December 1968

On-Line Computing With a Hierarchy of Processors

Richard P. Morton
University of Pennsylvania

On-Line Computing With a Hierarchy of Processors

Abstract

Time shared computer systems have been based upon the two techniques of multiprogramming and swapping. Multiprogramming is based on restricting each program to a portion of the total computer memory. Swapping requires considerable overhead time for loading and unloading programs. To alleviate the size restriction due to multiprogramming, segmentation is employed, resulting in fact in vastly increased swapping.

A new system architecture is proposed for time shared computing that alleviates the high overhead or program size restriction. It utilizes a hierarchy of processors, where each processor is assigned tasks on the basis of four factors: interactive requirements, frequency of use, execution time, and program length.

In order to study the hierarchical approach to system architecture, the Moore School Problem Solving Facility (MSPSF) was built and used. The study of the manner of operation and the reactions of the users clarified and defined the Hierarchy of Processors system architecture.

The Moore School Problem Solving Facility was implemented on second generation equipment, the IBM 7040, and therefore it is not possible to adequately compare the efficiency with third generation computers operating in a swapping mode. The conclusions of this dissertation center around the methodology of designing such a system, including the specification of facilities for each level of the hierarchy.

Six major conclusions are given:

(1) Three processors in the hierarchy have been necessary, but it is conceivable that more may be employed in other future situations.

(2) Each of the processors in the hierarchy should be general purpose.

(3) Program compatibility between the processors is necessary.

(4) The assigning of tasks to the processors within the system should be optionally user directed or automatic. Similarly, if a task exceeds the resources of the processor to which it has been assigned, redirection should be possible either automatically or by the user.

(5) A macro language is necessary between every pair of processors for effective communication. Such a language processor, IXSYS, has been constructed and its use is described in detail in the dissertation, demonstrating the need and utility.

(6) In addition to the three hierarchical processors, a separate processor may be advantageously used for storage, retrieval and management of information in files. Such a processor should be directly accessible from each of the other processors.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-69-13.

University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
Philadelphia, Pennsylvania 19104

TECHNICAL REPORT

ON-LINE COMPUTING WITH A HIERARCHY OF PROCESSORS

by

Richard P. Morton

December 1968

Submitted to the
Office of Naval Research
Information Systems Branch
Washington, D. C. 20360

and

Rome Air Development Center
Griffiss Air Force Base, New York

under
Contract NOnr 551(40)
Research Project No. 003-08-01

Moore School Report No. 69-13

ON-LINE COMPUTING WITH A HIERARCHY OF PROCESSORS

Richard Phillip Morton

A DISSERTATION

in

Electrical Engineering

Presented to the Faculty of the Graduate School of Arts and Sciences of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy.

1968

Noah Prywes

Supervisor of Dissertation

Graduate Group Chairman

ABSTRACT

Time shared computer systems have been based upon the two techniques of multiprogramming and swapping. Multiprogramming is based on restricting each program to a portion of the total computer memory. Swapping requires considerable overhead time for loading and unloading programs. To alleviate the size restriction due to multiprogramming, segmentation is employed, resulting in fact in vastly increased swapping.

A new system architecture is proposed for time shared computing that alleviates the high overhead or program size restriction. It utilizes a hierarchy of processors, where each processor is assigned tasks on the basis of four factors: interactive requirements, frequency of use, execution time, and program length.

In order to study the hierarchical approach to system architecture, the Moore School Problem Solving Facility (MSPSF) was built and used. The study of the manner of operation and the reactions of the users clarified and defined the Hierarchy of Processors system architecture.

The Moore School Problem Solving Facility was implemented on second generation equipment, the IBM 7040, and therefore it is not possible to adequately compare the efficiency with third generation computers operating in a swapping mode. The conclusions of this dissertation center around the methodology of designing such a system, including the specification of facilities for each level of the hierarchy.

Six major conclusions are given:

(1) Three processors in the hierarchy have been necessary, but it is conceivable that more may be employed in other future situations.

(2) Each of the processors in the hierarchy should be general purpose.

(3) Program compatibility between the processors is necessary.

(4) The assigning of tasks to the processors within the system should be optionally user directed or automatic. Similarly, if a task exceeds the resources of the processor to which it has been assigned, redirection should be possible either automatically or by the user.

(5) A macro language is necessary between every pair of processors for effective communication. Such a language processor, IXSYS, has been constructed and its use is described in detail in the dissertation, demonstrating the need and utility.

(6) In addition to the three hierarchical processors, a separate processor may be advantageously used for storage, retrieval and management of information in files. Such a processor should be directly accessible from each of the other processors.

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to my supervisor, Dr. Noah S. Prywes for his guidance throughout the course of this work. His assistance during the composition of this document has been particularly invaluable.

Technical assistance was received from several members of the Moore School Staff, notably Michael S. Wolfberg. Others include D. K. Hsiao, D. M. Kristol, M. Gelblat, P. Flint, P. R. Weinberg, and C. A. Kapps. Members of the University Computer Center staff that assisted me in the generation of some of the programs include W. S. Mosteller, R. B. Keller, R. Iovacchini, P. Wolfgang, and W. Jansen.

Administrative assistance came from Drs. J. F. Lubin and G. L. Schrenk of the Computer Center. The entire Computer Center Staff has my thanks for their help.

For typing and all the other secretarial functions which cause things to get done, my sincerest thanks go to Miss Connie Murray.

My appreciation to Information Systems Branch of the Office of Naval Research and the Rome Air Development Center is sincerely acknowledged for their support of this research under Contract Nonr 551(40).

Finally, I would like to thank my cheering squad for all that they have contributed in the way of support and encouragement. Included are my wife, parents, brother, sister, parents-in-law, aunts, uncles, and friends.

INDEX

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (continued)

1. INTRODUCTION

1.1 Problem Background

1.1.1 Remote Access Computing

Computing systems which allow remote access have experienced phenomenal growth within the past five years. The variety of special and general purpose systems allowing remote access has grown to the point where recently novel applications are now widespread. Major computing facilities either have already installed equipment for remote access or such equipment is planned or on order. New families of computers have been designed and are being marketed for remote access computing. New industries of computer communications and terminals for remote access have grown up during this period.

The purposes of this relatively new style of computer system have been twofold. First, remote terminals have made the computer more readily available to the user of conventional computing systems. Remote access has meant the use of a conveniently located terminal, such as in his office where a user prepares his program, runs it immediately, corrects errors, and reruns it until he has obtained the functions that he requires. Total elapsed time has been shortened from days or weeks to minutes or hours. The user is able to get answers in a fraction of the time it previously required.(1)

(1) There have been several comparisons made between online and offline computing. See, for example, H. Sackman, W. J. Erikson, and E. E. Gran, Exploratory Experimental Studies Comparing Online and Offline Programming Performance, Comm. ACM, 11:1, 1968; or M. Schatzoff, R. Tsao, and R. Wiig, An Experimental Comparison of Time Sharing and Batch Processing, Comm. ACM, 10:5, 1967.

Second, remote computing has opened totally new areas of computer usage. Applications considered too uneconomical to require a computer installation can now be carried out by sharing a portion of the cost of a remote computer. New applications have been made possible which require immediate accessibility to the computer. Examples of such new areas include automated libraries, ticket reservation systems, computer assisted instruction, computer graphics, and others that involve man-machine interaction in an essential way.

1.1.2 Problems with Existing Systems

To date, general purpose remote access systems have been based primarily on the concept of time sharing. There has been widespread discussion in the professional and technical publications concerning the meaning, uses, and problems of time sharing.(2) Time sharing systems require each user's program to be loaded and processed for a small segment of time, then dumped out, and another user's program loaded, again to be dumped after a short segment. Much of the literature on time sharing has been devoted to determining the proper "tuning" of the process (segment length and loading priorities), in attempts to reduce the overhead of the load and dump swapping process.

Perhaps the best documented example of tuning a time sharing system is given by Schwartz and Weissman.(3) They were able to determine that 50% of the programs run on their system required 0.6 seconds or less, and that 85% were completed in 1.8 seconds or less. Consequently, they

(2) For an extensive discussion of time sharing plus a more detailed analysis of time shared systems, see Thomas N. Pike, Jr., Time-Shared Computer Systems, in <u>Advances in Computers</u>, Vol. 8, New York: Academic Press, Inc., 1967.

(3) Jules I. Schwartz and Clark Weissman, The SDC time-sharing system revisited, <u>Proc. ACM National Conference</u>, Washington: Thompson Book Co., 1967.

established a primary time segment length of 0.6 seconds, and a priority system that called for any program which had used less than three segments to remain in the highest priority level. They then established two other levels of priority for intermediate and long jobs.

The efforts to reduce the cost of swapping in time sharing systems have centered around two areas, 1) reducing the swapping time by improving the swapping mechanism,(4) and 2) reducing the effects of swapping through multiprogramming.(5) The first of these is frequently prohibitively expensive, although it is hoped that future technology will reduce such costs. Multiprogramming has similar high memory cost implications. In particular, by segmenting the main memory to allow several programs to be resident at the same time, each program is then restricted to a small portion of the available storage.

To combat these problems, the concepts of paging and virtual memory have been introduced.(6) It has been shown, however, primarily by Nielson(7), that paging leads to an inordinately high percentage of time being spent on system overhead. Nielson showed that, after an extensive series of tests involving a variety of hardware configurations, tuning the system, and improvements in the paging algorithm, the best results that could be expected for the IBM 360/67 was 67% utilization for execution

(4) See, for example, Kurt Fuchel and Sidney Heller, Consideration in the Design of a Multiple Computer System with Extended Core Storage, Preprint of ACM Symposium on Operating System Principles, New York: Assoc. for Computing Machinery, 1967.

(5) J. B. Dennis, Segmentation and the design of multiprogrammed computer systems, J. ACM, 12:4, 1965.

(6) V. A. Vyssotsky, F. J. Corbato, and R. M. Graham, Structure of the Multics Supervisor, Proc. AFIPS Fall Joint Computer Conf., New York: Spartan Books, 1965.

(7) Norman R. Nielson, The simulation of time sharing systems, Comm. ACM, 10:7, 1965.

and 33% for overhead and idle time. Without the detailed study and careful optimization which led to this 67% execution, Nielson found that a great many other, seemingly reasonable, configurations yielded as little as 5% execution time and 95% overhead and idle time.

Another approach to the solution of the swapping problem has been to limit the scope of the system by restricting the resources available to the user to those which can reside permanently in main memory. The advantage of this is that only the actual users' programs, which are presumably small, need to be swapped, while the compilers, loaders, etc. simply reside in core. The basic difficulty with this approach is that there are usually only one or two programming languages available. Similarly the size of the program is restricted to the small portion of core not taken up by the operating system and compilers. An example of this kind of system is the QUICKTRAN system of IBM.(8)

1.1.3 Objectives of the Research Reported Here

To solve these problems, a hierarchical system architecture is considered with particular emphasis on the contributions of this approach towards reducing system overhead. However this approach must be considered within the context of providing versatility of types of usage.

