117	CALL GETUC ; If A in [esc, space] then
0015	CD0000	E 118	CALL ?ESCSP ; EXIT(procedure);
0018	CA2B00	C 119	JZ VTEXT ;
		120	; Until A in Video_test_table;
001B	217904	C 121	LXI H,VTTAB ;
001E	CD0000	E 122	CALL SEARCH ;
0021	C21200	C 123	JNZ VT05 ;
		124	; Video_test_routine(A);
0024	EB	125	XCHG ;
0025	CD0000	E 126	CALL INDIRECT ;
		127	; endloop;
0028	C30000	C 128	JMP VTEST ;
		129	; End VTEST;
		130 VTEXT:	; End VTEST;
002B	C9	131	RET ;
		132 ;	
		133 ;	=====
		134 ;	
		135 ;	Initialise output routines for VDU test.
		136 ;	
		137 VINIT:	; VDU_test_initilise;
002C	3E43	138	MVI A,'C' ;
002E	320000	D 139	STA VTOUTR ; VDU_Output := 'C'; {Console}
		140	;

VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		141 ;	
		142 ;	Initialise dynamic output routine to be
		143 ;	
		144 ; VTOUT: CALL CO	
		145 ; RET	
		146 ;	
0031	3ECD	147	MVI A,(CALL) ;
0033	320100	148	STA VTOUT ; VDU_Output_routine[0] := (CALL);
		149	;
0036	210000	150	LXI H,CO ;
0039	220200	151	SHLD VTOUT+1 ; VDU_Output_routine[1] := LOW CO;
		152	;
		153	MVI A,(RET) ; VDU_Output_routine[2] := HIGH CO;
003C	3EC9	154	STA VTOUT+3 ; VDU_Output_routine[3] := (RET);
003E	320400	155	;
		156	RET ; EndProc;
0041	C9	157 ;	
		158 ;*****	
		159 ;	
		160 ;	Video test routine #1 : SCROLL test.
		161 ;	
		162 SCROLL:	; Procedure ScrollTest;
		163	;
0042	CD6002	164	CALL VTCLRS ; VDU_Test_Clear_Screen;
0045	3E20	165	MVI A,SPACE ;
0047	320700	166	STA STARTC ; StartCharacter := Space;
		167	;
		168 SC02:	; Repeat
		169	;
004A	3A0700	170	LDA STARTC ; Ch := StartCharacter;
004D	0650	171	MVI B,80 ; For I := 1 to 80 do
		172 SC10:	; Begin
004F	CD0100	173	CALL VTOUT ; VDU_Out(Ch);
0052	CD7500	174	CALL NEXTC ; Ch := NextCharacter(Ch);
0055	05	175	DCR B ;
0056	C24F00	176	JNZ SC10 ; End;
		177	;
0059	3A0700	178	LDA STARTC ;
005C	CD7500	179	CALL NEXTC ;
005F	320700	180	STA STARTC ; StartCharacter := NextCharacter(
		181	StartCharacter);
		182	;
0062	CD5502	183	CALL VTCRLF ; VDU_NewLine;
		184	;
0065	CD0000	185	CALL CITEST ;
0068	CA4A00	186	JZ SC02 ; Until ConsoleInputTest AND
		187	;
006B	CD0000	188	CALL CI ;
006E	CD0000	189	CALL ?ESCSP ; ConsoleInput IN [ESC, ' '];
0071	C24A00	190	JNZ SC02 ;
		191 SC20:	;
0074	C9	192	RET ; EndProc;
		193 ;	
		194 ;-----	
		195 ;	
		196 NEXTC:	; Function NEXTC (Present_character[A] : char):
		197	char[A];
		198	;
		199	Begin
0075	3C	200	INR A ; NEXTC := Present_character + 1;
0076	FE7F	201	CPI DEL ; If NEXTC = Chr(del) then
0078	C27D00	202	JNZ NEXTEX ;
		203	;
007B	3E20	204	MVI A,SPACE ; NEXTC := Chr(Space);
		205	NEXTEX: ;
007D	C9	206	RET ; End;
		207 ;*****	
		208 ;	
		209 DCURP:	; Procedure CursorPositioning;
		210	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:24:20 20-APR-88

VDU

PAGE 4

VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
007E	CD6002	C 211	CALL VTCLRS ; VDU_ClearScreen;
0081	215C04	C 212	LXI H,DCMSG ;
0084	CD7502	C 213	CALL VTOUTMSG ; OutMessage(VDU_Test_Message);
0087	3E2A	214	MVI A,'*' ;
0089	67	215	MOV H,A ; TmpCh := '*';
008A	320600	D 216	STA VTCHAR ; VDU_test_Char := '*';
		217 DC02:	Loop
008D	3E00	218	MVI A,0 ;
008F	320500	D 219	STA YPOS ; Ypos := 0;
		220	;
		221 DC05:	Repeat
0092	CD0000	E 222	CALL CITEST ;
0095	CAB100	C 223	JZ DC10 ;
		224	;
0098	CD0000	E 225	CALL CI ;
009B	CD0000	E 226	CALL ?ESCSP ;
009E	CAD300	C 227	JZ DC30 ;
		228	;
00A1	FE7F	229	CPI DEL ;
00A3	CAB100	C 230	JZ DC10 ;
		231	;
00A6	E67F	232	ANI 01111111B ;
00A8	FE21	233	CPI SPACE+1 ;
00AA	FAB100	C 234	JM DC10 ;
		235	;
00AD	67	236	MOV H,A ;
00AE	320600	D 237	STA VTCHAR ;
		238 DC10:	End;
		239	;
00B1	3A0500	D 240	LDA YPOS ;
00B4	CDD400	C 241	CALL DRAW ; Draw(Ypos, TmpCh);
00B7	3A0500	D 242	LDA YPOS ;
00BA	3C	243	INR A ;
00BB	320500	D 244	STA YPOS ;
00BE	FE0B	245	CPI MAXY/2 ;
00C0	C29200	C 246	JNZ DC05 ;
		247	;
00C3	3A0600	D 248	LDA VTCHAR ;
00C6	BC	249	CMP H ;
00C7	3E20	250	MVI A,' ' ;
00C9	CACD00	C 251	JZ DC15 ;
00CC	7C	252	MOV A,H ;
		253 DC15:	;
00CD	320600	D 254	STA VTCHAR ;
00D0	C38D00	C 255	JMP DC02 ; EndLoop;
		256	;
		257 DC30:	;
00D3	C9	258	RET ; Return from <DCURP>
		259 ;	;
		260 ;	;
		261 ;	;
		262 DRAW:	;
		263	;
00D4	320E00	D 264	STA LMIN ; Lmin := Ypos;
00D7	320C00	D 265	STA CMIN ; Cmin := Ypos;
		266	;
00DA	320800	D 267	STA TOPCX ; TopCornerX := Ypos;
00DD	320900	D 268	STA TOPCY ; TopCornerY := Ypos;
00E0	47	269	MOV B,A ;
		270	;
00E1	3E4F	271	MVI A,MAXX ;
00E3	90	272	SUB B ;
00E4	320A00	D 273	STA BOTCX ; BottomCornerX := MaxX - Ypos;
00E7	320D00	D 274	STA CMAX ; Cmax := MaxX - Ypos;
		275	;
00EA	3E17	276	MVI A,MAXY ;
00EC	90	277	SUB B ;
00ED	320B00	D 278	STA BOTCY ; BottomCornerY := MaxY - Ypos;
00F0	320F00	D 279	STA LMAX ; Lmax := MaxY - Ypos;
		280	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:24:20 20-APR-88

VDU

PAGE 5

VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		281	DR05:
		282	
00F3	3A0C00	D 283	LDA CMIN
00F6	47	284	MOV B,A
00F7	3A0D00	D 285	LDA CMAX
00FA	B8	286	CMP B
00FB	FA7801	C 287	JM DR10
		288	
00FE	3A0E00	D 289	LDA LMIN
0101	47	290	MOV B,A
0102	3A0F00	D 291	LDA LMAX
0105	B8	292	CMP B
0106	FA7801	C 293	JM DR10
		294	
0109	3A0900	D 295	LDA TOPCY
010C	47	296	MOV B,A
010D	3A0C00	D 297	LDA CMIN
0110	CD3702	C 298	CALL PRTAT
		299	
0113	3A0B00	D 300	LDA BOTCY
0116	47	301	MOV B,A
0117	3A0D00	D 302	LDA CMAX
011A	CD3702	C 303	CALL PRTAT
		304	
011D	3A0F00	D 305	LDA LMAX
0120	47	306	MOV B,A
0121	3A0900	D 307	LDA TOPCY
0124	CD3702	C 308	CALL PRTAT
		309	
0127	3A0E00	D 310	LDA LMIN
012A	47	311	MOV B,A
012B	3A0A00	D 312	LDA BOTCX
012E	CD3702	C 313	CALL PRTAT
		314	
0131	3A0E00	D 315	LDA LMIN
0134	47	316	MOV B,A
0135	3A0800	D 317	LDA TOPCX
0138	CD3702	C 318	CALL PRTAT
		319	
013B	3A0F00	D 320	LDA LMAX
013E	47	321	MOV B,A
013F	3A0A00	D 322	LDA BOTCX
0142	CD3702	C 323	CALL PRTAT
		324	
0145	3A0B00	D 325	LDA BOTCY
0148	47	326	MOV B,A
0149	3A0C00	D 327	LDA CMIN
014C	CD3702	C 328	CALL PRTAT
		329	
014F	3A0900	D 330	LDA TOPCY
0152	47	331	MOV B,A
0153	3A0D00	D 332	LDA CMAX
0156	CD3702	C 333	CALL PRTAT
		334	
0159	3A0C00	D 335	LDA CMIN
015C	3C	336	INR A
015D	320C00	D 337	STA CMIN
		338	
0160	3A0D00	D 339	LDA CMAX
0163	3D	340	DCR A
0164	320D00	D 341	STA CMAX
		342	
0167	3A0E00	D 343	LDA LMIN
016A	3C	344	INR A
016B	320E00	D 345	STA LMIN
		346	
016E	3A0F00	D 347	LDA LMAX
0171	3D	348	DCR A
0172	320F00	D 349	STA LMAX
0175	C3F300	C 350	JMP DR05

```

; While Cmin<=Cmax AND Lmin<=Lmax do
; Begin

```

```

; PrintAT(Cmin, TopCornerY);

```

```

; PrintAT(Cmax, BottomCornerY);

```

```

; PrintAT(TopCornerY, Lmax);

```

```

; PrintAT(BottomCornerX, Lmin);

```

```

; PrintAT(TopCornerX, Lmin);

```

```

; PrintAT(BottomCornerX, Lmax);

```

```

; PrintAT(Cmin, BottomCornerY);

```

```

; PrintAT(Cmax, TopCornerY);

```

```

; Cmin:=Cmin+1;

```

```

; Cmax:=Cmax-1;

```

```

; Lmin:=Lmin+1;

```

```

; Lmax:=Lmax-1;

```

```

; End;

```

LOC	OBJ	LINE	SOURCE STATEMENT
		351	DR10:
0178	3A0E00	D 352	LDA LMIN ;
017B	47	353	MOV B,A ; If Lmin <= Lmax THEN
017C	3A0F00	D 354	LDA LMAX ;
017F	B8	355	CMP B ;
0180	FAC701	C 356	JM DR20 ;
		357	;
		358	DR15: ;
		359	While Lmin<=Lmax do ;
			Begin ;
0183	3A0E00	D 360	LDA LMIN ;
0186	47	361	MOV B,A ;
0187	3A0F00	D 362	LDA LMAX ;
018A	B8	363	CMP B ;
018B	FA0B02	C 364	JM DR25 ;
		365	;
018E	3A0F00	D 366	LDA LMAX ;
0191	47	367	MOV B,A ;
0192	3A0900	D 368	LDA TOPCY ;
0195	CD3702	C 369	CALL PRTAT ;
		370	PrintAT(TopCornerY, Lmax); ;
0198	3A0E00	D 371	LDA LMIN ;
019B	47	372	MOV B,A ;
019C	3A0A00	D 373	LDA BOTCX ;
019F	CD3702	C 374	CALL PRTAT ;
		375	PrintAT(BottomCornerX, Lmin); ;
01A2	3A0E00	D 376	LDA LMIN ;
01A5	47	377	MOV B,A ;
01A6	3A0800	D 378	LDA TOPCX ;
01A9	CD3702	C 379	CALL PRTAT ;
		380	PrintAT(TopCornerX, Lmin); ;
01AC	3A0F00	D 381	LDA LMAX ;
01AF	47	382	MOV B,A ;
01B0	3A0A00	D 383	LDA BOTCX ;
01B3	CD3702	C 384	CALL PRTAT ;
		385	PrintAT(BottomCornerX, Lmax); ;
01B6	3A0E00	D 386	LDA LMIN ;
01B9	3C	387	INR A ;
01BA	320E00	D 388	STA LMIN ;
		389	Lmin:=Lmin+1; ;
01BD	3A0F00	D 390	LDA LMAX ;
01C0	3D	391	DCR A ;
01C1	320F00	D 392	STA LMAX ;
		393	Lmax:=Lmax-1; ;
01C4	C38301	C 394	JMP DR15 ;
		395	DR20: ;
		396	Else ;
		397	While Cmin <= Cmax do ;
01C7	3A0C00	D 398	LDA CMIN ;
01CA	47	399	MOV B,A ;
01CB	3A0D00	D 400	LDA CMAX ;
01CE	B8	401	CMP B ;
01CF	FA0B02	C 402	JM DR25 ;
		403	Begin ;
01D2	3A0900	D 404	LDA TOPCY ;
01D5	47	405	MOV B,A ;
01D6	3A0C00	D 406	LDA CMIN ;
01D9	CD3702	C 407	CALL PRTAT ;
		408	PrintAT(Cmin, TopCornerY); ;
01DC	3A0B00	D 409	LDA BOTCY ;
01DF	47	410	MOV B,A ;
01E0	3A0D00	D 411	LDA CMAX ;
01E3	CD3702	C 412	CALL PRTAT ;
		413	PrintAT(Cmax, BottomCornerY); ;
01E6	3A0B00	D 414	LDA BOTCY ;
01E9	47	415	MOV B,A ;
01EA	3A0C00	D 416	LDA CMIN ;
01ED	CD3702	C 417	CALL PRTAT ;
		418	PrintAT(Cmin, BottomCornerY); ;
01F0	3A0900	D 419	LDA TOPCY ;
01F3	47	420	MOV B,A ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:24:20 20-APR-88

VDU

PAGE 7

VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
01F4	3A0D00	D 421	LDA CMAX ;
01F7	CD3702	C 422	CALL PRTAT ; PrintAT(Cmax, TopCornerY);
		423	; ;
01FA	3A0C00	D 424	LDA CMIN ;
01FD	3C	425	INR A ;
01FE	320C00	D 426	STA CMIN ; Cmin:=Cmin+1;
		427	; ;
0201	3A0D00	D 428	LDA CMAX ;
0204	3D	429	DCR A ;
0205	320D00	D 430	STA CMAX ; Cmax:=Cmax-1;
		431	; ;
0208	C3C701	C 432	JMP DR20 ; End;
		433 DR25:	; ;
		434	; ;
020B	C9	435	RET ; EndProc;
		436 ;	
		437 ;	
		438 ;	
		439 PVTOUT:	; Procedure Print_VDU_Output_Routine;
		440	; ;
020C	3A0000	D 441	LDA VTOUTR ; Case VDU_Output_Routine of
		442	; ;
020F	FE43	443	CPI 'C' ; 'C' :
0211	C21A02	C 444	JNZ PVT005 ;
		445	; ;
0214	21ED03	C 446	LXI H, VTOC ; Msg := VDU_output_Console;
0217	C33302	C 447	JMP PVT25 ;
		448	; ;
		449 PVT005:	; ;
021A	FE30	450	CPI '0' ; '0' :
021C	C22502	C 451	JNZ PVT010 ;
		452	; ;
021F	21FD03	C 453	LXI H, VTO0 ; Msg := VDU_output_USART0;
0222	C33302	C 454	JMP PVT25 ;
		455	; ;
		456 PVT010:	; ;
0225	FE31	457	CPI '1' ; '1' :
0227	C23002	C 458	JNZ PVT015 ;
		459	; ;
022A	210504	C 460	LXI H, VTO1 ; Msg := VDU_output_USART1;
022D	C33302	C 461	JMP PVT25 ;
		462	; ;
		463 PVT015:	; Else
		464	; ;
0230	210D04	C 465	LXI H, VTU ; Msg := VDU_output_unknown;
		466	; End;
		467 PVT25:	; ;
0233	CD0000	E 468	CALL OUTMSG ; OutMessage(Msg);
		469	; ;
0236	C9	470	RET ; EndProc;
		471 ;	
		472 ;	
		473 ;	
		474 ; B : Row	
		475 ; A : Col	
		476 ;	
		477 PRTAT:	; PrintAT(Row, Col : Byte);
0237	F5	478	PUSH PSW ;
		479	; ;
0238	3E1B	480	MVI A, ESC ;
023A	CD0100	D 481	CALL VTOUT ; VDU_output(ESC);
		482	; ;
023D	3E59	483	MVI A, GOTOC ;
023F	CD0100	D 484	CALL VTOUT ; VDU_output(GotoCommand);
		485	; ;
0242	78	486	MOV A, B ;
0243	C620	487	ADI OFFSET ;
0245	CD0100	D 488	CALL VTOUT ; VDU_output(Row+Offset);
		489	; ;
0248	F1	490	POP PSW ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:24:20 20-APR-88

VDU

PAGE 8

VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
0249	C620	491	ADI OFFSET ;
024B	CD0100	492	CALL VTOUT ; VDU_output(Col+Offset);
		493	;
024E	3A0600	494	LDA VTCHAR ;
0251	CD0100	495	CALL VTOUT ; VDU_output(VDU_Test_character);
		496	;
0254	C9	497	RET ; EndProc;
		498	;
		499	;
		500	;
		501	VTCLRF: ; Procedure VDU_test_NewLine;
0255	3E0D	502	MVI A,CR ;
0257	CD0100	503	CALL VTOUT ; VDU_output(CR);
025A	3E0A	504	MVI A,LF ;
025C	CD0100	505	CALL VTOUT ; VDU_output(LF);
		506	;
025F	C9	507	RET ; EndProc;
		508	;
		509	;
		510	;
		511	VTCLRS: ; Procedure VDU_test_Clear_screen;
0260	3E1B	512	MVI A,ESC ;
0262	CD0100	513	CALL VTOUT ; VDU_Output(ESC);
0265	3E48	514	MVI A,HOMEC ;
0267	CD0100	515	CALL VTOUT ; VDU_Output(Home_cursor);
026A	3E1B	516	MVI A,ESC ;
026C	CD0100	517	CALL VTOUT ; VDU_Output(ESC);
026F	3E4A	518	MVI A,CLRSC ;
0271	CD0100	519	CALL VTOUT ; VDU_Output(ClearToEndOfScreen);
		520	;
0274	C9	521	RET ; EndProc;
		522	;
		523	;
		524	;
		525	HL : Ptr (Pointer to the start address of the string.
		526	;
		527	VTOUTMSG: ; Procedure VDU_test_outmessage(Ptr:Word);
		528	;
0275	7E	529	MOV A,M ; While Memory[Ptr] <> 0 do
0276	FE00	530	CPI 0 ;
0278	CA8202	531	JZ VTOMEX ; Begin
		532	;
027B	CD0100	533	CALL VTOUT ; VDU_output(Memory[Ptr];
027E	23	534	INX H ; Ptr := Ptr + 1;
027F	C37502	535	JMP VTOUTMSG ; End;
		536	VTOMEX: ;
0282	C9	537	RET ; EndProc;
		538	;
		539	;
		540	;
		541	VTSETC: ; Procedure VDU_test_set_Console;
0283	210000	542	LXI H,CO ;
0286	220200	543	SHLD VTOUT+1 ; VDU_Output[1] := LOW CO;
		544	;
0289	3E43	545	MVI A,'C' ;
028B	320000	546	STA VTOUTR ; VDU_Output_routine := 'C';
		547	;
028E	C9	548	RET ; EndProc;
		549	;
		550	;
		551	;
		552	VTSET0: ; Procedure VDU_test_set_USART0;
028F	210000	553	LXI H,U00 ;
0292	220200	554	SHLD VTOUT+1 ; VDU_Output[1] := LOW USART0_Output;
		555	;
0295	3E30	556	MVI A,'0' ;
0297	320000	557	STA VTOUTR ; VDU_Output_routine := '0';
		558	;
029A	C9	559	RET ; EndProc;
		560	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:24:20 20-APR-88

