THE DESIGN AND IMPLEMENTATION OF AN IMAGE PROCESSING SYSTEM

Jeanmary Miketinac

Submitted for M.Sc.(Comp.Sci.) Degree
University of Cape Town
April 1978

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. H. Mitchell, for his interest and help during completion of the thesis.

I would also like to express my thanks to Prof. K.J. MacGregor for introducing me to the field of Image Processing and to both him and Mr. A.A. Jackson for their constant interest, information and stimulating suggestions during the development stages.

Finally, I would like to thank Dr. D.N. Swingler (once of Electrical Engineering, U.C.T., now in engineering at University of Saskatchewan, Canada) for giving me my first insight into Fourier analysis.

Contents

Appendices

1 An Introduction to Image Processing

Image processing can be thought of as any operation carried out on an image once it has been recorded on some medium. The development of a photographic film is a simple case of image processing. The picture has been taken and therefore recorded on the film. The processing steps are the developing of the film and the printing of pictures.

Image processing can be accomplished optically or by digital computation. Each has its advantages and disadvantages. The optical system for example obtains instantaneous results with its set of filters and lenses correctly positioned on an optical bench whereas the digital computer requires a lot more time to accomplish the same result. The digital advantage however is the fact that it is more flexible since lenses and filters can only be fabricated for a limited number of applications. Once the digital system is fully developed, it is also possible for reproduction of analyses whereas this is difficult if not impossible for an optical system. Also, untrained personnel can operate the digital system with a minimum of instruction whereas this is not possible for an optical system.

The four main branches of image processing are image capture, whether by the imaging system (camera) or the processing system (developer), image enhancement which attempts to present an image in a form most useful to the viewer, pattern recognition which detects the presence of a particular object throughout an image, and image display.

The need for an image processing system developed at U.C.T. in 1976 when a number of different departments recognized how useful it could be in their areas of research. An Image Processing Unit was established and work started on developing a viable system in early 1977. One of the problems initially encountered was uncertainty as to the feasibility of this being undertaken by a small research group in a university environment. Most currently available

systems of this type have been designed by large institutions with plenty of staff and computer availability, i.e. VICAR designed by Jet Propulsion Laboratory, and ER-MAN II by IBM. Another factor in the design of the system was the need for a large degree of portability since U.C.T. in the early stages of design was in the process of choosing its next computer system which need not be the same as the current one.

The task to be carried out was analyzed and divided up amongst the three members of the group - K.J. MacGregor, A.A. Jackson and the author. The processing of images was to be carried out by a set of application tasks which would be driven and supported by a set of general purpose I/O and file handling routines. The division of labour required A.A. Jackson to develop pattern recognition application tasks, K.J. MacGregor to develop some image capture and display tasks and the author to develop the support I/O and file handling modules as well as some enhancement tasks.

Once this division of labour had been arrived at, group discussions took place as the embryonic system developed, to coordinate routine interfaces and determine the system specifications. Most of the interfacing was required for the interrelated, multipurpose driver/support modules which call each other as well as service the independent application tasks. Once these had been decided upon, each member independently designed and implemented his area of the system.

It is the object of this thesis to discuss the author's contribution to the image processing system mentioned above. Three of the branches of image processing are touched upon herein: image capture, enhancement and display. Image capture here means 'the making available of an image for further processes to be carried out on it'. This is accomplished by the set of general purpose interrelated I/O and file handling routines for convenience called the NUCLEUS and some tape I/O application tasks. The 'further processes' are carried out by the application routines which are task specific - they perform one operation

only. Therefore to obtain an image in the required form may necessitate the use of several application tasks.

Chapter 2 contains an introduction to digital images and the architecture of the system, the VICAR system being used as a guideline. Chapters 3 to 5 contain more detailed descriptions of the NUCLEUS and the application tasks with the programming specifications being relegated to Appendices.

2 Digital Images and an Overview of the System

2.1 Digital Images

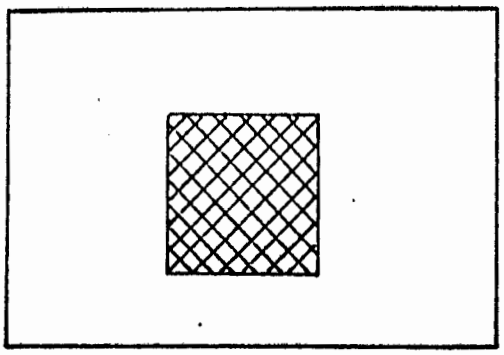
Digital images consist of numbers instead of levels of gray (as in black and white pictures). These numbers actually represent the degree of 'black-white' (intensity) at the given spot. Given a picture as in Figure 1 which is a normal photo of continuous data, it is scanned 'n' times. Each scan becomes one line of digital data. Each scan takes time 't' from beginning to end, and if the line is sampled at intervals of 't/m' there are 'm' digital values (pixels) produced for the scan line. Each pixel is a reflection of the intensity at the sampling position.

Any image can be represented in this format by operating on it with a digitising device to produce the above results. In this respect digital image processing is a tool available to a large variety of disciplines. The medical world is a very good example of an area where this could be used to advantage. X-rays and radio isotope scans are frequently used in this environment and they may not always be of the quality that the analyst would like. Digitising these images and analyzing them to sharpen up certain features is a great asset to the medical world (Hepburn, 1975).

At present the system deals mainly with satellite pictures derived from the NOAA (weather) and LANDSAT (Earth Resources) satellites. These satellites scan the earth and send back the data to tracking stations on earth in digital form. When out of range of a tracking station the satellite is either turned off or if data is required from the flight path it is stored on tape until its possible transmission to a station.

LANDSAT images are produced as described above (ERTS Reference Manual). An area of the earth is scanned, and as it is being done the line is sampled at regular

Digitisation of a picture



a picture of continuous data

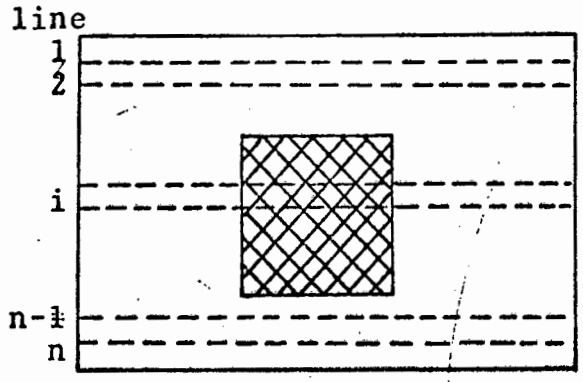


image is scanned 'n' times

line

1	0	0	0				...				0	0	0
2	0	0	0				...				0	0	0
i	0	0	0	...	0	127	...	127	0	...	0	0	0
n-1	0	0	0				...				0	0	0
n	0	0	0				...				0	0	0

each scan line is sampled 'm' times to give 'm' digital values for the line

- Figure 1 -

intervals giving the pixel values. The satellite is more sophisticated than the simple model described above in that it scans six lines simultaneously and in four wavelength bands. These wavelength bands are known as band 4 (0.5-0.6μm), band 5 (0.6-0.7μm), band 6 (0.7-0.8μm) and band 7 (0.8-1.1μm). Due to this added refinement we have 4 images of the same ground scene, all different however because each wavelength band shows up different aspects of the surface. Digital values for LANDSAT images range between 0 and 127 for bands 4 to 6, and 0 to 63 for band 7. Zero represents no intensity present (black) and 127 (or 63) is maximum intensity (white).

2.2 An Introduction to the U.C.T. Digital Image Processing System (DIPS)

Prior to working on the system a study was made of the IBM VICAR system which is available at U.C.T. as a document (COSMIC, 1968). This system consists of a core resident nucleus called VMAST which handles all I/O, data set management and the opening, closing and double buffering of data sets, a non-resident job control manager called VMJC which is loaded into an overlay area, and a set of application tasks also loaded into the overlay area.

Since VICAR is a good working system it was used as a guideline for the new system. However, it was soon decided that it would be more practical to design a new nucleus which would incorporate the same duties as the nucleus and job control manager of the VICAR system. Many of the VICAR nucleus routines were kept by name but changed in context to do what was required. In many cases the calling parameters of the NUCLEUS routines remain identical to those of the VICAR system. DIPS is written in FORTRAN which allows for faster writing, easier debugging and a great deal more mobility than a low level language.

St. Lucia area, Natal Band 7

St. Lucia area, Natal Band 5

1190-07143-4 100 -749 1700-2149 2 FEB 78 UCT-IPU
LSATIN STRECH SKEW

St. Lucia area, Natal Band 4
Caption above has the following meaning:
XXXX-XXXXX-Y tapename-band number,
the next two sets of numbers are the area of the image displayed followed by the date of processing and UCT-IPU.
The bottom line indicates the processing that has taken place on the image (see Appendix A).

Only where it has been absolutely necessary has UNIVAC assembler been used. Such cases for this would be the packing and unpacking of data words which can be done clumsily and expensively in FORTRAN but is much more efficient in assembler as it allows for the dissection of words.

The reasons for a complete redesign are two-fold:

(1) The VICAR nucleus (VMAST) and job control manager (VMJC) are written for the most part in IBM assembler and to produce a workable system would have required a rewrite in any case.

(2) Due to architectural differences in machine software it became necessary to reconstruct the NUCLEUS. Also, VICAR was found wanting in several areas and these have been improved upon.

Each system has been designed to take advantage as much as possible of system peculiarities and because of this each has an advantage over the other in a certain area.

Due to machine software differences it is possible for the VICAR system to maintain a smaller core resident portion than DIPS. VICAR operates by entering VMAST which has a small loader section to load VMJC into an overlay area. VMJC handles all initiation necessary for an application task and upon completion enters a resident routine, LINKUSER. The application task to be loaded is learnt from an input task record and a variable is changed to this name in LINKUSER which then loads the task into the overlay area (VMJC is overwritten). This obviates the need for a large number of searches on task name and an ultimate subroutine call to the required task.

DIPS on the other hand requires that the equivalent of VMAST and VMJC both be core resident. This is necessary because of software differences between IBM and UNIVAC. IBM when linking keeps a table of routine names and start addresses allowing for the dynamic change of routine names

within the program. However, UNIVAC's linking operates on an absolute address system making the changing of the routine name to the correct one prior to call impossible as all addresses are resolved prior to execution. Therefore, DIPS instead of loading the required task upon request must initiate a subroutine call. Upon completion of the task the RETURN statement is going to return control to the caller which if it is not core resident will have been overwritten. Essentially the difference here is that VICAR does a return to the caller's caller allowing the caller to be overwritten whereas this is not possible on the UNIVAC system.

For the same reason VICAR allows the addition of new or amended application tasks into the system without bringing the entire system into the operation. DIPS on the other hand requires the involvement of the entire system in the MAP process since it operates on an absolute addressing system.

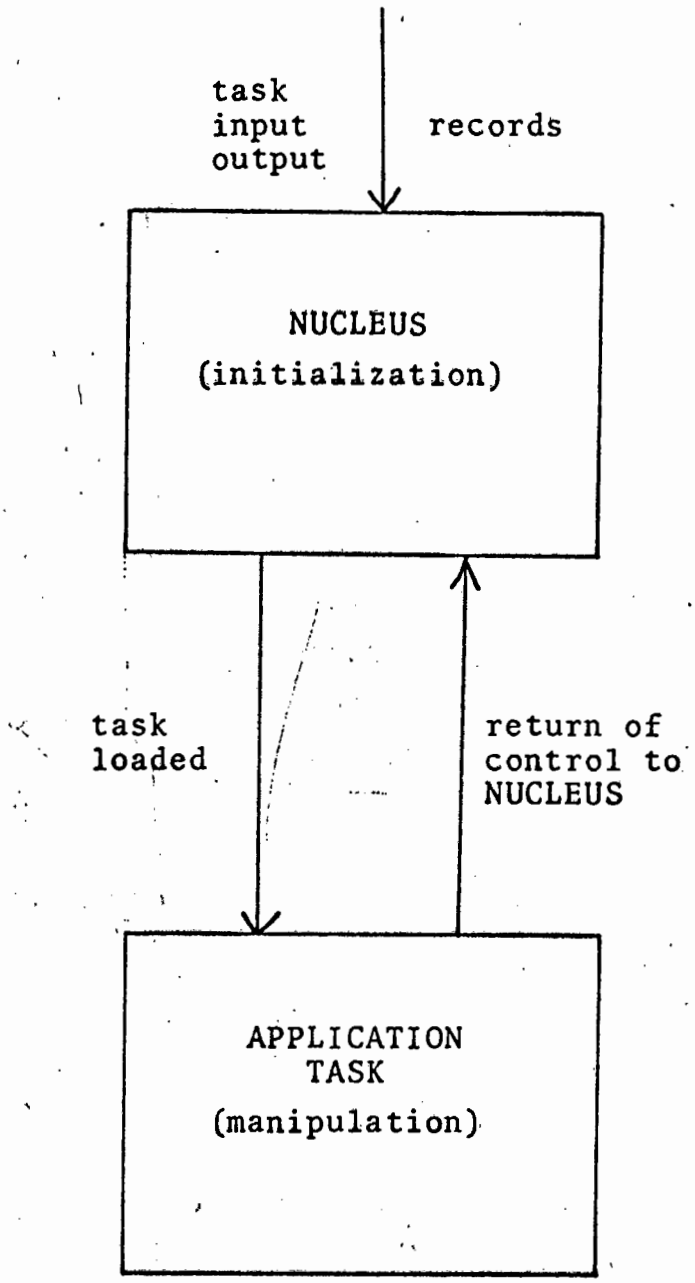
Another IBM-UNIVAC software difference allows the DIPS system to dynamically assign files from within the run. All that the system requires are the filenames as input and they are assigned internally. VICAR on the other hand not having this capability requires that all files be known beforehand and the necessary JCL be set up either by the user or VTRAN which is a VICAR language translator producing the JCL and input required for the VICAR system from a set of VICAR language statements input by the user.

The image processing system that has been designed is a disk oriented system for the analysis and display of images from any source which are in line scan digital form. The system consists of two distinct sections (see Figure 2):

(1) The NUCLEUS:

This section contains all the routines that are required to be core resident for the duration of the

The sections of the System



- Figure 2 -

run. These routines deal with input/output, setting up of files, initiation of the application task, and parameter transfer.

(2) The Application Tasks:
These tasks do the required manipulation of the data for the user. These routines are not core resident but are loaded into an overlay area upon being called (see Figure 3).

It was decided to use a segmented system with overlaying as this drastically cuts down on the core requirements. Since the application tasks are required one at a time, they can easily use the same storage during a run. All routines that are dependent upon one another have been placed in the same segment so each segment is totally independent only requiring the NUCLEUS routines (see Figure 3 for segmentation).

The user communicates with the system via a number of input records which contain all the necessary information about the task to be carried out. There are five types of records for this purpose:

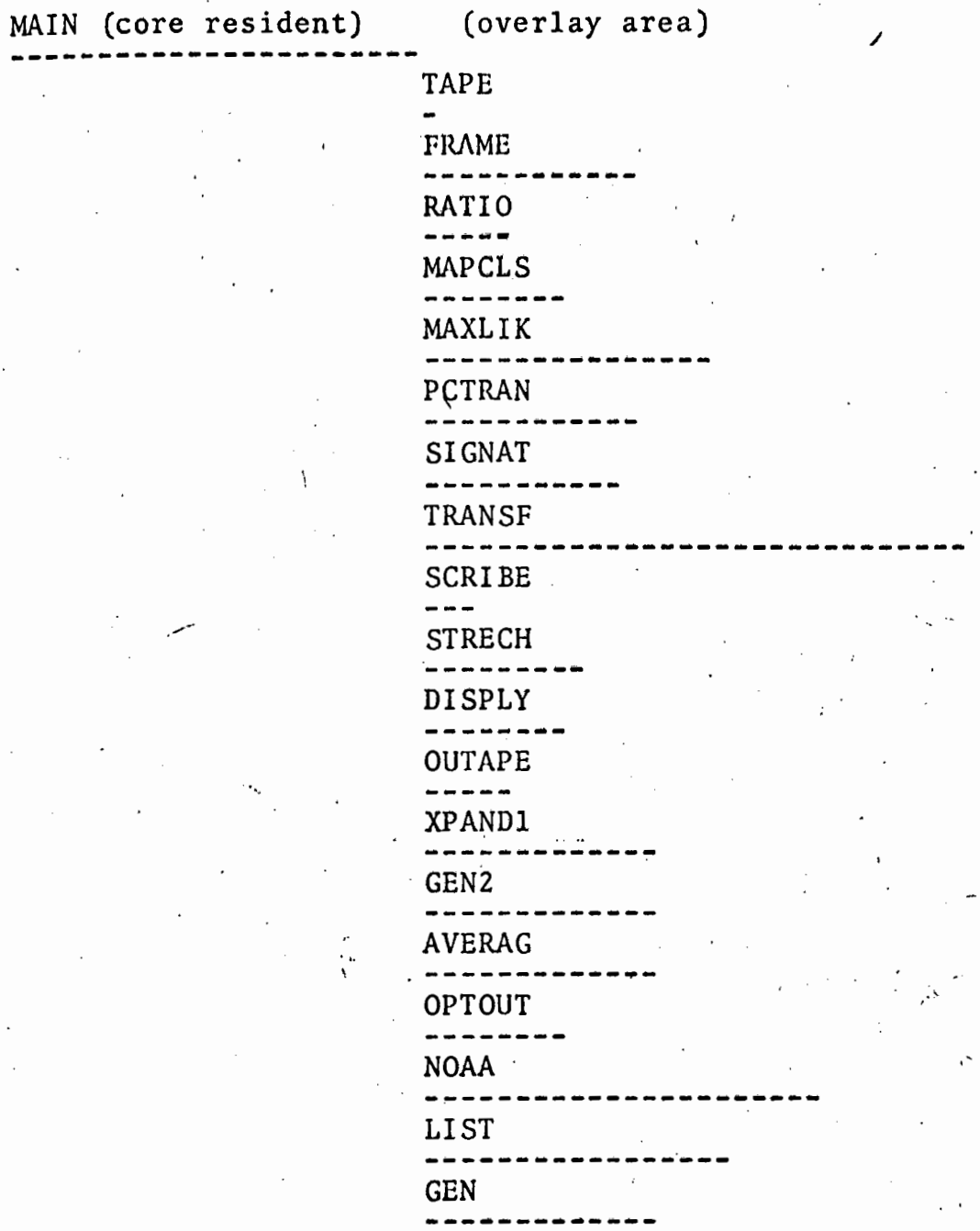
(1) Task record:
This establishes exactly what task is to be carried out, the number of inputs (maximum ten), number of outputs (maximum ten), the starting position and the number of pixels to be processed. The format is as follows:

T AAAAAA NI NO SL SS NL NS

where

T - the symbol indicating this is a task record (col. 1),
AAAAAA - the name of the task to be carried out (col. 7-12),
NI - number of inputs,
NO - number of outputs,
SL - starting line for processing,
SS - starting sample in the lines for processing,

Segmentation of the system with sizes to scale



Scale: one - per 1000 decimal words (rounded up).

- Figure 3 -

NL - number of lines to be processed,
NS - number of samples to be processed.
NI, NO, SL, SS, NL, and NS are all free format.

(2) Input record:
This states the name of the file which is to be used as input for the task (maximum of 10 such records). The format is

I BBBBBBBBBB

where
I - is the symbol indicating this is an input record (col. 1),
BBBBBBBBBB - the filename (col. 13-22 for tape files as given) and (col. 13-18 for disk files).
The number of input records must equal NI from the task record.

(3) Output record:
This states the name of the file which is to be used as output for the task (maximum of 10 such records). The format is

O CCCCCC

where
O - the symbol indicating an output record (col. 1),
CCCCCC - the output file name (col. 13-18).
The number of output records must equal NO from the task record.

(4) Parameter record:
This record contains the input data from the user that the task needs to continue with the job. These parameters are fixed and must always be present for a task or errors will occur. Parameters for all tasks can be found in the user manual compiled by the Image Processing Unit. There are three formats for parameter records:

i) P i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8 i_9 i_{10}

where

P - the symbol indicating a parameter record (col. 1),

i_1-i_{10} - integer parameters, 10 per record, free format.

ii) P r_1 r_2 r_3 r_4 r_5

where

P - the symbol indicating a parameter record,

r_1-r_5 - real parameters, 5 per record, free format.

iii) P $C_1C_2C_3C_4C_5C_6C_7C_8C_9C_{10}$

where

P - the symbol indicating a parameter record,

C_1-C_{10} - character parameter containing 6 characters, 10 per record (packed) starting in column 7.

(5) End of Job:

This record indicates the end-of-job has been reached.

L

where

L - the symbol for end-of-job (col. 1).

The ordering of these records is important. For a given task the task record appears first, then all input records, all output records and finally all parameter records. This is repeated for all tasks to be carried out and at the end comes the end-of-job record.

The new system as seen has five types of records each specifying a different action to the NUCLEUS. The task record and the parameter records were kept from the VICAR system but changed in format. The input and output records are new innovations. They specify specifically the files to be used whereas the VICAR system specified the files through preliminary JCL and via the task card. Also, the records are less regimented. Wherever it has been possible the fields are free format, therefore fewer errors in misplacing will occur. Four of the VICAR records were discarded as extraneous. These were the NOTE, LABEL,

RELABEL and BLANK records. The NOTE record was used for logging messages which is not really needed. The LABEL and RELABEL records dealt with the file labels in that the user could specify his own labels. This can lead to chaos if not controlled hence it was done away with and the labels are strictly controlled by the NUCLEUS and to some extent by the application task. The BLANK record had no use at all and was therefore discarded. The VICAR system also has an end of job indicator, /*, which is still used but has been changed to 'L'.

Another change between the two systems is in the area of parameter transfer. VICAR required that all data input records be read prior to entry to the application task. Due to the large variety of types and non-specific ordering, VMJC never knew what it was going to receive next hence it had to read one record ahead of itself prior to handling the current input. This also required a lot of routines to manipulate all these data records. Since a number of these have now been discarded, DIPS has no need for routines to handle them. Also the parameter records are now read by the application task itself. Since it knows the type and number of parameters it requires, this does away with a host of conversion routines as needed in VMJC and also the necessity of a parameter buffer in VMAST which had to be as large as the greatest number of parameters ever to be input by an application task.

3 The Core Resident Portion of DIPS - The NUCLEUS

3.1 Introduction

The NUCLEUS acts as the central organizer of the system. It contains all the routines needed by the application tasks for input/output, parameter passing and label handling. As well as this the MAIN routine of the NUCLEUS handles the setting up of job control for the assignment of files, initial label processing and global parameter processing prior to entry to the application routine.

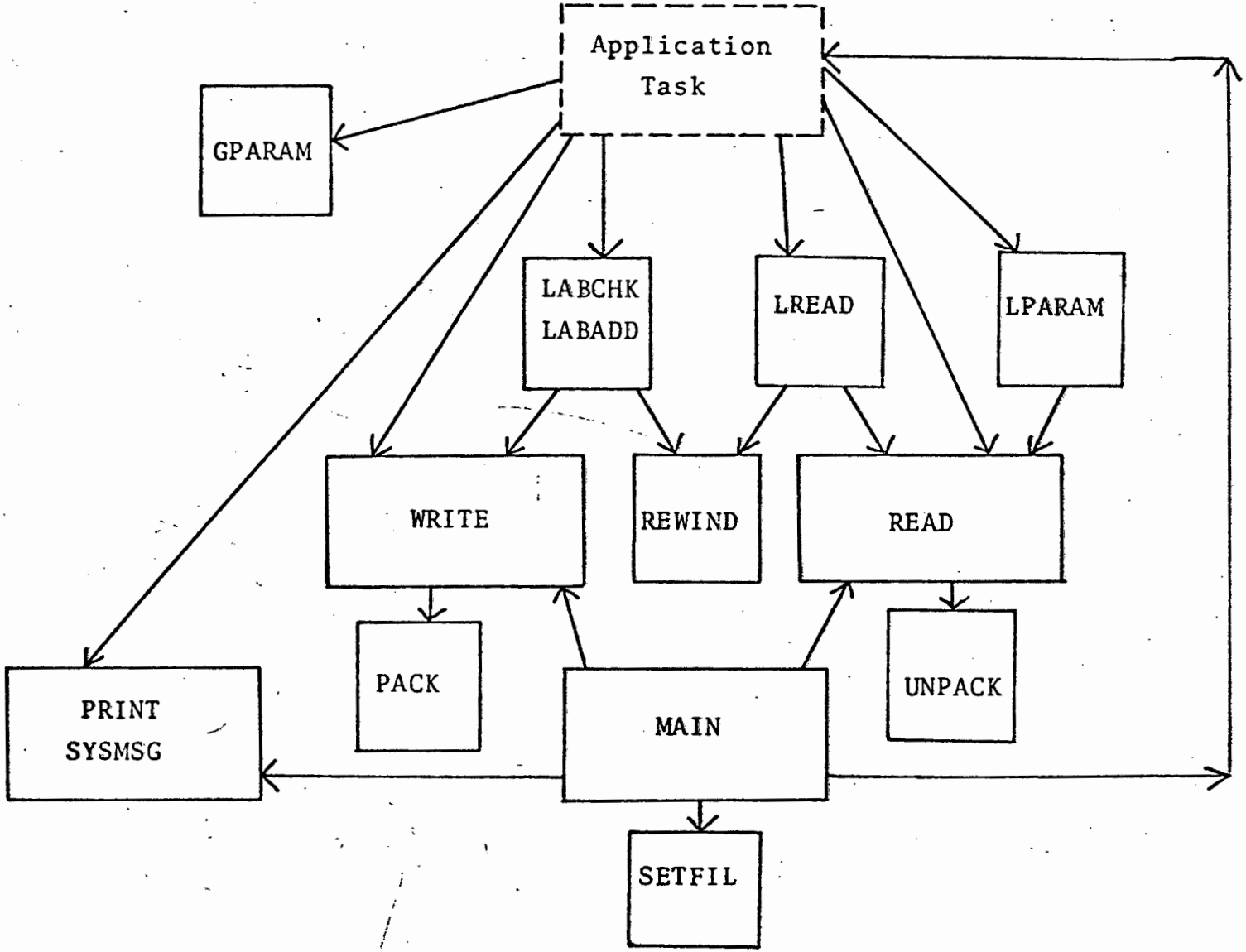
The NUCLEUS consists of eight distinct sections with different tasks. These sections are: (see Figure 4 for interrelation)

(1) Dummy routines: three routines; END, OPEN, ERROR.
(2) The MAIN routine.
(3) Parameter processing: two routines; LPARAM, GPARAM.
(4) Label processing: three routines; LREAD, LABCHK, LABADD.
(5) Printing (line printer): two routines; PRINT, SYSMSG.
(6) Data formatting: two routines; PACK, UNPACK.
(7) File handling: two routines; SETFIL, REWIND.
(8) Input/Output: two routines; READ, WRITE.

3.2 The NUCLEUS sections

3.2.1 The Dummy Routines

It was found that application routines could be taken directly from the VICAR system with little change. These application tasks however contained calls to OPEN, ERROR and END which were not envisaged in DIPS. To allow for this they were incorporated into the NUCLEUS. Their role is strictly to accomodate application routines from VICAR. OPEN was initially a routine for opening data sets in VICAR. Since this is not required in the new system it simply returns to the calling program. END was used to close data sets and re-enter the nucleus (VMAST) but it


Application
Task
GPARAM
LABCHK
LABADD
LREAD
LPARAM
WRITE
REWIND
READ
PACK
UNPACK
PRINT
SYSMSG
MAIN
SETFIL

- Figure 4 -

now does a direct return to the caller. ERROR is slightly more than a dummy routine in that if called it halts the job. If any error message is required it must be printed prior to a call to ERROR.

3.2.2 The MAIN Routine

This routine takes the place of the job control manager (VMJC) in the VICAR system. It handles all control data dynamically and does preliminary parameter processing and label handling. Once all this has been accomplished the application task required is loaded into the overlay area.

The MAIN routine can logically be divided up into four sections:

(1) that of initialization,
(2) input of the data records required,
(3) label processing, and
(4) determination of the application routine required and its subsequent call.

The initialization step is quite short. It consists of printing out the time elapsed for the just completed application task (this section is missed the first time through). Each element of an array POINT (scratch file indicators) is set to .FALSE. indicating that there are currently no scratches in use.

Following this the data record reading step is entered. Firstly, the task record is read. If the symbol is not 'T' for task or 'L' for last, an error is printed and the job is halted. The task name is saved in PROCES. The other values NI, NO, SL, SS, NL, and NS are all stored in their relevent places in IGLBL (see Figure 5). It is checked that the SL and SS are greater than zero. If not they are set to one. The number of inputs and outputs are both checked for correct size. If too big an error message is printed and the job is halted.

IGLBL

START LINE	(from task card)
START SAMPLE	(from task card)
NUMBER OF LINES	(from task card)
NUMBER OF SAMPLES	(from task card)
NUMBER OF LINES	(from system input label)
NUMBER OF SAMPLES	(from system input label)
NUMBER OF INPUTS	(from task card)
NUMBER OF OUTPUTS	(from task card)
not used	
not used	

- Figure 5 -

Next the input records are read. If NI is zero this section is skipped. Upon reading an input record, it is tested for validity. The task name is then checked for tape input (INTAPE). If this is so, a routine MYTAPE is called to determine the kind of tape to be read. If ordinary file input, it is assigned and allocated a unit number in SETFIL.

At this point the task name is checked for tape output. If so, there will be no output records since the tape name is not known but is searched for and the application task OUTAPE is called directly. The input files are those that are to go out to tape.

The output records are now read. If NO is zero this section is skipped. All the output records are read, tested for validity and the files assigned via SETFIL.

The task name is tested for application task COPIN which reads in a partially processed file from tape. If it is required it is called directly. The test for COPIN is made at this point as it is now known what the file(s) that the tape will load into are called from the output records. There is no label processing for this call which is literally a direct copy of a previously used file back onto disk. There is also no label processing for OUTAPE which is a direct copy of a disk file, labels and all, to tape. OUTAPE and COPIN are the reverse of each other, they do no manipulation hence no label processing.

The label processing begins at this point. There are different kinds of processing depending on the type of input and output combinations.

(1) NI=0, a system label and user label for the task are created and written to the output file. The application task section is then entered.

(2) NO=0, the secondary input files are positioned at the beginning of the data. The primary input is positioned at the beginning of the data and the application task section is entered.

(3) NI, NO≠0, the secondary inputs are positioned at the beginning of the data. The system label read in from the primary input is updated by the values in IGLBL(1)-(4) for the output file(s). The primary input is positioned past the labels - as they are read they are written to the output file(s) (with the new system label). A user label for the current task is added and written out as well. The application task section is entered.

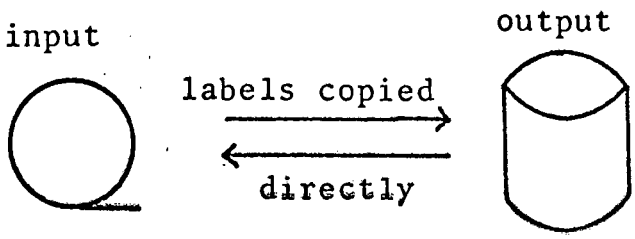
(4) Input=tape, NO≠0, a system label is created from IGLBL(1)-(4). The tapename (as taken from the input record) is inserted into the system label words 1 and 2. A user label is added and these are written out to the output(s). The application task section is entered.
NOTE: This is raw data input, not COPIN.

See Figure 6 for label processing differences.

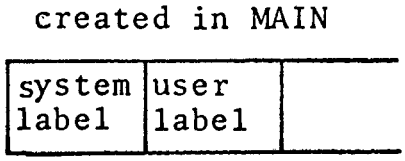
At this point the application task section is entered. The first occurrence is the testing of the first character of the task name to determine which area to start searching in. Once this has been decided a jump is made to the relevant area depending on this character. In each subsection a series of tests for the application task required is carried out until the correct one is found. An example of this would be a task called SIGNAT. The first letter, S, prescribes a jump into the area for application tasks beginning with S. Once in this area the task name SIGNAT is compared to all the S routines available - in this case it would compare against STRECH and fail, then SCRIBE and fail, and finally SIGNAT which would succeed

Label Processing Differences

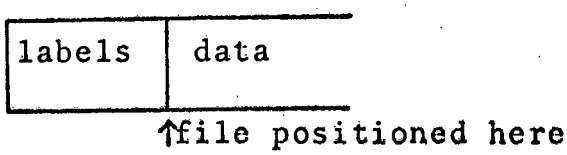
(1) COPIN - OUTAPE



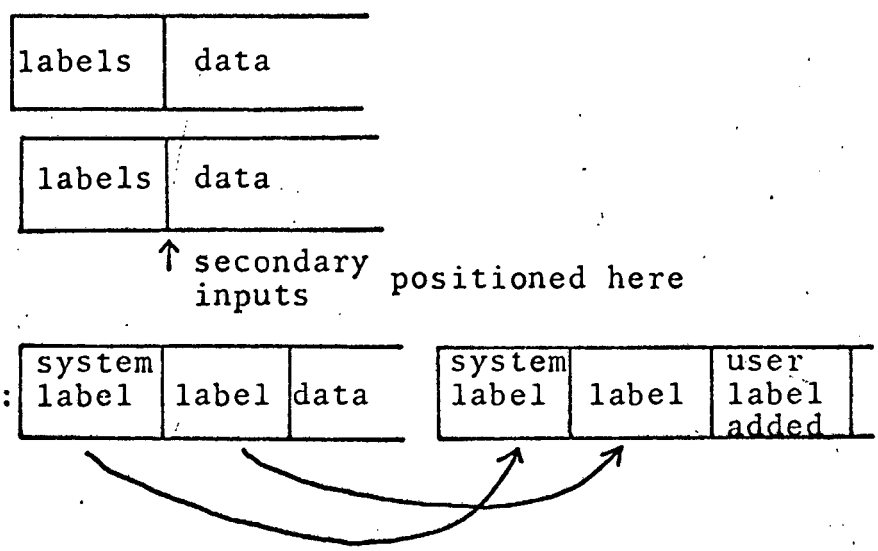
(2) NI=0, example = GEN



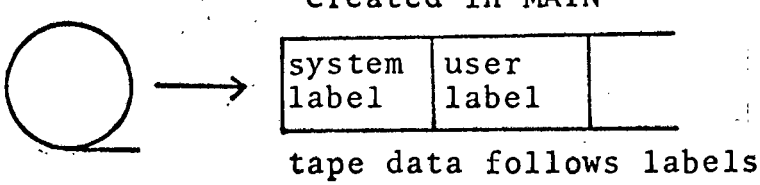
(3) NO=0, example = LIST



(4) NI, NO≠0, example = STRECH



(5) Input = tape, NO≠0, example = LSATIN



- Figure 6 -

and initiate the loading and calling of SIGNAT. However if SIGNAT did not exist it would cause a branch to the error section thus calling PRINT with an error message and halting the job.

3.2.3 The Parameter Processing Routines

These routines are used exclusively by the application task to make the data required available to it. This section consists of two routines GPARAM and LPARAM.

The parameters for each application task specify such things as picture size, number of inputs and outputs and user parameters. The user parameters are provided by a call to LPARAM which reads in the parameter records.

LPARAM is usually called (if needed) near the beginning of the routine so as to make the parameters immediately available to the routine. Upon entry to the routine the real buffer is checked for size. Real parameters are treated slightly differently than integer because of the difference in representation in a computer word.

If the parameters to be read in are integer or character a divisor is set to 10. If they are real it is set to 5. The number of parameters to be read is divided by the appropriate divisor to obtain the number of records to be read. An incremental value is added for the case of integer division yielding a zero answer. The increment will push the answer up to its correct value of 1.

The records are read (via READ) one at a time and processed. Character parameters are loaded directly into the array as they are read in character format so no conversion is required. Integer parameters are decoded into the parameter array. Real parameters are decoded into the real buffer which has been equivalenced to an integer buffer. From this buffer they are transferred to the array. This slight variation is needed as there is only one array in the call statement to take the parameters and it is integer. Handling the real parameters in this way fools

the routine into thinking it is handling integers only.

After all the records have been successfully read the subroutine returns control to the application task. In the case of an expected parameter card not containing the symbol 'P', an error message is sent to the printer via PRINT and the job is halted.

The parameters for picture size, number of inputs and outputs etc. are supplied by the subroutine GPARAM. In most application tasks GPARAM is the first subroutine called as it contains the size of the image and other important data. On entry to GPARAM a loop is entered which executes ten times. This loop places the ten words of IGLBL into the ten word array provided by the application task. Upon completion control returns to the application task.

The VICAR system consisted of one routine, PARAM, which when called passed over one array containing both global and local parameters. For application routines with varying parameter requirements this is clumsy hence the change to two routines in the new system. This allows each application task to declare its own parameter area which will be of varying sizes for different tasks. The application tasks also know what parameters they require whether they are integer, real or character etc.. Removing the parameter processing from MAIN does away with a host of routines required for checking out parameter types and doing conversions. The VICAR system always read one record ahead as it never knew how many parameters to expect. This seperation of tasks has also done away with this.

3.2.4 The Label Processing Routines

These routines are used exclusively by the application task to do label processing of their own if required. There are three label processing routines, LREAD, LABCHK, and LABADD (designed and implemented by A.A. Jackson).

These routines allow the application task access to the labels on the files assigned to it. There are various kinds of access that the task may require: check the system label on the output files for correctness, read the labels on the input files or add new labels to the output files (see Figures 7-10 for label formats).

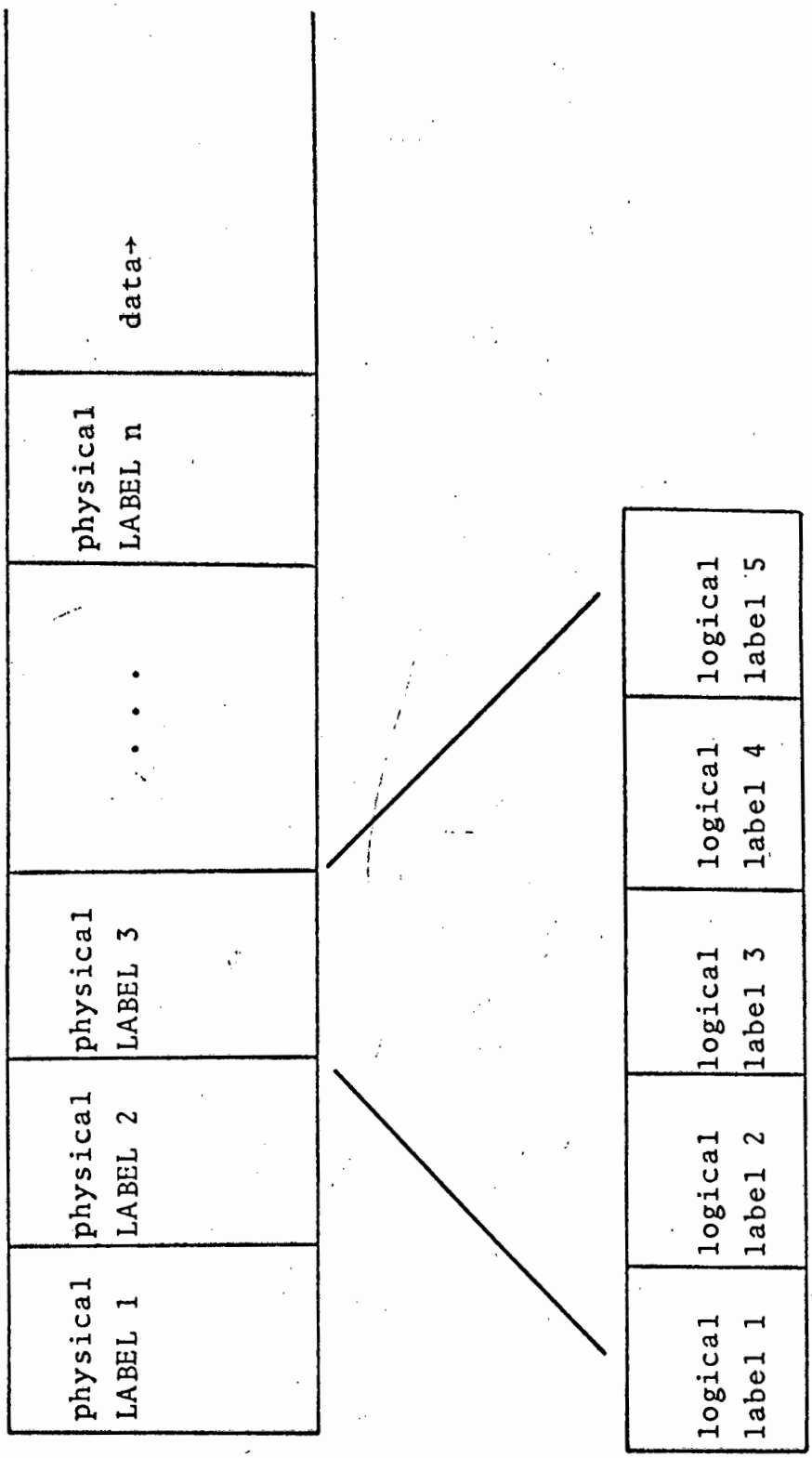
The subroutine LABCHK provides the capability for checking the correctness of the system label. This routine must be called only after all normal output to the file has been completed. It is usually called by all applications tasks that do any file output as the contents of the system label must be checked for correctness.