The dual emphasis is to attain an effective system with wide applicability. The system must have a large number of available programming languages, as well as easy expansion to include new subsystems as desired. The system must also be free from restrictions on the nature of programs which can be executed within the system. The user would have access to all the capabilities of the computing complex.

(8) IBM 7040/7044 Remote Computing System, IBM System Reference Library No. 7040-25, Form C28-6800.

1.2 The Methodology Taken to Secure the Objective

The effectiveness of a processor can be improved if it is designed for a special class of problems. The system approach taken here consists of utilizing a number of processors in a computer network or in a computer complex, where each of the processors is designed to handle a special class of problems. The problems that are submitted to the total system are classified and accordingly dynamically routed to the processor designed to handle the respective class of problems most effectively. The system design consists of establishing a classification for user problems and definition of respective processors. In fact, a hierarchy of problems as well as processors based on "complexity" is suggested. When a problem is recognized by one of the processors in the hierarchy to be more "complex", it is then passed to the next higher processor in the hierarchy designed to serve the next level of "complex" problems. Thus, two hierarchies, that of problems and that of processors, are suggested.

Four parameters are suggested that jointly establish the class of "complexity" of a problem. These are 1) the allowed delay of interactive response, 2) the frequency of a problem type being submitted to the system, 3) the requirement for main memory storage, and 4) the requirement for execution time. Table 1 illustrates these parameters for various tasks.

This classification is based on the observation that the most frequent tasks and those requiring the most interactive response generally require little high speed memory storage and short execution times. Conversely, the problems requiring large high speed storage capacity and lengthy execution times are relatively infrequent and the interactive response is not as essential.

Table 1 Example of Typical Task Requirements in MSPSF

Task	Interactive Requirements (max. wait in sec.)	Frequency of use	High Speed Storage Required (words)	Execution Time (sec.)	Processor Level
Input/Output each character	0.1	10/sec	100(PDP-8)	100 μsec.	1
Input for a line	1	3/sec	1000(PDP-8) 200 (7040)	.3	1
Input Editing	5	1/10 sec	800 (7040)	5	1
Output Examination	1	3/sec	1000(PDP-8) 200 (7040)	.2	1
Storage and Retrieval	15	1/5min	12K (7040)	.5	2
Statistics of Retrieved Data	2 min	1/10min	2K (7040)	.5 min	2
Program Assembly	3 min	1/10min	24K (7040)	1 min	3
Program Compilation (FORTRAN)	3 min	1/10min	32K (7040)	2 min	3
Program Execution (testing and debugging)	3 min	1/10min	8K (7040)	1 min	3

An example of a three level hierarchy corresponding to Table 1 is indicated in Fig. 1. The first processor in the hierarchy handles those programs which have minimal memory requirements and length of execution with maximum frequency and response requirements. These are the routines directly related to terminal control and input/output.

The middle level processor in the hierarchy handles those programs with more extensive, but still limited, memory and time requirements. Notable among these is the information storage and retrieval system, plus a library of executable programs maintained in the storage and retrieval system.

The highest level in the hierarchy is intended for programs requiring greater resources of the computing system. These programs include compilers, assemblers, and other systems programs which require extensive high speed memory storage capacity, as well as user programs which cannot be fitted into the more restricted middle level.

The processors in the hierarchy may be realized each as a separate computer unit or as programming subsystems within one equipment unit. Combinations of software-hardware may represent each processor.

1.3 Development of the System Architecture Concept Through Design, Implementation and Experimentation

The hierarchy of processors system concept described previously can be realized in numerous ways. The exploration of this concept is carried out independently of the specific software-hardware combinations that may be selected. The approach that has been developed may later be translated into a desired configuration based on the local requirements and equipment. Therefore, the exploration of the system concept has been carried out through modification of the Moore School Problem Solving

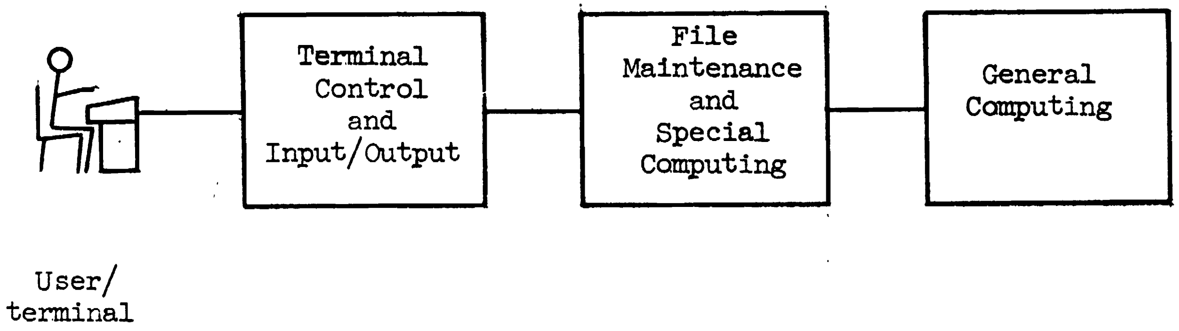

Terminal Control and Input/Output
File Maintenance and Special Computing
General Computing
User/ terminal

Figure 1 System Function Partitioning

Facility (MSPSF) to operate in the conceived manner. The advantages of this approach were the existing hardware and software subsystems which were used as building blocks to obtain the hierarchical processor system.

It needs to be emphasized here that the shortcomings found in MSPSF contributed as well to generating the final system concept discussed in the Conclusion section. MSPSF does not include the system architecture presented in the Conclusions. Rather, MSPSF employs some of the basic design tenets, thus serving as a vehicle for the investigation which was conducted.

The primary objective, to provide as broad an applicability as possible, has been achieved in the MSPSF by modifying the operating system IBSYS of the IBM-7040. The consequence is that all of the facilities of IBSYS are available to the remote user. This includes a wide variety of programming languages, a subroutine library, and a number of special purpose subsystems (see Table 2).

This report, therefore, includes the description of the Moore School Problem Solving Facility that has been implemented and an evaluation of the utilization of the system by a community of students and faculty.

Implementation of MSPSF has been greatly simplified by the hierarchical nature of the system. The interfaces between the processors are controlled by languages. Interpretation of calls on macros is provided in some of these languages. The specification of these macros provides for the convenient definition of new subsystems.

Table 2 Subsystems of IBSYS

Programming languages

- FORTRAN
- COBOL
- MAP
- MAD*
- WATFOR*
- LISP*
- IPL-V*
- SNOBOL*
- L6*
- ALGOL*

Subroutine library of over 500 subroutines

Loader

Post mortem dump system including snapshots

Sort system

UPDATE

Utility package

System editor

Automated accounting system*

Job controller for automatic sequencing of compilation, assembly, load, execution, and dump

* These items are local to the University of Pennsylvania

Many of the software components used in the Moore School Problem Solving Facility have been adapted from two previously existing systems. First, the remote terminal input/output system and the information storage and retrieval system (see Fig. 1) constituted an earlier version of MSPSF. These subsystems were useful in providing the present design with a basic remote access capability and a file maintenance system upon which to build, and relieved the necessity for duplicating existing subsystems.

The present MSPSF implementation consists mostly in tying existing components together, supplying the necessary links between them, and making modifications to each to account for the new environment in which they are to operate.

Five outstanding questions concerning the hierarchical approach to system architecture were to be clarified through the implementation and use of the system.

(1) What is the nature of each processor, including specific reference to such questions as to how many processors there should be, whether they should be special purpose or general purpose, and how tasks should be divided between the processors?

(2) What should be the nature of interfacing languages between the processors, and how do users use such languages?

(3) Are there any obstacles to wide applicability?

(4) What system software resources need to be supplied, and how should they be distributed among the various processors?

(5) What is the relationship between the file management facility and the rest of the system?

A block diagram of the hardware components is given in Fig. 2. The primary component is the IBM-7040 computer, together with a card reader/punch, a line printer, twelve tape drives, and a large 1301-II disk file. Adjacent to the IBM-7040 and attached to it is a D.E.C. PDP-8 computer. The PDP-8 has two remote terminal interface units, one with four Teletype ports and one high speed (2400 baud) Dataphone port. At the end of the high speed line are two devices, a D.E.C.-338 display terminal and a Sanders 720 control unit with two alphanumeric keyboard displays.

1.4 The Evolution of MSPSF

Early experiments in man-machine-memory organization(10) resulted in the mechanization of an information retrieval system at the Naval Aviation Supply Office in Philadelphia.(11) This system was one of the pioneering efforts in remote accessed data bases. The early equipment in 1961 consisted of an IBM 1401 computer with a 1407 console, 1405 disk, a line printer and magnetic tape units. In addition, an Information Products Corporation 1501 display station was incorporated into the system for remote inquiry. This system was transferred in 1962 to an IBM 1440, with 2 IBM 1301 disc units and 12 typewriter terminals.

These early results showed the feasibility of using such a system as a basis for a Problem Solving Facility for general use by scientists and engineers. Such a system was gradually developed at the Moore

(10) The Multi-List System, Technical Report No. 1, Vols. I and II, University of Pennsylvania, Moore School of Electrical Engineering, M. S. Report 62-10, 1961.

(11) Barry Zimmerman, David Lefkovitz, and Noah S. Prywes, The Naval Aviation Supply Office Inventory Retrieval System - A Case Study in File Automation, Management Science, 10:3, 1964.