VDU

PAGE 9

VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		561 ;	-----
		562 ;	
		563 VTSET1:	; Procedure VDU_test_set_USART1;
029B	210000	E 564	LXI H,U01 ;
029E	220200	D 565	SHLD VTOUT+1 ; VDU_Output[1] := LOW USART1_Output;
		566	; VDU_Output[2] := HIGH USART1_Output;
02A1	3E31	567	MVI A,'1' ;
02A3	320000	D 568	STA VTOUTR ; VDU_Output_routine := '1';
		569	;
02A6	C9	570	RET ; EndProc;
		571 ;	
		572 ;	*****
		573 ;	
		574 ;	VDU Message data.
		575 ;	
		576 VTMSG1:	
02A7	0D0A0909	577	DB CR,LF,TAB,TAB,TAB,TAB,' VDU test routines'
	09092056		
	44552074		
	65737420		
	726F7574		
	696E6573		
02BF	0D0A0909	578	DB CR,LF,TAB,TAB,TAB,TAB,' -----'
	0909202D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
	2D2D2D2D		
02D7	0D0A	579	DB CR,LF
02D9	0D0A	580	DB CR,LF
02DB	0D0A0909	581	DB CR,LF,TAB,TAB,TAB,' Direct cursor positioning test (D)'
	09202044		
	69726563		
	74206375		
	72736F72		
	20706F73		
	6974696F		
	6E696E67		
	20746573		
	74202020		
	20284429		
0307	0D0A	582	DB CR,LF
0309	0D0A0909	583	DB CR,LF,TAB,TAB,TAB,' Scroll test (S)'
	09202020		
	20202020		
	20202020		
	5363726F		
	6C6C2074		
	65737420		
	20202020		
	20202020		
	20202020		
	20285329		
0335	0D0A	584	DB CR,LF
0337	0D0A0909	585	DB CR,LF,TAB,TAB,TAB,'The following output devices are available'
	09546865		
	20666F6C		
	6C6F7769		
	6E67206F		
	75747075		
	74206465		
	76696365		
	73206172		
	65206176		
	61696C61		
	626C65		
0366	0D0A	586	DB CR,LF
0368	0D0A0909	587	DB CR,LF,TAB,TAB,TAB,' Console (C)'
	09202020		
	20202020		

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:24:20 20-APR-88 VDU PAGE 10
 VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
	20202020		
	20436F6E		
	736F6C65		
	20202843		
	29		
0385	0D0A0909	588	DB CR,LF,TAB,TAB,TAB,' USART0 (0)'
	09202020		
	20202020		
	20202020		
	20555341		
	52543020		
	20202830		
	29		
03A2	0D0A0909	589	DB CR,LF,TAB,TAB,TAB,' USART1 (1)'
	09202020		
	20202020		
	20202020		
	20555341		
	52543120		
	20202831		
	29		
03BF	0D0A	590	DB CR,LF
03C1	0D0A0909	591	DB CR,LF,TAB,TAB,' The output device is currently set to:',0
	20546865		
	206F7574		
	70757420		
	64657669		
	63652069		
	73206375		
	7272656E		
	746C7920		
	73657420		
	746F3A00		
		592 ;	
03ED	20537973	593	DB ' System console',0
	74656D20		
	636F6E73		
	6F6C6500		
03FD	20555341	594	DB ' USART0',0
	52543000		
0405	20555341	595	DB ' USART1',0
	52543100		
040D	20556E6B	596	DB ' Unknown',0
	6E6F776E		
	00		
		597 ;	
		598 VMSG2:	
0416	0D0A	599	DB CR,LF
0418	0D0A0909	600	DB CR,LF,TAB,TAB,TAB,' <ESC> or <space> to return to main menu'
	09203C45		
	53433E20		
	6F72203C		
	73706163		
	653E2074		
	6F207265		
	7475726E		
	20746F20		
	6D61696E		
	206D656E		
	75		
0445	0D0A	601	DB CR,LF
0447	0D0A0909	602	DB CR,LF,TAB,TAB,TAB,' Enter option >',0
	0920456E		
	74657220		
	6F707469		
	6F6E203E		
	00		
		603 ;	
045C	1B592C3C	604	DB ESC,GOTOC,12+OFFSET,28+OFFSET
0460	3C455343	605	DB '<ESC> or <space> to exit'

VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
		606	DB 0
		607 ;	
		608 ;	-----
		609 ;	
		610 ;	This table has the following format:
		611 ;	
		612 ;	Command_record
		613 ;	Command_record
		614 ;	.
		615 ;	.
		616 ;	Command_record
		617 ;	0 ; End for table marker.
		618 ;	
		619 ;	The format of each command record is as follows:
		620 ;	
		621 ;	Byte 0 : Command letter.
		622 ;	Byte 1 : Least significant byte of the command routine.
		623 ;	Byte 2 : Most significant byte of the command routine.
		624 ;	
		625 ;	-----
		626 ;	
		627	VTTAB: ; VDU test command table.
		628 ;	
0479 43		629	DB 'C' ;
047A 8302 C		630	DW VTSETC ; Set output device to CONSOLE.
		631 ;	
047C 30		632	DB 'O' ;
047D 8F02 C		633	DW VTSETO ; Set output device to USART0.
		634 ;	
047F 31		635	DB '1' ;
0480 9B02 C		636	DW VTSET1 ; Set output device to USART1.
		637 ;	
0482 44		638	DB 'D' ;
0483 7E00 C		639	DW DCURP ; Perform direct cursor positioning test.
		640 ;	
0485 53		641	DB 'S' ;
0486 4200 C		642	DW SCROLL ; Perform Scroll test.
		643 ;	
0488 00		644	DB 0 ; End of table marker.
		645 ;	
		646 ;	*****
		647 ;	
		648 ;	VDU test data area.
		649 ;	
		650	DSEG
		651 ;	
0000		652	VTOUTR: DS BYTE ; BYTE : VDU_output_routine;
		653	; { 'C' = CONSOLE,
		654	; 'O' = USART0,
		655	; '1' = USART1 }
		656 ;	
0001		657	VTOUT: DS 4 ; Dynamic VDU test output routine
		658 ;	
		659	; BYTE:0 (CALL)
		660	; BYTE:1 LSB of Address
		661	; BYTE:2 MSG of address
		662	; BYTE:3 (RET)
		663 ;	
0005		664	YPOS: DS BYTE ; BYTE : Y_position;
0006		665	VTCHAR: DS BYTE ; CHAR : VDU_test_character;
0007		666	STARTC: DS BYTE ; CHAR : Start_character;
		667 ;	
0008		668	TOPCX: DS BYTE ; BYTE : TopCornerX;
0009		669	TOPCY: DS BYTE ; BYTE : TopCornerY;
000A		670	BOTCX: DS BYTE ; BYTE : BottomCornerX;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:24:20 20-APR-88

VDU

PAGE 12

VDU test routines for the UCT SABus diagnostics

LOC	OBJ	LINE	SOURCE STATEMENT
000B		671	BOTCY: DS BYTE ; BYTE : BottomCornerY;
		672	;
000C		673	CMIN: DS BYTE ; BYTE : Cmin;
000D		674	CMAX: DS BYTE ; BYTE : Cmax;
000E		675	LMIN: DS BYTE ; BYTE : Lmin;
000F		676	LMAX: DS BYTE ; BYTE : Lmax;
		677	;
		678	End of <DIAGVDU.A85> module
		679	;
		680	END

0 error(s) detected

External symbols

?ESCAP	CI	CITEST	CLRS	CO	GETUC	INDIRE	OUTMSG	SEARCH	U00	U01

Public symbols

VINIT	VTEST

User symbols

?ESCAP 000A E	BELL 0007 A	BOTCX 000A D	BOTCY 000B D	BS 0008 A	BYTE 0001 A	CI 0005 E
CITEST 0008 E	CLRLC 004B A	CLRS 0004 E	CLRSC 004A A	CMAx 000D D	CMIN 000C D	CO 0006 E
CR 000D A	CTRLQ 0011 A	CTRLS 0013 A	CTRLZ 001A A	DC02 008D C	DC05 0092 C	DC10 00B1 C
DC15 00CD C	DC30 00D3 C	DCMSG 045C C	DCURP 007E C	DEL 007F A	DNCMD 0042 A	DR05 00F3 C
DR10 0178 C	DR15 0183 C	DR20 01C7 C	DR25 020B C	DRAW 00D4 C	EOT 0004 A	ESC 001B A
FALSE 0000 A	FF 000C A	GETUC 0002 E	GOTOC 0059 A	HONEC 0048 A	INDIRE 0007 E	K 0400 A
LF 000A A	LFCMD 0044 A	LMAX 000F D	LMIN 000E D	MAXX 004F A	MAXY 0017 A	MINX 0000 A
MINY 0000 A	NEXTC 0075 C	NEXTEX 007D C	NULL 0000 A	OFFSET 0020 A	OUTMSG 0003 E	PRTAT 0237 C
PVT25 0233 C	PVTO05 021A C	PVTO10 0225 C	PVTO15 0230 C	PVTOUT 020C C	QUOTE 0027 A	REVLf 0049 A
RTCMD 0043 A	SC02 004A C	SC10 004F C	SC20 0074 C	SCROLL 0042 C	SEARCH 0009 E	SPACE 0020 A
STARTC 0007 D	TAB 0009 A	TOPCX 0008 D	TOPCY 0009 D	TRUE 00FF A	U00 0001 E	U01 0000 E
UPCMD 0041 A	VINIT 002C C	VT05 0012 C	VTCHAR 0006 D	VTCLRS 0260 C	VTCLRF 0255 C	VTEST 0000 C
VTEXT 002B C	VTMSG1 02A7 C	VTMSG2 0416 C	VT00 03FD C	VT01 0405 C	VT0C 03ED C	VTOMEX 0282 C
VTOUT 0001 D	VTOUTM 0275 C	VTOUTR 0000 D	VTSET0 028F C	VTSET1 029B C	VTSETC 0283 C	VTTAB 0479 C
VTU 040D C	WORD 0002 A	YPOS 0005 D				

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$Title('SABus monitor interface section');
		2	\$PageLength(75)
		3	;
		4	NAME SABMON
		5	;
		6	;
		7	;
		8	; This module contains all the equates and routines necessary to allow
		9	; application programs to interface with the SABus monitor.
		10	;
		11	;
		12	;
		13	PUBLIC CITEST, UOTEST, U1TEST, CI, UI0, UI1, CO, UO0, UO1
		14	PUBLIC BREAK, GETUC, GETECHO, GETWD, ASCBIN, ECHO, PRVAL
		15	PUBLIC NMOUT, OUTMSG, DCMPI, DCMPI, TRANSP, VALDL, GETCM
		16	PUBLIC GETHX, GET1NM, GET2NM, GET3NM
		17	;
		18	PUBLIC COTEST
		19	;
		20	;
		21	;
F00A		22	CITEST EQU 0F00AH ; Test if char in USART 0
F00D		23	UOTEST EQU 0F00DH ; Test if char in USART 1
F010		24	U1TEST EQU 0F010H ; Test if char in USART 2
F013		25	CI EQU 0F013H ; Input char from USART 0
F016		26	UI0 EQU 0F016H ; Input char from USART 1
F019		27	UI1 EQU 0F019H ; Input char from USART 2
F01C		28	CO EQU 0F01CH ; Output char to USART 0
F01F		29	UO0 EQU 0F01FH ; Output char to USART 1
F022		30	UO1 EQU 0F022H ; Output char to USART 2
F025		31	BREAK EQU 0F025H ; Implement XON/XOFF protocol and sense ESC
F028		32	GETUC EQU 0F028H ; Input char from USART 0 - convert to UC
F02B		33	GETECHO EQU 0F02BH ; Input from console and echo
F02E		34	GETWD EQU 0F02EH ; Input up to four hex digits from console
F031		35	ASCBIN EQU 0F031H ; Convert ASCII to binary checking validity
F034		36	ECHO EQU 0F034H ; Output char to console checking for ESC, CR
F037		37	PRVAL EQU 0F037H ; Print nibble as hex digit
F03A		38	NMOUT EQU 0F03AH ; Print byte as two hex digits
F03D		39	OUTMSG EQU 0F03DH ; Print a string on the console
F043		40	DCMP EQU 0F043H ; Compare HL to DE
F046		41	DCMPI EQU 0F046H ; Compare (HL) to DE
F049		42	TRANSP EQU 0F049H ; Enter transparent communication mode
F04C		43	VALDL EQU 0F04CH ; Test if valid delimiter
F04F		44	GETCM EQU 0F04FH ; Return to monitor command fetch routine
F052		45	GETHX EQU 0F052H ; Fetch 0/1 hex number, error return to monitor
F055		46	GET1NM EQU 0F055H ; Fetch one hex number, error return to monitor
F058		47	GET2NM EQU 0F058H ; Fetch 2 hex numbers, error return to monitor
F05B		48	GET3NM EQU 0F05BH ; Fetch 3 hex numbers, error return to monitor
		49	;
		50	;
		51	;
0001		52	TXRDY EQU 00000001B ; USART Receiver ready status bit
		53	;
0000		54	CODATA EQU 0 ; Console USART data register
0001		55	COSTAT EQU 1 ; Console USART status register
		56	;
		57	CSEG ; Start of CODE segment
		58	;
		59	COTEST: ; Procedure ConsoleOutputTest : Boolean;
		60	;
0000 DB01		61	IN COSTAT ; If ConsoleStatus AND ReceiverReady then
		62	Return(TRUE)
0002 E601		63	ANI TXRDY ; Else
		64	Return(FALSE)
0004 C9		65	RET ; EndProc;
		66	;
		67	END ; End of SABMON module.

0 error(s) detected

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:21:23 20-APR-88 SABMON PAGE 2
SABus monitor interface section

LOC	OBJ	LINE	SOURCE STATEMENT									
Public symbols												
ASCBIN	BREAK	CI	CITEST	CO	COTEST	DCMP	DCMPI	ECHO	GET1NM	GET2NM	GET3NM	GETCM
GETECH	GETHX	GETUC	GETWD	NMOUT	OUTMSG	PRVAL	TRANSP	UOTEST	U1TEST	U10	U11	U00
U01	VALDL											

User symbols												
ASCBIN F031 A	BREAK F025 A	CI F013 A	CITEST F00A A	CO F01C A	CODATA 0000 A	COSTAT 0001 A						
COTEST 0000 C	DCMP F043 A	DCMPI F046 A	ECHO F034 A	GET1NM F055 A	GET2NM F058 A	GET3NM F05B A						
GETCM F04F A	GETECH F02B A	GETHX F052 A	GETUC F028 A	GETWD F02E A	NMOUT F03A A	OUTMSG F03D A						
PRVAL F037 A	TRANSP F049 A	TXRDY 0001 A	UOTEST F00D A	U1TEST F010 A	U10 F016 A	U11 F019 A						
U00 F01F A	U01 F022 A	VALDL F04C A										

APPENDIX J

LINK LISTING

The following pages contain an index of the symbols used in the LINK utility listing. In order to keep the index reasonably short, the local symbols, those defined and referenced within a subroutine, are not listed.

As well as the symbol name and page reference, a symbol type is also given. The possible symbol types are:

- a) CONSTANT (in this case a value is also given)
- b) SUBROUTINE
- c) BYTE ARRAY (in this case the length is also given)
- d) TASK
- e) TABLE
- f) MESSAGE

NOTE: The page reference given in the index below is the page on which the symbol is first defined.

<u>SYMBOL</u>	<u>DESCRIPTION</u>	<u>PAGE REFERENCE</u>
ASCBIN	CONSTANT, VALUE = 0F031H.....	J-40
ASCII	CONSTANT, VALUE = 0.....	J-10
BELL	CONSTANT, VALUE = 007H.....	J-9
BREAK	CONSTANT, VALUE = 0F025H.....	J-40
BS	CONSTANT, VALUE = 008H.....	J-9
BUFINIT	SUBROUTINE.....	J-17
BYTE	CONSTANT, VALUE = 1.....	J-9
CH	BYTE VARIABLE.....	J-38
CHANGE	SUBROUTINE.....	J-35
CHARS	WORD VARIABLE.....	J-21
CHRBUF	BYTE ARRAY, LENGTH = ENTLEN.....	J-38
CHROUT	SUBROUTINE.....	J-36
CI	CONSTANT, VALUE = 0F013H.....	J-40
CITEST	CONSTANT, VALUE = 0F00AH.....	J-40
CLOST	MESSAGE.....	J-20
CLRLC	CONSTANT, VALUE = 'K'.....	J-22
CLRS	SUBROUTINE.....	J-23
CLRSC	CONSTANT, VALUE = 'J'.....	J-22
CMPBH	SUBROUTINE.....	J-20

CMPDH	SUBROUTINE.....	J-20
CMPHB	SUBROUTINE.....	J-20
CMPHD	SUBROUTINE.....	J-20
CO	CONSTANT, VALUE = 0F01CH.....	J-40
CODATA	CONSTANT, VALUE = 0.....	J-40
COL	BYTE VARIABLE.....	J-25
CONNIB	SUBROUTINE.....	J-24
COSTAT	CONSTANT, VALUE = 1.....	J-40
COTEST	SUBROUTINE.....	J-40
CR	CONSTANT, VALUE = 00DH.....	J-9
CTRLQ	CONSTANT, VALUE = 011H.....	J-9
CTRLS	CONSTANT, VALUE = 013H.....	J-9
CTRLZ	CONSTANT, VALUE = 01AH.....	J-9
DCMP	CONSTANT, VALUE = 0F043H.....	J-40
DCMPI	CONSTANT, VALUE = 0F046H.....	J-40
DEL	CONSTANT, VALUE = 07FH.....	J-9
DELENT	MESSAGE.....	J-13
DNCMD	CONSTANT, VALUE = 'B'.....	J-22
ECHO	CONSTANT, VALUE = 0F034H.....	J-40
EDINIT	SUBROUTINE.....	J-36
EDIT	SUBROUTINE.....	J-34
EDMSG1	MESSAGE.....	J-38
EDMSG2	MESSAGE.....	J-38
EDMSG3	MESSAGE.....	J-38
EDMSG4	MESSAGE.....	J-38
EDWRT	SUBROUTINE.....	J-36
ENTER	SUBROUTINE.....	J-35
ENTLEN	CONSTANT, VALUE = 6.....	J-10
EOT	CONSTANT, VALUE = 004H.....	J-9
ESC	CONSTANT, VALUE = 01BH.....	J-9
FALSE	CONSTANT, VALUE = 0.....	J-9
FF	CONSTANT, VALUE = 00CH.....	J-9
GET1NM	CONSTANT, VALUE = 0F055H.....	J-40
GET2NM	CONSTANT, VALUE = 0F058H.....	J-40
GET3NM	CONSTANT, VALUE = 0F05BH.....	J-40
GETADR	SUBROUTINE.....	J-36
GETCM	CONSTANT, VALUE = 0F04FH.....	J-40
GETECHO	CONSTANT, VALUE = 0F02BH.....	J-40
GETHEX	SUBROUTINE.....	J-34
GETHX	CONSTANT, VALUE = 0F052H.....	J-40
GETUC	CONSTANT, VALUE = 0F028H.....	J-40
GETWD	CONSTANT, VALUE = 0F02EH.....	J-40
GOTO	SUBROUTINE.....	J-23
GOTOC	CONSTANT, VALUE = 'Y'.....	J-22
GTPPOS	SUBROUTINE.....	J-37
HEADER	MESSAGE.....	J-25
HEX	CONSTANT, VALUE = 1.....	J-10
HOME	SUBROUTINE.....	J-23
HOMECL	CONSTANT, VALUE = 'H'.....	J-22
IBUF	BYTE ARRAY, LENGTH = IBUFL.....	J-20

IBUFIP	WORD VARIABLE.....	J-21
IBUFL	CONSTANT, VALUE = 8000.....	J-17
IBUFOP	WORD VARIABLE.....	J-21
IBUFR	SUBROUTINE.....	J-19
IBUFW	SUBROUTINE.....	J-18
INIT	SUBROUTINE.....	J-11
IPTRI	SUBROUTINE.....	J-19
ITAB	TABLE.....	J-12
K	CONSTANT, VALUE = 1024.....	J-10
KBDTSK	TASK.....	J-28
KSTKL	CONSTANT, VALUE = 50.....	J-10
LF	CONSTANT, VALUE = 00AH.....	J-9
LFCMD	CONSTANT, VALUE = 'D'.....	J-22
LINE	BYTE VARIABLE.....	J-25
LOCKED	BYTE VARIABLE.....	J-21
LOWLIM	CONSTANT, VALUE = 2000.....	J-17
MAINTSK	SUBROUTINE.....	J-11
MAXX	CONSTANT, VALUE = 79.....	J-22
MAXY	CONSTANT, VALUE = 23.....	J-22
MCLRS	SUBROUTINE.....	J-37
MEMPTR	WORD VARIABLE.....	J-16
MGOTO	SUBROUTINE.....	J-37
MHOME	SUBROUTINE.....	J-37
MINX	CONSTANT, VALUE = 0.....	J-22
MINY	CONSTANT, VALUE = 0.....	J-22
MODE	BYTE VARIABLE.....	J-25
MSGWRT	SUBROUTINE.....	J-18
MSTKL	CONSTANT, VALUE = 50.....	J-10
NCHARS	CONSTANT, VALUE = 128.....	J-10
NEWTASK	SUBROUTINE.....	J-15
NMOUT	CONSTANT, VALUE = 0F03AH.....	J-40
NMPRT	SUBROUTINE.....	J-24
NUL	CONSTANT, VALUE = 000H.....	J-9
NUMTSK	BYTE VARIABLE.....	J-16
OBUF	BYTE ARRAY, LENGTH = OBUFL.....	J-21
OBUFIP	WORD VARIABLE.....	J-21
OBUFL	CONSTANT, VALUE = 4000.....	J-17
OBUFOP	WORD VARIABLE.....	J-21
OBUFR	SUBROUTINE.....	J-19
OBUFW	SUBROUTINE.....	J-19
OFFSET	CONSTANT, VALUE = 20H.....	J-22
OPTRI	SUBROUTINE.....	J-20
OUTMSG	CONSTANT, VALUE = 0F03DH.....	J-40
POS	SUBROUTINE.....	J-24
PRINT	SUBROUTINE.....	J-24
PRTMSG	SUBROUTINE.....	J-25
PRVAL	CONSTANT, VALUE = 0F037H.....	J-40
QUOTE	CONSTANT, VALUE = 027H.....	J-9
REVLf	CONSTANT, VALUE = 'I'.....	J-22
RTCMD	CONSTANT, VALUE = 'C'.....	J-22