The routine is divided up into three sections each with a different task. These sections are: date/band/calibration or latitude longitude insertion, global checking and output of the new label.

A positive indicator leads to entry to the insertion section. At this point the first label block is read via LREAD. The indicator is tested for date/band/calibration or latitude/longitude insertion and the required task is carried out by inserting the data items into their requisite positions. If the required task was latitude/longitude insertion the output section is entered immediately. At this point the indicator is tested to see whether global checking is required. If not the output section is entered.

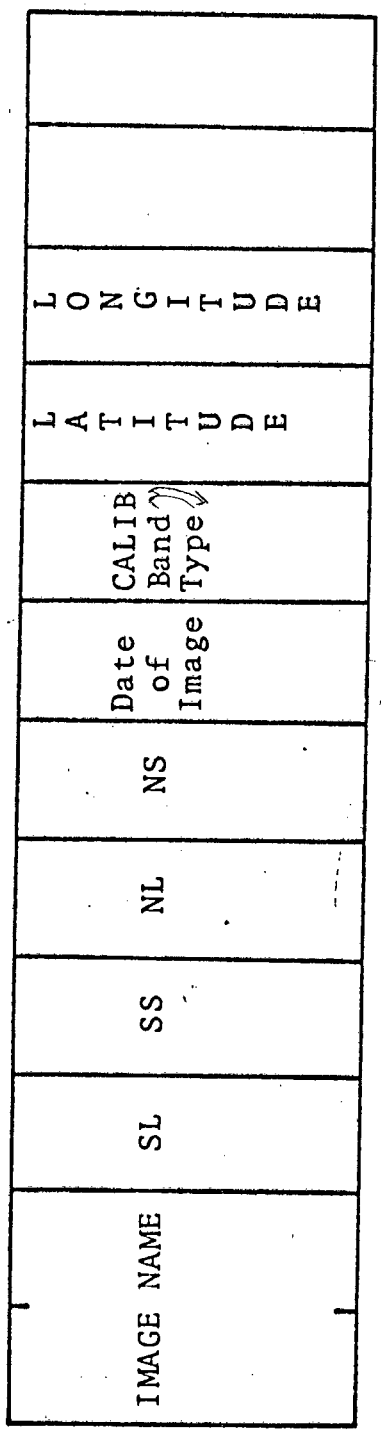
If global checking is required this section of the routine is entered. A logical item is set to .TRUE. prior to the checking. The items passed to the routine are then checked against the same item in IGLBL. If there is any difference in any of the four items checked (SL, SS, NL, NS), the logical item is set to .FALSE.. The logical item is then tested and if found to be .FALSE. the system label needs updating. If the indicator prescribed global checking only the system label must be read via a call to LREAD at this point. Otherwise the label is already available via the insertion section. An immediate return to the

File Structure



- Figure 7 -

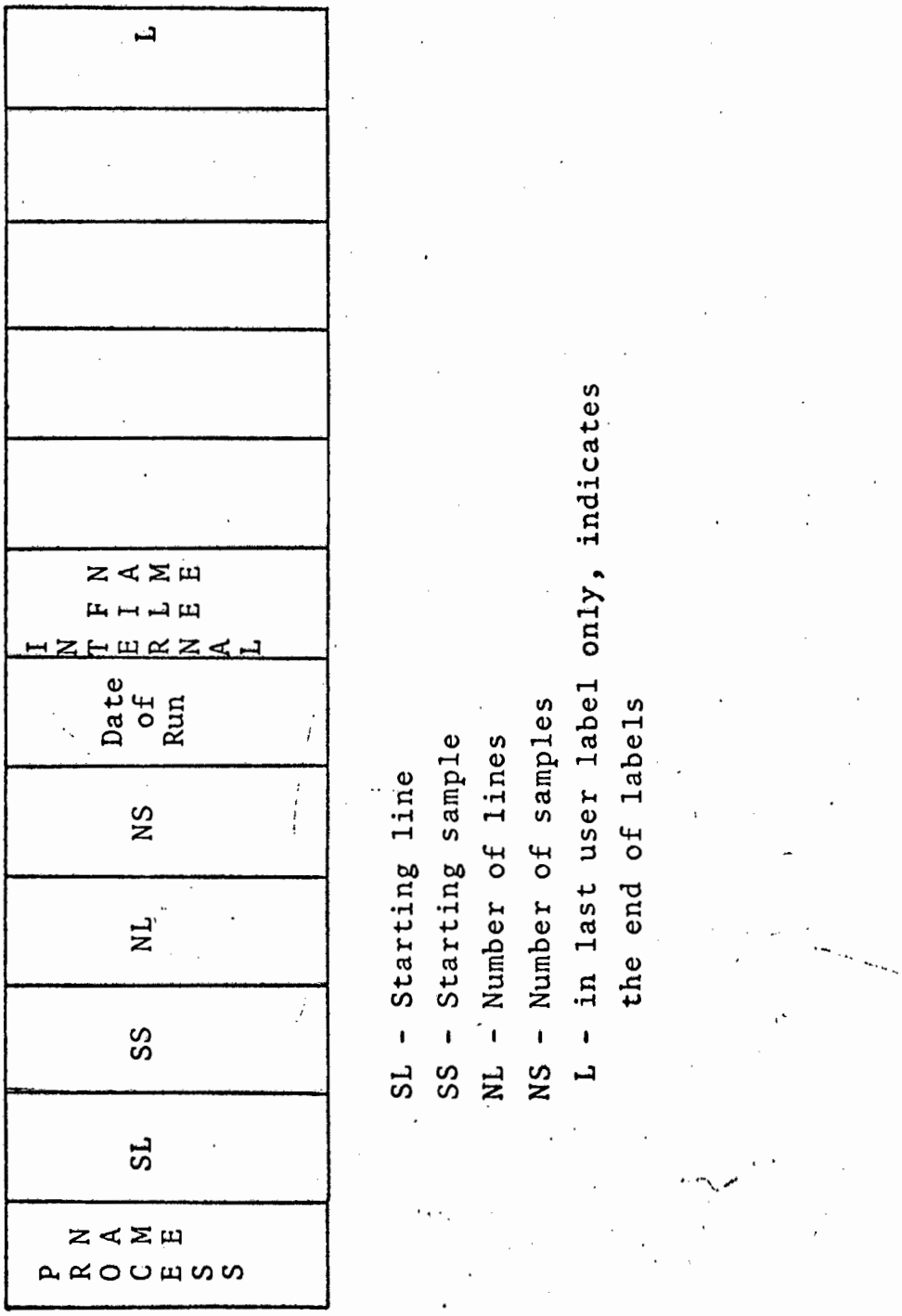
The System Label



SL - Starting line
SS - Starting sample
NL - Number of lines
NS - Number of samples
CALIB - calibration

- Figure 8 -

The User Label



- Figure 9 -

FOURIE	XX	YY	ZZ_1	internal file name	REAL IMAGE	ZZ_2	X_1	X_2			

FOURIE - identifies the label as a Fourier statistic label

XX - number of segments

YY - overlapping between segments

$ZZ_{1,2}$ - number of lines/samples in transform

REAL IMAGE - the type of data on the file, REAL implies a real transform file, IMAGE, the imaginary part,

$X_{1,2}$ - number of segments in the horizontal/vertical directions

The internal file name of the initial image being processed is inserted so that the real and imaginary parts of two different images are not processed together.

- Figure 10 -

application task is made for the case of global checking only when all values are correct.

At this point the global values are updated in the system label (if prescribed by the indicator and the logical item). The file is rewound via a call to REWIND and the label is then output in its new form. Following this the subroutine returns control to the application routine.

The reading of labels on the files (apart from in MAIN) is done by LREAD. This routine will read all labels on an input file and the system label only on an output file. It is called by an application task whenever it requires to read labels from an input device. Application tasks have no access to the labels on the output files except through LABCHK and LABADD.

Upon entry the file number is tested to see whether an input or output device is being operated on. For output devices the file is rewound via a call to REWIND and the first label block (containing the system label) is read. This only occurs through LABCHK. After reading the label block, LREAD returns control to LABCHK. For input devices the current position in the file is checked. If this is zero a previous label read has been done and the next label block is read. If this is not zero the file is positioned in the data area. It is then rewound, the current position is set to zero and a label block is read. After reading of the label block control returns to the application task.

The last routine belonging to this group, LABADD, adds new user labels to the output file prior to any normal I/O. At present this routine is used to add on labels which will distinguish non-image files from image files.

On entry to the routine the specified output file is positioned at the end of the last label block ready for data output. A backspace is performed to get to the beginning of this label block. This label is read and the end-

of-labels symbol is located and cleared (if not found an error message is printed out and the job is halted). The last word of the new label vector is then set to ' L', the end-of-label flag, and these new labels are output to the file. The last label block if not already full is filled with labels and output. Then the rest of the new labels are output, five logical labels per block, until they have all been added to the file. When completed control returns to the application task.

None of these routines are available in VICAR. There is a label read but it is used by VMJC for label reading. It was decided to incorporate a seperate label reading routine to keep the READ routine from becoming too complex. LREAD is short and does little but having it as a seperate routine prevents numerous tests and jumps in READ. The actual reading of the label is done in READ but LREAD does the repositioning.

LABCHK was looked upon as a very good idea. In many cases the system label will have certain values for SL, SS, NL, and NS. In the application routine for some reason it may be necessary to change these. If the new values are not put into the system label errors will result as the image in the file will be different than that thought to be there. The insertion of band/calibration/date and latitude/longitude which are not known till the end of the run are also secondary useful additions. LABCHK operates only on output files.

LABADD is the most recent addition to the group. During development of application routines it was found that some files would be non-image files. In these cases special user labels are added to distinguish these files from the ordinary image files as errors will occur if they are confused. An example of this is Fourier files where the intermediate files contain transforms. These are definitely non-image and the added labels are used to distinguish them (see Figure 10).

3.2.5 The Print Routines

All output to the printer is handled by these routines. The subroutine SYSMSG was incorporated into the system for the purpose of application tasks taken from VICAR which might call it. Upon entry it immediately calls PRINT which actually handles the output to the printer. Upon completion control returns to the application task.

It is far wiser to call PRINT for output to the printer as this does not involve two subroutine calls. Upon entry the number of characters in the message is converted to number of words and this is inserted in character format into the format specification for the output. The message is then written out to the printer with this created format. After completing this an error indicator is tested for non-zero. If non-zero, the run is halted otherwise a return to the caller is taken.

At present PRINT is one of the two routines in which the error indicator is used. If a call to PRINT is made with this parameter set to non-zero, it is a flag for halting the job due to some unrecoverable error.

3.2.6 The Data Formatting Routines

This section consists of two routines written in UNIVAC assembler, PACK and UNPACK. The reason for their existence is twofold. Early in the development of the system an attempt was made to read in an entire NOAA image. The try was unsuccessful as the file area assigned was too small. Upon reflection it was found that if the job had succeeded it would have rolled out a lot of the disk just to make room for the one image. It was soon seen from this sobering example that image size would have to be cut down by packing the data before output and unpacking prior to use by the application task. The second reason is that packing and unpacking are simplicity

itself in assembler. If this had been written as part of READ and WRITE in FORTRAN, it would be clumsy and expensive.

These routines do just as their names imply. PACK packs four data items to a word and UNPACK takes one packed word and reproduces the four data words for the application routine. This is allowable as the data is essentially 8 bit whereas the UNIVAC word is 36 bits in length. PACK is called by WRITE and UNPACK by READ. All image files are packed however labels and non-image files are not. Packing and unpacking can be bypassed by setting one of the parameters in the WRITE/READ call.

3.2.7 The File Handling Routines

These routines are used exclusively by the NUCLEUS for file assignment and manipulation. There are two file handling routines, SETFIL and REWIND.

SETFIL is called from MAIN after determining the file names for the run from the input/output records. SETFIL assigns the file and allocates it a unit number (after having freed the unit number as it may have been used in a previous task). Unit numbers for input files are 11-20, output files are 31-40 and scratch files are 41-50. The file is also rewound to the initial position - an event which is needed as it may be positioned anywhere due to use by a previous application task. WRITE calls SETFIL to assign scratch files.

This routine was non-existent in VICAR as all files were known prior to the run and assigned accordingly. This method allows for the dynamic assignment of files all along the way. A whole suite of application tasks starting from a raw data tape and ending up with a final processed image also resident on tape ready for display can be done without prior knowledge of the control language. All that

the system requires are the data records as input (as described in Chapter 2.2) and it does the rest.

REWIND repositions the specified file to its initial position. It is used by the NUCLEUS routines LREAD and LABCHK. All repositioning of files is strictly controlled by the NUCLEUS to prevent errors of positioning occurring by allowing its full use by the application tasks.

3.2.8 The Input/Output Routines

The primary task of the system is to process digital images into a form required by the user. This requires vast amounts of file input and output. To prevent the scattering of input/output throughout the system, two routines have been provided which do all I/O that is likely to be required. These routines are READ and WRITE.

An important concept to be understood for the full knowledge of input/output in the system is the assignment of file numbers and unit numbers (see Figure 11). The application tasks, in calling READ and WRITE, use file numbers. READ and WRITE convert these into the appropriate unit numbers. For example, if an application task has 4 inputs and 3 outputs it will call READ as follows: CALL READ (IND,DSRN,...) where DSRN is 1-4 depending on the file to be read. As for WRITE, the sequence would be exactly the same: CALL WRITE (IND,DSRN,...) where DSRN is 1-3 depending on the output file. Since input files have unit numbers 11-20, the unit number is the file number plus ten, for outputs, 31-40, the unit number is the file number plus thirty. Scratch files are used as follows: CALL READ/WRITE (IND,DSRN,...) where DSRN is -1 to -10 depending on the number of scratches being used and which one is desired. The scratch unit numbers are therefore the absolute value of the file number plus forty as scratch units are 41-50.

The Conversion of File Numbers to Unit Numbers

Input Unit Numbers	Output Unit Numbers	Scratch Unit Numbers
11-20	31-40	41-50

```
APPLICATION TASK
      :
CALL READ(IND,1,...)
      :
CALL READ(IND,-3,...)
      :
CALL WRITE(IND,4,...)
      :
CALL WRITE(IND,-1,...)
      :
CALL LABCHK(IND,4,...)
      :
RETURN
END

SUBROUTINE REWIND
      (IND,DSRN,...)
      :
I=DSRN+30
REWIND I
      :
RETURN
END
```

```
SUBROUTINE READ(IND,DSRN,...)
            :
NUM=DSRN+10
IF(DSRN.LT.0)NUM=IABS(DSRN)+40
            :
RETURN
END

SUBROUTINE WRITE(IND,DSRN,...)
            :
NUM=DSRN+30
IF(DSRN.LT.0)NUM=IABS(DSRN)+40
            :
RETURN
END

SUBROUTINE LABCHK(IND,DSRN,...)
            :
CALL LREAD(IND,DSRN+20,...)
            :
RETURN
END

SUBROUTINE LREAD(IND,DSRN,...)
            :
CALL REWIND(IND,DSRN-20,...)
CALL READ(IND,DSRN,...)
            :
RETURN
END
```

- Figure 11 -

The label processing routines are a bit awkward in this respect as output files are read for the checking of the system label. This is handled in the following way:

(1) LABCHK is called to check the system label on the output device
i.e. CALL LABCHK(IND,1,...), output file 1.

(2) LABCHK calls LREAD to read the first label on the file:
CALL LREAD(IND,DSRN+20,...).

(3) LREAD calls REWIND to position the file ready for label input. This necessitates (for output files only) a call to REWIND with the correct file number hence CALL REWIND(IND,DSRN-20,...).
This is due to the fact that REWIND does the relevent conversions to unit numbers itself hence the file number is required.

(4) Finally READ is called CALL READ(IND,DSRN,...), so the file number through the steps is:
   (a) CALL LABCHK(IND,1,...), file number 1 from application task.
   (b) CALL LREAD(IND,1+20,...), file number 21 from LABCHK.
   (c) CALL REWIND(IND,21-20,...), file number 1 from LREAD.
   (d) CALL READ(IND,21,...), file number 21 from LREAD. Here READ adds 10 and the unit number is arrived at, 31, which is correct (see Figure 11).

The READ/WRITE routines have exactly the same calling sequence as the VICAR equivalent. It was left so because it was not known what calling parameters would be needed for the new routines and it is far easier to have too many parameters than too few. As the routines grew in complexity, all the parameters were used for some purpose. At present of eight parameters all are used except one. Most are used for the same purpose as they were in the VICAR system.

READ is the routine that does all input. It has four types of input to contend with which are all drastically different, making for a complex routine. These input types are:

(1) Input of image file data for an application task. This data is read in packed, unformatted with the capability for random access.

(2) Input of labels for the NUCLEUS and application routines via the NUCLEUS. This data is read in sequentially, unpacked, unformatted with a fixed record length.

(3) Input of non-image files which are read in unpacked, unformatted but of variable record length for different runs.

(4) The input of the data records to the NUCLEUS which are read in unpacked and formatted.

These various types of data and formats lead to a lot of checking and sorting out, hence the routine is largely doing a chore of sorting out the type of data it is to read in.

WRITE is the routine that does all the output, apart from printing to the printer which is done directly through PRINT. WRITE has two different types of output to contend with:

(1) Image file output for the application task which is output packed and unformatted sequentially.

(2) Unpacked, unformatted output which includes both labels and non-image files.

If scratch files are needed for a task, it is WRITE that assigns them via SETFIL and takes care of initialization. Scratch files have no labels hence the processing is slightly different to ordinary files when repositioning as labels do not have to be skipped. There is also the capability to write randomly to scratch files whereas this is not allowed on ordinary files. This is necessary as they are input/output files and if it is read and

subsequently written to, the file may be incorrectly positioned for the write.

The actual read, write and file manipulation functions are carried out using the NTRAN link to the system I/O routines written in assembler rather than FORTRAN. This allows more flexibility and can cut I/O overheads by as much as an order of magnitude.

READ does all input for the system. Upon entry, the type of input is checked. If data input records or tape catalogue are to be read, control goes to this section where the function is carried out. Following the main flow of events, the unit number is determined, If the file number specified is negative a check is made to see if the relevent scratch is assigned. If not, an error message is output and the run halted. After obtaining the unit number, a test is made to see if label input is required and if so a branch is made to this section where labels are read.

For ordinary input, the NL, NS, current position and number of labels are extracted from RDTABL (see Appendix). For an input file, a check is made to see that it is positioned correctly past the labels. This is indicated by the current position being zero (if an LREAD has been executed) whereas on exit from MAIN it is always one. Once the position is correct, the record desired is checked against the number of records to determine if it is larger. If this is so, it will cause an end-of-file condition and an error message is printed out and the run halted.

If random access is desired, the file is positioned at the record required, otherwise the next record in sequence is read. If a non-image file is being read, the section for this type of input is branched to. Normal packed input is read in at this point. If there is an error return, control branches to the error section where the error status is determined, output and the run halted. The input is then inserted into the caller's buffer after checking that the amount of data required by the user is less than or equal to the amount read and that the skipping

value is not negative. If either of these events occurs an error message is printed and the run is halted, otherwise the data is unpacked and inserted into the buffer and control returns to the caller.

WRITE is very similar to READ in many ways in that it has the same parameters and is called in the same way. It handles all output in the system apart from messages to the printer which are handled by PRINT.

Upon entry to WRITE, the unit number is determined. If a scratch file is indicated, the file number is checked to make sure it is less than 10 (only 10 scratch files allowed). If it is above this value an error message is printed and the job is halted. For a bona fide file number the file is assigned and the RDTABL entries are set up. The record number to be written is then checked and the file positioned ready for the write. A record number of zero results in an error message and the halting of the run.

Back to the events carried out for scratches and ordinary files, the skipping factor is checked to make sure it is not negative. If this condition arises an error message is printed out and the job is halted. If unpacked output is indicated this section of the routine is branched to.

For ordinary output, PACK is called to format the data correctly and then it is written out. If an error return is indicated, control branches to the error section where the error status is determined, printed and the run halted otherwise WRITE returns to the caller.

4 The File Save Routines

4.1 Introduction

These routines are all tape handling tasks for the input and output of data from and to tape. There is also a small routine called from the NUCLEUS to do tape catalogue searches prior to initiation of the tape handling tasks. The purpose of these tasks is to see to the saving of partially processed images which would otherwise be lost at the end of a run, as all disk files will be temporary. The opposite job is to return the file back to disk upon request by the user for further processing. Although they are not core resident, these tasks come into the realm of image capture as they save the image and make it reavailable when required.

4.2 The Catalogue Search

This is a semi-nucleus task in that it is called prior to application task entry, but it is not core resident. It resides in a segment by itself called TAPE (see Figure 3). MAIN in the NUCLEUS calls the routine, MYTAPE, whenever the task name is INTAPE (the name is self explanatory).

Upon receiving this task name, instead of assigning the input file, MYTAPE is called. Upon entry to MYTAPE, the external tape catalogue is assigned. This is the catalogue of raw data tapes (LANDSAT and NOAA). The catalogue is read line by line (see Figure 12 for format of catalogue) until a match between the input file name and the name in the catalogue is made. Upon finding a match, the task name is set to 'NOAAIN'. Word seven of the catalogue line is then tested and if equal to 'L 1', the tape is LANDSAT and the task name is reset to 'LSATIN'. The catalogue is then released and control returns to MAIN.

In the case of no match between the input file and the tape catalogue, the error indicator is set to 3 in

```
                                                    MT0000
1600 11087-08371  T%10267717201234   L   1MT0001  2340  2340  2340  2340
1600 11190-07143  T%10217715201234   L   1MT0002        2340  2340  1240
1600 11180-08015  T%11017721341234   L   1MT0003  2340  2340  2340  2340
1600 11084-07265  T%11027726591234   L   1MT0004  2340  2340  2340  2340
1600 1N5-37587TR  T%11047712051       L   2MT0005  3433.
1600 1N5-35977TR  T%11047713181       L   2MT0006  3445
1600 1N5-36327TR  T%11057769191       L   2MT0007  3445
1600 11181-08005  T%11077712301234   L   1MT0008  2340  2340  2340  2340
1600 11222-08340  T%11227712541234   L   1MT0009  2340  2340  2340  2340
1600 11180-08010  T%11087711141234   L   1MT0010  2340  2340  2340  2340
1600 11110-08125  T%11167712271234   L   1MT0011  2340  2340  2340  2340
1600 11222-08433  T%11237713251234   L   1MT0012  2340  2340  2340  2340
1600 11121-08370  T%12087714161234   L   1MT0013  2340  2340  2340  2340
1600 11208-12350  T%12097712331234   L   1MT0014  2340  2340  2340  2340
1600 11289-12402  T%12127714071234   L   1MT0015  2340  2340  2340  2340
1600 11209-12404  T%01127712241234   L   1MT0016  2340  2340  2340  2340
1600 11145-08073  T%01167712111234   L   1MT0017  2340  2340  2340  2340
```

tapename determines whether LANDSAT (L 1) or NOAA (L 2)

- Figure 12 -

the READ routine to indicate the end of the catalogue has been reached. In this case the task name is set to 'COPIN ' which indicates that the file is a preprocessed file resident in the internal tape library. The tape catalogue is released, the error indicator reset to zero and control returns to MAIN.

4.3 The Tape Handling Routines

There are two routines that take care of input and output to tapes of partially processed images. These routines are OUTAPE and COPIN. Both these routines are application tasks and are available to the system user through the medium of the data input records.

OUTAPE is called from MAIN in the application section. This application task is called with one parameter, CALL OUTAPE (P_1) where P_1 is the current date provided by MAIN. Directly upon entry, GPARAM and LPARAM are called. Of prime importance from GPARAM is the seventh global parameter, the number of inputs. LPARAM provides the routine with the parameters.

The internal tape catalogue (see Figure 13 for format) is assigned and is positioned at the end. The end-of-file mark and the last record are backspaced over, and then the last record is rewritten. The name of the last tape written to is extracted from the catalogue. The new tape-name is created by adding one to the latest tapename.

At this point, a message is sent to the computer console asking the operators to load the required tape. The tape is then assigned and processing begins.

The labels are read via LREAD and the number of samples per record is extracted from the system label. The labels are written out to the tape. The data is then read and written out to tape until the end-of-file on $FILE_i$ is reached whereupon and end-of-file mark is written onto the tape. The internal tape catalogue is updated with

The Internal Tape Catalogue

file name	tape name	date of run
	0314IP	080277
FILE01	0315IP	120477
FILE02	0315IP	120477
PENIN4	0316IP	010778
PENIN5	0316IP	010778
PENIN6	0316IP	010778
PENIN7	0316IP	010778
XREAL	0317IP	011778
XIMAG	0317IP	011778

NOTE: The first record is a dummy record.

- Figure 13 -

the name of the file written out, the tapename, and the current date. The next file is then transferred to the same tape in exactly the same way.

Upon completing the output to tape of all the files, an extra end-of-file mark is written to signify end-of-tape and an end-of-file is written to the tape catalogue. Both the tape and the catalogue are freed and control returns to MAIN.

If during processing, the end-of-tape is reached while in the middle of transferring a file, extra precautions are required. If the file being written to the tape is the first one, an error message is written out indicating that the file is too large to fit on tape. For other cases, the tape is repositioned to the end of the previous file output, an end-of-tape mark is written and the tape is freed. Control then branches to the section where a tape name is determined and a new tape is allocated for the rest of the files to be output.

In case of any I/O errors during processing, control jumps to the I/O error status section, where the status word is obtained and printed out. This is done by decoding the status word into an error message buffer. The first digit of the status word must be converted from decimal to octal and the resulting code looked up in 'SPERRY UNIVAC 1100 SERIES PROGRAMMER REFERENCE', volume 15, pg 4-41. In the case of NTRAN errors, a negative value will probably be printed out, and this can be found in the UNIVAC reference as above under NTRAN.

COPIN does exactly the opposite of OUTAPE. When a user requires to continue processing on a partially processed image, COPIN is the task used to reload the file onto disk.

Upon entry to COPIN, GPARAM and LPARAM are immediately called. The tape catalogue is then assigned (see Figure 13).

At this point a loop starts to load the files. This

loop is executed NO times. The names of the files to be loaded from are in an array called LPAR and as the catalogue is read, each name is checked against LPAR(i). When a match between LPAR(i) and the catalogue is established, the tape is assigned. A message is sent to the computer console asking the operators to load the tape and it is then available for the task.

The tape is then positioned forward past the unwanted files to the correct position. Firstly, the labels are read and transferred to disk. At this stage the number of samples in each data record is extracted from the system label. After the labels are transferred, the data is transferred until the end-of-file is reached. Upon reaching this point, an end-of-file is written to the disk file, the catalogue is rewound and the process repeated for the next file to be reloaded.

Upon completing all reloading, the catalogue and the latest tape in use are both freed and control returns to MAIN.

In case of any I/O errors, NTRAN or FORTRAN, the same procedure is followed as that in OUTAPE.

Both OUTAPE and COPIN use an assembler routine to position themselves correctly on tape. This routine is called MOVEB and it provides the capability to move forward or backward over an end-of-file mark to the position required.

The routine consists of two I/O packets, one for forward movement and one for backward movement. The packets are called IOPKT (move back) and OIPKT (move forward). The registers are saved upon entry, the filename is loaded into the packets, the type of move is determined and the move is executed via IOW$. The status of the move is then loaded from the relevent packet into a register. From the register, it is stored for access by the calling routine. The registers are restored, and control returns to the caller.

It is not known whether this tape facility exists in VICAR although it is assumed so. These routines were designed from scratch with no reference to VICAR.

5 The Application Tasks

5.1 Introduction

The purpose of the tasks described in this section is to manipulate the data in some way to present it in a more suitable form for the user. Included are two distinctly different types of tasks. Firstly, there is a line printer display routine which is quite important as a quick way of determining whether the processing is progressing as planned. Secondly, a number of enhancement routines which change the image in some way to make it more presentable to the viewer. Those included are a routine which scribes rectangles around desired areas in a picture and a number of Fourier transform tasks which change the form of the image by weighting it in some way in the frequency domain. Upon retransforming, a vastly different image is produced by this manipulation of the Fourier transform.

There are many more routines available than the few described here. For a full list of application tasks and their purpose, see Appendix A.

5.2 The Line Printer Display Routine

The routines which produce the line printer displays of an image are located in a segment called DISPLY (see Figure 3). There are three routines required for the carrying out of the task. These are DISPLY, CHPACK and PAGE. DISPLY is the task which is called by the user, and the other two are transparent routines called by DISPLY.

Since the line printer only has the capability to print characters and in a fixed format, this had to be turned to advantage in some way. The method of producing an image on the printer rests on the capability of being able to represent different levels of gray in some manner. Since there is a finite number of values represented in an image,

they can be represented by line printer characters. With a correct choice of characters, an acceptable image can be produced. The method used herein is as follows: sixteen levels of gray have been represented using different print characters (with overprinting). To bring the image values into range, they are divided by some value to index into the correct print character. This method of display destroys a lot of detail due to the clustering of many pixel values into one print character. It is useful however, as a quick method of determining how one's processing is progressing. Another disadvantage is the distortion due to the differences in printer character spacing down the page and across the page. This can be rectified by prescribing other values, than one, of line and pixel increment.

Upon entry to DISPLY, GPARAM and LPARAM are called to obtain the global and local parameters. The character set used is printed out prior to the displaying of the image proper.

At this point, the system label on the file is read to determine the type of image (LANDSAT, NOAA) for annotation purposes. Two headings are then set up. The first heading is for the first swathe and the second heading for swathes 2 to n. These headings contain the type of image (NOAA, LANDSAT), the band number, swathe number and the image name. Prior to printing the headings, the print control is changed via PAGE to cause continuous printing over pages.

After printing the heading, the pixel markers at the top of the image are computed. They are then output at the top of the image, and the output buffer is saved in order to print the pixel values along the bottom of the image as well.

The output of the image is now executed. A line at a time is read from the file and the pixel values for one page width are converted to print characters. This is

accomplished by dividing the pixel value by sixteen and using the result to index the character arrays CHARS and OVER. CHARS contains the initial character and OVER contains the overprinting character (the first eight characters of OVER are blank). A flag is used to indicate whether overprinting is desired or not. No overprinting is required in a line if only the first eight characters are used. Once the line has been completed, it is packed and output to printer, the first word containing the line number. If overprinting is indicated, (by FLAG), the second array is packed and printed over the first line. This is done for all the lines required and upon completion the pixel values are printed along the bottom.

At this point, the value of the start point of printing is increased to the endpoint, plus one, of the previous swathe. If the number of pixels left to be displayed is zero, the endpoint has been reached and the end of the routine is branched to.

The pixel values for the next swathe are computed, packed and output. The line is again saved for output at the bottom of the swathe. The file is read from the desired point, with the already printed part skipped on input. The required part of the line is processed in the same way as the first swathe, packed and output. The pixel values are printed along the bottom of the swathe upon its completion.

The start point and number of pixels to be displayed for the next swathe are then computed. If there is still output to be displayed, the previous step for the non-first swathe is repeated. When all output is completed, the print control is returned to normal via PAGE and control returns to MAIN in the NUCLEUS.

The packing routine CHPACK packs an array of characters, one per word into six per word ready for output to the printer. Similar to PACK and UNPACK, CHPACK is written in UNIVAC assembler.

The print control routine PAGE causes the printer to print continuously over page edges until the control is reset at the end of the task. The routine is very short and simple written in UNIVAC assembler. Upon entry, the two registers to be used are saved. The type of print control desired is determined and the relevent packet is loaded. An executive request (PRTCN$) is then executed to carry out the command. Upon completion, the registers are restored and control returns to DISPLY.

DISPLY exists in VICAR as routine DISPLAY. The ideas of characters and overprinting were used but the routine itself was not used as it required a call to OUTCON. This was found to be an IBM assembler output conversion program for the conversion of the annotation for DISPLAY. Rather than convert both to the new system, DISPLY was redesigned with the headings etc. printed from the routine itself (using PRINT). For an example of DISPLY see Figure 14.

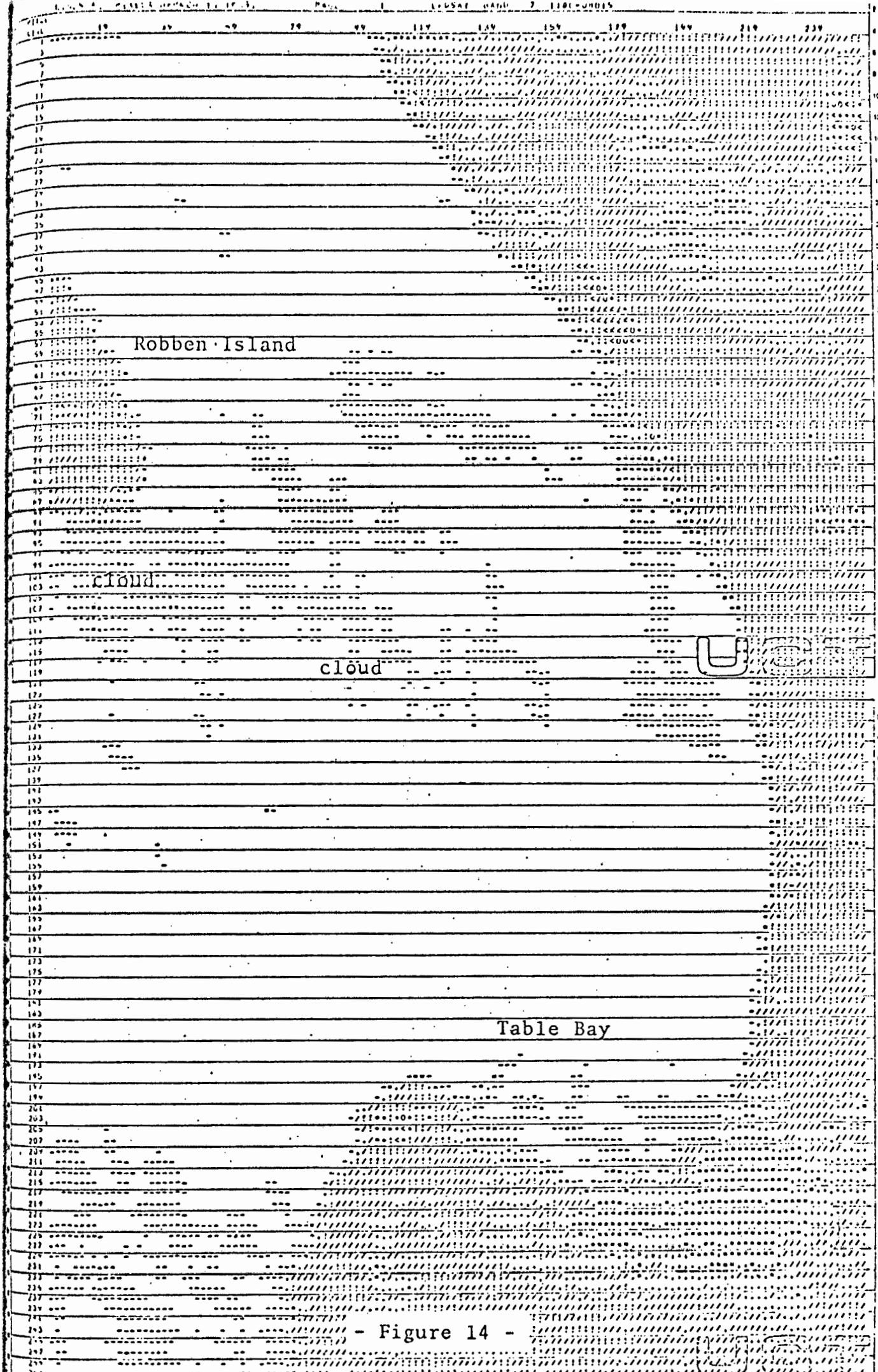
5.3 The Image Enhancement Routines

5.3.1 Introduction

Image enhancement involves the modifying of an image so as to present it to the user in such a way as to make the viewing and interpretation meaningful to him. Three types of image enhancement have been provided herein:

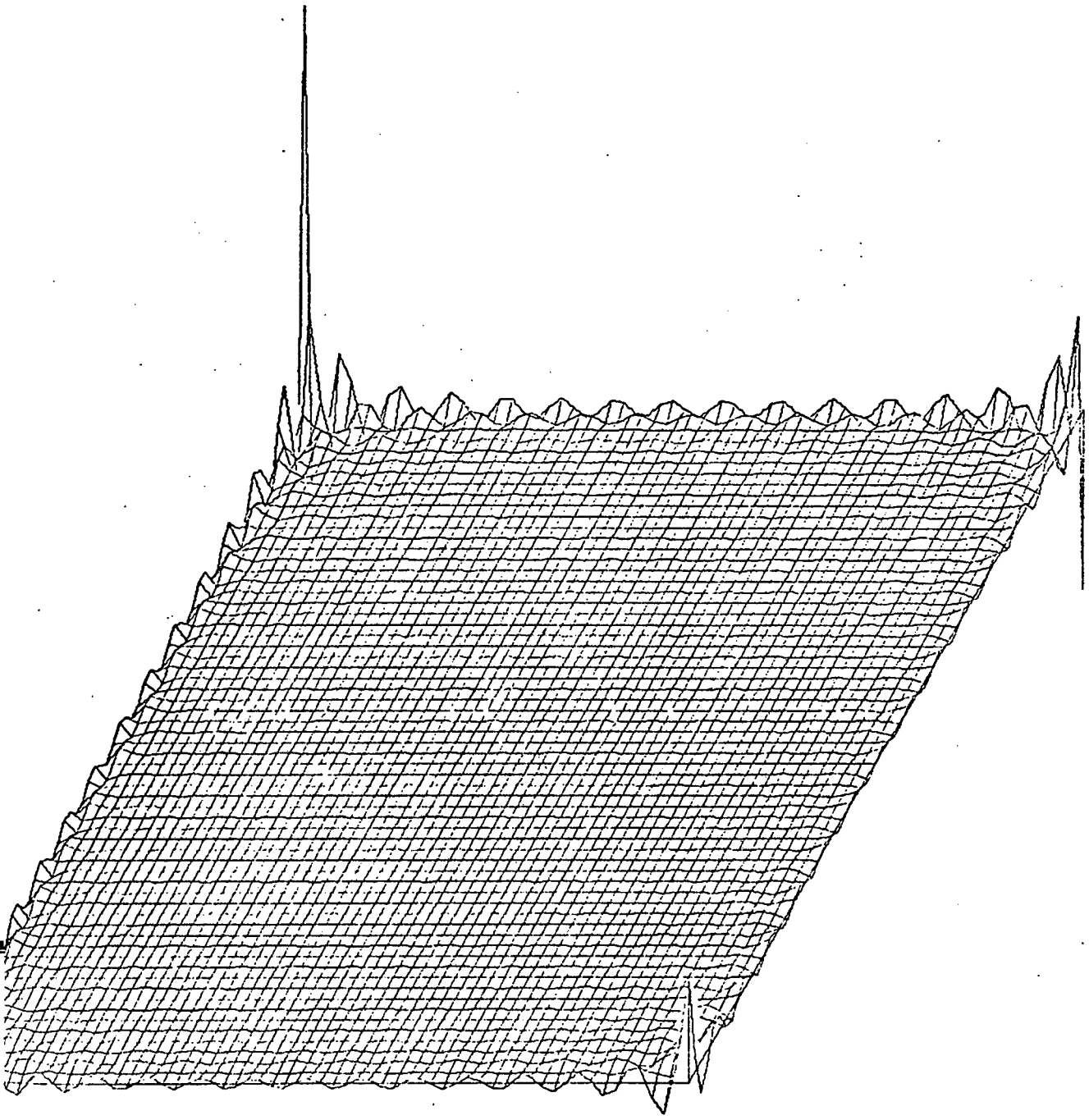
(1) the scribing of rectangles around desired areas in an image,
(2) smoothing of an image to remove unwanted noise, and
(3) edge enhancement.