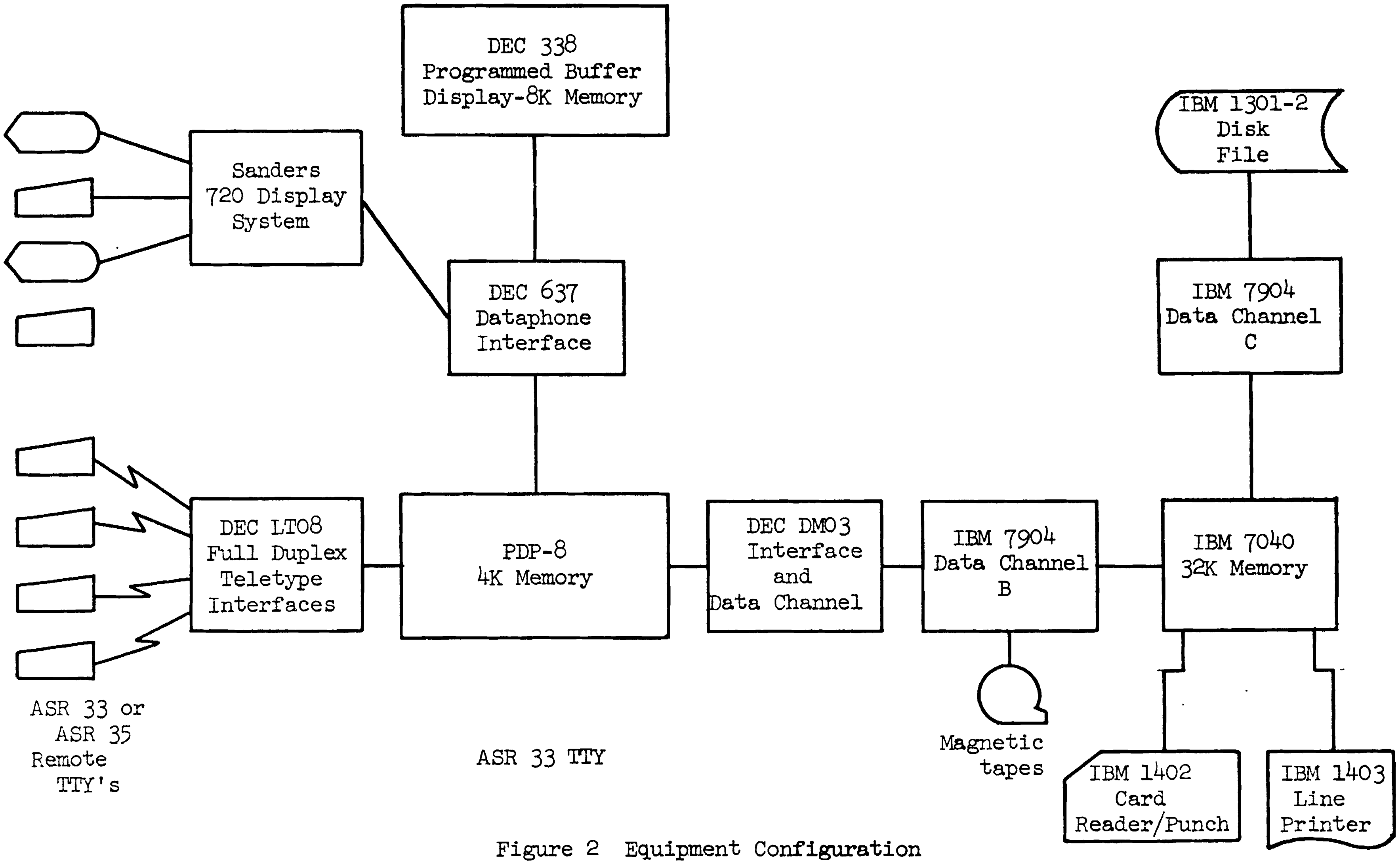

DEC 338 Programmed Buffer Display-8K Memory
Sanders 720 Display System
DEC 637 Dataphone Interface
IBM 1301-2 Disk File
IBM 7904 Data Channel C
DEC LT08 Full Duplex Teletype Interfaces
PDP-8 4K Memory
DEC DM03 Interface and Data Channel
IBM 7904 Data Channel B
IBM 7040 32K Memory
ASR 33 or ASR 35 Remote TTY's
ASR 33 TTY
Magnetic tapes
IBM 1402 Card Reader/Punch
IBM 1403 Line Printer

Figure 2 Equipment Configuration

School, University of Pennsylvania. Wexelblat(12) provided the implementation of a language processor, MULTILANG, and executive programs. The storage and retrieval system was implemented by Freedman.(13) A satellite PDP-8 processor was interfaced for servicing remote terminal facilities.(14) These are the three components that were mentioned in the previous section used in the construction of the presently described system. The information storage and retrieval system has been strengthened by the addition of a privacy and security protection mechanism by Hsiao.(15) This system allows each user to store records in files, with the added facility of being able to selectively allow access to the files by others. The input/output system has been expanded to accommodate communication for computer graphics experiments which use the DEC-338 processor as a highly sophisticated graphical terminal for the MSPSF.(16)

The users of MSPSF are constantly adding to its capabilities, and new and interesting experiments are using MSPSF in fresh ways. Notable among these are experiments with list-oriented programming languages,(17)

(12) R. Wexelblat, The Development and Mechanization of a Problem Solving Facility, University of Pennsylvania, Moore School of Electrical Engineering, Ph.D. Dissertation, 1965.

(13) H. Freedman, A Storage and Retrieval System for Real-Time Problem Solving, University of Pennsylvania, Moore School of Electrical Engineering, M. S. Report 66-05, 1965.

(14) R. P. Morton and M. S. Wolfberg, The Input/Output and Control System of the Moore School Problem Solving Facility, University of Pennsylvania, M. S. Report 67-30, 1967.

(15) David K. Hsiao, A File System for a Problem Solving Facility, University of Pennsylvania, Moore School of Electrical Engineering, Ph.D. Dissertation, 1968.

(16) Work in progress by Michael S. Wolfberg.

(17) Charles A. Kapps, SPRINT: A direct approach to list processing languages, Proc. AFIPS Spring Joint Computer Conf. New York: Spartan Books, Inc., 1967.

another expandable concept in operating systems,[18] a game playing system involving two players at different terminals who interact through the common data base,[19] and an online hospital research retrieval package.[20]

MSPSF was in operation for four hours per day from March through June, 1968, and continues on a more restricted basis at this writing. During this time as many as four remote users have shared the system with background batch work.

1.5 Outline of Report

This report is divided into five chapters (the first being this introduction) and three appendices. Chapter 2 describes MSPSF organization and use. An idealized model of a hierarchical processor system is presented first. The restrictions which have led to the present MSPSF model are then pointed out. The resulting system components are subsequently described, with particular attention paid to how these components are interfaced.

Chapter 3 presents a detailed description of an interfacing IXSYS language for defining new subsystems. The syntax of the language is defined using a Backus Normal Form specification. Each command in the language is discussed in detail, and a complete discussion of macros

(18) T. J. Ostrand, An Expanding Computer Operating System, University of Pennsylvania, Moore School of Electrical Engineering, M. S. Report 67-16, 1967.

(19) Philip Bursky, et al., A Man/Machine Competitive Game - A Naval Duel, University of Pennsylvania, Moore School of Electrical Engineering, M. S. Report 68-34, 1968.

(20) Marvin Gelblat, Internal communication, University of Pennsylvania, Moore School of Electrical Engineering.

and related parameters is also given. Lastly, some examples are presented and discussed.

Chapter 4 presents an evaluation of MSPSF from the viewpoint of the several users. Many useful and constructive comments are included and related to the needs of the different types of users, depending on levels of sophistication and requirements for system resources.

The Conclusion, Chapter 5, summarizes the characteristics of the Hierarchical Processors system architecture based on the experience with MSPSF.

2. DESCRIPTION OF THE MOORE SCHOOL PROBLEM SOLVING FACILITY

A hierarchical structure such as illustrated in Fig. 1 has been adopted for MSPSF as shown in Fig. 3. An additional level in the hierarchy, scheduling, has been added to the structure to meet the accounting requirements of the University Computer Center.

The respective processors are described below both functionally and operationally. The use of the system is traced through the different control languages illustrating the roles of the various processors and how information is passed between them. The three levels in the hierarchy have control languages designed for the particular problems encountered at that level and for interfacing with the next higher level in the hierarchy. These are:

1) The input/output processor language provides aids to the user for the preparation of his input and the examination of his output. This language also serves as the control language for the interface with the job scheduling processor.

2) The language of the special computing processor MULTILANG is designed to make easy communication with the storage and retrieval facilities available on that level.

3) The control language of the general computing processor is the language of the operating system (IBSYS), designed to facilitate the use of the subprocessors and input/output devices.

4) An additional language IXSYS provides an interface between the storage and retrieval facilities and the general computing processor. The IXSYS language will be described in detail in Chapter 3.

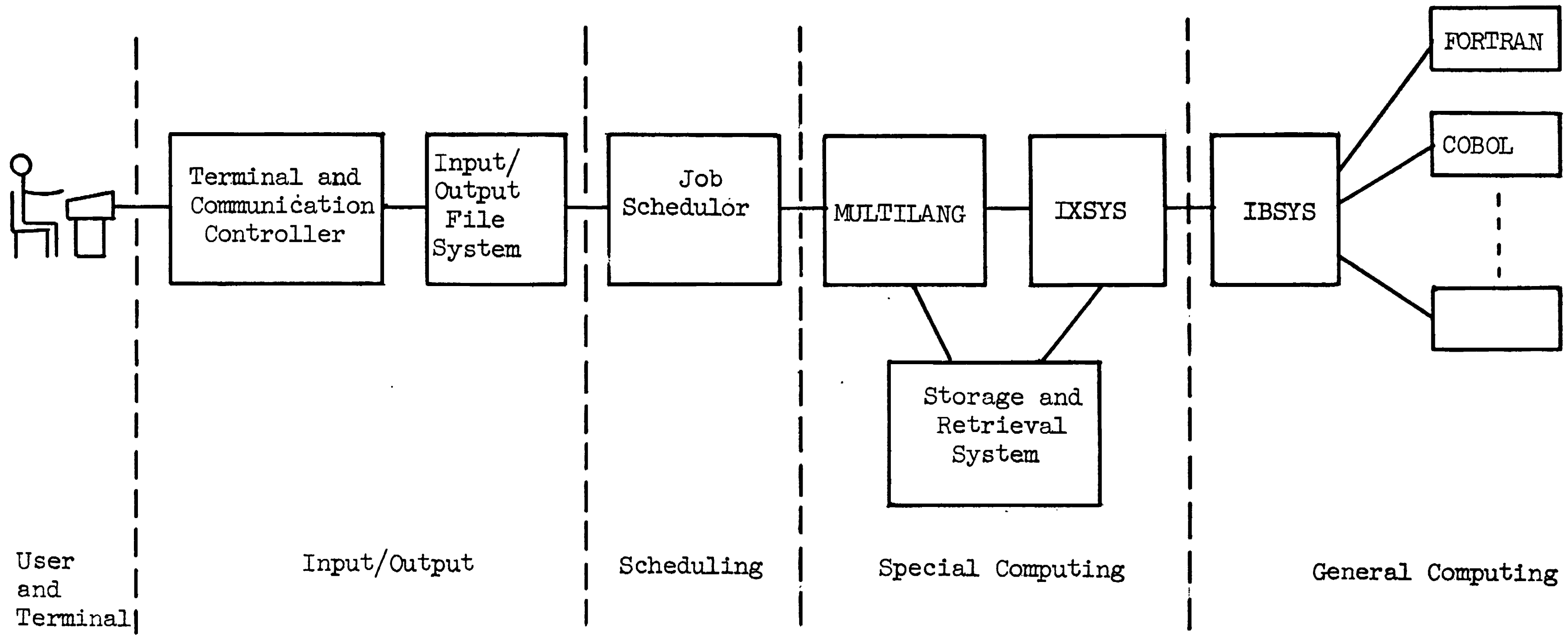


Figure 3 Major System Functions and Component Processors

2.1 <u>The Input/Output Processor</u>

The input/output processor is divided into two parts as shown in Fig. 3. The terminal and communication controller is device oriented. The input/output file system provides the users with editing facilities of temporary storage for input and output.

