A Thesis

Presented to

the Faculty of Graduate Studies and Research

The University of Manitoba

In Partial Fulfillment

of the requirements for the Degree

Master of Science

In Computer Science

---

by

R. J. Bomford

May 1972.

ABSTRACT

The thesis describes a package of programs which enables users of Alberta Gas Trunk Line's Supervisory Control System to develop programs and integrate them into the system in an on-line background mode. The package consists of three independent programs:

(i) UTILITY - which provides the standard utility functions such as:
- listing cards or tape,
- copying cards to tape,
- editing source tapes,

which are necessary for the preparation and updating of source programs.

(ii) SYMBOL - which assembles programs written in XDS Symbol Assembler language (prepared by UTILITY) into machine code for the XDS 920.

(iii) LOADER - which loads the machine code (produced by SYMBOL) and integrates it into the Supervisory System.

A brief history of the evolution of Alberta Gas Trunk Line's Supervisory System is given with a view to demonstrating the justification and urgency of the package.

The appendices contain examples of use of the package within the environment of Alberta Gas Trunk Line's Supervisory System.

ACKNOWLEDGEMENTS

I wish to express my sincere thanks to Professor J. Wells, my supervisor, for his advice and assistance which have been essential to me in the preparation of this thesis.

Special thanks to Dr. S. R. Clark (Victoria) for his interest, comments, and infinite patience.

I would also like to thank Mrs. J. Hauser for her help in typing this thesis.

TABLE OF CONTENTS

Page

Abstract ii

Acknowledgements iv

Table of Figures x

CHAPTER

1 History of Alberta Gas Trunk Line's Supervisory Control System 1

1.1 Original System 1

1.2 System Upgrading 4

1.3 Program Development 6

2 Design Philosophy 12

2.1 Integration with Existing System 12

2.1.1 Executive 12

2.1.2 I/O System.......... 17

2.2 Access to Programs 27

2.3 User Interface 27

2.4 Error Analysis 29

2.5 Statistics 30

2.6 Operator Intervention 31

Page

CHAPTER

3 Utility 33

3.1 Abstract.. 33

3.2 User Interface.. 33

3.2.1 Command Input Device 33

3.2.2 Source Input Device 35

3.2.3 Updated Output Device.......................... 36

3.2.4 List Output Device 36

3.2.5 Operator Message Device 38

3.3 Command Language 39

3.3.1 General 39

3.3.2 Analyzer....................................... 43

3.3.3 Syntax Description 48

3.3.3.1 Blocked By............................. 49

3.3.3.2 Commands From 50

3.3.3.3 Copy 50

3.3.3.4 End 51

3.3.3.5 List Commands.......................... 51

3.3.3.6 List 54

3.3.3.7 Page 54

3.3.3.8 Pause 55

3.3.3.9 Reverse 55

Page

3.3.3.10 Rewind 56

3.3.3.11 Sequence.............................. 56

3.3.3.12 Skip 56

3.3.3.13 Tabs At 57

3.3.3.14 TX 57

3.4 Program Size and Run-Time Statistics............... 58

4 Symbol 59

4.1 Abstract 59

4.2 General Description................................ 59

4.3 Extensions 67

4.3.1 Overlays 68

4.3.2 Symbol Tables 72

4.3.3 Global Definitions.......................... 75

4.3.4 Loader Information 78

4.3.5 Program Statistics.......................... 78

4.3.6 Assembly Options 79

4.3.6.1 Rewind 79

4.3.6.2 List Errors 79

4.3.6.3 List All 79

4.3.6.4 No List 79

4.3.6.5 No BO 79

4.3.6.6 BO 80

4.3.6.7 GO 80

Page

4.3.6.8 TX 80

4.3.6.9 "Delta Records" 80

4.4 Standard Binary Language 82

4.5 Program Size and Run-Time Statistics 83

5 LOADER 86

5.1 Abstract 86

5.2 Objectives 86

5.2.1 Diagnostic and Loading Information 86

5.2.2 Complete or Partial Load 87

5.2.3 No Restrictions on Program Size 87

5.2.4 Multiple Input Tapes 88

5.2.5 System Security 89

5.3 Method 90

5.3.1 General 90

5.3.2 Handling Binary Records 93

5.3.3 Resolving External References 98

5.3.4 Load Address Map 101

5.4 User Commands 103

5.4.1 General 105

5.4.2 Specific Commands 105

5.4.2.1 Rewind 105

5.4.2.2 New System 105

Page

5.4.2.3 GO 106

5.4.2.4 END 106

5.4.2.5 DUMP................................. 106

5.4.2.6 TX 106

5.5 Program Size and Run-Time Statistics 106

6 Conclusions and Further Developments 108

6.1 Utility Development 111

6.2 Symbol Development 112

6.3 Loader Development 113

APPENDIX ....

1 Use of the Package 114

REFERENCES 129

TABLE OF FIGURES

FIGURE		PAGE
1.1	November 1967 Configuration	2
1.2	January 1972 Configuration	5
2.1	Executive	14
2.2	Card Read Handler	19
2.3.1	Read Magnetic Tape	21
2.4.1	Write Magnetic Tape	24
3.1	Typewriter Input	34
3.2	Print Output	37
3.3	Utility Controller	42
3.4	Analyzer	44
3.5.1	Copy/Skip/and List	52
4.1	Symbol Controller	66
4.2	Overlay Handler	70
4.3	Search RAD Resident Symbol Tables	74
4.4	Purge RAD Resident Symbol Tables	77
5.1	Loader Controller	91
5.2	Delta Record Handler	94
5.3	Data Record Handler	95
5.4	POP Reference and Definition Handler	96
5.4	External Reference and Definition Handler	96

FIGURE		PAGE

FIGURE		PAGE
5.5	End Record Handler	99
5.6	Resolving External References	100
5.7	Sort Algorithm	104
A1.1	Console Typewriter I/O	115
A1.2	Generate Coreload	116
A1.3	Test Assembly	117
A1.4	Update	119
A1.5	Test Assembly	120
A1.6	Final Assembly	121
A1.7.1	Load	123

CHAPTER I

HISTORY OF ALBERTA GAS TRUNK LINE'S SUPERVISORY CONTROL SYSTEM

1.1 ORIGINAL SYSTEM

In May of 1966 Alberta Gas Trunk Line contracted Automatic Electric (Canada) Limited to provide a computer controlled Supervisory System to facilitate centralized control and monitoring of Alberta Gas Trunk Line's pipeline system.

The initial system design was undertaken by Automatic Electric in cooperation with Alberta Gas Trunk Line; Automatic Electric provided the hardware and software expertise while Alberta Gas Trunk Line provided the applications objectives. This was the first such system designed and built by Automatic Electric; they planned to develop Alberta Gas Trunk Line's system as a prototype and to market the system as a supervisory package with their CONITEL 2100 communications hardware, Because the system had considerable market potential, Automatic Electric spent a good deal of time and expense incorporating sophisticated programming techniques which normally would not have been justified on a system of this size. After spending five to six man years developing the system, Automatic Electric Management abandoned plans for marketing the system and advised their personnel to complete and deliver Alberta Gas Trunk Line's system.

The original system is illustrated in figure 1.1.

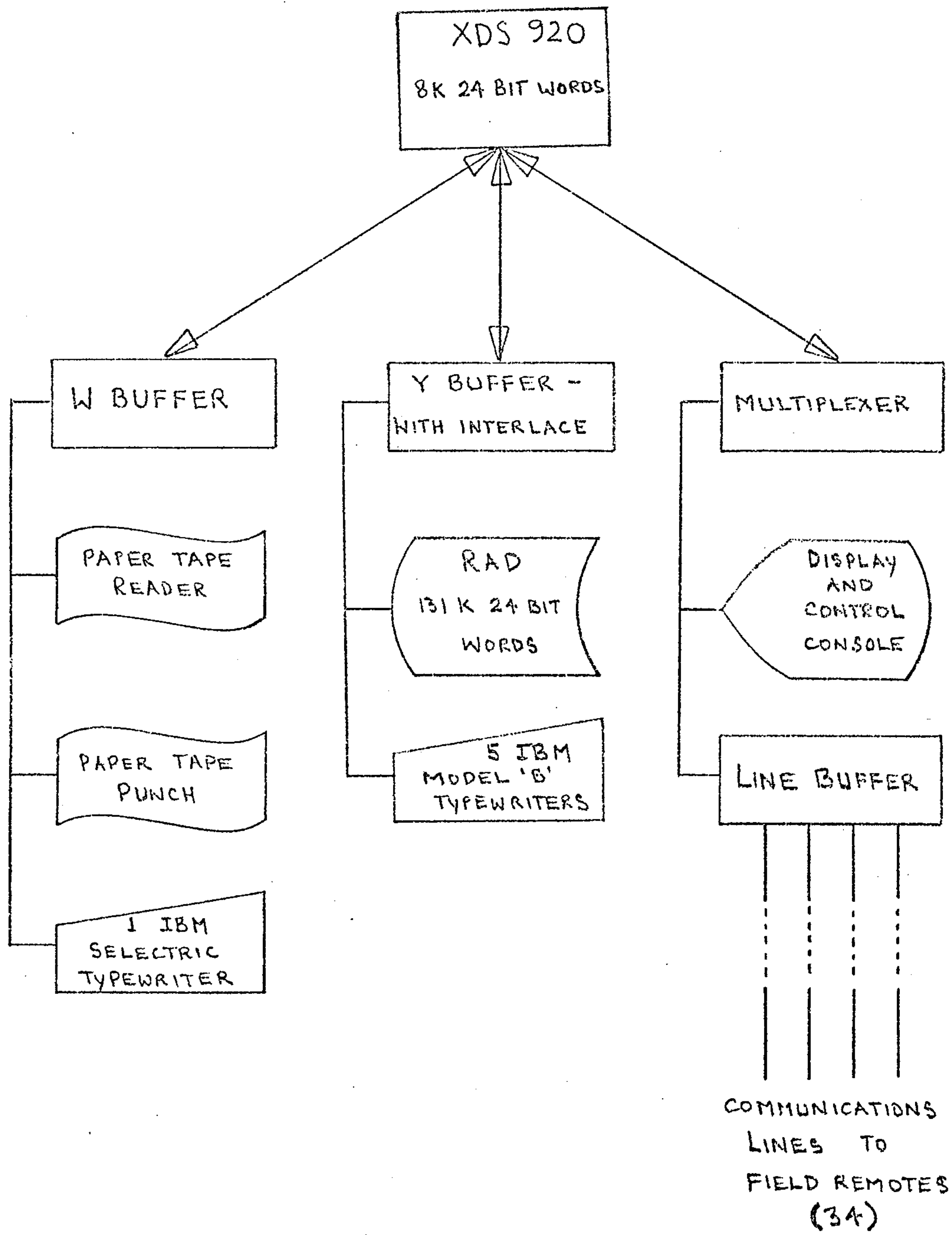


Figure 1.1 November 1967 configuration

As a direct result of the cooperative approach to system design and Automatic Electric's subsequent business decisions, Alberta Gas Trunk Line received a Supervisory System with the following properties:

(i) The system was overdesigned in the "Supervisory" area. The executive and operating system were considerably more sophisticated and complex than required.

(ii) The system was underdesigned in the "Applications" area. The applications programs were an unsuccessful attempt at satisfying Alberta Gas Trunk Line's Gas Control requirements due to a large extent to the fact that Alberta Gas Trunk Line had not analyzed their system requirements adequately prior to specifying the system. The requirements of the Supervisory System are still growing and changing as experience with the system leads to a better understanding of the functions it must perform.

(iii) The system was almost completely neglected in the "support" area. Such essential support programs as utility routines, assemblers, loaders, debugging aids, test routines, and simulators were generally ignored or were far below standard. The obvious reason for this inconsistency was Automatic Electric's decision not to market the system commercially. Because support software is typically the last developed it

was the first area to suffer from Automatic Electric's haste to complete the project. The less obvious explanation for neglect in this area was Alberta Gas Trunk Line's lack of experience, and their inability to forecast the programming changes necessary to keep pace with expansion and growth in the pipeline system.

1.2 SYSTEM UPGRADING

The initial system did not permit in-house program development and updating since such peripherals as card reader, line printer, and magnetic tape drives were not purchased. The first system update, done in the fall of 1968, was taken to Chicago and completed with the facilities and staff supplied by Automatic Electric.

Although of a minor nature, the first update did demonstrate that an alternative to travelling to Chicago and using Automatic Electric staff and equipment had to be developed. Alberta Gas Trunk Line proceeded to expand the system to enable in-house program development and system updating. The hardware was upgraded as illustrated in figure 1.2.

Alberta Gas Trunk Line had considered maintenance of the Supervisory System as requiring only extensions in the applications area - e.g. adding a new station or changing the format of an existing station and/or its associated reports. Alberta Gas Trunk Line had failed to appreciate that the system would have to evolve in overall structure as it was used, and as Gas Control developed new techniques centered around the real-

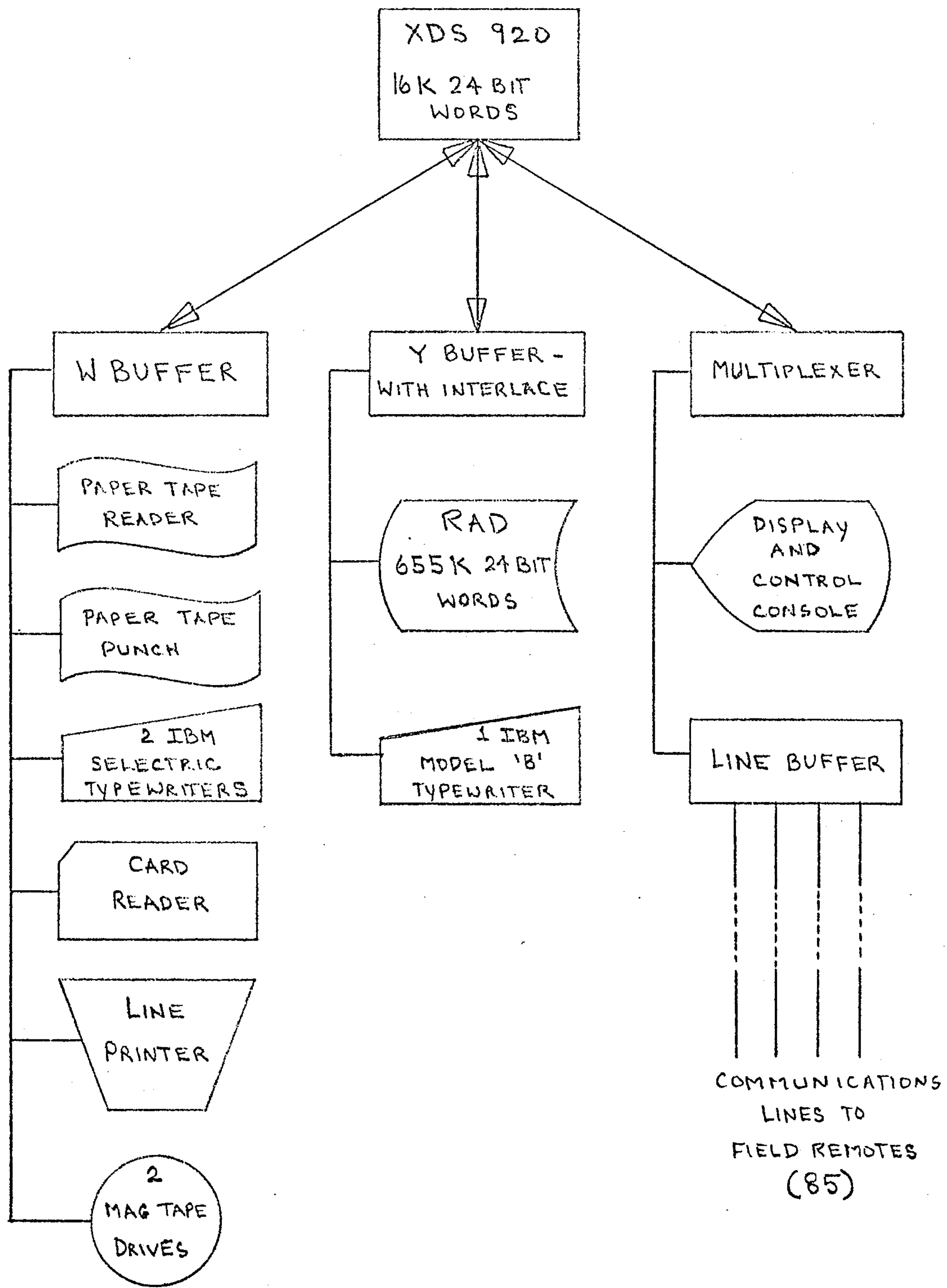


Figure 1.2 January 1972 configuration

time system. Experienced programmers were required to undertake the program development as existing staff did not have a programming background. Coincident with the purchase of the additional hardware, new programming staff were hired - of which the author was one.

1.3 PROGRAM DEVELOPMENT

The first task was to evaluate the present system and to update it since it had not been updated in approximately a year. The changes in the applications programs were extensive, as a large number of new meter and compressor stations had been "patched in" over the year through the I/O typewriter. The maintenance programmers responsible for the modifications had either transferred to other departments or had left the Company, leaving the programming group with an unfamiliar and poorly documented system.

Although the line printer had been purchased for off-line program development, its speed relative to the typewriters made it imperative to incorporate it into the real-time system, and to rebuild all system reports for the printer. Virtually all system reports had to be redesigned and rewritten since the existing reports had been written for 30 inch logging typewriters and had to be reoriented to 120 character lines.

An investigation of the existing software and hardware for program development proved to be most discouraging:

(i) There were no key punches or key punch operators on site - all programs had to be written on coding sheets and sent out for key punching. This required a three to four day turn-around for the modification of a few cards.

(ii) The existing software for program development was "off-line", and very inefficient. To compound the problem, Gas Control was becoming more dependent on the Supervisory System and a conflict of interest rapidly developed.

An evaluation of the system support software illustrated severe inadequacies which had to be corrected before efficient program development could be possible. The software provided consisted of the following three programs, all off-line:

(i) An EDITOR program written in FORTRAN II and still containing several "bugs". The program was extremely slow and provided a restricted set of commands designed specifically for tape editing.

(ii) An ASSEMBLER program which was the XDS supplied SYMBOL Assembler with minor modifications made by Automatic Electric. The SYMBOL assembler itself operated as specified by XDS, the problems being:

- All symbol tables were core resident. With the size of our Supervisory System and the number of global symbols, the core available for symbol table expansion was inadequate with the existing data base design. It was necessary to segment the system and load only the essential symbol tables for each individual system tape in order to keep within available core. Assemblies had to be set up several times, with a consequent loss of time, in order to eliminate as much redundant information as possible.

- There was no provision for "conditional test assemblies".

(iii) A LOADER written by Automatic Electric which operated as specified but which was far below standards which should have been expected. It was admittedly designed for the initial hardware configuration of paper tape input of object code and typewriter listing of the final symbol table (Load Address Map). The programmer-analyst utilized the computer console (lights, push buttons, and toggles) as the user interface rather than the input/output typewriter. It was

impossible to use the program without a detailed list of program halts and a program listing. Output of the "LOAD ADDRESS MAP" was not sorted and therefore was very difficult to search visually for a specific item.

It was clear that support software had to be developed in order for program development and system updating to take place. Priorities had to be established since any delay for extensive support program development would be intolerable to a system so badly in need of revision.

Due to the condition of the system and our unfamiliarity with it, an iterative approach to system updating was adopted. The majority of the computer time was spent on editing and listing tapes with an infrequent assembly to indicate the remaining "bugs" to be eliminated. The initial improvements made in the support software area were as follows:

(i) A minor modification was made to the assembler, permitting the user to specify options necessary for "test assemblies" e.g. list all or list errors, binary output (of object code) or no binary output. The typical assembly time for "test assemblies" was reduced considerably by not requiring a complete listing.

(ii) An off-line UTILITY package was developed to replace the tape editor written in FORTRAN. The package was programmed in SYMBOL and was therefore able to take advantage of hardware features not available to FORTRAN, e.g. tape scan features, continuous tape reading, etc. The significant result of this UTILITY package, aside from the tremendous saving in time, was the experience gained which formed the basis for the present on-line version developed and incorporated into this thesis.

(iii) The LOADER was modified to permit binary input of object code from magnetic tape but was otherwise left virtually intact. Very little downtime was required for system loading at this point, and the time necessary to redesign and reprogram the LOADER could not be spared.

In spite of the significant improvements in the support programs, the first major update took in excess of six months to complete! The hardships suffered by Gas Control for that period are incalculable - because of program development they were without the Supervisory System for an average of three to four hours a day. The emergency backup system proved to be inadequate and was rarely used.

At this point Management was approached with the proposal that the author be given the necessary time and assistance to develop as a thesis topic a package of programs which would enable "on-line" program preparation and system updating. Management was easily convinced of the necessity of such a package and guaranteed their cooperation.

CHAPTER 2

DESIGN PHILOSOPHY

This chapter outlines the common areas of design philosophy arising as a consequence of the package being designed for the same operating system and the same users.

2.1 INTEGRATION WITH THE EXISTING SYSTEM

Because the package is designed for an existing environment, the interface with the operating system is critical, and will be described in some detail for those areas which require special attention.

2.1.1 EXECUTIVE

The real-time executive is a standard multiprogramming executive employing simple round robin scheduling. The executive allocates the remaining core memory not used by the core resident operating system to four program partitions associated with job priorities according to the following table:

PRIORITY	FUNCTION
1	Data retrieval and system scan.
2	System calculations and report formatting.
3	Output of summary reports (background).
4	Background jobs such as: - program patching and inspection, - system simulation, - program development and system updating.

The executive algorithm is flow charted in figure 2.1. The queueing scheme is FIFO and no job swapping takes place. Once a job comes to the head of its job queue, it remains the only active job in that priority until it is completed or otherwise released. Lower priority programs are effectively blocked from the system until a higher priority job requests I/O and waits for the I/O to finish. When a job is completed, the executive drops to the next lower priority to prevent blocking of jobs when a high priority queue builds up.

Although the executive determines "when" jobs are to be run, other portions of the operating system initiate the jobs by attaching them to their job queues. Jobs are linked to their job queues when any of the following external stimuli are sensed:

(i) Line Buffer Interrupt - Which saves each block of data from a remote station and queues the data processing routine when the data is complete i.e. information from each remote is sent as several "blocks", each causing an interrupt to the CPU from the line buffer.

(ii) Push Button Interrupt - Which queues a program to analyze the cause for the interrupt and to execute the desired function.