The scribing task is an example of a minor enhancement although it is an enhancement nonetheless. The other two enhancements are accomplished quickly and easily in the frequency domain. This is accomplished by applying a Fourier transform to a function (the image) and breaking it down into its individual frequency components (see Figures 15 and 16).



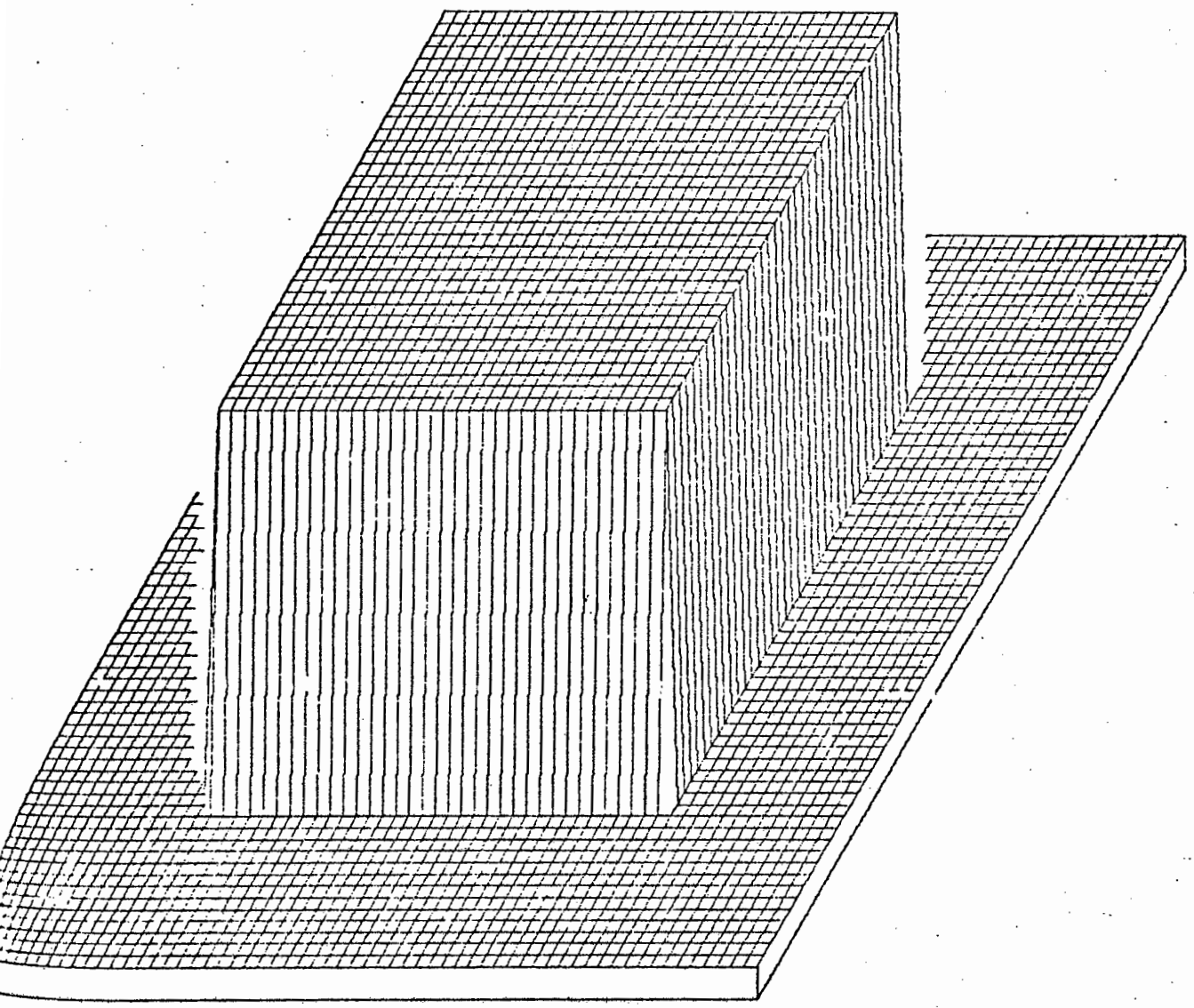
- Figure 14 -

The Transform of an Image



- Figure 16 -

An Image



- Figure 15 -

5.3.2 The Scribing Task

This application task named SCRIBE, is an example of a task taken from the VICAR system and adapted to this system. The task scribes rectangles around areas in an image. Up to thirty rectangles may be scribed in one execution of the task. The pixel value of the scribed lines may be specified by the user or defaulted. In addition, simple background pictures may be generated. The maximum length of lines in the output is 2048 samples. The results of this task are useful in that an entire image may be printed with rectangles scribed around certain portions. These rectangles can be used to illustrate the location of an inset with respect to the overall image.

Upon entry to SCRIBE, GPARAM is called to obtain the global parameters. LPARAM is then called twice, the first call obtains the number of parameter words (NPW) to follow as this is variable depending on the number of rectangles to be scribed. When this is known, the rest of the parameters are read in.

A parameter is now tested a number of times to determine the input type, whether from file or self-generated, and if generated, the generation type required. If the background is to be self-generated, a line is filled with the desired pattern.

A loop is now entered which actually executes the scribing. The SL, SS, NL and NS for rectangle$_i$ are obtained. If the line being worked on is outside the scribing area for the rectangle, the next rectangle is worked on until all 30 have been evaluated and the line is output. If there are less than 30 rectangles to be scribed, the SL, SS, NL and NS will be zero for the latter ones, causing a looping to occur with no evaluation until all 30 are completed and the line is output.

If the line being worked on is the top line of a rectangle, the line values between SS_i-1 and SS_i+NS_i are

1180-08015-7 354-1173 346 -955
SKEW 2 FEB 78 UCT-IPU55
Cape Peninsula Band 7

Note slightly different caption format. Processing included LSATIN, STRECH and SCRIBE as well as SKEW. These were not included due to lack of room on the negative.

set to the scribing value. For lines between the top and bottom of a rectangle, the line values in positions SS_i-1 and SS_i+NS_i are set to the scribing value. The last line in a rectangle is treated the same as the first line and the line values between SS_i and SS_i+NS_i are set to the scribing value. A single line is worked on for all 30 rectangles, allowing for the overlapping of rectangles, as a line is changed for each rectangle in turn.

Upon completion of the scribing and output, LABCHK is called to validate the system label. Directly after this, control returns to MAIN in the NUCLEUS.

5.3.3 The Fourier Transform Routines

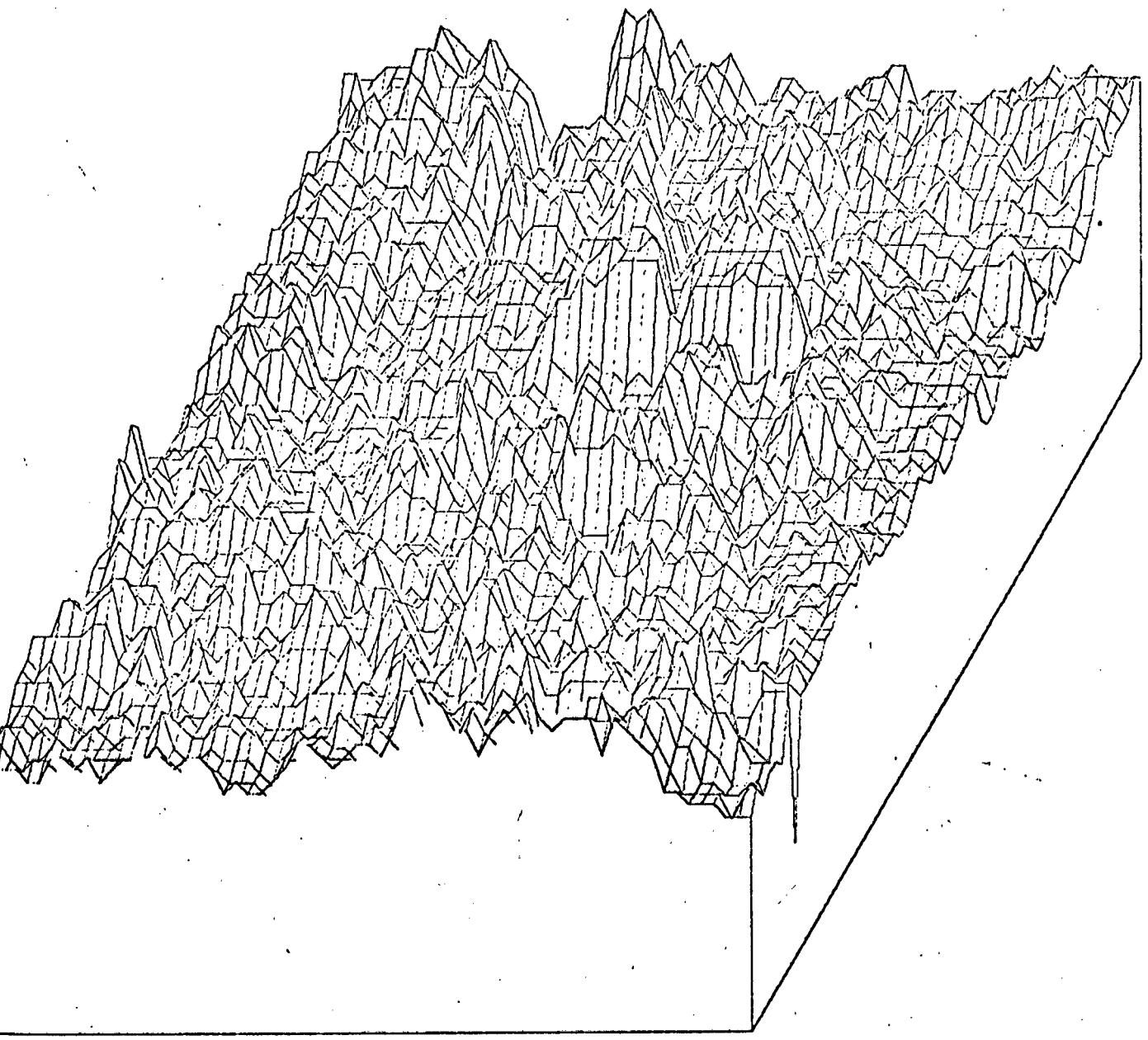
5.3.3.1 Introduction

A problem in almost all image processing is the presence of unwanted, random fluctuations superimposed upon the image, generally known as 'noise'. Therefore, a noisy image 'g' can be represented by

$$g = s + n,$$

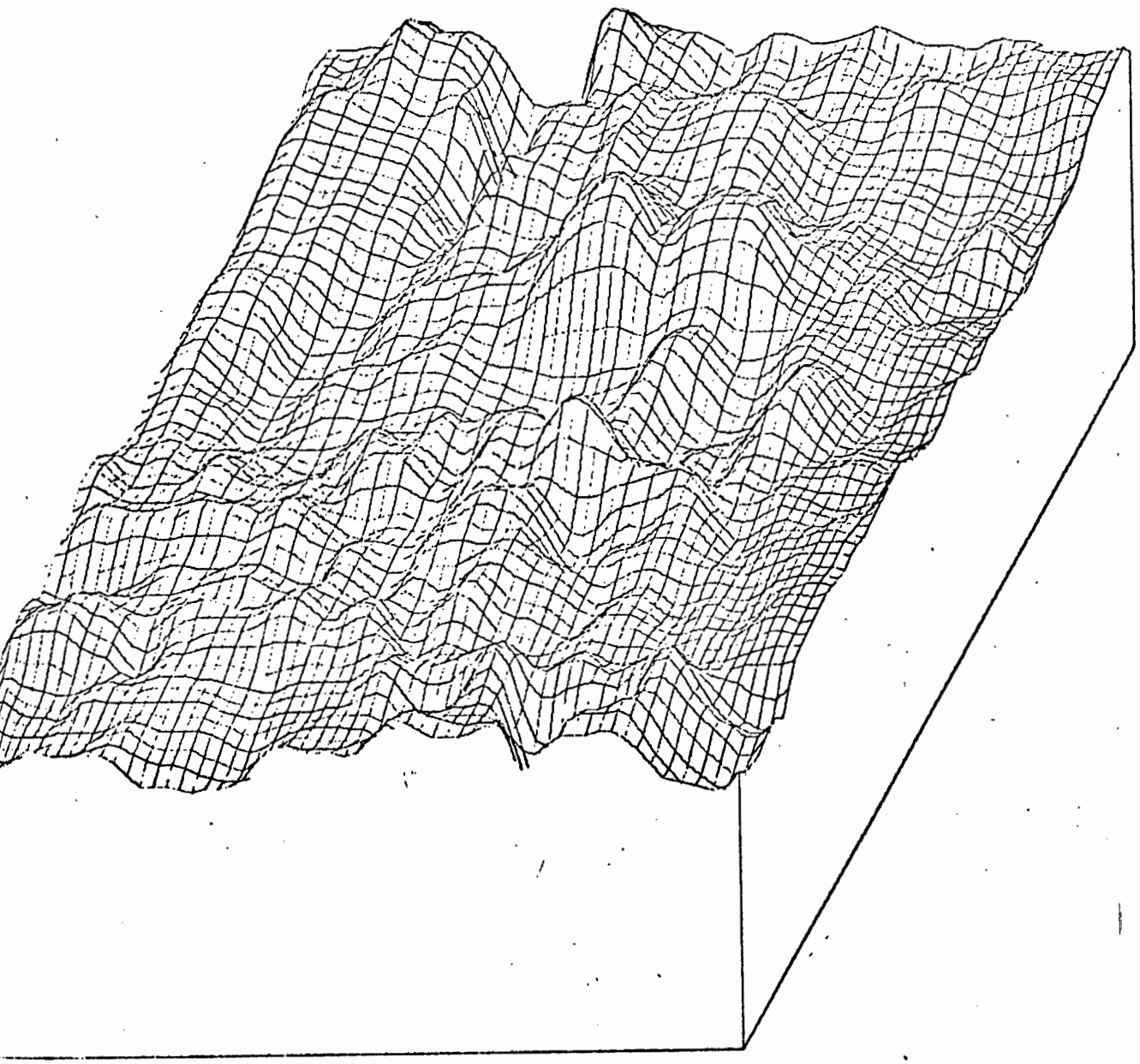
where 's' is the signal and 'n' is the noise. A better approximation to 's' can be obtained by decreasing the high frequency contribution to 'g' while leaving the low frequency components more or less unaltered. In this, a considerable amount of the high frequency noise is removed while leaving the image information relatively unchanged. However, using this technique, one has to compromise between removing noise and loosing resolution (high frequency components of the image). This results in all sharp edges in the picture becoming rounded. This attentuation of the high frequency components is done by the application of a Gaussian smoothing function to the transform. Upon retransformation, the image is obviously smoothed and less noisy (see Figures 17 and 18).

An Original Image
(area near St. Lucia Estuary)

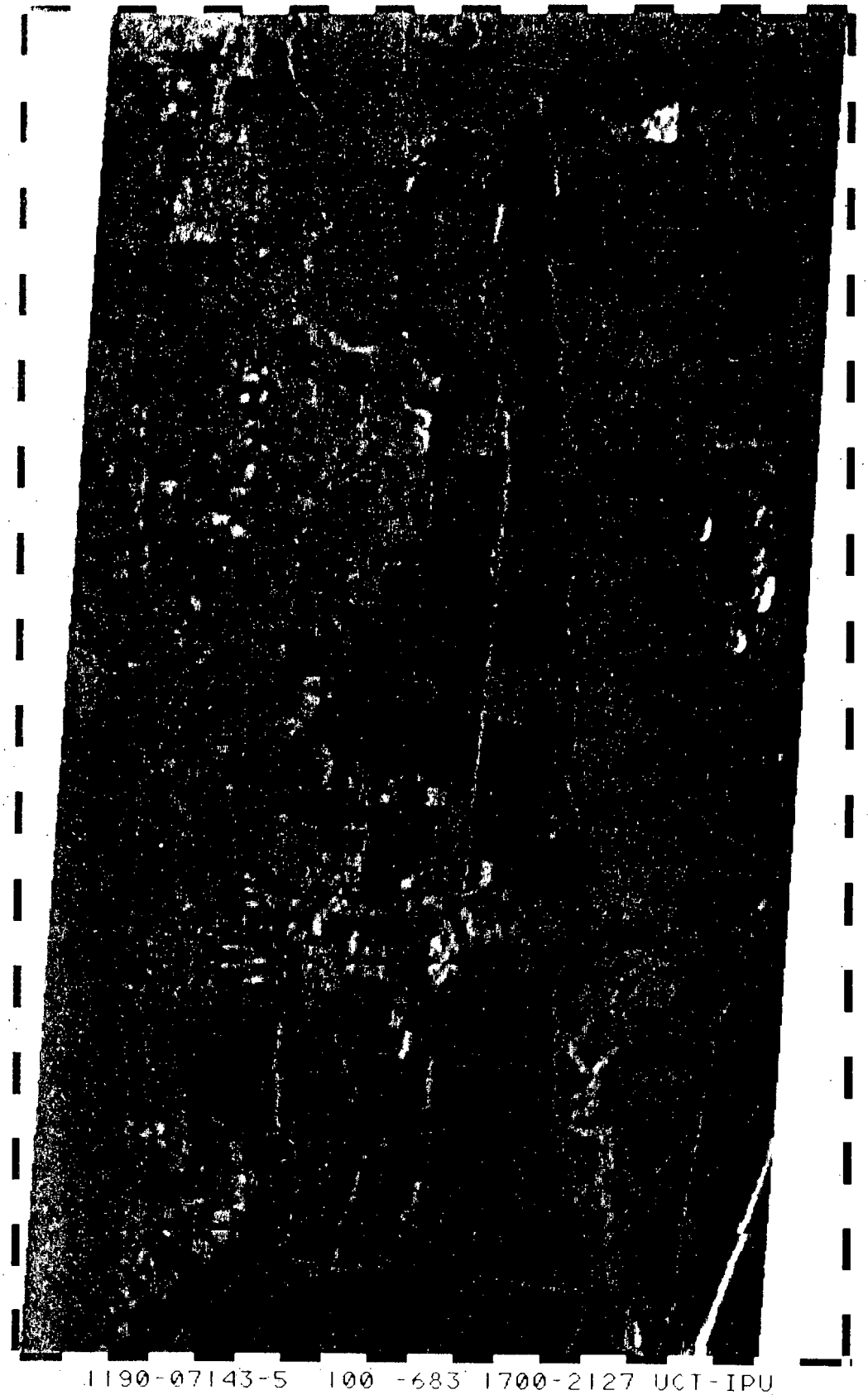


- Figure 17 -

Figure 17 Smoothed



- Figure 18 -



St. Lucia area, Natal Band 5, smoothed

Processing as above as well as LSATIN and TRANS2.

Edge enhancement is accomplished by the simple removal of a number of the low frequency components, as edges are obviously created through the contribution of the high frequency components (see Figure 19). Care must be taken however, not to remove too many low frequency components, or the result upon retransformation is a jumble of noise (see Figure 20).

VICAR contains a host of programs for doing Fourier transforms. However, it was virtually impossible to use any of them as a guideline as they require a specialized hardware unit attached to the system for the sole purpose of performing Fourier transforms. All the Fourier transform routines were designed according to the probable needs of a future user. The actual transform algorithm was taken from Applied Statistics, 1975 (see end of this section). These routines seemed superior to all the algorithms examined, as the unscrambling of coefficients after a transform was also included. All other algorithms did the transform and then glibly stated that unscrambling was required.

In general, a cosine curve with 'k' waves per basic interval, amplitude A[k] and phase angle Φ[k] may be written

$$Y = A[k]\cos(k\theta - \Phi[k]),\ 0 \leq \theta < 2\pi,\ 0 \leq \Phi < 2\pi. \quad (1)$$

This may be expressed in a more useful form by the use of a trigonometric function

$$\cos(R-S) = \cos S \cos R + \sin S \sin R \quad (2)$$

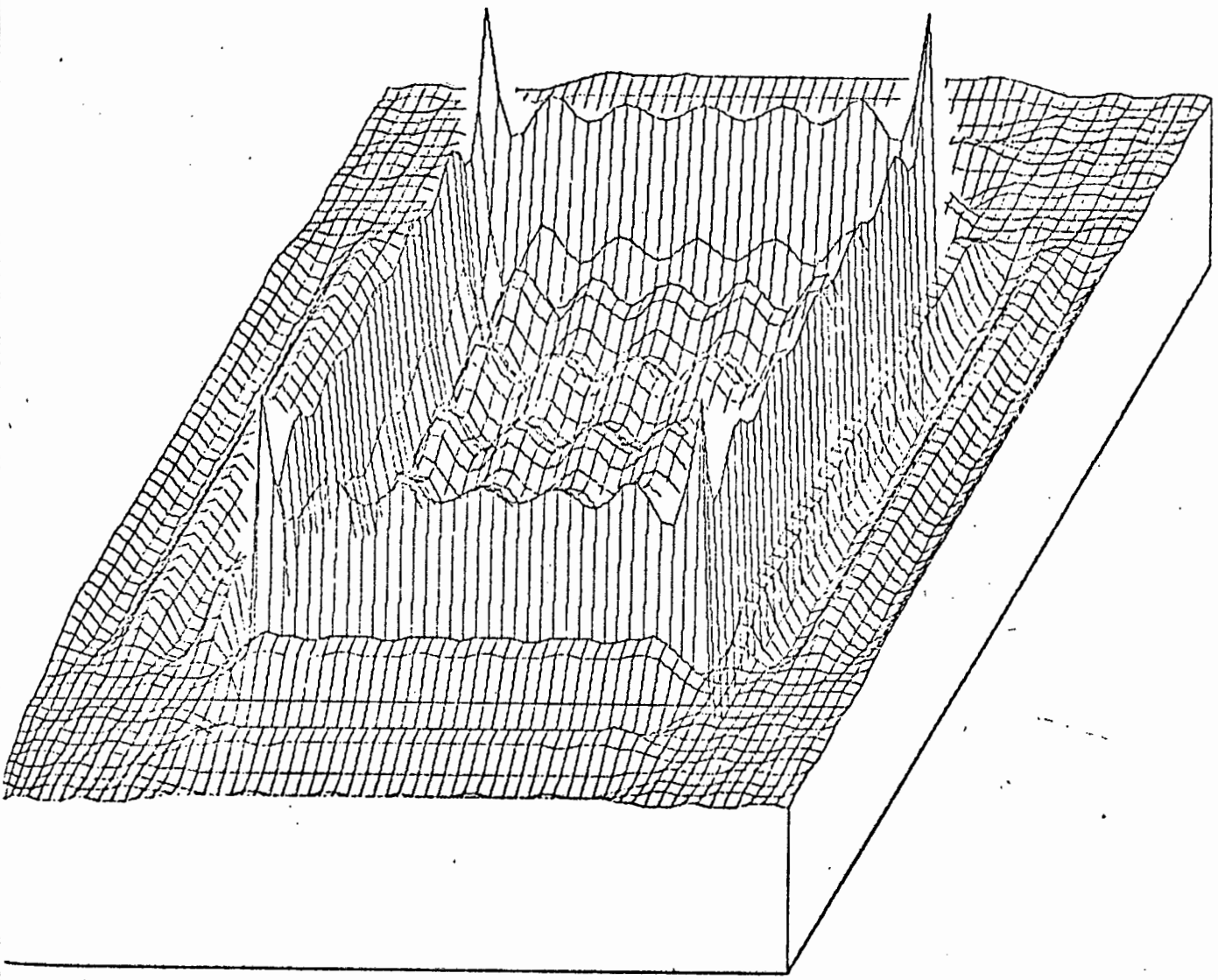
thus equation (1) becomes

$$Y = A[k]\cos(\Phi[k])\cos(k\theta) + A[k]\sin(\Phi[k])\sin(k\theta). \quad (3)$$

Define $a[k] = A[k]\cos(\Phi[k])$ and

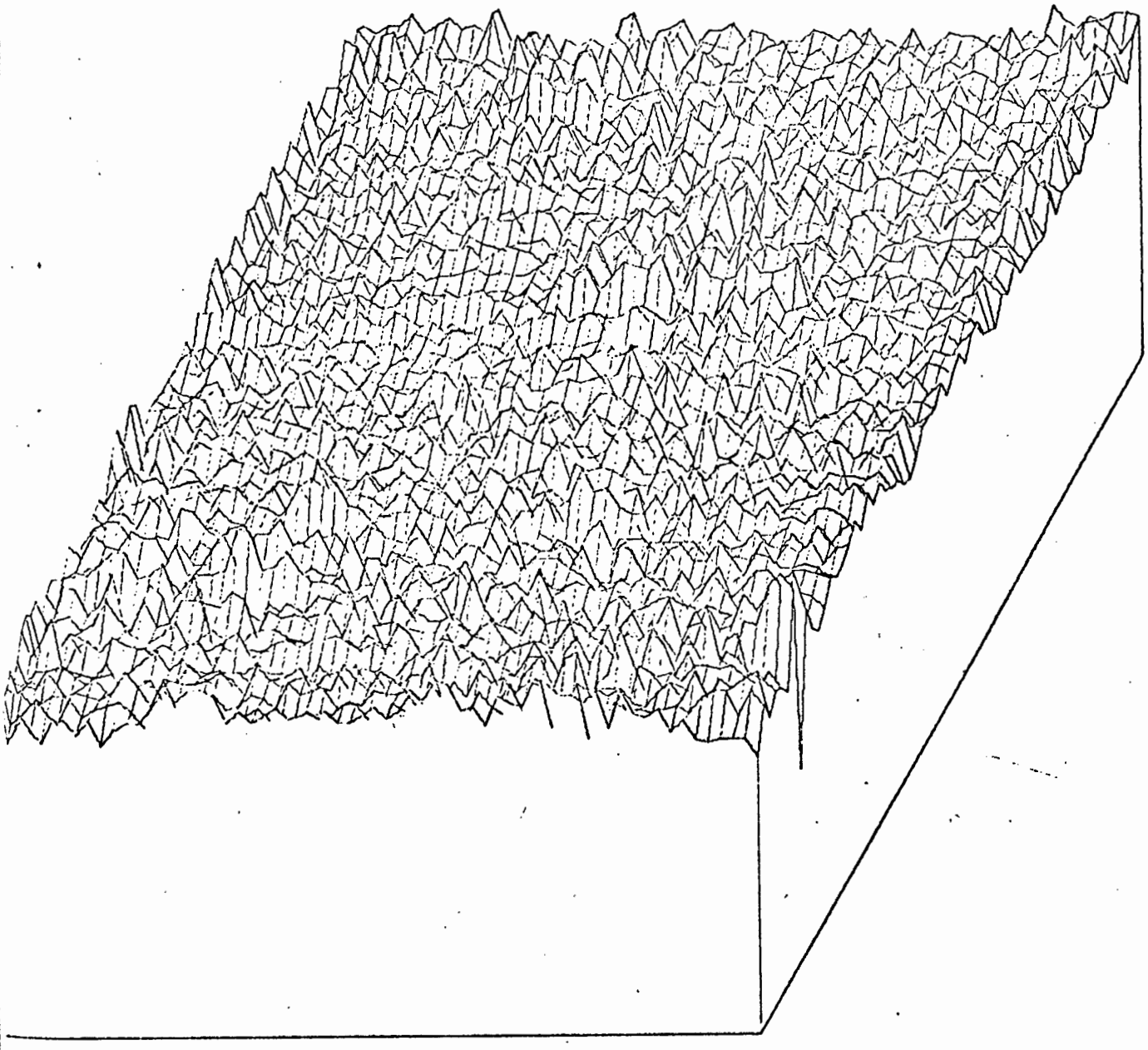
Edge Enhanced Box

(Box in Figure 15)

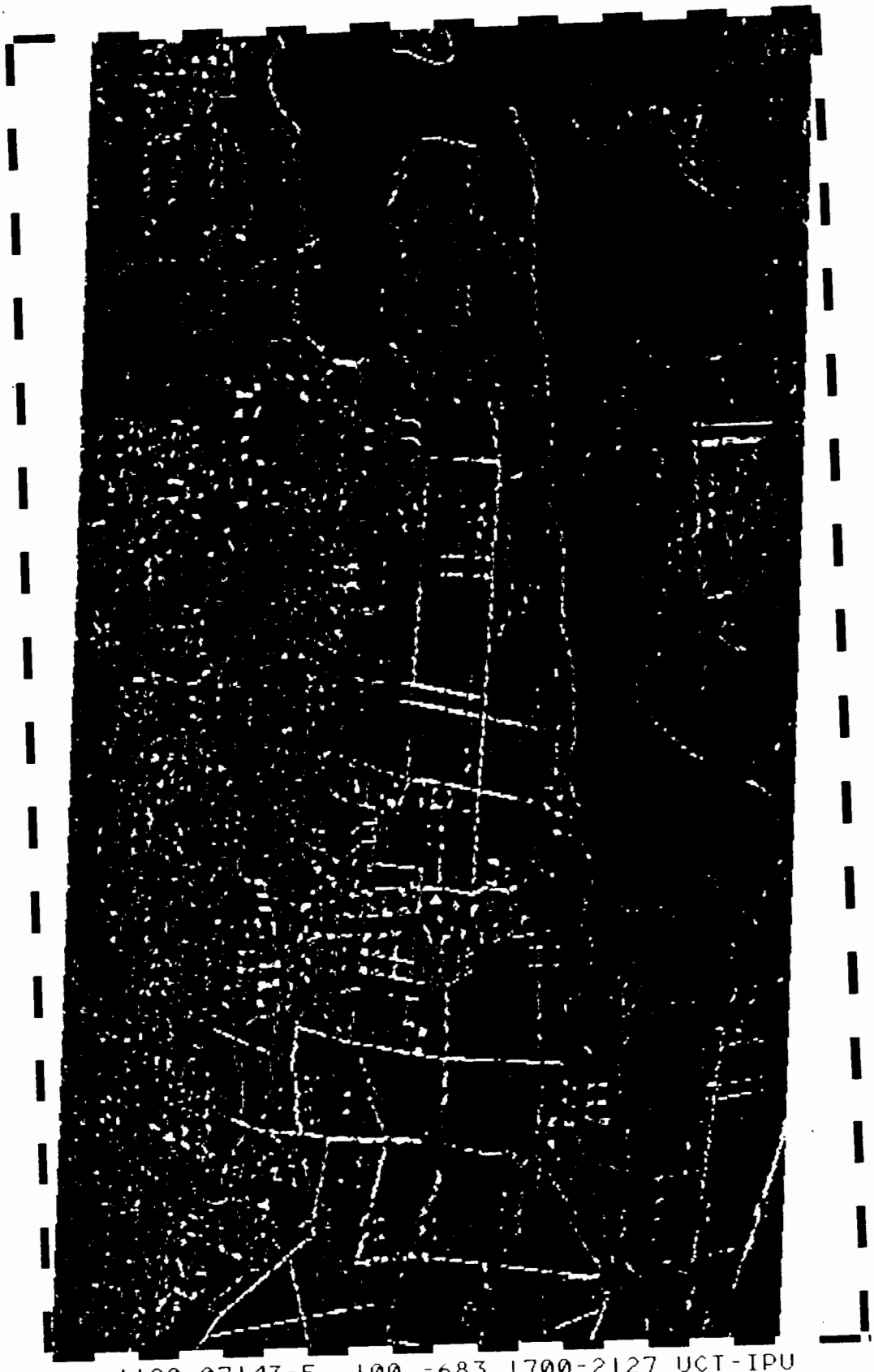


- Figure 19 -

Too Much Edge Enhancement of Figure 17



- Figure 20 -



St. Lucia area, Natal Band 5, edge enhanced

Processing as above as well as LSATIN, TRANS2, and LN2DWT.

$b[k] = A[k]\sin(\Phi[k])$, then equation (3) becomes

$$Y = a[k]\cos(k\theta) + b[k]\sin(k\theta). \qquad (4)$$

The sum of periodic functions of the type given in equation (4) is known as a Fourier series

$$x[\theta] = \sum_{k=0}^{\infty} \{ a[k]\cos(k\theta) + b[k]\sin(k\theta) \}. \qquad (5)$$

Usually, the function in question will be $x[t]$ measured in units of time or space 't'. This is rarely measured in radians so a conversion must be performed. So far θ, which varies between 0 and 2π has been the independent variable, 't' varies between 0 and T (the basic interval) necessitating the conversion

$$\theta = \frac{2\pi t}{T} \text{ radians} \qquad (6)$$

for continuous functions.
For discrete equi-spaced data, the spacing between successive observations is given by

$$\Delta\theta = \frac{2\pi}{n} \text{ radians.} \qquad (7)$$

Any observation may be denoted by a subscript [j]. Therefore

$$\theta[j] = \frac{2\pi j}{n} \text{ assuming } \theta[0] = 0. \qquad (8)$$

Hence, for discrete equi-spaced data, equation (5) can now be rewritten as

$$x[j] = \sum_{k=0}^{\infty} \{ a[k]\cos(2\pi kj/n) + b[k]\sin(2\pi kj/n) \}. \qquad (9)$$

At least two points are required to specify a sinusoidal curve. Because of this, the maximum frequency k_{max} calculable from a sequence of equi-spaced data is n/2 for 'n' even and (n-1)/2 for 'n' odd. Consequently ∞ is replaced by n/2 (or (n-1)/2 for 'n' odd) in equation (9). The calculation of coefficients for $k > k_{max}$ will show that they are periodic with the a's symmetric and the b's asymmetric about k=0 and $k=k_{max}$.

Due to the orthogonality of cos(R2πt/T) and sin(S2πt/T), it can be shown (J.N. Raynor, 1971) that a[k] and b[k] are easily found and have the following values

$$\tilde{a}[0] = \frac{1}{n} \sum_{j=0}^{n-1} x[j] \tag{10}$$

$$a[k] = \frac{2}{n} \sum_{j=0}^{n-1} x[j]\cos(2\pi kj/n) \tag{11}$$

$$b[k] = \frac{2}{n} \sum_{j=0}^{n-1} x[j]\sin(2\pi kj/n). \tag{12}$$

In order to make $\tilde{a}[0]$ comparable with the others, it is frequently defined as being $\frac{1}{2}a[0]$ where

$$a[0] = \frac{2}{n} \sum_{j=0}^{n-1} x[j]\cos(2\pi jk/n),\ k=0. \tag{13}$$

Likewise for n even

$$\tilde{a}[n/2] = \frac{1}{n} \sum_{j=0}^{n-1} x[j](-1)^j. \tag{14}$$

This may be set equal to $\frac{1}{2}a[n/2]$ and

$$a[n/2] = \frac{2}{n} \sum_{j=0}^{n-1} x[j]\cos\left(\frac{2\pi jn/2}{n}\right),\ k=n/2. \tag{15}$$

By McLaurin's expansion, a function y[t] may be expanded as a power series in t. Using this technique, it can be shown that

$$\sin t = i \left\{ \frac{\exp(-it)-\exp(it)}{2} \right\} \text{ where } i=\sqrt{-1}, \qquad (16)$$

and

$$\cos t = \frac{\exp(it)+\exp(-it)}{2}. \qquad (17)$$

Therefore equation (9) can be rewritten as

$$x[j] = \tfrac{1}{2}a[0] + \sum_{k=1}^{n/2} \left\{ a[k]\frac{\exp(i2\pi jk/n)+\exp(-2\pi jk/n)}{2} + b[k]\frac{i\{\exp(-2\pi jk/n)-\exp(i2\pi jk/n)\}}{2} \right\} \qquad (18)$$

and by rearranging

$$x[j] = \tfrac{1}{2}a[0] + \sum_{k=1}^{n/2} \left\{ \frac{a[k]-ib[k]}{2}\exp(i2\pi jk/n) + \frac{a[k]+ib[k]}{2}\exp(-i2\pi jk/n) \right\}. \qquad (19)$$

Now let $c[k] = \frac{a[k]-ib[k]}{2}$, $c[-k] = \frac{a[k]+ib[k]}{2}$, and $c[0] = \frac{a[0]+i0}{2}$ then equation (19) becomes

$$x[j] = c[0] + \sum_{k=1}^{n/2} \{ c[k]\exp(i2\pi jk/n)+c[-k]\exp(-2\pi jk/n) \qquad (20)$$

which is the same as

$$x[j] = \sum_{k=-n/2}^{n/2} c[k]\exp(i2\pi jk/n), \qquad (21)$$

which is the equivalent complex form of equation (5).

The complex Fourier coefficients may be found in a similar way as $a[k]$ and $b[k]$ were previously found.

Therefore

$$c[k] = \frac{1}{n} \sum_{j=0}^{n-1} x[j]\exp(-i2\pi jk/n). \qquad (22)$$

The fast Fourier transform algorithm (Gentleman and Sande, 1966) reduces the number of calculations from a number proportional to n^2 to a number approximately proportional to nlogn. The algorithm is developed from equation (22) with summation over positive frequencies only;

$$nc[k] = \sum_{j=0}^{n-1} x[j]\exp(-i2\pi jk/n). \qquad (23)$$

'n' is then factorized i.e. $n = r_1 r_2 r_3$ and

$j = j_2 + r_2 j_1 + r_1 r_2 j_0$, $k = k_0 + r_0 k_1 + r_0 r_1 k_2$ where k's and j's are integers

$$0 \leq j_0, k_0 \leq r_0 - 1$$

$$0 \leq j_1, k_1 \leq r_1 - 1$$

$$0 \leq j_2, k_2 \leq r_2 - 1.$$

Then equation (23) becomes

$$nc[k_0 + r_0 k_1 + r_0 r_1 k_2] = \sum_{j_2=0}^{r_2-1} \sum_{j_1=0}^{r_1-1} \sum_{j_0=0}^{r_0-1} x[j_2 + r_2 j_1 + r_1 r_2 j_0]$$

$$\times \exp\left[-\frac{i2\pi}{n}(\{k_0 + r_0 k_1 + r_0 r_1 k_2\}\{j_2 + r_2 j_1 + r_1 r_2 j_0\})\right]. \qquad (24)$$

If $\exp\{-2\pi i(\ldots)\}$ is written $\epsilon(\ldots)$, the exponential term expanded and terms of the form $\exp(-2\pi i \times \text{integer}) = 1$ disappear, equation (24) becomes

$$nc[k_0 + r_0 k_1 + r_0 r_1 k_2] = \sum_{j_2=0}^{r_2-1} \left[\sum_{j_1=0}^{r_1-1} \left\{ \sum_{j_0=0}^{r_0-1} \right. \right.$$

$$x[j_2+r_2j_1+r_1r_2j_0] \times \epsilon\left(\frac{k_0(j_2+r_2j_1+r_1r_2j_0)}{r_0r_1r_2}\right)\}$$

$$\epsilon\left(\frac{k_1(j_2+r_2j_1)}{r_1r_2}\right)\Big]\ \epsilon\left(\frac{k_2j_2}{r_2}\right) \quad . \tag{25}$$

This is a nested series of discrete Fourier transforms. Upon completion, the result obtained is $c[k_2+r_2k_1+r_1r_2k_0]$ which has reversed subscripts. A final step involves unscrambling the coefficients (Rabiner and Gold, 1975).