2.1.1 The terminal and communication controller

The terminal and communication controller receives input from the terminals and transmits it to I/O File System. It also sends output from output files to the terminals. To aid in this process the controller recognizes and interprets special characters such as tab and carriage return on input and carriage control characters on output. It provides for the specification of tab settings. For terminals, such as Teletypes, which contain no buffering facilities, the controller provides a one line buffer, as well as such local editing functions such as backspace (represented by $\longleftarrow$) and line erase (CTRL C).

In addition to these services, the communication controller also performs services of which the user is unaware. Foremost among these is code conversion. As shown in Fig. 2, there are four types of terminals in MSPSF: an on-line Teletype, three remote Teletypes, two alphanumeric display stations, and the DEC-338 display. Each type of terminal uses some specific character code which it transmits to the communication controller. This code must then be translated into the code of the rest of the system. Similarly, output characters must be in the code of the device to which they are sent. There are also transmission controls, such as synchronization and error checking, which must be provided by the communication controller.

The communication controller must handle several remote terminals simultaneously. Consequently, this component must be implemented in an essentially time-shared way, eliminating possibilities for missing characters from a terminal. In MSPSF this implementation takes the form of a program called PSF, which occupies almost half of the 8192 word memory of the PDP-8.

2.1.2 Input-Output files

Each remote terminal is provided with one input file where user prepared input is placed temporarily and one output file where the output of processing by the computing processors is placed. Commands to the input-output file system allow the user to add to the input file and edit it, examine or clear any file, and ask for length of the files.

An important capability of the input-output file processor is the handling of a variety of terminal types and applications. The viewing of output by users is non-destructive and the information remains in the files. For instance, an information retrieval user can scan quickly through an output to examine items relevant to his interest and later view again those items considered only of secondary relevance. Another example is where a user writing a program examines first error messages, looks at the results of the program, and then, if necessary, goes back to the program listing to find errors.

This non-destructive output is of special importance to a programmable terminal such as the DEC-338. Frequently such devices have limited storage capacity and a multiplicity of users. As each user comes to the terminal he may call for his programs to be retrieved and left on his output file. He may then use them as frequently as needed without retrieving them again. A system to allow loading DEC-338 programs from the output file is currently under development for MSPSF.

The input/output file system is implemented as an integral part of the nucleus of the IBM-7040 operating system. It occupies about 1500 words of core storage on a permanent basis. Some of this storage is memory protected and those areas that are not protected are reloaded wherever necessary.

2.1.3 The input/output processor language

The user of MSPSF is provided with a set of commands which are recognized by the input/output file system. These commands allow the user to prepare input, examine output, and control the execution of his jobs. The commands are summarized in Table 3.(21)

The terminal control language reflects the specific terminals used in MSPSF. The differences between single line Teletype terminals and multiple line display terminals require different facilities for input preparation and output examination. Some of the commands are specifically designed for one class of terminals. For example, the SEE command will fetch up to 10 lines at a time for displays but only one line for Teletypes. Thus, its use is limited almost exclusively to displays. Conversely, the PRINT command prints at the full speed of the Teletype. Users of displays which have associated typing devices, e.g., the DEC-338, can skim through input or output faster with the display than with the printer. Consequently, these users limit the use of the PRINT command to those portions for which a permanent copy is desired.

(21) A more complete description of these commands can be found in Morton and Wolfberg, op. cit.

Table 3 Summary of the Input/Output Processor Language

a. Data input commands

Command	Meaning
APPEND	Add input to the end of the input file.
INSERT n	Insert input in the middle of the input file ahead of line n
DELETE n m	Delete lines n through m from the input file.
EDIT n m	Replace lines n through m of the input file with new data.

b. Data examination commands

Command	Meaning
SEE f n	Display (or print) line n from file f. For display terminals lines n through line n + 9 are displayed. File may be either input or output.
ROLL.	Like SEE, but continue to next line or group of lines. Parameters f and n may be supplied to skip n line on file f.
PRINT f	Print file f. Parameters may also be supplied which specified either starting line or both starting and ending lines to be printed.

c. Control commands

Command	Meaning
SIGNIN	This must be the first command given.
CLEAR f	Clear file f.
START	Place this console on the queue for jobs to be scheduled and run.

Table 3 Summary of the Input/Output Processor Language (cont.)

c. Control commands (continued)

STOP	Terminate the job for this console if it is running immediately. If a job for this console is on the waiting queue, take it off the queue.
SIGNOUT	Clear input and output, stop a job if one was started, and the next command from this console must be SIGNIN.

d. Teletype oriented commands

Command	Meaning
FULL	This is a full duplex Teletype. This means all input must be "echoed" back to the Teletype to be printed.
HALF	This is a half duplex Teletype. Do not echo input characters.

e. Display oriented commands

Command	Meaning
LOWER n m	Lower line n by m lines. Line n refers to lines 1 - 10 on the screen.
INTERCHANGE n m	Interchange lines n and m on the screen.
TABKEY x	Character x is to be treated as a tab character.
TAB n	Set a tab stop at character position n.
NOTAB n	Clear the tab stop at character position n. If n is omitted all tab stops are cleared.
COPY	Copy the data lines on the screen onto the teleprinted associated with this display.
FROM	Same as COPY but issue a form feed first.
CEASE	Terminate a previous PRINT request.

Many commands, although not designed for a specific terminal type, have different formats for different terminals. Single line devices such as Teletypes require modal operation for input. Thus, a user types APPEND, INSERT, or EDIT and what follows is considered data until another command is typed. From displays, however, the command is transmitted at the same time that the data is. If more data is to be transmitted than can be placed on the screen at one time, an additional occurrence of the command must accompany each data transmission.

The command syntax is likewise different for displays and Teletypes. For displays, the screen format is as follows: the first ten lines are the user's data, the eleventh line is for messages from the input/output system to the user, and the twelfth line is for commands. From Teletypes, a command line is terminated by an ALT MODE or ESC character, while data lines are terminated by a carriage return.

Some examples of the use of the input/output processor language are given below in Sect. 2.5, and particularly in Fig. 7 of Sect. 2.5.2.

2.2 Job Scheduling Processor

When a remote terminal user has completed preparing input (stored in the input file) he may direct the system (through the START command, see Table 2) to perform the requested processing. The input is then communicated by the Input/Output processor to the Job Scheduling Processor. The terminal number for that user is placed in the queue of waiting terminals. At the present only one job may be executed at a time by the computing processors. Whenever one job ends, the next job on the queue is run. If all remote terminal jobs have been processed, then background batch jobs will be run.

A user may change his mind after placing a job in the queue and direct the job scheduler to remove it (STOP). Similarly once a job has started to be run by the computing processors, the user may stop it (STOP), for instance, when he finds from the output that it is running incorrectly.

The job scheduler is implemented as an addition to the supervisor IBSYS of the operating system of the IBM 7040. The supervisor calls the job scheduler whenever the start of a new job is detected. If a remote job is waiting in the queue, the job scheduler generates the necessary control card images to cause the special computing processor to be loaded by the supervisor. When no more remote jobs are left in the queue, the supervisor runs the next job from the background batch. If no background jobs are waiting then a remote job will be run as soon as the START command is received.

The job scheduler requires about 250 locations but since the supervisor is not resident, this does not detract from the available space for the user. When no remote jobs are to be scheduled, the job scheduler requires about 500 μs overhead to the supervisor's task. This is considered negligible since the normal supervisor operation requires either one or ten seconds, depending on whether output is on tape or line printer. When a remote job is scheduled, the supervisor's operation is slightly faster since the control cards are generated in memory instead of read from tape.

2.3 The Special Computing Processor

There are three major components which make up the special computing processor of MSPSF: the MULTILANG interpreter, the storage and retrieval system and IXSYS. In addition, there is a growing number of routines, called worker programs, which are prepared by the user of MSPSF.

IXSYS is one of these worker programs. The language used to communicate with these components is called MULTILANG. A brief description of this language[22] and the roles of the several components of the special computing processor follows.

Allocation of the 32K memory of the IBM 7040 for the special computing processor is approximately as follows: 6700 for operating system nucleus including input/output routines; the MULTILANG interpreter occupies 8200; the storage and retrieval system 5100 ; leaving 12K for worker programs, input/output buffers, and available space for retrieved data.

2.3.1 The MULTILANG Language

The basic element of MULTILANG is the <u>key</u> or descriptor. Keys may be combined using the logical connectives and the arithmetic connective "TO" to form <u>descriptions</u>. A <u>statement</u> is a sequence of descriptions separated by slashes (/). Statements may be grouped into a <u>procedure</u>, in which case the statements may be labeled. Procedures may also contain additional simplifying aids, such as "local names" which stand for frequently used complex descriptions. Statement labels may be used in place of descriptions in a statement to specify variable exits. Other types of operands may be element numbers (described below) and data in the form of numbers or character strings.

MULTILANG procedures are executed interpretively, one statement at a time. As each statement is read from the input file, the first description is taken to be that of a worker program to be executed. The

[22] A more thorough description of MULTILANG is given in Wexelblat, op. cit.

additional descriptions are treated as parameters for the worker program. For example, consider the following MULTILANG statement:

RETRIEVE/RPM&ECM&121866

Here RETRIEVE is the description of a worker program which calls on the storage and retrieval system to retrieve data matching the description given in the parameter and then prints them out. In this case such data items would have to contain the three keys RPM, ECM, and 121866.

The basic unit of data retrieved by this statement is a record. A subcomponent of a record is an element. A description may refer to a specific element of the retrieved records. A set of records may be organized into a file. A detailed description of the file system is given in Hsiao.(23)

2.3.2 The MULTILANG Interpreter

The MULTILANG interpreter is loaded automatically by the job scheduling processor when a job is scheduled for a remote user. The interpreter thus reads the user's input file and interprets MULTILANG statements as described in the previous section. After each statement has been executed control returns to the interpreter to execute the next statement in a procedure or to read the next statement from the input. In this way, the MULTILANG interpreter serves as the supervisor for the special computing processor.