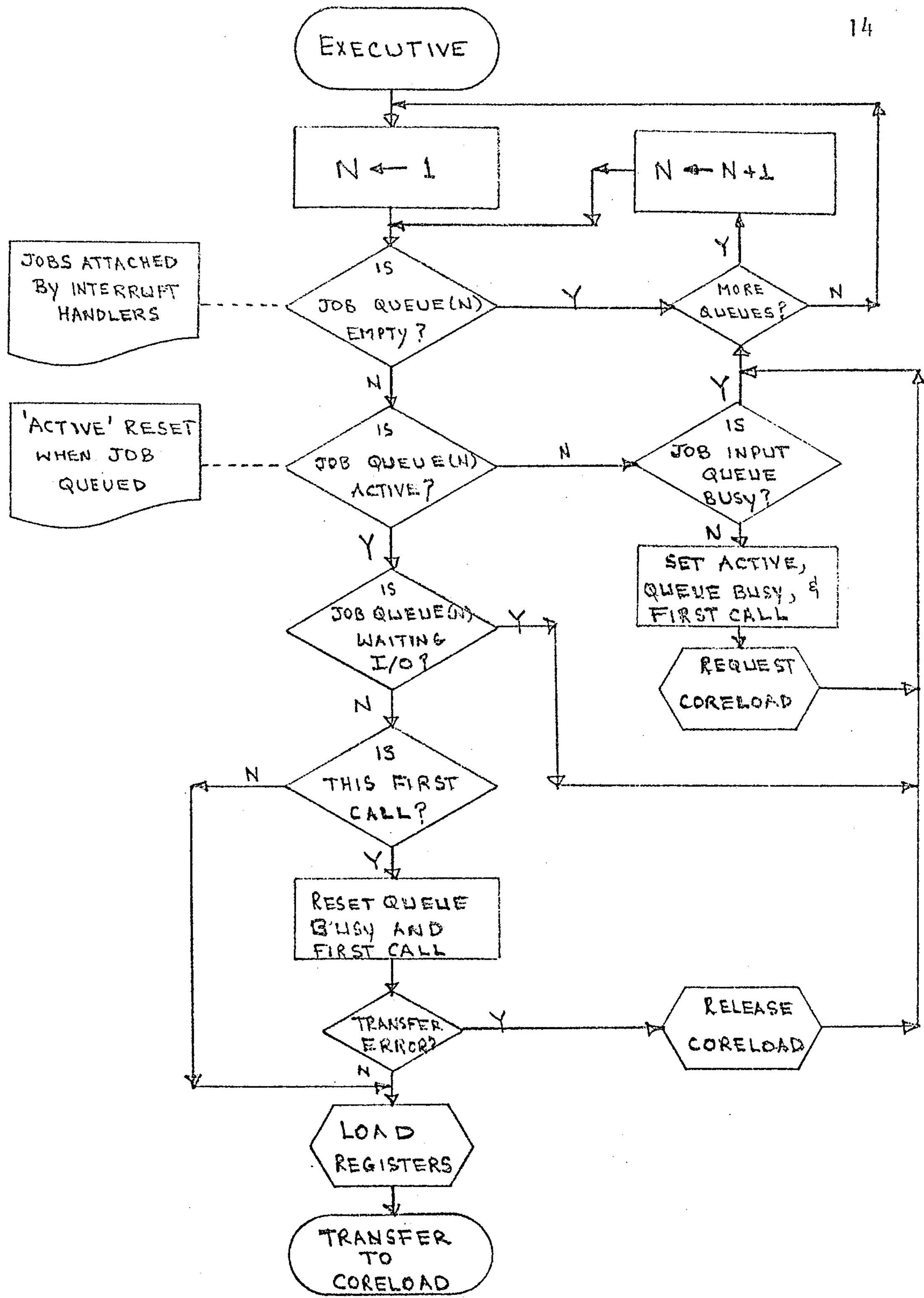


Figure 2.1 Executive

(iii) User Requests - Which queue jobs as commands from the I/O typewriters are analyzed.

External interrupts cause the executive to restart its loop at priority one since the interrupt may have queued a higher priority job than the one being executed when the interrupt occurred.

Integration of the package with the executive presents no difficulty since it is designed to handle such jobs at user request (iii). An initial consideration is to ensure that adequate memory is available. It was possible to allocate only 4K words of core memory to priority four without seriously degrading the operation of foreground jobs. Experience with the XDS SYMBOL assembler indicated that auxiliary storage would be necessary to supplement the limited core available. RAD storage provided the logical solution because of its availability (500K) and its fast access time.

The only core resident structures in the system are:

(i) the executive,

(ii) the I/O system,

(iii) the interrupt handlers,

(iv) the system POP library.

All other information resides on the RAD in structures known as coreloads. This structure has been created to provide a means of referencing RAD structure by name, with the SYMBOLIC reference being assigned a RAD address at load time. The scheme is analogous to that provided by SYMBOL for inter-program communication of SYMBOLIC core locations, i.e. external definitions and external references which are associated at load time. A coreload definition is created by prefixing a group of source programs with a "delta record" which provides the LOADER with the SYMBOLIC name and execution bias to be used (see Sections 4.3.4 and 5.3 for a further description of delta records). All programs between delta records are considered to be part of the previous coreload and therefore the length of the coreload can be determined by the LOADER. The RAD address associated with a coreload requires a complete word (24 bits) to represent it as 14 bits are required for the starting RAD address and 8 bits are required for the length (in 64 word "sectors").

The LOADER relocates all information to its execution address determined from the "priority" supplied in the delta record. Dynamic program relocation is prohibitive because of the absence of base-displacement addressing.

2.1.2 INPUT/OUTPUT SYSTEM

The objective of providing an I/O system in a real-time environment is primarily to provide a scheduling mechanism which will maximize throughput by overlapping processing and I/O. Providing a standardized and efficient package allows the applications programmer to take full advantage of the capabilities of the peripherals without becoming involved with the specialized techniques of I/O programming.

Absence of interlace hardware on one of the two I/O channels makes overlapping of I/O and processing an impossibility for the "higher speed" devices on this channel for the following reasons:

(i) Once I/O is begun the CPU must be able to transfer information at the rate of the device or a transfer error will occur. The physical rate of movement of the device generally determines the minimum rate of transfer.

(ii) The absence of an interlace to synchronize block data transfers makes it necessary for the CPU to word transfer information on this channel.

(iii) The operating system is not re-entrant and therefore disables the interrupt system when performing "system" functions. The interrupt system is disabled for sufficiently long periods to cause a transfer error as in (i) above for "higher speed" devices.

The above considerations make it necessary to disable the interrupt system and dedicate the CPU to the device for such peripherals as card reader, line printer, and magnetic tape. A line printer handler was incorporated into the I/O system because it was required by so many programs but handlers for the magnetic tapes and card reader, being in lesser demand, were included only in the programs requiring them.

The handlers to be described are those in UTILITY as they are representative of those provided in each of the programs.

(i) CARD READER

The handler flow charted in figure 2.2 reads eighty column BCD cards from the card reader. Feed check and validity errors cause a message to be output to the user, requiring him to restart the job to reread the card. A time-out routine is incorporated to produce messages to the user at regular intervals if the card reader is not ready.

(ii) MAGNETIC TAPE

The magnetic tape handlers take advantage of the fact that all records are fixed length card images to utilize a continuous read (or write) technique. Overhead is decreased considerably by requiring only a single read (or write) to dispose of several records in a buffer area, without stopping and starting for each inter-record gap as is conventional. The gap is read (or written) but the tape is given a

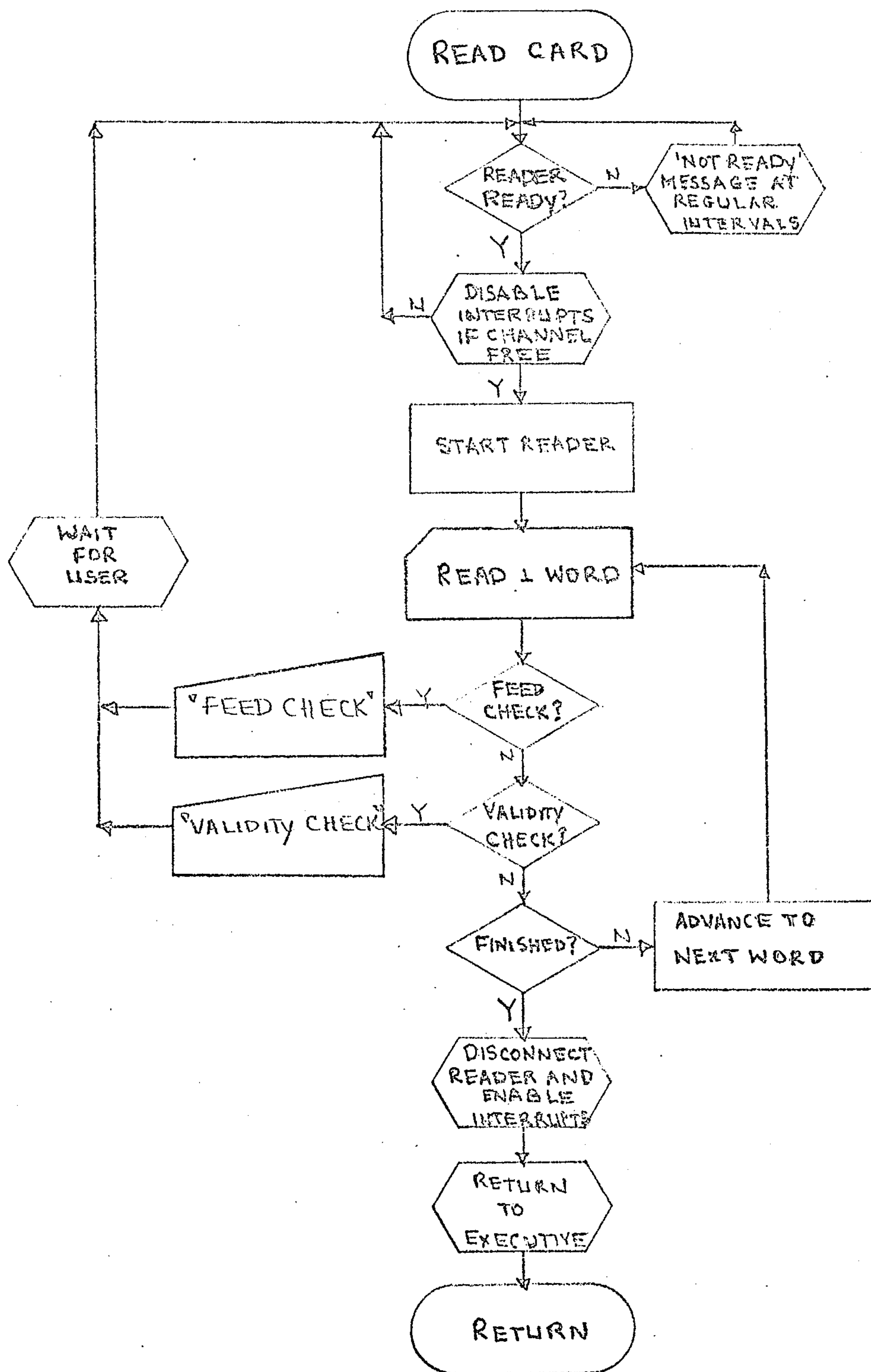


Figure 2.2   Card Read Handler

command to continue before it has had time to stop and become not ready.

The buffering scheme saves considerably on overhead since the tape is moved only when the buffer area becomes empty (or full). The number of records that can be handled by one read (or write) is a function of:

(i) the buffer size,

(ii) the number of records per block (blocking factor),

(iii) the maximum time the interrupt system can be disabled without losing information from the line buffer.

The handlers employ a time-out routine to produce messages to the user at regular intervals if the tape unit is not ready.

READ

The magnetic tape read handler is flow charted in figure 2.3.1. Each call to the handler will return the next record from the read buffer; the tape is physically read only when the buffer area becomes empty.

WRITE

The magnetic tape write handler is flow charted in figure 2.4.1. Each call to the handler will store the record in the write buffer; the tape is physically written only when the buffer area becomes full.