SCNINIT	SUBROUTINE.....	J-23
SCROLL	MESSAGE.....	J-25
SIOTSK	TASK.....	J-30
SPACE	CONSTANT, VALUE = 020H.....	J-9
SSTKL	CONSTANT, VALUE = 50.....	J-10
START	MAIN PROGRAM.....	J-10
STARTM	MESSAGE.....	J-25
STOP	BYTE VARIABLE.....	J-25
STOPM	MESSAGE.....	J-29
SXFER	SUBROUTINE.....	J-16
TAB	CONSTANT, VALUE = 009H.....	J-9
TABINIT	SUBROUTINE.....	J-12
TABLE	TABLE.....	J-14
TASKINIT	SUBROUTINE.....	J-15
TASKN	BYTE VARIABLE.....	J-16
TASKQ	STRUCTURE.....	J-16
TRANSP	CONSTANT, VALUE = 0F049H.....	J-40
TRUE	CONSTANT, VALUE = NOT FALSE.....	J-9
TSKPTR	WORD VARIABLE.....	J-16
TXRDY	CONSTANT, VALUE = 00000001B.....	J-40
U0TEST	CONSTANT, VALUE = 0F00DH.....	J-40
U1TEST	CONSTANT, VALUE = 0F010H.....	J-40
UI0	CONSTANT, VALUE = 0F016H.....	J-40
UI1	CONSTANT, VALUE = 0F019H.....	J-40
U00	CONSTANT, VALUE = 0F01FH.....	J-40
U01	CONSTANT, VALUE = 0F022H.....	J-40
UPCMD	CONSTANT, VALUE = 'A'.....	J-22
UPLIM	CONSTANT, VALUE = IBUFL-20.....	J-17
VALDL	CONSTANT, VALUE = 0F04CH.....	J-40
VDUINIT	SUBROUTINE.....	J-31
VDUTSK	TASK.....	J-31
VSTKL	CONSTANT, VALUE = 50.....	J-10
WORD	CONSTANT, VALUE = 2.....	J-9
XFER	SUBROUTINE.....	J-15

```

.ENABLE SUBSTITUTION
.DISABLE LOWERCASE
;
; This command file assembles, links and locates the SABus line monitor
;
; Enter the letters corresponding to the modules to be assembled
; <return> assembles all modules.
;
; M - LINKMAIN      P - LINKPROC    B - LINKBUF    S - LINKSCN
; I - LINKSIO       K - LINKKBD     V - LINKVDU    E - LINKEDIT
;
.RETRY2:
.ASKS [1.:8: "MPBSIKVE"] MODS Modules to assemble?
.SETF MAIN
.SETF PROC
.SETF BUF
.SETF SCN
.SETF SIO
.SETF KBD
.SETF VDU
.SETF EDIT
.SETN MODCT <STRLEN>
.GMODS:
.SETS TMOD MODS[MODCT:MODCT]
.TEST "MPBSIKVE" TMOD
.IF <STRLEN> <> 0 .GOTO MODOK
;
; Don't know module "TMOD"
;
.GOTO RETRY2
.MODOK:
.IF TMOD = "M" .SETT MAIN
.IF TMOD = "P" .SETT PROC
.IF TMOD = "B" .SETT BUF
.IF TMOD = "S" .SETT SCN
.IF TMOD = "I" .SETT SIO
.IF TMOD = "K" .SETT KBD
.IF TMOD = "V" .SETT VDU
.IF TMOD = "E" .SETT EDIT
.DEC MODCT
.IF MODCT <> 0 .GOTO GMODS
.;
.ASK FULIST Full assembler listing
.IFT FULIST .SETS SW "/FU/SY"
.IFF FULIST .SETS SW "/-FU/-SY"
.;
.;      Start assembly of selected modules:
.;
.IFF MAIN .GOTO NOMAIN
;
;      Assemble the main module LINKMAIN.A85
;
PIP LINKMAIN.O85;*/DE/NM
AS5 LINKMAIN/-DB, LINKMAIN'SW'=LINKMAIN
.IF <EXSTAT> <> 1 .GOTO AFAIL

```

```

.;
.NOMAIN:
.IFF PROC .GOTO NOPROC
;
;       Assemble the task swapping module LINKPROC.A85
;
PIP LINKPROC.O85;*/DE/NM
AS5 LINKPROC/-DB,LINKPROC'SW'=LINKPROC
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NOPROC:
.IFF BUF .GOTO NOBUF
;
;       Assemble buffer routines: LINKBUF.A85
;
PIP LINKBUF.O85;*/DE/NM
AS5 LINKBUF/-DB,LINKBUF'SW'=LINKBUF
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NOBUF:
.IFF SCN .GOTO NOSCN
;
;       Assemble screen routines: LINKSCN.A85
;
PIP LINKSCN.O85;*/DE/NM
AS5 LINKSCN/-DB,LINKSCN'SW'=LINKSCN
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NOSCN:
.IFF SIO .GOTO NOSIO
;
;       Assemble task LINKSIO.A85
;
PIP LINKSIO.O85;*/DE/NM
AS5 LINKSIO/-DB,LINKSIO'SW'=LINKSIO
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NOSIO:
.IFF KBD .GOTO NOKBD
;
;       Assemble keyboard task LINKKBD.A85
;
PIP LINKKBD.O85;*/DE/NM
AS5 LINKKBD/-DB,LINKKBD'SW'=LINKKBD
.IF <EXSTAT> <> 1 .GOTO AFAIL
.;
.NOKBD:
.IFF VDU .GOTO NOVDU
;
;       Assemble VDU task LINKVDU.A85
;
PIP LINKVDU.O85;*/DE/NM
AS5 LINKVDU/-DB,LINKVDU'SW'=LINKVDU
.IF <EXSTAT> <> 1 .GOTO AFAIL
.NOVDU:

```

```

.IFF EDIT .GOTO NOEDIT
;
;       Assemble EDIT routine LINKEDIT.A85
;
PIP LINKEDIT.O85;*/DE/NM
AS5 LINKEDIT/-DB,LINKEDIT'SW'=LINKEDIT
.IF <EXSTAT> <> 1 .GOTO AFAIL
.NOEDIT:
;
;-----
;
;       Link all modules and system library.
;
.OPEN DDD.CMD
.ENABLE DATA
LINK.REL,TI:/ES=LINKMAIN
LINKPROC, LINKBUF, LINKSCN, LINKSIO, LINKKBD, LINKVDU, LINKEDIT, SABMON
.DISABLE DATA
.CLOSE DDD.CMD
LK5 @DDD
.IF <EXSTAT> <> 1 .GOTO LFAIL
PIP DDD.CMD;*/DE/NM
;
;       Locate LINK monitor
;
.OPEN DDD.CMD
.ENABLE DATA
LINK.ABS,TI:=LINK.REL
ORDER=CODE,DATA,STACK
LIMIT=0:EFFF
CODE=1000
//
.DISABLE DATA
.CLOSE DDD.CMD
LC5 @DDD.CMD
.IF <EXSTAT> <> 1 .GOTO LCFAIL
PIP DDD.CMD;*/DE/NM
;
;       Convert absolute object file to downloadable hex file
;
OH5 LINK.H85=LINK.ABS
.;
.;       Clean up by removing absolute object and relocatable object.
.;
PIP LINK.REL;*/DE/NM
.;
.EXIT
.;
;-----
.;
.;       Error exit points
.;
.AFAIL:
;
; Assembly failed.

```

LINK LISTING

```
;
.EXIT
.;
.LFAIL:
;
; link failed.
;
.EXIT
.;
.LCFAIL:
;
; Locate failed.
;
.EXIT
```