2.3.3 The storage and retrieval system

The storage and retrieval system is part of the special computing processor. It manages the data(24) file and performs the information

(23) Op. cit.

(24) Unless explicitly stated, references to data in the data file may include programs.

storage and retrieval functions.

The tasks related to data storage involve storage allocation on the 1301 disk and updating the key directories. This latter task also includes adding pointers to the record to be stored linking its several keys with other records containing the same keys. Thus a list structure of the records is maintained.

The tasks related to data retrieval are as follows. First, a MULTILANG description is converted into a search strategy. This strategy uses the first key mentioned as the key whose list is to be searched. This key is called the primary key. In the case of a disjunction of keys, after the first primary key list is exhausted another primary key is taken.

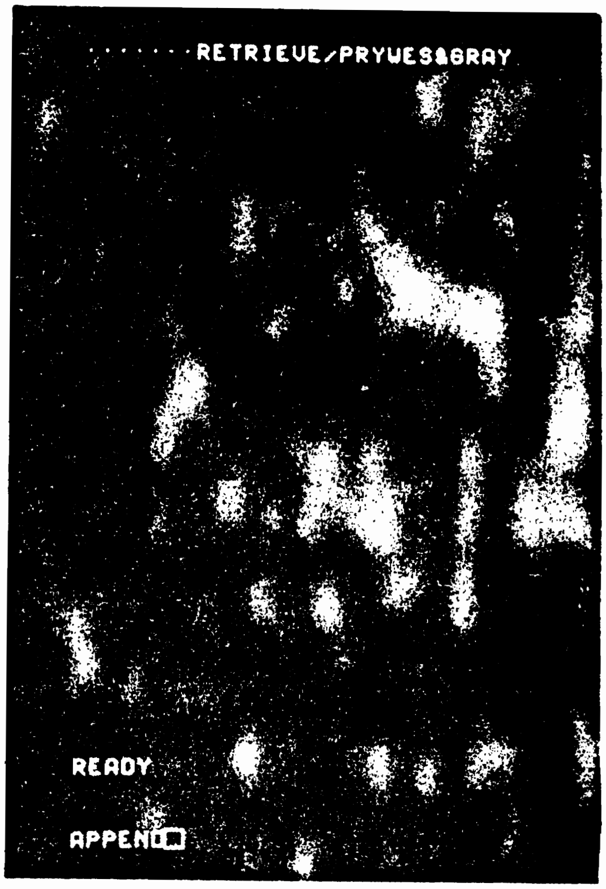
Next, all records on the primary key list are retrieved from the disk and checked to make sure that they contain all other conditions called for in the description.

For example, consider the statement used earlier in which the description of the data to be retrieved was

RPM&ECM&121866

In this case RPM is the primary key. All items containing the key RPM will be examined, and any which also contain the keys ECM and 121866 will be passed on to the requesting worker program as satisfying the given description.

An example of the input and output involving the use of MULTILANG language and the storage and retrieval system is given in Figs. 4 and 5. These figures are photographs of the DEC-338 as it is used as a terminal to MSPSF.

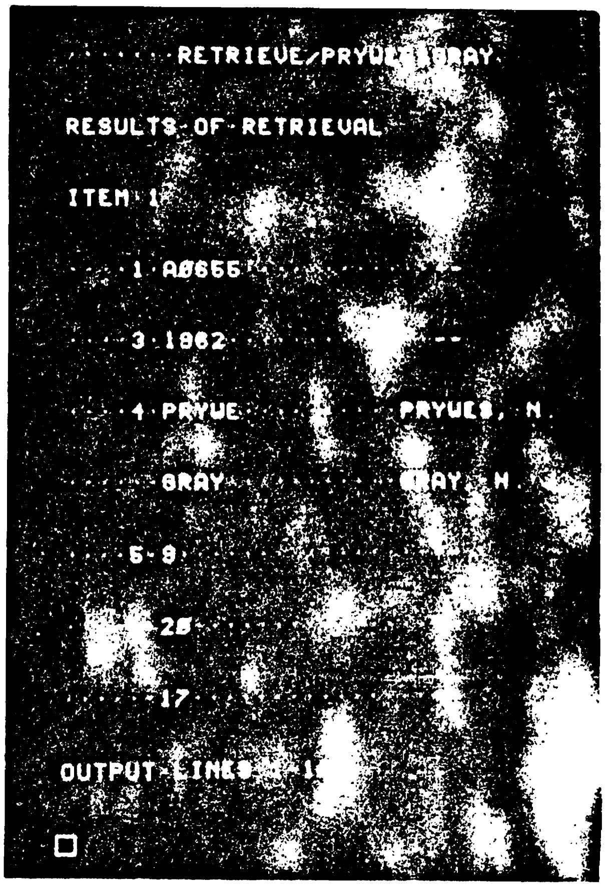


·······RETRIEVE/PRYWES&GRAY

READY

APPEND

Figure 4 Display With Input for Retrieval Request



```
······RETRIEVE/PRYU[illegible]RAY·

RESULTS·OF·RETRIEVAL·

ITEM·1

····1·A0665·········

····3·1962···········

····4·PRYUE·········PRYUES,·N.

······GRAY···········[illegible]AY,·H.

····5·9·

······25······

······17······

OUTPUT·LIN[illegible]·1

□
```

Figure 5 Display with First Ten Lines of **Output**
from Retrieval Request

2.3.4 The IXSYS worker program

The IXSYS worker program provides the interface between the special and general computing processors. In this capacity IXSYS has three major functions: preprocessing the input for the general computing processor, postprocessing the output from the general processor, and interpreting the IXSYS language.

Input preprocessing is required for three reasons. First, some additional control statements (besides those supplied by the user) must be generated to allow output postprocessing. Second, the input statements, on the input file are not in the proper format to be handled by the general computing processor. Third, the input must be scanned for interpretation of the statements in the IXSYS language.

Output postprocessing is necessary for two reasons. Binary output which is normally punched is instead stored in the data files for use at a later time. Alphanumeric output, which is normally printed, is diverted to the user's output file. As with the input, there is a format difference requiring an additional conversion step.

The IXSYS language is described in detail in Chapter 3. Briefly, it allows the inclusion of input to the general computing processor of source or binary information previously stored in the data file. This means that not all input need actually be on the user's input file. For binary decks this is essential since there is no way to get them on an input file. In addition the IXSYS language includes a macro facility for defining and using frequently required input procedures.

2.3.5 Some additional worker programs

A list of commonly used worker programs is given in Table 4. This list is not intended to be exhaustive, but merely to point out the

Table 4 Some Useful Worker Programs

Name	Action
ADAKY	Add a key to all items matching the given description.
DELKY	Delete a key from items matching the given description.
COUNT	Count the items matching the given description.
RETRIEVE	Output all items matching the given description.
DELETE	Delete all items matching the given description.
MODEL	Modify (in some specified way) a specified element of all items matching the given description.
STORE	Make an item out of the remainder of the input file and store it under the given keys.
RESTORE	Retrieve an item matching the given description which was previously saved by STORE and restore it to the input file.
PUNCH	Punch an item previously saved by STORE. Sequence punching may be indicated.
ADDMP	A binary deck on the specified device is to be made into an item and saved as a worker program.
SIGNIN	Provide name and project number.

kinds of data manipulation required by all users, regardless of the nature of their particular data. For example, the pair of programs STORE and RESTORE are used to save a user's input file and retrieve it so that he may continue his work at a later time without having to rekey it.

2.4 The General Computing Processor

The general computing processor is constituted around a modified IBSYS operating system for the IBM 7040 computer. This operating system can be divided into two classes of programs, the supervisor and the subsystems. The supervisor of IBSYS performs such functions as accounting, peripheral unit assignments, and input/output device dependent services.

The subsystems of IBSYS include a wide variety of programming languages -- FORTRAN, COBOL, ALGOL, LISP, MAD, IPL-V, SNOBOL, WATFOR, and L6 -- a macro assembler and a loader. Several programming and debugging aids are also included, such as, a subroutine library of frequently used programs and a post mortem dump program. Additional subsystems are an Update program for maintaining program decks on magnetic tape, a utility package for duplicating and dumping tapes, and a generalized sort monitor. The special computing processor also appears to the operating system as a subsystem, although remote users need not be concerned with this.

Most of these subsystems either occupy or make use of the entire 32K core memory of the IBM 7040. Some, however, were written for a 16K machine as well, and consequently, require many more overlays than might otherwise be required.

2.5 Examples of Use

A scenario consisting of a solution to a problem is presented below. It demonstrates the various aspects of the use of MSPSF. The scenario consists of first, the terminal being initialized. Then the input is prepared, in this case, consisting of a simple FORTRAN program. After a compilation, some errors are corrected. The data is then added by retrieving it from the storage and retrieval file. The subsequent run provides the results needed. Lastly the program is saved to be used again at a later time.

The example presented here was run from a teletype terminal in order to record all activity. In the future, the input typed by the user has been underlined to distinguish it from the computer output.

2.5.1 Initializing the use of a terminal

Initialization of a terminal of MSPSF is accomplished in two steps (see Fig. 6). First, the input/output processor must be initialized for each user. This is accomplished by typing the two commands SIGNIN and FULL. Prior to the SIGNIN command, the response is CONSOLE AVAILABLE. Full duplex operation requires the terminal controller to echo each character as it is typed, thus providing immediate acknowledgement of transmission line errors.

The next step is to initialize the job scheduling system by executing the worker program SIGNIN with two parameters, the user's name and his Computer Center project number. The SIGNIN program is allowed to run without charge to the user, but subsequent jobs are charged to the user's project number.

```
?   CONSOLE AVAILABLE
SIGNIN?
FULL?
A?   NO INPUT LINES
  SIGNIN/*MORTON*/*ØØ1613*
START?
P O?
          SIGNIN/*MORTON*/*ØØ1613*
```

Figure 6 Initializing the Terminal

2.5.2 Preparing Input

Figure 7 illustrates input preparation. First, the input and output files are cleared and the APPEND mode is entered. The response from the APPEND command to the I/O processor informs the user of the number of existing lines on the input file so he will know where the new input starts. In this case there was no previous input since the file was just cleared.

Next, the input is typed, making use of tab settings (initially set at every eight character positions). After completing the input, it is printed for examination with all the column settings shown. Here it can be seen that the use of tabs has resulted in the correct spacings.