It is quite easy to go from the one dimensional Fourier transform (as above) to the more useful two dimensional transform. If the transform is denoted by an N×N matrix 'F' and 'x' and 'y' are column vectors of length 'N', then y, the F transform of x is simply

$$y = Fx. \tag{26}$$

Now if $X = (x_1,x_2,\ldots,x_n)$ and $Y = (y_1,y_2,\ldots,y_n)$ where x_i and y_i are column vectors as before,

$$y_i = Fx_i \tag{27}$$

From this it is obvious that

$$Y = FX, \tag{28}$$

i.e. Y is the column transform of X. Transposing, and repeating the operation gives

$$Y' = FY^T$$
$$Y' = F(FX)^T. \tag{29}$$

Transposing again, we have the two dimensional F transform of X is

$$Y'' = FXF^T. \tag{30}$$

Hence, the two dimensional transform can be calculated by first performing one dimensional transforms on the rows, and then the columns, or vice versa (Hepburn, 1975).

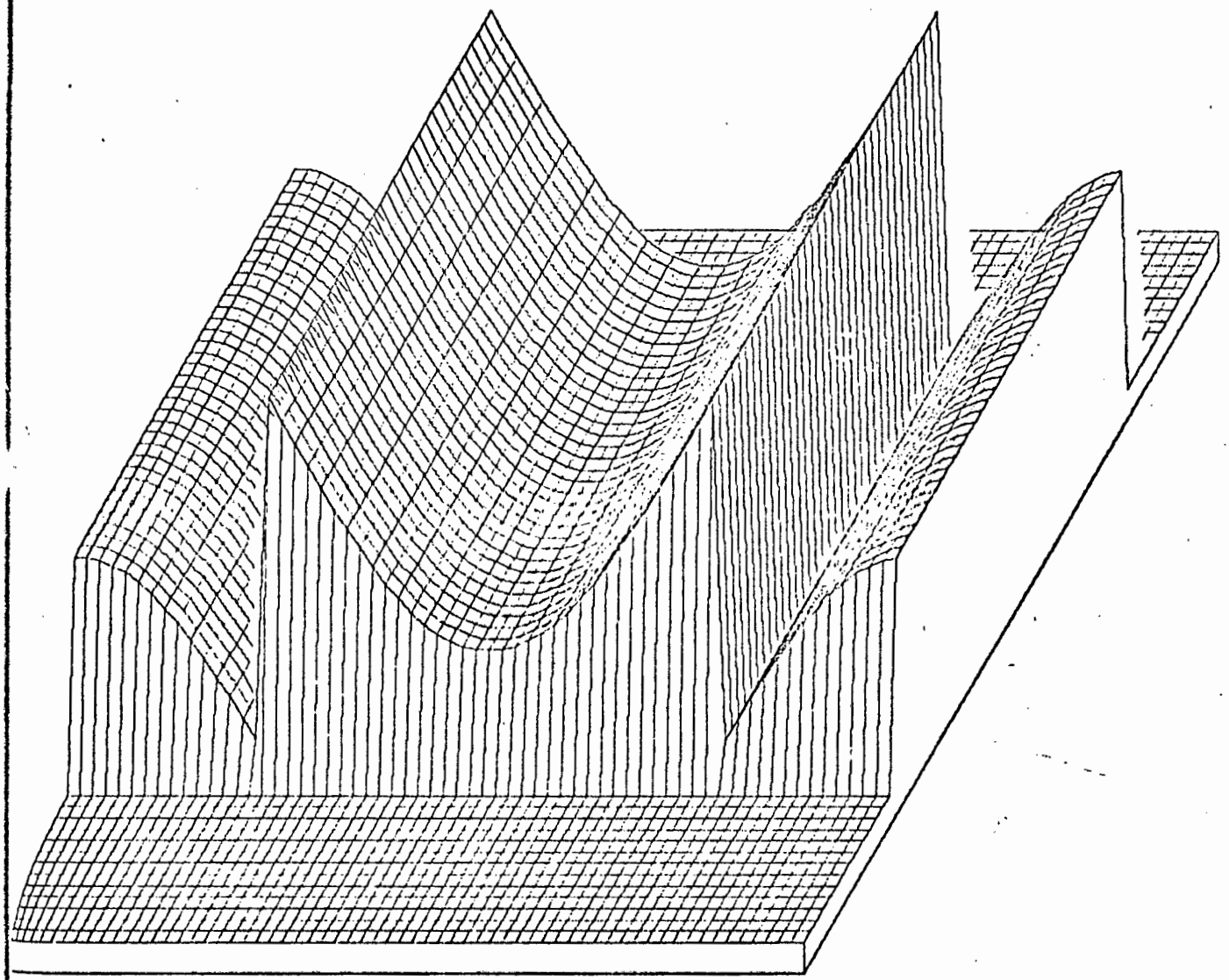
The routines that actually do the Fourier transform have been taken from Applied Statistics (1975) as published by D.M. Monro. The subroutine names have been altered and a few comments have been added, otherwise they remain unchanged. These routines are FASTF (now called FFTDRV), FASTG (now called FOURIE), and SCRAM (now called UNSCRM). FFTDRV is called by the application task as a link to the actual Fourier transform and unscrambling subroutines. FOURIE does the actual Fourier transform (D.M. Monro, 1975). UNSCRM is called by FFTDRV in order to unscramble the coefficients prior to return to the application task. Since they are a computer implementation of the contents of 5.3.3.1, there will be no further discussion here.

5.3.3.2 One Dimensional Transforms

A one dimensional Fourier transform is accomplished using the application task TRANSF. This routine transforms each line in the input image, weights it according to the user's wishes and outputs the line again (see Figure 21 for a one dimensional edge enhancement of Figure 15). Upon entry to TRANSF, GPARAM is called. The line length is set to the power of two that is greater than the number of samples (NS) as the transform only works with line lengths that are powers of two. If the number of samples is equal to a power of two, the line length is equal to the number of samples. If the number of samples is greater than 4096, the line length is set to 4096. The difference between the line length and the number of samples is computed (see Figure 22).

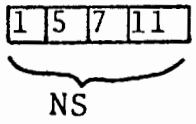
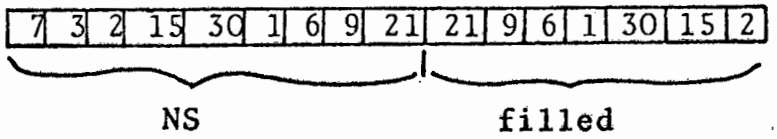
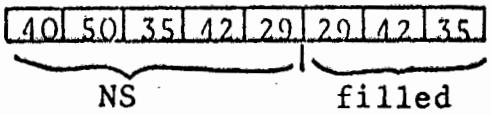
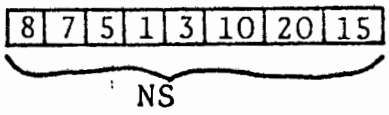
The local parameters are obtained from LPARAM and the transforming begins. A line of input is read and transferred to the transform buffer. If the difference between line length and number of samples is greater than

A One Dimensional Edge Enhancement of a Box (Figure 15)



- Figure 21 -

Number of Samples and Line Length for a One Dimensional Transform

NS	Line Length	Image of Line
4	4	
9	16	
5	8	
8	8	

The transform is done on the line of length 'line length', the output however upon retransformation is chopped back down to length 'NS'.

- Figure 22 -

zero, the extra positions become a mirror image of the end of the line (see Figure 22). FFTDRV is called to perform the transform. Upon return, a keyword is examined to determine the type of weighting required on the transform. If it is to be an edge enhancement, the specified number of low frequency components are removed from the real and imaginary parts of the transform. If a weighting array is required, LN1DWT is called.

When the transform has been manipulated, FFTDRV is called once again to perform the reverse transform. The real part of the result is then written to the file.

Once all the lines in the image have been processed, LABCHK is called to check the system label, and control returns to MAIN.

As seen, TRANSF calls a weighting array generating routine called LN1DWT. It generates an array to multiply the transforms by in order to change the appearance of the image upon retransformation. The function used to compute the weights is

$$e^{-x/P_2},$$

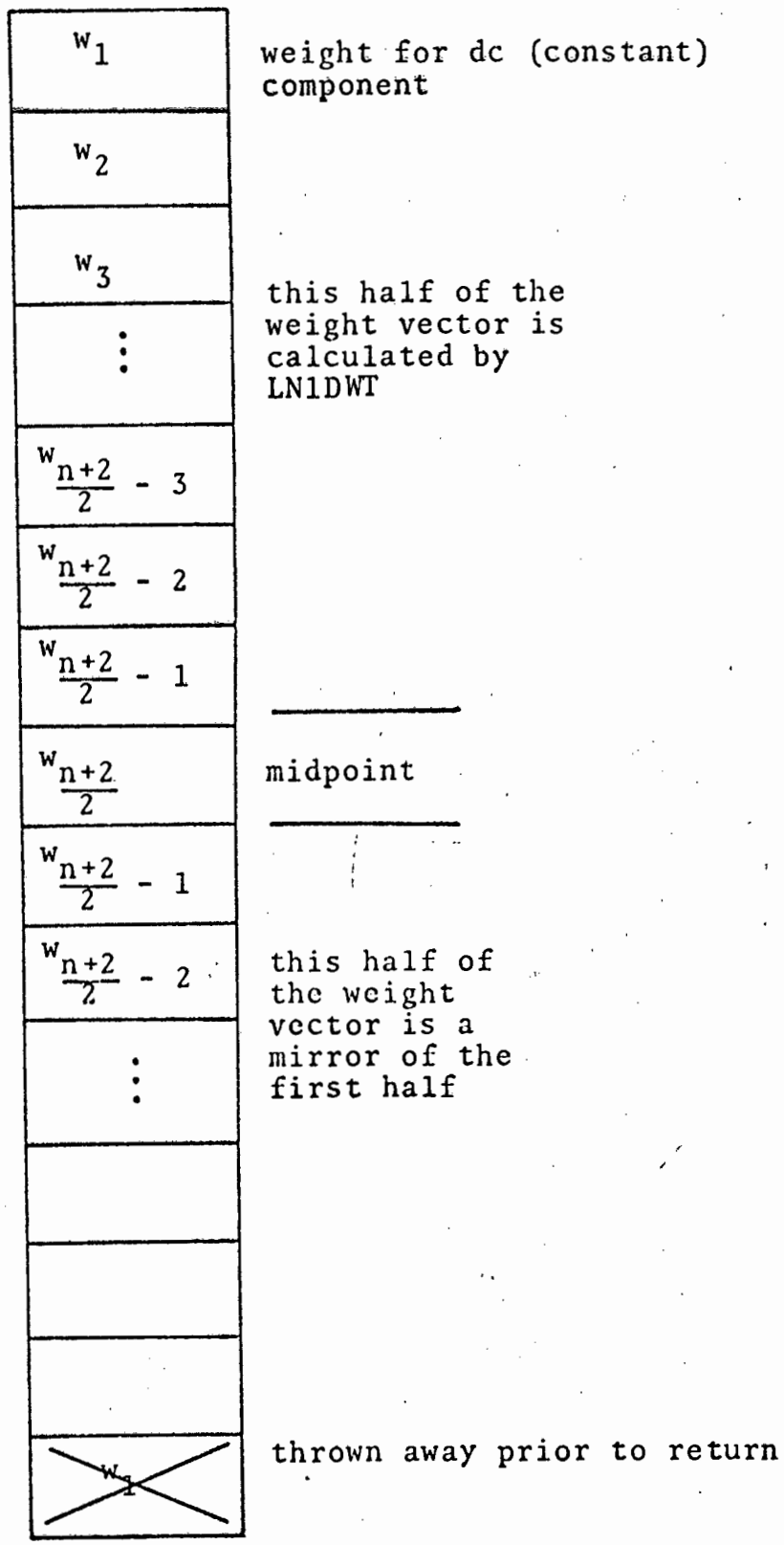
where $x=(I-1)^2$, I is the frequency and P_2 is an input parameter.

Due to the symmetry of the transform, weights are calculated for the first half of the array. The latter half of the weight matrix becomes a mirror image of the calculated portion (see Figure 23). Upon completion of the weight calculation, control returns to TRANSF, with it.

5.3.3.3 Two Dimensional Transforms

The two dimensional Fourier transform is divided up into two tasks. One does the transform and outputs it to file. This allows the user to do anything he desires

Calculation of a One Dimensional Weight Vector



- Figure 23 -

to the transforms prior to retransforming. The task which accomplishes this is TRANS2.

The two dimensional Fourier transform has a maximum allowable size of 1000×1000. The minimum allowable is 4×4. Due to core considerations, the maximum in core size allowed is 64×64. This means that > 64×64 cannot be executed in core. To get around this problem, the transforms are executed in 64×64 segments (see Figure 24). These segments are output to two files, one for the real part and one for the imaginary part of the transform.

Upon entry to TRANS2, after calling GPARAM and LPARAM, the size of the area to be transformed is checked. Areas > 64×64 are broken down into 64×64 segments with the appropriate overlap calculated to bring the size of the transform down to NL×NS or less.

A new statistics label is created (see Figure 10). This label is added to the output files via LABADD. It is added so that the weighting task and retransforming task can identify the size, overlap and type of file (real or imaginary).

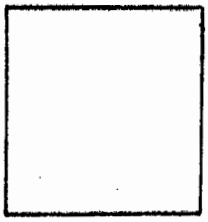
If the area to be transformed is looked upon as an NL×NS array, it is divided into the requisite number of segments of 64. If NL×NS is 64×64 or less, one segment only is transformed (see Figure 24).

The transform section begins here. The required amount of data is read in and transferred to the transform matrix. FFTDRV is called to transform the area line by line. Upon return, the transform is transposed and FFTDRV is recalled to perform the second transform.

Upon completion of the transform, the real and imaginary parts of the transform are output to seperate files for later processing. If the area consists of more than one segment, the process is repeated for each segment in turn.

In the case of overlap between segments, an amount of overlap is calculated and the larger of this calculation

Method of Implementing a Two Dimensional Fourier Transform using Segments



size: 4×4 → 64×64
done as one segment.

size: > 64×64
example: 256×256 divided up into 16 segments.
The segments are transformed in the order 1,2,3, ... ,13,14,15,16.

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

- Figure 24 -

and an overlap parameter is taken. This overlap area is a common area processed by both segments. For overlap = P_1, this common area is of size $(P_1-1)*2$. Upon retransforming of the picture, P_1-1 from the edge of each segment is discarded (see Figure 25). This overlaying is done to avoid boundary effects at the edges of the segments as much as possible.

Upon completion of all transforms, LABCHK is called twice to validate the system label on the real and imaginary transform output files and control then returns to MAIN in the NUCLEUS.

The routine which weights the transforms as calculated above is called LN2DWT. The weighting function used for the calculation of the smoothing matrix is

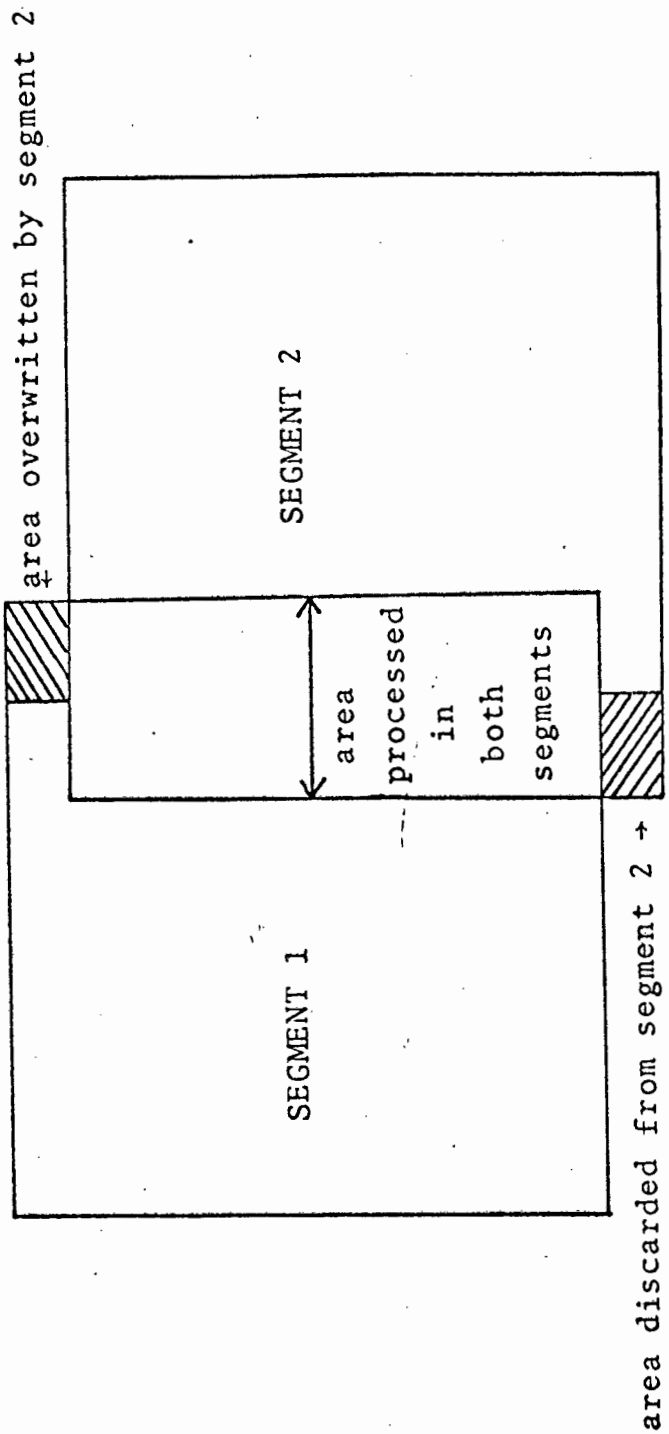
$$e^{-x/P_2},$$

where $x=(I-1)^2+(J-1)^2$, I and J are the frequencies in the x and y directions and P_2 is an input parameter. The algorithm used for calculating the matrix was taken from Techniques in Image Enhancement and Restoration (Hepburn, 1975). It has been generalized for sizes from 4×4 to 64×64.

Upon entry, the global and local parameters are obtained via GPARAM and LPARAM. LREAD is then called in order to locate the latest FOURIE statistics label. Once found, the Fourier information is extracted prior to computation. However, if this statistical data from the real file does not match that from the imaginary file, error messages are output to the printer and the job is halted.

Once the files have been validated, the data extracted from the label is used to determine the size of the weight matrix. If smoothing is desired, the above function is calculated into the matrix. The transforms are then multiplied by the weight matrix, point by point. Upon completion, the segments are written out to their appropriate files (real/imaginary) and the process is repeated for all segments.

Diagram of an Overlay Area



- Figure 25 -

In the case of edge enhancement, a weight matrix is not calculated, but the required number of low frequency components are removed symmetrically from around 'dc' (the constant) in the real and imaginary transforms (see Figures 26 and 27). As each segment is processed, it is output to the proper file until all segments of the image have been processed.

Finally LABCHK is called to validate the system labels on the two output files. Once this has been accomplished, control returns to MAIN in the NUCLEUS.

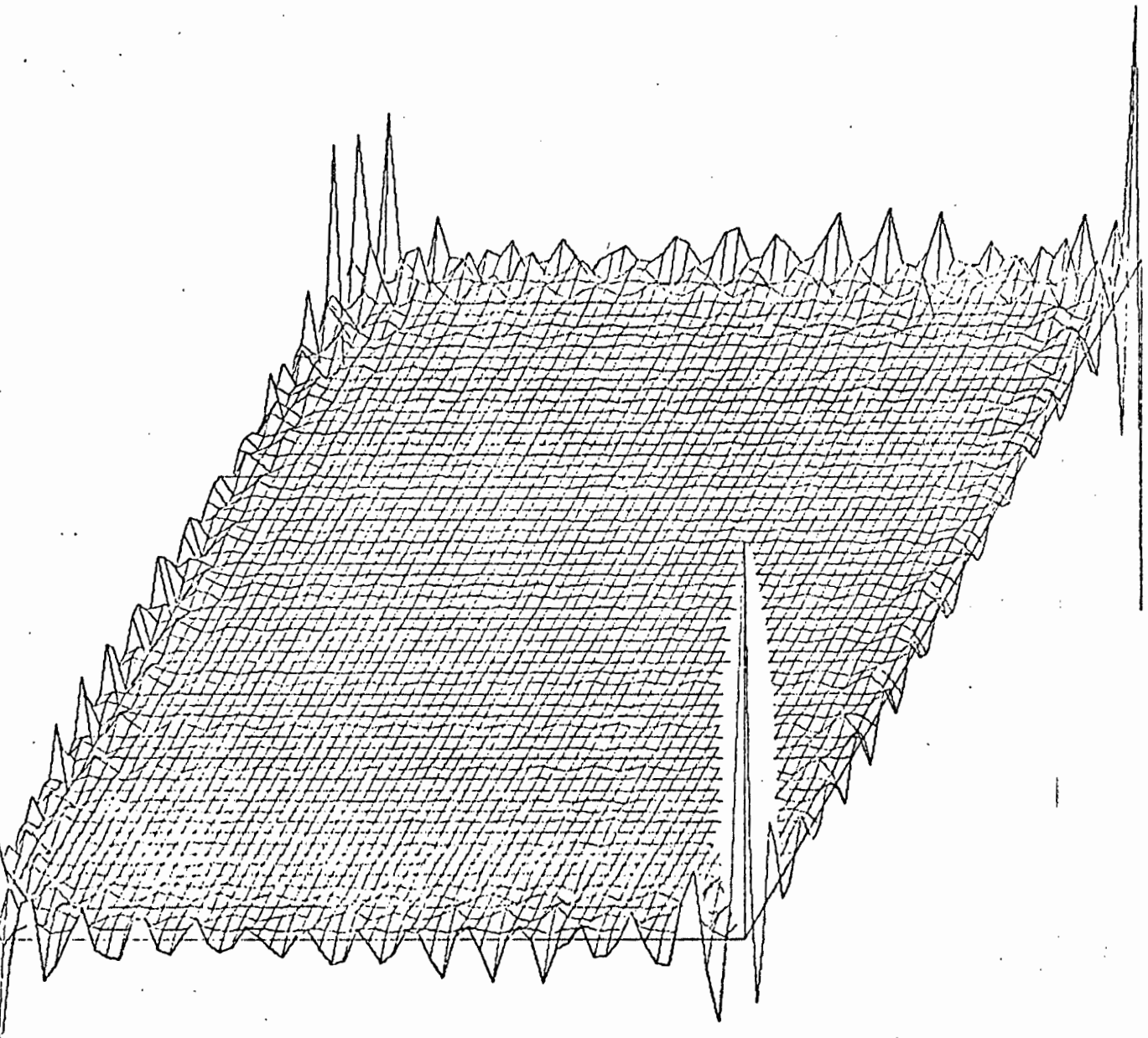
The image is reconstructed (retransformed) by the application task RETRAN. Upon entry to the routine, the global and local parameters are obtained via GPARAM and LPARAM. LREAD is then called in order that the latest FOURIE label is found. The information is extracted from this label in order to determine the size of the transform. Once all the data is extracted, it is encoded into a message for output to the printer describing the size, number of segments, overlap and the initial input to the transform.

The two input files are checked for compatability with respect to size, overlap, number of segments and input file. If these do not match, an error message is printed out, and the run is halted. The files are then tested to see that the types are correct - primary input is the real transform part, and secondary input the imaginary part. If this is incorrect, an error message is printed and the run is halted.

The retransforming begins at this point. A segment is read in from the real file and from the imaginary file. The segment is transformed via a call to FFTDRV, it is transposed and then transformed again. When only one segment is present, LABCHK is called, and control returns to MAIN.

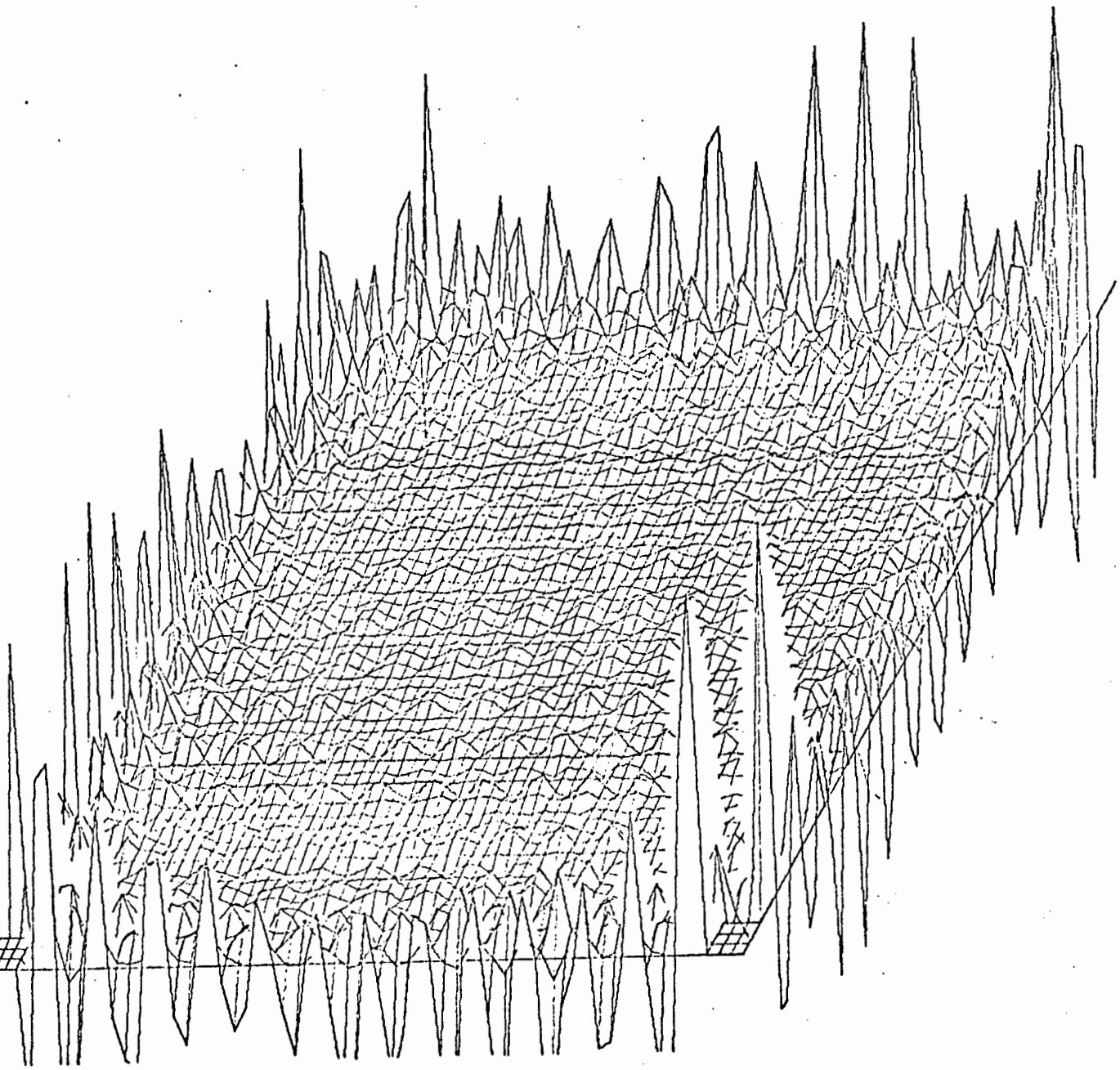
When there is more than one segment present, a scratch file is used for intermediate output. In the segmentation

A Fourier Transform



- Figure 26 -

Some Low Frequency Components Removed
from Figure 26



- Figure 27 -

process, the line number changes more rapidly than the sample numbers. Therefore, a swathe down the image is processed (see Figure 24). These swathes are output to the scratch file with a record length equal to the image size. The first set of segments (swathe 1) is output. During the processing of the second set of segments (swathe 2), assuming it is not the last set, the first segment on the scratch file is re-read, the new segment is added onto the end and it is rewritten to the scratch. When the last set of segments is reached, the segments written to the scratch file are re-read, the last segment is added at the end and the result is written to the output file (see Figure 28). In between each segment processing, the start line or start sample is calculated to accomodate the overlap. The overlap is also important during output as part of each overlaying segment is discarded.

Upon completion, LABCHK is called to validate the system label and a number of messages are printed out. Following successful execution of the task, control returns to MAIN in the NUCLEUS.

Use of Scratch Files in RETRAN

Assume an image of 256×256 with no overlap.
Record = 256.

SCRATCH FILE

OUTPUT FILE

pass 1:

segment 1	empty
2	empty
3	empty
4	empty

empty

pass 2:

1	5	empty
2	6	"
3	7	"
4	8	"

empty

pass 3:

1	5	9	empty
2	6	10	"
3	7	11	"
4	8	12	"

empty

pass 4:

1	5	9	empty
2	6	10	"
3	7	11	"
4	8	12	"

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

- Figure 28 -

6 Conclusion

The viability and scope of the U.C.T. Digital Image Processing System should now be apparent to the reader. A large degree of success has been met in designing DIPS and it currently has a number of interested users. To name one, the botanists are highly interested in the system for its pattern recognition routines. A great degree of success has been obtained in producing classified images which correspond almost exactly to aerial photographs and topographical maps of the same areas.

The NUCLEUS as seen is a complex, multipurpose group of interrelated routines required for the convenience of the application task. It takes care of the 'image capture' area of the processing in that it makes all files initially available to the run and all further I/O and file handling comes into its realm.

The routine DISPLY as described is the only image display routine that has been mentioned. This routine has proved invaluable in the development of the system as it is a quick and easy way of getting a visual picture of what has happened during a test run. DIPS, however, contains a number of other display routines. An example is OPTOUT which prepares an image for the process of digital → picture negative conversion. Once the negative has been produced the image can be printed for human viewing.

In the area of image enhancement there is still scope for further development. Now that the basic Fourier transform and retransform tasks are developed, it should be fairly easy to add new filtering and weight routines which come between these two facilities. Currently images can only be smoothed or edge enhanced but as DIPS is a developing system there are plenty of ideas for new ways of making an image more acceptable to the human eye.

A great deal of success has been achieved on the portability front. Previous known systems of this kind have all been designed in low level languages making them

very machine dependent. DIPS on the other hand is written mostly in FORTRAN which is a relatively universal computer language making for a great deal more mobility. Naturally all installations introduce their own quirks to the system which would have to be changed upon relocation. Also, a number of the supporting routines have been written in UNIVAC assembler which is non-portable. This has been kept to a minimum however.

I/O is also a very machine dependent aspect of a system even in a high level language. To deal with this, all I/O has been limited to three routines obviating the need for extensive search for I/O statements throughout the sytem.

The problem of job control language is easily handled in DIPS via a UNIVAC innovation, ERTRAN, which provides the linkage from FORTRAN to EXEC 8 for the purpose of executing executive requests. This area of the system is obviously very machine dependent and would have to be altered to its equivalent form (if it existed) upon change to another system. These requests are limited to a small number of routines in order to make the job of conversion as painless as possible.

DIPS is still a young growing system but already a strong framework has been constructed to allow ennumerable application modules to be added. Such a system will provide a suitable vehicle for research into many different aspects of image processing.

References

BERNSTEIN R. — Digital Image Processing of Earth Observation Sensor Data, I.B.M. Journal of Research and Development, Vol. 20, No. 1, Jan. 1976

COSMIC — JPL Digital Image Processing System Manual, Computer Software Management and Information Center, University of Georgia, 1968

GENERAL ELECTRIC — ERTS Reference Manual, Space Division, Valley Forge Space Center, Philadelphia (year unknown)

GENTLEMAN W.M., SANDE G. — Fast Fourier Transforms-For Fun and Profit, 1966 Fall Joint Computer Conference, AFIPS Proc., 29: 563-575 (1966)

HEPBURN J.S. — Techniques in Image Restoration and Enhancement, Masters Thesis, U.C.T., 1975

HEPBURN J.S., SWINGLER D.N. — Image Processing Using Coherent Optical and Digital Techniques of Value in Med-Diagnosis, South African Journal of Science, Vol. 71, April 1975

MACGREGOR K.J., JACKSON A.A., MIKETINAC J. — A Portable Image Processing System, Program 77, Computer Society of South Africa, Sept. 1977, (conference paper)

MONRO D.M. — Complex Discrete Fast Fourier Transform, Applied Statistics, Vol. 24, No. 1, 1975

RABINER L.R., GOLD B. — Theory and Application of Digital Signal Processing, Englewood Cliffs, London, Prentice Hall, 1975

RAYNOR J.N. — An Introduction to Spectral Analysis, London: Pion., 1971

REEVES R.G. (ed.) — Manual of Remote Sensing, Volume 1, The American Society of Photogrammetry, 1975

Appendix A
Application Tasks

GEN	generates a picture given an initial value and a horizontal and vertical increment.
GEN2	same as GEN but gives modulo x.
AVERAG	reduces a picture by averaging.
XPAND1	enlarges a picture N times by repeating pixels.
CONCAT	creates one composite picture from up to 10 input pictures each of the same size.
NOAAIN	inputs a NOAA picture or part thereof.
PCTRAN	principal components transform of an image.
SIGNAT	generates multivariant normal signatures from input training sets.
LIST	lists the pixel values in the specified area of an image.
RATIO	calculates ratio of two or more images.
STRECH	used to change the pixel DN values of a picture by generating a transfer function on the domain of input values.
OPTOUT	writes an image to tape in optronix format.
MAPCLS	prints out classification map of a classified image.
SKEW	corrects image for earth rotation and pixel aspect ratio.
LSATIN	inputs a LANDSAT image or part therof.
COPIN	partially processed image input from tape.
OUTAPE	partially processed image output to tape.
OPTIN	reads in an image which has been digitised on the optronix.
SCRIBE	scribes rectangles around specified areas in an image.
CLFRBW	frames and adds annotation to a classified image.
MAXLIK	maximum likelihood classification.
TRANSF	one dimensional Fourier transform of an image.
LN2DWT	two dimensional Fourier transform weight function.
DISPLY	line printer display routine.

TRANS2	two dimensional Fourier transform of an image.
RETRAN	two dimensional Fourier retransform of an image.
FRAME	frames and adds annotation to an image.
DIVIDE	divide all the values in the specified part of the image by the input parameter to bring the image in range of 0 - 255.
PPCLAS	modified parallelipiped classification.
NNCLAS	nearest neighbour classification.
TRNTX1	develop training sets using nearest neighbour rule with merge/elimination on overlaps.
TRNTX2	develop training sets using nearest neighbour rule with cast system for overlaps.
TXCLUS	develop hierarchical classification dendrogram of a set of pixels.
ADCLUS	adaptive clustering of an image.
ITCLUS	iterative clustering of an image.

Appendix B
Error Messages

LPARAM:

1) LPARAM REAL BUF TOO SMALL
2) NO PARAMETER CARD(S)

LABADD:

1) **NO END OF LABEL SYMBOL FOUND

WRITE:

1) TOO MANY SCRATCH FILES
2) RECORD NUMBER NOT SPECIFIED FOR SCRATCH FILE
3) NEGATIVE SKIPPING ENCOUNTERED
4) PACKING BUFFER TOO SMALL
5) I/O ERROR STATUS WORD = _____

READ:

1) ATTEMPT TO READ FROM EMPTY SCRATCH FILE-_____
2) ATTEMPT TO READ PAST EOF IN ROUTINE _____
3) READ BUFFER TOO SMALL
4) ATTEMPT TO PASS LONGER LINE THAN THAT READ-_____
5) NEGATIVE SKIPPING ENCOUNTERED
6) UNPACKING BUFFER TOO SMALL
7) I/O ERROR STATUS WORD = _____

MAIN:

1) TOO MANY INPUTS
2) TOO MANY OUTPUTS
3) APPLIC. ROUTINE NOT FOUND
4) NO TASK CARD FOUND
5) NO INPUT CARD(S) FOUND
6) NO OUTPUT CARD(S) FOUND

SCRIBE:

1) PARAMETER 2 ILLEGAL VALUE (must be in range 0-5)

DISPLY:

1) ***ERR-ENDSAMP LT STARTING SAMP
2) ***ERR-ENDLINE LT STARTING LINE

TRANSF:

1) *ERR-ENDSAMP LT STARTING SAMP

2) ***ERR-ENDLINE LT STARTING LINE

3) TRANSF-PARAM2 INVALID) make parameter positive and
4) TRANSF-PARAM3 INVALID non-zero.

TRANS2:

1) ***SIZE ERROR IN 2D FOURIER

area of transform greater than 1000×1000 or less than 64×64 but not a power of two i.e. 4×4, 8×8, 16×16, 32×32.

RETRAN:

1) **DIFFERENT SIZE FILES FOR REVERSE TRANSFORM

real and imaginary files do not match in size.

2) INCOMPATABLE FILES FOR TRANSFORM

primary input is not real or secondary is not imaginary part of transform.

3) _____ VALUE OVER 512

meaningless image as there is overflow when packing.