LOC OBJ LINE SOURCE STATEMENT

```

1 $TITLE('RS-232 line monitor')
2 $pagelength(75)
3 ;
4     NAME    LINKMAIN
5 ;
6 -----;
7 ;
8 ;
9     Link monitor for the SABus kits.
10 ;
11     Hardware: 1 SABus kit with:
12             2 RS-232 lines
13             VDU and keyboard.
14 ;
15     Connection:
16 ;
17     +-----+
18     |         |
19     | HOST system |-----+ BOX |-----+ Remote system |
20     |         |
21     +-----+
22
23     |
24     +-----+
25     |         |
26     |     KIT     |
27     |         |
28     +-----+
29 -----;
30 ;
31 $INCLUDE(ASCII.A85)
32 ;     NAME    ASCII
33 -----;
34 ;
35     PUBLIC  NULL, EOT, BELL, BS, TAB, LF, CR, FF
36     PUBLIC  CTRLS, CTRLQ, CTRLZ, ESC, SPACE, QUOTE, DEL
37 ;
38 -----;
39     ASCII constants.
40 ;
0000 = 41 NULL    EQU    000H        ; Null
0004 = 42 EOT    EQU    004H        ; End of text marker.
0007 = 43 BELL   EQU    007H        ; Bell.
0008 = 44 BS     EQU    008H        ; Back space.
0009 = 45 TAB    EQU    009H        ; Horiz Tab.
000A = 46 LF     EQU    00AH        ; Line feed.
000D = 47 CR     EQU    00DH        ; Carriage return.
000C = 48 FF     EQU    00CH        ; Form Feed.
0013 = 49 CTRLS  EQU    013H        ; Control 'S'
0011 = 50 CTRLQ  EQU    011H        ; Control 'Q'
001A = 51 CTRLZ  EQU    01AH        ; Control 'Z'
001B = 52 ESC    EQU    01BH        ; ESC
0020 = 53 SPACE  EQU    020H        ; ASCII Space.
0027 = 54 QUOTE  EQU    027H        ; Single quote (')
007F = 55 DEL    EQU    07FH        ; ASCII Delete character.
56 ;
57 ;     END
58 $INCLUDE(COMEQU.A85)
59 ;
60 -----;
61 ;
62     Logical equates.
63 ;
0000 = 64 FALSE  EQU    0            ; Logical FALSE.
00FF = 65 TRUE   EQU    (NOT FALSE) AND OFFH ; Logical TRUE.
66 ;
67     Storage size Equates ('In terms of bytes ).
68 ;
0002 = 69 WORD   EQU    2            ; 1 Word = 2 Bytes
0001 = 70 BYTE   EQU    1            ; 1 Byte = 1 Byte.

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
RS-232 line monitor

14:18:27 21-APR-88

LINKMA PAGE 2

LOC	OBJ	LINE	SOURCE STATEMENT
0400		= 71 K	EQU 1024 ; 1kbytes = 1024 bytes
		= 72 ;	
		73 ;	
		74 ;	-----
		75 ;	
		76 ;	Equates:
		77 ;	
		78 ;	Mode flag values:
		79 ;	
0000		80 ASCII	EQU 0 ; ASCII mode flag
0001		81 HEX	EQU 1 ; HEX mode flag
		82 ;	
		83 ;	ASCII lookup table size equates:
		84 ;	
0080		85 NCHARS	EQU 128 ; Number of characters in lookup table
0006		86 ENTLEN	EQU 6 ; Lookup table entry length
		87 ;	
		88 ;	-----
		89 ;	
0020		90	STKLN 32 ; Short main stack
		91 ;	
		92 ;	-----
		93 ;	
		94 ;	External routines:
		95 ;	
		96	EXTRN TASKINIT, NEWTASK, XFER ; IMPORT task swapping routines.
		97	EXTRN BUFINIT ; IMPORT buffer init routine
		98	EXTRN HOME, CLRS, SCNINIT ; IMPORT screen_IO routines
		99	EXTRN MODE ; IMPORT MODE flag
		100	EXTRN PRINT, POS, NMPRT ; IMPORT Screen I/O routines
		101	EXTRN IBUFR ; IMPORT Input buffer read routine
		102	EXTRN UOTEST, U1TEST, UI1, UI0 ; IMPORT USART I/O routines
		103	EXTRN SXFER ; IMPORT Task swapping routine
		104	EXTRN MSGWRT ; IMPORT Message output routine
		105 ;	
		106 ;	-----
		107 ;	
		108 ;	Declare tasks:
		109 ;	
		110	EXTRN SIOTSK ; IMPORT serial polling task
		111	EXTRN VDUTSK ; IMPORT VDU output task
		112	EXTRN KBDTSK ; IMPORT keyboard task
		113 ;	
		114 ;	-----
		115 ;	
		116 ;	Define global symbols:
		117 ;	
		118	PUBLIC TABLE, ENTLEN, NCHARS ; EXPORT data structures
		119 ;	
		120 ;	
		121 ;	-----
		122 ;	
		123 ;	Stack sizes:
		124 ;	
0032		125 MSTKL	EQU 50 ; Main Task stack size
0032		126 SSTKL	EQU 50 ; SIO task stack size
0032		127 VSTKL	EQU 50 ; VDU task stack size
0032		128 KSTKL	EQU 50 ; KBD task stack size
		129 ;	
		130 ;	-----
		131 ;	
		132	CSEG ; Start of code segment
		133 ;	
		134 START:	; PROGRAM Start;
		135 ;	
0000	310000	S 136	LXI SP,STACK ; InitStackPointer;
0003	CD2D00	C 137	CALL INIT ; Initialise;
		138 ;	
0006	214B00	C 139	LXI H,MAINTSK ;
0009	113200	C 140	LXI D,MSTKL ;

LINK LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0
RS-232 line monitor

14:18:27 21-APR-88

LINKMA PAGE 3

LOC	OBJ	LINE	SOURCE STATEMENT
000C	CD0000	E 141	CALL NEWTASK ; NewTask(MainTask, MainTaskSL);
		142	;
000F	210000	E 143	LXI H,SIOTSK ;
0012	113200	144	LXI D,SSTKL ;
0015	CD0000	E 145	CALL NEWTASK ; NewTask(SIOTask, SIOTaskSL);
		146	;
0018	210000	E 147	LXI H,VDUTSK ;
001B	113200	148	LXI D,VSTKL ;
001E	CD0000	E 149	CALL NEWTASK ; NewTask(VDUTask, VDUTaskSL);
		150	;
0021	210000	E 151	LXI H,KBDTSK ;
0024	113200	152	LXI D,KSTKL ;
0027	CD0000	E 153	CALL NEWTASK ; NewTask(KBDTask, KBDTaskSL);
		154	;
002A	CD0000	E 155	CALL XFER ; SwapTasks;
		156	; EndProgram;
		157 ;	
		158 ;-----	
		159 ;	
		160 ; Initialisation:	
		161 INIT: ; Procedure Initialise;	
		162 ;	
002D	CD0000	E 163	CALL UOTEST ; If U0_Rec_ready then
0030	C40000	E 164	CNZ UI0 ; Ch := USART0_input; {Clear USART 0};
		165	;
0033	CD0000	E 166	CALL UI1TEST ; If U1_Rec_ready then
0036	C40000	E 167	CNZ UI1 ; Ch := USART1_input; {Clear USART 1};
		168	;
0039	3E00	169	MVI A,ASCII ;
003B	320000	E 170	STA MODE ; Mode := ASCII;
		171	;
003E	CD0000	E 172	CALL TASKINIT ; InitialiseMultitasking;
0041	CD0000	E 173	CALL BUFINIT ; InitialiseBuffers;
0044	CD0000	E 174	CALL SCNINIT ; InitialiseScreen;
0047	CD7E00	C 175	CALL TABINIT ; InitialiseLookupTables;
		176	;
004A	C9	177	RET ; EndProc;
		178 ;	
		179 ;=====	
		180 ;	
		181 MAINTSK: ; Task MainTask;	
		182 ; Repeat	
004B	CD0000	E 183	CALL IBUFR ; Ch := InputBufferRead;
004E	4F	184	MOV C,A ;
004F	3A0000	E 185	LDA MODE ;
0052	FE00	186	CPI ASCII ; Case Mode of
0054	CA6800	C 187	JZ ASCIIIM ;
		188	;
		189	HEX : Begin
0057	3E03	189	MVI A,3 ;
0059	CD0000	E 190	CALL POS ; Position(3);
005C	79	191	MOV A,C ;
005D	CD0000	E 192	CALL NMPRT ; PrintNumber(Ch);
0060	3E20	193	MVI A,SPACE ;
0062	CD0000	E 194	CALL PRINT ; Print(Space);
		195	;
0065	C34B00	C 196	JMP MAINTSK ; End;
		197	;
		198	;
		199 ASCIIIM: ; ASCII : Begin	
0068	79	200	MOV A,C ;
0069	E67F	201	ANI 01111111B ; Ch := Ch and 01111111B
006B	2600	202	MVI H,0 ;
006D	6F	203	MOV L,A ;
006E	29	204	DAD H ;
006F	54	205	MOV D,H ;
0070	5D	206	MOV E,L ;
0071	29	207	DAD H ;
0072	19	208	DAD D ;
0073	110000	D 209	LXI D,TABLE ;
0076	19	210	DAD D ;
			Ptr := Ch * 6;
			Ptr := Ptr + TABLE;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
RS-232 line monitor

14:18:27 21-APR-88

LINKMA PAGE 4

LOC	OBJ	LINE	SOURCE STATEMENT
0077	EB	211	XCHG ;
0078	CD0000 E	212	CALL MSGWRT ; PrintMessage(Ptr);
007B	C34B00 C	213	JMP MAINTSK ; End;
		214	; EndTask
		215	;
		216	=====
		217	;
		218	TABINIT: ; Procedure InitTable;
		219	;
		220	; Copy the initial control character table into the main character
		221	lookup table.
		222	;
007E	210000 D	223	LXI H, TABLE ; { Table Pointer in HL, TabPtr }
0081	11AF00 C	224	LXI D, ITAB ; { Init Table Pointer in DE, InitPtr }
0084	01C000	225	LXI B, ITABL ; { Init Table Length in BC, TabLen }
		226	;
		227	TABIO5: ; For I := 1 to TabLen do
		228	; Begin
0087	1A	229	LDAX D ; Ch := Memory(InitPtr);
0088	77	230	MOV M, A ; Memory(TabPtr) := Ch;
0089	23	231	INX H ; TabPtr := TabPtr + 1;
008A	13	232	INX D ; InitPtr := InitPtr + 1;
008B	0B	233	DCX B ;
008C	79	234	MOV A, C ;
008D	B0	235	ORA B ;
008E	C28700 C	236	JNZ TABIO5 ; End;
		237	;
		238	; Copy the printable ASCII character set into the table, all entries
		239	being one character long.
		240	;
0091	3E20	241	MVI A, ' ' ; Ch := Ord(' ');
0093	110500	242	LXI D, ENTLN-1 ;
		243	TABII10: ; Repeat
0096	3601	244	MVI M, 1 ; Memory(TabPtr) := 1;
0098	23	245	INX H ; TabPtr := TabPtr + 1;
0099	77	246	MOV M, A ; Memory(TabPtr) := Ch
009A	19	247	DAD D ; TabPtr := TabPtr + EntryLength - 1;
009B	3C	248	INR A ; Ch := Ch + 1;
009C	FE7F	249	CPI 7FH ;
009E	C29600 C	250	JNZ TABII10 ; until Ch = 7FH
		251	;
		252	; Copy the delete key entry into the table
		253	;
00A1	0606	254	MVI B, 6 ;
00A3	116F01 C	255	LXI D, DELENT ;
		256	TABII15: ; For I := 1 to 6 do
		257	; Begin
00A6	1A	258	LDAX D ; Ch := Memory(DeleteEntry+I);
00A7	77	259	MOV M, A ; Memory(TabPtr) := Ch;
00A8	23	260	INX H ; TabPtr := TabPtr + 1;
00A9	13	261	INX D ;
00AA	05	262	DCR B ;
00AB	C2A600 C	263	JNZ TABII15 ; End;
		264	;
00AE	C9	265	RET ; EndProc;
		266	;
		267	=====
		268	;
		269	; Initial table of control character strings:
		270	;
		271	ITAB:
00AF	053C4E55	272	DB 5, '<NUL>'
	4C3E		
00B5	053C534F	273	DB 5, '<SOH>'
	483E		
00BB	053C5354	274	DB 5, '<STX>'
	583E		
00C1	053C4554	275	DB 5, '<ETX>'
	583E		
		276	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
RS-232 line monitor

14:18:27 21-APR-88

LINKMA PAGE 5

LOC	OBJ	LINE	SOURCE STATEMENT
00C7	053C454F	277	DB 5, '<EOT>'
	543E		
00CD	053C454E	278	DB 5, '<ENQ>'
	513E		
00D3	053C4143	279	DB 5, '<ACK>'
	4B3E		
00D9	053C4245	280	DB 5, '<BEL>'
	4C3E		
		281	
00DF	043C4253	282	DB 4, '<BS>',0
	3E00		
00E5	043C4854	283	DB 4, '<HT>',0
	3E00		
00EB	043C4C46	284	DB 4, '<LF>',0
	3E00		
00F1	043C5654	285	DB 4, '<VT>',0
	3E00		
		286	
00F7	043C4646	287	DB 4, '<FF>',0
	3E00		
00FD	043C4352	288	DB 4, '<CR>',0
	3E00		
0103	043C534F	289	DB 4, '<SO>',0
	3E00		
0109	043C5349	290	DB 4, '<SI>',0
	3E00		
		291	
010F	053C444C	292	DB 5, '<DLE>'
	453E		
0115	053C4443	293	DB 5, '<DC1>'
	313E		
011B	053C4443	294	DB 5, '<DC2>'
	323E		
0121	053C4443	295	DB 5, '<DC3>'
	333E		
		296	
0127	053C4443	297	DB 5, '<DC4>'
	343E		
012D	053C4E41	298	DB 5, '<NAK>'
	4B3E		
0133	053C5359	299	DB 5, '<SYN>'
	4E3E		
0139	053C4554	300	DB 5, '<ETB>'
	423E		
		301	
013F	053C4341	302	DB 5, '<CAN>'
	4E3E		
0145	043C454D	303	DB 4, '',0
	3E00		
014B	053C5355	304	DB 5, '<SUB>'
	423E		
0151	053C4553	305	DB 5, '<ESC>'
	433E		
		306	
0157	043C4653	307	DB 4, '<FS>',0
	3E00		
015D	043C4753	308	DB 4, '<GS>',0
	3E00		
0163	043C5253	309	DB 4, '<RS>',0
	3E00		
0169	043C5553	310	DB 4, '<US>',0
	3E00		
		311 ;	
00C0		312 ITABL EQU	\$-ITAB
		313 ;	
016F	053C4445	314 DELENT: DB	5, '<DEL>'
	4C3E		
		315 ;	
		316 ;	
		317 ;	

UCT 8080/8085 CROSS ASSEMBLER, V1.0
RS-232 line monitor

14:18:27 21-APR-88

LINKMA PAGE 6

LOC	OBJ	LINE	SOURCE STATEMENT
		318	DSEG ; Start of the data segment
		319	TABLE:
0000		320	DS ENTLEN * NCHARS
		321	;
0000		322	END START ; End of <LINKMAIN.A85>

0 error(s) detected

External symbols

BUFINI	CLRS	HOME	IBUFR	KBDTSK	MODE	MSGWRT	NEWTAS	NMPRT	POS	PRINT	SCNINI	SIOTSK
SXFER	TASKIN	UOTEST	UI TEST	UIO	UI1	VDUTSK	XFER					

Public symbols

ENTLEN NCHARS TABLE

User symbols

ASCII	0000 A	ASCIIM	0068 C	BELL	0007 A	BS	0008 A	BUFINI	0014 E	BYTE	0001 A	CLRS	0008 E
CR	000D A	CTRLQ	0011 A	CTRLS	0013 A	CTRLZ	001A A	DEL	007F A	DELENT	016F C	ENTLEN	0006 A
EOT	0004 A	ESC	001B A	FALSE	0000 A	FF	000C A	HEX	0001 A	HOME	0010 E	IBUFR	0011 E
INIT	002D C	ITAB	00AF C	ITABL	00C0 A	K	0400 A	KBDTSK	0007 E	KSTKL	0032 A	LF	000A A
MAINTS	004B C	MODE	0004 E	MSGWRT	000A E	MSTKL	0032 A	NCHARS	0080 A	NEWTAS	0012 E	NMPRT	0003 E
NULL	0000 A	POS	000D E	PRINT	000C E	QUOTE	0027 A	SCNINI	0006 E	SIOTSK	0005 E	SPACE	0020 A
SSTKL	0032 A	START	0000 C	SXFER	0001 E	TAB	0009 A	TABI05	0087 C	TABI10	0096 C	TABI15	00A6 C
TABINI	007E C	TABLE	0000 D	TASKIN	0002 E	TRUE	00FF A	UOTEST	0013 E	UI TEST	000F E	UIO	0000 E
UI1	000E E	VDUTSK	000B E	VSTKL	0032 A	WORD	0002 A	XFER	0009 E				

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$Title('Multi-tasking kernel for use by the LINK utility')
		2	\$Pagelength(75)
		3	;
		4	NAME TASKING
		5	;
		6	-----
		7	;
		8	PUBLIC TASKINIT ; EXPORT Kernal initialisation routine
		9	PUBLIC NEWTASK ; EXPORT Task initialisation routine
		10	PUBLIC SXFER, XFER ; EXPORT Task swapping routines.
		11	;
		12	-----
		13	;
		14	CSEG ; Start of the code segment
		15	;
		16	TASKINIT: ; Procedure KernalInit;
0000	210000	17	LXI H, MEMORY ;
0003	221A00	18	SHLD MEMPTR ; Memory_pointer := MEMORY;
0006	3E00	19	MVI A, 0 ;
0008	321800	20	STA NUMTSK ; Number_of_task := 0;
000B	321900	21	STA TASKN ;
000E	210200	22	LXI H, TASKQ-2 ;
0011	220000	23	SHLD TSKPTR ; Task_pointer := Top_of_task_queue;
0014	C9	24	RET ; Endproc;
		25	;
		26	-----
		27	;
		28	HL = Start address of task
		29	DE = Stack space wanted.
		30	;
		31	NEWTASK: ; Start of new task routine
0015	E5	32	PUSH H ; Save start address of task
0016	2A1A00	33	LHLD MEMPTR ; Fetch old memory pointer
0019	19	34	DAD D ; Add stack length
001A	221A00	35	SHLD MEMPTR ; Save memory pointer
001D	D1	36	POP D ; Restore Task address to DE
		37	;
001E	2B	38	DCX H ;
001F	72	39	MOV M, D ;
0020	2B	40	DCX H ;
0021	73	41	MOV M, E ; PUSH DE onto stack (SP=HL)
		42	;
0022	E5	43	PUSH H ; Save new stack pointer
		44	;
0023	3A1800	45	LDA NUMTSK ;
0026	5F	46	MOV E, A ; Ereg = Number_of_tasks
0027	3C	47	INR A ;
0028	321800	48	STA NUMTSK ; Number_of_tasks = Number_of_Task + 1
		49	;
002B	210400	50	LXI H, TASKQ ;
002E	1600	51	MVI D, 0 ;
0030	19	52	DAD D ;
0031	19	53	DAD D ; HL = TASKQ + 2*Number_of_tasks
0032	D1	54	POP D ; Restore new stack pointer to DE
		55	;
0033	73	56	MOV M, E ;
0034	23	57	INX H ;
0035	72	58	MOV M, D ; Save stack pointer in task queue
		59	;
0036	C9	60	RET ; Return from the <NEWTASK> routine
		61	;
		62	-----
		63	;
		64	Does not save any registers.
		65	;
		66	SXFER: ; Procedure SwapTasks;
0037	210000	67	LXI H, 0 ;
003A	39	68	DAD SP ;
003B	EB	69	XCHG ;
003C	2A0000	70	LHLD TSKPTR ;

LINK LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:18:43 21-APR-88

TASKIN PAGE 2

Multi-tasking kernal for use by the LINK utility

LOC	OBJ	LINE	SOURCE STATEMENT
003F	73	71	MOV M,E ;
0040	23	72	INX H ;
0041	72	73	MOV M,D ; Save Stack pointer of previous task;
0042	23	74	INX H ;
0043	3A1800	D 75	LDA NUMTSK ;
0046	47	76	MOV B,A ;
0047	3A1900	D 77	LDA TASKN ;
004A	B8	78	CMP B ;
004B	C25300	C 79	JNZ MORE ; If Task_Number > Number_of_Tasks then
		80	;
004E	210400	D 81	LXI H,TASKQ ;
0051	3E00	82	MVI A,0 ; Task_Number := 0;
		83	MORE: ;
0053	3C	84	INR A ;
0054	321900	D 85	STA TASKN ; Task_Number := Task_Number + 1;
0057	220000	D 86	SHLD TSKPTR ;
005A	5E	87	MOV E,M ;
005B	23	88	INX H ;
005C	56	89	MOV D,M ; Restore stack pointer of the next task
005D	EB	90	XCHG ;
005E	F9	91	SPHL ;
005F	C9	92	RET ; EndProc
		93	;
		94	;
		95	;
		96	SXFER: ; Procedure SaveTaskSwap;
0060	F5	97	PUSH PSW ;
0061	C5	98	PUSH B ;
0062	D5	99	PUSH D ; SaveRegisters;
0063	E5	100	PUSH H ;
0064	CD3700	C 101	CALL XFER ; SwapTasks;
0067	E1	102	POP H ;
0068	D1	103	POP D ; RestoreRegisters;
0069	C1	104	POP B ;
006A	F1	105	POP PSW ;
006B	C9	106	RET ; EndProc;
		107	;
		108	;
		109	;
		110	Data area.
		111	;
		112	DSEG ; Start of the data segment
		113	;
0000		114	TSKPTR: DS 2 ; Task queue pointer.
0002		115	DS 2 ; These two byte will die
0004		116	TASKQ: DS 20 ; Allow a maximum of 10 tasks.
0018		117	NUMTSK: DS 1 ; Active task counter
0019		118	TASKN: DS 1 ; Current active task number
001A		119	MEMPTR: DS 2 ; Free memory pointer
		120	;
		121	END ; End of <LINKPROC.A85>

0 error(s) detected

Public symbols

NEWTAS SXFER TASKIN XFER

User symbols

MEMPTR 001A D MORE 0053 C NEWTAS 0015 C NUMTSK 0018 D SXFER 0060 C TASKIN 0000 C
 TASKN 0019 D
 TASKQ 0004 D TSKPTR 0000 D XFER 0037 C

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$Title('Buffer management routines for use in the LINK utility')
		2	\$PageLength(75)
		3	;
		4	NAME BUFFER
		5	;
		6	-----;
		7	;
		8	Buffer routines: LINKBUF.A85
		9	;
		10	-----;
		11	;
1F40		12	IBUFL EQU 8000 ; Input buffer length
07D0		13	LOWLIM EQU 2000 ; Low limit for input buffer
1F2C		14	UPLIM EQU IBUFL-20 ; There must always be 20 spare characters.
		15	;
0FA0		16	OBUFL EQU 4000 ; Output buffer length
		17	;
		18	-----
		19	;
		20	\$INCLUDE(COMEQU.A85)
		21	;
		22	-----
		23	;
		24	Logical equates.
		25	;
0000		26	FALSE EQU 0 ; Logical FALSE.
00FF		27	TRUE EQU (NOT FALSE) AND OFFH ; Logical TRUE.
		28	;
		29	Storage size Equates (In terms of bytes).
		30	;
0002		31	WORD EQU 2 ; 1 Word = 2 Bytes
0001		32	BYTE EQU 1 ; 1 Byte = 1 Byte.
0400		33	K EQU 1024 ; 1kbytes = 1024 bytes
		34	;
		35	;
		36	-----
		37	;
		38	PUBLIC BUFINIT
		39	PUBLIC IBUFW, IBUFR
		40	PUBLIC OBUFW, OBUFR
		41	PUBLIC MSGWRT
		42	;
		43	-----
		44	;
		45	EXTRN XFER, SXFER
		46	EXTRN DCMF
		47	EXTRN POS
		48	EXTRN PRINT
		49	;
		50	-----
		51	;
		52	CSEG ; Start of the code segment
		53	;
		54	BUFINIT: ; Procedure Buffer_initialise;
		55	Begin
0000 210000 D		56	LXI H,IBUF ;
0003 22401F D		57	SHLD IBUFIP ; IBUF_Input_pointer := Input_buffer;
0006 22421F D		58	SHLD IBUFOP ; IBUF_Output_pointer := Input_buffer;
0009 3E00		59	MVI A,FALSE ;
000B 32461F D		60	STA LOCKED ; Locked := False;
000E 210000		61	LXI H,0000H ;
0011 22441F D		62	SHLD CHARS ; Character_in_buffer := 0;
		63	;
0014 21471F D		64	LXI H,OBUF ;
0017 22E72E D		65	SHLD OBUFIP ; OBUF_Input_pointer := Output_buffer