2.5.3 Debugging and running

Once the program input has been completed, the START command is given followed by the PRINT command followed by the output printed. Figure 8 shows the compiler output and the indicated error. In Fig. 9 the error is corrected and some sample data is appended to test the algorithm. The job is then rerun and the results are shown in Fig. 10. The numbers are verified by a hand calculation. Format statement 200 has some errors. After correcting the format, the program is ready for a production run.

```
                 Indicates terminal character for command line
C I@             Clear the input file          } Input/Output processor
C O@             Clear the output file         }     language
A@ NO INPUT LINES  Enter APPEND Mode           }
 IXSYS           Call IXSYS worker program. This is a MULTILANG Statement.
$IBJOB          }
$IBFTC          } IBSYS control cards
 N=0             Indicates a TAB was typed.
 ASQ=0.
 ASUM=0.
   20 REAS (5,100) A
  100 FORMAT (F8.4)
 IF (A.LE.0.)GOTO 10
 ASUM=A+ASUM
 ASQ=A*ASQ@      Indicates line erase
 ASQ=A*A+ASQ
 N=N;←+1         Indicates character erase                         The
 GOTO 20                                                           Fortran
   10 AN=N                                                         Input
 AMEAN=ASUM/AN
 SD=SQRT(ASQ/AN-(AMEAN**2))
 WRITE (6,200) N,AMEAN,SD
  200 FORMAT 1←(1X,I3,' VALUES',/1X,'MEAN = ',F8.4,/1X,
     1 'STANDARD DEVIATION = ',F8.4)
 STOP
 END
$ENTRY               An IBSYS control card
P I@                 Print the input file
        IXSYS
$IBJOB               Note TAB expanded
$IBFTC
        N=0
        ASQ=0.
        ASUM=0.
   20   REAS (5,100) A
  100   FORMAT (F8.4)
        IF (A.LE.0.)GOTO 10
        ASUM=A+ASUM
        ASQ=A*A+ASQ
        N=N+1
        GOTO 20
   10   AN=N
        AMEAN=ASUM/AN
        SD=SQRT(ASQ/AN-(AMEAN**2))
        WRITE (6,200) N,AMEAN,SD
  200   FORMAT (1X,I3,' VALUES',/1X,'MEAN = ',F8.4,/1X,
     1  'STANDARD DEVIATION = ',F8.4)
        STOP
        END
$ENTRY
@                    Indicates end of input file
```

Figure 7 Preparing the Input

```
      ØØ1613.           MORTON                                FORTRAN SOUR
   CE LIST                          Ø6/19/68       PAGE   3   Page Heading
                ISN          SOURCE STATEMENT

                   Ø  $IBFTC

                   1          N=Ø

                   2          ASQ=Ø.

                   3          ASUM=Ø.

                   4      2Ø  REAS (5,1ØØ) A

    ERROR-SEVERITY 4,ISN-ØØØØ4  INCORRECT ARGUMENT LIST.
                                                   FORTRAN error message

                   5     1ØØ  FORMAT (F8.4)

                   6          IF (A.LE.Ø.)GOTO 1Ø

                  11          ASUM=A+ASUM

                  12          ASQ=A*A+ASQ

                  13          N=N+1

                  14          GOTO 2Ø

                  15      1Ø  AN=N

                  16          AMEAN=ASUM/AN

                  17          SD=SQRT(ASQ/AN-(AMEAN**2))

                  2Ø          WRITE (6,2ØØ) N,AMEAN,SD

                  21     2ØØ  FORMAT (1X,I3,' VALUES',/1X,'MEAN =  ',F
   8.4,/1X,
                           1  'STANDARD DEVIATION =  ',F8.4)

                  22          STOP

                  23          END

      ØØ1613           MORTON                                 Page
                                                   PAGE   4   Heading
```

Figure 8 The Compiler Output

```
P I 6            Print input starting at line 6
       ASUM=0.
   20  REAS (5,100) A   This line is in error
  100  FORMAT          Indicates printing terminated at user request
EDIT 7                  Correct the error
   20 READ (5,100) A
C 0                Clear the output file
A    22 INPUT LINES Enter APPEND mode starting with line 23
8.6
5.3
7.4
10.1
7.3
9.2
6.1
-0.0               This line will terminate the program
```

Figure 9 Editing the Error and Adding the Test Data

```
 OBJECT PROGRAM IS BEING ENTERED INTO STORAGE.   System loader
                                                    message
  7 VALUES
                                         Program output
MEAN =     7.7143

STANDARD DEVIATION =    1.5751
```

Figure 10 The Test Results

The production data is obtained from the data files and the RESTORE operation places it in the input (Fig. 11). This data is then augmented by some changes, and again the job is run and the results obtained.

Finally, the program and data are saved in the retrieval file for possible use at a later data. (Fig. 12)

```
                                   TAB here
A0   30 INPUT LINES    Enter APPEND mode at line 31
RESTORE/RPM&DATA1      Call for data to be brought to input file
C 00              Clear output file
START 310         Run job starting at line 31
P 00              Print output
RESTORE COMPLETED
0

P I 310           Print input from line 31
        RESTORE/RPM&DATA1    TAB expanded
17.6
21.4
18.3
15.1
17.0
19.8
18.5
16.9
17.4
20.0              This data restored from data file
15.5
16.3
16.4
17.1
19.2
17.5
18.8
16.9
17.7
19.3
0

D 23 310          Delete the test data and the RESTORE request
A0   42 INPUT LINES   Enter APPEND mode at line 43
17.7
19.6              Add new data plus terminator
18.0
-0.0
C 00
START0            Run it
P 0 78 820        Print output lines 78 through 82
 23 VALUES

MEAN =   17.9130
STANDARD DEVIATION =    1..4830
0
```

Figure 11 Obtaining the Production Data

```
C 0@                    Clear the output file
INS 1@                  Insert ahead of line 1
 STORE/RPM/MEAN/STDV    This line gets inserted
P I 23@                 Print input starting at line 23
$ENTRY
17.6
21@                     Terminate printing
INS 24@                 Insert ahead of line 24
$STORE                  End the first STORE request  } These lines inserted
 STORE/RPM/DATA2        Start next STORE request     }
A@   48 INPUT LINES     Enter APPEND mode
$STORE                  End second STORE request
START@                  Run it
P 0@                    Print output
          STORE/RPM/MEAN/STDV
STORE COMPLETED
          STORE/RPM/DATA2
STORE COMPLETED
@
```

Figure 12 Saving the Program and Data

3. THE IXSYS LANGUAGE

3.1 Introduction

As stated in Chapter 1, important parts of MSPSF are languages designed to allow easy and convenient use of the various processors and subsystems of MSPSF.

One of these languages is interpreted by the IXSYS program to provide the interface between the special and general computing processors. This language has two functions.

(1) It provides a link between the functions special computing processor (including the storage and retrieval system) and the general computing processor.

(2) It provides the control statements necessary for the use of the subsystems of the general computing processor.

Five requirements for the IXSYS language are given below:

(1) Programs and data records stored in the data files may be specified to be included in the input to the general computing processor.

(2) New subsystems may be defined and added for use in the general computing processor. This requires that the subsystem definition include system control card images as well as facilities for making parametric string substitution into such card images at appropriate points. Subsystem definitions should make use of the calling facilities from (1) above with the appropriate parameters. New subsystems defined through this method may refer to and use existing subsystems.

(3) Calls on subsystems may be as concise as the user desires for both the regular subsystems of the operating system and for user defined subsystems.

(4) Subsystem calls may occur at any point in the input. This means that statements not relevant to the subsystem may precede or follow subsystem calls. This also implies that it should be possible to define new subsystems in terms of old ones.

(5) The language should not conflict with any existing languages in the system (for instance by containing strings that are commonly used by other programs).

Throughout the discussion the term "card" is used to mean a card image whether the source of that card image is a card from the card reader, a card image from magnetic tape or disk, a line from a remote terminal, or a card image generated internally by some program.

3.2 Syntax of the Language

There are two parts of the IXSYS language which need syntactic specification, commands and substitutable parameter instances. The specification of commands is context free and is given in Backus Normal Form in Table 5; the syntax of parameters is not context free and is described in Sect. 3.4.3.

The choice of syntax for the IXSYS language has been greatly influenced by the fifth requirement above. Two symbols are used for designating command parts. The left bracket symbol was selected since it occurs on remote terminal keyboards but not on keypunches. Consequently, the previously existing software did not use it. It is unlikely that a user's program would use it. Its absence from the keypunch does not, however,

Table 5 Backus Normal Form Specification
of IXSYS Commands

```
< command > ::= < regular command > | < condensed command >

< regular command > ::= < IXSYS part > < string > < new card >

                        < command statement >

< command statement > ::= < R comm > | < B comm > | < M comm > |

                          < D comm > | < I comm > | < T comm > |

                          < F comm >

< R comm > ::= [R/ < description >

< B comm > ::= [B/ < description >

< M comm > ::= [M/ < description > |

               [M/ < description > / < M paramlist >

< D comm > ::= [D/ < D paramlist >

< I comm > ::= [I/ | [I/ < ch string >

< T comm > ::= [T/ | [T/ < TF string >

< F comm > ::= [F/ | [F/ < TF string >

< M paramlist > ::= < M param > | < M paramlist > / < M param >

< M param > ::= * < ch* string > *

< D paramlist > ::= < ch*'/, string > | < D paramlist > / < ch*/ string >

< TF string > ::= < ch= string > | = | = < ch= string > |

                  < ch= string > = | < ch= string > = < ch= string >

< condensed command > ::= < IXSYS part > / < description > |

                          < IXSYS part >/< description >/< M paramlist >

                          < IXSYS part > / MACRO / < D paramlist >

< IXSYS part > ::= < seven spaces > IXSYS

< string > ::= < ch string > | < new card > | < string > < ch string > |

               < string > < new card >
```

Table 5 Backus Normal Form Specification of IXSYS Commands (continued)

```
< ch string > ::= < any string of alphanumeric characters >

< ch* string > ::= < any string of alphanumeric characters not contain-
                     ing * >

< ch*'/, string > ::= < any string of alphanumeric characters not con-
                         taining *, ', or / or comma >

< ch= string > ::= < any string of alphanumeric characters not contain-
                     ing = >

< description > ::= < any MULTILANG description not containing formats,
                      element numbers, or labels >
```

exclude the use of the IXSYS language by card users since the left bracket code may be obtained by the multi-punch facility. The selection of the slash in a command is, again, an attempt to reduce the probability of conflicting with a user's program specification.

3.3 <u>IXSYS Commands</u>

The IXSYS commands are divided into three classes; retrieval commands, macro definition and call commands, and input control commands for use within macros.

3.3.1 Retrieval commands

The retrieval commands allow users to incorporate stored source and binary decks into the input stream. There are two retrieval commands, one for source cards and one for binary cards. The need for two commands arises from the fact that the storage and retrieval system recognizes this difference and requires a query to specify if binary is to be retrieved.