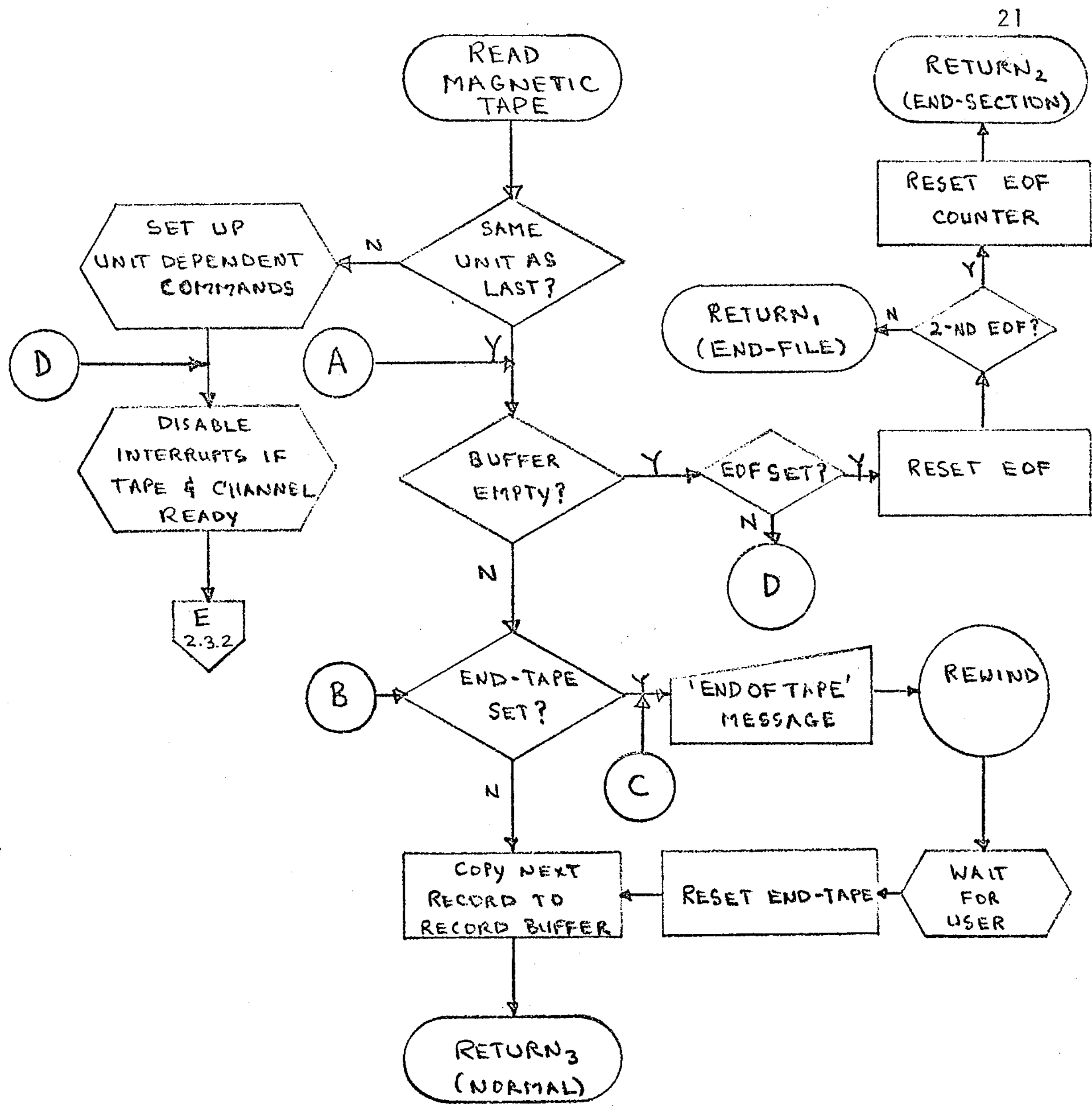


Figure 2.3.1 Read Magnetic Tape

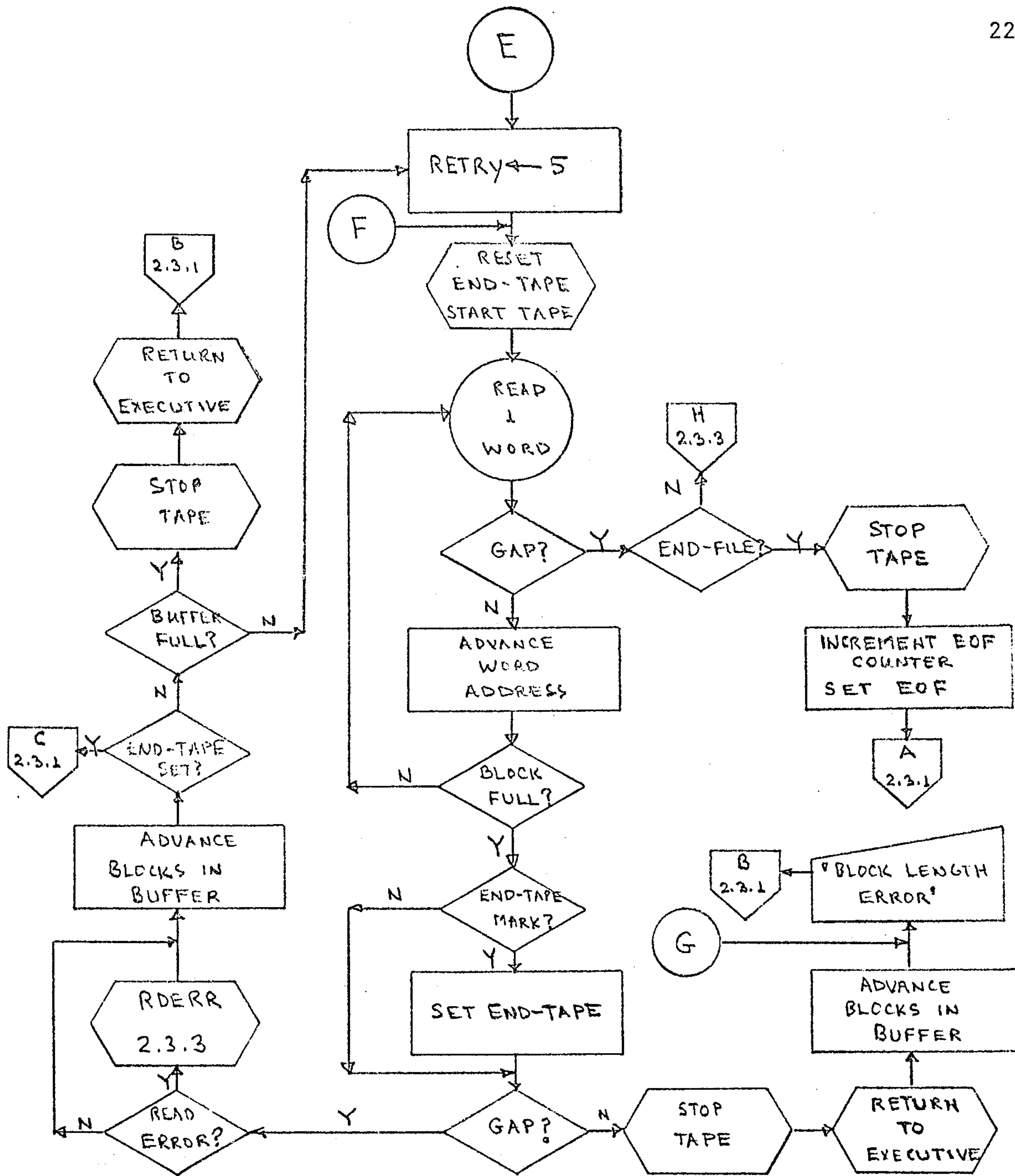


Figure 2.3.2

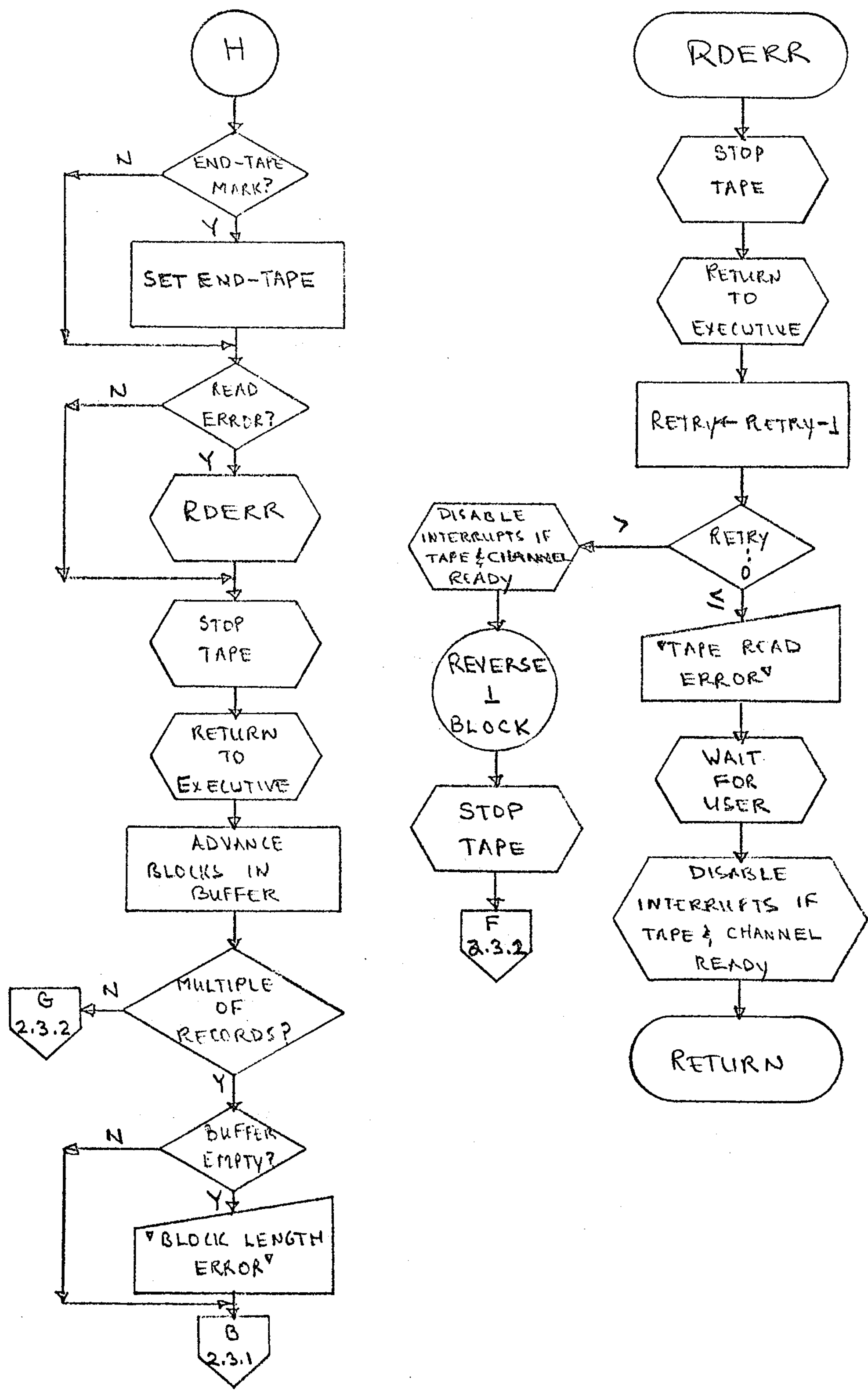


Figure 2.3.3

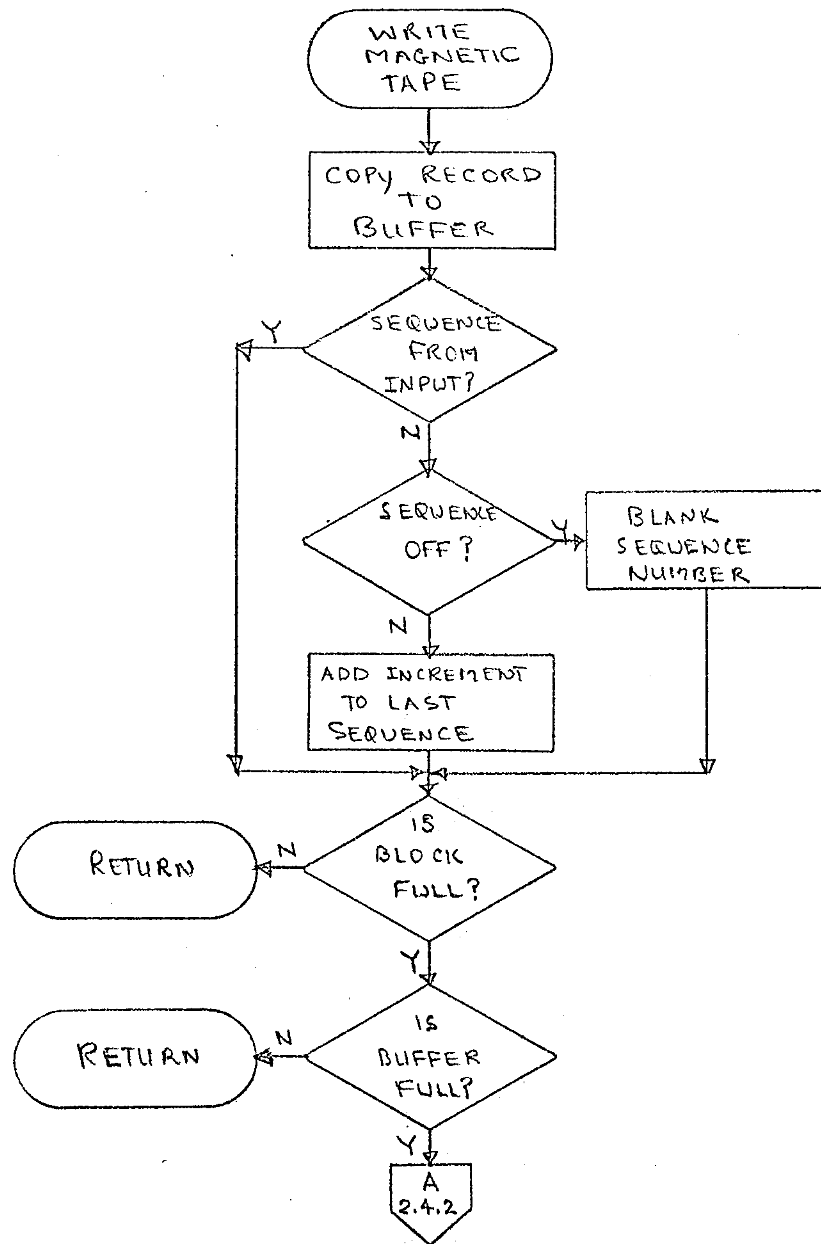


Figure 2.4.1 Write Magnetic Tape

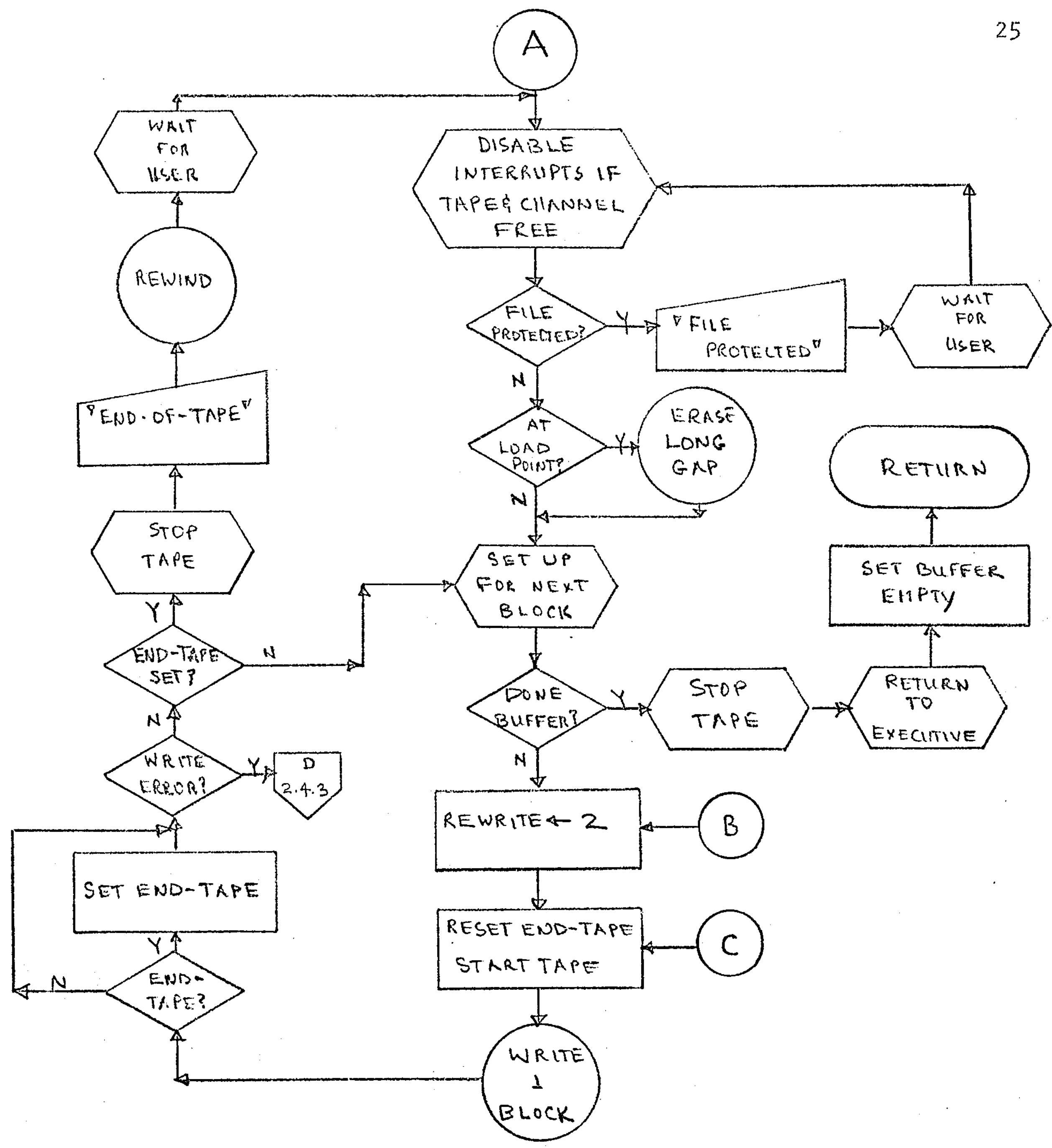


Figure 2.4.2

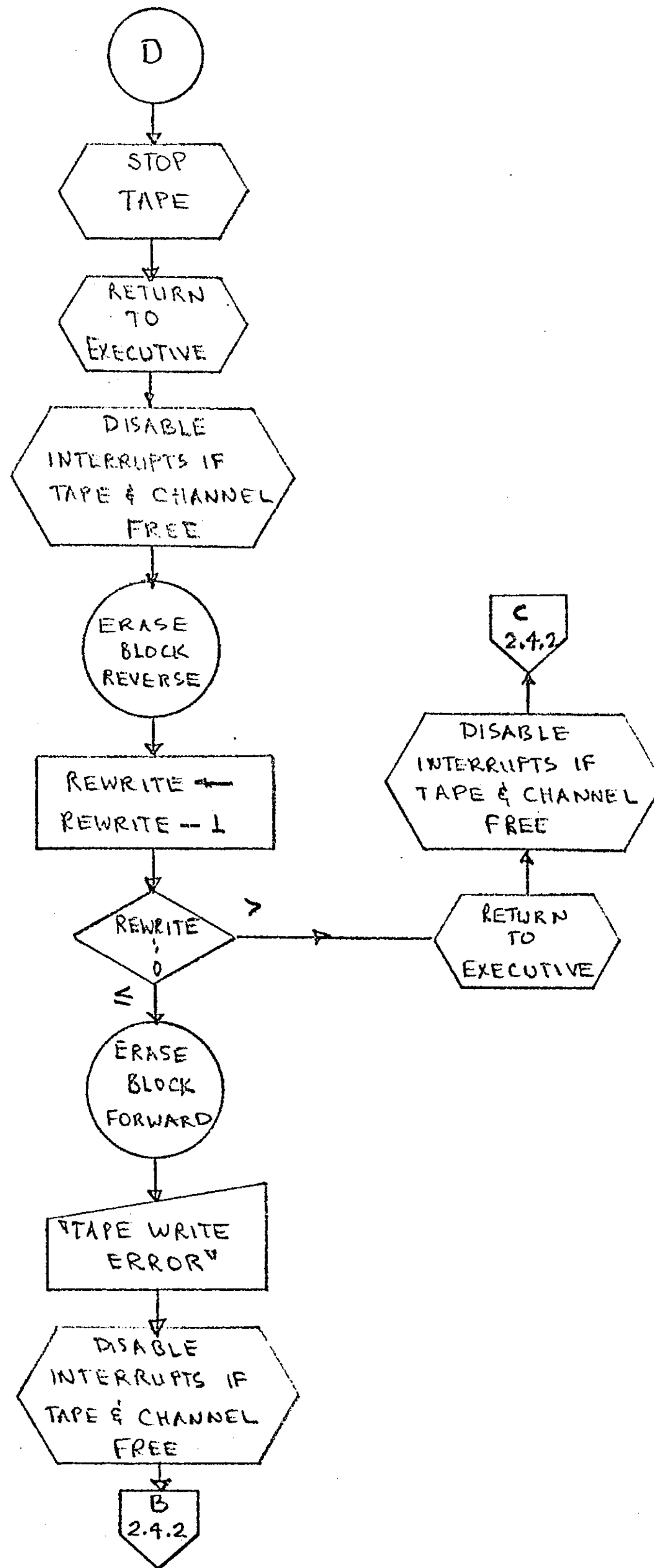


Figure 2.4.3

2.2 ACCESS TO PROGRAMS

Each program is requested by typing its call letters - UTILITY, SYMBOL, or LOADER - as a standard request through either I/O typewriter. The program will be loaded only if no other jobs are active in that priority; otherwise a message is output advising the user to try later rather than queue the job to its job queue. Since the user must be available to provide the options requested by the program, it may not be practical for him to wait for the job to come to the head of the job queue if a long job (e.g. a simulation) is already in progress in priority 4.

2.3 USER INTERFACE

The software and hardware forming the user interface were chosen with the following primary considerations:

(i) Device Speed and Availability

The physical size of the Supervisory System in source form (100,000 records) dictated very early that all source would be kept on magnetic tape and revisions only (i.e. updates) would be input from the card reader. RAD storage is used whenever possible because of faster access and because RAD I/O, being on the interlaced channel, can be overlapped with processing. Any large amounts of printed output are routed to the line printer in preference to the typewriter.

The design of the I/O system makes it necessary to ensure externally that conflicts over use of peripherals do not occur. Conventional I/O in this system is not spooled but queued directly to the device queue and input (or output) as soon as possible (FIFO). Because more than one priority may be competing for use of the line printer, it is necessary to reserve the printer for the job in progress and not permit other priorities to share the device. Background jobs are required to set a "reserve printer" flag if they must use the printer and to reset this flag when they are finished. The I/O system aborts all other requests for the printer as though they never occurred until the flag is reset. Background jobs are given this priority because their output is normally of relatively long duration and because all system output is "on demand" and can be requested again when the background job is completed. See Section 2.6 for a means of allowing urgent reports when a background job is in progress.

(ii) Real-time Environment

The programs are to be run as "background" jobs which implies that they cannot interfere with or delay the "foreground" task of monitoring and controlling the pipeline. All instances where the program is required to wait for status to change in order to proceed (e.g. device ready, user input) are in a "wait loop" which returns control to the

executive between tests, allowing other jobs to use this non productive time.

(iii) User

Use of the system is on an "open shop" basis where each programmer is responsible for running his own job. The primary user of the package is the maintenance programmer whose responsibilities include routine system updating and maintenance. All communication between the program and the user (e.g. error messages, user requests, and user input) must be specific and unambiguous since the user is neither an experienced programmer nor an operator.

(iv) Flexibility

It was known that areas of the specific programs would undergo numerous revisions and improvements before the user requirements were satisfied. All programs were segmented and built as modules which can be individually modified or replaced. Command decoders are constructed so that the syntax of the commands can be changed without restructuring and reprogramming the associated decoding and analysis logic.

2.4 ERROR ANALYSIS

Error analysis is performed at two levels in each of the programs:

(i) At the program level

Diagnostic messages output as a result of detecting operational and logic errors during execution are detailed and specific. The programs are written to trap as many errors as possible

and to force the user to correct the problem before proceeding. All error messages have been designed and formatted to provide the user with the maximum available information in order that he may debug the problem interactively.

(ii) At the user level

Diagnostic output as a result of detecting syntactic errors in user input is over-simplified and provides no specific indication of the reason for the error other than the message "SYNTAX ERROR".

The simplicity of the command language structures permits this approach since the user is able to visually detect almost all errors with no difficulty.

2.5 STATISTICS

Each program in the package provides statistics output when the job is released, giving job time in hours, minutes, and seconds and total program usage (since the last system load) in hours and minutes. These statistics are useful for estimating programming activity and for projecting system time required for future projects. The format of this line can be seen in the examples in the appendix.

2:6 OPERATOR INTERVENTION

It is frequently necessary to delay or abort the job once it has started for either of the following reasons:

(i) Errors are discovered which make continuing the job unnecessary.

(ii) Gas Control may require a report and cannot wait for the job to finish.

Three commands are available through the I/O typewriter which accomplish the necessary intervention:

HOLD - which causes the background program to empty its print buffer, skip a page, release the printer, and wait for the job to be released or restarted.

CONTINUE - which causes the background job to reserve the printer and continue from the point of interruption.

RELEASE - which causes the background job to empty its I/O buffers, skip a page, release the printer, and release the job from the system.

These commands are input to the system command decoder which operates in a foreground priority and therefore will take precedence over the background job. The commands alter only the "reserve printer"

flag which the background job must test to initiate the action described. Requesting the commands when no background job is active has no effect on the rest of the system.

CHAPTER 3

UTILITY

3.1 ABSTRACT

The UTILITY program provides an on-line means of performing the standard utility functions necessary for program preparation and updating. Such functions as:

(i) listing cards / tape,

(ii) copying card / typewriter / tape input to tape,

(iii) editing tape with changes from cards / typewriter,

are typical of those implemented.

3.2 USER INTERFACE

The user interface consists of five functions assigned to I/O devices according to the following:

3.2.1 COMMAND INPUT DEVICE (CID)

This function handles input of all user commands, including new source records. This function is assigned by default to the card reader but can be assigned by the user to the typewriter.

The handler incorporated for reading cards is described in Section 2.1.2 and is flow charted in figure 2.2.

Although typewriter I/O is handled by the I/O system, the handler flow charted in figure 3.1 is required to perform the following functions on each input line:

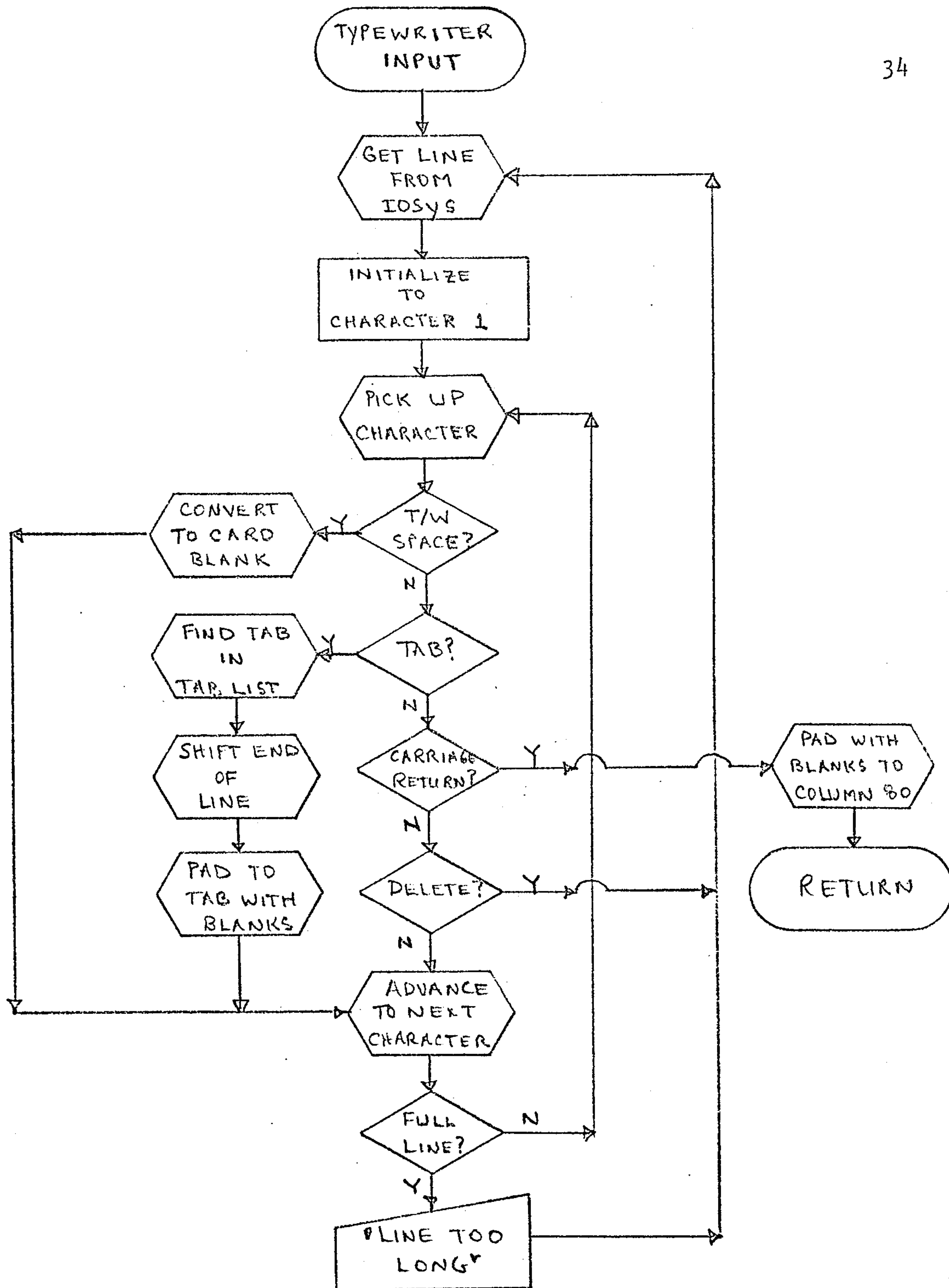


Figure 3.1 Typewriter Input

- All typewriter spaces are converted to card blanks for compatibility with card input.
- All records containing tabs are reconstructed to full 80 character images. The SYMBOL tab list will be used by default (columns 8, 16, 36, and 73), unless a tab list is assigned by the user.
- All records are padded with blanks to a full 80 characters.
- Detection of a "delete" character will delete the line and request the next.

3.2.2 SOURCE INPUT DEVICE (SID)

This function handles input of all source information to be updated. This function is unconditionally assigned to magnetic tape unit one and is referred to as the "INPUT tape". The device is not referenced when generating a new program from the CID as there are no records "to be updated".

The handler provided for reading magnetic tape is described in Section 2.1.2 and flow charted in figure 2.3.1. If a blocking factor is not specified, the handler assumes one record per block (unblocked). If the blocking factor on the input tape is greater than the specified (or default) blocking factor, a "block length error" will be detected and an appropriate message will be output to the user.

3.2.3. UPDATED OUTPUT DEVICE (UOD)

This function handles output of all source records. This function is unconditionally assigned to magnetic tape unit two and is referred to as the "OUTPUT tape". The device is referenced in all cases where a program is being generated.

The handler provided for writing magnetic tape is described in Section 2.1.2 and flow charted in figure 2.4.1. If a blocking factor is not specified, the handler will write one record per block (unblocked).

3.2.4 LIST OUTPUT DEVICE (LOD)

This function handles listing of all commands as they are executed and all specific listing requests. This function is normally assigned to the line printer and can be changed only by changing the device assignment in the I/O system.

Although the line printer output is handled by the I/O system, the handler flow charted in figure 3.2 is required to perform the following functions on each record to be printed:

- It strips the trailing blanks from each record, packs several records into a buffer area, and outputs the buffer area to the RAD for future output by the printer. UTILITY can thus continue processing without having to wait for the line to be printed.

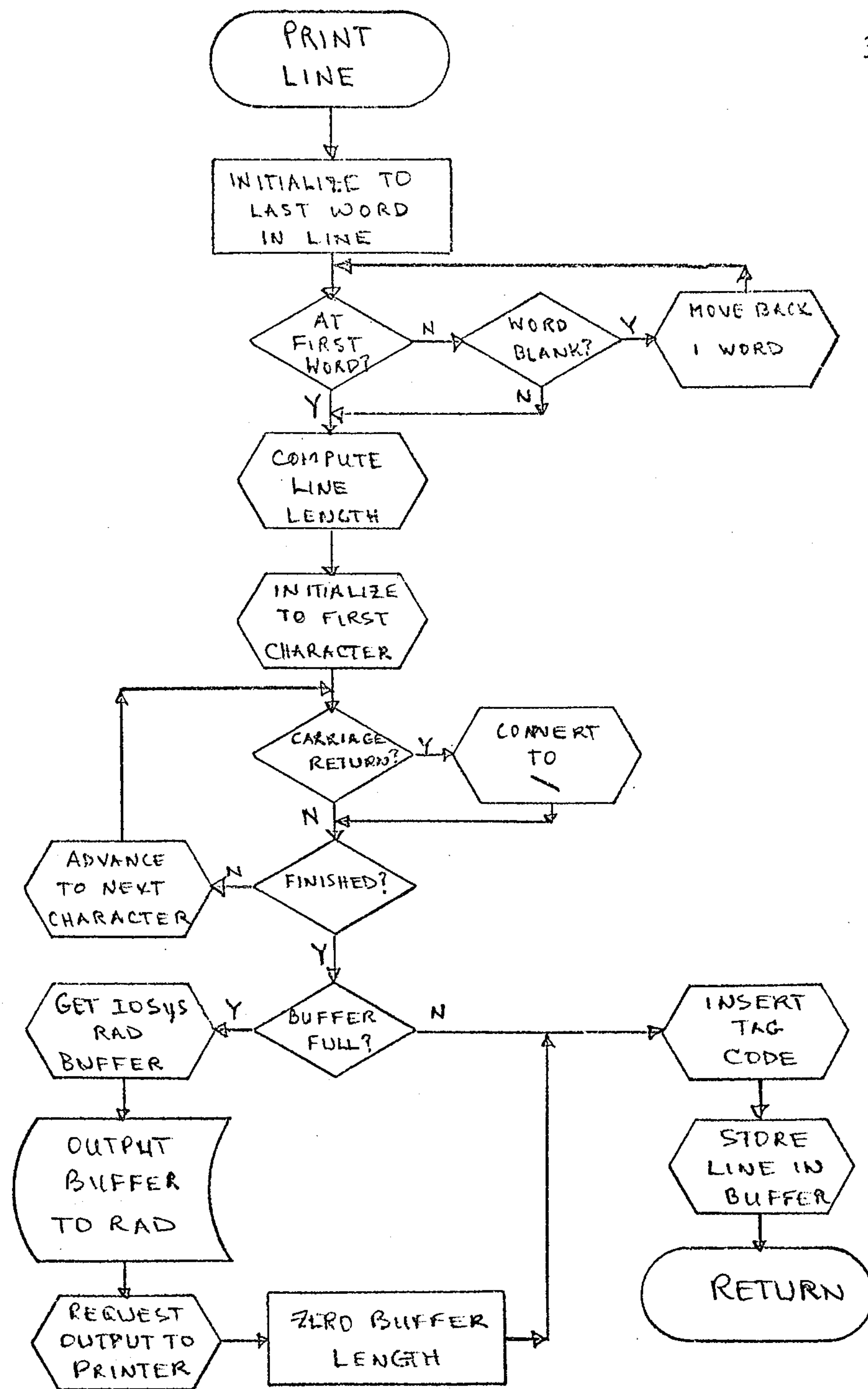


Figure 3.2 Print Output

- All carriage return codes imbedded in the output line are translated to the printable character \. Besides not being printable, carriage return codes are used by the I/O system to indicate end-of-line and would cause an erroneous line advance.

- An identification tag is attached to the beginning of each output line indicating the type of record being printed. These tags form an "edit trail" for possible future bench checking. An index is provided as an argument in each call to the handler to indicate the tag to be appended. See Section 3.3.1 for a description of the tags provided.

3.2.5 OPERATOR MESSAGE DEVICE (OMD)

This function handles the output of all error messages and requests for operator intervention. The function is normally assigned to the console I/O typewriter and can be changed only by changing the device assignment in the I/O system. Non-terminal error messages are also output to the LOD to complete the "edit trail" when the CID is assigned to the card reader.

No handler is required in UTILITY as all output is queued to the I/O system and handled like any other I/O request.

3.3 COMMAND LANGUAGE

The command language forms the nucleus of the UTILITY program and to be successful must provide a functional and flexible interface between the user and the program.