LN2DWT:

1) LN2DWT-PARAM2 INVALID) make parameter non-zero and
2) LN2DWT-PARAM3 INVALID positive.
3) **DIFFERENT SIZE FILES FOR WEIGHTING) as for RETRAN
4) INCOMPATABLE FILES FOR WEIGHTING

COPIN:

1) BUFFER TOO SMALL IN COPIN
2) I/O ERROR STATUS WORD = _____
3) ERROR IN TAPE MOVE STATUS=_____

OUTAPE:

1) BUFFER TOO SMALL IN OUTAPE
2) I/O ERROR STATUS WORD = _____
3) FILE TOO LARGE FOR TAPE
4) ERROR IN TAPE MOVE STATUS=_____

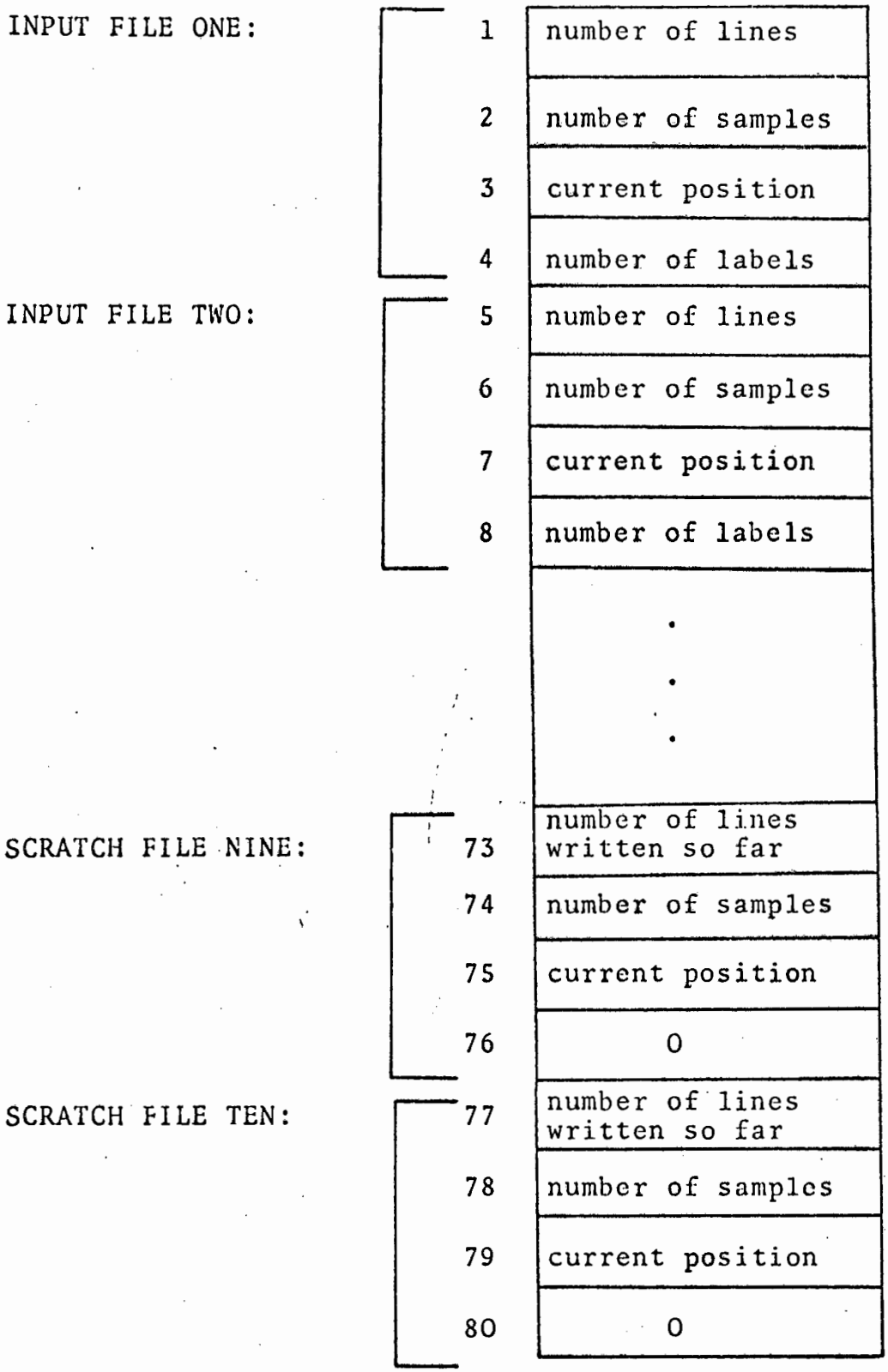
Appendix C
Programming Specifications for the NUCLEUS

To facilitate communication between the various routines, a number of common data areas have been used. These areas, there purpose and the routines they appear in are as follows:

(1) BLANKCOMMON containing a ten word array IGLBL common to MAIN, GPARAM and LABCHK. This common block is set up in the MAIN routine as part of the preparatory work prior to entry to the application routine. IGLBL is set up from information taken from the task card and the primary system input label. GPARAM needs the information in order to pass it over to the application routine and LABCHK needs it to check whether the output file actually contains the amount of information that IGLBL presumes it contains.

(2) Labelled common CHECK containing RDTABL(80), (see Figure 29) and PROCES, common to MAIN, WRITE, READ, LREAD and REWIND. The array RDTABL is set up for the input files in the MAIN routine. Each input file is allocated 4 elements of the array. The primary input uses RDTABL(1)-(4), input 2 uses RDTABL(5)-(8) etc.. The first entry for an input is the number of lines, the second entry is the number of samples per line, the third entry is the current line number ready for input, and the fourth entry is the number of physical labels on the file. Allowing for 10 inputs, this uses up the first 40 array elements. The last 40 are initiated by WRITE in the event of any scratch files being used (up to 10 allowed). This data is used by READ to ensure that the proper record is going to be read next, forsee an end-of-file condition and to read past labels in the event of a rewind. The only one of these four entries which is changed is the third one, indicating the current position in the file. LREAD accesses the third entry and returns it to zero as it repositions the file to the beginning again prior to label reading.

RDTABL



- Figure 29 -

REWIND likewise changes the current position (for scratch files) to zero. PROCES is the name of the application task currently active. It is kept by all these routines, but used only by READ and WRITE in case of file errors, the routine in error is immediately known.

(3) Labelled common LABELS containing LABREC(60) common to MAIN, LABCHK, OUTAPE and COPIN. (These last two are application routines). This array is the area into which the labels are placed as they are read. The reason for making this a common area was to save storage space as none of the above routines use the space at the same time.

(4) Labelled common READER containing W(2), PCARD(15) and ICARD common to MAIN, LPARAM and MYTAPE. W is a format statement for reading in an 80 column card in A-format (13A6,A2). PCARD is the buffer into which the card is placed starting at PCARD(2). PCARD(1) is used for print control when the card is printed. ICARD is the unit number from which the cards are read. At present this is set to 7. MAIN uses this area to read in the task, input and output records, LPARAM uses it to read the parameter records and MYTAPE uses it to read the tape catalogues. MYTAPE, however, does not use ICARD.

(5) Labelled common SCRTCH containing POINT(10), STCARD(4) and NAME(10) common to MAIN, WRITE and READ. POINT is a logical array, each element corresponding to a scratch file. As scratch files are assigned, POINT elements are set to .TRUE.. If only one scratch file is in use POINT(1)=.TRUE. and POINT(2)-(10)=.FALSE.. The array is initialized to .FALSE. in MAIN at the beginning of each new application task. Only when a scratch file is required via a WRITE call, is it assigned and the POINT element set to .TRUE.. STCARD is the buffer used for the ERTRAN call to assign the file. NAME is an array of ten scratch file names to be assigned temporarily as needed. These names are SCTR01-SCTR10.

(6) Labelled common ERROR containing MSG(9) common to WRITE, READ, COPIN and OUTAPE. These are used in the event of a read or write error. The NTRAN error code, the unit number and the record number (where available) are encoded into the buffer (for all of the above). As well, in COPIN and OUTAPE, there is FORTRAN I/O which on error inserts the I/O status word into the buffer (in decimal requiring a conversion). These error conditions can be found in 'SPERRY UNIVAC 1100 SERIES PROGRAMMER REFERENCE', volume 15.

This next section deals with each routine from the programming standpoint. The calling sequence, calling parameters and their meaning are all given in detail. It is hoped that this will serve as documentation for the NUCLEUS of the system. Along with this documentation, a flowchart for each routine is provided.

1) The Dummy Routines:
   a) CALL OPEN (P_1,P_2,P_3,P_4,P_5,P_6) where P_1-P_6 are dummies,
   b) CALL ERROR - no parameters, and
   c) CALL END - no parameters.

2) The Parameter Processing Routines:
   a) CALL LPARAM (P_1,P_2,P_3,P_4) where

      P_1 - error indicator (not used),
      P_2 - the array into which the parameters are to be placed,
      P_3 - number of parameters to be read,
      P_4 - type of parameter indicated as follows:
         0 - integer,
         1 - real,
         2 - character.

   b) CALL GPARAM (P_1,P_2) where

      P_1 - error indicator (not used),
      P_2 - 10 word array to place the parameters into.

3) The Label Processing Routines
   a) CALL LABCHK (P_1,P_2,P_3,P_4) where

P_1 - error indicator (not used),
P_2 - file number from the application task,
P_3 - a 6 word array containing information to be inserted into the system label (see below),
P_4 - an indicator for the type of checking to be carried out,
0 - global checking - P_3 contains SL, SS, NL, NS as on the output file in locations one to four.
1 - band/date/calibration insertion into words 7 and 8, P_3 contains these items in locations five and six.
-1 - both of the above, P_3 contains both of the above.
2 - latitude and longitude insertion into words 9 and 10, P_3 contains the latitude and longitude in the first two locations.

b) CALL LREAD (P_1,P_2,P_3,P_4) where

P_1 - error indicator (not used),
P_2 - file number from the application task, this number varies from 1-10 for input files, and 21-30 for output files,
P_3 - number of words to be read,
P_4 - buffer where the input is to be placed.

c) CALL LABADD (P_1,P_2,P_3,P_4) where

P_1 - error indicator (not used),
P_2 - output file number,
P_3 - number of labels to be added,
P_4 - vector of length P_3*12 containing labels to be added.

) <u>The Print Routines</u>

a) CALL SYSMSG (P_1,P_2,P_3) where

P_1 - not used,
P_2 - the message to be printed, either in an array or in the argument list in quotes (must be in character format),

P_3 - number of characters in the message including carriage control in the first position in the message.
Possible carriage control characters are
blank - space 1 line
0 - double space
+ - suppress spacing
1 - skip to top of next page.

b) CALL PRINT (P_1,P_2,P_3,P_4) where

P_1 - error indicator,
non-zero - indicates an error,
zero - no error condition exists.
P_2 - not used,
P_3 - number of characters in message,
P_4 - message buffer or message in quotes including carriage control (must be in character format).

5) The Data Formatting Routines

a) CALL PACK (P_1,P_2,P_3,P_4,P_5) where

P_1 - error indicator (not used),
P_2 - number of unpacked words,
P_3 - number of packed words = (P_2+3)/4,
P_4 - start of the unpacked array,
P_5 - the array where the packed result is to be stored.

b) CALL UNPACK (P_1,P_2,P_3,P_4,P_5) where

P_1 - error indicator (not used),
P_2 - starting point of unpacking in P_4,
P_3 - number of words in P_4,
P_4 - array of input to be unpacked,
P_5 - unpacked result, P_5 should be of length P_4*4.

6) The File Handling Routines

a) CALL SETFIL (P_1,P_2,P_3) where

P_1 - error indicator (not used),
P_2 - a 4 word array - the file name is passed in word 3 of the array (without a '.' following it),
P_3 - the unit number to be assigned to the file.

b) CALL REWIND (P_1,P_2,P_3) where

P_1 - error indicator (not used),
P_2 - file number of file to be rewound,
P_3 - type of file,
-1 - scratch file,
0 - input file,
1 - output file.

7) The Input/Output Routines

a) CALL READ (P_1,P_2,P_3,P_4,P_5,P_6,P_7,P_8) where

P_1 - error indicator (used for formatted input, tape catalogue, to indicate EOF has been reached in which case it returns to caller with a value=3),
P_2 - file number from caller,
P_3 - record to be read, zero indicates read the next in sequence,
P_4 - 0,1 - image file input, packed, unformatted,
2 - non-image input, unpacked, unformatted,
other - formatted input from cards or tape catalogue, in this case P_4 is the format for reading the data,
P_5 - number of elements to be skipped at the beginning of a record,
P_6 - number of words to be input,
P_7 - input buffer,
P_8 - 60 - label input,
other - indicates file input, the type determined by P_4.

b) CALL WRITE (P_1,P_2,P_3,P_4,P_5,P_6,P_7,P_8) where

P_1 - error indicator (not used),
P_2 - file number from caller,
P_3 - record to be written (for scratch files only) not used for ordinary output files,
P_4 - 0,1 - image file output, packed, unformatted,
2 - non-image file output, unpacked, unformatted,

other - formatted output,

P_5 - number of records to be skipped at the beginning of the output,

P_6 - number of words to be output,

P_7 - output buffer,

P_8 - 60 - label output,

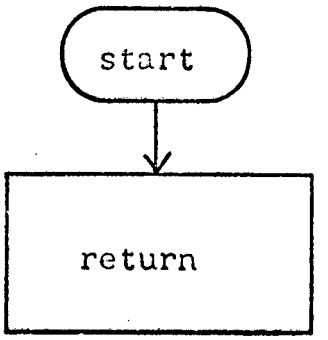
other - file output as indicated by P_4.

NOTE: All the above parameters are integer.

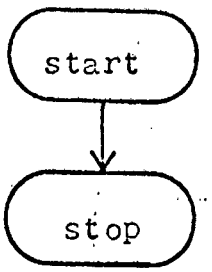
To obtain the runstreams for the application tasks see Appendix D.

Flowcharts for OPEN, ERROR and END

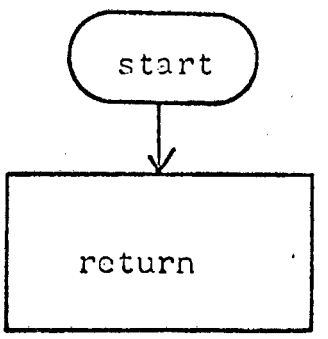
Subroutine OPEN



Subroutine ERROR

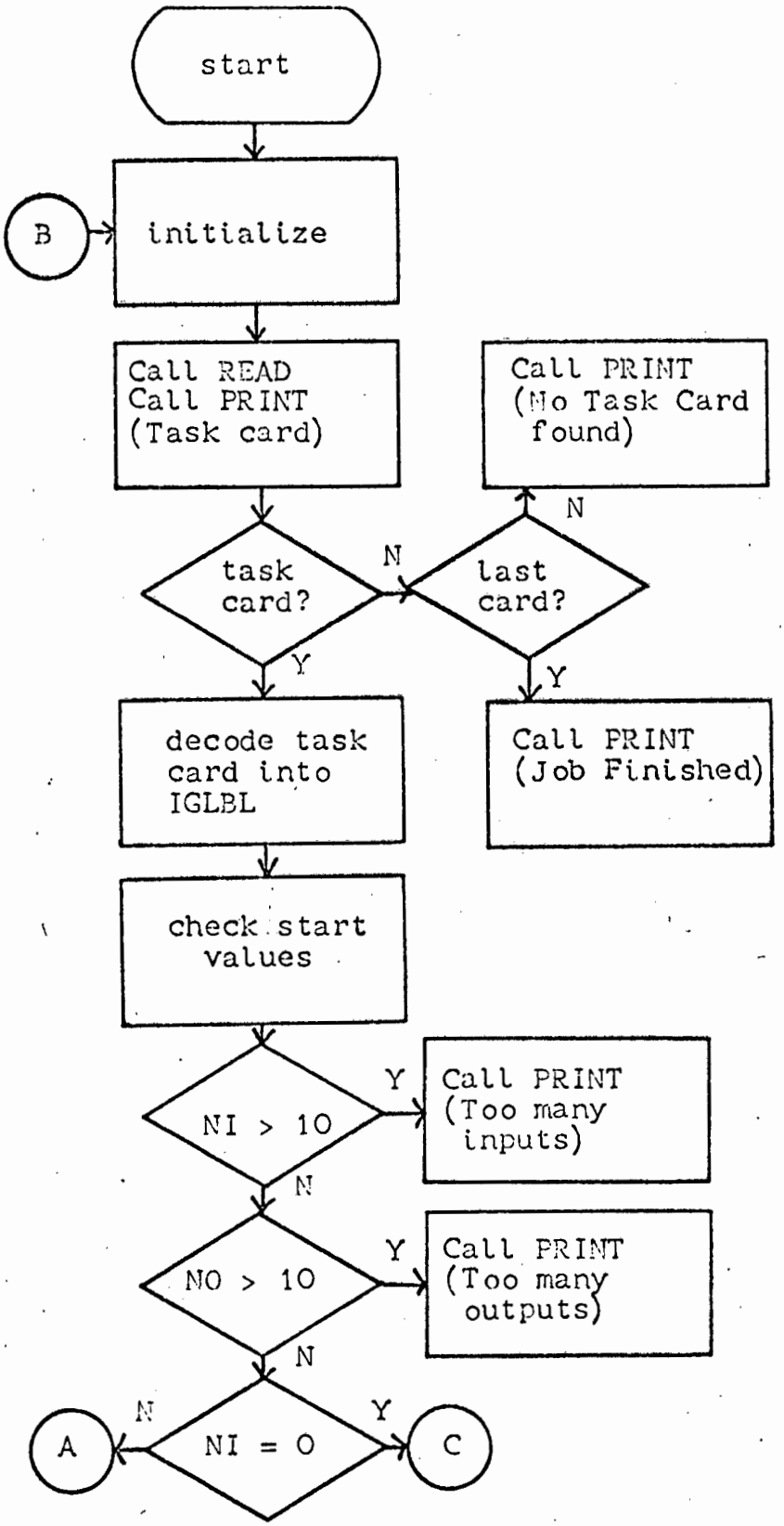


Subroutine END

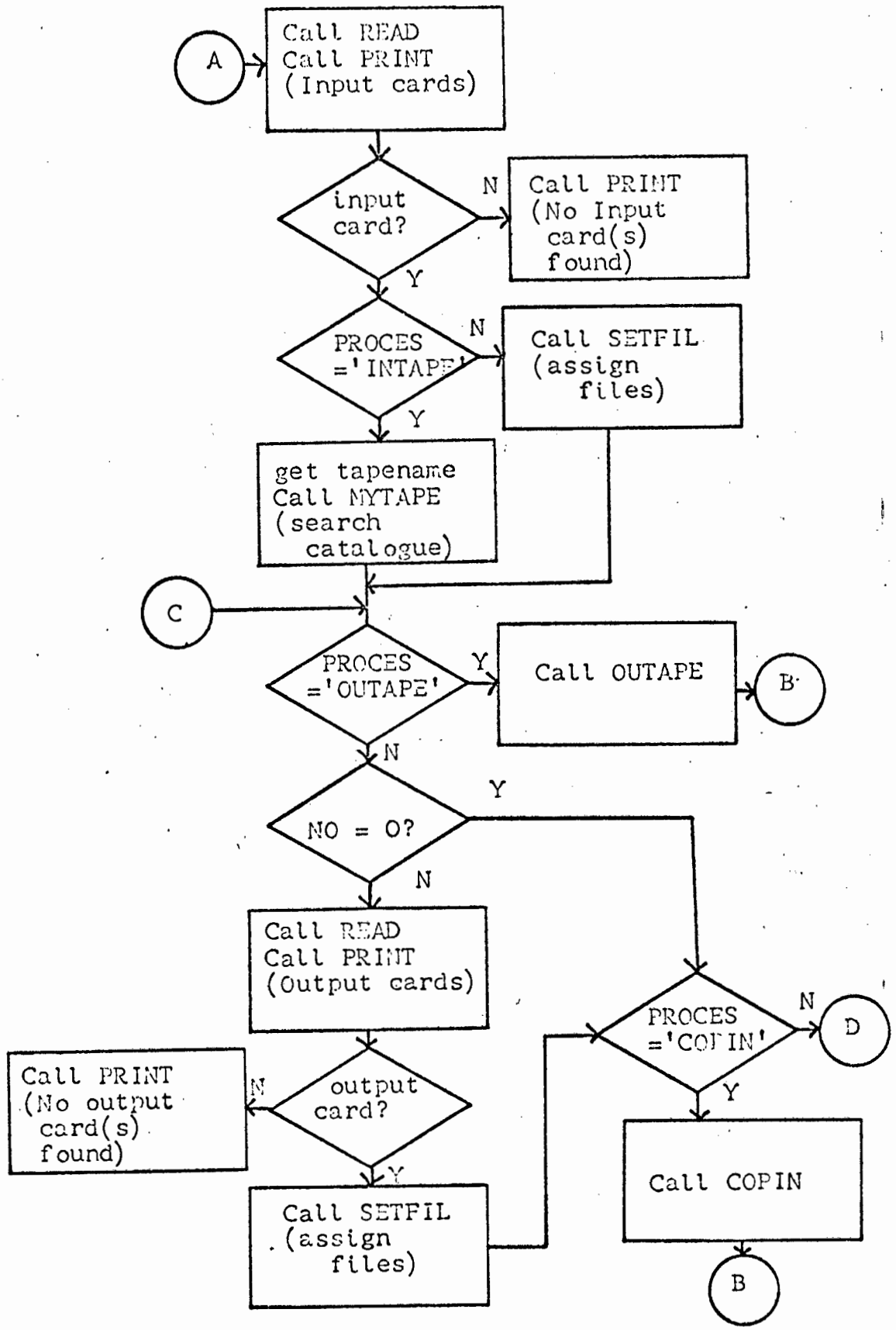


- Figure 30 -

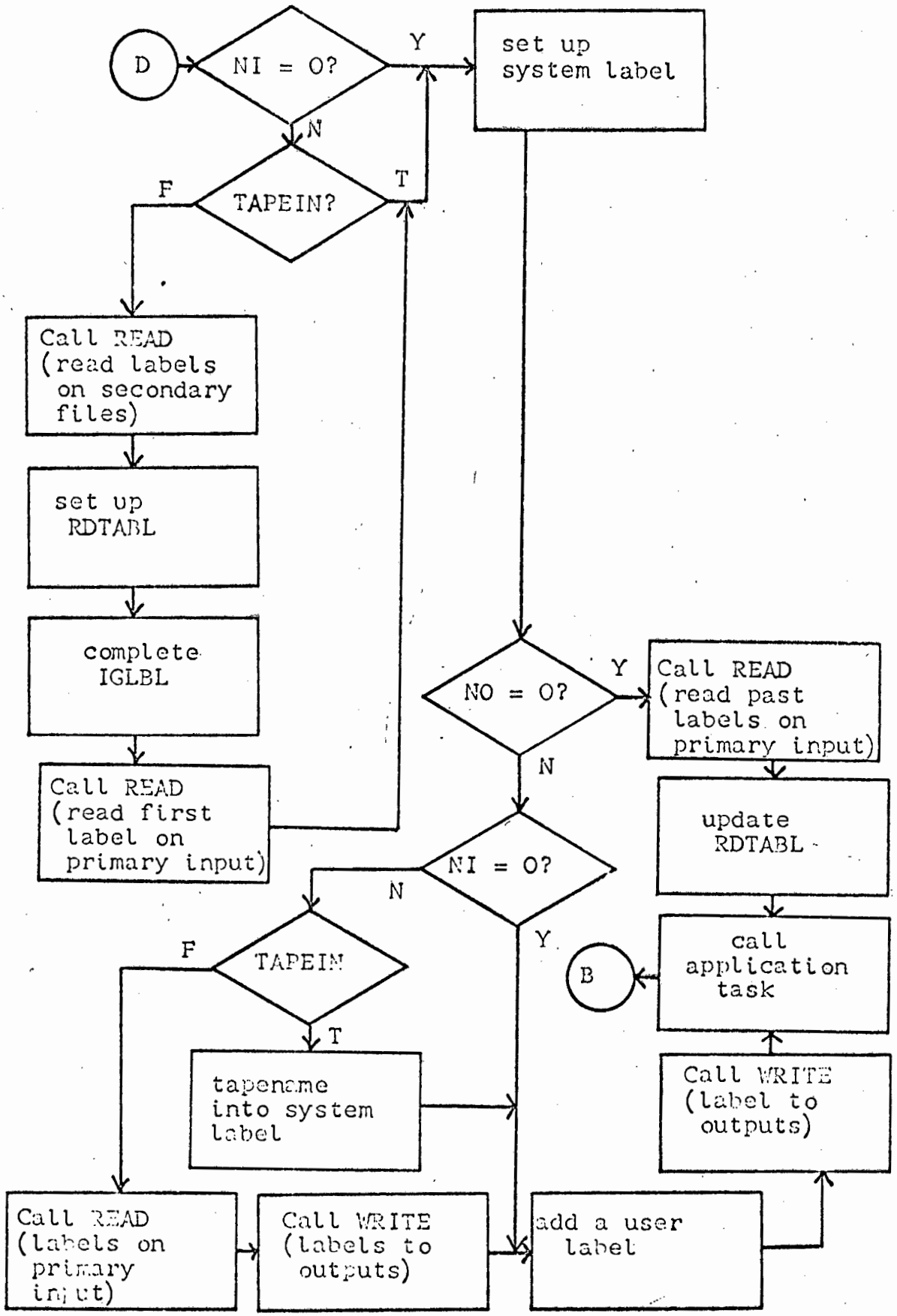
Flowchart of the MAIN Routine



- Figure 31 -

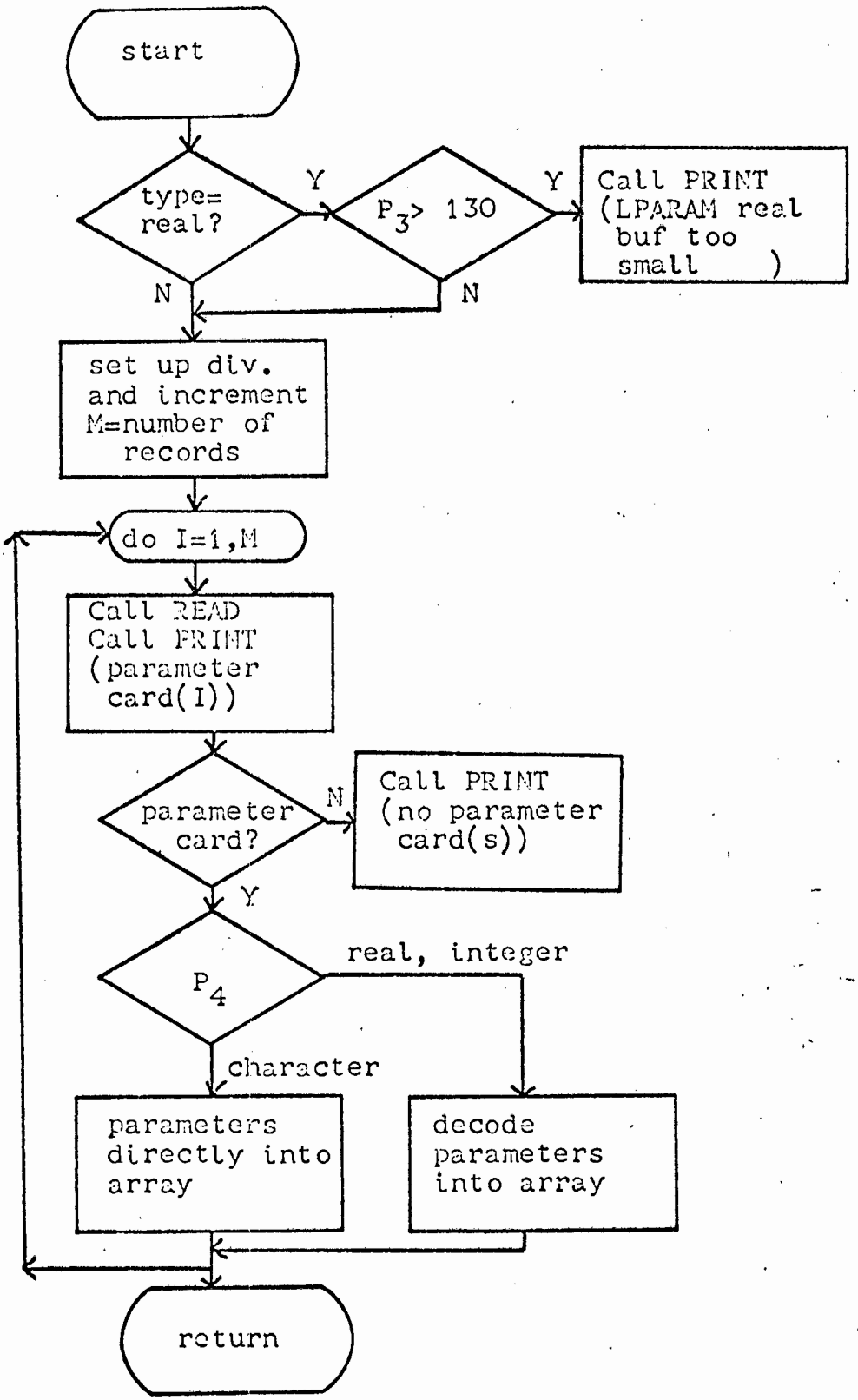

A
Call READ
Call PRINT
(Input cards)
input card?
N
Call PRINT
(No Input card(s) found)
Y
PROCES ='INTAPE'
N
Call SETFIL
(assign files)
Y
get tapename
Call MYTAPE
(search catalogue)
C
PROCES ='OUTAPE'
Y
Call OUTAPE
B
N
NO = 0?
Y
N
Call READ
Call PRINT
(Output cards)
Call PRINT
(No output card(s) found)
N
output card?
Y
Call SETFIL
(assign files)
PROCES ='COPIN'
N
D
Y
Call COPIN
B

- Figure 31 -


D
NI = 0?
Y
set up system label
N
F
TAPEIN?
T
Call READ (read labels on secondary files)
set up RDTABL
complete IGLBL
Call READ (read first label on primary input)
NO = 0?
Y
Call READ (read past labels on primary input)
N
update RDTABL
NI = 0?
N
Y
F
TAPEIN
T
B
call application task
tapename into system label
Call WRITE (label to outputs)
Call READ (labels on primary input)
Call WRITE (labels to outputs)
add a user label