3.3.1.1 Retrieval

Source statements which have been stored in the data file may be included in an input stream for the general computing processor by use of the Retrieval command. The syntax of this command corresponds to < Rcomm > in Table 5. The records retrieved are in the format generated by the worker program STORE (see Appendix 3). These records may contain any source card images, including programs, data, and control cards, mixed in any desired way.

The Retrieval command may also be used to guarantee against ambiguity between a user's data and IXSYS commands, since the data retrieved is not processed any further by IXSYS. Thus, if a user must include data which conforms to some IXSYS command, he may first save it using STORE, and then include it in his input using Retrieve.

3.3.1.2 Binary Retrieval

Binary decks which have been punched by the MAP assembly program (possibly called by FORTRAN, COBOL, ALGOL, or L6) may be stored in the data file using the worker program ADDMP (see Sect. 2.3.5). These decks may later be included in an input stream to the general computing facilities by using the Binary Retrieval command. Since it is not possible to put a binary deck on an input file, the only way to include them in the input stream is through this command.

The syntax for the Binary Retrieval command corresponds to < Bcomm > in Table 5.

3.4 IXSYS Macros

Requirements (2), (3), and (4) of Section 3.1 - the ability to define subsystems, to provide an easy procedure for calling subsystems, and to make subsystem calls applicable at any point in an input stream - are satisfied by a macro capability within IXSYS. These capabilities are presented below and are illustrated through a discussion of some examples (Sect. 3.6).

The IXSYS macro facilities have two standard capabilities. First, repeated use of the same procedure, with substitution of parameters for specific application is allowed. Thus, a given sequence of statements to be used many times in some language may be defined as a macro once, given a name, and subsequently referred to only by name.

Second, macros in IXSYS provide a convenience for those users who are not familiar with IBSYS control statements and who may wish to perform some complex task requiring such knowledge. (The use of macros in this context is becoming more widely used, particularly in time sharing environments where communication with the supervisor is tedious without standard macro instruction for doing so.) This will be made clearer by an example below of a macro which allows a programmer to use the PDPMAP language on the 7040 to assemble a program for the PDP-8 computer. As will be seen, this is a complex task made trivial by IXSYS macros.

3.4.1 Macro Definitions

A macro definition has three principal parts: a heading card, prototype cards, and the terminating card.

The macro definition header card is the Define command specified below. It contains the macro name and the list of formal parameters.

The macro prototype cards includes any symbolic cards including IBSYS control cards, source statements in a programming language, data cards, or any IXSYS commands. These prototype cards may include specifications for formal parameter instances; presumably, each formal parameter occurs on at least one prototype card.

The macro terminating card contains [END] as the first five characters and a blank in column six.

3.4.2 Macro Calls

A macro call is made through use of the Macro command given below. It contains a MULTILANG description of the macro and a list of the actual parameters to be used. This is explained further in the discussion of the Macro command in Sect. 3.4.4.2.

3.4.3 IXSYS Parameters

3.4.3.1 Formal Parameters

Formal parameters appear on the macro definition heading card in the format of MULTILANG keys, spearated by a slash (/). Although they may be of arbitrary length, only the first five characters are retained by the IXSYS program. They specify the strings to be considered as formal parameters, and the order in which the actual parameters appear in the macro call.

Formal parameters also appear on macro prototype cards enclosed in square brackets ([ ]). Here the entire formal parameter is retained. One purpose of the brackets, then, is to specify the extent of the formal parameter since this may be more than five characters.

If the corresponding actual parameter is omitted in a macro call, the formal parameter itself becomes the actual parameter. In this case the complete parameter as it appears on the prototype card is used. This convention provides default conditions which simplify macro calls in the most common case..

Another purpose of the square brackets is to specify spacing. If the right bracket (]) following a formal parameter is preceded by a space then that parameter will always end in the column of the right bracket. If the corresponding actual parameter is shorter, the remaining columns will be spaces; if it is longer, it will be truncated.

Not every string enclosed in brackets need be a formal parameter. Brackets which do not delimit formal parameters are treated as text characters; those which do are dropped. In order for a string enclosed in brackets to be a formal parameter, it must match one of those listed on the macro definition heading card, i.e., if the string contains no more than five characters it must conform exactly to one of the listed parameters; if it is longer than five characters, the first five must conform to a listed parameter.

3.4.3.2 Actual Parameters

Actual parameters occur only in macros calls. Each actual parameter is a string of characters in the form of a MULTILANG literal (enclosed in a pair of asterisks). Such a string may contain any character except asterisk.

3.4.3.3 Null Parameters

As noted earlier, when an actual parameter is omitted from a macro call the formal parameter is taken as the actual parameter. A special symbol is provided to explicitly allow a null parameter. This symbol is [N/cc where cc is a pair of identifying characters which may optionally be included to distinguish between different null parameters.

This special symbol may be used either as an actual parameter or a formal parameter. In the first case each instance of that parameter will be made null. Here cc need not be given. In the case where the null symbol is used as a formal parameter, the parameter is made null only if no actual parameter is supplied and the default condition applies. Here cc is necessary if there is more than one such formal parameter and they need to be distinguished.

In any situation when a parameter is made null, the bracket convention for spacing still holds. Thus, if a parameter is null then if, in the formal parameter instance, the right bracket is preceded by a space, then the entire field of the parameter is filled with spaces. If the right bracket is not preceded by a space, then the parameter is omitted entirely (actually squeezed out). For example, suppose the following card is being processed and CDE is a formal parameter:

```
AB[CDE]FGH
```

Now suppose a null actual parameter is supplied for CDE. Then the result will be

ABFGH

However, consider the second example:

AB[[N/1]CDE

Now if no actual parameter is supplied the result will be

AB   CDE

3.4.4 The Macro Commands

Each of the two commands described below has two syntax specifications, one corresponding to < regular command > and one corresponding to < condensed command > in Table 4. The regular forms are discussed in Sects. 3.4.4.1 and 3.4.4.2. The condensed forms are described in Sect. 3.4.4.3.

3.4.4.1 Define

The Define command signals the start of an IXSYS macro definition. It also serves as the macro definition heading card. The syntax for the Define command corresponds to < Dcomm > or < Dcond comm > as given in Table 4. The first member of < D param list > is the name of the macro and the remaining members are its formal parameters. Thus the format of < D comm > is

[D/name/fp1/fp2/.../fpn

where name is the macro name and is used as a key under which the macro is stored in the data file; and fp1,fp2,...,fpn are the formal parameters of the macro.

3.4.4.2 Macro

The Macro command is used to specify a call for a macro expansion to be included in an input stream at the point of the command. The syntax for the macro command corresponds to < Mcomm > or < M cond comm >

as given in Sect. 3.2. Here < description > is used to retrieve the macro and < M param list > is the list of actual parameters. (In addition to having the name as a key, each macro also has the key IX.MA.) The general format of < Mcomm > is

[M/desc/*ap1*/*ap2*/.../*apn*

where desc is a MULTILANG description of the macro being called; and ap1,ap2,...,apn are the actual parameters of the call.

3.4.4.3 The Condensed Forms

As stated earlier each of the above two commands may be incorporated directly into the call on IXSYS. The equivalent formats to the ones given in the preceding sections are:

for a definition

IXSYS/MACRO/name/fp1/fp2/.../fpn

and for a call

IXSYS/desc/*ap1*/*ap2*/.../*apn*

The latter of these supplies the method for achieving the objective of easy use of subsystems. The user need simply supply the name of the subsystem (the name of the macro) and any required parameters.

3.5 Input Control Commands

Three commands are provided to allow greater choice in the source and the selection of the input prepared by IXSYS for the general computing components.

3.5.1 Input

Just as the retrieval commands allow for inserting stored data in the middle of prepared input, the input command allows for the inclusion of prepared input in the middle of an IXSYS macro. The syntax of this command corresponds to < Icomm > in Table 5. The format of the Input command is

```
[I/cccccc
```

where cccccc may be one of the following three:

blank - read all the input on the user's input file or until a $JOB card is read;

a six digit octal number - read that many lines from the input file or until a $JOB card is read;

otherwise - read until a card is encountered starting with cccccc.

3.5.2 If True

Selection from a fixed set of alternative choices may be performed using the If True command. The syntax of the If True corresponds to < Tcomm > in Table 5. The format for this command is:

```
[T/string1=string2
```

which means if string1 is identical (in at least the first six characters) to string2 then include the card which follows the If True command card. Otherwise, skip one card.

The inclusion of a group of cards may be achieved by having the included card contain a Retrieval or Macro command.

3.5.3 If False

If False is similar to If True except that inclusion occurs if string1 is not identical to string2. It is given by < Fcomm > in Table 5.

The primary purpose for having both If True and If False is to allow for conditions on two or more parameters. For example,

```
[F/p1=X
[T/p2=Y
```

will include the next card if and only if p1=X <u>or</u> p2=Y. Similarly, any other logical connective may be achieved with various combinations of If True and If False.

3.6 Examples and Analysis

Two examples are provided and described in detail in order to illustrate the power that the (relatively simple) macro facility of IXSYS offers to the user. In Sect. 3.6.3 a discussion and some additional examples are given to show how the objectives of Sect. 3.1 have been met.

3.6.1 Example 1

Example 1 shows how a subsystem not originally included in the general computing facilities may be defined using existing subsystems.

The two macros defined in Example 1 are used by PDP-8 programmers to assemble programs for the PDP-8 on the 7040 using the PDPMAP subsystem. These macros are useful in removing the need to key on a remote terminal the lengthy entries necessary to assemble a PDP-8 program. Without this capability it would have been virtually prohibitive for PDPMAP programmers to use the remote terminals of MSPSF.

In addition, the use of macros in a PDPMAP job does not require extensive knowledge of IBSYS control card, which otherwise would be needed. The PDPMAP programmer is thus also not distracted from his main task, namely, that of writing a PDP-8 program. This is particularly true for a beginning programmer who would have been forced to learn both the IBSYS and PDPMAP systems.