001A 22E92E D		66	SHLD OBUFOP ; OBUF_Output_pointer := Output_buffer
		67	;
001D C9		68	RET ; End;
		69	;
		70	-----

LINK LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:18:51 21-APR-88

BUFFER PAGE 2

Buffer management routines for use in the LINK utility

LOC	OBJ	LINE	SOURCE STATEMENT
		71 ;	
		72 ;	Input buffer write.
		73 ;	
		74 IBUFW:	; Procedure IBUF_write(Character:A_register);
001E	4F	75	MOV C,A ; begin
		76 IBW05:	
001F	3A461F	D 77	LDA LOCKED ;
0022	FEFF	78	CPI TRUE ;
0024	C24300	C 79	JNZ IBW20 ; If Locked then
		80	Begin
0027	2A441F	D 81	LHLD CHARS ;
002A	11D007	82	LXI D,LOWLIM ;
002D	CD0000	E 83	CALL DCMP ; If Character < lower_limit then
0030	D24000	C 84	JNC IBW15 ;
		85	Begin
0033	110001	C 86	LXI D,CLOST ;
0036	0600	87	MVI B,0 ;
0038	CD6C00	C 88	CALL MSGWRT ; Message_write(Chars lost);
		89	
003B	3E00	90	MVI A,FALSE ;
003D	32461F	D 91	STA LOCKED ; Locked := False;
		92	
		93 IBW15:	End;
0040	C36B00	C 94	JMP IBWRET ; End
		95 ;	
		96 ;	
		97 ;	
		98 IBW20:	Else
		99	Begin
0043	2A441F	D 100	LHLD CHARS ;
0046	112C1F	101	LXI D,UPLIM ;
0049	CD0000	E 102	CALL DCMP ; If Characters > Upper_limit then
004C	DA5700	C 103	JC IBW30 ;
		104	
004F	3EFF	105	MVI A,TRUE ;
0051	32461F	D 106	STA LOCKED ; Locked := True
0054	C36B00	C 107	JMP IBW35 ;
		108	
		109 IBW30:	Else
		110	Begin
0057	2A401F	D 111	LHLD IBUFIP ;
005A	71	112	MOV M,C ;
005B	23	113	INX H ;
005C	70	114	MOV M,B ;
005D	2B	115	DCX H ;
005E	CD7C00	C 116	CALL IPTRI ; IBUF(Input_pointer) := Ch;
0061	22401F	D 117	SHLD IBUFIP ; Inc(Input_pointer);
0064	2A441F	D 118	LHLD CHARS ;
0067	23	119	INX H ; Characters := Characters+1;
0068	22441F	D 120	SHLD CHARS ;
		121	End;
		122 IBW35:	End;
		123	
		124 IBWRET:	
006B	C9	125	RET ; EndProc;
		126 ;	
		127 ;	
		128 ;	
		129 MSGWRT:	; Procedure MessageWrite(Ptr:Word(DE));
006C	1A	130	LDAX D ;
006D	CD0000	E 131	CALL POS ; PositionCursor;
0070	1A	132	LDAX D ;
0071	47	133	MOV B,A ; StringLength := Memory(Ptr)
		134	
		135	For I := 1 to StringLength;
		136 MWRT05:	Begin
0072	13	137	INX D ; Ptr := Ptr + 1;
0073	1A	138	LDAX D ;
0074	CD0000	E 139	CALL PRINT ; Print(Message(Ptr));
0077	05	140	DCR B ;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:18:51 21-APR-88

BUFFER PAGE 3

Buffer management routines for use in the LINK utility

LOC	OBJ	LINE	SOURCE STATEMENT
0078	C27200	C 141	JNZ MWRT05 ; End;
		142	;
007B	C9	143	RET ; EndProc;
		144 ;	
		145 ;	
		146 ;	
		147 IPTRI:	; Function InputBufferPointerIncrement;
		148	; (Pointer:Word(HL)) : Word(HL);
007C	D5	149	PUSH D ;
007D	23	150	INX H ;
007E	23	151	INX H ; Pointer := Pointer + 2;
007F	11401F	D 152	LXI D,IBUF+IBUFL ;
0082	CD400	C 153	CALL CMPDH ; Pointer > EndOfBuffer then
0085	C28B00	C 154	JNZ IPTROK ;
0088	210000	D 155	LXI H,IBUF ; Pointer := StartOfBuffer;
		156 IPTROK:	;
008B	D1	157	POP D ;
008C	C9	158	RET ; EndProc;
		159 ;	
		160 ;	
		161 ;	
		162 IBUFR:	; Function Input_buffer_read : Character;
		163	; Begin
		164	; Repeat
008D	CD0000	E 165	CALL XFER ; Transfer;
0090	2A441F	D 166	LHLD CHARS ;
0093	7D	167	MOV A,L ;
0094	B4	168	ORA H ;
0095	CA8D00	C 169	JZ IBUFR ; Until Character_in_buffer <> 0
		170	;
0098	2B	171	DCX H ; Characters_in_buffer :=
0099	22441F	D 172	SHLD CHARS ; Characters_in_buffer - 1;
		173	;
009C	2A421F	D 174	LHLD IBUFOP ;
009F	4E	175	MOV C,M ; Creg := Input_buffer(IBUF_pointer);
00A0	23	176	INX H ; IBUF_pointer := IBUF_pointer + 1;
00A1	46	177	MOV B,M ; Breg := Input_buffer(IBUF_pointer);
00A2	2B	178	DCX H ; IBUF_pointer := IBUF_pointer - 1;
00A3	CD7C00	C 179	CALL IPTRI ;
00A6	22421F	D 180	SHLD IBUFOP ; IBUF_pointer := Inc(IBUF_pointer);
00A9	79	181	MOV A,C ;
00AA	C9	182	RET ; End;
		183 ;	
		184 ;	
		185 ;	
		186 OBUFV:	; OutputBufferWrite(Ch:Char(A));
00AB	F5	187	PUSH PSW ;
		188 OBUFV5:	; Repeat
00AC	CD0000	E 189	CALL SXFER ; SaveSwapTasks;
00AF	2AE92E	D 190	LHLD OBUFOP ;
00B2	EB	191	XCHG ;
00B3	2AE72E	D 192	LHLD OBUFIP ;
00B6	CDE400	C 193	CALL OPTRI ;
00B9	CD400	C 194	CALL CMPDH ;
00BC	CAAC00	C 195	JZ OBUFV5 ; Until Inc(OutBufInPtr) <> OutBufOutPtr;
		196	;
00BF	EB	197	XCHG ;
00C0	2AE72E	D 198	LHLD OBUFIP ; OutBufInPtr := OutBufInPtr + 1;
00C3	F1	199	POP PSW ;
		200	;
00C4	77	201	MOV M,A ; OutBuf(OutBufInPtr) := Ch;
00C5	EB	202	XCHG ;
00C6	22E72E	D 203	SHLD OBUFIP ;
00C9	C9	204	RET ; EndProc;
		205 ;	
		206 ;	
		207 ;	
		208 OBUFR:	; Function OutputBufferRead : Char;
		209	; Repeat
00CA	CD0000	E 210	CALL XFER ; SwapTasks;

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:18:51 21-APR-88

BUFFER PAGE 4

Buffer management routines for use in the LINK utility

LOC	OBJ	LINE	SOURCE STATEMENT
00CD	2AE72E	D 211	LHLD OBUFIP ;
00D0	EB	212	XCHG ;
00D1	2AE92E	D 213	LHLD OBUFOP ;
00D4	CDF400	C 214	CALL CMPDH ;
00D7	CACA00	C 215	JZ OBUFR ; Until OutBufInPtr <> outBufOutPtr;
		216	;
00DA	7E	217	MOV A,M ; Ch := OutBuf(OutBufOutPtr);
00DB	F5	218	PUSH PSW ;
00DC	CDE400	C 219	CALL OPTRI ; OutBufPtrInc;
00DF	22E92E	D 220	SHLD OBUFOP ;
00E2	F1	221	POP PSW ; Return(Ch);
		222	;
00E3	C9	223	RET ; EndProc;
		224 ;	
		225 ;	
		226 ;	
		227	OPTRI: ; Function OutputBufferPointerIncrement;
		228	; (Pointer:Word(HL)) : Word(HL);
00E4	D5	229	PUSH D ;
00E5	23	230	INX H ; Pointer := Pointer + 1;
00E6	11E72E	D 231	LXI D,OBUF+OBUFL ;
00E9	CDF400	C 232	CALL CMPDH ; Pointer > EndOfBuffer then
00EC	C2F200	C 233	JNZ OPTROK ;
00EF	21471F	D 234	LXI H,OBUF ; Pointer := StartOfBuffer;
		235	OPTROK: ;
00F2	D1	236	POP D ;
00F3	C9	237	RET ; EndProc;
		238 ;	
		239 ;	
		240 ;	
		241	CMPDH: ; Function CompareDH or CompareHD : Boolean;
		242	CMPHD: ;
00F4	7C	243	MOV A,H ;
00F5	BA	244	CMP D ; If Reg.H = Reg.D and
00F6	C0	245	RNZ ;
00F7	7D	246	MOV A,L ;
00F8	BB	247	CMP E ; Reg.L = Reg.E then
		248	;
		249	; Return(TRUE) else Return(FALSE);
		250	;
00F9	C9	251	RET ; EndProc
		252 ;	
		253 ;	
		254 ;	
		255	CMPBH: ; Function CompareBH or CompareHB : Boolean;
		256	CMPHB: ;
00FA	7C	257	MOV A,H ;
00FB	B8	258	CMP B ; If Reg.H = Reg.B and
00FC	C0	259	RNZ ;
00FD	7D	260	MOV A,L ;
00FE	B9	261	CMP C ; Reg.L = Reg.C then
		262	;
		263	; Return(TRUE) else Return(FALSE);
		264	;
00FF	C9	265	RET ; EndProc
		266 ;	
		267 ;	
		268 ;	
0100	113C4368 61726163 74657273 204C4F53 543E	269	CLOSTL: DB CLOSTL, '<Characters LOST>'
0011		270	CLOSTL EQU \$-(CLOSTL+1)
		271 ;	
		272 ;	
		273 ;	
		274	DSEG ; Start of the data segment
		275 ;	
0000		276	IBUF: DS IBUFL ; Input buffer

LINK LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:18:51 21-APR-88 BUFFER PAGE 5
Buffer management routines for use in the LINK utility

LOC	OBJ	LINE	SOURCE STATEMENT
1F40		277	IBUFIP: DS WORD ; Input buffer input pointer
1F42		278	IBUFOP: DS WORD ; Input buffer output pointer
		279	;
1F44		280	CHARS: DS WORD ; Character counter
1F46		281	LOCKED: DS BYTE ; Buffers locked flag
		282	;
1F47		283	OBUF: DS OBUFL ; Output buffer
2EE7		284	OBUFIP: DS WORD ; Output buffer input pointer
2EE9		285	OBUFOP: DS WORD ; Output buffer output pointer
		286	;
		287	-----
		288	;
		289	END ; End of <LINKBUF.A85>

0 error(s) detected

External symbols

DCMP	POS	PRINT	SXFER	XFER

Public symbols

BUFINI	IBUFR	IBUFW	MSGWRT	OBUFR	OBUFW

User symbols

BUFINI 0000 C	BYTE 0001 A	CHARS 1F44 D	CLOST 0100 C	CLOSTL 0011 A	CMPBH 00FA C	CMPDH 00F4 C
CMPBH 00FA C	CMPDH 00F4 C	DCMP 0001 E	FALSE 0000 A	IBUF 0000 D	IBUFIP 1F40 D	IBUFL 1F40 A
IBUFOP 1F42 D	IBUFR 008D C	IBUFW 001E C	IBW05 001F C	IBW15 0040 C	IBW20 0043 C	IBW30 0057 C
IBW35 006B C	IBWRET 006B C	IPTRI 007C C	IPTR0K 008B C	K 0400 A	LOCKED 1F46 D	LOWLIM 07D0 A
MSGWRT 006C C	MWRT05 0072 C	OBUF 1F47 D	OBUFIP 2EE7 D	OBUFL 0FA0 A	OBUFOP 2EE9 D	OBUFR 00CA C
OBUFW 00AB C	OBUFW5 00AC C	OPTRI 00E4 C	OPTROK 00F2 C	POS 0003 E	PRINT 0004 E	SXFER 0000 E
TRUE 00FF A	UPLIM 1F2C A	WORD 0002 A	XFER 0002 E			

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$Title('Screen routines for the LINK utility')
		2	\$Pagelength(75)
		3	;
		4	NAME SCREEN
		5	;
		6	-----;
		7	Screen routines: ;
		8	-----;
		9	;
		10	\$INCLUDE(COMEQU.A85)
		= 11	;
		= 12	-----
		= 13	;
		= 14	Logical equates.
		= 15	;
0000		= 16	FALSE EQU 0 ; Logical FALSE.
00FF		= 17	TRUE EQU (NOT FALSE) AND OFFH ; Logical TRUE.
		= 18	;
		= 19	Storage size Equates (In terms of bytes).
		= 20	;
0002		= 21	WORD EQU 2 ; 1 Word = 2 Bytes
0001		= 22	BYTE EQU 1 ; 1 Byte = 1 Byte.
0400		= 23	K EQU 1024 ; 1kbytes = 1024 bytes
		= 24	;
		25	\$INCLUDE(VT52.A85)
		= 26	NAME VT52
		= 27	;
		= 28	-----
		= 29	;
		= 30	PUBLIC HOMECL, CLRSC, GOTOC, CLRCL, REVLF, OFFSET
		= 31	PUBLIC UPCMD, DNCMD, LFCMD, RTCMD
		= 32	PUBLIC MINX, MINY, MAXX, MAXY
		= 33	;
		= 34	-----
		= 35	;
		= 36	VT52 ESC characters
		= 37	;
0048		= 38	HOMECL EQU 'H' ; Home cursor command
004A		= 39	CLRSC EQU 'J' ; Clear screen command
0059		= 40	GOTOC EQU 'Y' ; Direct cursor position command
004B		= 41	CLRCL EQU 'K' ; Clear line command
0049		= 42	REVLF EQU 'I' ; Reverse line feed
0020		= 43	OFFSET EQU 20H ; Direct cursor position offset
		= 44	;
0041		= 45	UPCMD EQU 'A' ; Up cursor command
0042		= 46	DNCMD EQU 'B' ; Down cursor command
0044		= 47	LFCMD EQU 'D' ; Left cursor command
0043		= 48	RTCMD EQU 'C' ; Right cursor command
		= 49	;
0000		= 50	MINX EQU 0 ; Min x value (Col)
004F		= 51	MAXX EQU 79 ; Max x value (Col)
0000		= 52	MINY EQU 0 ; Min y value (Row)
0017		= 53	MAXY EQU 23 ; Max y value (Row)
		= 54	;
		= 55	END
		56	\$INCLUDE(ASCII.A85)
		= 57	NAME ASCII
		= 58	-----
		= 59	;
		= 60	PUBLIC NULL, EOT, BELL, BS, TAB, LF, CR, FF
		= 61	PUBLIC CTRLS, CTRLQ, CTRLZ, ESC, SPACE, QUOTE, DEL
		= 62	;
		= 63	-----
		= 64	ASCII constants.
		= 65	;
0000		= 66	NULL EQU 000H ; Null
0004		= 67	EOT EQU 004H ; End of text marker.
0007		= 68	BELL EQU 007H ; Bell.
0008		= 69	BS EQU 008H ; Back space.
0009		= 70	TAB EQU 009H ; Horiz Tab.

UCT 8080/8085 CROSS ASSEMBLER, V1.0
Screen routines for the LINK utility

14:19:06 21-APR-88

SCREEN PAGE 2

LOC	OBJ	LINE	SOURCE STATEMENT
000A		= 71	LF EQU 00AH ; Line feed.
000D		= 72	CR EQU 00DH ; Carriage return.
000C		= 73	FF EQU 00CH ; Form Feed.
0013		= 74	CTRLS EQU 013H ; Contorl 'S'
0011		= 75	CTRLQ EQU 011H ; Control 'Q'
001A		= 76	CTRLZ EQU 01AH ; Control 'Z'
001B		= 77	ESC EQU 01BH ; ESC
0020		= 78	SPACE EQU 020H ; ASCII Space.
0027		= 79	QUOTE EQU 027H ; Single quote (')
007F		= 80	DEL EQU 07FH ; ASCII Delete character.
		= 81 ;	
		= 82 ;	END
		83 ;	
		84 ;	-----
		85 ;	
		86	PUBLIC SCNINIT
		87	PUBLIC GOTO, HOME, CLRS, PRTMSG
		88	PUBLIC PRINT, NMPRT, POS
		89	PUBLIC STOP, LINE, COL, MODE
		90 ;	
		91 ;	-----
		92 ;	
		93	EXTRN OUTMSG, CO, COTEST ; IMPORT Console_test
		94	EXTRN XFER ; IMPORT Transfer
		95	EXTRN OBUFW ; IMPORT OutBufferWrite
		96 ;	
		97 ;	-----
		98 ;	
		99	CSEG ; Start of the code segment;
		100	;
		101	SCNINIT: ; Procedure ScreenInit;
		102	LXI H,STARTM ;
0000	21CA00	C 103	CALL OUTMSG ; OutMessage(StartMessage)
0003	CD0000	E 104	LXI H,HEADER ;
0006	21BF00	C 105	CALL OUTMSG ; OutMessage(HeaderMessage)
0009	CD0000	E 106	MVI A,FALSE ;
000C	3E00	107	STA STOP ; Stop := False;
000E	320300	D 108	MVI A,0 ;
0011	3E00	109	STA LINE ; Line := 0;
0013	320000	D 110	MVI A,4 ;
0016	3E04	111	STA COL ; Col := 4;
0018	320100	D 112	;
		113	RET ; EndProc;
001B	C9	114 ;	
		115 ;	-----
		116 ;	
		117	HOME: ; Procedure HomeCursor;
001C	3E1B	118	MVI A,ESC ;
001E	CD6B00	C 119	CALL PRINT ; Print(ESC);
0021	3E48	120	MVI A,HOMEC ;
0023	CD6B00	C 121	CALL PRINT ; Print(HomeCommand)
		122	;
0026	C9	123	RET ; EndProc;
		124 ;	
		125 ;	-----
		126 ;	
		127	CLRS: ; Procedure ClearScreen;
0027	3E1B	128	MVI A,ESC ;
0029	CD6B00	C 129	CALL PRINT ; Print(Escape);
002C	3E4A	130	MVI A,CLRSC ;
002E	CD6B00	C 131	CALL PRINT ; Print(ClearScreen)
		132	;
0031	C9	133	RET ; EndProc;
		134 ;	
		135 ;	-----
		136 ;	
		137	GOTO: ; Procedure Goto(Row,Col:Byte);
0032	F5	138	PUSH PSW ;
0033	3E1B	139	MVI A,ESC ;
0035	CD6B00	C 140	CALL PRINT ; Print(Escape);

UCT 8080/8085 CROSS ASSEMBLER, V1.0
Screen routines for the LINK utility

14:19:06 21-APR-88

SCREEN PAGE 3

LOC	OBJ	LINE	SOURCE STATEMENT
0038	3E59	141	MVI A,GOTOC ;
003A	CD6B00	C 142	CALL PRINT ; Print(GotoCommand);
003D	F1	143	POP PSW ;
003E	C620	144	ADI OFFSET ;
0040	CD6B00	C 145	CALL PRINT ; Print(Row+OFFSET);
0043	78	146	MOV A,B ;
0044	C620	147	ADI OFFSET ;
0046	CD6B00	C 148	CALL PRINT ; Print(Col+OFFSET);
		149	;
0049	C9	150	RET ; EndProc;
		151 ;	
		152 ;	
		153 ;	
		154	NMPRT: ; Procedure PrintNumber(Val:BYTE)
004A	F5	155	PUSH PSW ;
004B	E6F0	156	ANI 11110000B ;
004D	0F	157	RRC ;
004E	0F	158	RRC ;
004F	0F	159	RRC ;
0050	0F	160	RRC ; MSNibble := Val AND 1111000B SHR 4;
0051	CD6100	C 161	CALL CONNIB ; Ch := ConvertNibble(MSNibble);
0054	CD6B00	C 162	CALL PRINT ; Print(Ch);
0057	F1	163	POP PSW ;
0058	E60F	164	ANI 00001111B ; LSNibble := Val AND 00001111B;
005A	CD6100	C 165	CALL CONNIB ; Ch := ConvertNibble(LSNibble);
005D	CD6B00	C 166	CALL PRINT ; Print(Ch);
		167	;
0060	C9	168	RET ; EndProc;
		169 ;	
		170 ;	
		171 ;	
		172	CONNIB: ; Function ConvertNibble(Nib:BYTE):Char;
0061	FE0A	173	CPI 9+1 ;
0063	FA6800	C 174	JM CON05 ; If Nib > 9 then
0066	C607	175	ADI 7 ; Nib := Nib + 7;
0068	C630	176	CON05: ADI '0' ; Nib := Nib + '0';
006A	C9	177	RET ; EndProc;
		178 ;	
		179 ;	
		180 ;	
		181	PRINT: ; Procedure Print(Ch:Char);
006B	C5	182	PUSH B ;
006C	D5	183	PUSH D ;
006D	E5	184	PUSH H ;
006E	CD0000	E 185	CALL OBUFW ; OutputBufferWrite(Ch);
0071	E1	186	POP H ;
0072	D1	187	POP D ;
0073	C1	188	POP B ;
0074	C9	189	RET ; EndProc;
		190 ;	
		191 ;	
		192 ;	
		193	POS: ; Procedure Position(CharCount:Byte);
0075	D5	194	PUSH D ;
0076	E5	195	PUSH H ;
0077	C5	196	PUSH B ;
		197	;
0078	4F	198	MOV C,A ;
0079	3A0100	D 199	LDA COL ;
007C	57	200	MOV D,A ;
007D	81	201	ADD C ;
007E	FE50	202	CPI MAXX+1 ;
0080	FAA300	C 203	JM COLOK ; If CharCount+COL > MAXX then
		204	Begin
0083	3A0000	D 205	LDA LINE ;
0086	0600	206	MVI B,0 ;
0088	CD3200	C 207	CALL GOTO ; Goto(LINE, 0)
008B	3A0000	D 208	LDA LINE ;
008E	FE15	209	CPI MAXY-2 ;
0090	CA9500	C 210	JZ BOTTOM ; If LINE <> MAXY-2 then

UCT 8080/8085 CROSS ASSEMBLER, V1.0
Screen routines for the LINK utility

14:19:06 21-APR-88

SCREEN PAGE 4

LOC	OBJ	LINE	SOURCE STATEMENT
0093	C603	211	ADI 3 ; LINE := LINE + 3;
		212	BOTTOM: ;
0095	320000 D	213	STA LINE ;
0098	21BC00 C	214	LXI H, SCROLL ; PrintMessage(ScrollMessage)
009B	CDB200 C	215	CALL PRTMSG ;
009E	3E04	216	MVI A, 4 ;
00A0	81	217	ADD C ;
00A1	1604	218	MVI D, 4 ; COL := 4;
		219	COLOK: ; End;
00A3	320100 D	220	STA COL ;
00A6	3A0000 D	221	LDA LINE ;
00A9	C1	222	POP B ;
00AA	80	223	ADD B ;
00AB	42	224	MOV B, D ;
00AC	CD3200 C	225	CALL GOTO ; Goto(Line, Col)
		226	;
00AF	E1	227	POP H ;
00B0	D1	228	POP D ;
00B1	C9	229	RET ; EndProc;
		230	;
		231	-----
		232	;
		233	PRTMSG: ; Procedure PrintMessage(Ptr:Address);
		234	;
00B2	7E	235	MOV A, M ; While memory(Ptr) <> 0 do
00B3	B7	236	ORA A ; Begin
00B4	C8	237	RZ ; Ch := memory(Ptr);
00B5	CD6B00 C	238	CALL PRINT ; Print(Ch);
00B8	23	239	INX H ; Ptr := Ptr + 1;
00B9	C3B200 C	240	JMP PRTMSG ; End;
		241	;
		242	;
		243	-----
		244	;
00BC	0A0A0A	245	SCROLL: DB LF, LF, LF
00BF	54583A0D	246	HEADER: DB 'TX:', CR, LF
	0A		
00C4	52583A0D	247	DB 'RX:', CR, LF
	0A		
00C9	00	248	DB 0
		249	;
00CA	1B481B4A	250	STARTM: DB ESC, HOME, ESC, CLRSC, 0
	00		
		251	;
		252	-----
		253	;
		254	DSEG ; Start of data segment
		255	;
0000		256	LINE: DS BYTE ; LINE variable
0001		257	COL: DS BYTE ; COLUMN variable
0002		258	MODE: DS BYTE ; MODE flag
0003		259	STOP: DS BYTE ; STOP flag
		260	;
		261	-----
		262	;
		263	END ; End of <LINKSCN.A85>

0 error(s) detected

External symbols

CO	COTEST	OBUPW	OUTMSG	XPER

Public symbols

CLRS	COL	GOTO	HOME	LINE	MODE	NMPRT	POS	PRINT	PRTMSG	SCNINI	STOP

UCT 8080/8085 CROSS ASSEMBLER, V1.0
Screen routines for the LINK utility

14:19:06 21-APR-88

SCREEN PAGE 5

LOC	OBJ	LINE	SOURCE STATEMENT
User symbols			
BELL	0007 A	BOTTOM 0095 C	BS 0008 A
CO	0003 E	COL 0001 D	COLOK 00A3 C
CTRLQ	0011 A	CTRLS 0013 A	CTRLZ 001A A
FALSE	0000 A	FF 000C A	GOTO 0032 C
K	0400 A	LF 000A A	LPCMD 0044 A
MINY	0000 A	MODE 0002 D	NMPRT 004A C
POS	0075 C	PRINT 006B C	PRTMSG 00B2 C
SCROLL	00BC C	SPACE 0020 A	STARTM 00CA C
WORD	0002 A	XFER 0004 E	
			BYTE 0001 A
			CON05 0068 C
			CONNIB 0061 C
			DEL 007F A
			GOTOC 0059 A
			HEADER 00BF C
			LINE 0000 D
			MAXX 004F A
			NULL 0000 A
			OBUPW 0000 E
			QUOTE 0027 A
			REVLf 0049 A
			STOP 0003 D
			TAB 0009 A
			TRUE 00FF A
			UPCMD 0041 A
			CLRLC 004B A
			CLRS 0027 C
			CLRSC 004A A
			COTEST 0001 E
			CR 000D A
			EOT 0004 A
			ESC 001B A
			HOME 001C C
			HOMEc 0048 A
			MAXY 0017 A
			MINX 0000 A
			OUTMSG 0002 E
			RTCND 0043 A
			SCNINI 0000 C