3.3.1 GENERAL

All records input from the CID are considered to be UTILITY commands, either EXPLICIT or IMPLICIT.

(i) EXPLICIT

Explicit commands state specifically what function is required and are recognized by having an equals symbol (=) in character position one of the input record. A rigorous description of all explicit commands is given in Section 3.3.3.

(ii) IMPLICIT

Implicit commands do not state specifically the function to be performed but imply action to be taken. They are characterized by not having an equals symbol (=) in character position one.

Implicit commands are those source records from the CID which are to be written to the OUTPUT tape (UOD). They may require positioning of the INPUT and OUTPUT tapes, depending on whether or not a sequence number is given in positions 73 to 80 of the record.

SEQUENCE GIVEN implies two operations before the record is copied to the UOD:

- All records with sequence numbers less than the given sequence number are copied from the SID to the UOD.
- All subsequent records are skipped until a sequence number is read from the SID which is greater than the given sequence number.

By providing a sequence number equal to an existing sequence number on the SID, the record from the CID will replace the original. Otherwise the record from the CID will be inserted between the two records having sequence numbers lower and higher than that given.

SEQUENCE NOT GIVEN does not require positioning of the INPUT and OUTPUT tapes. The new source record is copied directly to the UOD at the current position.

All commands are listed on the LOD by default after execution by UTILITY. This listing may be suppressed by the user if desired. All commands are listed with an identification tag in character positions one through eight according to the following table:

TYPE OF COMMAND	IDENTIFIER	
explicit	COMAND	
implicit, replacing an existing record	REPLAC	
implicit, not replacing an existing record	INSERT	
implicit, no sequence given		i.e. blank

All listings of implicit commands also contain the sequence number assigned to this record on the OUTPUT tape, giving the facility to correct faulty insertions of source records without re-listing the updated program.

The listing of all commands as they are executed provides a complete "edit trail" which is useful for bench checking and for detecting reasons for incorrect updates.

The UTILITY control program is flow charted in figure 3.3.

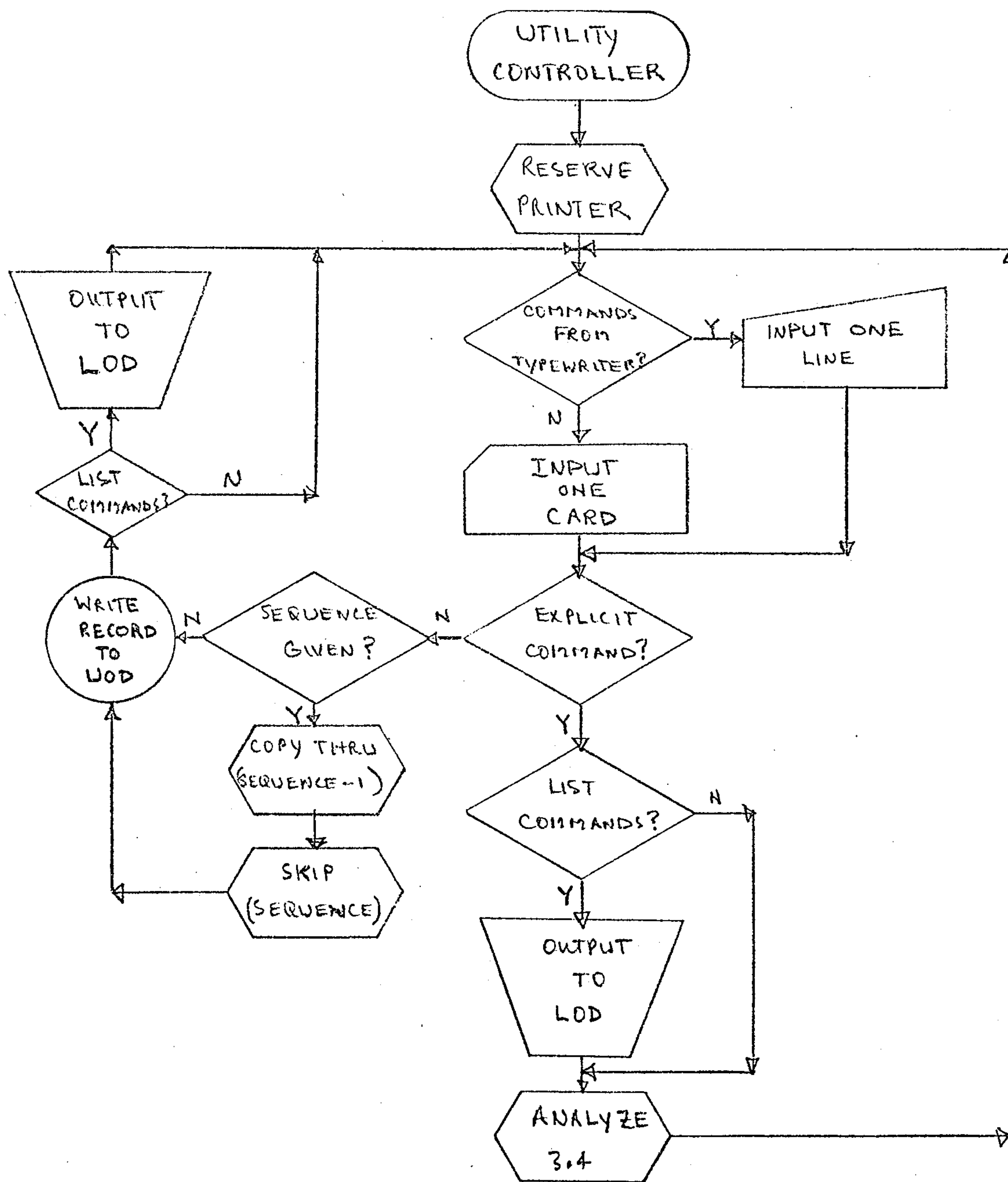


Figure 3.3 Utility Controller

3.3.2 ANALYZER

A command analyzer is provided to decode all "explicit" commands according to their syntactic definitions and transfer control to the required program module when the command has been analyzed.

The prerequisites for the analyzer are:

(i) It must allow complete flexibility in structuring of individual commands.

(ii) It must permit the syntactic definitions of statements in the language to be changed without requiring extensive reprogramming.

Error analysis and fast turnaround are not major considerations since UTILITY is a background job and is intended to be used interactively from the I/O typewriter.

A version of the analyzer developed by Cheatham and Sattley [1] is used by UTILITY to analyze commands. (See Figure 3.4.).

The analyzer may be defined as the algorithm which performs the recognition of allowable input strings in the language by using an encodement of the syntax specification as data. The syntactic specification of a language is a concise and compact representation of the structure of the language, i.e. a set of "grammar rules" in table form for forming allowable statements in the language.

Notation:

RECOG: Recognizer which returns "Success" or "Failure".

GENER: Generation routine - null if not a "Generating" structure.

SOURCE: Syntactic type being analyzed.

GOAL: First component of syntactic type.

CHAR: Character position in the input string.

GEN: Generated parameter list pointer.

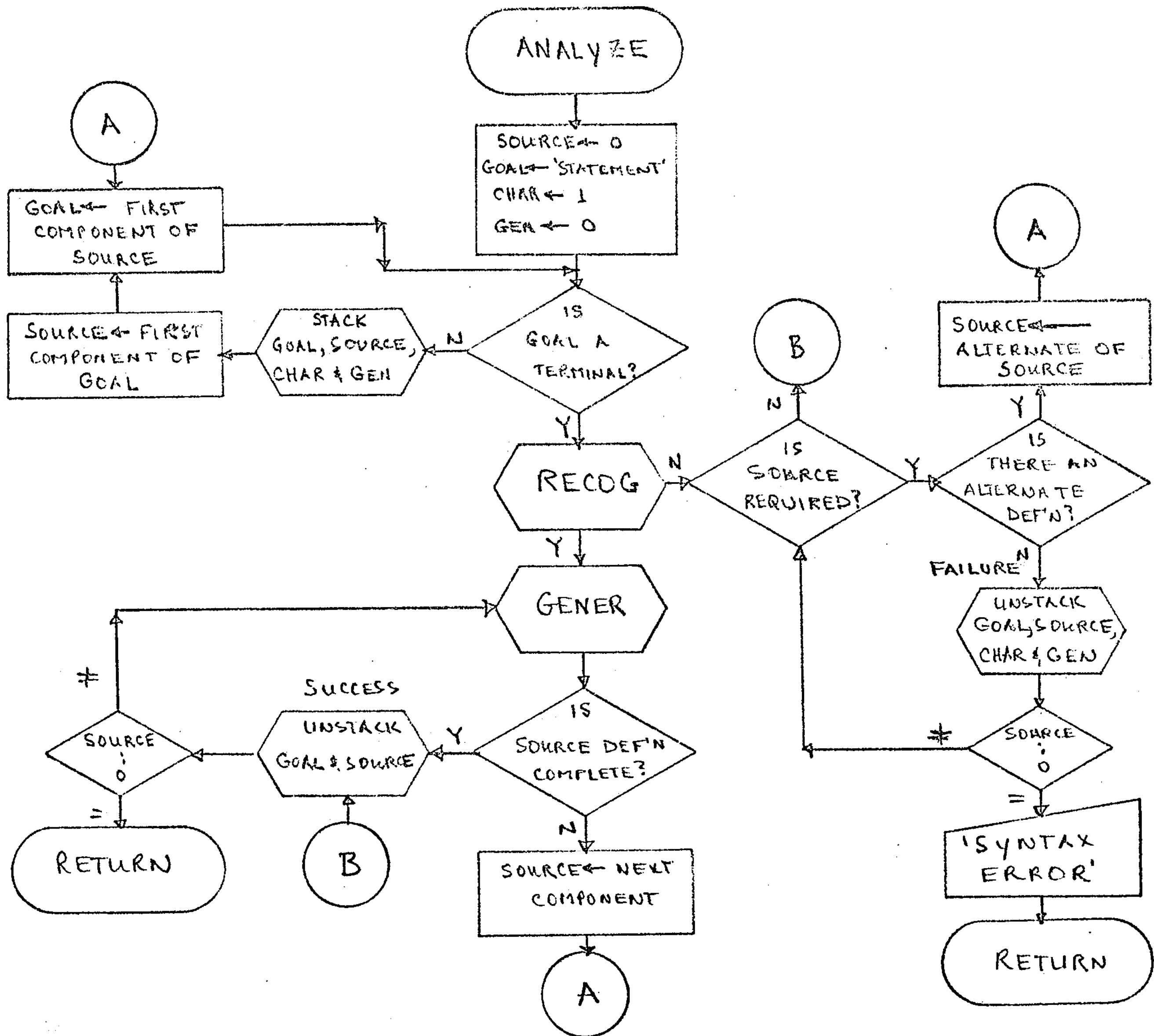


Figure 3.4 Analyzer

The function of the analyzer is to take each source statement and, by reference to the encoded syntax specification tables, to "predict" how the statement is constructed. The construction is verified by exhausting as many possibilities as necessary to find an allowable construction. Exhausting all possibilities indicates that the statement is not made up of allowable strings in the language and is syntactically invalid. A history is kept of the allowable strings which formed the statement, to be used after transferring control to the required program module.

This "history" takes the form of object code generation in a conventional compiler application but in this case consists of a "parameter list" of components of the statement. Each command in the language is an independent entity, the largest construction being a "statement" rather than a "program". Once the statement has been analyzed and the parameter list generated, the required modules of UTILITY are executed to perform the requested function.

There are two classes of strings or <u>syntactic types</u> in the structural definition of the language:

(i) <u>terminal types</u> - By definition, terminal types are the single characters of the source alphabet since it is from these basic components that all valid strings are generated.

(ii) <u>defined types</u> - Defined types are defined in terms of other defined types or terminal types. Two conditions must be satisfied for the syntax specification to be correct:

- Any defined type occurring as a component of any definition must be defined in some other definition (completeness).

- Every defined type must ultimately be constructed entirely out of terminal characters (connectedness).

The ANALYZER uses a routine called the RECOGNIZER to detect terminal types in the input string. Rather than recognizing only terminal characters, the RECOGNIZER is designed to detect the larger set consisting of those defined types which are defined strictly in terms of terminal characters. When the ANALYZER determines from the syntactic definitions that one of these "terminal types" should exist at this point in the input string, it requests the RECOGNIZER to determine its presence or absence.

The RECOGNIZER is capable of recognizing five different "terminal types":

(i) <u>A CARRIAGE RETURN</u> character which indicates the end of the statement.

(ii) A SLASH (/) character which indicates the start of the comment field and therefore the end of the statement to be analyzed (see Section 3.3.3).

(iii) A COMMA (,) character which is used as a delimiter in certain statements.

(iv) A NUMBER which is defined to be any string of numeric characters (0 - 9) of total length not greater than eight characters.

(v) A NAME which is defined to be any string of alphabetic characters (A - Z) of total length not greater than eight characters. The RECOGNIZER is asked by the ANALYZER if a "specific" name exists; the RECOGNIZER checks the name recognized against a list of keywords before returning success or failure.

The ANALYZER uses a "pushdown stack" to facilitate its "goal directed" parse of the statement. Each non-terminal or "defined type" is stacked until the ANALYZER either satisfies that definition or the definition fails. The stack forms a tree structure of those definitions still being examined. Successful analysis of a statement occurs only if the complete statement has been analyzed and the stack is empty.

3.3.3 SYNTAX DESCRIPTION

NOTATION

To illustrate and describe the formats of "explicit" commands, the following notational conventions will be used:

(i) Parenthesis (rounded brackets) will be used to enclose parameters which are optional, i.e. they may be left out if not required.

(ii) Brackets (squared brackets) will be used to enclose a set of parameters of which one must be chosen.

(iii) A and B will be used to represent sequence numbers.

(iv) N will be used to represent a repetition counter.

(v) All other words used are keywords and appear in context as they would in an actual statement.

TERMINOLOGY

The following words, with their associated definitions, appear as keywords in the command language:

(i) INPUT refers to the SID (tape 1).

(ii) OUTPUT refers to the UOD (tape 2).

(iii) RECORD refers to an 80 character (20 word) BCD card image.

(iv) BLOCK refers to one or more RECORDS on magnetic tape followed by an inter-record gap.

(v) FILE refers to one or more BLOCKS on magnetic tape followed by an end-file (EOF) mark.

(vi) SECTION refers to one or more FILES on magnetic tape followed by an end-file (EOF) mark.

All explicit commands must begin with an = in column one, but otherwise are "free form". Each command may be commented if desired by following the command with a slash (/), followed by any string of characters the user wishes. Using the notation defined, the general form of an explicit command is:

= COMMAND (/ COMMENT)

There are 14 "explicit" UTILITY commands available to the user. The syntax and function of each will be described in Sections 3.3.3.1 through 3.3.3.14.

3.3.3.1 [INPUT / OUTPUT] BLOCKED BY N

This command is used to set the blocking factor for the INPUT and OUTPUT tapes where "N" represents the number of records per block. Default blocking factor is one record per block, i.e. unblocked.

A "block length error" will be output if the INPUT blocking factor specified is less than the actual INPUT blocking factor. The maximum blocking factor is determined by the time it takes to read/write the block and is limited to twenty-five records per block by the buffer size.

3.3.3.2 COMMAND (S) FROM [TW | CARD (S)]

This command is used to assign the CID to the typewriter (TW) or card reader (CARDS). Default input is from the card reader.

3.3.3.3 COPY (AND LIST) [A | A THRU B | THRU B | N [RECORD (S) | FILE (S) | SECTION (S)]]

This command copies, and optionally lists, the specified records from the SID to the UOD. If a starting sequence number is not given, copying begins with the next record in the buffer. If a starting sequence number is given, the routine skips all records preceding that record.

The routine flow charted in figure 3.5.1 is used to copy/skip list all records from the designated tape.

3.3.3.4 END [FILE (S) / SECTION (S)]

This command will write a single end-file (END FILE) or double end-file (END SECTION) mark to the UOD.

3.3.3.5 (DO NOT) LIST COMMAND (S)

This command is used to enable or disable the listing of records from the CID. Default is to list commands.

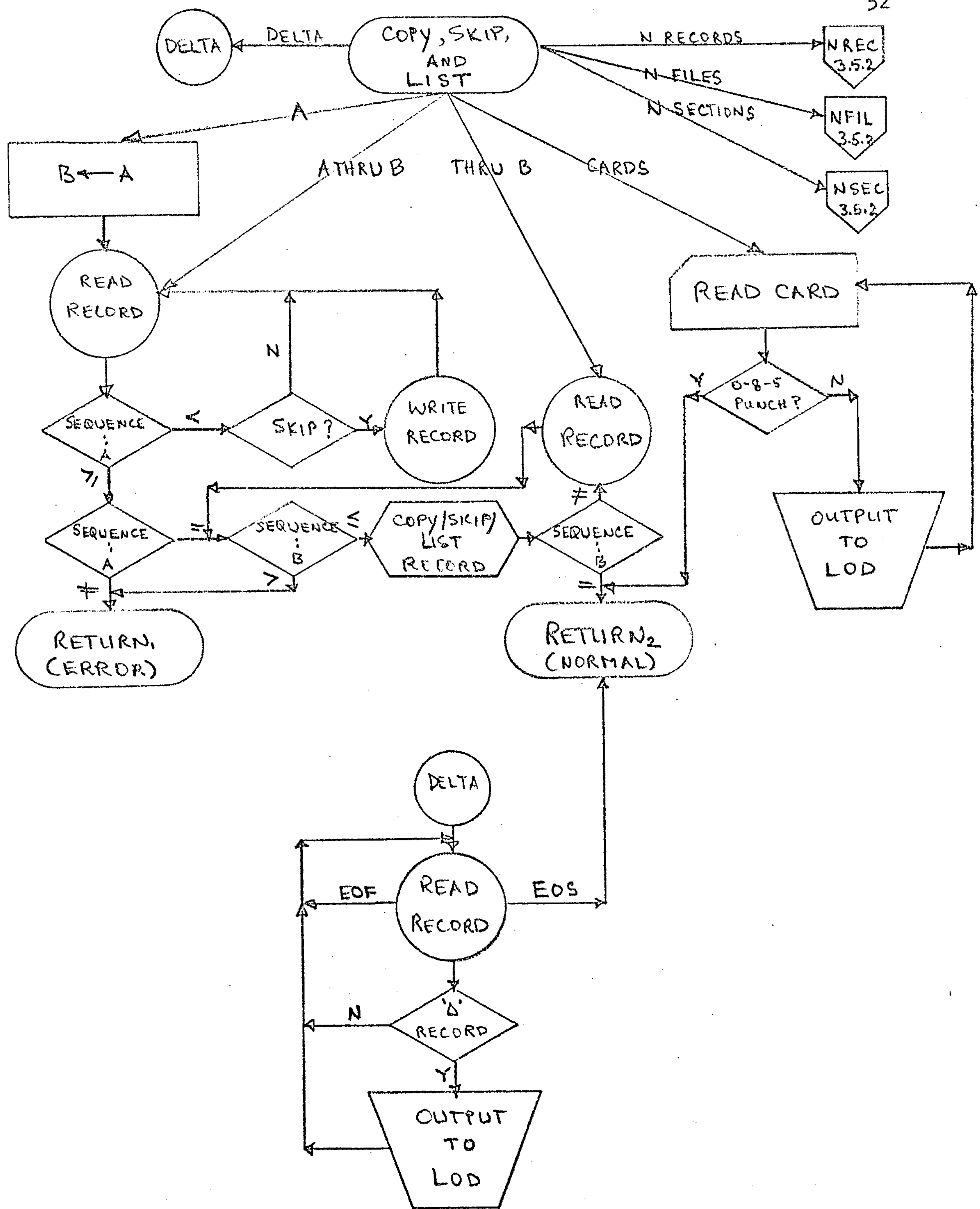


Figure 3.5.1 Copy, Skip, and List

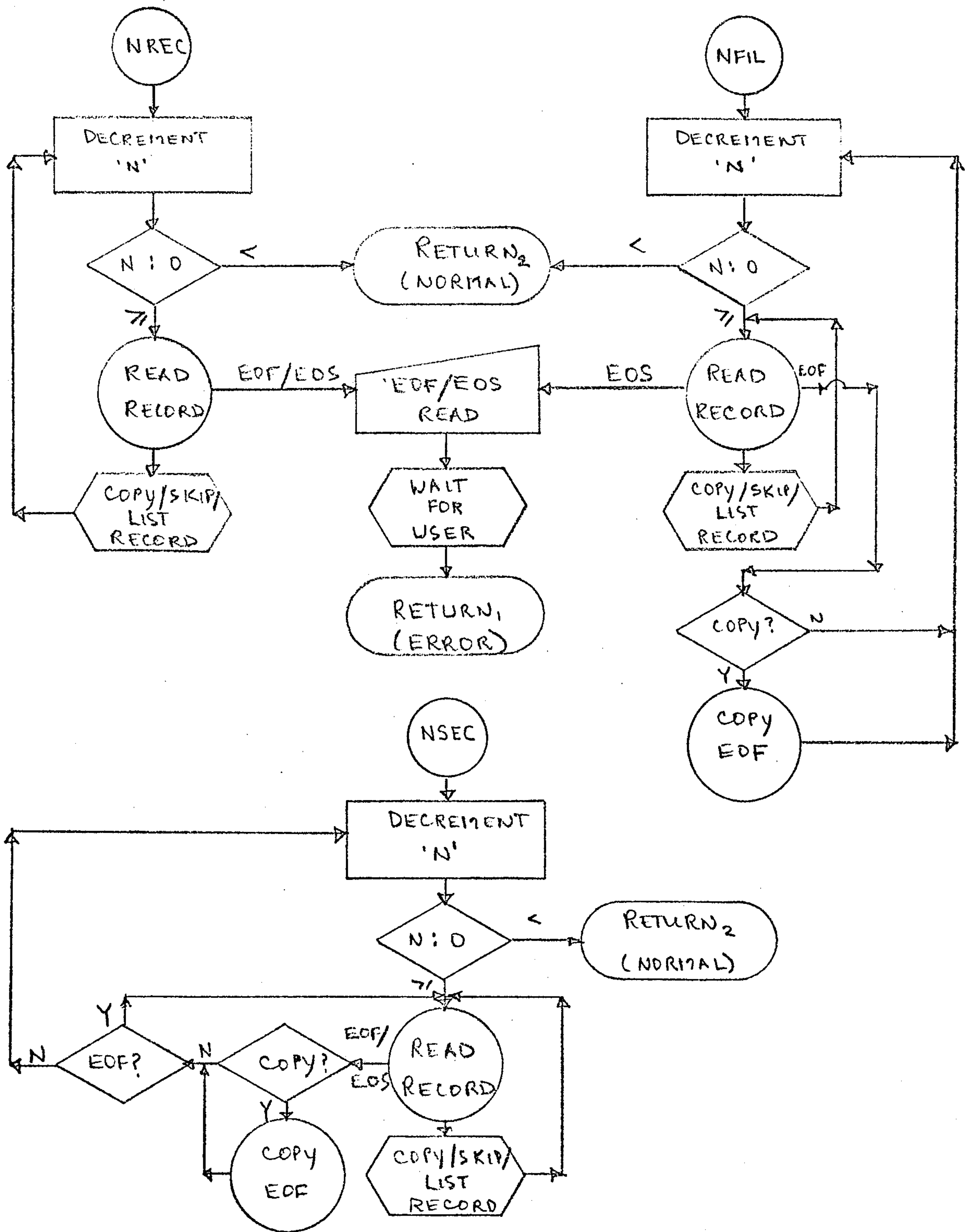
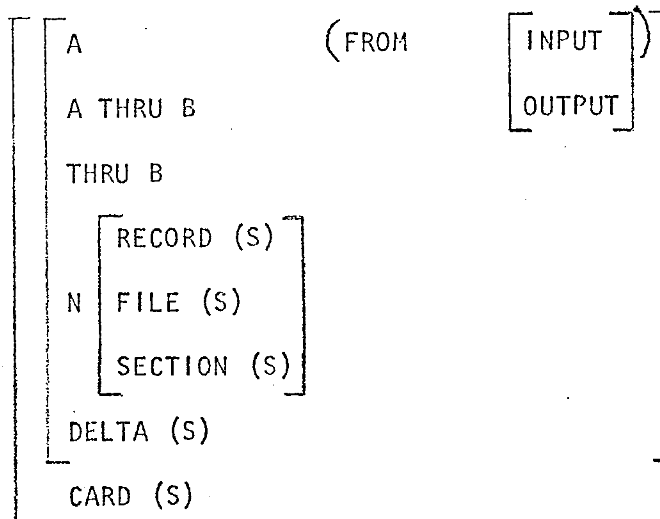


Figure 3.5.2

3.3.3.6 LIST



This command will list the specified records from the indicated input device. Default input for listing tapes is INPUT (SID). If a starting sequence number is not given, listing starts with the next record in the buffer. "List deltas" will list all records with a delta (△) in character position one. "Delta records" contain LOADER information and delimit coreloads on source tapes (see Section 4.3.4). "List cards" will list all cards from the card reader until a card is read with a 0 - 8 - 5 punch in column one.