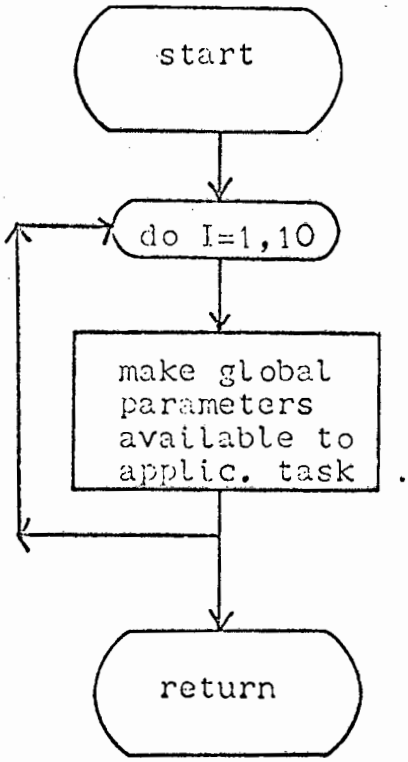
- Figure 31 -

Flowchart for Subroutine LPARAM



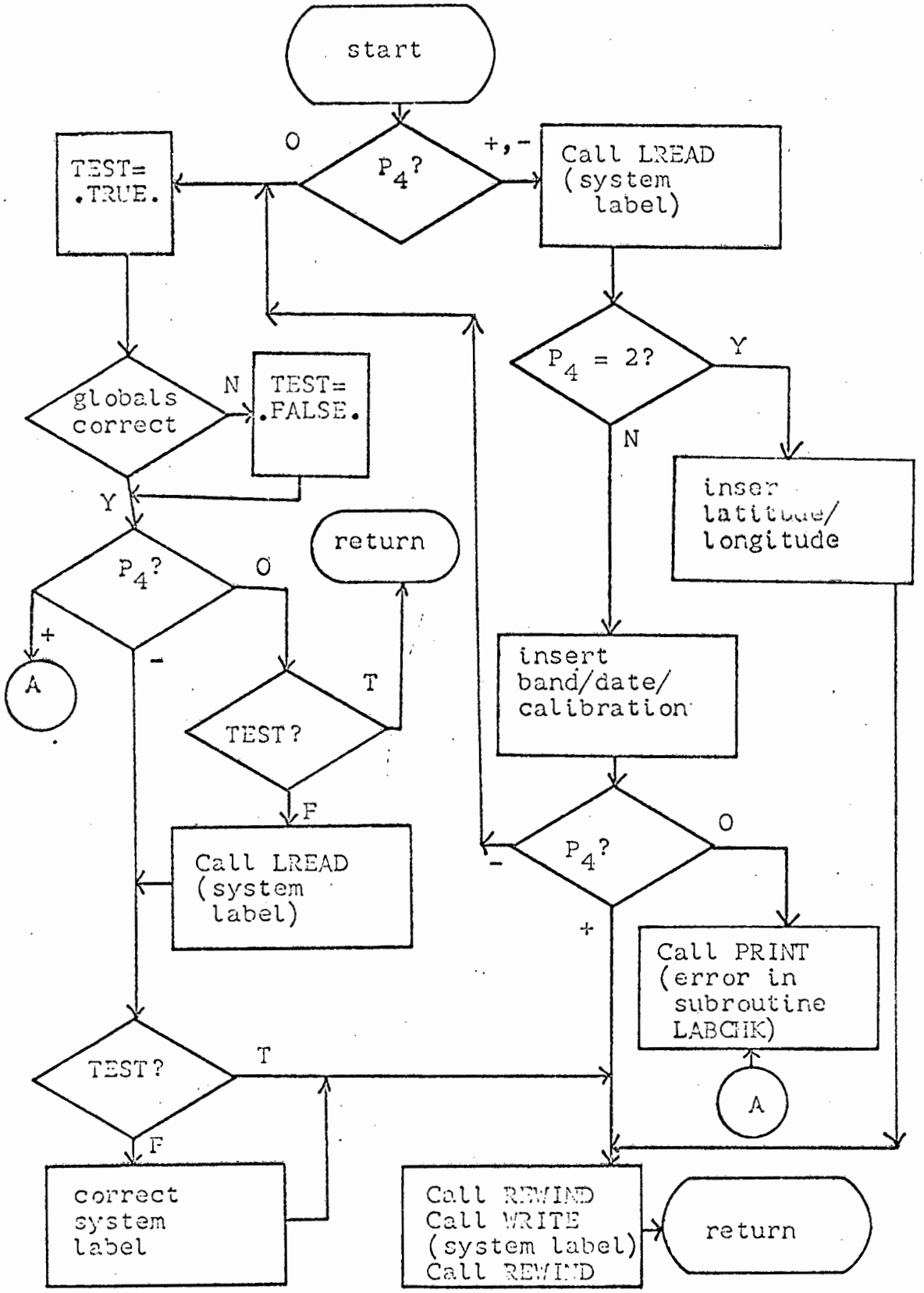
- Figure 32 -

Flowchart for Subroutine GPARAM



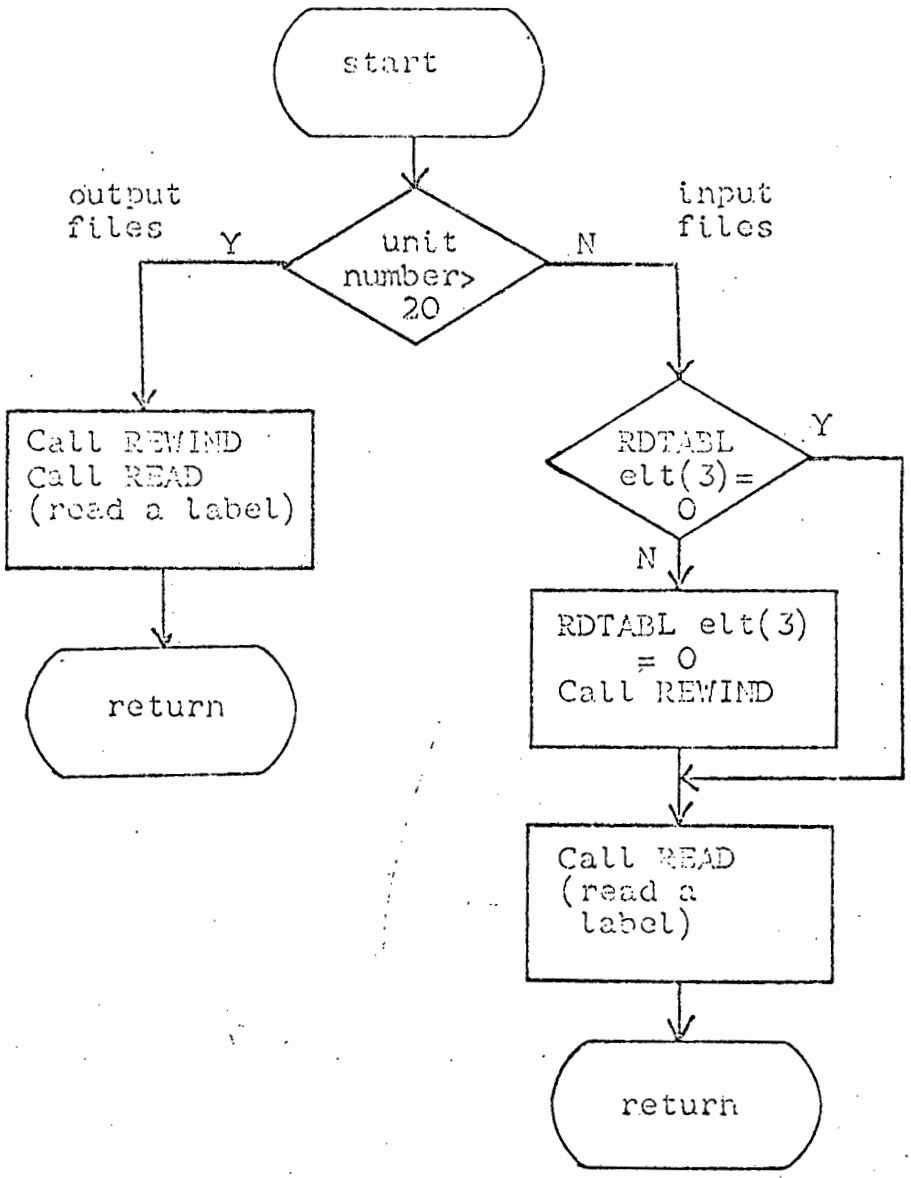
- Figure 33 -

Flowchart for Subroutine LABCHK



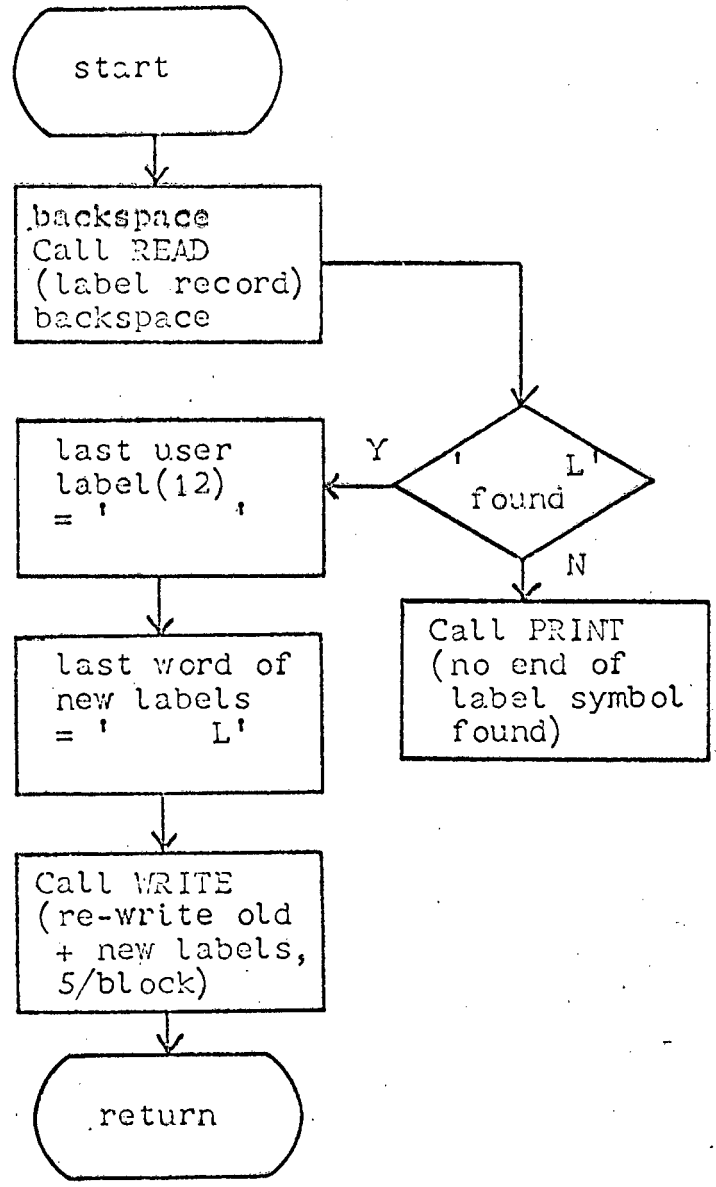
- Figure 34 -

Flowchart for Subroutine LRTAD



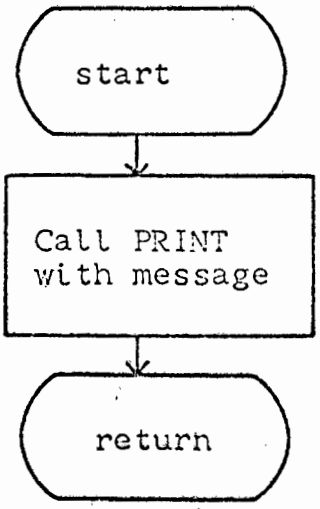
- Figure 35 -

Flowchart for Subroutine LABADD



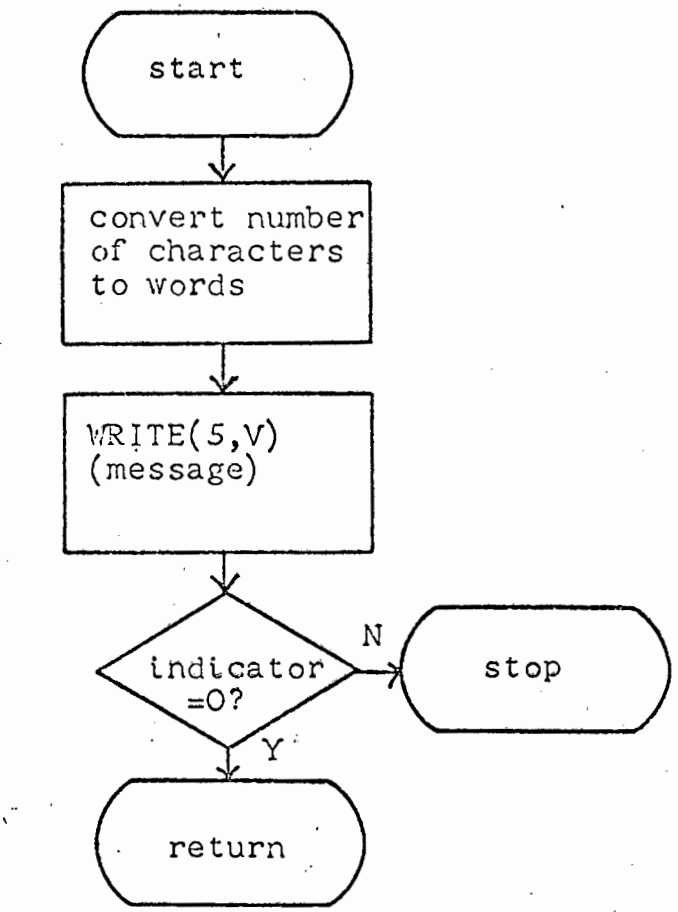
- Figure 36 -

Flowchart for Subroutine SYSMSG



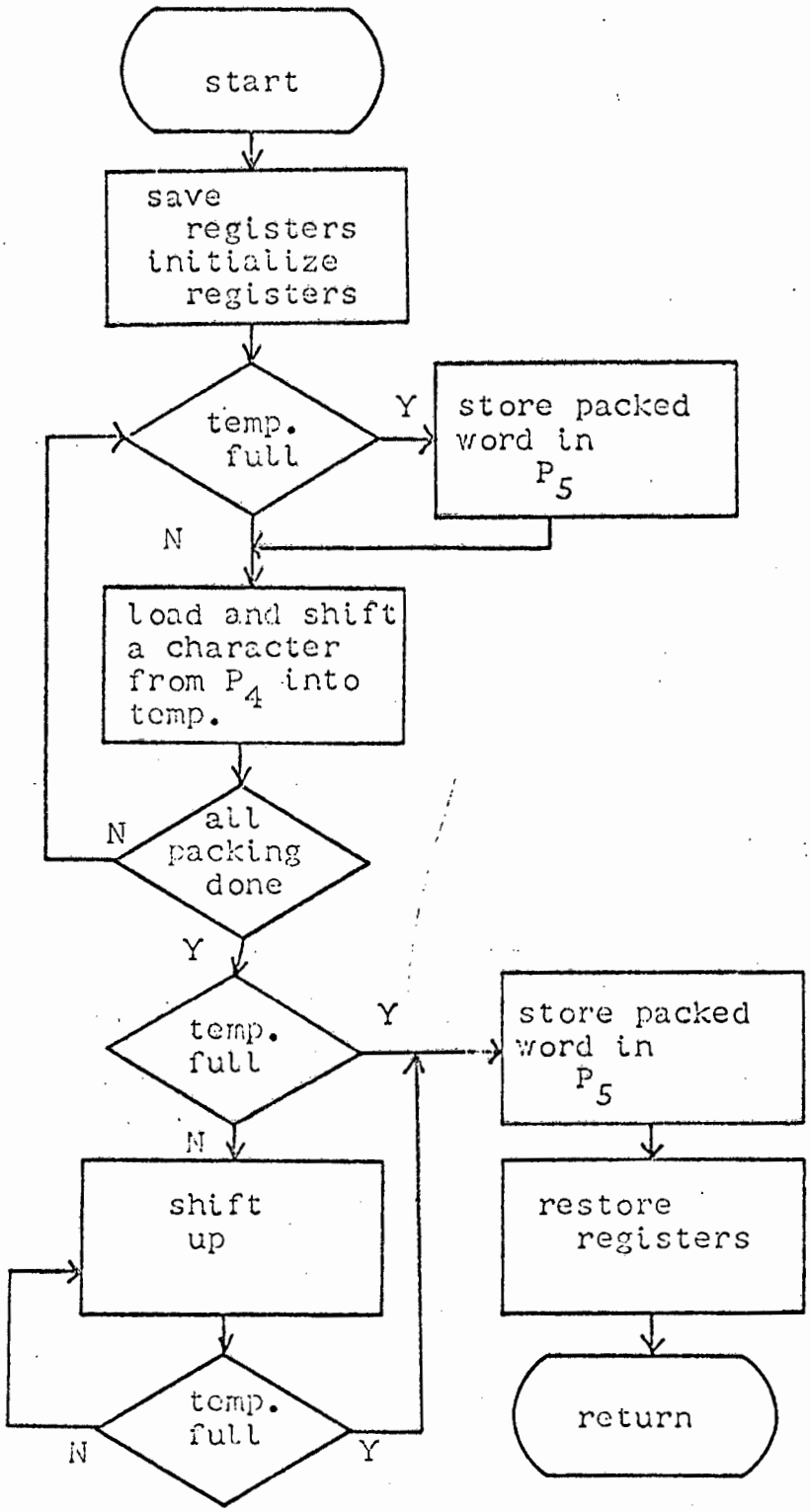
- Figure 37 -

Flowchart for Subroutine PRINT



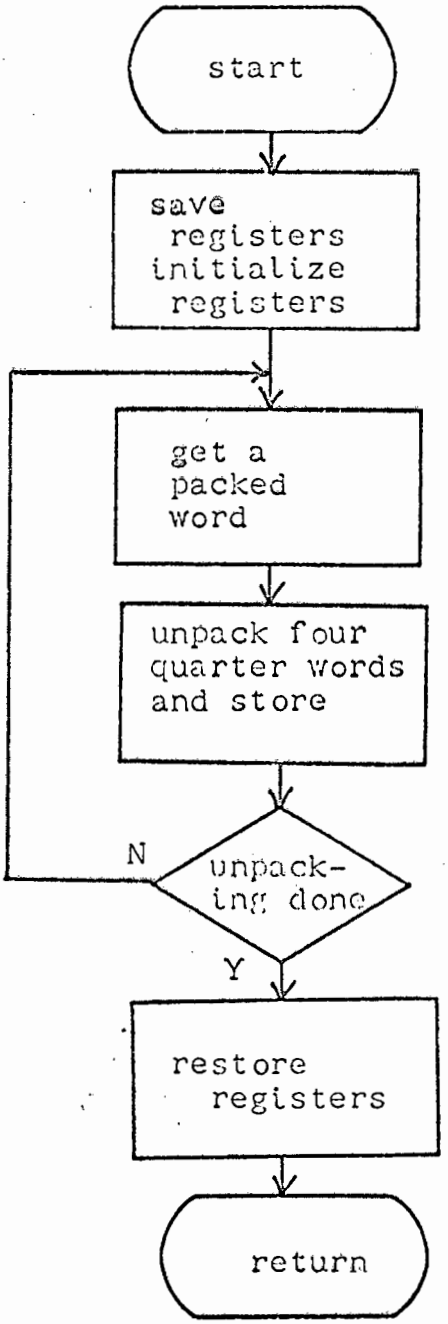
- Figure 38 -

Flowchart for Subroutine PACK



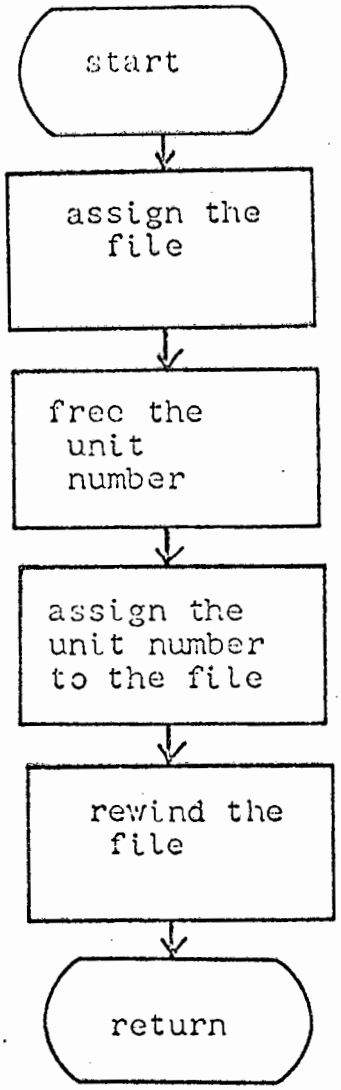
- Figure 39 -

Flowchart for Subroutine UNPACK



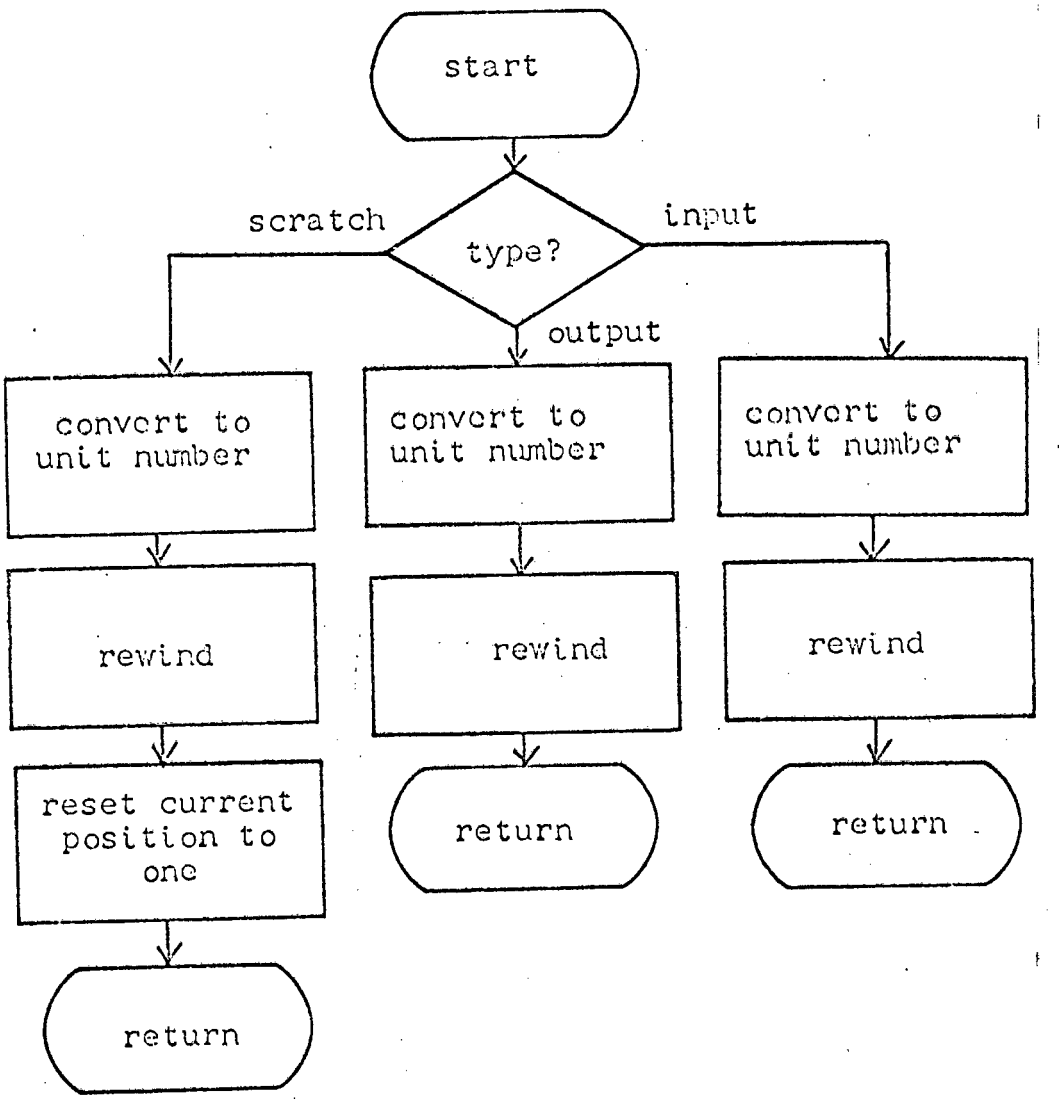
- Figure 40 -

Flowchart for Subroutine SETFIL



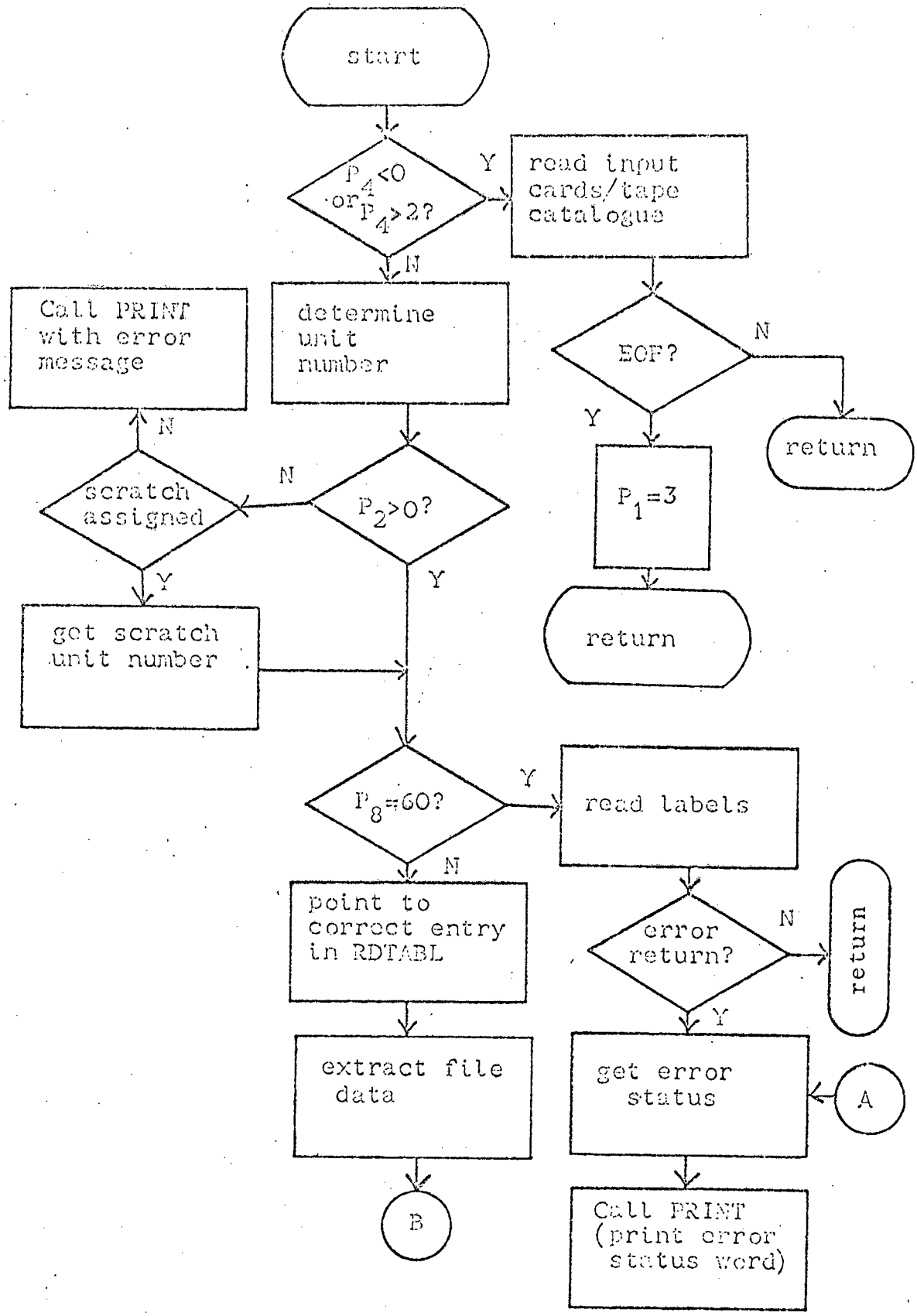
- Figure 41 -

Flowchart for Subroutine REWIND

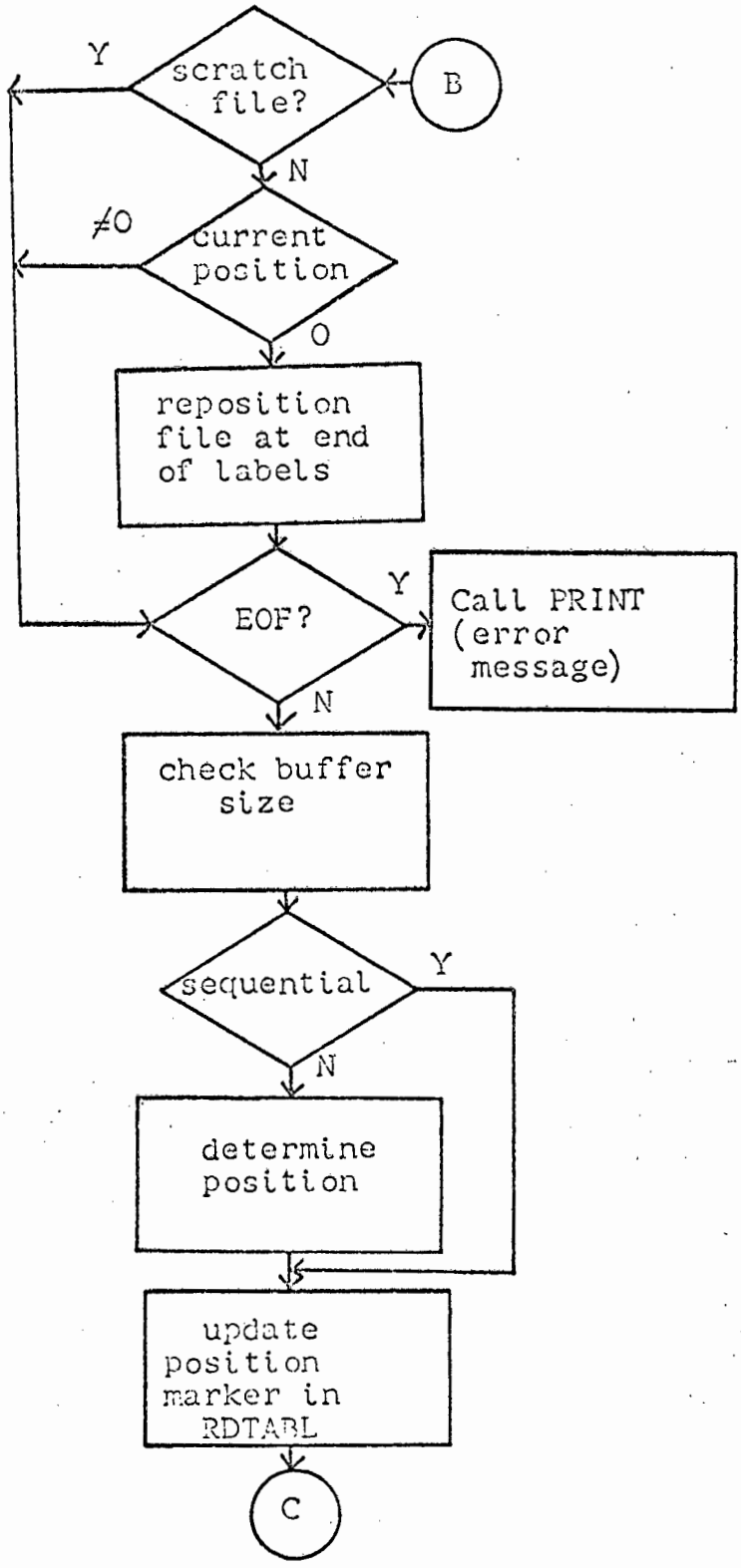


- Figure 42 -

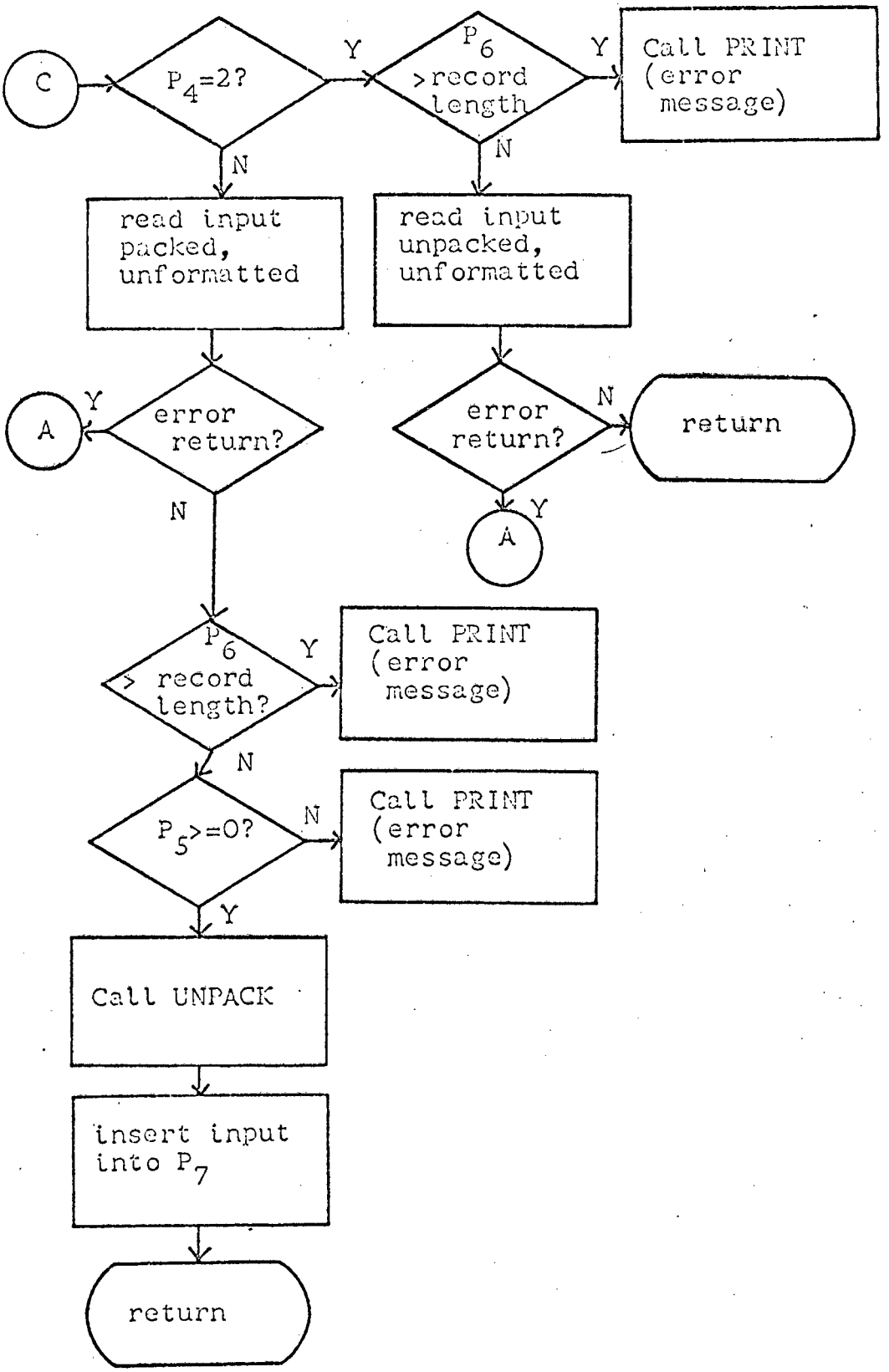
Flowchart for Subroutine READ



- Figure 43 -

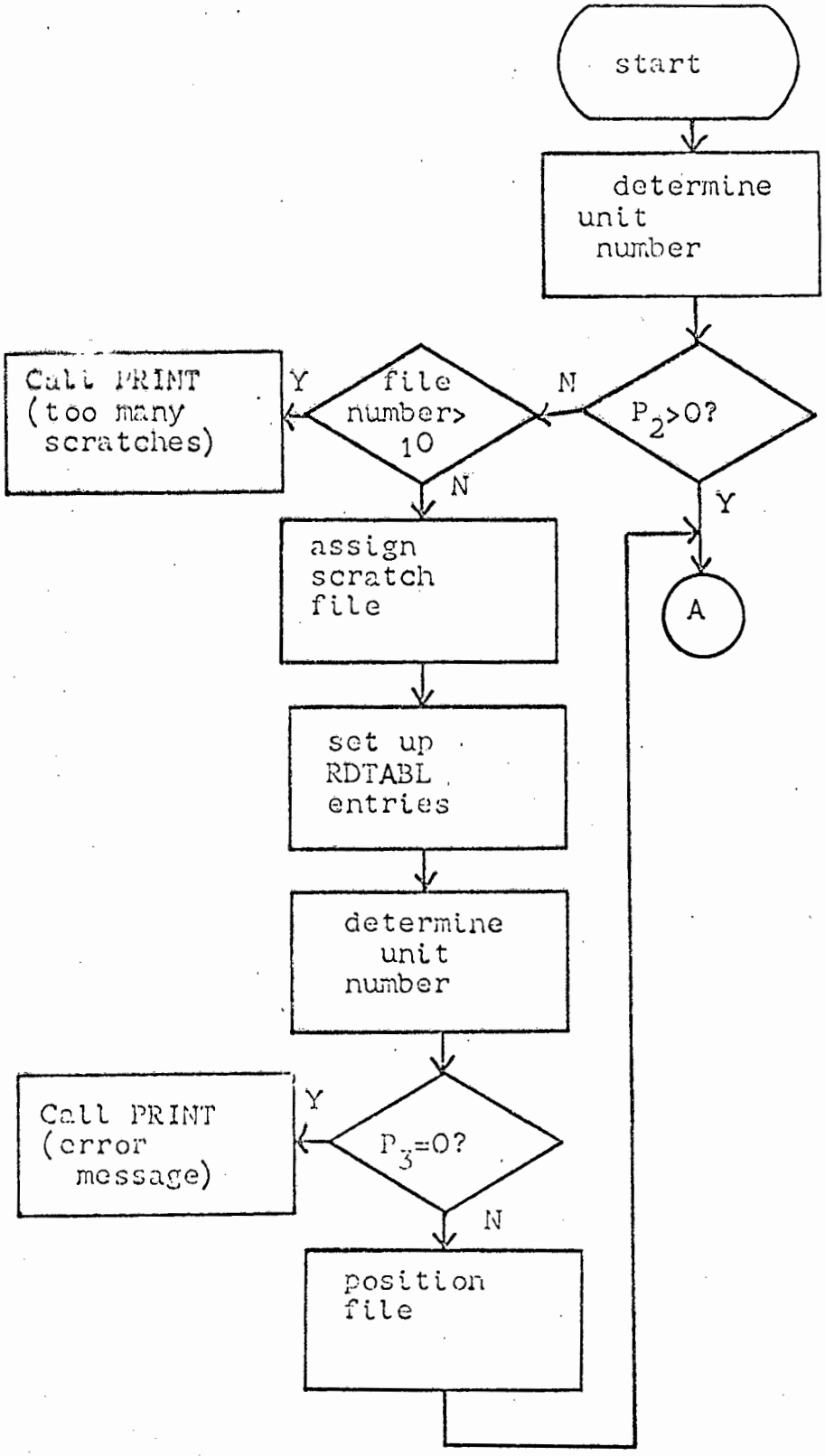

scratch file?
Y
B
N
current position
≠0
0
reposition file at end of labels
EOF?
Y
Call PRINT (error message)
N
check buffer size
sequential
Y
N
determine position
update position marker in RDTABL
C

- Figure 43 -

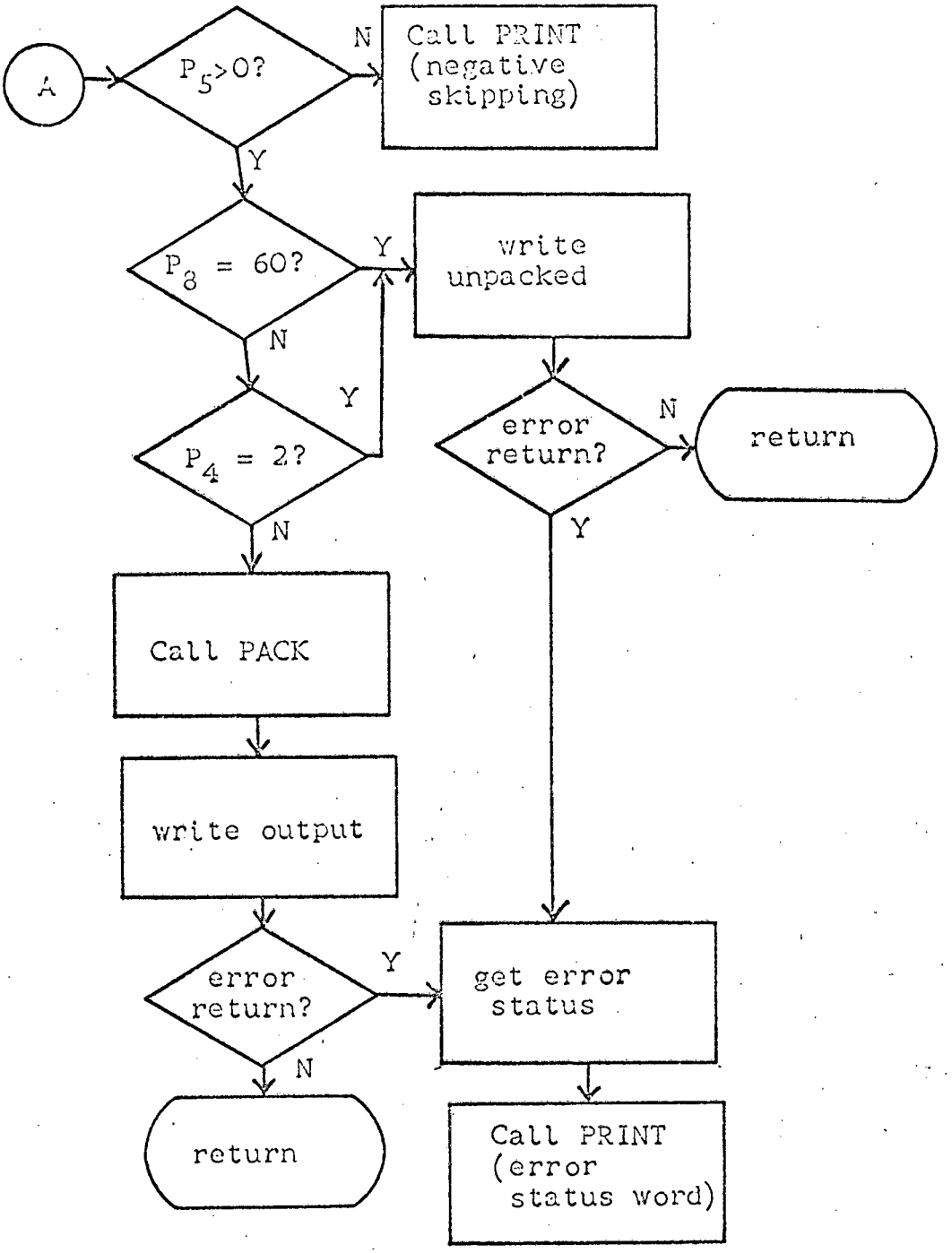

C
P4=2?
Y
P6 >record length
Y
Call PRINT (error message)
N
N
read input packed, unformatted
read input unpacked, unformatted
A
Y
error return?
error return?
N
return
N
A
Y
P6 > record length?
Y
Call PRINT (error message)
N
P5>=0?
N
Call PRINT (error message)
Y
Call UNPACK
insert input into P7
return

- Figure 43 -

Flowchart for Subroutine WRITE



- Figure 44 -


A
P_5>0?
N
Call PRINT (negative skipping)
Y
P_8 = 60?
Y
write unpacked
N
P_4 = 2?
Y
N
error return?
N
return
Y
Call PACK
write output
error return?
Y
get error status
N
return
Call PRINT (error status word)

- Figure 44 -

Appendix D

Runstreams for Application Tasks

This section deals with each application task discussed herein. The runstreams, parameters and supporting subroutines are dealt with from a programming standpoint. Along with this documentation, a flowchart for each routine is provided.

1) The Save Routines

a) CALL MYTAPE (P_1,P_2) where

P_1 - error indicator (not used),
P_2 - task name,
on entry - INTAPE,
on exit - NOAAIN, LSATIN, or COPIN.

This routine is not an application task in the true sense of the word as it is not user called but is used exclusively by MAIN.

b) OUTAPE:

```
12345678901234567890123456789012345678901
T     OUTAPE  NI  NO  SL  SS  NL  NS
I             FILEAA
I             FILEBB
      :
I             FILENN
P     NAMEAANAMEBB ... NAMENN   where n ≤ 10.
```

NI - number of inputs,
NO - number of outputs (=0),
SL,SS,NL,NS - not used as a whole file is copied,
FILEAA-FILENN - the name of the file(s) to be copied to tape,
NAMEAA-NAMENN - a six character parameter indicating the name of FILEAA-FILENN on the tape (NAMEii can be the same as FILEii).

c) COPIN:

```
12345678901234567890123456789012345678901
T     INTAPE  NI  NO  SL  SS  NL  NS
I     (blank card)
```

O FILEAA
O FILEBB
⋮
O FILENN
P NAMEAANAMEBB ... NAMENN where n ≤ 10.
NI - number of inputs (=1),
NO - number of outputs (=number of files),
FILEAA-FILENN - the file that reloaded tape files are to be inserted into,
NAMEAA-NAMENN - a six character parameter indicating the name of the required FILEii as it is on tape (this may be the same as FILEii).

Both OUTAPE and COPIN use an assembler routine to position themselves correctly on tape. This routine is called MOVEB.

d) CALL MOVEB(P_1,P_2) where
P_1 - status,
in - 0, backward move
1, forward move
out - 1, no error
≠1, error
P_2 - filename.

2) The Display Task

a) DISPLY:

12345678901234567890123456789012345678 90
T DISPLY NI NO SL SS NL NS
I FILEAA
P P_1 P_2 (integer)
NI - number of inputs (=1),
NO - number of outputs (=0),
SL - start line in image of displaying,
SS - start sample in image of displaying,
NL,NS - number of lines and samples to be displayed,

FILEAA - the name of the file which contains the image to be displayed,

P_1,P_2 - integer parameters indicating the line increment and pixel increment if not all are to be displayed.

DISPLY uses a routine CHPACK to pack the display characters six per word ready for output.

b) CALL CHPACK (P_1,P_2,P_3,P_4,P_5) where

P_1 - error indicator (not used),
P_2 - length of unpacked buffer,
P_3 - length of packed buffer,
P_4 - unpacked buffer,
P_5 - packed buffer.

DISPLY uses a routine PAGE to initiate and halt printing continuously over page edges.

c) CALL PAGE (P_1) where

P_1 - an indicator representing the type of print control desired:
0 - cause continuous printing
1 - return to normal with page throws etc..

3) The Enhancement Tasks

a) SCRIBE:

```
12345678901234567890123456789012345678901
T     SCRIBE  NI  NO  SL  SS  NL  NS
I             FILEAA
O             FILEBB
P   P1
P   P2  P3  P4  P5  ...
P   P12   ...
```

NI - number of inputs (=1 for scribing of an image and zero for generating pictures),
NO - number of outputs (=1),
SL,SS,NL,NS - coordinates on the input image of the area to be processed,

FILEAA,FILEBB - name of the input/output file,

P_1 - (integer), number of parameters to follow which must be at least 8, and not greater than 124,

P_2 - the value that the scribed line will be set to:

0 - the value of the scribed boundaries is set equal to zero if the average of the background is greater than 128 and is set equal to 255 if the average of the background is less than 128

DN - where 0 less than DN and DN is less than 255, the scribed boundaries are set equal to the DN

P_3 - tells the source of the background,

0 - the background comes from an input file

1 - background is self-generated at a value=P_4

2 - background increases linearly along each line from 0 toward P_4, incremented by P_5. When the calculated value would exceed P_4, the cycle repeats starting at 0.

3 - background decreases along each line from P_4 toward 0, decremented by P_5. When the calculated value would be less than 0, the cycle repeats beginning at P_4.

4 - background is constant along each line with the DN of each line increasing from 0 toward P_4, incremented by P_5. When the calculated value exceeds P_4, the cycle repeats beginning at 0.

5 - background is constant along each line with the DN of each line decreasing from P_4 to 0, decreasing by P_5. When the calculated value would be less than 0, the cycle repeats beginning at P_4.

P_4 - an integer as required by P_3,

P_5 - an integer as required by P_3,

P_6-P_{125} - these come in groups of four, i.e. P_6-P_9,

P_{10}-P_{13}, and indicate respectively, the starting line, starting sample, number of lines and number of samples of each area around which rectangles are to be scribed. Up to 30 rectangles may be specified, and they may overlap. NL=0 or NS=0 results in a single line two pixels wide. NL=-1, NS=-1 results in a line one pixel wide.

b) TRANSF:

```
12345678901234567890123456789012345678901234567890
T     TRANSF   NI  NO  SL  SS  NL  NS
I              FILEAA
O              FILEBB
P   P1  P2                              (integer)
P   P3                                  (real)
P     P4                                (character)
```

NI - number of inputs (=1, FILEAA),

NO - number of outputs (=1, FILEBB),

SL,SS,NL,NS - size of image,

P_1 - an indicator relating whether the weighted matrix is to be added back into the original or not,
1 - add back into original,
other - take weighted transform only,

P_2 - the number of low frequency components to remove for edge enhancement,

P_3 - the factor of the exponent for the computation of the weighting vector,

P_4 - a keyword describing the type of weighting to be carried out,
'EDGE ' - edge enhancement,
'SMOOTH' - construct an exponential weight matrix for smoothing.