```

LOC  OBJ      LINE      SOURCE STATEMENT
                                1 $Title('Keyboard task of the LINK utility')
                                2 $Pagelength(75)
                                3 ;
                                4      NAME      KBDTASK
                                5 ;
                                6 -----;
                                7 ;                      Keyboard task:
                                8 -----;
                                9 ;
0000      10 $INCLUDE(ASCII.A85)          ; Include ASCII equates
                                11 ;      NAME      ASCII
                                12 -----;
                                13 ;
                                14 ;      PUBLIC  NULL, EOT, BELL, BS, TAB, LF, CR, FF
                                15 ;      PUBLIC  CTRLS, CTRLQ, CTRLZ, ESC, SPACE, QUOTE, DEL
                                16 ;
                                17 -----;
                                18 ;      ASCII constants.
                                19 ;
0004      20 NULL      EQU      000H      ; Null
0007      21 EOT      EQU      004H      ; End of text marker.
0008      22 BELL     EQU      007H      ; Bell.
0009      23 BS      EQU      008H      ; Back space.
000A      24 TAB      EQU      009H      ; Horiz Tab.
000D      25 LF      EQU      00AH      ; Line feed.
000C      26 CR      EQU      00DH      ; Carriage return.
0013      27 FF      EQU      00CH      ; Form Feed.
0011      28 CTRLS    EQU      013H      ; Control 'S'
001A      29 CTRLQ    EQU      011H      ; Control 'Q'
001B      30 CTRLZ    EQU      01AH      ; Control 'Z'
0020      31 ESC      EQU      01BH      ; ESC
0027      32 SPACE    EQU      020H      ; ASCII Space.
007F      33 QUOTE    EQU      027H      ; Single quote (')
                                34 DEL      EQU      07FH      ; ASCII Delete character.
                                35 ;
                                36 ;      END
                                37 $INCLUDE(VT52.A85)          ; Include VT52 escape sequences
                                38 ;      NAME      VT52
                                39 ;
                                40 -----;
                                41 ;
                                42 ;      PUBLIC  HOMECL, CLRSC, GOTOC, CLRLC, REVLF, OFFSET
                                43 ;      PUBLIC  UPCMD, DNCMD, LFCMD, RTCMD
                                44 ;      PUBLIC  MINX, MINY, MAXX, MAXY
                                45 ;
                                46 -----;
                                47 ;
                                48 ;      VT52 ESC characters
                                49 ;
0048      50 HOMECL   EQU      'H'      ; Home cursor command
004A      51 CLRSC    EQU      'J'      ; Clear screen command
0059      52 GOTOC    EQU      'Y'      ; Direct cursor position command
004B      53 CLRLC    EQU      'K'      ; Clear line command
0049      54 REVLF    EQU      'I'      ; Reverse line feed
0020      55 OFFSET   EQU      20H      ; Direct cursor position offset
                                56 ;
0041      57 UPCMD    EQU      'A'      ; Up cursor command
0042      58 DNCMD    EQU      'B'      ; Down cursor command
0044      59 LFCMD    EQU      'D'      ; Left cursor command
0043      60 RTCMD    EQU      'C'      ; Right cursor command
                                61 ;
0000      62 MINX     EQU      0        ; Min x value (Col)
004F      63 MAXX     EQU      79       ; Max x value (Col)
0000      64 MINY     EQU      0        ; Min y value (Row)
0017      65 MAXY     EQU      23       ; Max y value (Row)
                                66 ;
                                67 ;      END
                                68 ;
0000      69 ASCII    EQU      0        ; ASCII mode flag value
0001      70 HEX      EQU      1        ; Hexadecimal mode flag value

```

UCT 8080/8085 CROSS ASSEMBLER, V1.0
Keyboard task of the LINK utility

14:19:25 21-APR-88

KBDTAS PAGE 2

LOC	OBJ	LINE	SOURCE STATEMENT
		71 ;	
		72 ;	
		73 ;	
		74	PUBLIC KBDTSK ; Export keyboard task entry point
		75 ;	
		76 ;	
		77 ;	
		78	EXTRN BUFINIT, SCNINIT
		79	EXTRN STOP, MODE
		80	EXTRN CITEST, CI, PRINT, GETUC
		81	EXTRN XFER
		82	EXTRN OUTMSG
		83	EXTRN EDIT
		84 ;	
		85 ;	
		86 ;	
		87	CSEG ; Start of the code segment
		88 ;	
		89	KBDTSK: ; Repeat
0000	CD0000	E 90	CALL XFER ; Transfer_tasks
0003	CD0000	E 91	CALL CITEST ;
0006	CA0000	C 92	JZ KBDTSK ; Until keypressed
		93	;
0009	CD0000	E 94	CALL GETUC ; A := Get_uppercase
000C	FE1B	E 95	CPI ESC ; Character = ESC ?
000E	C21800	C 96	JNZ NOTESC ;
		97 ;	
		98 ;	
		99 ;	
		100 ;	Character = ESC, exit with message.
		101 ;	
		102	
0011	216500	C 103	LXI H,STOPM ;
0014	CD0000	E 104	CALL OUTMSG ; OutMessage('Stopped')
		105	;
0017	76	106	HLT ; Return to the monitor
		107 ;	
		108 ;	
		109 ;	
		110	NOTESC: ;
0018	FE20	111	CPI SPACE ; Space ?
001A	C22700	C 112	JNZ NOTSP ;
		113	;
001D	3A0000	E 114	LDA STOP ; Yes - then toggle the stop flag
0020	2F	115	CMA ;
0021	320000	E 116	STA STOP ;
0024	C30000	C 117	JMP KBDTSK ;
		118 ;	
		119 ;	
		120 ;	
		121	NOTSP: ; Change to hex mode ?
0027	FE48	122	CPI 'H' ;
0029	C23400	C 123	JNZ NOTH ;
		124	
002C	3E01	125	MVI A,HEX ; Yes - then set the mode flag = HEX
002E	320000	E 126	STA MODE ;
0031	C30000	C 127	JMP KBDTSK ;
		128 ;	
		129 ;	
		130 ;	
		131	NOTH: ;
0034	FE41	132	CPI 'A' ; Change to ASCII mode ?
0036	C24100	C 133	JNZ NOTA ;
		134	;
0039	3E00	135	MVI A,ASCII ; Yes - then set the mode flag = ASCII
003B	320000	E 136	STA MODE ;
003E	C30000	C 137	JMP KBDTSK ;
		138 ;	
		139 ;	
		140 ;	

LINK LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0
Keyboard task of the LINK utility

14:19:25 21-APR-88

KBDTAS PAGE 3

LOC	OBJ	LINE	SOURCE STATEMENT
		141	NOTA: ;
0041	FE46	142	CPI 'F' ; Flush buffers ?
0043	C24F00	143	JNZ NOTF ;
		144	;
0046	CD0000	145	CALL BUFINIT ; Yes - then re-initialise buffers and
0049	CD0000	146	CALL SCNINIT ; clear the screen.
004C	C30000	147	JMP KBDTSK ;
		148	;
		149	-----
		150	;
		151	NOTF: ;
004F	FE45	152	CPI 'E' ; Enter edit mode ?
0051	C25D00	153	JNZ NOTE ;
		154	;
0054	CD0000	155	CALL EDIT ; Yes then call the edit routine
0057	CD0000	156	CALL SCNINIT ; and complete, clear the screen.
005A	C30000	157	JMP KBDTSK ;
		158	;
		159	-----
		160	;
		161	NOTE: ; The letter typed is not a valid command
		162	; letter, therefore issue a beep.
005D	3E07	163	MVI A,BELL ;
005F	CD0000	164	CALL PRINT ;
0062	C30000	165	JMP KBDTSK ;
		166	;
		167	-----
		168	;
		169	STOPM:
0065	1B481B4A	170	DB ESC, HOMEC, ESC, CLRSC, 'Stopped', 0
	53746F70		
	70656400		
		171	;
		172	END ; End of <LINKKBD.A85>

0 error(s) detected

External symbols

BUFINI	CI	CITEST	EDIT	GETUC	MODE	OUTMSG	PRINT	SCNINI	STOP	XFER

Public symbols

KBDTSK

User symbols

ASCII	0000	A	BELL	0007	A	BS	0008	A	BUFINI	000A	E	CI	0006	E	CITEST	0009	E	CLRLC	004B	A
CLRSC	004A	A	CR	000D	A	CTRLQ	0011	A	CTRLS	0013	A	CTRLZ	001A	A	DEL	007F	A	DNCMD	0042	A
EDIT	0003	E	EOT	0004	A	ESC	001B	A	FF	000C	A	GETUC	0000	E	GOTOC	0059	A	HEX	0001	A
HOMEC	0048	A	KBDTSK	0000	C	LF	000A	A	LPCMD	0044	A	MAXX	004F	A	MAXY	0017	A	MINX	0000	A
MINY	0000	A	MODE	0002	E	NOTA	0041	C	NOTE	005D	C	NOTESC	0018	C	NOTF	004F	C	NOTH	0034	C
NOTSP	0027	C	NULL	0000	A	OFFSET	0020	A	OUTMSG	0005	E	PRINT	0008	E	QUOTE	0027	A	REVLf	0049	A
RTCMD	0043	A	SCNINI	0004	E	SPACE	0020	A	STOP	0001	E	STOPM	0065	C	TAB	0009	A	UPCMD	0041	A
XFER	0007	E																		

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$Title('Serial polling task for the LINK utility')
		2	\$Pagelength(75)
		3	;
		4	NAME SIOTASK
		5	;
		6	-----;
		7	Serial polling task:;
		8	-----;
		9	;
		10	PUBLIC SIOTSK ; Export task entry point
		11	;
		12	-----
		13	;
		14	EXTRN UOTEST, U1TEST ; USART test routines
		15	EXTRN UI0, UI1 ; USART input routines
		16	EXTRN IBUFW ; Input buffer write routine
		17	EXTRN XFER ; Task swapping routine
		18	;
		19	-----
		20	;
		21	CSEG ; Start of Code segment
		22	;
		23	SIOTSK: ; Task SIO;
		24	Repeat
0000	CD0000	E 25	CALL UOTEST ;
0003	CA0E00	C 26	JZ UOCLR ; If U0_Test then
		27	Begin
0006	CD0000	E 28	CALL UI0 ; Ch := U0_Input;
0009	0600	29	MVI B,0 ;
000B	CD0000	E 30	CALL IBUFW ; InputBufferWrite(Ch,0)
		31	UOCLR: ; End;
000E	CD0000	E 32	CALL U1TEST ;
0011	CA1C00	C 33	JZ U1CLR ; If U1_Test then
		34	Begin
0014	CD0000	E 35	CALL UI1 ; Ch := U1_Input;
0017	0601	36	MVI B,1 ;
0019	CD0000	E 37	CALL IBUFW ; InputBufferWrite(Ch,1)
		38	U1CLR: ; End;
		39	;
001C	CD0000	E 40	CALL XFER ; SwapTasks;
		41	;
001F	C30000	C 42	JMP SIOTSK ; Forever;
		43	EndTask;
		44	;
		45	END ; End of <LINKSIO.A85>

0 error(s) detected

External symbols

IBUFW UOTEST U1TEST UI0 UI1 XFER

Public symbols

SIOTSK

User symbols

IBUFW	0001 E	SIOTSK	0000 C	UOCLR	000E C	UOTEST	0005 E	U1CLR	001C C	U1TEST	0004 E	UI0	0000 E
UI1	0003 E	XFER	0002 E										

```

LOC   OBJ   LINE   SOURCE STATEMENT
      1 $Title('VDU output task for use with the LINK utility')
      2 $PageLength(75)
      3 ;
      4         NAME      VDUTASK
      5 ;
      6 ;-----;
      7 ;                               VDU output task:
      8 ;-----;
      9 ;
     10 $INCLUDE(COMEQU.A85)
= 11 ;
= 12 ;-----;
= 13 ;
= 14 ;                               Logical equates.
= 15 ;
0000   = 16 FALSE EQU      0                ; Logical FALSE.
00FF   = 17 TRUE  EQU      (NOT FALSE) AND OFFH ; Logical TRUE.
      18 ;
      19 ;                               Storage size Equates ( In terms of bytes ).
      20 ;
0002   = 21 WORD  EQU      2                ; 1 Word = 2 Bytes
0001   = 22 BYTE  EQU      1                ; 1 Byte = 1 Byte.
0400   = 23 K     EQU      1024            ; 1kbytes = 1024 bytes
      24 ;
      25 ;
      26 ;-----;
      27 ;
      28         PUBLIC VDUTSK, VDUINIT      ; EXPORT init routine and antry point
      29 ;
      30 ;-----;
      31 ;
      32         EXTRN  STOP                  ; Address of STOP flag
      33         EXTRN  COTEST, CO           ; Monitor routines
      34         EXTRN  OBUFR                ; Output buffer read
      35         EXTRN  XFER                ; Task wrapping routine
      36 ;
      37 ;-----;
      38 ;
      39         CSEG                        ; Start of the code segment
      40 ;
      41 ;-----;
      42 ;
      43 VDUINIT:                          ; Procedure VDU_Initialise;
0000   C9      44         RET                ; EndProc;
      45 ;
      46 ;-----;
      47 ;
      48 VDUTSK:                          ; Task VDU_output;
      49 ;
      50 ;                               Repeat
      51 ;                               Repeat
0001   CD0000   E  52         CALL  XFER                ; SwapTasks;
0004   3A0000   E  53         LDA    STOP                ;
0007   FEFF     C  54         CPI    TRUE                ;
0009   CA0100   C  55         JZ     VDUTSK                ; Until (NOT STOP) AND
      56 ;
000C   CD0000   E  57         CALL  COTEST                ;
000F   CA0100   C  58         JZ     VDUTSK                ; (ConsoleOutputTest);
      59 ;
0012   CD0000   E  60         CALL  OBUFR                ; Ch := OutputBufferRead;
0015   CD0000   E  61         CALL  CO                  ; ConsoleOutput(Ch);
0018   C30100   C  62         JMP    VDUTSK                ; Forever
      63 ;                               EndTask;
      64 ;
      65 ;-----;
      66 ;
      67         END                                ; End of <LINKVDU.A85>

```

0 error(s) detected

LINK LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:19:37 21-APR-88
VDU output task for use with the LINK utility

VDUTAS PAGE 2

LOC	OBJ	LINE	SOURCE STATEMENT

External symbols

CO	COTEST	OBUFR	STOP	XFER

Public symbols

VDUINI	VDUTSK

User symbols

BYTE	0001 A	CO	0003 E	COTEST	0002 E	FALSE	0000 A	K	0400 A	OBUFR	0000 E	STOP	0001 E
TRUE	00FF A	VDUINI	0000 C	VDUTSK	0001 C	WORD	0002 A	XFER	0004 E				

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$Title('ASCII lookup table editing routine for the LINK utility')
		2	\$Pagelength(75)
		3	;
		4	NAME EDIT
		5	;
		6	-----;
		7	Character editor
		8	-----;
		9	;
		10	\$INCLUDE(COMEQU.A85) ; Hardware and other common equates
		= 11	;
		= 12	-----;
		= 13	;
		= 14	Logical equates.
		= 15	;
0000		= 16	FALSE EQU 0 ; Logical FALSE.
00FF		= 17	TRUE EQU (NOT FALSE) AND OFFH ; Logical TRUE.
		= 18	;
		= 19	Storage size Equates (In terms of bytes).
		= 20	;
0002		= 21	WORD EQU 2 ; 1 Word = 2 Bytes
0001		= 22	BYTE EQU 1 ; 1 Byte = 1 Byte.
0400		= 23	K EQU 1024 ; 1kbytes = 1024 bytes
		= 24	;
		25	\$INCLUDE(VT52.A85) ; VT52 control sequence equates
		= 26	NAME VT52
		= 27	;
		= 28	-----;
		= 29	;
		= 30	PUBLIC HOMECL, CLRSC, GOTOC, CLRCL, REVLF, OFFSET
		= 31	PUBLIC UPCMD, DNCMD, LFCMD, RTCMD
		= 32	PUBLIC MINX, MINY, MAXX, MAXY
		= 33	;
		= 34	-----;
		= 35	;
		= 36	VT52 ESC characters
		= 37	;
0048		= 38	HOMECL EQU 'H' ; Home cursor command
004A		= 39	CLRSC EQU 'J' ; Clear screen command
0059		= 40	GOTOC EQU 'Y' ; Direct cursor position command
004B		= 41	CLRCL EQU 'K' ; Clear line command
0049		= 42	REVLF EQU 'I' ; Reverse line feed
0020		= 43	OFFSET EQU 20H ; Direct cursor position offset
		= 44	;
0041		= 45	UPCMD EQU 'A' ; Up cursor command
0042		= 46	DNCMD EQU 'B' ; Down cursor command
0044		= 47	LFCMD EQU 'D' ; Left cursor command
0043		= 48	RTCMD EQU 'C' ; Right cursor command
		= 49	;
0000		= 50	MINX EQU 0 ; Min x value (Col)
004F		= 51	MAXX EQU 79 ; Max x value (Col)
0000		= 52	MINY EQU 0 ; Min y value (Row)
0017		= 53	MAXY EQU 23 ; Max y value (Row)
		= 54	;
		= 55	END
		56	\$INCLUDE(ASCII.A85) ; ASCII character set equates
		= 57	NAME ASCII
		= 58	-----;
		= 59	;
		= 60	PUBLIC NULL, EOT, BELL, BS, TAB, LF, CR, FF
		= 61	PUBLIC CTRLS, CTRLQ, CTRLZ, ESC, SPACE, QUOTE, DEL
		= 62	;
		= 63	-----;
		= 64	ASCII constants.
		= 65	;
0000		= 66	NULL EQU 000H ; Null
0004		= 67	EOT EQU 004H ; End of text marker.
0007		= 68	BELL EQU 007H ; Bell.
0008		= 69	BS EQU 008H ; Back space.
0009		= 70	TAB EQU 009H ; Horiz Tab.

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:19:47 21-APR-88

EDIT

PAGE 2

ASCII lookup table editing routine for the LINK utility