The routine flow charted in figure 3.5.1 is used to copy/skip/list all records from the designated input tape.

3.3.3.7 PAGE

This command causes a page skip on the LOD.

3.3.3.8 PAUSE

This command causes UTILITY to pause and to wait for the user to input a command on the typewriter to restart it. The command, complete with comment, is output to the OMD and the user is requested to restart UTILITY or to release it. This command is normally used for changing SID tapes so that other active coreloads can use the time the operator takes to mount the tape.

3.3.3.9 REVERSE [INPUT / OUTPUT] N [BLOCK (S) / FILE (S) / SECTION (S)]

This command is used to reverse the specified tape the amount requested or to the load point marker, whichever occurs first. If the load point marker, unexpected end-file, or unexpected end-section are read, a message is output on the OMD and UTILITY proceeds.

The user must be aware that this command applies to movement of the tape and not to the current record addressed in the buffer area. Because of the buffering of input and output, the user must reverse at least one more buffer than actual sequence numbers would indicate.

```
3.3.3.10  REWIND          [INPUT ]
                          [OUTPUT]
```

This command will rewind tape one (INPUT) or tape 2 (OUTPUT).

```
3.3.3.11  SEQUENCE        [FROM         [A BY B]]
                          [             [INPUT ]]
                          [OFF                  ]
```

This command determines the method of sequencing of all records output to the UOD. From A BY B gives a starting sequence (A) and increment (B); FROM INPUT indicates that no resequencing should take place, i.e. as read from the SID; OFF will blank all sequence numbers. Default sequencing is FROM INPUT.

```
3.3.3.12  SKIP (AND LIST)    [A                    (FROM   [INPUT ])]
                             [A THRU B                     [OUTPUT]  ]
                             [THRU B                                 ]
                             [    [RECORD (S) ]                      ]
                             [N   [FILE (S)   ]                      ]
                             [    [SECTION (S)]                      ]
```

This command skips, and optionally lists, the specified records from the indicated tape. Default input for skipping tapes is INPUT (SID). If a starting sequence number is not given, skipping starts from the next record in the buffer. If a starting sequence number is given and skipping is FROM INPUT, the routine copies all preceding records from the SID to the UOD.

The routine flow charted in figure 3.5.1 is used to copy/skip/list all records from the designated input tape.

3.3.3.13 TAB (S) AT T_1, T_2, --- , T_N

This command is used to set internal tabs for reconstructing typewriter input of commands. T_1 to T_N are the character positions of the tabs. Default tabs are at columns 8, 16, 36, and 73 for SYMBOL. A maximum of ten tabs are permitted.

3.3.3.14 TX

This command is used to "sign off" or release UTILITY.

3.4 PROGRAM SIZE AND RUN-TIME STATISTICS

The present version of UTILITY consists of 2500_{10} source records and occupies 7760_8 words of core memory, allocated as follows:

Program and Constants	5510_8 words
Buffer Areas	2250_8 words
	7760_8 words (4K)

Run-time statistics are as follows:

(i) Read Magnetic Tape: 2300 - 2500 records/minute (unblocked)

(ii) Write Magnetic Tape: 2300 - 2500 records/minute (unblocked)

(iii) Line Printer Listing: 125 - 150 records/minute

(iv) Reading Cards : 150 - 175 records/minute

CHAPTER 4

SYMBOL

4.1 ABSTRACT

The SYMBOL program provides an on-line means of translating source programs written in XDS assembler language SYMBOL into machine code for the XDS 920.

The basic assembler is the off-line SYMBOL assembler supplied by XDS, extended and modified to meet the requirements and operating environment of Alberta Gas Trunk Line's Supervisory System.

SYMBOL will be described on a macro scale only; further information concerning the syntax and semantics of the XDS SYMBOL language can be found in the XDS reference manuals [4], [5], [6], and [7].

4.2 GENERAL DESCRIPTION

The SYMBOL assembler performs the following functions:

(i) It reads source statements from magnetic tape unit one (unblocked).

(ii) It translates these source statements from XDS SYMBOL language into machine code for the XDS 920.

(iii) It outputs the machine code to magnetic tape unit two in XDS standard binary language.

(iv) It produces a listing of the source and object code on the line printer for future reference.

SYMBOL performs these functions by defining and referencing four tables as it analyzes each statement:

(i) MNEMONIC TABLE

The mnemonic table defines all operation code symbols recognized by the assembler. Besides the standard 920 instruction list, the table is expanded during assembly to contain the user-defined operation codes (from OPD and FORM statements) and all undefined OP codes which are considered to be externally resolvable programmed operators (POPS).

Each entry in the table contains the following information:

- one to six characters of the symbol,
- type of operand field to follow,
- whether standard op-code or user defined,
- how to decode and use the operand field if the op-code is user-defined or a SYMBOL "directive",
- whether the op-code is "local" or "global" (see Section 4.3.3),
- the op-code to be used in the assembled instruction.

(ii) LABEL TABLE

The label table defines all labels which may be referenced by the program being assembled. The table is developed during assembly with the addition of each label recognized as statements are processed.

Each entry in the table contains the following information:

- one to six characters of the label,
- whether its address is relocatable or absolute,
- whether it is "local" or "global",
- whether it is "external" or not,
- the value (address) of the label.

(iii) LITERAL TABLE

The literal table defines all constants in the program which are referenced by value rather than by name. Literals are assigned sequentically as they are recognized to a "literal pool" which starts at the next relative location past the last word used by the program instructions.

Each entry in the table contains the following information:

- value of the literal,
- whether the value is relocatable or absolute,
- the relative location the literal will occupy in the object program.

(iv) REFERENCE TABLE

The reference table defines those symbolic references within the program which are not found in the label table and which are assumed to exist in some context external to the program.

Each entry in the table contains the following information:

- one to six characters of the symbol,
- the relative location of the last data word to reference this symbol.

To allow the loader to resolve these "external references", all data words referencing this symbol are "chained" with the address portion of the last reference location containing the address of the second last reference and so on until the first reference location where the address portion is zero.

SYMBOL is a two pass assembler, back spacing to the start of the program and re-reading the source statements for pass two.

PASS 1

(i) Source lines are read and a location counter is maintained for defining labels.

(ii) Label fields are analyzed and the labels are entered into the LABEL table.

(iii) Operation codes are analyzed and if not defined they are entered into the MNEMONIC table as POP's.

(iv) Only operand fields of "directives" are analyzed since they may cause the location counter to be incremented as a function of their operand fields.

At the end of pass one all external labels and POPS are output to the binary output device (magnetic tape two) to provide loader information for building global label and mnemonic tables. All entries to the MNEMONIC and LABEL tables must be made during pass one. The program reverses the symbolic input tape to the first record of the program in preparation for pass two.

PASS 2 (i) Source lines are completely processed and both object code and a listing are generated.

(ii) References to symbols not in the LABEL table are entered into the REFERENCE table.

(iii) Literals are processed and entries are made in the LITERAL table.

At the end of pass two all literals and references are output and listed. Once the table areas have been purged of all local definitions (see Section 4.3.3.), SYMBOL is readied to accept the next program.

SYMBOL will continue assembling programs until an end-section (double EOF) is read. The operator must then request SYMBOL to continue or to sign off. The SYMBOL controller is flow charted in figure 4.1.

SYMBOL provides error detection by the appearance of up to four single character diagnostic flags in columns one - four of the output listing.

The following table describes the meaning of these flags.

FLAG	ERROR	ACTION
D	DUPLICATE	Duplicate definition or reference. All references take the value of the first definition.
E	EXPRESSION	Illegal expression in operand field. Operand interpretation terminated.
I	INSTRUCTION	Instruction mnemomic not defined. Treated as implicit POP reference.
L	LABEL	Illegal symbol in label field.
O	OVERFLOW	Symbol table overflow. The assembly continues but the definition is NOT made.

FLAG	ERROR	ACTION
P	PARENTHESIS	Too many parenthesis levels or unequal number of left and right parenthesis. Operand interpretation is terminated.
R	RELOCATION	Operand expression involves the illegal use of one or more relocatable items. The correct, but non-relocatable, value of the expression is determined and output.
T	TRUNCATION	Significant bits were lost due to left-hand truncation in inserting a value into a specific field. The value is truncated modulo 2^n where the field size is "n" bits.
U	UNDEFINED	A reference has been made to an undefined symbol in the address field. Zero is substituted for the undefined value.
*	EXTERNAL	An external address reference has been made (which may or may not be in error).

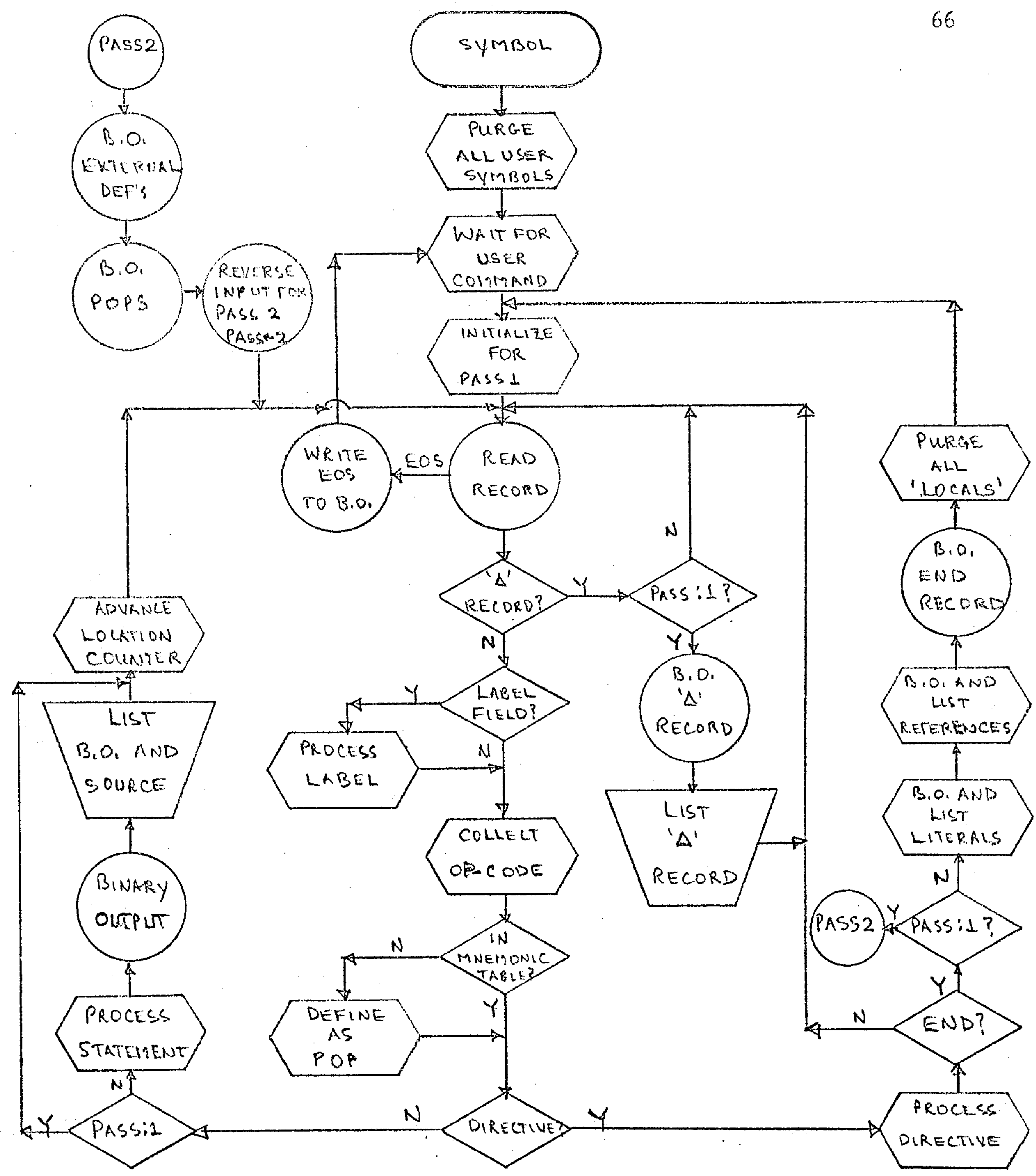


Figure 4.1 Symbol Controller

SYMBOL makes no attempt to indicate the position in the source line where the error occurred. Despite the simplicity of the flag fields (single character), they are sufficiently exhaustive to make most errors immediately obvious. There are few enough so that the user quickly becomes familiar with their interpretation and rarely has need for a cross referencing table.

The INSTRUCTION flag (I) and the EXTERNAL flag (*) may or may not indicate an actual error. To provide symbolic inter-program communication, SYMBOL treats all references to undefined op-code mnemonics and undefined labels as though they will be defined externally by other programs at load time.

4.3 EXTENSIONS

Although SYMBOL is basically the stand alone off-line version of the assembler supplied by XDS, several modifications and extensions were necessary to satisfy the requirements of Alberta Gas Trunk Line.

Interfacing with the Supervisory System required such changes as:

(i) Development of a resident I/O package to handle source input from magnetic tape unit one, binary output on magnetic tape unit two, listing on the line printer, and operator messages to the console typewriter. The package was developed to satisfy specific user and operating system

requirements and is similar to the package described in Section 2.1.2.

(ii) Ensuring that the system executive could not alter hardware indicators (overflow), via the interrupt system, which are used and tested in SYMBOL to detect truncation and overflow errors.

SYMBOL will assemble all valid statements in the XDS assembler language SYMBOL as well as specific extension to be discussed in sections 4.3.3 and 4.3.4.

4.3.1 OVERLAYS

It was possible to allocate a maximum of 4K 24 bit words of core memory to the priority four partition without jeopardizing the operation of foreground programs. Because the amount of RAD memory available is without practical limitation, (almost 500K words), RAD was substituted for core by using overlaying.

An overlay handler was built which utilizes an overlay area of five sectors (320 words), overlaying this area with new program segments as they are required. The table size represents an area large enough to perform the required functions without excessive overlay overhead and not so large as to limit the amount of core available to the rest of SYMBOL.

The overlay handler is flow charted in figure 4.2. The handler contains the following features which increase its flexibility and speed:

(i) The overlay handler maintains a "pushdown stack" of return addresses to provide restoration of the last overlay segment when the current segment is released. This technique permits one overlay to call another but is not yet used in this implementation.

(ii) To decrease overhead, overlay segments are not "saved" when one overlay calls another i.e. overlays are "read only". Data storage to be saved must be defined as part of the non-overlaid program area.

(iii) Overlays may have multiple entry points, permitting several logically related functions to be grouped into a single overlay. The handler recognizes that subsequent calls are to the same overlay segment and will not perform RAD transfer for each call.

Parts of SYMBOL have been overlaid according to the following table:

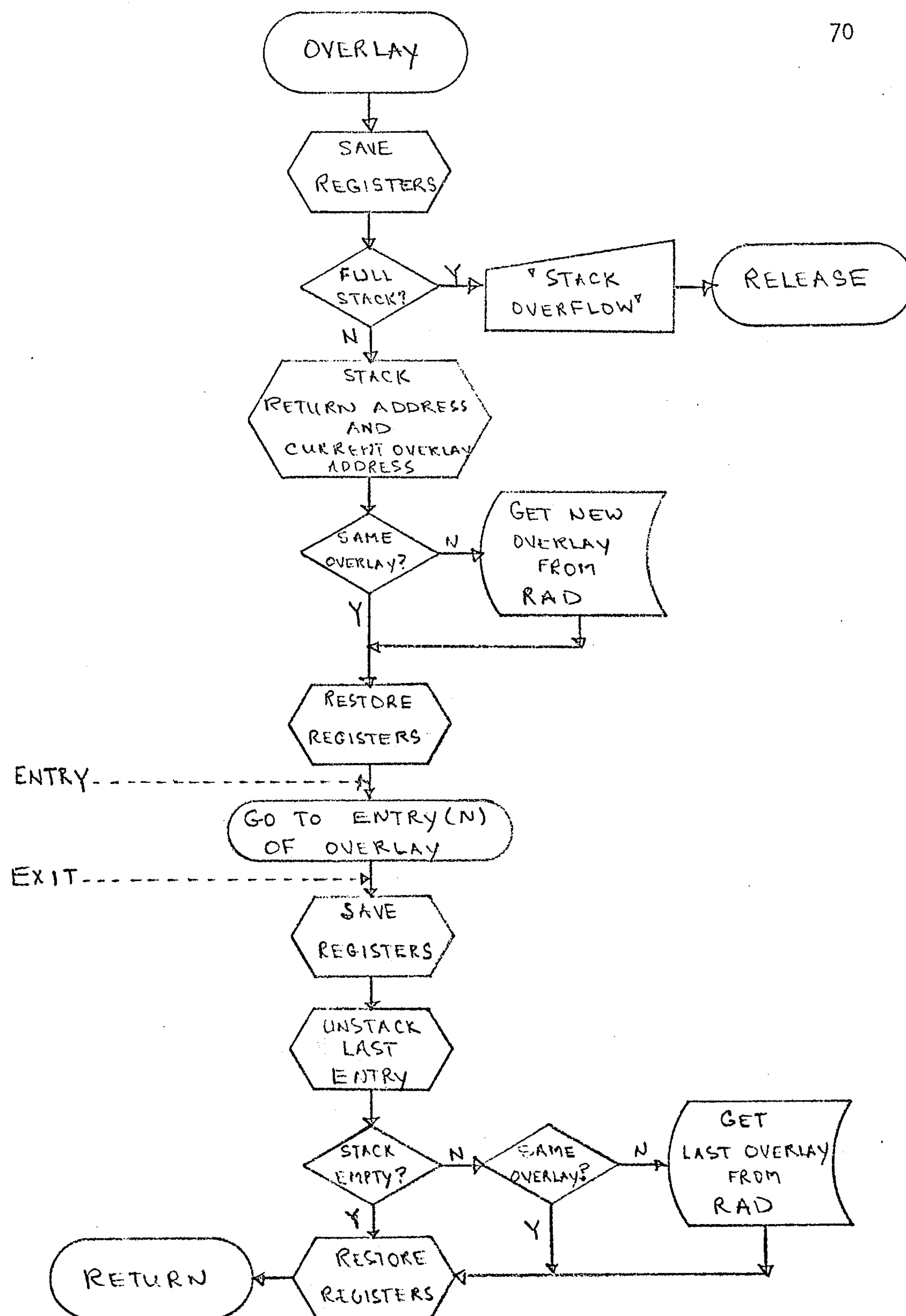


Figure 4.2 Overlay Handler

OVERLAY	FUNCTIONS PERFORMED
1	User command decoder and analyzer - (see Section 4.3.6.).
2	Directives DED and DEC.
3	Directive Copy.
4	End of pass one - outputs external definitions, freezes MNEMONIC table, outputs POPS, and initializes for pass two.
5	End of pass two - outputs literals and external references, outputs END and outputs program statistics.
6	Purges MNEMONIC, LABEL, REFERENCE and LITERAL tables at end of pass two and on initial loadings, and initializes for pass one.
7	Directives EQU, OPD, FORM, AORG, ORG, BSS, DATA, BCD, TEXT, and FORM references.
8	Outputs job statistics and signs off.

Overlay seven is a good example of point (iii); minimizing RAD transfers by grouping related logic modules. Those directives analyzed by overlay seven are the most common; once the overlay is input no more RAD transfers are necessary until a different overlay is required e.g. end of pass.

4.3.2. SYMBOL TABLES

To provide enough symbol table storage space to assemble Alberta Gas Trunk Line's Supervisory System requires that some tables be RAD resident rather than entirely core resident as in the XDS version.

The LABEL, REFERENCE, and LITERAL tables are RAD resident and the number of entries they are able to hold is a function of the amount of RAD allocated to each table. The MNEMONIC table is left core resident in order to minimize assembly times. Reference to figure 4.1 will demonstrate that the MNEMONIC table is searched for all op-codes during both pass one and pass two. Since searching the MNEMONIC table constitutes a large proportion of the overhead for pass one, significant efficiency is gained by having the table core resident.

All entries in the MNEMONIC table are sorted according to their symbolic identifiers in order that a binary search technique can be employed to make new entries or to find a given entry.

Each RAD resident symbol table is constructed of (n + 1) segments of three sectors (192 words) each, where n is a prime number. The search algorithm is flow charted in figure 4.3. The first four characters of the symbol are arithmetically hashed by dividing by n, giving a remainder to be used as a starting segment number for the symbol search. Each segment is searched linearly until the given symbol or a spare entry is found. The search will continue from segment to segment until the last (n + 1) segment is searched. The last (n + 1) segment is an overflow segment since division by n can give a maximum remainder of n-1 (segment n). The number of "collisions" (symbols hashing to the same segment) is kept to a minimum by keeping n as large a prime as possible. RAD transferring is minimized by sacrifing RAD storage for a "random" hashing algorithm and therefore a lower table packing density.

All references to the RAD tables are made through common "search" and "move" routines and all use the same buffer area in core. The handler which sets up the RAD transfers to input and output segments minimize overhead by saving "current" table segments <u>only</u> if it is indicated that the segment has been modified since it was input to "transient core". Those segments of SYMBOL which change the contents of the symbol tables are charged with the responsibility of indicating to the handler that the RAD segment is modified and therefore must be saved.