TRANSF calls a weighting array generating routine called LN1DWT.

c) CALL LN1DWT (P_1,P_2,P_3,P_4) where

P_1 - line length from TRANSF (must be a power of 2),
P_2 - P_3 from TRANSF,
P_3 - the array where the weights are to be placed, must be of length P_1+1,
P_4 - not used.

d) TRANS2:

```
12345678901234567890123456789012345678901234567890
T     TRANS2  NI  NO  SL  SS  NL  NS
I             FILEAA                    (image)
O             FILEBB                (real transform)
O             FILECC                (imaginary transform)
P   P1  P2                              (integer)
P     P3                                (character)
```

NI - number of inputs (=1, FILEAA),
NO - number of outputs (=2, FILEBB, FILECC),
SL,SS,NL,NS - size of image, must be 4×4, 8×8, 16×16, 32×32, or ≥ 64×64 but ≤1000×1000,
P_1 - the overlapping between segments of the transform,
P_2 - specifies whether smoothing or edge enhancement is desired,
zero - smoothing,
non-zero - edge enhancement,
P_3 - the name of the input file (FILEAA) for label purposes.

e) LN2DWT:

```
12345678901234567890123456789012345678901234567890
T     LN2DWT  NI  NO  SL  SS  NL  NS
I             FILEAA            (real transform)
I             FILEBB            (imaginary transform)
O             FILECC            (real transform)
O             FILEDD            (imaginary transform)
O             FILEFF            (weight matrix)
P     P1                        (character)
P   P2                          (real)
P       P3                      (integer)
```

NI - number of inputs (=2, FILEAA, FILEBB)
NO - number of outputs (=2 for edge enhancement, FILECC, FILEDD, =3 for smoothing, FILECC, FILEDD, FILEFF),
SL,SS,NL,NS - not used,
P_1 - the type of weighting to be carried out,
'SMOOTH' - smooth the transform,
'EDGE ' - edge enhancement desired,
P_2 - value used in the exponent of the function for creation of the smoothing weight matrix,
P_3 - the number of low frequency components to be removed for edge enhancement..

f) RETRAN:

```
12345678901234567890123456789012345678901234567890
T     RETRAN  NI   NO   SL   SS   NL   NS
I             FILEAA                   (real transform)
I             FILEBB                   (imaginary transform)
O             FILECC                   (image)
P     P1   P2                          (integer)
```

NI - number of inputs (=2),
NO - number of outputs (=1),
SL,SS,NL,NS - not used,
P_1 - a parameter used for division of each pixel of the resulting image in order to bring it into the range 0 - 255 if this is required,
P_2 - specifies whether smoothing or edge enhancement took place,
zero - smoothing,
non-zero - edge enhancement.

All the transform tasks call the Fourier transform routine FFTDRV which calls FOURIE and UNSCRM.

g) CALL FFTDRV (P_1,P_2,P_3,P_4) where

P_1 - in - real part of sequence,
out - real part of transform,
P_2 - in - imaginary part of sequence,
out - imaginary part of transform,

P_3 - size of transform, this must be positive and a power of 2,

P_4 - type of transform,
positive - forward transform,
negative - reverse transform,
zero - no transform.

h) CALL FOURIE (P_1,P_2,P_3,P_4) where

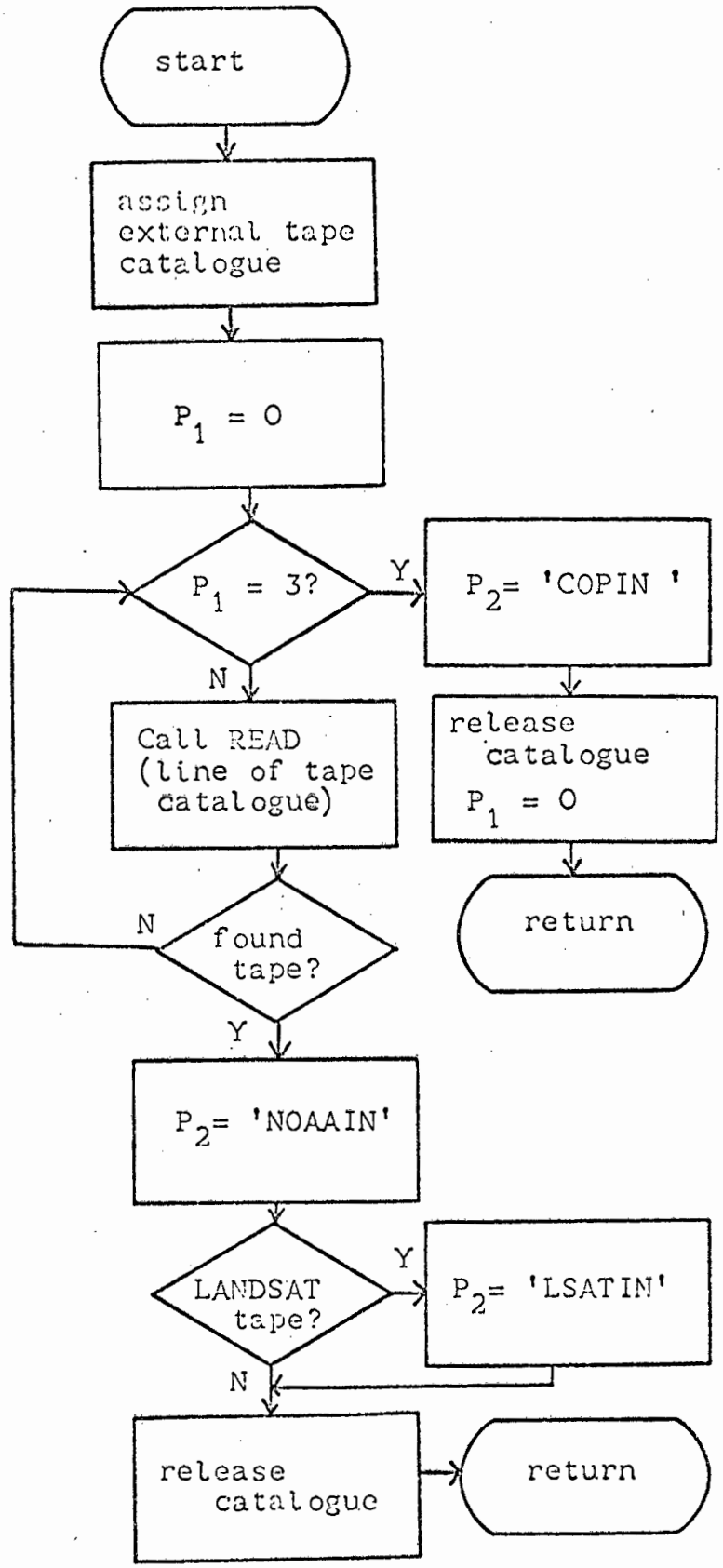
P_1,P_2,P_3,P_4 - same as FFTDRV.

i) CALL UNSCRM (P_1,P_2,P_3,P_4) where

P_1,P_2,P_3 - same as FFTDRV,

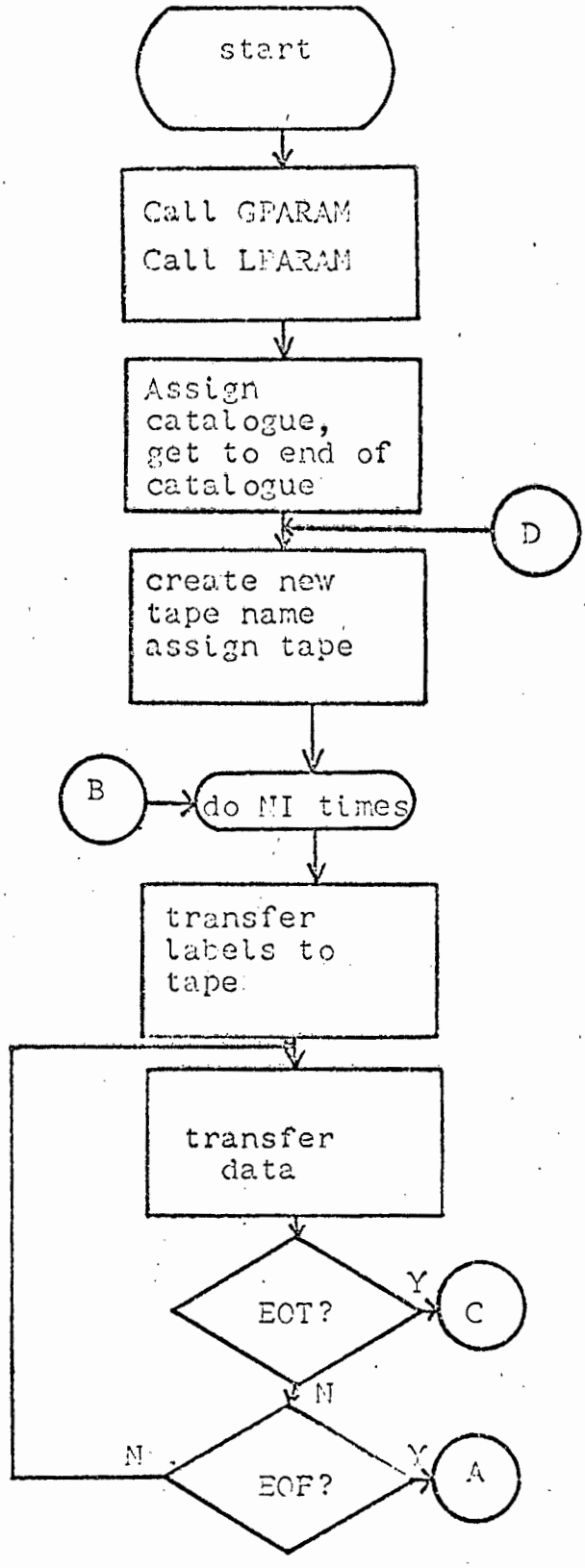
P_4 - the power of 2 of the size of the transform, i.e. if transform is of length 64, P_4=6.

Flowchart of Subroutine MYTAPE

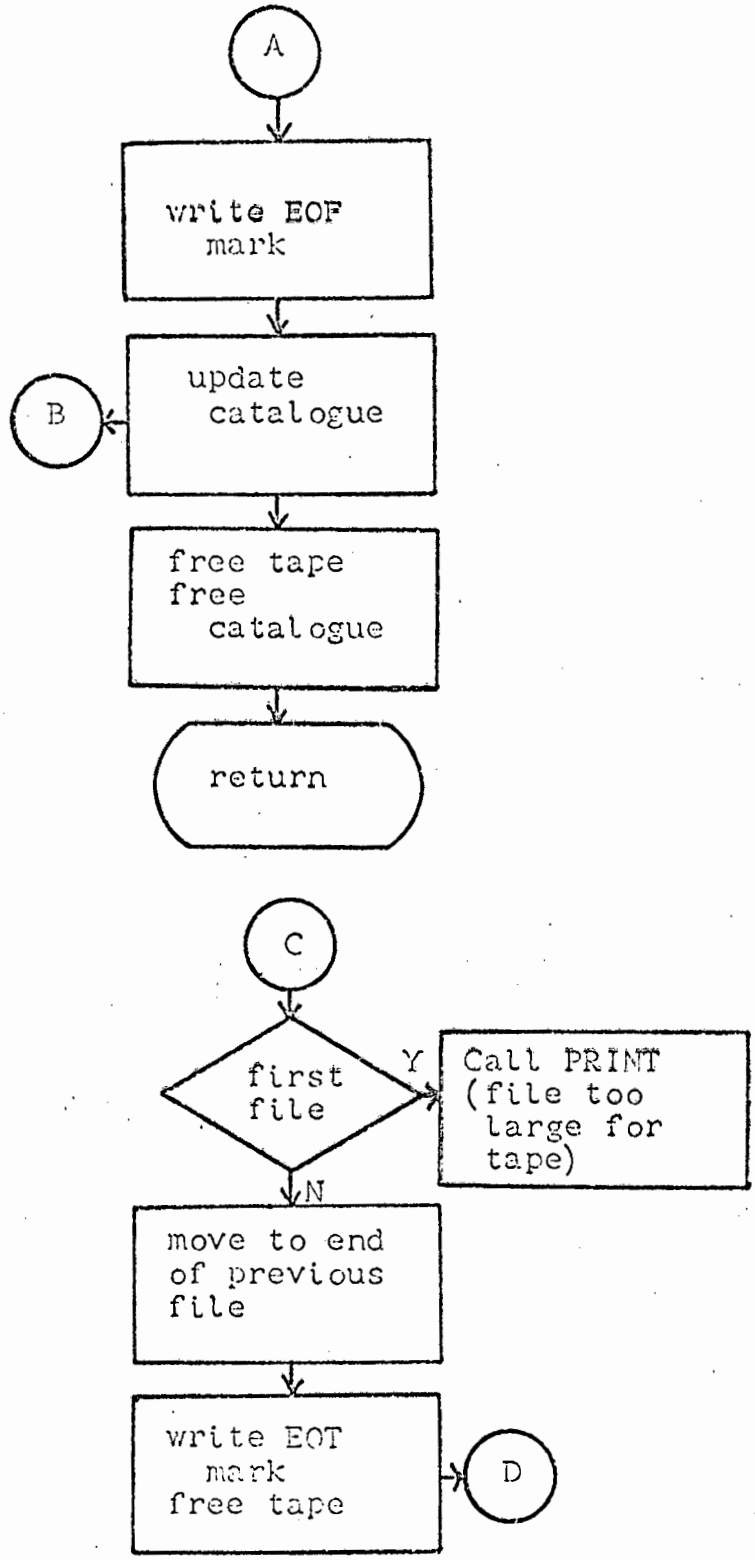


- Figure 45 -

Flowchart of Subroutine OUTAPE

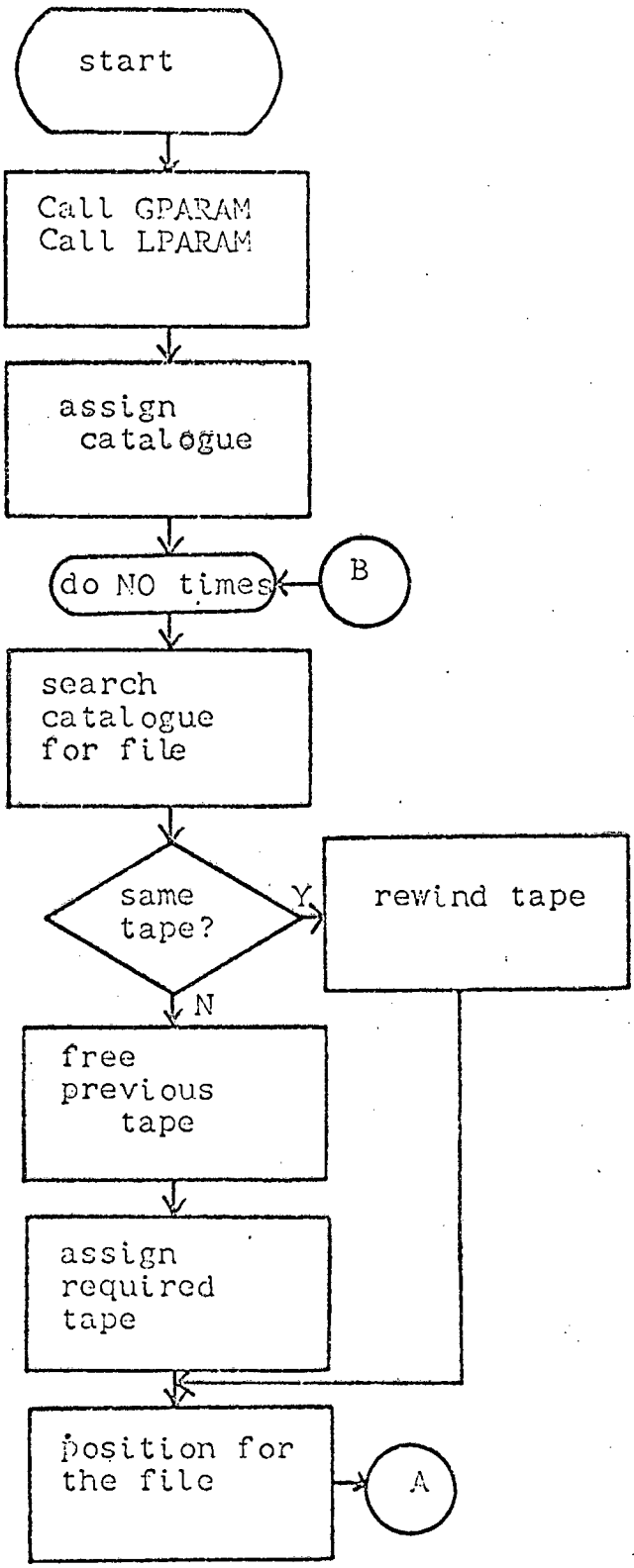


- Figure 46 -

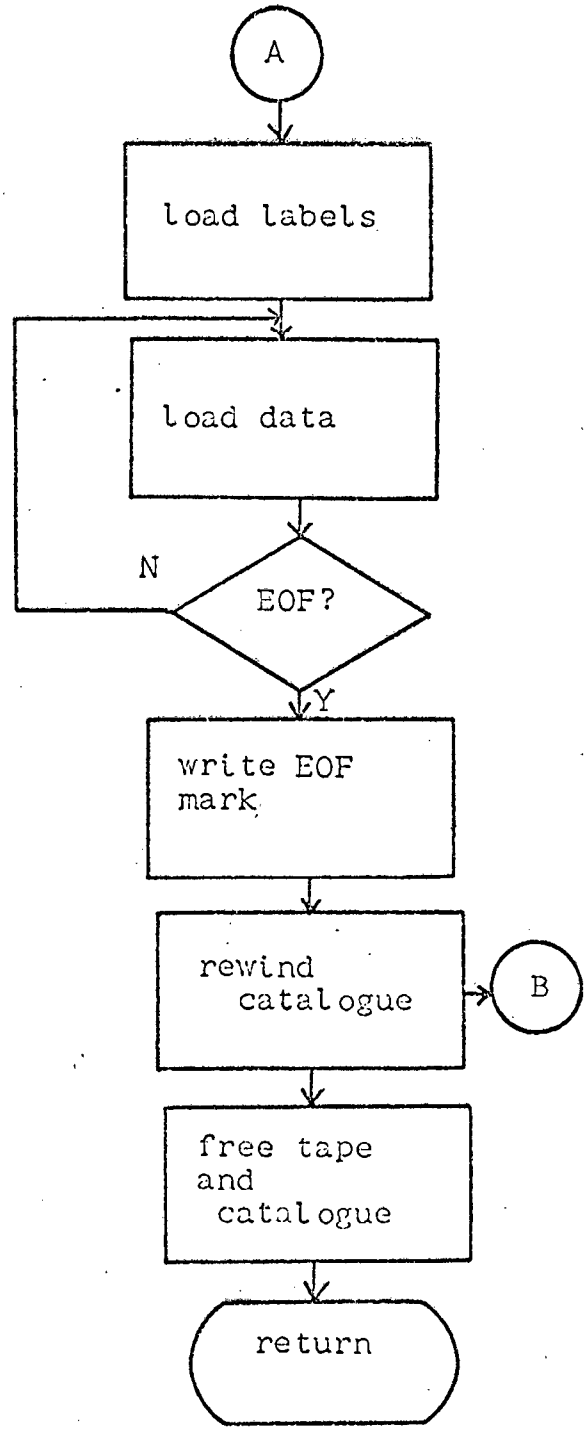

A
write EOF mark
update catalogue
B
free tape
free catalogue
return
C
first file
Y
Call PRINT (file too large for tape)
N
move to end of previous file
write EOT mark
free tape
D

- Figure 46 -

Flowchart of Subroutine COPIN

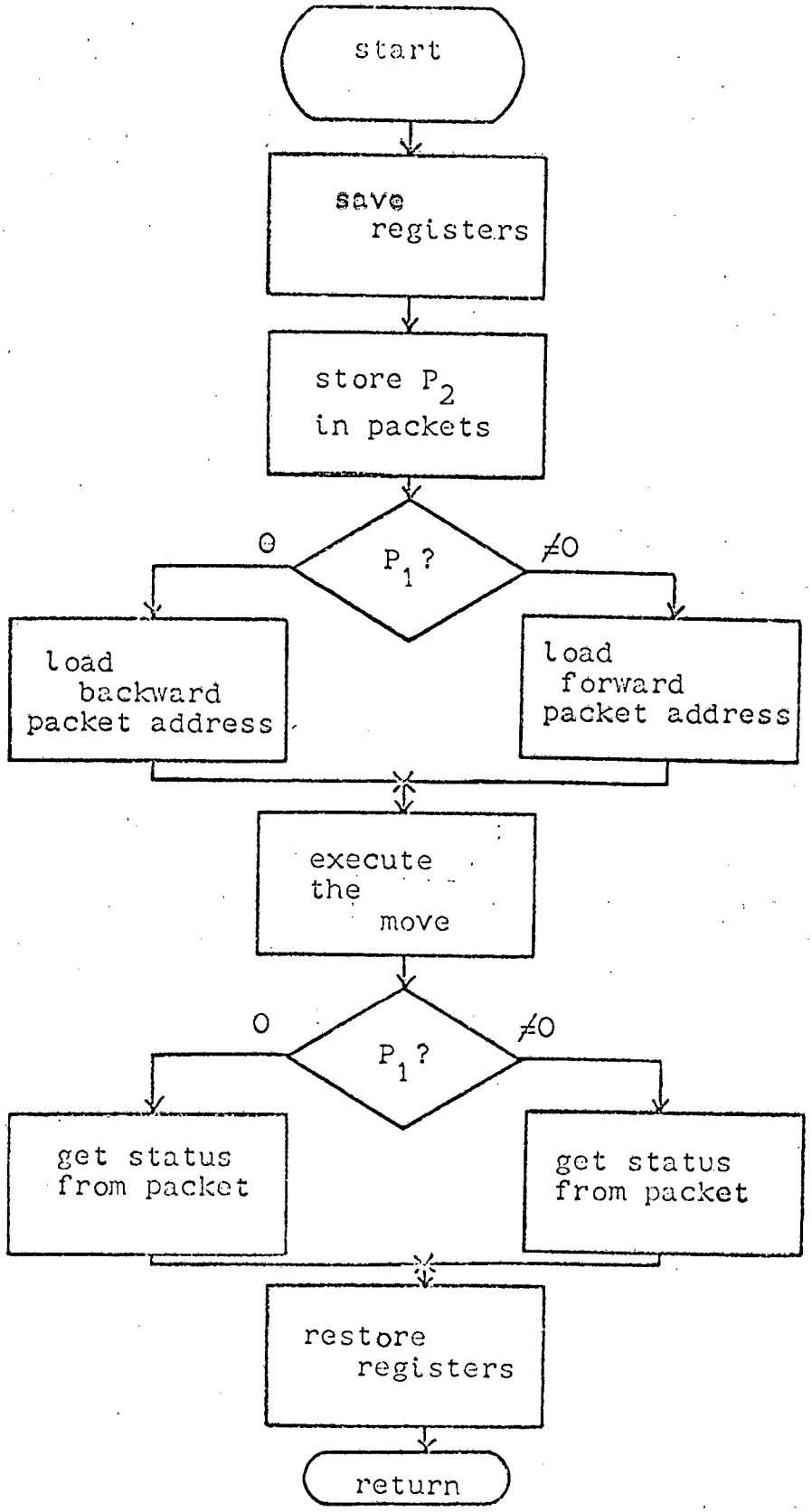


- Figure 47 -


A
load labels
load data
N
EOF?
Y
write EOF mark
rewind catalogue
B
free tape and catalogue
return

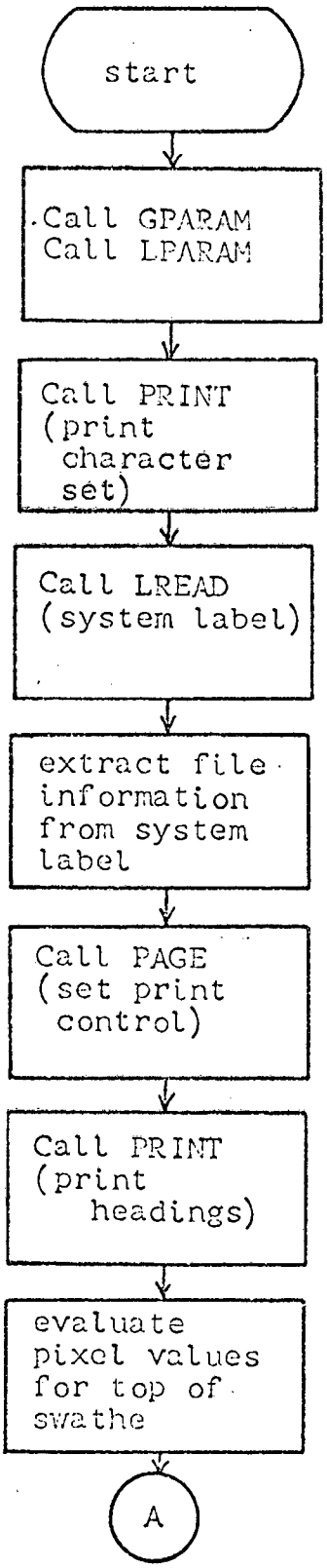
- Figure 47 -

Flowchart for Subroutine MOVER

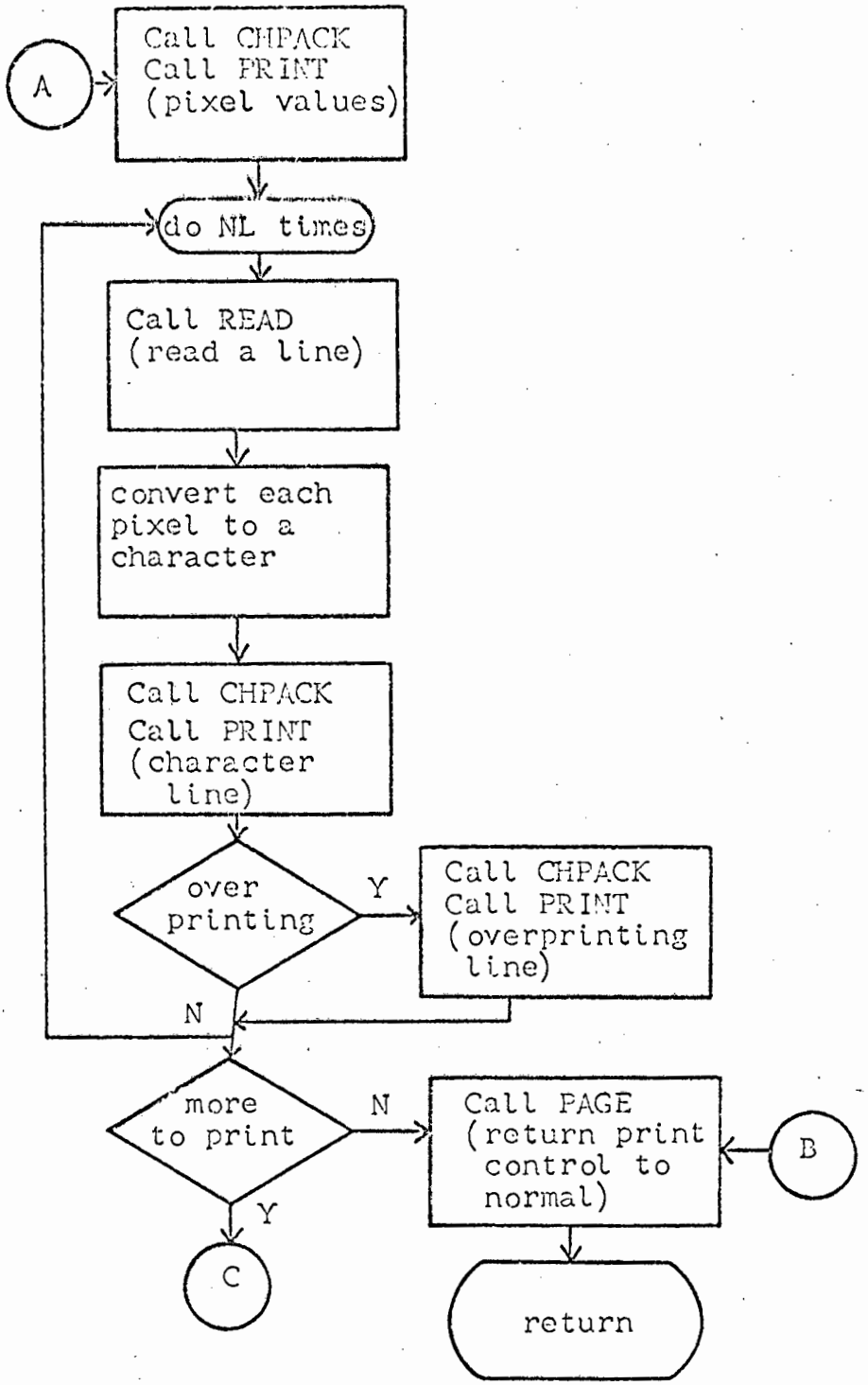


- Figure 48 -

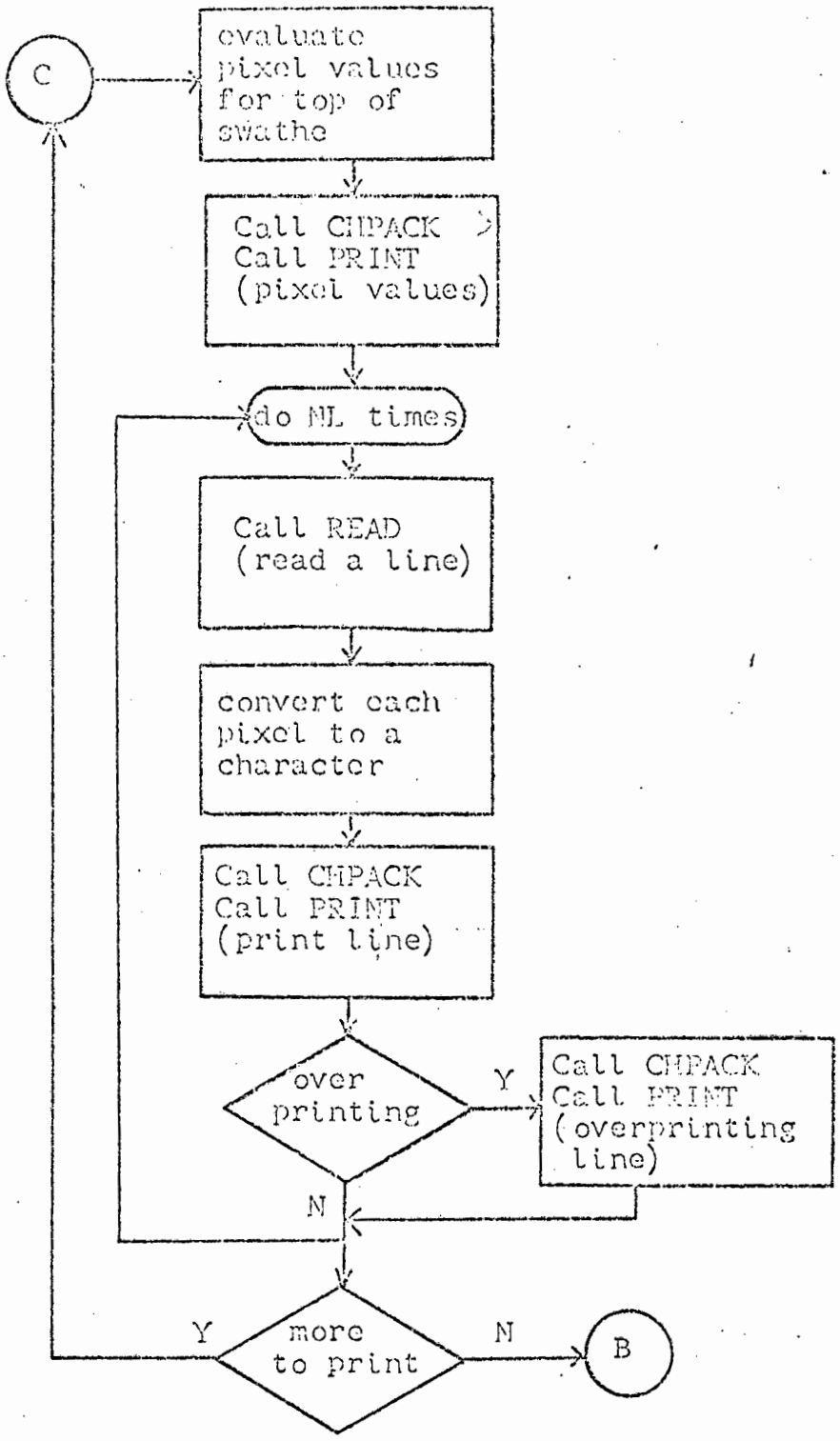
Flowchart for Subroutine DISPLY



- Figure 49 -

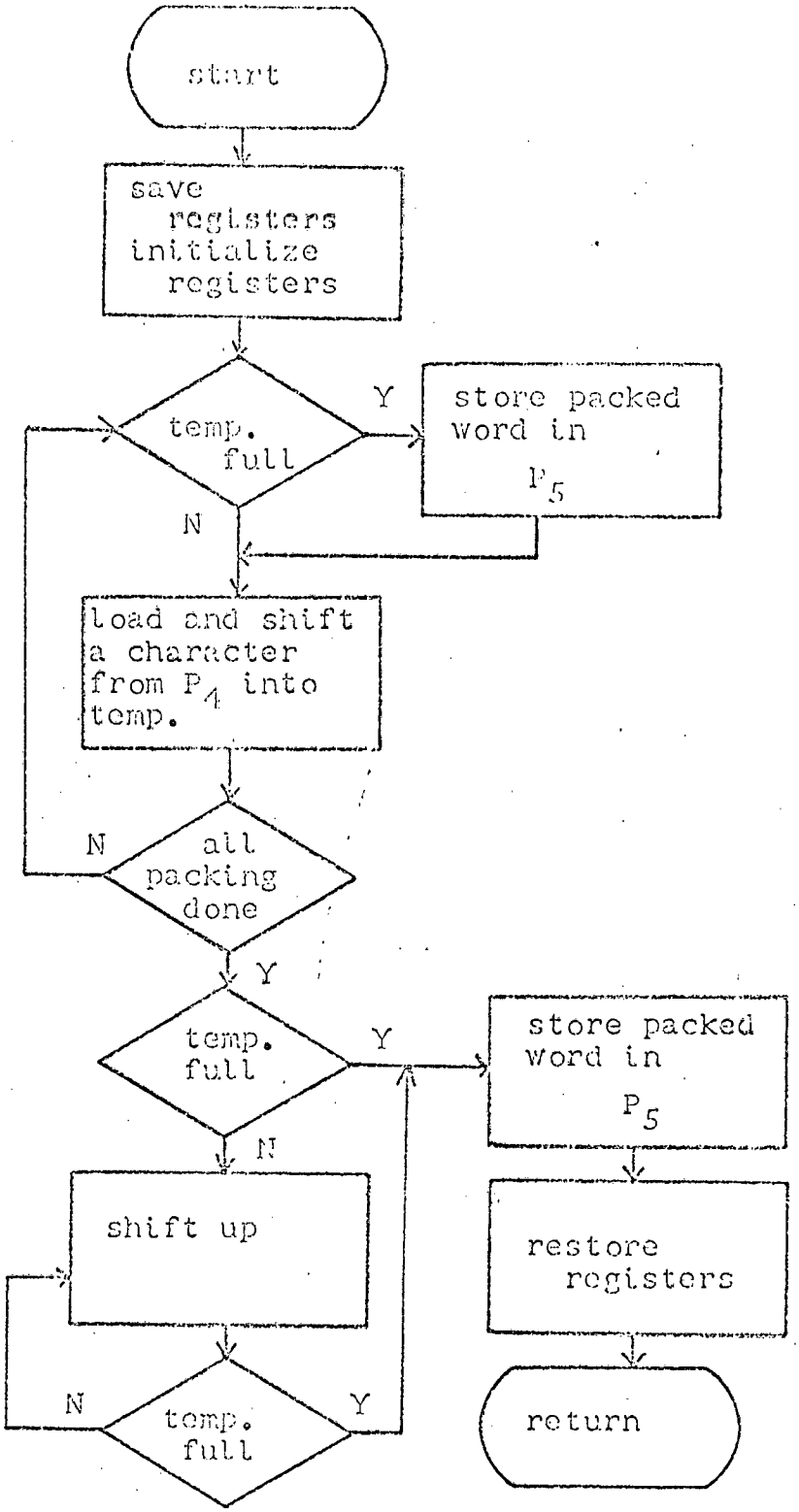

A
Call CHPACK
Call PRINT
(pixel values)
do NL times
Call READ
(read a line)
convert each
pixel to a
character
Call CHPACK
Call PRINT
(character
line)
over
printing
Y
Call CHPACK
Call PRINT
(overprinting
line)
N
more
to print
N
Call PAGE
(return print
control to
normal)
B
Y
C
return

- Figure 49 -


C
evaluate pixel values for top of swathe
Call CHPACK
Call PRINT
(pixel values)
do NL times
Call READ
(read a line)
convert each pixel to a character
Call CHPACK
Call PRINT
(print line)
over printing
Y
Call CHPACK
Call PRINT
(overprinting line)
N
Y
more to print
N
B