```

LOC   OBJ       LINE      SOURCE STATEMENT
000A      = 71 LF      EQU    00AH      ; Line feed.
000D      = 72 CR      EQU    00DH      ; Carriage return.
000C      = 73 FF      EQU    00CH      ; Form Feed.
0013      = 74 CTRLS   EQU    013H      ; Control 'S'
0011      = 75 CTRLQ   EQU    011H      ; Control 'Q'
001A      = 76 CTRLZ   EQU    01AH      ; Control 'Z'
001B      = 77 ESC     EQU    01BH      ; ESC
0020      = 78 SPACE   EQU    020H      ; ASCII Space.
0027      = 79 QUOTE   EQU    027H      ; Single quote (')
007F      = 80 DEL     EQU    07FH      ; ASCII Delete character.
      = 81 ;
      = 82 ;      END
      = 83 ;
      = 84 ;-----
      = 85 ;
      = 86      PUBLIC EDIT      ; EXPORT EDIT routine
      = 87 ;
      = 88 ;-----
      = 89 ;
      = 90      EXTRN  OUTMSG, NMOUT ; IMPORT message and number output routines
      = 91      EXTRN  CI, CO, GETUC ; IMPORT character I/O routines
      = 92      EXTRN  TABLE      ; IMPORT character definition table
      = 93      EXTRN  ASCBIN      ; IMPORT ASCII to binary conversion routine
      = 94      EXTRN  CITEST      ; IMPORT Terminal status routine
      = 95 ;
007F      = 96 NCHARS EQU    07FH      ; Characters_in_table = 07FH
0006      = 97 ENTLEN EQU    006H      ; Character_length  = 006H
      = 98 ;
      = 99 ;-----
      = 100 ;
      = 101      CSEG      ; Start of code segment
      = 102 ;
      = 103 ;-----
      = 104 ;
      = 105 EDIT:      ; Procedure LookupTableEdit;
0000      = 106      CALL  MHOME      ; MonitorHomeCursor;
0003      = 107      CALL  MCLRS      ; MonitorClearToEndOfScreen;
0006      = 108      LXI   H,EDMSG1    ;
0009      = 109      CALL  OUTMSG      ; OutMessage(EditMessage1);
000C      = 110      CALL  EDINIT      ; EditorInit;
      = 111 EDLOOP:      ; Repeat
000F      = 112      LXI   H,EDMSG2    ;
0012      = 113      CALL  OUTMSG      ; OutMessage(EditMessage2);
0015      = 114      CALL  GETHEX      ; Ch := GetHexValue;
0018      = 115      JZ    EXIT        ; If ExitFlag then ExitRoutine;
      = 116 ;
001B      = 117      CALL  CHANGE      ; UpdateASCIILookupEntry;
001E      = 118      JMP   EDLOOP      ; Forever;
      = 119 EXIT:      ;
0021      = 120      RET      ; EndProc;
      = 121 ;
      = 122 ;-----
      = 123 ;
      = 124 GETHEX:      ; Function GetHexadecimal : BYTE;
      = 125      Repeat
0022      = 126      CALL  GETUC      ; Ch := GetUpperCase;
0025      = 127      MOV   B,A        ;
0026      = 128      CPI   ESC        ;
0028      = 129      JZ    GHEXIT      ; If Ch <> Escape then
      = 130 ;
002B      = 131      CALL  ASCBIN      ; Val := ASCIItoBinary(Ch);
002E      = 132      JNC  GETHEX      ; Until Ch in ['0'..'9', 'A'..'F']
      = 133 ;
0031      = 134      RLC      ;
0032      = 135      RLC      ;
0033      = 136      RLC      ;
0034      = 137      RLC      ;
0035      = 138      MOV   C,A        ; MSNibble := Val SHL 4;
      = 139 ;
0036      = 140      MOV   A,B        ;

```


ASCII lookup table editing routine for the LINK utility

LOC	OBJ	LINE	SOURCE STATEMENT
0037	CD0000	E 141	CALL CO ; ConsoleOutput(Ch);
		142	;
		143	GH05: ; Repeat
003A	CD0000	E 144	CALL GETUC ; Ch := GetUpperCase;
003D	47	145	MOV B,A ;
003E	FE1B	146	CPI ESC ; If Ch <> Escape then
0040	CA5500	C 147	JZ GHEXIT ;
		148	;
0043	CD0000	E 149	CALL ASCBIN ; Val := ASCIItoBinary(Ch);
0046	D23A00	C 150	JNC GH05 ; Until Ch in ['0'..'9', 'A'..'F'];
		151	;
0049	81	152	ADD C ;
004A	E67F	153	ANI 01111111B ; Val := (Val + MSNibble) and 01111111B;
004C	4F	154	MOV C,A ;
004D	78	155	MOV A,B ;
004E	CD0000	E 156	CALL CO ; ConsoleOutput(Ch);
0051	3E01	157	MVI A,1 ;
0053	B7	158	ORA A ;
0054	79	159	MOV A,C ; Return(Val);
		160	GHEXIT: ;
0055	C9	161	RET ; EndProc;
		162	;
		163	;
		164	;
		165	CHANGE: ; Procedure UpdateASCIILookupEntry(Ch:Char);
0056	320000	D 166	STA CH ;
0059	219E01	C 167	LXI H,EDMSG3 ;
005C	CD0000	E 168	CALL OUTMSG ; OutputMessage(EditMessage3);
005F	3A0000	D 169	LDA CH ;
0062	CDDA00	C 170	CALL GETADR ; Adr := GetAddress(Ch);
0065	CDE700	C 171	CALL CHROUT ; OutputCurrentString(Adr);
0068	21B301	C 172	LXI H,EDMSG4 ;
006B	CD0000	E 173	CALL OUTMSG ; OutputMessage(EditMessage4);
006E	CD7800	C 174	CALL ENTER ; ReadNewString;
0071	3A0000	D 175	LDA CH ;
0074	CDB800	C 176	CALL EDWRT ; UpdateScreen;
		177	;
0077	C9	178	RET ; EndProc;
		179	;
		180	;
		181	;
		182	ENTER: ; Start of the <ENTER> routine.
0078	210100	D 183	LXI H,CHRBUF ;
007B	0600	184	MVI B,0 ; Init counter = 0
		185	ENT05: ;
007D	CD0000	E 186	CALL CI ; Read character
0080	CD0000	E 187	CALL CO ; Echo character
0083	FE0D	188	CPI CR ; Char = <CR> ?
0085	CA9100	C 189	JZ ENTEND ; Yes - then exit
		190	;
0088	77	191	MOV M,A ; Save in buffer
0089	23	192	INX H ; Inc buffer pointer
008A	04	193	INR B ; Inc character counter
008B	78	194	MOV A,B ;
008C	FE05	195	CPI ENTLEN-1 ; Character counter = Length - 1 ?
008E	FA7D00	C 196	JM ENT05 ; No - then loop
		197	;
		198	ENTEND: ;
0091	78	199	MOV A,B ; Fetch new definition length
0092	B7	200	ORA A ;
0093	CAA900	C 201	JZ ENT30 ; If zero then leave old def
		202	;
0096	3A0000	D 203	LDA CH ;
0099	CDDA00	C 204	CALL GETADR ; Get address of character in buffer
009C	70	205	MOV M,B ; Save length in buffer
009D	23	206	INX H ; Inc buffer pointer
		207	;
009E	110100	D 208	LXI D,CHRBUF ; DE = Start address of temp char buffer
		209	ENT25: ;
00A1	1A	210	LDAX D ; Fetch character from temp buffer

ASCII lookup table editing routine for the LINK utility

LOC	OBJ	LINE	SOURCE STATEMENT
00A2	77	211	MOV M,A ; Write to main table
00A3	23	212	INX H ; Inc ptr
00A4	13	213	INX D ; Inc ptr
00A5	05	214	DCR B ; Dec character counter
00A6	C2A100	C 215	JNZ ENT25 ; Ch counter = zero ? - no then loop
		216	ENT30: ;
00A9	C9	217	RET ; Return from the <ENTER> routine
		218	;
		219	-----
		220	;
		221	EDINIT: ; Edit mode initialise
00AA	3E00	222	MVI A,0 ; ch = 0
		223	EDI05: ;
00AC	F5	224	PUSH PSW ;
00AD	CDB800	C 225	CALL EDWRT ; EditWrite(ch)
00B0	F1	226	POP PSW ;
00B1	3C	227	INR A ; Inc ch
00B2	FE80	228	CPI NCHARS+1 ;
00B4	C2AC00	C 229	JNZ EDI05 ; Loop until ch > Nchars
00B7	C9	230	RET ;
		231	;
		232	-----
		233	;
		234	EDWRT: ; Procedure EditWrite(Ch:Char(A));
00B8	4F	235	MOV C,A ;
00B9	E670	236	ANI 01110000B ;
00BB	0F	237	RRC ;
00BC	57	238	MOV D,A ;
00BD	0F	239	RRC ;
00BE	0F	240	RRC ;
00BF	82	241	ADD D ;
00C0	47	242	MOV B,A ; Col := (Ch div 16)*10
00C1	79	243	MOV A,C ;
00C2	E60F	244	ANI 00001111B ;
00C4	C602	245	ADI 2 ; Row := (Ch mod 16) + 2
		246	;
00C6	CD1B01	C 247	CALL MGOTO ; MonitorGoto(Row, Col)
		248	;
00C9	79	249	MOV A,C ;
00CA	CD0000	E 250	CALL NMOUT ; NumberOutput(Ch);
00CD	3E3D	251	MVI A,'=' ;
00CF	CD0000	E 252	CALL CO ; ConsoleOutput('=');
		253	;
00D2	79	254	MOV A,C ;
00D3	CDDA00	C 255	CALL GETADR ; Adr := GetAddress(Ch);
00D6	CDE700	C 256	CALL CHROUT ; OutputCurrentString(Adr);
00D9	C9	257	RET ; EndProc;
		258	;
		259	-----
		260	;
		261	GETADR: ; Function GetAddress(Ch:Char) : WORD;
00DA	6F	262	MOV L,A ;
00DB	2600	263	MVI H,0 ; Reg.HL := Ch;
00DD	29	264	DAD H ; Reg.HL := Reg.HL * 2;
00DE	54	265	MOV D,H ;
00DF	5D	266	MOV E,L ; Reg.DE := Reg.HL;
00E0	29	267	DAD H ; Reg.HL := Reg.HL * 2;
00E1	19	268	DAD D ; Reg.HL := Reg.HL + Reg.DE; {Ch*6}
00E2	110000	E 269	LXI D, TABLE ;
00E5	19	270	DAD D ; Adr := Ch*6+TABLE;
00E6	C9	271	RET ; EndProc;
		272	;
		273	-----
		274	;
		275	CHROUT: ; Procedure OutputCurrentString(Adr:Word);
		276	;
00E7	46	277	MOV B,M ; Len := Memory(Ptr);
00E8	C5	278	PUSH B ;
00E9	23	279	INX H ; Ptr := Ptr + 1;
		280	;

ASCII lookup table editing routine for the LINK utility

LOC	OBJ	LINE	SOURCE STATEMENT
		281	CHRO05: ; For I := 1 to len do
		282	; Begin
00EA	7E	283	MOV A,M ; Ch := Memory(Ptr);
00EB	CD0000	E 284	CALL CO ; ConsoleOutput(Ch);
00EE	23	285	INX H ; Ptr := Ptr + 1;
00EF	05	286	DCR B ;
00F0	C2EA00	C 287	JNZ CHRO05 ; End;
		288	;
00F3	C1	289	POP B ;
00F4	3E05	290	MVI A,ENTLEN-1 ;
00F6	90	291	SUB B ; Spaces := EntryLength - Len;
		292	;
00F7	CA0401	C 293	JZ CHRO15 ; If space <> 0 then
00FA	47	294	MOV B,A ;
		295	; For I := 1 to spaces do
		296	CHRO10: ;
00FB	3E20	297	MVI A, ' ' ;
00FD	CD0000	E 298	CALL CO ; ConsoleOutput(' ');
0100	05	299	DCR B ;
0101	C2FB00	C 300	JNZ CHRO10 ;
		301	;
		302	CHRO15: ;
0104	C9	303	RET ; EndProc;
		304	;
		305	-----
		306	;
		307	MHOME: ; Monitor Home cursor
0105	3E1B	308	MVI A,ESC ;
0107	CD0000	E 309	CALL CO ; Console output ESC character
010A	3E48	310	MVI A,HOMEC ;
010C	CD0000	E 311	CALL CO ; Console output HOME command character
		312	;
010F	C9	313	RET ; Return
		314	;
		315	-----
		316	;
		317	MCLRS: ; Monitor Clear screen
0110	3E1B	318	MVI A,ESC ;
0112	CD0000	E 319	CALL CO ; Console output ESC character
0115	3E4A	320	MVI A,CLRSC ;
0117	CD0000	E 321	CALL CO ; Console output Clear screen command
		322	;
011A	C9	323	RET ; Return
		324	;
		325	-----
		326	;
		327	MGOTO: ; Monitor goto routine
011B	F5	328	PUSH PSW ;
011C	3E1B	329	MVI A,ESC ;
011E	CD0000	E 330	CALL CO ; console output ESC character
0121	3E59	331	MVI A,GOTOC ;
0123	CD0000	E 332	CALL CO ; console output GOTO command character
0126	F1	333	POP PSW ;
0127	C620	334	ADI OFFSET ;
0129	CD0000	E 335	CALL CO ; console output ROW + offset
012C	78	336	MOV A,B ;
012D	C620	337	ADI OFFSET ;
012F	CD0000	E 338	CALL CO ; console output COL + offset
		339	;
0132	C9	340	RET ; Return
		341	;
		342	-----
		343	;
		344	; A = Character number (0..4)
		345	GTPPOS: ; Get character position
0133	C62D	346	ADI PPOS ; Add prompt position
0135	47	347	MOV B,A ;
0136	3E16	348	MVI A,22 ;
0138	CD1B01	C 349	CALL MGOTO ; MonitorGoto(22, CharNum+PPOS)
013B	C9	350	RET ; Return

UCT 8080/8085 CROSS ASSEMBLER, V1.0

14:19:47 21-APR-88

EDIT

PAGE 6

ASCII lookup table editing routine for the LINK utility

LOC	OBJ	LINE	SOURCE STATEMENT
		351 ;	
		352 ;-----	
		353 ;	
013C	0909092A	354	EDMSG1: DB TAB,TAB,TAB,'*** Link character editor ***',0
	2A2A204C		
	696E6B20		
	63686172		
	61637465		
	72206564		
	69746F72		
	202A2A2A		
	00		
015D	1B593420	355	EDMSG2: DB ESC,GOTOC,20+OFFSET,0+OFFSET
0161	456E7465	356	DB 'Enter character to change (2 hex digits - ESC to exit) > '
	72206368		
	61726163		
	74657220		
	746F2063		
	68616E67		
	65202832		
	20686578		
	20646967		
	69747320		
	2D204553		
	43200974		
	6F206578		
	69742920		
	3E20		
019B	1B4A	357	DB ESC, CLRSC
019D	00	358	DB 0
019E	1B593620	359	EDMSG3: DB ESC,GOTOC,22+OFFSET,0+OFFSET
01A2	43757272	360	DB 'Current value = ',0
	656E7420		
	76616C75		
	65203D20		
	00		
01B3	20204E65	361	EDMSG4: DB ' New value? > _____', BS, BS, BS, BS, BS, 0
	77207661		
	6C75653F		
	203E205F		
	5F5F5F5F		
	08080808		
	0800		
		362 ;	
002D		363	PPOS EQU \$-EDMSG3-2
		364 ;	
		365 ;-----	
		366 ;	
		367	DSEG ; Start of the data segment
		368 ;	
0000		369	CH: DS BYTE ; Character to change
0001		370	CHRBUF: DS ENTLEN ; Temp character
		371 ;	
		372 ;-----	
		373 ;	
		374	END ; End of <LINKEDIT.A85>

0 error(s) detected

External symbols

ASCBIN	CI	CITEST	CO	GETUC	NMOUT	OUTMSG	TABLE

Public symbols

EDIT

LINK LISTING

UCT 8080/8085 CROSS ASSEMBLER, V1.0 14:19:47 21-APR-88
ASCII lookup table editing routine for the LINK utility

EDIT PAGE 7

LOC	OBJ	LINE	SOURCE STATEMENT
User symbols			
ASCBIN 0005 E	BELL 0007 A	BS 0008 A	BYTE 0001 A
CHRO05 002A C	CHRO10 00FB C	CHRO15 0104 C	CHROUT 0027 C
CLRSC 004A A	CO 0004 E	CR 000D A	CTRLQ 0011 A
DNCMD 0042 A	EDI05 00AC C	EDINIT 00AA C	EDIT 0000 C
EDMSG3 019E C	EDMSG4 01B3 C	EDWRT 00B8 C	ENT05 007D C
ENTER 0078 C	ENTLEN 0006 A	EOT 0004 A	ESC 001B A
GETADR 00DA C	GETHEX 0022 C	GETUC 0001 E	GH05 003A C
HOMEC 0048 A	K 0400 A	LF 000A A	LFCMD 0044 A
MGOTO 011B C	MHOME 0105 C	MINK 0000 A	MINY 0000 A
OFFSET 0020 A	OUTMSG 0002 E	PPOS 002D A	QUOTE 0027 A
TAB 0009 A	TABLE 0000 E	TRUE 00FF A	UPCMD 0041 A
			WORD 0002 A
			CHANGE 0056 C
			CHRBUF 0001 D
			CITEST 0007 E
			CLRLC 004B A
			CTRLZ 001A A
			DEL 007F A
			EDMSG1 013C C
			EDMSG2 015D C
			ENT25 00A1 C
			ENT30 00A9 C
			ENTEND 0091 C
			FALSE 0000 A
			FF 000C A
			GOTOC 0059 A
			GTPPOS 0133 C
			MAXX 004F A
			MAXY 0017 A
			MCLRS 0110 C
			NMOUT 0006 E
			NULL 0000 A
			RTCMD 0043 A
			SPACE 0020 A

LOC	OBJ	LINE	SOURCE STATEMENT
		1	STitle('SABus monitor interface section');
		2	\$PageLength(75)
		3	;
		4	NAME SABMON
		5	;
		6	-----
		7	;
		8	; This module contains all the equates and routines necessary to allow
		9	; application programs to interface with the SABus monitor.
		10	;
		11	-----
		12	;
		13	PUBLIC CITEST, UOTEST, U1TEST, CI, UI0, UI1, CO, UO0, UO1
		14	PUBLIC BREAK, GETUC, GETECHO, GETWD, ASCBIN, ECHO, PRVAL
		15	PUBLIC NMOUT, OUTMSG, DCMPI, TRANSP, VALDL, GETCM
		16	PUBLIC GETHX, GET1NM, GET2NM, GET3NM
		17	;
		18	PUBLIC COTEST
		19	;
		20	-----
		21	;
F00A		22	CITEST EQU 0F00AH ; Test if char in USART 0
F00D		23	UOTEST EQU 0F00DH ; Test if char in USART 1
F010		24	U1TEST EQU 0F010H ; Test if char in USART 2
F013		25	CI EQU 0F013H ; Input char from USART 0
F016		26	UI0 EQU 0F016H ; Input char from USART 1
F019		27	UI1 EQU 0F019H ; Input char from USART 2
F01C		28	CO EQU 0F01CH ; Output char to USART 0
F01F		29	UO0 EQU 0F01FH ; Output char to USART 1
F022		30	UO1 EQU 0F022H ; Output char to USART 2
F025		31	BREAK EQU 0F025H ; Implement XON/XOFF protocol and sense ESC
F028		32	GETUC EQU 0F028H ; Input char from USART 0 - convert to UC
F02B		33	GETECHO EQU 0F02BH ; Input from console and echo
F02E		34	GETWD EQU 0F02EH ; Input up to four hex digits from console
F031		35	ASCBIN EQU 0F031H ; Convert ASCII to binary checking validity
F034		36	ECHO EQU 0F034H ; Output char to console checking for ESC, CR
F037		37	PRVAL EQU 0F037H ; Print nibble as hex digit
F03A		38	NMOUT EQU 0F03AH ; Print byte as two hex digits
F03D		39	OUTMSG EQU 0F03DH ; Print a string on the console
F043		40	DCMP EQU 0F043H ; Compare HL to DE
F046		41	DCMPI EQU 0F046H ; Compare (HL) to DE
F049		42	TRANSP EQU 0F049H ; Enter transparent communication mode
F04C		43	VALDL EQU 0F04CH ; Test if valid delimiter
F04F		44	GETCM EQU 0F04FH ; Return to monitor command fetch routine
F052		45	GETHX EQU 0F052H ; Fetch 0/1 hex number, error return to monitor
F055		46	GET1NM EQU 0F055H ; Fetch one hex number, error return to monitor
F058		47	GET2NM EQU 0F058H ; Fetch 2 hex numbers, error return to monitor
F05B		48	GET3NM EQU 0F05BH ; Fetch 3 hex numbers, error return to monitor
		49	;
		50	-----
		51	;
0001		52	TXRDY EQU 00000001B ; USART Receiver ready status bit
		53	;
0000		54	CODATA EQU 0 ; Console USART data register
0001		55	COSTAT EQU 1 ; Console USART status register
		56	;
		57	-----
		58	;
		59	CSEG ; Start of CODE segment
		60	;
		61	COTEST: ; Procedure ConsoleOutputTest : Boolean;
		62	;
0000 DB01		63	IN COSTAT ; If ConsoleStatus AND ReceiverReady then
		64	Return(TRUE)
0002 E601		65	ANI TXRDY ; Else
		66	Return(FALSE)
0004 C9		67	RET ; EndProc;
		68	;
		69	-----
		70	;

UCT 8080/8085 CROSS ASSEMBLER, V1.0
SABus monitor interface section

14:24:14 21-APR-88

SABMON PAGE 2

LOC	OBJ	LINE	SOURCE STATEMENT
		71	END ; End of <SABMON.A85>

0 error(s) detected

Public symbols

ASCBIN	BREAK	CI	CITEST	CO	COTEST	DCMP	DCMPI	ECHO	GET1NM	GET2NM	GET3NM	GETCM
GETECH	GETHX	GETUC	GETWD	NMOUT	OUTMSG	PRVAL	TRANSP	UOTEST	U1TEST	UI0	UI1	U00
U01	VALDL											

User symbols

ASCBIN F031 A	BREAK F025 A	CI F013 A	CITEST F00A A	CO F01C A	CODATA 0000 A	COSTAT 0001 A
COTEST 0000 C	DCMP F043 A	DCMPI F046 A	ECHO F034 A	GET1NM F055 A	GET2NM F058 A	GET3NM F05B A
GETCM F04F A	GETECH F02B A	GETHX F052 A	GETUC F028 A	GETWD F02E A	NMOUT F03A A	OUTMSG F03D A
PRVAL F037 A	TRANSP F049 A	TKRDY 0001 A	UOTEST F00D A	U1TEST F010 A	UI0 F016 A	UI1 F019 A
U00 F01F A	U01 F022 A	VALDL F04C A				

APPENDIX K

KIT LISTING

This appendix contains the source listing on the communications program, KIT, as well as the command file used to build it.

.; The command file used to compile, assemble and task build the KIT
.; communication program is as follows:

```
RTL KIT,KIT,KIT=KIT/CO/TA:S 1000/CN:F
.IF <EXSTAT> <> 1 .RTLERR
MAC KIT=KIT
.IF <EXSTAT> <> 1 .MACERR
.OPEN RTLTSK.CMD
.ENABLE DATA
KIT/CP,/ -SP=KIT
@LB:[1,54]COMINTFCE
/
STACK=300
LIBR=RTLLIB:RW
ASG=TI:1:2
ASG=SY:3
TASK=...KIT
PRI = 65
//
.DISABLE DATA
.CLOSE RTLTSK.CMD
TKB @RTLTSK
.IF <EXSTAT> <> 1 .TKBERR
PIP KIT.MAC;* ,KIT.OBJ;* ,RTLTSK.CMD;*/PU/LD
.GOTO STOP
.;
.;-----
.;
.;
.RTLERR:
;
;   There are errors in you RTL source code, please correct and re-compile
;
.GOTO STOP
.MACERR:
;
;   There are errors in you MAC file, please correct and re-compile
;
.GOTO STOP
.TKBERR:
;
;   The TASK builder reported and error, please fix it.
;
.STOP:
.EXIT

.; Where COMINTFCE.CMD contains the following:
.;
.; LB:[1,1]RSXBA1
.; LB [1,1]RTLEXC/LB
.; LB:[1,1]MTSLIB/LB
```


RSX-11 RTL/2 COMPILER - V6.0 - MAY 81 - SPL 22-APR-88 12:40

>KIT,KIT,KIT=KIT

TABLES=S 1000

TABLES=

CSNAME=

CNTROL=F

RELOCATABLE=

OPTION=

TITLE=

IDENT=

PSECT=

ASECT=

CONCORDANCE=

DEBUG=

```

1      %=====
2      %
3      %                SABus KIT INTERFACE PROGRAM                %
4      %
5      %=====
6
7      % Define boolean values %
8
9      LET FALSE    = 0;
10     LET TRUE     = HEX FF;
11
12     % Define ASCII characters %
13
14     LET ETX      = HEX 03; % ASCII end of text character %
15     LET HT       = HEX 09; % ASCII tab character %
16     LET LF       = HEX 0A; % ASCII line feed character %
17     LET CR       = HEX 0D; % ASCII carriage return character %
18     LET ENDFILE  = HEX 1A; % End of file character ^Z (character sent) %
19     LET SP       = HEX 20; % ASCII space character %
20     LET EOF      = HEX 80; % PDP end of file character %
21
22     % Define KIT command character, command = 'KIT <Filename>/<command>' %
23
24     LET SAVCOM    = 'S'; % Save file command, KIT to PDP transfer %
25     LET RECCOM    = 'R'; % Recall file command, PDP to KIT transfer %
26     LET UICCOM    = 'U'; % Send protection UIC %
27     LET NULCOM    = 0 ; % Null command %
28
29     % KIT acknowledge characters %
30
31     LET ACKCHAR   = '@'; % General KIT acknowledge character %
32     LET NOERR     = '0'; % KIT no error code, follows the ack character %
33
34     % Define PDP logical unit numbers for terminal and single file %
35
36     LET TERMINALLUN = 1;
37     LET KEYBOARDLUN = 2;
38     LET PDPFILELUN  = 3;
39
40     % Define PDP file number, for a single file %
41

```

```

42      LET PDPFILENUMBER = 1;
43
44      % KIT RRCEL error number %
45
46      LET      ERRNUM      = 1000;
47
48      % Declare MTSLIB procs used by KIT %
49
50      EXT PROC ( ) BYTE INF, TTIN, HSIN, GPIN;
51      EXT PROC (BYTE) OUTF, TTOUT, HSOUT, GPOUT;
52      EXT PROC (INT) RRCEL;
53      EXT PROC (REF ARRAY BYTE, REF FNBLK) FNAME;
54      EXT PROC (REF FNBLK, INT, INT) INT  FSTRTIN;
55      EXT PROC (REF ARRAY BYTE) TWRT;
56      EXT PROC (REF ARRAY BYTE, REF ARRAY BYTE) INT TREAD;
57      EXT PROC (REF ARRAY INT) FINIT;
58      EXT PROC (INT) OWRT, IWRT;
59      EXT PROC (INT) INT FCLOSE;
60      EXT PROC (REF FNBLK, INT, INT) INT FSTRTOUT;
61      EXT PROC (REF ARRAY BYTE, INT) DBGWRT;
62      EXT PROC ( ) GETMCR;
63      EXT PROC (INT, REF ARRAY BYTE) ASSIGN;
64
65      % Define I/O modes %
66
67      MODE FNBLK (INT DEVN, REF BYTE DEVT,
68                  INT DIRN, REF BYTE DIRT,
69                  INT NAMN, REF BYTE NAMT);
70
71      MODE IOCL (REF ARRAY BYTE BFR, INT N,DV,PTR,MD,TRM);
72
73      % Define I/O status mode %
74
75      MODE IOS (BYTE STATUS,TERMCH, INT LEN);
76
77      % Declare external data block to allow 1 active file %
78
79      EXT DATA FILE01;
80      REF ARRAY INT FDBTABLE;
81      ENDDATA;
82
83      SVC DATA RRSIO; PROC ( ) BYTE IN; PROC (BYTE) OUT; ENDDATA;
84      SVC DATA RRSER; BYTE TERMCH, IOFLAG; ENDDATA;
85      SVC DATA RRERR; LABEL ERL; INT ERN; PROC (INT) ERP; ENDDATA;
86      SVC DATA RRERRX; INT LINENO; BYTE UEFLAG, ERRLUN; INT RSXDSW; ENDDATA;
87      SVC DATA RRCHAN; REF IOCL INCL,OUTCL; ENDDATA;
88
89      % Declare terminal driver local data %
90
91      ENT DATA TTIOLOCAL;
92      IOCL TTINCL := (TTINBUF,0,1,1,0,0);
93      IOCL TTOUTCL:= (TTOUTBUF,0,2,0,0,0);
94      ARRAY(132) BYTE TTINBUF := ( ' '(132));
95      ARRAY(132) BYTE TTOUTBUF:= (LF,' '(131));
96      ENDDATA;
97
98      % Declare file local data %

```