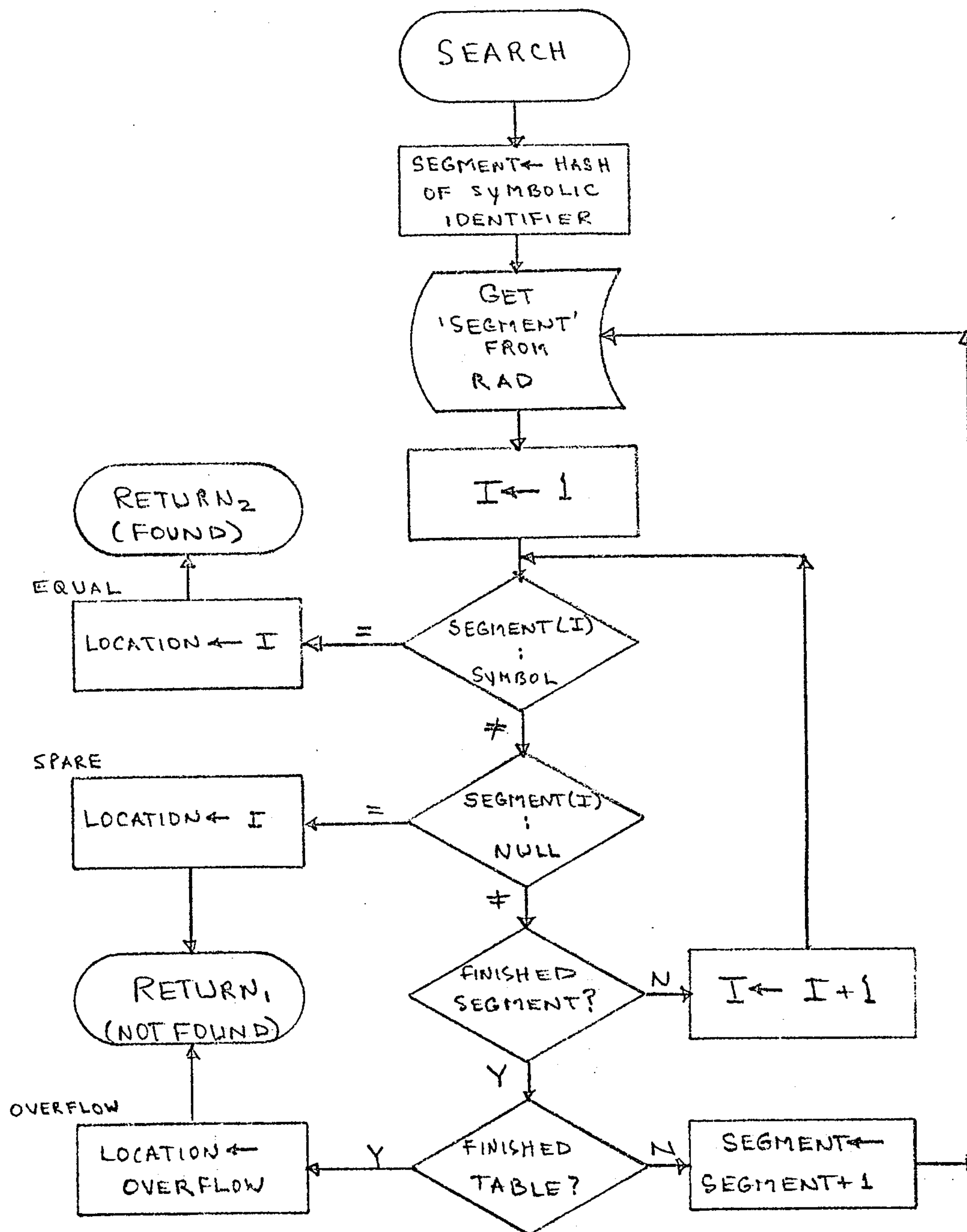


Figure 4.3 Search RAD Resident Symbol Tables

4.3.3. GLOBAL DEFINITIONS

Global definitions are defined within the context of this system to be those labels and mnemonics which, once entered into their respective tables, remain until SYMBOL is released. Reference to figure 4.1 will illustrate that all user defined symbols are purged only on initial load; for the remainder of the assembly only the local symbols are purged at the end of each program.

This facility to declare global op-codes and labels provides users with a powerful method of inter-program communication. The data base has been constructed using a two dimensional array concept with the first co-ordinate being the table name and the second co-ordinate the relative table location. Global labels allow users to define elements of a structure as global and to refer to the relative table locations symbolically in subsequent programs. Considerable memory is saved in the applications programs packing several relative table address into single computer words using FORM definitions - the addresses of which cannot be resolved using the "external reference" scheme of inter-program communication.

Global labels are defined by prefixing the label field with a slash (/). The user may define a label to be global with the definition or on a line subsequent to the definition of the label. A set of labels may be defined to be global by listing them following the first symbol and separated by commas. A single slash in the first character position of the line defines all labels in the line to be global.

Operation codes are also defined to be global by prefixing the label field with a slash. The symbol can be declared global only in the definition line-multiple or delayed definitions are not possible.

All global entries in the symbol tables are tagged to indicate that they are not to be purged at the end of each program. The addresses of global labels are relocated to their absolute execution addresses and the label is made absolute so that the loader will not incorrectly relocate references to these addresses. All subsequent references to a global label will contain an absolute execution address rather than a relative table address.

Purging of symbol tables requires a purge routine to remove all locally defined symbols and shift the table to remove all gaps. Purging of the MNEMONIC table, which is core resident, is accomplished by deleting all local user defined symbols and recalculating the table boundary parameters used by the binary search algorithm. Because of the hashing algorithm employed for RAD tables (section 4.3.2), it is necessary to re-hash all global symbols to ensure that they are left at the lowest possible address in the lowest possible segment. The purge routine for RAD resident tables is flow charted in figure 4.4.

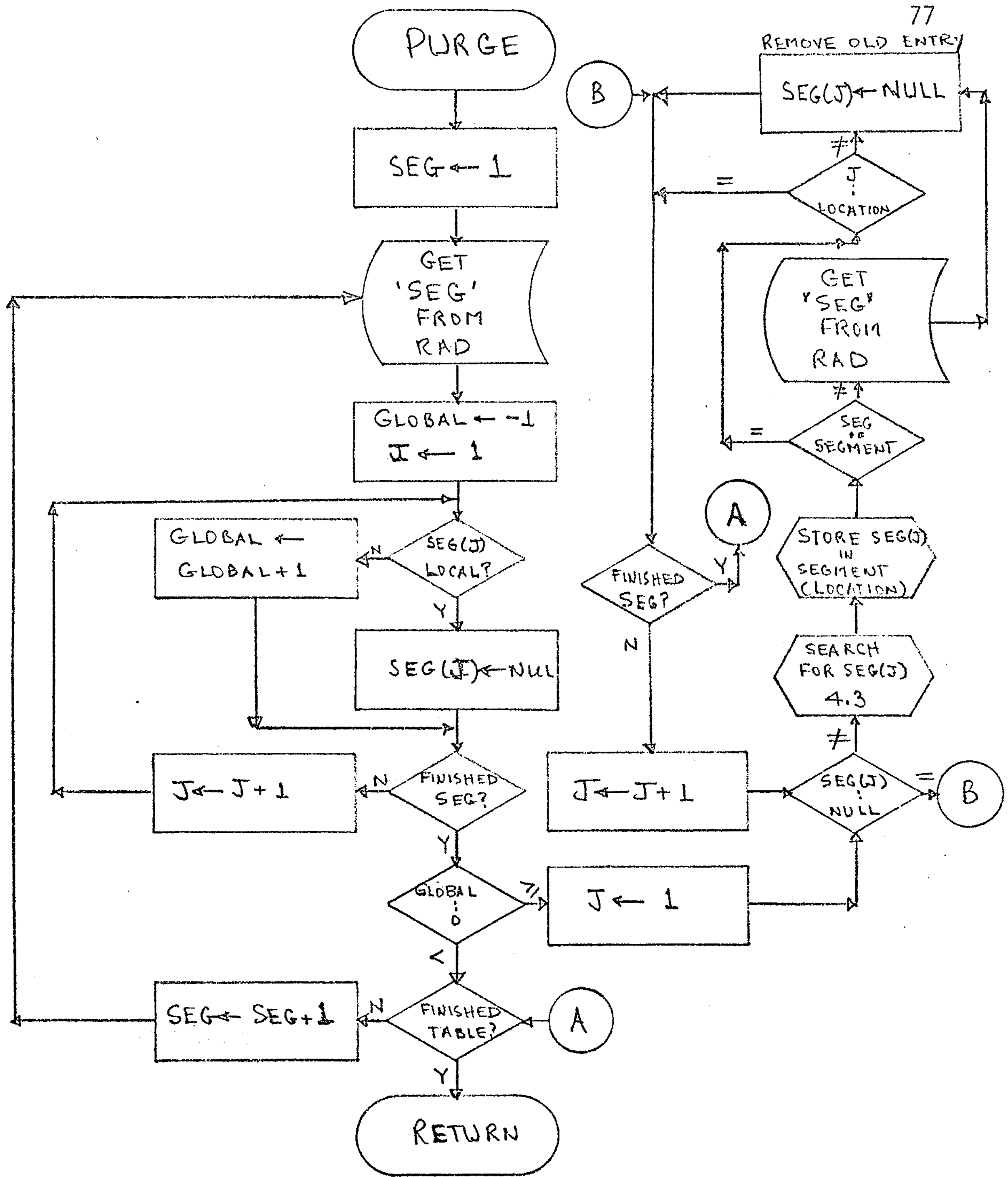


Figure 4.4   Purge RAD Resident Symbol Tables

4.3.4. LOADER INFORMATION

All source records containing a delta (Δ) in character position one are considered to contain information for the loader and are listed and output during pass two.

Each "delta record" must have the following format:

Δn ←--- 1 to 7 blanks ---→ NAME ←-- 1 to 7 blanks --→ BIAS

where,

n is the coreload priority,

NAME is the 1 to 6 character coreload name,

BIAS is an optional octal execution bias (for overlays).

Each coreload must begin with a "delta record" in order to provide the loader with a base address for execution relocation. A coreload may consist of several programs (assembled relative to location 0) but can have only one "delta record" preceding each coreload.

4.3.5. PROGRAM STATISTICS

The following statistics are maintained for each program assembled and are output following the program listing:

(i) total assembly time (from real-time clock),

(ii) program size (maximum location used),

(iii) number of LABELS, LITERALS, and REFERENCES in the symbol tables.

Each line of statistics is headed by an asterisk (*) so that it will be output when the user requests the "list errors" option (see Section 4.3.6).

4.3.6 ASSEMBLY OPTIONS

SYMBOL provides the facility to rewind tapes and to specify the mode of output from the assembler from the console typewriter.

Using the notation described in Section 3.3.3, the following commands are available through the I/O typewriter to control the assembly process.

4.3.6.1 REWIND $\begin{bmatrix} \text{SI} \\ \text{BO} \end{bmatrix}$

This command will rewind tape one (SI) or tape two (BO).

4.3.6.2 LIST ERRORS

This command instructs SYMBOL to list only tagged statements (Section 4.2). This is the default listing mode of SYMBOL.

4.3.6.3 LIST ALL

This command instructs SYMBOL to list all statements.

4.3.6.4 NO LIST

This command instructs SYMBOL to suppress all list output.

4.3.6.5 NO BO

This command instructs SYMBOL to suppress all binary output. This is the default binary output mode of SYMBOL.

4.3.6.6 BO

This command instructs SYMBOL to produce binary output of object code.

4.3.6.7 GO

This command instructs SYMBOL to continue with the next operation.

4.3.6.8 TX

This command is used to "sign off" or release the SYMBOL program.

4.3.6.9 "DELTA RECORDS"

Any record with a delta (**Δ**) in character position one is considered to be loader information and is output directly to the binary output tape.

Each command may be input individually or several may be combined in the same command, separated by commas. Each command is analyzed and executed interpretively and therefore transfer commands, (GO or TX), must occur last in a sequence. The analyzer will request new commands until a transfer command is sensed. Blanks are ignored except as delimiters, allowing "free form" input.

Error analysis consists of the message "SYNTAX ERROR" output when analysis fails. Because commands are analyzed and executed interpretively, all commands preceding the error will have been executed.

The user is able to input commands and change assembly mode (see figure 4.1) in only two instances:

(i) On initial loading.

(ii) When an end-section (double EOF) has been sensed, indicating the end of this "batch" of assemblies. The user can request SYMBOL to continue (GO) if there is more input to assemble; he can specify new output modes; or he can sign off (TX).

End-sections are used to delimit assemblies where the options are constant. Segmentation is necessary when designing global table descriptions (no binary output and no listing), and when the source must be input from several tapes.

The ability to specify the output of the assembler allows users to perform "test assemblies", listing only error lines until all syntactic "bugs" are eliminated. Test assemblies which use the default modes of output (list errors and no binary output) have become a standard phase of program development.

4.4 STANDARD BINARY LANGUAGE

XDS has specified a "standard binary language" for the 9 - series computers with the intention that the language be both computer and medium independent. The subset output by the assembler will be described briefly.

The first word of each record is a control word which provides sufficient information for the LOADER to handle the record. The control word specifies:

(i) The type of record.

(ii) The number of words in the record (word count).

(iii) A check-sum for detecting longitudinal parity errors.

The following types of records may be output by SYMBOL:

(i) Data records which contain:

- The relative load address of the block of data.
- The block of instructions or constants to be loaded, i.e. the program.
- The flags indicating whether load relocation and/or POP relocation should be applied to the data words.

(ii) External References and Definitions which contain:

- The 1 to 6 characters of the symbol.
- If reference, the address of the last reference to the symbol.
- If definition, the address (value) of the symbol.

(iii) POP References and Definitions which contain:

- The 1 to 6 characters of the symbol.
- The temporary sequence number used for the POP for this program.
- If definition, the origin of the POP routine.

(iv) End records which contain:

- The last word address used by the program, i.e. the length of the program.
- The transfer address if used (ignored by the LOADER in this application).

4.5 PROGRAM SIZE AND RUN-TIME STATISTICS

The present version of SYMBOL consists of 4700_{10} source records and requires 37500_8 words of memory, allocated as follows:

Program and constants		6400_8 words
Buffer areas		700_8 words
Overlay area		500_8 words
	Core	10000_8 words (4K)
Overlays		3300_8 words
Tables		24200_8 words
	RAD	27500_8 words (12K)
	TOTAL MEMORY	37500_8 words (16K)

The core resident MNEMONIC table can be extended to hold 50 user-defined operation codes (FORMS, OPD'S and POPS) before overflowing.

The RAD resident LABEL table can hold a maximum of 2400 labels if hashing is completely random. Because the search algorithm is "circular" (figure 4.3), packing densities in excess of 75% (1800 labels) are to be expected before overflow. As the number of labels approaches the 75% density, the searching of the table will become much less efficient because of the RAD transferring required.

The efficiency can be improved by increasing the number of RAD segments allocated to the table. The RAD resident REFERENCE and LITERAL tables can each hold a maximum of 512 entries. Additional capacity can be achieved in the same manner as the LABEL table if required.

The average run-time statistics are as follows:

Complete Assembly (listing and binary output):

95 - 105 statements/minute.

Test Assembly (no listing and no binary output):

450 - 500 statements/minute.

Purging of Symbol tables at the end of each program requires 10 - 15 seconds and will decrease the number of statements/minute by that amount. Overhead for purging becomes more noticeable and makes SYMBOL proportionately less efficient when the batch includes many short programs.

CHAPTER 5

LOADER

5.1 ABSTRACT

The LOADER program provides an on-line means of loading XDS 920 object programs in standard binary language format, grouping the programs into "coreload" modules on the RAD, and integrating them into Alberta Gas Trunk Line's Supervisory System.

5.2 OBJECTIVES

Although the primary requirement is to load the object code produced by SYMBOL, several features have been stressed to increase the user control and utility of LOADER.

5.2.1 DIAGNOSTIC AND LOADING INFORMATION

The LOADER program provides detailed diagnostic output to the user during the load in order that he may easily diagnose and correct errors caused by illegal object code formats (Δ records), duplicate definitions, and unresolved references. Diagnostic messages are designed and formatted to provide the maximum available information regarding the exact cause and location of the error: coreload name, relative location, and reference to the specific cause of the error are given in all cases. Except for catastrophic errors, such as persistent tape read error, the LOADER will continue with the remainder of the load, forcing the user to diagnose the reason for the error before integrating the program (s) into the system.

The LOADER provides detailed output of all global definitions as a LOAD ADDRESS MAP. All definitions are sorted and output both alphabetically and by address to provide the user a visual means of determining the execution address of object code when the system is loaded and in operation.

5.2.2. COMPLETE OR PARTIAL LOAD

System maintenance and updating procedures require that the entire system be updated and reloaded regularly to maintain an integrated system. Modifications must be implemented between updates which do not justify a complete reload. These applications programs invariably refer symbolically to the system data base, requiring addresses from the definition table prepared at load time.

The LOADER allows users to load an entire system (see Section 5.4.2.2) and, by retaining the global definitions table, to add or replace programs in an existing system. All symbolic references to system locations are resolved automatically by referencing the permanent definitions table.

5.2.3. NO RESTRICTIONS ON PROGRAM SIZE

The LOADER avoids program and system size restrictions by using RAD for all expandable storage tables. The illusion of "virtual memory" is achieved by using small core buffer areas in conjunction with RAD handlers which co-ordinate the transfer of segments to and from core.

The global definitions table is constructed and accessed in a manner analogous to the RAD resident tables of SYMBOL (See Section 4.3.2). A three sector (192 word) buffer area in core is overlaid by a handler which accesses any table location as though the entire table were core resident.

The program being loaded is generated directly to RAD in single sector (64 word) segments. Two single sector core buffer areas are accessed by handlers for input and output, performing RAD transfers only when sector boundaries are detected. The entire load is first loaded to a contiguous area of temporary RAD, requiring a temporary area slightly larger than the final total system size.

5.2.4 MULTIPLE INPUT TAPES

The Supervisory System is maintained on several source tapes, each terminated with an end-section mark. It is convenient to segment the system and to assemble each tape independently because of time considerations, producing several binary output tapes each ending with an end-section mark.

The LOADER recognizes end-section marks output by SYMBOL as indicating the end of a segment of binary information, possibly the end of information on the tape. Control is returned to the user at each end-section mark (see figure 5.1) in order that the LOADER can be instructed further (see Section 5.4). The user may use this feature to

load any system or segment of the system from as many individual tapes as desired, provided the last program on any binary tape is followed by an end-section mark.

5.2.5 SYSTEM SECURITY

The LOADER provides system security primarily by forcing the user to become involved with the operation; by requiring that he specify and initiate all potentially destructive operations. System security is especially important in view of the fact that the LOADER operates in a background mode in a real-time environment. All possible measures must be taken to ensure that the user does not accidentally destroy or overwrite any segment of the active system.

The following specific measures are employed:

(i) The user is required to specify through the input typewriter the starting permanent RAD address of the program or system being loaded. The LOADER echoes this address and the user must confirm it before the LOADER will proceed.

(ii) The LOADER advises the user of all end-section marks read from the input tape (Section 5.2.4). The user must initiate any further action by the LOADER.

(iii)The LOADER will not transfer the load from temporary to permanent RAD unless the user requests the transfer.

Although definitely not "fail safe", the precautionary measures incorporated into the LOADER are effective in that they force the user to monitor and initiate every critical phase of the loading procedure.

5.3 METHOD

5.3.1 GENERAL

The LOADER controller which determines the sequence of operations performed is flow charted in figure 5.1. Once the user has specified the starting permanent RAD address for the load, the LOADER performs the load in three passes:

Pass One
- (i) reads the binary information from magnetic tape,
- (ii) builds the definitions table of POPS, external definitions, and coreload names,
- (iii) relocates information according to the priority of the coreload,
- (iv) translates all POP references to their final sequence numbers,
- (v) stores the program, followed by its external references, on temporary RAD.

Pass one is repeated for each program until the user specifies that the last end-section read is the end of the load (Section 5.4.2.4), at which time pass two is initiated.

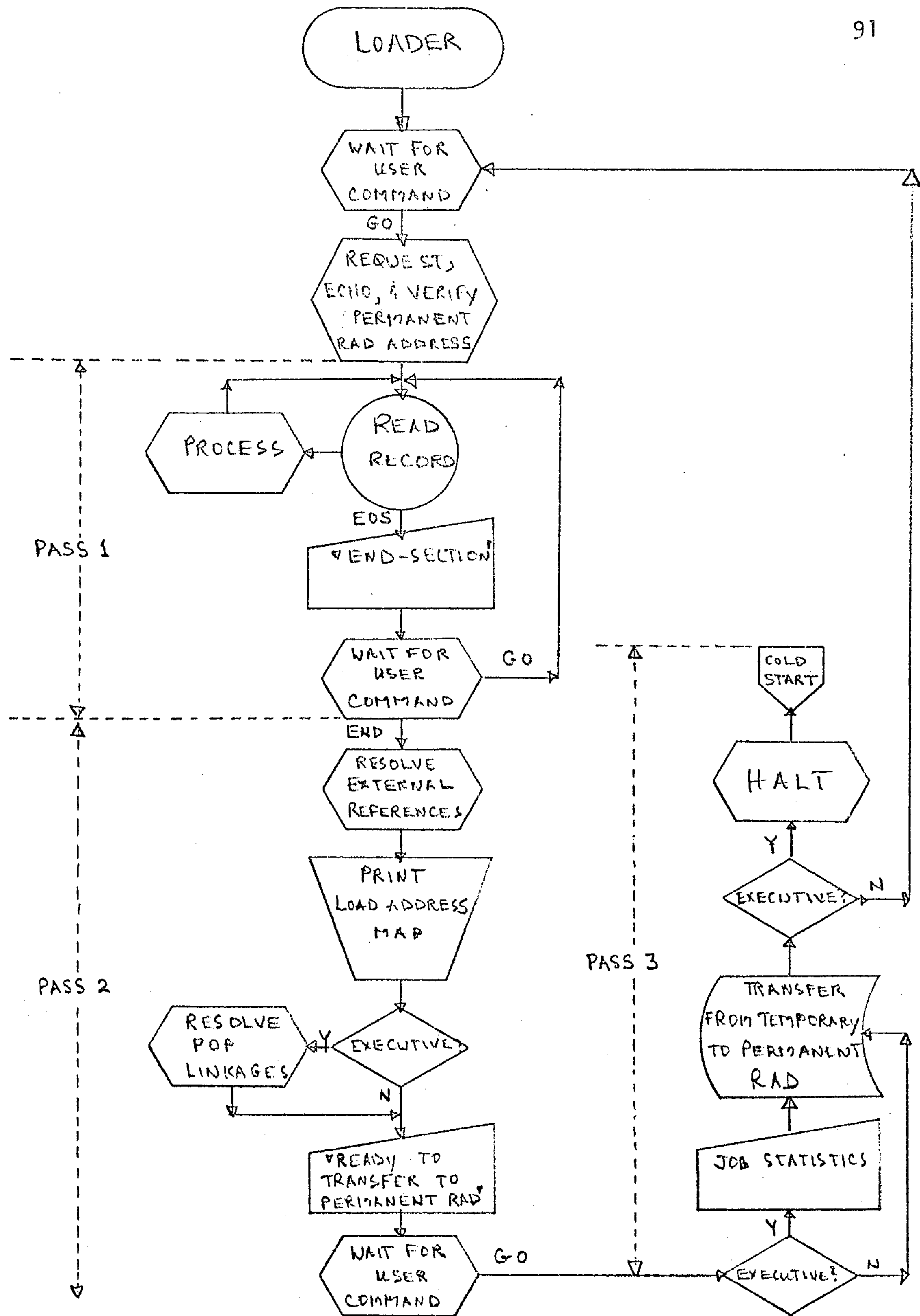


Figure 5.1 Loader Controller

Pass Two (i) reads the binary information from temporary RAD,

(ii) resolves external references by searching the definitions table generated in pass one,

(iii) purges external references from the end of each program,

(iv) stores the programs back on temporary RAD,

(v) outputs the external definitions table to the printer.

Once the definitions table has been output the user must initiate the transfer to permanent RAD. The user has the opportunity to check all diagnostic and definitions output and abort the load if errors must be resolved before integrating the load into the system.

Pass Three (i) reads the binary information from temporary RAD,

(ii) transfers the programs to permanent RAD.

Because pass three may be replacing segments of a running system, the system is disabled and a resident RAD handler does all RAD transferring. If the load was not a complete system, control is returned to the user who can initiate another load or sign off. If the LOAD was a complete system the LOADER halts, allowing the user to save the system on magnetic tape and then initiate the "cold start" procedure.

5.3.2 HANDLING BINARY RECORDS

The LOADER contains independent handlers to deal with each of the five possible different ypes of records:

(i) DELTA RECORDS

The delta record handler is flow charted in figure 5.2. Its function is to analyze the delta records described in section 4.3.4 and to establish the priority, name, and overlay bias for the coreload to follow. The handler also enters the previous coreload name and address into the definitions table.

(ii) DATA RECORDS

The data record handler is flow charted in figure 5.3. Data records contain the actual instructions and constants of the program and as such may have load relocation and/or POP relocation applied to any of the data words. The load relocation (execution bias) is determined by the previous delta record. The POP translation table will have been generated prior to reading the data records since POP reference and definition records are the first output from SYMBOL. This table will be accessed to convert all temporary POP sequence numbers to their final sequence numbers.