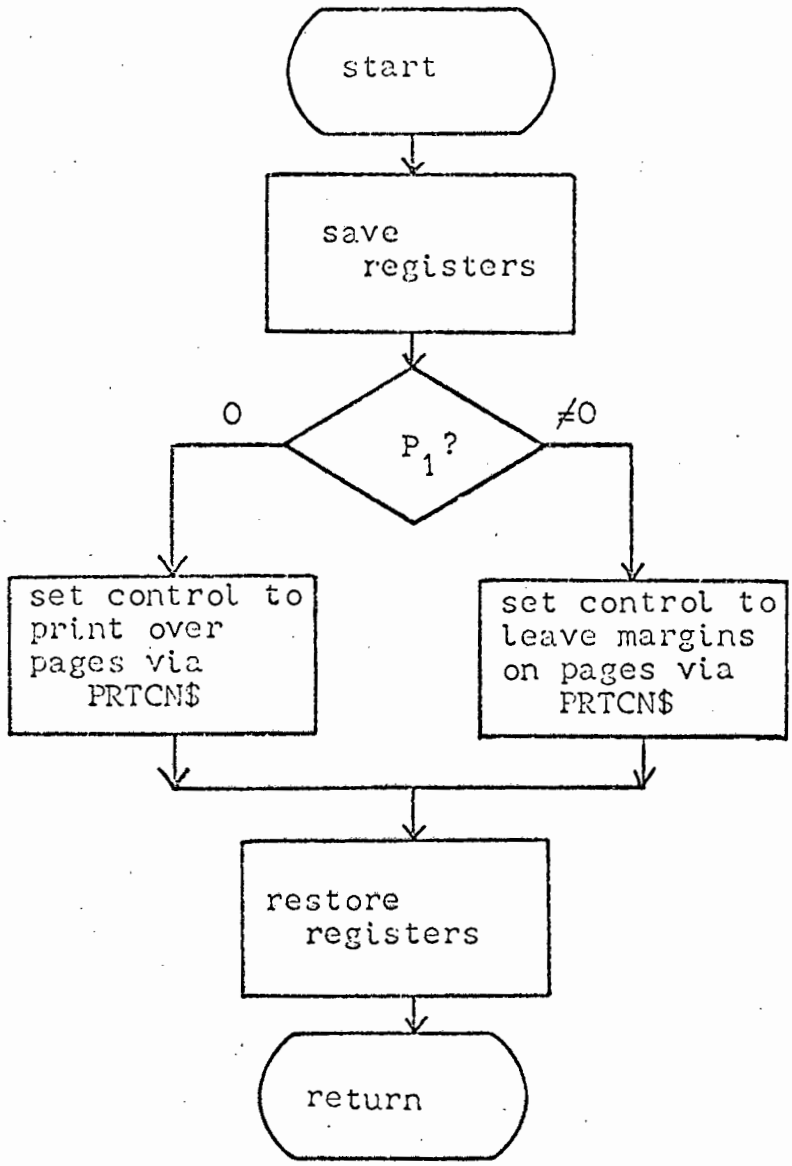
- Figure 49 -

Flowchart for Subroutine CHPACK



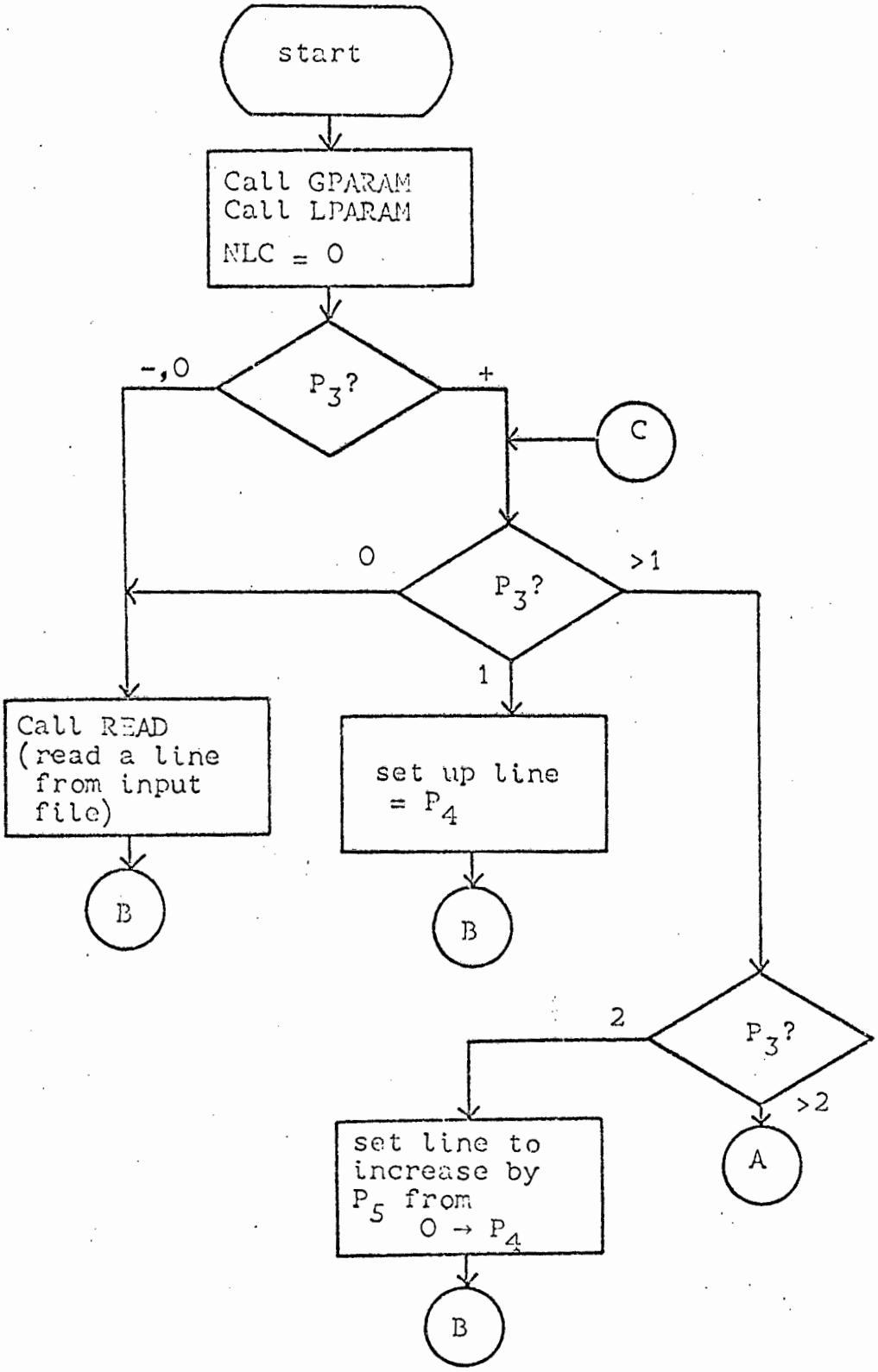
- Figure 50 -

Flowchart for Subroutine PAGE

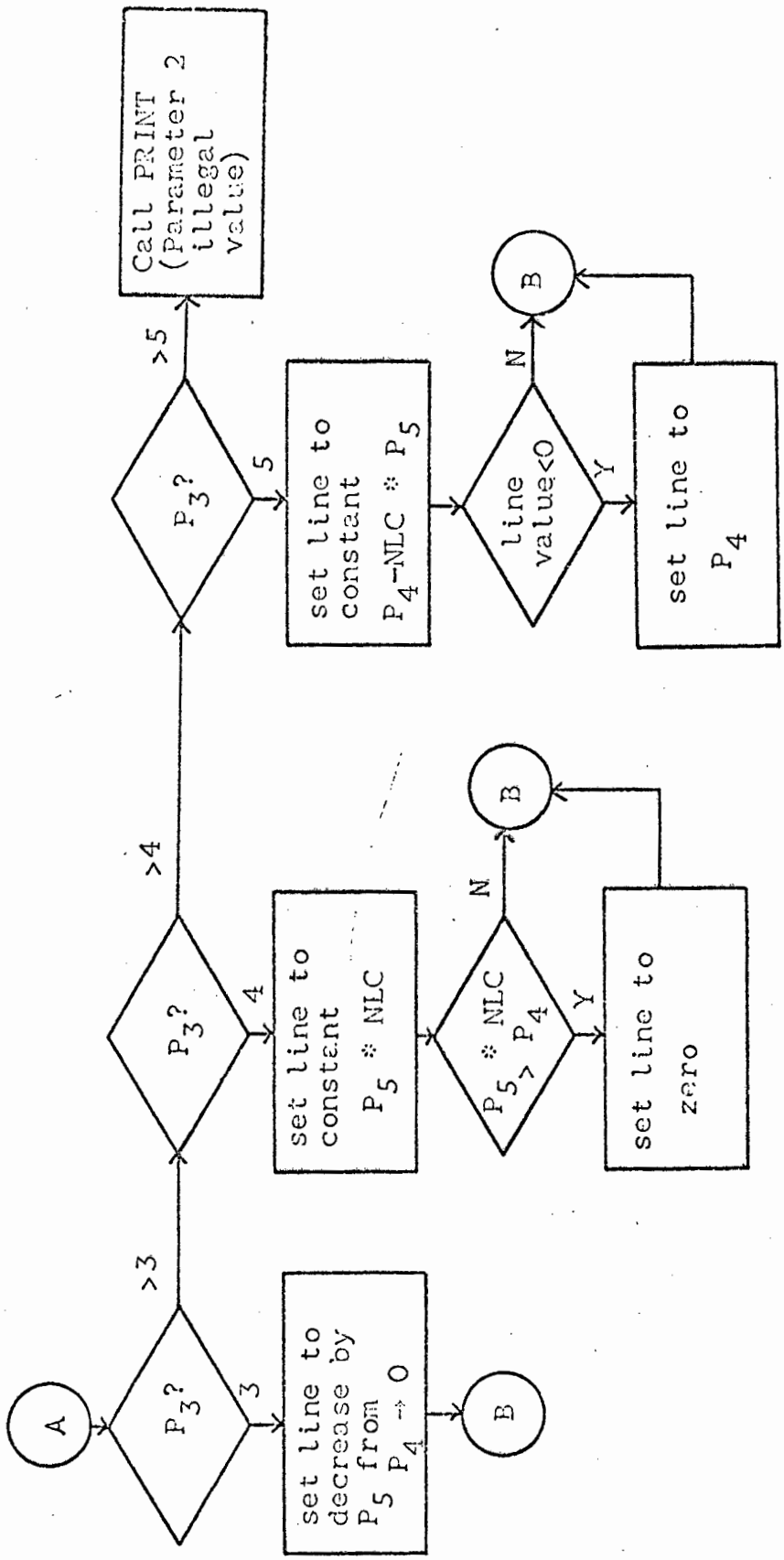


- Figure 51 -

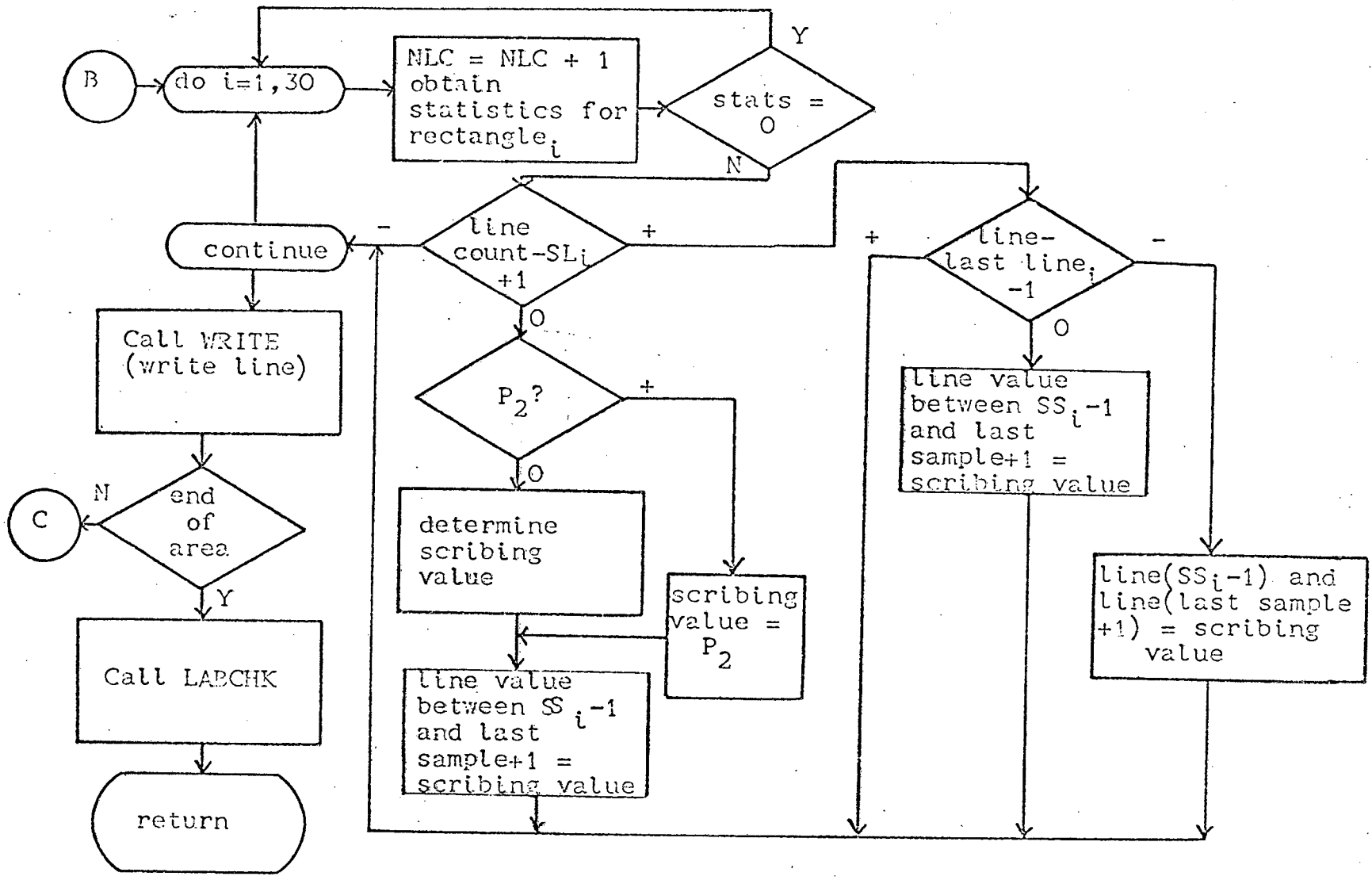
Flowchart for Subroutine SCRIBE



- Figure 52 -

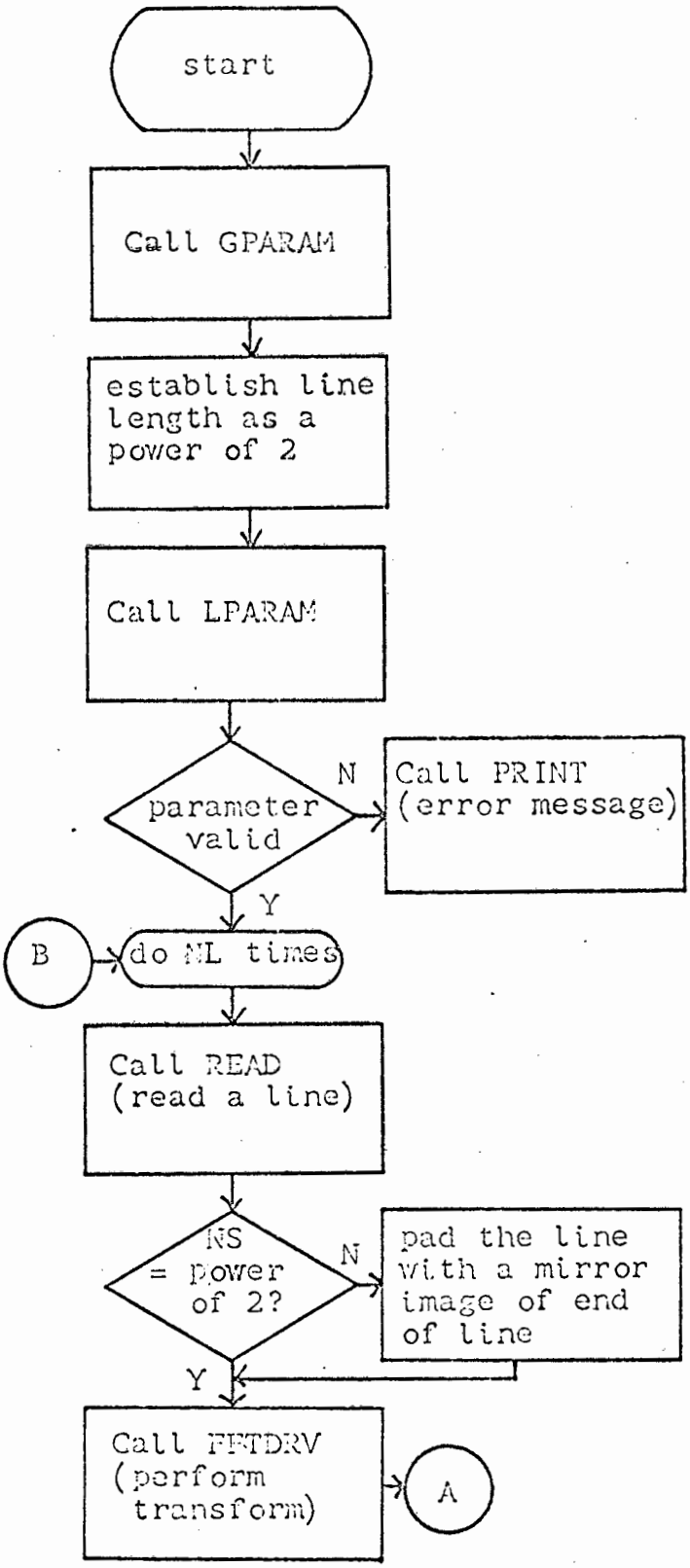

A
P3?
3
set line to decrease by P5 from P4 → 0
B
>3
P3?
4
set line to constant P5 * NLC
P5 * NLC > P4
N
B
Y
set line to zero
>4
P3?
5
set line to constant P4-NLC * P5
line value<0
N
B
Y
set line to P4
>5
Call PRINT (Parameter 2 illegal value)

- Figure 52 -

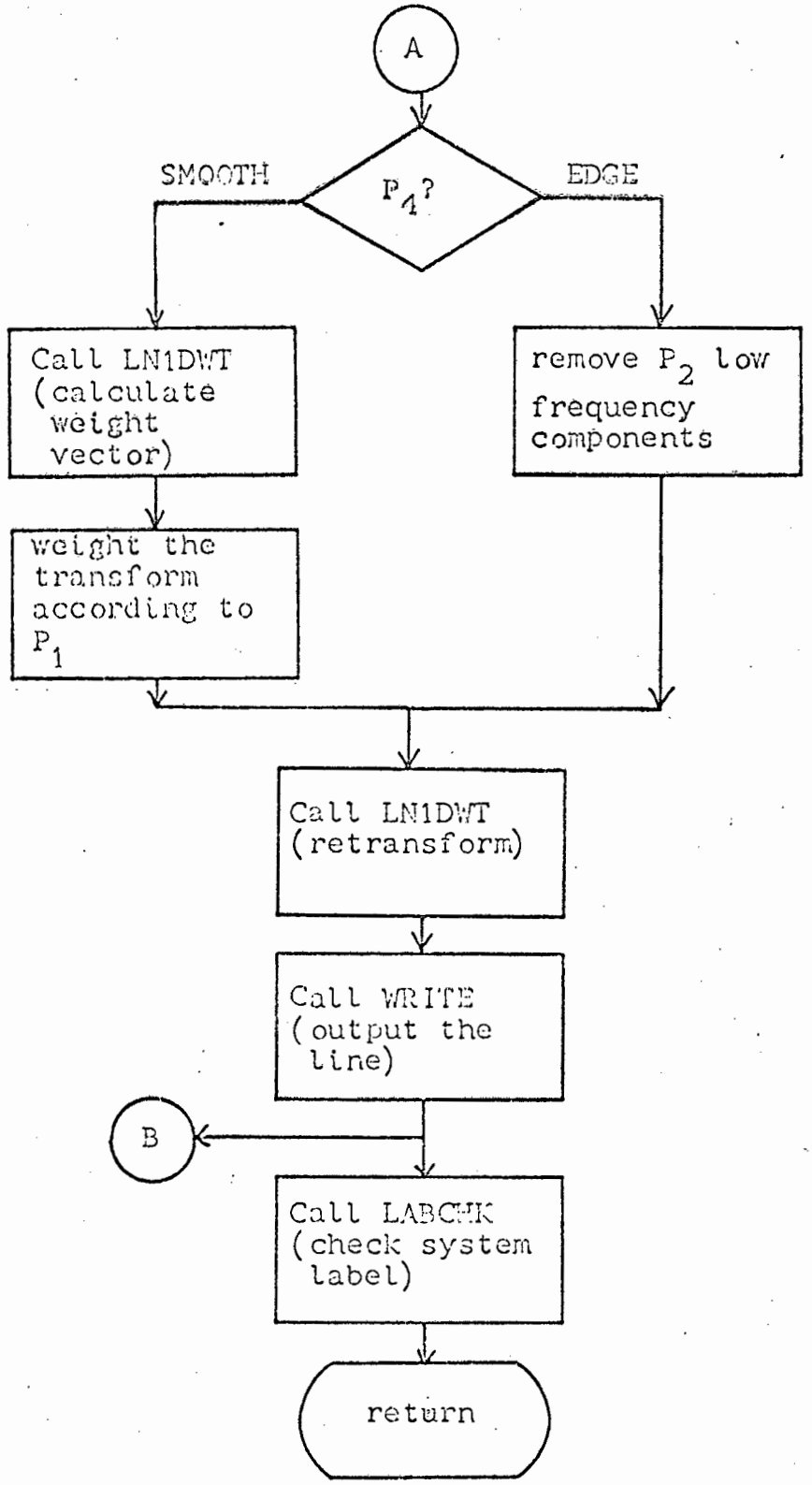

B
do i=1,30
NLC = NLC + 1
obtain
statistics for
rectangle$_i$
stats =
0
Y
N
continue
line
count-SL$_i$
+1
-
+
Call WRITE
(write line)
O
P_2?
+
O
C
N
end
of
area
Y
determine
scribing
value
scribing
value =
P_2
Call LABCHK
line value
between SS$_i$-1
and last
sample+1 =
scribing value
return
+
line-
last line$_i$
-1
-
O
line value
between SS$_i$-1
and last
sample+1 =
scribing value
line(SS$_i$-1) and
line(last sample
+1) = scribing
value

- Figure 52 -

Flowchart for Subroutine TRANSF

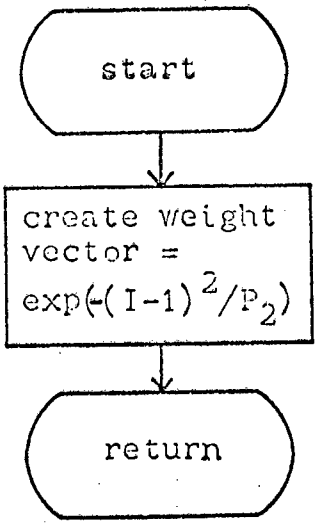


- Figure 53 -


A
SMOOTH
P_1?
EDGE
Call LN1DWT (calculate weight vector)
remove P_2 low frequency components
weight the transform according to P_1
Call LN1DWT (retransform)
Call WRITE (output the line)
B
Call LABCHK (check system label)
return

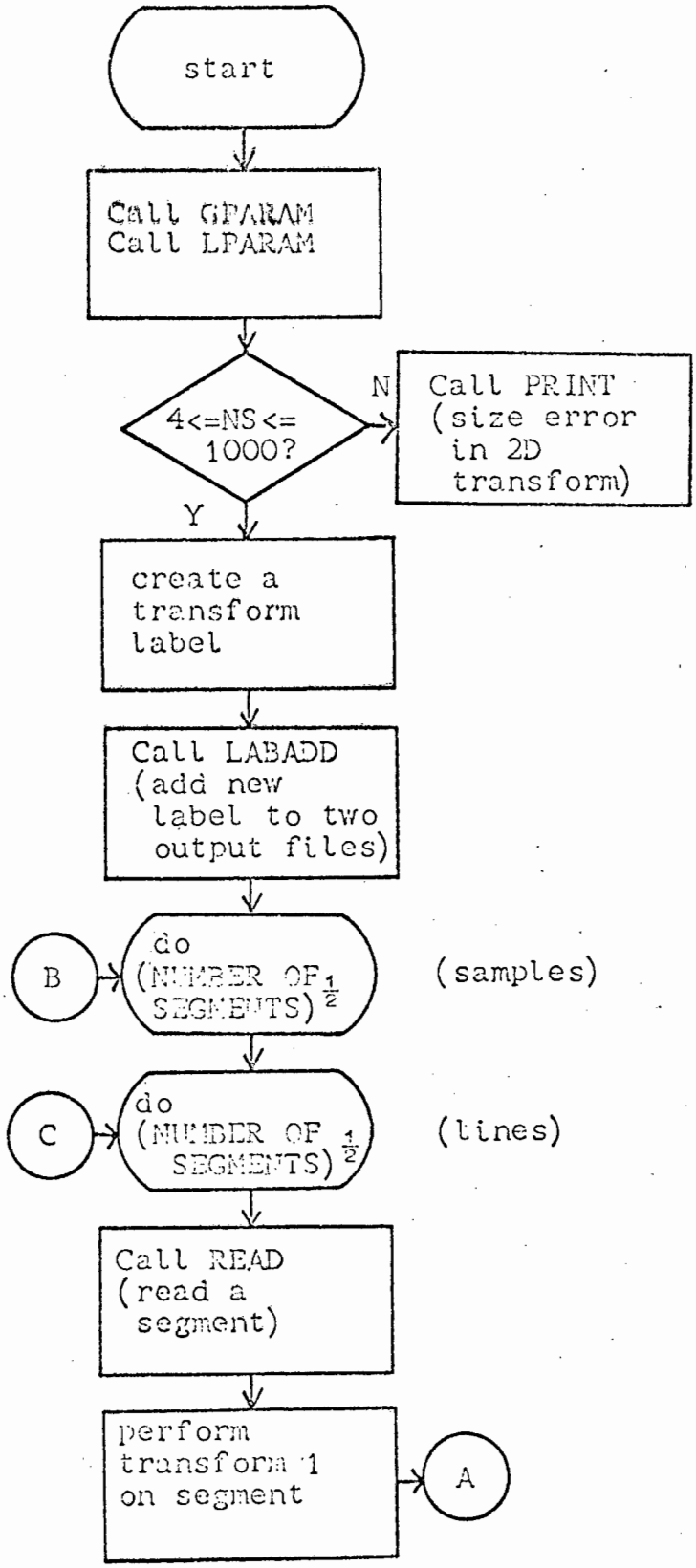
- Figure 53 -

Flowchart for Subroutine LN1DWT

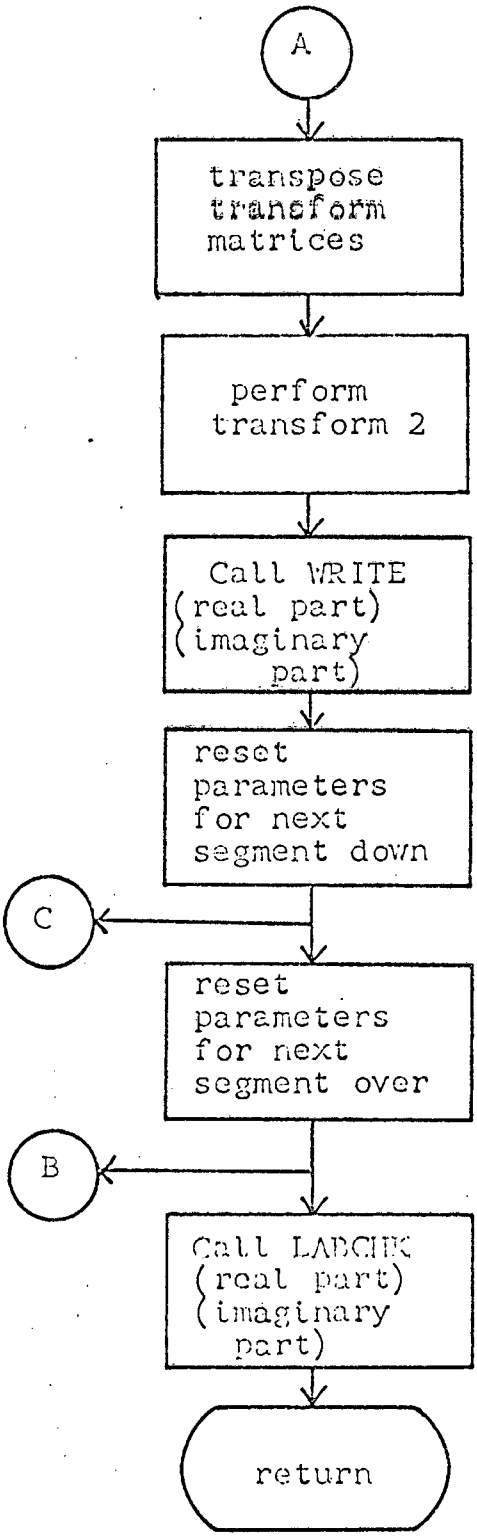


- Figure 54 -

Flowchart for Subroutine TRANS2

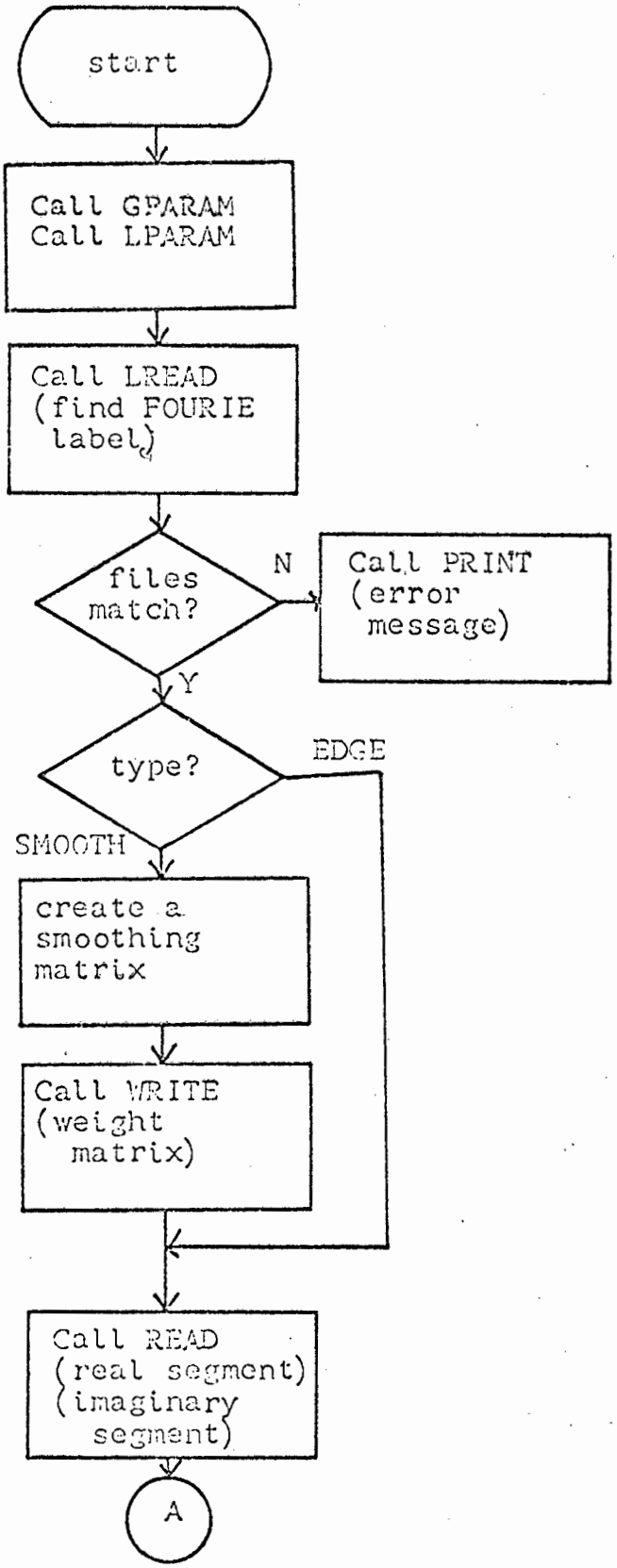


- Figure 55 -

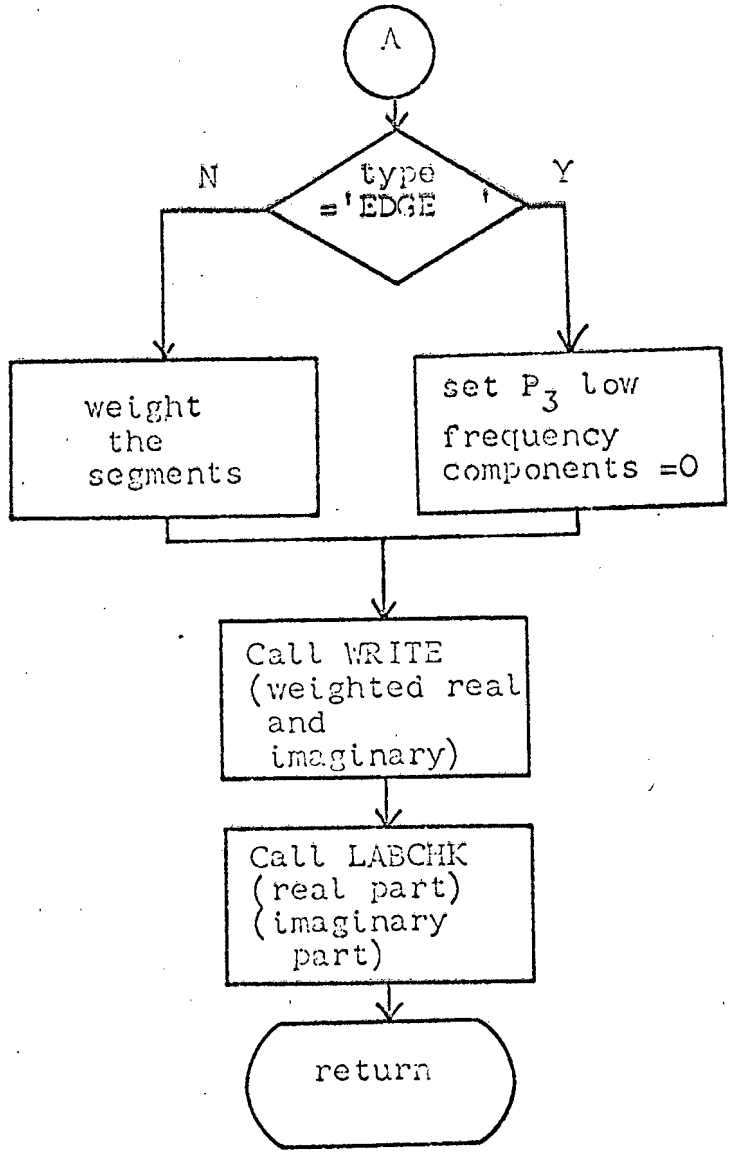

A
transpose transform matrices
perform transform 2
Call WRITE (real part) (imaginary part)
reset parameters for next segment down
C
reset parameters for next segment over
B
Call LABCHK (real part) (imaginary part)
return

- Figure 55 -

Flowchart for Subroutine LN2DWT

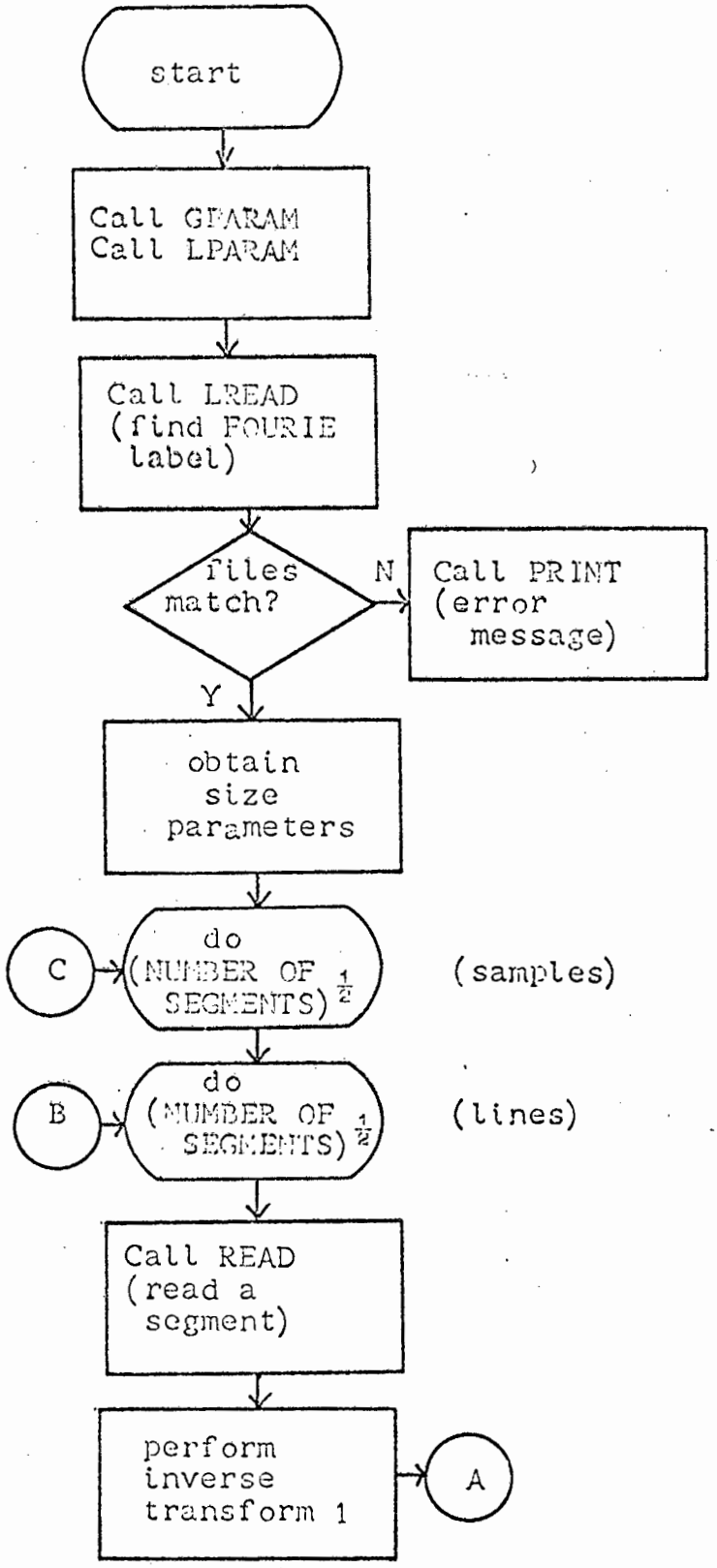


- Figure 56 -

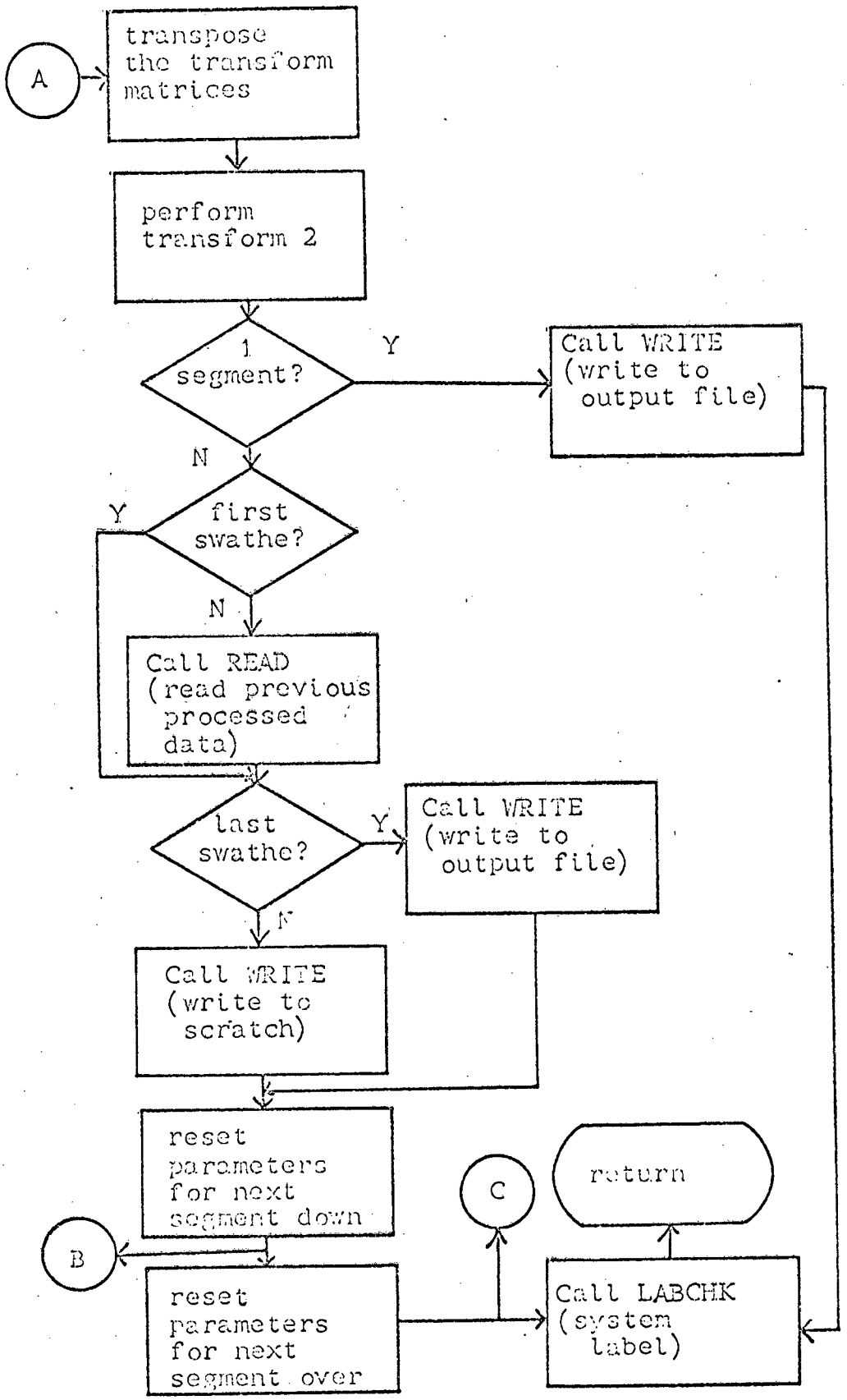

A
type ='EDGE'
N
Y
weight the segments
set P_3 low frequency components =0
Call WRITE (weighted real and imaginary)
Call LABCHK (real part) (imaginary part)
return

- Figure 56 -

Flowchart for Subroutine RETRAN



- Figure 57 -


A
transpose the transform matrices
perform transform 2
1 segment?
Y
Call WRITE (write to output file)
N
Y
first swathe?
N
Call READ (read previous processed data)
last swathe?
Y
Call WRITE (write to output file)
N
Call WRITE (write to scratch)
reset parameters for next segment down
B
reset parameters for next segment over
C
return
Call LABCHK (system label)

- Figure 57 -