```

99
100 ENT DATA FILEIOLOCAL;
101     BYTE DUMMYBYTE;
102     FNBLK INFNB := (0,DUMMYBYTE,0,DUMMYBYTE,0,DUMMYBYTE);
103     FNBLK OUTFNB := (0,DUMMYBYTE,0,DUMMYBYTE,0,DUMMYBYTE);
104     IOCL INFILECL := (INFBUF,0,0,0,0,0);
105     IOCL OUTFILECL := (OUTFBUF,0,0,0,0,0);
106     ARRAY(512) BYTE INFBUF := (' '(512));
107     ARRAY(512) BYTE OUTFBUF := (' '(512));
108 ENDDATA;
109
110     %=====
111     %                                     %
112     %             Local data area             %
113     %                                     %
114     %=====
115
116 DATA LOCAL;
117     IOS IOSTAT;
118     BYTE COMMAND,CHAR, LASTCHAR;
119     INT LEN,COMLEN,COMPOINT,FILENAMELENGTH;
120     BYTE LASTCOMMAND := FALSE, INLINECOMMAND := FALSE;
121     REF ARRAY BYTE COMMANDLINE := "#SP(80)#";
122     REF ARRAY BYTE PDPFILENAME := "#SP(30)#";
123     REF ARRAY BYTE TERMB := "#CR,LF,EOF#";
124     REF ARRAY BYTE STRING := "#SP(132)#";
125 ENDDATA;
126
127     %=====
128     %                                     %
129     %             START OF MAIN PROGRAM             %
130     %                                     %
131     %=====
132
133 ENT PROC RRJOB ();
134
135     %-----%
136     %     INITIALISE INPUT/OUTPUT     %
137     %-----%
138
139     FINIT (FDBTABLE);
140
141     ASSIGN (PDPFILELUN , "SY:");
142     ASSIGN (TERMINALLUN, "TI:");
143     ASSIGN (KEYBOARDLUN, "TI:");
144
145     %-----%
146     % INIT ERROR ROUTINE TO KITERP %
147     %-----%
148
149     ERP := KITERP;           % Reassign the ERROR handler %
150
151     SWTTIN ();               % Input from the terminal %
152     SWTTOUT();               % Output to the terminal %
153
154     GETFIRSTCOMMAND ();      % Fetch first command %
155     PROCESSCOMMAND ();

```

```

156      WHILE LASTCOMMAND=FALSE DO
157          GETNEXTCOMMAND ();
158          PROCESSCOMMAND ();
159      REP;
160  ENDPROC;
161
162      %=====
163      %
164      % GETFIRSTCOMMAND : This routine fetches a command, either from the %
165      % Command Line, if it is available, or by doing a read after prompt %
166      % if no command is present on the command line. The following %
167      % information is retrieved from the command line: %
168      %
169      %      COMMAND      : Command letter U|R|S %
170      %      PDPFILENAME : The name of the file on the PDP, space padded %
171      %                      and all in upper case. %
172      %
173      %=====
174
175  PROC GETFIRSTCOMMAND ();
176
177      % Move the rest of the command line to input buffer %
178
179      COMPOINT := 1;
180      GETMCR ();
181      IF INCL.N # INCL.PTR THEN
182          COMLEN := TREAD (COMMANDLINE, TERMB);
183          GETFILENAME ();
184          GETCOMMAND ();
185          INLINECOMMAND := TRUE;
186      ELSE
187          COMLEN := GETLINEAFTERPROMPT (COMMANDLINE, "@");
188          WHILE COMLEN=0 AND TERMCH#EOF DO
189              COMLEN := GETLINEAFTERPROMPT (COMMANDLINE, "@");
190          REP;
191          IF TERMCH=EOF THEN
192              COMMAND := NULCOM;
193              LASTCOMMAND := TRUE;
194          ELSE
195              GETFILENAME ();
196              GETCOMMAND ();
197          END;
198      END;
199  ENDPROC;
200
201      %=====
202      %
203      % GETNEXTCOMMAND : This routine performs the same basic function %
204      % as the GETFIRSTCOMMAND routine above, except that the command %
205      % is always fetched using a read after prompt operation. %
206      %
207      %=====
208
209  PROC GETNEXTCOMMAND ();
210      IF LASTCHAR=LF AND INLINECOMMAND#TRUE THEN
211          COMPOINT := 1;
212          COMLEN := GETLINEAFTERPROMPT (COMMANDLINE, "@");

```

```

213      WHILE COMLEN=0 AND TERMCH#EOF DO
214          COMLEN := GETLINEAFTERPROMPT (COMMANDLINE, "@");
215      REP;
216      IF TERMCH=EOF THEN
217          COMMAND := NULCOM;
218          LASTCOMMAND := TRUE;
219      ELSE
220          GETFILENAME ();
221          GETCOMMAND ();
222      END;
223      ELSEIF LASTCHAR=LF AND INLINECOMMAND=TRUE THEN
224          COMMAND := NULCOM;
225          LASTCOMMAND := TRUE;
226      ELSE
227          GETFILENAME ();
228          GETCOMMAND ();
229      END;
230  ENDPROC;
231
232      %=====
233      %
234      % PROCESSCOMMAND : This routine performs the operation indicated by %
235      % the command letter.
236      %
237      %=====
238
239  PROC PROCESSCOMMAND ();
240      IF      COMMAND=SAVCOM THEN SAVEFILE();
241      ELSEIF COMMAND=RECCOM THEN RECALLFILE();
242      ELSEIF COMMAND=UICCOM THEN SENDUIC();
243      ELSEIF COMMAND#NULCOM THEN ERP(1001);
244      END;
245  ENDPROC;
246
247      %=====
248      %
249      % GETFILENAME : This routine extracts the file name from the command %
250      % line and inserts it into the PDP file name buffer. The buffer %
251      % is space filled and all the characters are converted to upper %
252      % case when they are copied
253      %
254      %=====
255
256  PROC GETFILENAME ();
257      INT I, FILENAMEPOINTER := 1;
258
259      % Skip leading spaces on the command line %
260
261      WHILE COMMANDLINE(COMPOINT) = SP OR COMMANDLINE(COMPOINT) = HT DO
262          COMPOINT := COMPOINT + 1;
263      REP;
264
265      % Copy the file name from the file name buffer to the %
266      % file name buffer.
267
268      WHILE COMMANDLINE(COMPOINT) # '/' AND COMPOINT <= COMLEN DO
269          PDPFILENAME(FILENAMEPOINTER) := UPPERCASE(COMMANDLINE(COMPOINT));

```

```

270      COMPOINT:=COMPOINT+1;
271      FILENAMEPOINTER := FILENAMEPOINTER + 1;
272      REP;
273
274      FILENAMELENGTH := FILENAMEPOINTER - 1;
275
276      FOR I := FILENAMEPOINTER TO LENGTH PDPFILENAME DO
277          PDPFILENAME(I) := ' ';
278      REP;
279
280      IF COMPOINT > COMLEN THEN
281          ERP(1000);
282      END;
283
284      COMPOINT := COMPOINT + 1; % Move the command line pointer past the '/' %
285  ENDPROC;
286
287      %=====
288      %
289      % GETCOMMAND : This routine extract the command letter from the
290      % command line.
291      %
292      %=====
293
294  PROC GETCOMMAND ();
295      COMMAND := UPPERCASE(COMMANDLINE(COMPOINT));
296      COMPOINT := COMPOINT + 1;
297      LASTCHAR := COMMANDLINE(COMPOINT);
298      WHILE LASTCHAR#LF AND LASTCHAR#',' DO
299          COMPOINT := COMPOINT + 1;
300          LASTCHAR := COMMANDLINE(COMPOINT);
301      REP;
302      COMPOINT := COMPOINT + 1;
303  ENDPROC;
304
305      %=====
306      %
307      % UPPERCASE : This function convert a single character from lower
308      % case to upper case. If the character passed to the function is
309      % not a lower case character then the original character is
310      % returned.
311      %
312      %=====
313
314  PROC UPPERCASE (BYTE CH) BYTE;
315      IF CH >='a' AND CH <= 'z' THEN
316          CH := BYTE ( CH - 'a' + 'A' );
317      END;
318      RETURN(CH);
319  ENDPROC;
320

```

```

321      %=====
322      %
323      % KITERP : This routine is used to replace the normal RTL/2
324      % recoverable error handler. It checks for the following common
325      % errors:
326      %
327      %      i) No such file
328      %      ii) Invalid file name
329      %
330      %=====
331
332      PROC KITERP (INT ERRORNUMBER);
333          INT LOCALDSW;
334
335          LOCALDSW := RSXDSW;          % Save the DSW in error %
336          SWTTOUT ();                  % Assign terminal output %
337          OUT(ACKCHAR);                % Ack command from KIT %
338
339          FCLOSE(PDPFILENUMBER);       % Close file incase it was open %
340
341          % Comment out the error message, used during testing %
342
343          ***      TWRT("#LF#KIT -- ");          ***
344
345          ***      IF ERRORNUMBER = 999 THEN TWRT("NO FILE NAME GIVEN");          ***
346          ***      ELSEIF ERRORNUMBER = 1000 THEN TWRT("NO SWITCH GIVEN");          ***
347          ***      ELSEIF ERRORNUMBER = 1001 THEN TWRT("INVALID SWITCH");          ***
348          ***      ELSEIF LOCALDSW = -26 THEN TWRT("FILE DOES NOT EXIST");          ***
349          ***      ELSEIF LOCALDSW = -54 THEN TWRT("INVALID FILE NAME");          ***
350
351          ***      ELSE
352          ***          TWRT("ERROR NUMBER ");          ***
353          ***          IWRT(ERRORNUMBER);          ***
354          ***          TWRT(", DSW = ");          ***
355          ***          IWRT(LOCALDSW);          ***
356          ***      END;          ***
357
358          % End of commented out section %
359
360          IF ERRORNUMBER = 999 THEN OUT('1');
361          ELSEIF ERRORNUMBER = 1000 THEN OUT('2');
362          ELSEIF ERRORNUMBER = 1001 THEN OUT('3');
363          ELSEIF LOCALDSW = -26 THEN OUT('4');
364          ELSEIF LOCALDSW = -54 THEN OUT('5');
365          ELSEIF ERRORNUMBER = 1002 THEN OUT('6');
366          ELSE
367              OUT('7');
368          END;
369
370          OUT(LF);
371          GOTO ERL;
372      ENDPROC;

```

```

372      %=====
373      %
374      % RECALLFILE : The routine handles the /R commands. The operation
375      % of the command is as follows:
376      %
377      % i)  A 'No error' status code is sent  "@0"
378      % ii) The file is opened
379      % iii) The contents of the file are sent to the KIT
380      % iv) A EOF mark or Control-Z is sent.
381      %
382      %=====
383
384      PROC RECALLFILE ();
385          STARTINFILE(PDPFILENAME);
386
387          OUT(ACKCHAR);      % Ack command from the KIT %
388          OUT(NOERR);        % Indicate no error %
389
390          LEN := TREAD (STRING, TERMB);
391          WHILE TERMCH # EOF DO
392              SPTWRT (STRING, LEN);
393              OUT(LF);
394          LEN := TREAD (STRING, TERMB);
395          REP;
396          TWRT("#ENDFILE,CR,LF#");
397          CLOSEINPUTFILE();
398      ENDPROC;
399
400      %=====
401      %
402      % RECALLFILE : The routine handles the /S commands. The operation
403      % of the command is as follows:
404      %
405      % i)  A 'No error' status code is sent  "@0"
406      % ii) Open, for output, the specified file.
407      % iii) Read each line of the file, using the read after prompt
408      % routine.
409      % iv) When a EOF mark or Control-Z is received, close the output
410      % file.
411      %
412      %=====
413
414      PROC SAVEFILE ();
415          STARTOUTFILE(PDPFILENAME);
416          SWTTOUT();
417
418          OUT(ACKCHAR);      % Ack command from the KIT %
419          OUT(NOERR);        % Indicate no error %
420          OUT(ETX);          % Send buffer %
421
422          LEN := GETLINEAFTERPROMPT(STRING, "@");
423          WHILE TERMCH # EOF DO
424              SWFILOUT();
425              SPTWRT (STRING, LEN);
426              OUT(LF);
427              SWTTOUT();
428          LEN := GETLINEAFTERPROMPT(STRING, "@");

```



```

429      REP;
430
431      CLOSEOUTPUTFILE();
432  ENDPROC;
433
434      %=====
435      %
436      % SENDUIC : The routine handles the /U commands. The operation
437      % of the command is as follows:
438      %
439      % i)  Get the terminal UIC, using the get terminal characteristics
440      %      system call. This is done in the GETUIC function.
441      % ii) Send the UIC as a series of four ASCII character.
442      %
443      %=====
444
445  PROC SENDUIC ();
446      INT UIC := GETUIC ();
447
448      IF FILENAMELENGTH=0 THEN
449          OUT(LF);
450          OUT(ACKCHAR);
451          OUT(BYTE( ((UIC SRL 12) LAND HEX OF) + 'A' ));
452          OUT(BYTE( ((UIC SRL 8) LAND HEX OF) + 'A' ));
453          OUT(BYTE( ((UIC SRL 4) LAND HEX OF) + 'A' ));
454          OUT(BYTE( ((UIC          ) LAND HEX OF) + 'A' ));
455          OUT(ETX);
456      ELSE
457          IF UIC#(((PDPFILENAME(1) - 'A') SLL 12) +
458              ((PDPFILENAME(2) - 'A') SLL 8) +
459              ((PDPFILENAME(3) - 'A') SLL 4) +
460              ((PDPFILENAME(4) - 'A')          )) THEN
461              ERP(1002);
462      END;
463  END;
464  ENDPROC;
465
466      % File I/O procs %
467
468      %=====
469      %
470      % STARTINFILE : This opens the file specified for input.
471      %
472      %=====
473
474  PROC STARTINFILE(REF ARRAY BYTE INFILENAME) INT;
475      FNAME(INFILENAME, INFNB);
476      SWFILIN ();
477      RETURN(FSTR TIN(INFNB,1,PDPFILELUN));
478  ENDPROC;
479
480      %=====
481      %
482      % STARTOUTFILE : This opens the file specified for output.
483      %
484      %=====
485

```

```

486  PROC STARTOUTFILE (REF ARRAY BYTE OUTFILENAME) INT;
487      FNAME (OUTFILENAME, OUTFNB);
488      SWFILOUT ();
489      RETURN (FSTRTOUT(OUTFNB,1,PDPFILELUN));
490  ENDPROC;
491
492      %=====
493      %
494      % CLOSEINPUTFILE : This closes the currently open input file.
495      %
496      %=====
497
498  PROC CLOSEINPUTFILE() INT;
499      RETURN (FCLOSE (PDPFILENUMBER));
500  ENDPROC;
501
502      %=====
503      %
504      % CLOSEOUTPUTFILE : This closes the currently open output file.
505      %
506      %=====
507
508  PROC CLOSEOUTPUTFILE() INT;
509      RETURN (FCLOSE (PDPFILENUMBER));
510  ENDPROC;
511
512      %=====
513      %
514      % SWFILIN : This switches the input stream to come from the file.
515      %
516      %=====
517
518  PROC SWFILIN ();
519      INCL:=INFILECL;
520      IN:=INF;
521  ENDPROC;
522
523      %=====
524      %
525      % SWFILEOUT : This switches the output stream go to the file.
526      %
527      %=====
528
529  PROC SWFILOUT ();
530      OUTCL:=OUTFILECL;
531      OUT:=OUTF;
532  ENDPROC;
533
534      %=====
535      %
536      % SWTTIN : This switches the input stream to come from the terminal
537      %
538      %=====
539
540  PROC SWTTIN ();
541      INCL := TTINCL;
542      IN := GPIN;

```

```

543     ENDPROC;
544
545     %=====
546     %
547     % SWTTOUT : This switches the output stream to go to the terminal. %
548     %
549     %=====
550
551     PROC SWTTOUT ();
552         OUTCL := TTOUTCL;
553         OUT := GPOUT;
554     ENDPROC;
555
556     %=====
557     %
558     % SPTWRT : This routine outputs the first M characters stored in an %
559     %   array of bytes. %
560     %
561     %=====
562
563     PROC SPTWRT (REF ARRAY BYTE STR, INT LENTH);
564         INT I;
565         FOR I:= 1 TO LENTH DO
566             OUT (STR (I));
567         REP;
568     ENDPROC;
569
570     %=====
571     %
572     % GETUIC : This routine used the GET TASK CHARACTERISTICS to fetch %
573     %   UIC of the KIT task. %
574     %
575     %=====
576
577     PROC GETUIC () INT;
578         INT D17,D16,D15,D14,D13,D12,D11,D10,D7,D6,D5,D4,D3,D2,D1,D0;
579
580         CODE 100,0;
581
582         .MCALL      GTSK$$
583
584         MOV      #*D0, R1      ; Get the offset of D0 from the link cell
585         ADD      R5, R1      ; Make the address absolute
586         GTSK$$ R1      ; Get the task info
587         MOV      $DSW,*RSXDSW/RRERRX(R0);
588         BCC      *GTSKOK      ; C clear - directive succeeded
589
590         *RTL;
591         ERP(1002);
592     GTSKOK:
593         RETURN(D17);
594     ENDPROC;
595

```

```

596      %=====
597      %
598      % GETLINEAFTERPROMPT : This routine is used to fetch commands and
599      % lines from the terminal. The routines uses the read after prompt
600      % QIO to do this.
601      %
602      %=====
603
604      PROC GETLINEAFTERPROMPT (REF ARRAY BYTE TSTRING, PROMPTSTRING) INT;
605          INT LEN, L;
606
607          L := LENGTH(PROMPTSTRING);
608          LEN := LENGTH(TSTRING);
609
610          % Code block used to call QIOWSS Read_After_Prompt %
611          % uses register R3,R4, Returns QIO status in the %
612          % 4 bytes IOSTAT/LOCAL. It uses event flag 32. %
613
614          CODE 68,0;
615
616              .MCALL    QIOWSS
617              .GLOBL    IO.RST,IO.RPR,$DSW
618              .LIST     MEB
619
620              MOV      *TSTRING(5),R4
621              INC      R4                ; POINT TO 1ST BYTE .
622              MOV      *PROMPTSTRING(5),R3
623              INC      R3                ; INC PAST LEN BYTE.
624
625              QIOWSS #IO.RPR,#1,#32.,#46.,#*IOSTAT/LOCAL,,<R4,*LEN(5),,R3,*L(5),#0>
626
627              MOV      $DSW,*RSXDSW/RRERRX(R0)
628              MOVB     *IOSTAT/LOCAL+1,*TERMCH/RRSED(R0)
629              BCC      *OK
630
631              *RTL;
632
633              IF RSXDSW = -10 THEN
634                  TERMCH := EOF;
635              ELSE
636                  RRSEL(ERRNUM);
637              END;
638      OK:
639              IF IOSTAT.STATUS = HEX F6 THEN
640                  TERMCH := EOF;
641              END;
642              RETURN(IOSTAT.LEN);
643      ENDPROC;

```

COMPILED AS SYSTEMS MODULE

USE OF RESOURCES

RESOURCE	MAX	USED

KIT LISTING

IDENTIFIERS (I)	383	164
VARIABLES (I)	383	89
AT BRICK LVL (D)	335	75
ID NAMES (D)	335	158
NAME CHARS (N)	2701	950
GEN. LABELS (G)	335	50
CONSTANT POOL(C)	287	22
STRING POOL (S)	1000	273
ARRAY BOUNDS (A)	47	2
MODE INFO. (G)	670	149

BRICK SIZES (BYTES)

BRICK	DECIMAL	OCTAL
'SPOOL'	276	424
TTIOLO	292	444
FILEIO	1078	2066
LOCAL	26	32
RRJOB	94	136
GETFIR	144	220
GETNEX	148	224
PROCES	68	104
GETFIL	184	270
GETCOM	84	124
UPPERC	56	70
KITERP	172	254
RECALL	114	162
SAVEFI	126	176
SENDUI	220	334
STARTI	46	56
STARTO	46	56
CLOSEI	22	26
CLOSEO	22	26
SWFILI	24	30
SWFILO	24	30
SWTTIN	24	30
SWTTOU	24	30
SPTWRT	54	66
GETUIC	126	176
GETLIN	144	220
'TOTAL'	3638	7066

COMPILATION OK