(iii) EXTERNAL REFERENCES AND DEFINITIONS

The handler for external references and definitions is flow charted in figure 5.4. External definitions are inserted with

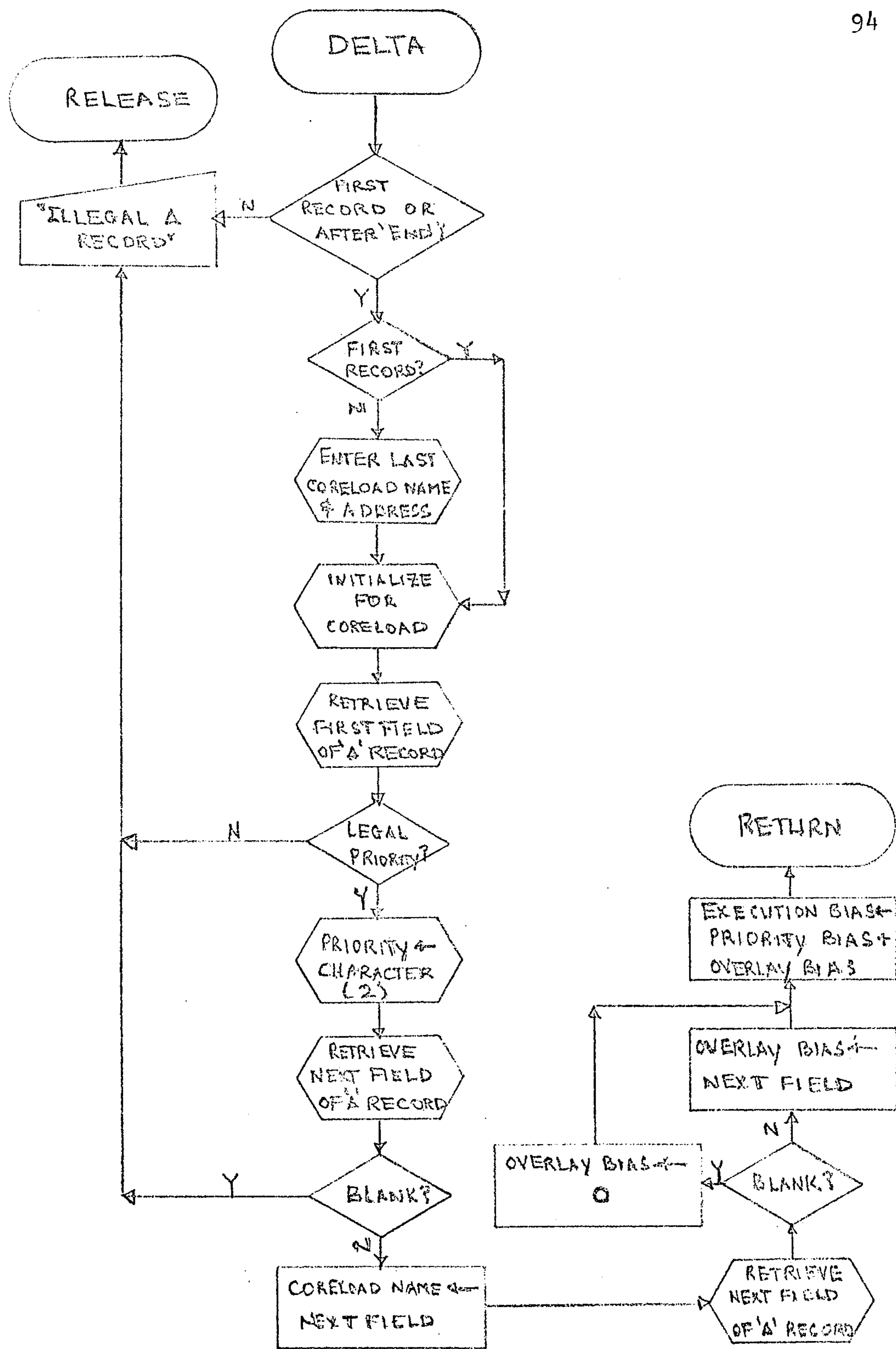


Figure 5.2 Delta Record Handler

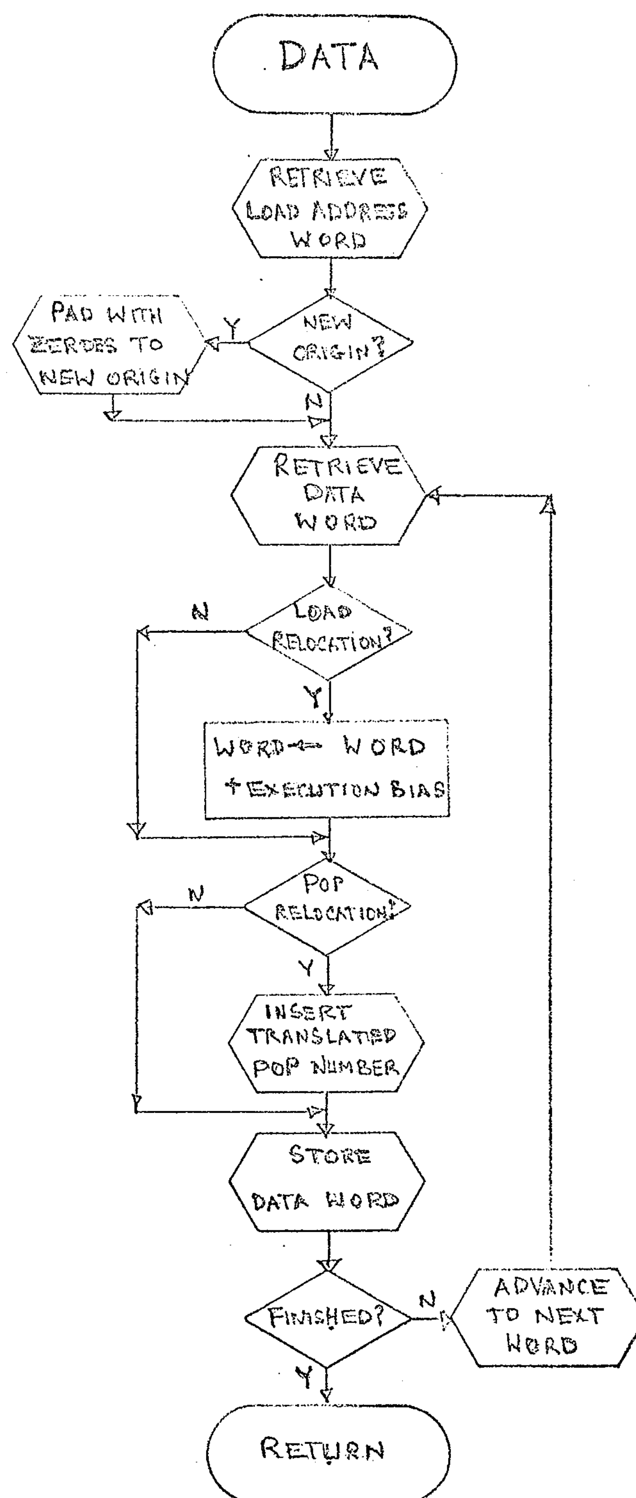
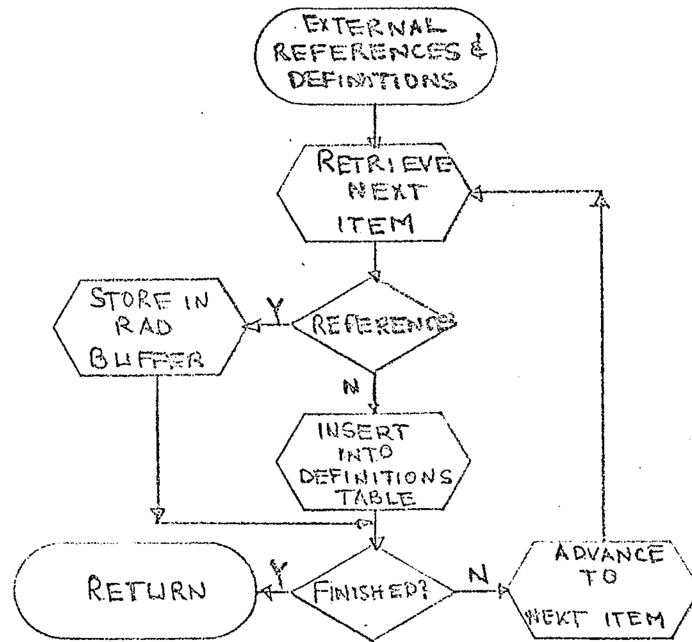


Figure 5.3 Data Record Handler



External Reference and Definition Handler

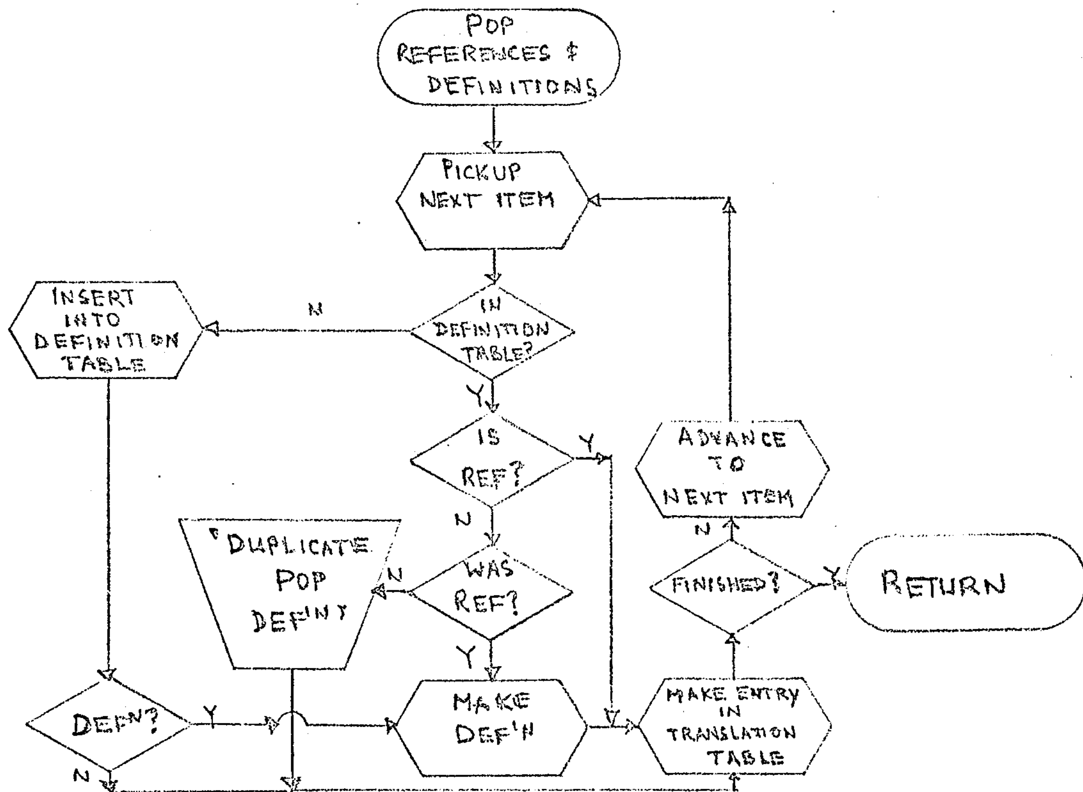


Figure 5.4 POP Reference and Definition Handler

their (core) execution addresses directly into the definitions table. External references cannot be resolved until pass two since the definitions table will not be complete until all programs have been read. During pass two all external references are output to temporary RAD following the last data word of the program and preceding the "end" record. The loader maintains a header for each program which forms part of the program output to temporary RAD. The starting address of unsatisfied references is one of the words of information stored in the header.

The LOADER handles duplicate definitions differently for a complete system load than a partial load. During a complete load the address of the first occurrence of the definition is retained whereas in a partial load the last occurrence is retained, since the user might be intentionally replacing an externally defined item.

(iv) POP REFERENCES AND DEFINITIONS

The handler for POP references and definitions is flow charted in figure 5.4. POP references and definitions are output by SYMBOL before pass two in order that the LOADER can build a translation table for assigning permanent POP sequence numbers before the data records are read. SYMBOL assigns all POPS temporarily as though they were local to the program, starting the sequence numbers at 0 and extending upwards as

required. The POP reference and definition output allows the LOADER to associate the global use of POP mnemonics with their temporary and permanent sequence numbers, and to build a translation table for the program being loaded. Duplicate POP definitions are treated like duplicate external definitions for a complete or partial load.

(v) END RECORDS

The handler for end records is flow charted in figure 5.5. The last word address of the program given in the end record (section 4.4) provides the length of the program and is used to update the core-load length, the origin of the next program, and the starting address of unsatisfied references in the program header. If the last word address given is not the same as the current location, the LOADER pads to the new origin with zeroes. The POP translation table is cleared in preparation for the start of the next program.

5.3.3. RESOLVING EXTERNAL REFERENCES

External references are not resolved until pass two when the definitions table is complete. Each external reference entry gives the address of the last location to reference that symbol and all references are chained to the reference location preceding it until the first reference with an address of 0, which terminates the chain. (Section 4.2). The algorithm flow charted in figure 5.6 for resolving

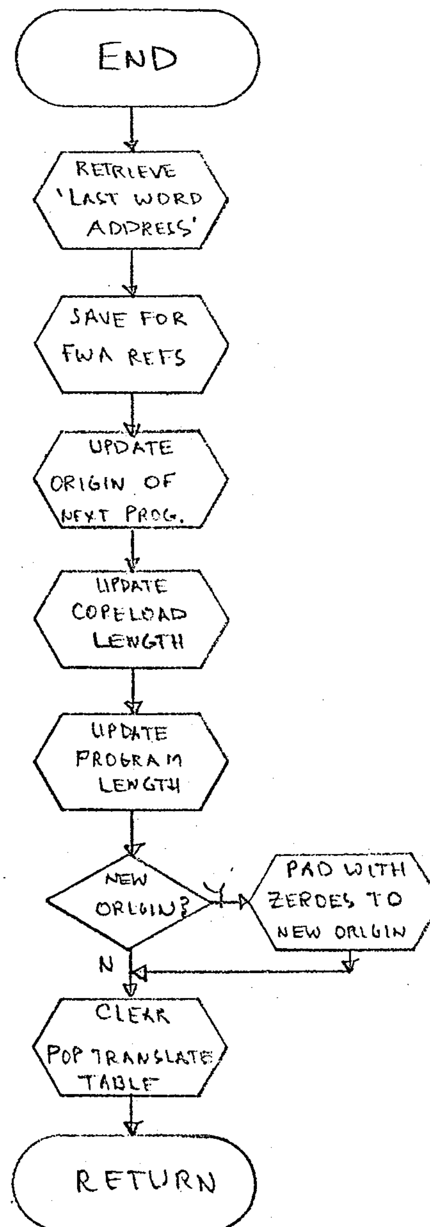


Figure 5.5 End Record Handler

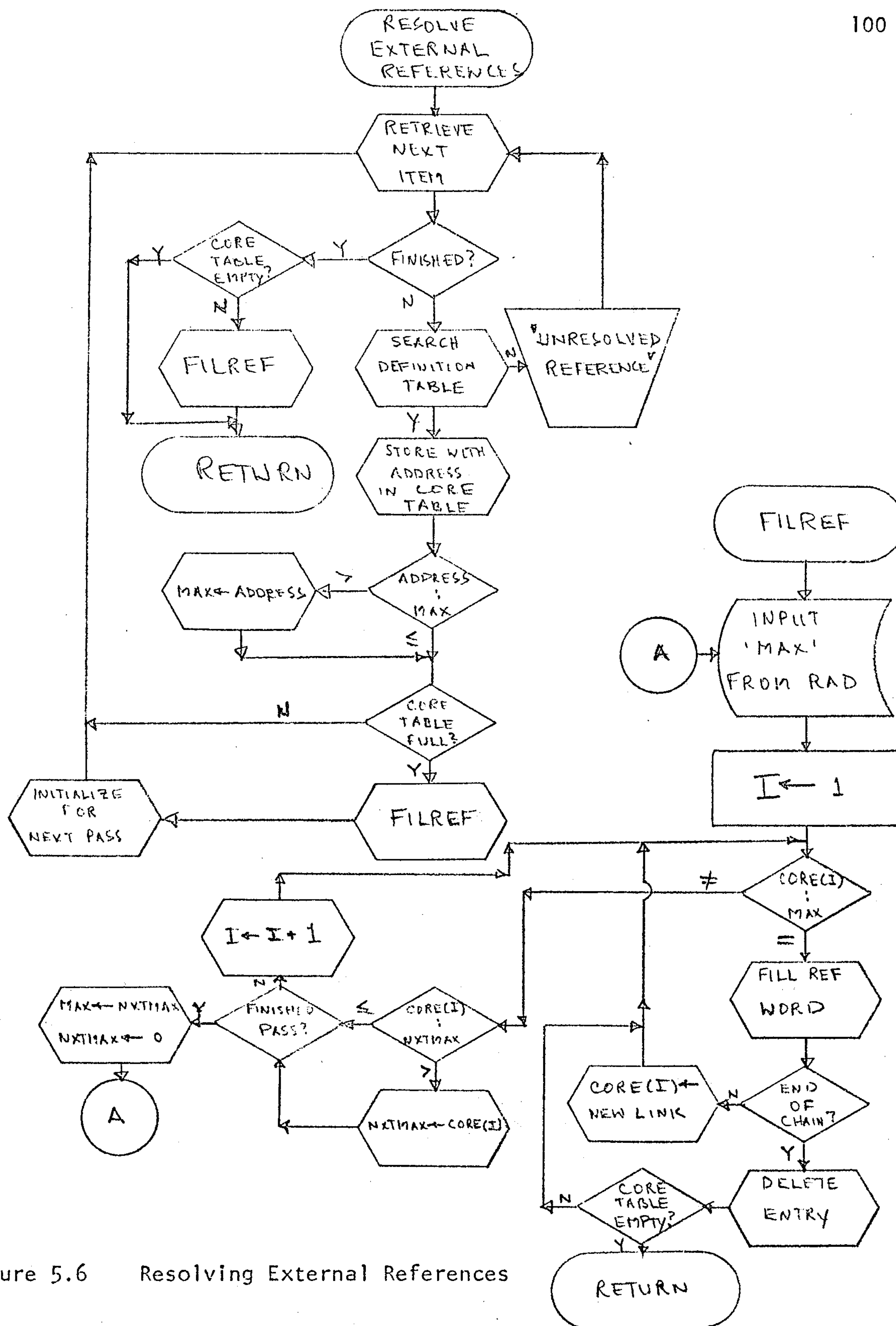


Figure 5.6 Resolving External References

external references uses a core table holding "n" entries and makes a complete pass backwards through all sectors (since the program is RAD resident) which contain external references. By keeping track of the next largest RAD address referenced by any entry in the reference chains, the algorithm is able to trade a slight increase in core manipulations for a significant decrease in RAD transferring. As each sector is input, all external references in this sector are resolved before going on to the next largest sector referenced. The procedure is repeated for the next "n" unresolved references and so on until all references have been resolved. The obvious technique of resolving all references in a single chain before going to the next would require all sectors having references to more than one external symbol to be input and output as many times as there are different external symbols.

5.3.4. LOAD ADDRESS MAP

The load address map output at the end of pass two provides the following information for each external symbolic name:

(i) The RAD or core address of the definition.

(ii) The priority if the entry is a coreload name.

(iii) The final POP sequence number if the entry is a POP.

(iv) A duplicate tag if the definition has been duplicated.

The POP output is separated from the coreload and external definitions in the listing (see example in appendix).

All entries in the definitions table have been made with the hashing algorithm described in section 4.3.2 and are known to be randomly distributed. The output from the reference table is sorted in two ways:

(i) In ascending order by address where core $<$ RAD.

(ii) In ascending order alphabetically using the collating sequence:

$$b < 0 < 1 < \text{----} < 9 < A < B < \text{----} < X < Y < Z$$

The standard sort techniques (e.g. "bubble" sort) which require physical rearrangement of the table are not feasible for two reasons:

(i) The number of RAD transfers would be prohibitive.

(ii) The reference table is a hash table used for the life of the system and must be left in the format dictated by the hash algorithm.

The sort algorithm employed is flow charted in figure 5.7. The technique used is similar in theory to that described in section 5.3.3. A core table with a capacity of "n" entries is loaded with the "n" smallest elements in the table by making a complete pass through the RAD table. This procedure is repeated with subsequent passes by ignoring all elements not larger than the largest element from the last pass until all elements have been sorted. If there are "x" elements in the table

having "y" segments, this sort algorithm will require $|x/n|y$ RAD transfers for a complete sort.

5.4 USER COMMANDS

Figure 5.1 illustrates that the user is requested to instruct the LOADER to take a specified course of action in the following instances:

(i) Immediately upon loading the program. The user may not wish to load programs but may want only an output of the definitions table (section 5.4.2.5).

(ii) After each end section is read. The user must instruct the LOADER to either continue with the next tape or indicate that all programs have been loaded.

(iii) After the load has been performed and before transferring it from temporary to permanent RAD.

Although the user is also requested at the start of the load to input the starting permanent RAD address to be used, this is not considered to be a command in the general sense. The LOADER will accept only an octal number in this case and none of the commands to be described in this section.

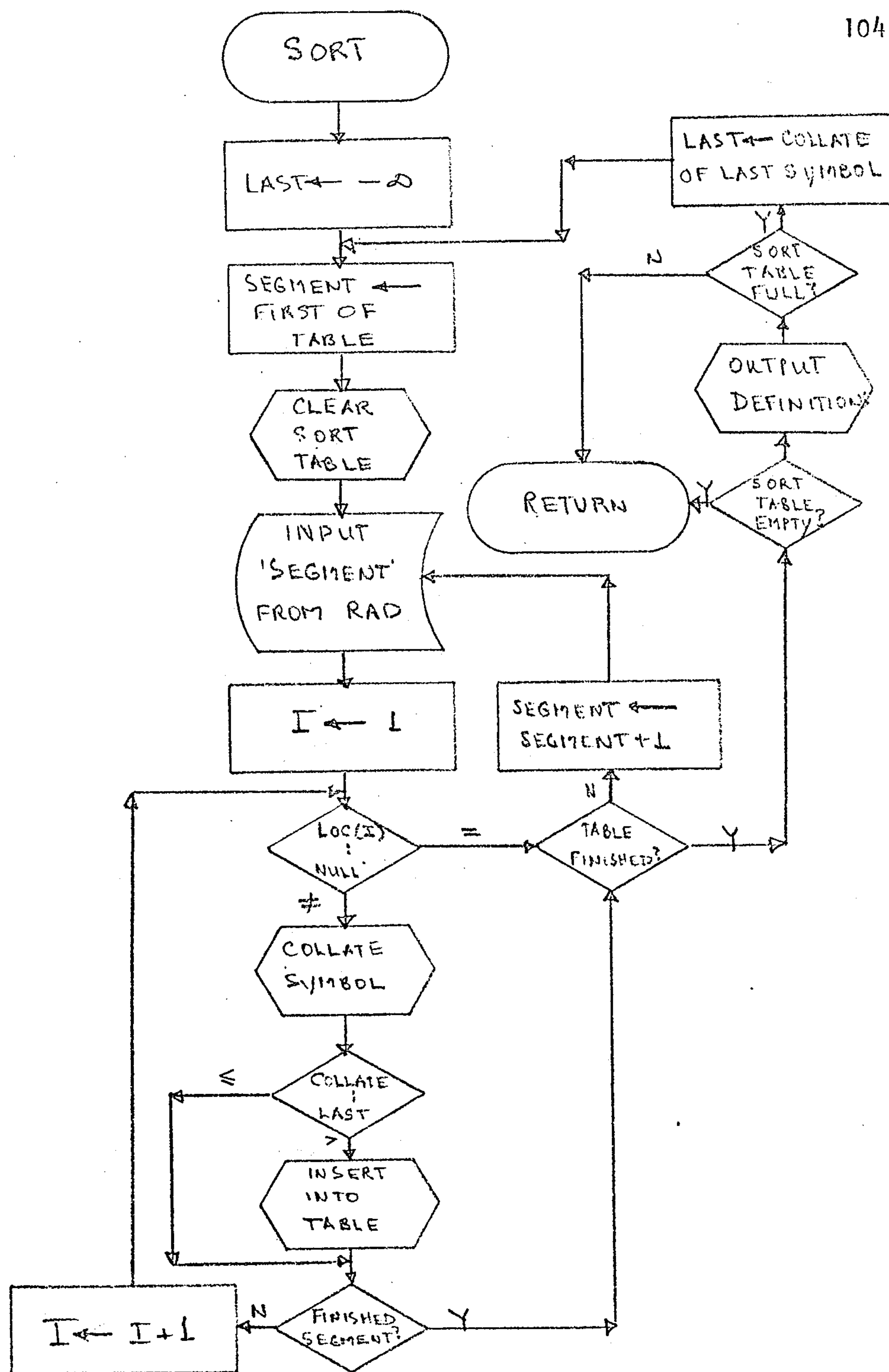


Figure 5.7 Sort Algorithm

5.4.1 GENERAL

The command analyzer is analogous to that used in the assembler (section 4.3.6). Commands may be input individually or several in the same statement separated by commas. Each command is executed interpretively until a transfer of control (GO, END, or TX) is executed.

Error analysis consists of the message "SYNTAX ERROR" output when analysis fails. Because commands are executed interpretively, all commands preceding the error will have been executed.

5.4.2 SPECIFIC COMMANDS

5.4.2.1 REWIND

This command rewinds magnetic tape unit two.

5.4.2.2 NEW SYSTEM

This command is to be used only when loading a completely new system as it causes the LOADER to purge the external definitions table at the start of the load. The LOADER also uses the first definition in cases of duplicate definitions when this option is invoked. Default is add to system if this option is not requested.

5.4.2.3 GO

This command causes the LOADER to continue with the present operation, i.e. return to the calling location.

5.4.2.4 END

This command causes the LOADER to begin pass two and is allowed only following an "end record" from tape.

5.4.2.5 DUMP

This command initiates the listing of the LOAD ADDRESS MAP (definitions table), and should be required only when multiple copies are required as the LOAD ADDRESS MAP is output automatically with each load.

5.4.2.6 TX

This command releases the LOADER after producing job statistics on the I/O typewriter.

5.5 PROGRAM SIZE AND RUN-TIME STATISTICS

The present version of the LOADER consists of 2100_{10} source records and requires 7700_8 words of core memory, allocated as follows:

Program and Constants	4600_8 words
Buffer Areas	3100_8 words
Core	7700_8 words (4K)

A variable amount of "temporary" RAD is used, amounting to a maximum of 1/10 more than the present system size (or 200K).

The run-time statistics are variable, depending on the tape density, the number of coreloads, and the number of tapes. The present complete system requires approximately twenty minutes to load, including all delays as a result of having to wait for the user to mount tapes (maximum of two to three minutes delay).

CHAPTER 6

CONCLUSIONS AND FURTHER DEVELOPMENTS

According to a recent survey conducted by Control Engineering [8], only 38% of users of on-line real-time systems are able to assemble programs on-line. The necessity of an on-line program development package for this application can be better appreciated if it is realized that Alberta Gas Trunk Line has several "uncommon" characteristics:

(i) Computer control of gas transmission facilities is an infant industry and as such does not have access to proven techniques for utilizing Supervisory Systems. New procedures and applications must be researched, developed, implemented, and improved within the company; an approach which requires considerable computer time for program development and testing.

(ii) The growth rate of the pipeline facilities is very rapid and requires constant updating of the Supervisory programs to keep pace with expansion.

(iii)The system is on-line 24 hours a day and is required to scan the pipeline <u>at least</u> every fifteen minutes. Although the effect of major changes often takes hours to observe, a response time of five to ten minutes is considered necessary for initiating stabilizing action in the event of "upset conditions".

(iv) The emergency backup system is so ineffective as to be considered useless except in "panic situations". It does not provide sufficient information display and control capabilities to replace the on-line system while program development is carried out.

Statistics kept for the package over a sixty day period indicate that usage averages 1.5 hours per day, allocated as follows:

UTILITY	45%
SYMBOL	50%
LOADER	5%

The allocation varies considerably with job mix since an increase in "new" programs results in an increase in "test assemblies" for each update by UTILITY. The proportion of time used by the UTILITY program will increase over that used by SYMBOL in periods of increased program development.

UTILITY has been in use for over a year while SYMBOL and LOADER have been in use for only three months at the date of this writing. Extrapolating the statistics for UTILITY back for a year gives a conservative total usage for the package to date of three hundred to four hundred hours. Downtime for program development has been virtually eliminated by the package, giving a tangible increase in system availability of over 6% per day (1.5 hours). Updating and program development are also more efficient since the user is assured that he can run and debug his job interactively and at his convenience.

Emphasis is gradually being shifted towards the "applications" programming area (e.g. simulations, trend analysis, optimization, closed loop control, and Management Information Systems), as the process of data gathering and display becomes more reliable and accessible. It is expected that program development will increase sharply as users become more aware of the numerous possibilities and capabilities of the system.

Use of the package can be extended over the life of the CPU, which has a remarkable record of reliability for the past six years. The CPU could not "conveniently" be replaced in less than two years and will probably be in use for up to five years if reliability does not become a problem. Size and speed are not immediate concerns since the foreground task of monitoring and controlling the system requires only 20 to 30% of the system time and leaves 450K words of RAD storage available for future development.

In spite of the fact that the present demands on the package are well within design limitations, there are general areas of improvement which would considerably increase throughput:

(i) Interlace on the W I/O channel (Buffer) would allow overlapping of I/O and processing for the devices attached to it and would make designing I/O handlers for the card reader and magnetic tapes more economic.

(ii) Faster card reader and line printer would increase throughput since the programs are presently almost completely I/O bound.

(iii) Redesign of the I/O system around spooling of I/O by job (i.e. priority) in conjunction with (i) and (ii) would eliminate waiting for I/O in most applications. RAD could be used as the intermediate storage medium and all references to specific I/O devices could be made indirectly to the designated area on the RAD. Conflicts arising over use of common peripherals would be virtually eliminated.

There are also specific areas in each of the programs which are scheduled for further development:

6.1 UTILITY DEVELOPMENT

(i) Addition of commands to permit editing and updating of binary output (object) tapes using coreload names and/or end-sections as editing boundaries. This feature would provide the ability to replace single programs on a binary tape and to reload the system without having to re-assemble the entire tape.

6.2 SYMBOL DEVELOPMENT

(i) Modify pass one to output source records to RAD and re-read from RAD rather than from magnetic tape for pass two.

(ii) Modify the I/O package to allow the user to select the source input device (i.e. cards or magnetic tape) rather than fixing it to magnetic tapes.

(iii) Allow the user to select the command input device (i.e. cards or typewriter) so that assemblies can be "batched" and signed off automatically with input from cards or magnetic tape.

(iv) Provision for an optional version which uses a RAD resident MNEMONIC table which will allow more than the present fifty user-defined operation code symbols.

(v) Provision for dumping global symbol tables (which are common to most application programs) to magnetic tape for subsequent input to other assemblies.

(vi) Provision to select the program (s) to be assembled (by specifying sequence start and end) from a batch of programs on a single tape.

6.3 LOADER DEVELOPMENT

(i) Modification to have all "new" entries to the definitions table tagged and purged from the table if the programs are not transferred to permanent RAD.

(ii) Default dump of the LOAD ADDRESS MAP which prints only the "new" definitions as in (i).

(iii) Provision for a version which does not require temporary RAD storage in the event the system grows to require over half of the RAD installed.

The fact that the author is familiar with the Supervisory System and has served as analyst, programmer, and user of the package, has been essential to its success in providing the services described. As the Supervisory System changes and users become more demanding, the package will also have to change to meet the new requirements. In conclusion, the package as described is performing a very essential service, and has been well received by all connected with the operation and upkeep of the Supervisory System.

APPENDIX I

USE OF THE PACKAGE

The appendix demonstrates the typical use of the package by following the development of a program (coreload) from generation onto magnetic tape through loading it into the system.

Figure A1.1 illustrates the console typewriter I/O, segmented as the I/O applies to each of the six separate jobs.

JOB (1)

This job represents generating the program from cards to magnetic tape. The output from UTILITY is given in figure A1.2.

JOB (2)

This job represents the first "test assembly" to detect syntax errors which can be caught by the assembler. The default options of SYMBOL are invoked and produce the listing given in figure A1.3. This test assembly illustrates two errors:

(i) The call to WAIT appears before the I/O has been requested.

(ii) A keypunching error in record 180 has created a POP reference to LDR rather than the legitimate mnemonic LDX.

A visual check of the remainder of the output shows that there are four legal external references and three legal POP references.

Figure A1.1 Console Typewriter I/O

1.

```
UTILITY

 **24MAR72     1233    UTILITY ØØ/ØØ/29 TO ØØ8/24
```

2.

```
SYMBOL
REWIND SI,GO
REWIND SI,TX

 **24MAR72     1234    SYMBOL  ØØ/ØØ/27 TO Ø1Ø/59
```

3.

```
UTILITY

 **24MAR72     1234    UTILITY ØØ/ØØ/11 TO ØØ8/25
```

4.

```
SYMBOL
REWIND SI,GO
REWIND SI,TX

 **24MAR72     1235    SYMBOL  ØØ/ØØ/25 TO Ø11/ØØ
```

5.

```
SYMBOL
REWIND SI,REWIND BO,LIST ALL,BO,GO
REWIND SI,REWIND BO,TX

 **24MAR72     1236    SYMBOL  ØØ/ØØ/49 TO Ø11/Ø1
```

6.

```
LOADER
REWIND,NEW SYSTEM,GO

INPUT PERMANENT RAD ADDRESS   2ØØØØ
PERMANENT RAD ADDRESS    #   ØØØ2ØØØØ
GO
END-SECTION READ
REWIND,END
READY TO TRANSFER TO FINAL RAD GO
TX

 **24MAR72     1237    LOADER  ØØ/Ø1/Ø3 TO ØØ1/23
```


121360

```
CØMMAND=      REWIND ØUTPUT
CØMMAND=      SEQUENCE FRØM 10 BY 10
       Δ4     SAMPLE                                                        00000010
       *  SAMPLE CØRELØAD TØ DEMØNSTRATE THE TYPICAL USE ØF THE ØN-LINE     00000020
       *    PRØGRAM DEVELØPMENT PACKAGE FØR GENERATING, UPDATING, AND       00000030
       *      LØADING A CØRELØAD.                                           00000040
              RØRG     0                                                    00000050
              RES      20                SYSTEM WØRK AREA                   00000060
              PZE      START             ADDRESS ØF FIRST INSTRUCTIØN       00000070
       JØBWA  PZE                        ADDRESS ØF WØRK AREA FØR THE JØB   00000080
              BCD      8, SAMPLE         CØRELØAD NAME                      00000090
       START  LDX      JØBWA                                                00000100
              LDA      2,2                                                  00000110
              STA      TABLQ+5           RAD ADDRESS ØF TABLE               00000120
              LDA      =TABLQ            FIRST WØRD ADDRESS ØF I/Ø WØRK AREA 00000130
              EAX      $+2                                                  00000140
              BRU      WAIT              WAIT FØR I/Ø TØ FINISH             00000150
              EAX      $+2                                                  00000160
              BRU      IØSYS             REQUEST INPUT ØF RAD TABLE         00000170
              LDR      =TABLE                                               00000180
              LDP      1,2                                                  00000190
              FLM      D126                                                 00000200
              STD      5,2               TABLE(5)=TABLE(1)*126/TABLE(3)     00000210
              LDA      =1*/15                                               00000220
              STA      TABLQ+1           I/Ø FUNCTIØN                       00000230
              LDA      =TABLQ                                               00000240
              EAX      $+2                                                  00000250
              BRU      IØSYS             REQUEST ØUTPUT ØF RAD TABLE        00000260
              EAX      $+2                                                  00000270
              BRU      WAIT              WAIT FØR I/Ø TØ FINISH             00000280
              BRU      RELEAS            RELEASE JØB                        00000290
       TABLQ  DATA     $,0,TABLE,01000                                      00000300
              PZE      WD                WAITING I/Ø FLAG FØR THIS PRIØRITY 00000310
              DATA     0,0                                                  00000320
       D126   DED      126.                                                 00000330
       TABLE  EQU      $+3               FIRST WØRD ADDRESS ØF TABLE AREA   00000340
              END                                                           00000350
CØMMAND=      END SECTIØN    /END ØF INFØRMATIØN ØN TAPE
CØMMAND=      REWIND ØUTPUT
CØMMAND=      TX
```

Figure A1.2 Generate Coreload

```
                                  1   Δ4      SAMPLE                                                              00000010
*     00035  0 01 00000          15           BRU      WAIT              WAIT FØR I/Ø TØ FINISH                   00000150
*     00037  0 01 00000          17           BRU      IØSYS             REQUEST INPUT ØF RAD TABLE               00000170
I     00040  1 01 00066          18           LDR      =TABLE                                                     00000180
I     00041  3 00 00001          19           LDP      1,2                                                        00000190
I     00042  1 03 00063          20           FLM      D126                                                       00000200
I     00043  3 02 00005          21           STD      5,2               TABLE(5)=TABLE(1)*126/TABLE(3)           00000210
*     00050  0 01 00037          26           BRU      IØSYS             REQUEST ØUTPUT ØF RAD TABLE              00000260
*     00052  0 01 00035          28           BRU      WAIT              WAIT FØR I/Ø TØ FINISH                   00000280
*     00053  0 01 00000          29           BRU      RELEAS            RELEASE JØB                              00000290
*     00060  0 00 00000          31           PZE      WD                WAITING I/Ø FLAG FØR THIS PRIØRITY       00000310
      00052  WAIT
      00050  IØSYS
      00060  WD
      00053  RELEAS
*** 00/00/19 ELAPSED TIME, MAXIMUM LØCATIØN =  00067
***    5 SYMBØLS ,      3 LITERALS ,     4 REFS
```

Figure A1.3   Test Assembly

JOB (3)

This job is an update by UTILITY to correct the errors detected in JOB (2). Four functions are to be performed:

(i) The call to WAIT is deleted.

(ii) The call to WAIT is inserted in the correct position.

(iii) The record with the keypunching error is replaced.

(iv) The remainder of the program is copied to the OUTPUT tape.

The output from the job is illustrated in figure A1.4. The first record of the "insert" contains two sequence numbers; the right most being the sequence number punched on the CID source record (implicit command), and the left most being the final sequence number assigned to the record on the OUTPUT tape.

JOB (4)

This job is a test assembly on the output from JOB (3) to verify that the update was successful. The output in figure A1.5 demonstrates that the errors have in fact been corrected and the program is free of visible errors.

JOB (5)

This job is the final assembly, complete with listing and binary output. The listing of the complete program as output by SYMBOL is given in figure A1.6

```
COMMAND=        REWIND INPUT
COMMAND=        REWIND OUTPUT
COMMAND=        SEQUENCE FROM 10 BY 10
COMMAND=        SKIP 140 THRU 150
INSERT          EAX     $+2                                                      00000160    00000171
                BRU     WAIT              WAIT FOR I/O TO FINISH                 00000170
REPLAC          LDX     *TABLE                                                   00000180    00000180
COMMAND=        COPY 1 SECTION
COMMAND=        REWIND INPUT
COMMAND=        REWIND OUTPUT
COMMAND=        TX
```

Figure A1.4 Update

```
                                   1   Δ4      SAMPLE                                                   00000010
*      00035  0 01 00000          15           BRU      IØSYS          REQUEST INPUT ØF RAD TABLE       00000150
*      00037  0 01 00000          17           BRU      WAIT           WAIT FØR I/Ø TØ FINISH           00000170
I      00041  3 00 00001          19           LDP      1,2                                             00000190
I      00042  1 02 00063          20           FLM      D126                                            00000200
I      00043  3 01 00005          21           STD      5,2            TABLE(5)=TABLE(1)*126/TABLE(3)   00000210
*      00050  0 01 00035          26           BRU      IØSYS          REQUEST ØUTPUT ØF RAD TABLE      00000260
*      00052  0 01 00037          28           BRU      WAIT           WAIT FØR I/Ø TØ FINISH           00000280
*      00053  0 01 00000          29           BRU      RELEAS         RELEASE JØB                      00000290
*      00060  0 00 00000          31           PZE      WD             WAITING I/Ø FLAG FØR THIS PRIØRITY 00000310
       00052  WAIT
       00050  IØSYS
       00060  WD
       00053  RELEAS
*** 00/00/18 ELAPSED TIME, MAXIMUM LØCATIØN =  00067
***     5 SYMBØLS ,      3 LITERALS ,     4 REFS
*
```

Figure A1.5   Test Assembly

Figure A1.6 Final Assembly

```
                                  1   Δ4      SAMPLE                                                         00000010
                                  2   *  SAMPLE CORELOAD TO DEMONSTRATE THE TYPICAL USE OF THE ON-LINE       00000020
                                  3   *    PROGRAM DEVELOPMENT PACKAGE FOR GENERATING, UPDATING, AND         00000030
                                  4   *       LOADING A CORELOAD.                                            00000040
   00000                          5           RORG     0                                                     00000050
   00000                          6           RES      20                   SYSTEM WORK AREA                 00000060
   00024  0 00 00030              7           PZE      START                ADDRESS OF FIRST INSTRUCTION     00000070
   00025  0 00 00000              8   JOBWA   PZE                           ADDRESS OF WORK AREA FOR THE JOB 00000080
   00026  12622144                9           BCD      8, SAMPLE            CORELOAD NAME                    00000090
   00027  47432512
   00030  0 71 00025             10   START   LDX      JOBWA                                                 00000100
   00031  2 76 00002             11           LDA      2,2                                                   00000110
   00032  0 35 00061             12           STA      TABLQ+5              RAD ADDRESS OF TABLE             00000120
   00033  0 76 00065             13           LDA      =TABLQ               FIRST WORD ADDRESS OF I/O WORK AREA 00000130
   00034  0 77 00036             14           EAX      $+2                                                   00000140
*  00035  0 01 00000             15           BRU      IOSYS                REQUEST INPUT OF RAD TABLE       00000150
   00036  0 77 00040             16           EAX      $+2                                                   00000160
*  00037  0 01 00000             17           BRU      WAIT                 WAIT FOR I/O TO FINISH           00000170
   00040  0 71 00066             18           LDX      =TABLE                                                00000180
I  00041  3 00 00001             19           LDP      1,2                                                   00000190
I  00042  1 02 00063             20           FLM      D126                                                  00000200
I  00043  3 01 00005             21           STD      5,2                  TABLE(5)=TABLE(1)*126/TABLE(3)   00000210
   00044  0 76 00067             22           LDA      =1*/15                                                00000220
   00045  0 35 00055             23           STA      TABLQ+1              I/O FUNCTION                     00000230
   00046  0 76 00065             24           LDA      =TABLQ                                                00000240
   00047  0 77 00051             25           EAX      $+2                                                   00000250
*  00050  0 01 00035             26           BRU      IOSYS                REQUEST OUTPUT OF RAD TABLE      00000260
   00051  0 77 00053             27           EAX      $+2                                                   00000270
*  00052  0 01 00037             28           BRU      WAIT                 WAIT FOR I/O TO FINISH           00000280
*  00053  0 01 00000             29           BRU      RELEAS               RELEASE JOB                      00000290
   00054  00000054               30   TABLQ   DATA     $,0,TABLE,01000                                       00000300
   00055  00000000
   00056  00000070
   00057  00001000
*  00060  0 00 00000             31           PZE      WD                   WAITING I/O FLAG FOR THIS PRIORITY 00000310
   00061  00000000               32           DATA     0,0                                                   00000320
   00062  00000000
   00063  00000007               33   D126    DED      126.                                                  00000330
   00064  37400000
          00000070               34   TABLE   EQU      $+3                  FIRST WORD ADDRESS OF TABLE AREA 00000340
                                 35           END                                                            00000350
   00065  00000054
   00066  00000070
   00067  00100000
   00052  WAIT
   00050  IOSYS
   00060  WD
   00053  RELEAS
*** 00/00/38 ELAPSED TIME, MAXIMUM LOCATION =  00067
***    5 SYMBOLS ,     3 LITERALS ,     4 REFS
```

JOB (6)

This job is the loading of the binary output from JOB (5) and integrating it into the system. The NEW SYSTEM option is requested to make the coreload independent and to keep the output of the LOAD ADDRESS MAP as brief as possible. Figures A1.7.1 through A1.7.5 give the output from the LOADER.

Figure A1.7.1 lists all external references as unresolved since the coreload was loaded as an independent system and therefore did not have access to the external definitions table of the existing system.

Figures A1.7.2 and A1.7.3 give the sorted output of the POP table. The eight digit octal number associated with each POP gives the following information (from left to right):

BIT	CONTENTS
0	= 1 if POP definition
1	= 1 if POP reference
2 - 8	POP sequence number
9 - 23	Origin of POP routine

All POPS in this load are references only since the executive, containing the POP library, has not been loaded. This would normally constitute a program error since it would be known that there are references to POPS which are not defined.

```
24MAR72     1237    UNRESØLVED REFERENCES

SAMPLE  UNRESØLVED REFERENCE    WAIT     AT 00052
SAMPLE  UNRESØLVED REFERENCE    IØSYS    AT 00050
SAMPLE  UNRESØLVED REFERENCE    WD       AT 00060
SAMPLE  UNRESØLVED REFERENCE    RELEAS   AT 00053
```

Figure A1.7.1 Load

Figure A1.7.2

24MAR72	1237	SYSTEM PØPS(BY ADDRESS)			
FLM	20200000	STD	20100000	LDP	20000000

24MAR72 1237 SYSTEM PØPS(ALPHABETICALLY)

FLM	20200000	LDP	20000000	STD	20100000

Figure A1.7.3

```
24MAR72     1237    EXTERNAL DEFINITIONS(BY ADDRESS)

 4  SAMPLE  40120000
```

Figure A1.7.4

```
24MAR72      1237     EXTERNAL DEFINITIONS(ALPHABETICALLY)

   4  SAMPLE   40120000
```

Figure A1.7.5

Figures A1.7.4 and A1.7.5 give the sorted output of the definitions table. The sorting in this case is meaningless since there is only one definition. The digit "4" preceding the coreload name SAMPLE is the priority of the coreload, while the 8 digit octal number associated with it contains the following (from left to right):

BIT	CONTENTS
0	= 1 if RAD address
1 - 8	length (in sectors) if RAD address
9	always = 0 (unused)
10 - 23	RAD/core address of definition.

The address 40120000 indicates that it is a RAD table of length 1 sector (less than 64 words) starting at RAD ADDRESS 20000.

REFERENCES

1. Cheatham, T.E., and Sattley, K.
Syntax Directed Compiling
In:
Rosen, S. (Ed.) Programming Systems and Languages
McGraw Hill (1967), 264-297.

2. Martin, J. Programming Real-Time Computer Systems
Prentice - Hall Inc. (1965).

3. Martin, J. Design of Real-Time Computer Systems
Prentice - Hall Inc. (1967).

4. Xerox Data Systems XDS 920 Computer Reference Manual (1965).

5. Xerox Data Systems XDS Symbol and Meta-Symbol
Publication Number 900506G (1969).

6. Xerox Data Systems XDS 900 Series Symbol Technical Manual
Publication Number 900688C (1967).

7. Xerox Data Systems XDS 920/930 Computer Programmed Operators Technical Manual Publication Number 900020F (1967).

8. Kompass, E.J., A Survey of On-Line Control Computer Systems Control Engineering (January 1972) 52-56.

9. Malia, T.C. and Dickson, G.W. Management Problems Unique to On-Line Real-Time Systems AFIPS Proceedings Volume 37 (1970) FJCC.