As an illustration Example 1 (see Table 6) uses all the IXSYS commands in at least one form. A step-by-step explication of these macros follows. Note that it is assumed that certain records have been pre-

Table 6 The PDPMAP and P.SYM Macro Definitions

```
*
1.          IXSYS/MACRO/PDPMAP/NAME/8DEF/TEST/*OO[N/1*
2.    $IBJOB          DECK
3.    [F/[[N/1]=DD
4.    [T/[[N/1]=SYMTAB
5.    $IBMAP [NAME   ]DD
6.    [F/[[N/1]=DD
7.    [T/[[N/1]=SYMTAB
8.    [F/
9.    $IBMAP [NAME   ]
10.   [R/[8DEF]
11.   [I/
12.   $IBSYS
13.   $SWITCH         S.SPP1,S.SU14
14.   $CLOSE          S.SU14,MARK,REWIND
15.   $IBJOB PDPMAP   NOMAP
16.   $IEDIT          U14
17.   $IBLDR [NAME   ]
18.   $IEDIT          IN
19.   $IBREL
20.   [B/PDPMAP
21.   $ENTRY          [TEST]
22.   [T/[[N/1]=SYMTAB
23.   [M/P.SYM
24.   $SWITCH         S.SPP1,S.SU14
25.   [END]
26.         IXSYS/MACRO/P.SYM
27.   $IBSYS
28.   $OPEN           S.SU14,REWIND
29.   $IBJOB SYMTAB   NOMAP,NOSOURCE
30.   $FILE           'S.FBIN',U14,U14,BLOCK=150,MIXED,LRL=14,TYPE3
31.   [B/SYMTAB'AND'DECK
32.   $ENTRY
33.   [END]
```

* Note: These numbers are included only for explanatory purposes and are not actually part of the statements.

viously stored in the data file.

Statement[19]	Explanation
1	IXSYS is being called to define a macro called PDPMAP that has four parameters. The first is the name of the PDPMAP program being written; the second specifies whether the program is for the PDP-8 or the DEC-338 with the PDP-8 the default option; the third parameter specifies whether the program is to be just assembled, punched on paper tape, or loaded into the PDP-8 for execution, with assembly only the assumed option; the fourth parameter specified whether a debug dictionary should be obtained, whether a PDP-8 symbol table should be punched, or neither, with neither the default option.
2	An IBSYS control card.
3-5	If the fourth parameter is DD or SYMTAB then the $IBMAP card should contain the program name and the DD option. Note that the name parameter is a fixed length field. This is because the DD option must be in column 16.
6-9	If the fourth parameter is not DD or SYMTAB then the $IBMAP card should contain just the program name.

(19) The statement numbers correspond to the line numbers in Table 1.

Statement (continued)	Explanation
10	Following the $IBMAP card goes a set of MAP macros which define the PDP-8 to the MAP Assembler. This set of definitions must be previously stored.
11	Read the PDPMAP program that the user has typed.
12-19	IBSYS control cards, with the program name substituted in card 17.
20	Retrieve a binary deck of a program called PDPMAP. This is the postprocessor which converts from 7040 code to PDP-8 code. There will be no confusion between the macro called PDPMAP and the binary deck called PDPMAP because MULTILANG keeps them separated.
21	A control card with the third parameter substituted.
22-23	If the SYMTAB option was given as the fourth parameter then the IXSYS macro P.SYM is to be included at this point.
24	A control card.
25	The macro definition terminating card.
26	Call IXSYS to define the macro P.SYM, which has no parameters.
27-30	Control cards.
31	Retrieve the binary deck called SYMTAB and DECK. It is the presence of card 31 which requires P.SYM to be a macro rather than just a retrieval. A retrieval would not allow for this additional retrieval in the middle.

Statement (continued)	Explanation
32	A control card.
33	The terminating card for P.SYM.

Table 7 shows three examples of the use of the PDPMAP macro. The first example is the simplest case where the only parameter supplied is the deck name. Example (b) shows the same job as (a) but this time the binary tape is to be punched. Note that the second parameter has been omitted but that the third parameter is supplied. Thus the default option 8DEF will be used as the second parameter. In Example (c) all parameters have been supplied, calling for a DEC-338 program to be punched with a symbol table.

3.6.2 Example 2

A medical research group collected a large amount of data on patients in a clinic for several separate studies. These studies varied widely in the number of patients involved and the amount of data collected per patient. The data was coded and placed on punched cards, each patient record varying from as little as three cards for the smallest study to as many as 22 cards in the largest.

Originally only sorting and counting operations were performed on the data, but after several years in depth studies were begun. Unfortunately, the data for the different studies were in different formats and of different lengths. Consequently, separate programs had to be written for each study; inter-study analysis was extremely difficult; and, to make matters worse, the programs were to be used by medical technicians rather than computer programmers.

```
IXSYS / PDPMAP /*TESTA*

TAD X

DCA Y

JMP Z

END
```

(a) A simple PDPMAP job with no options

```
IXSYS / PDPMAP/*TESTB*//*PUNCH*

TAD X

DCA Y

JMP Z

END
```

(b) A PDPMAP job for the PDP-8 to be punched

```
IXSYS / PDPMAP /*TESTC*/*338DEF*/*PUNCH*/*SYMTAB*

EDS VEC

OCT 4100, 4100

END
```

(c) A PDPMAP job for the DEC-338 in which both a binary tape and a symbol table are to be punched

Table 7 Examples of Using the PDPMAP Macro

MSPSF and the macro defined in Example 2 is extremely useful in this application for two reasons. First, from the programmer's point of view, much duplication of effort may be saved since all common elements need only occur once by storing them in the data file. Second, the user's job is reduced to dialing a telephone, signing-in, running a job consisting of one statement and printing the results. The one statement executed contains the name of the macro, the name of the study to be analyzed, and the name of the program used for the analysis.

As an illustration Example 2 serves two purposes. First, it shows how such a macro can satisfy the needs of the user in solving the problems mentioned above. Second, as in Example 1, Example 2 shows how a problem may be stated in a simplified manner rather than the detailed manner required by the operating system.

The macro definition is given in Table 8. Line 1 is the macro definition heading card. The name of the macro is CLINIC. Although three parameters are listed, only two are expected to be supplied. The third is provided to allow line 3 to be read as part of the definition. If line 3 contained just $JOB without brackets, it would terminate the reading of the definition.

Line 2 verifies that a necessary parameter has not been omitted. If it has been omitted, line 3 will be included and terminate the job immediately with an error message. If the required parameter has been included line 3 will be ignored.

Many cards which are common to all applications of this macro are retrieved in line 4. Some of these are system control cards and some are program statements.

Line 5 calls for the inclusion of those statements peculiar to each application such as DIMENSION and FORMAT statements.

Lines 6 through 10 are more program statements. Line 8 has two parameter substitutions. One of these is the subroutine which performs the required computation. The binary deck for this subroutine is retrieved from the data file and included in the input by line 11.

Line 12 is an IBSYS control card. Line 13 calls for the retrieval of the data on which the computation is to be made.

```
*
1.      IXSYS/MACRO/CLINIC/STUDY/TEST/JOB
2.  [T/STUDY=[STUDY]
3.  $[JOB]
4.  [R/CLIN1
5.  [R/[STUDY]&DIMFOR
6.      DO 2 I=1, NPATS
7.    2 READ (5,1)(DATA(I,J,K),K=1,NUMBER,K=1,NCARDS)
8.      CALL[TEST](NPATS,NUMBER,NCARDS,6H[STUDY],ARRAY1,...,ARRAYN)
9.      STOP
10.     END
11. [B/[TEST]
12. $ENTRY
13. [R/[STUDY]&DATA
14. [END]
```

* Note: These numbers are included only for explanatory purposes and are not actually part of the statements.

Table 8 Example 2

3.6.3 Conclusions regarding the IXSYS Language

The two examples given illustrate the usefulness of the macro facility and the other functions provided. These and other examples will be used to illustrate that the five specific objectives, outlined at the beginning of the section have been met.

1. The facility to save and call for programs and data has been supplied by the retrieval commands. This facility could, however, be extended to allow access to data stored in formats other than those currently allowed. A user could provide parameters to a retrieval request with a description of the data to be obtained.

2. The two examples above have shown how subsystems may be defined. Subsystem definitions may include all card images; subsystem definitions may make use of any of the retrieval commands; parameter substitution has been provided. Example 3 (Table 9) illustrates a macro for providing access to one of the existing subsystems of the general computing system FORTRAN. As seen in line 1, the macro name is FORTRAN, and it has two parameters. Line 2 shows that the second parameter is options for the $IBJOB control card, which may be omitted. The first parameter is the program name, as seen in line 3.

```
1.       IXSYS/MACRO/FORTRAN/NAME/*OO[N/1*
2.  $IBJOB      [[N/1]
3.  $IBFTC[NAME]
4.  [END]
```

Table 9 Example 3

3. Subsystem calls are very concise; all that is required is the name of the subsystem. This point has been particularly strengthened by the inclusion of the condensed forms and default parameters in the IXSYS language. Thus a call for FORTRAN, assuming Example 3 may be as short as:

```
IXSYS/FORTRAN
```

or as long as

```
IXSYS/FORTRAN/*JOB1*/*DECK,NOGO*
```

4. Subsystem calls may occur at any point in the input, except from cards accessed via the Retrieval command. This exception is provided on purpose (see (5) below). Subsystem calls may occur within other subsystems, and, in fact, by using the conditional commands or the define command it is possible to write recursive macros.

5. Much has already been said about efforts to make IXSYS as unobtrusive as possible. To summarize, the condensed forms allow some strings to be included as parameters to the IXSYS call, thereby eliminating them from consideration as data; format parameters must match some member of a list which must be given in advance; the use of such characters as [,], and / serve to protect against ambiguity in most cases; statements may be stored and later included via the Retrieval command to insure that they are not questioned further.

4. USER EVALUATION

4.1 The Community of Users

The users of the Moore School Problem Solving Facility come, naturally enough, mostly from the Moore School of Electrical Engineering at the University. They represent primarily research projects.

The nature of the work of the users ranges from simply using the editing facilities for preparing input, to constructing a complex graph theory system involving concurrent operation of the IBM-7040 and the DEC-338. Within this range there are storage and retrieval projects using only the special computing processor and programming jobs which use the general computing processor extensively. One information retrieval project is concerned with patient information from the cardio-vascular research group and Pathology Department at the University of Pennsylvania Hospital. Another project is concerned with interactive competitive games. Among projects using the general computing processor is one concerned with list structured memories and the theory of data structures. Another project is concerned with extendable operating systems. Table 10 summarizes these projects, indicating which facilities they use frequently and which they use occasionally.

While these projects are not numerous enough to allow significant statistical studies, they represent a broad range of computing requirements that demonstrate the need for "wide applicability". Similarly, although the number of users has been limited, they represent a wide range of levels of computer competence. Therefore this group of users is particularly suitable as a basis for evaluation of the services and functions of MSPSF.

		Facilities used (1 - frequently, 2 - occasionally)					
Project	Reason for using MSPSF	Editing	Storage & Retrieval	Computing on special level	IXSYS macros	IBSYS Sub-systems	Own Sub-systems
Hospital data input	Store data to be retrieved later	1	1				
Hospital data retrieval	Examine previously stored data	2	1	1			
Interactive game playing	Storage and retrieval of data for each player	2	1	1			
Graph theory system	Write programs for DEC-338 and IBM-7040	1	1		1	1	1
List structured memories	Write programs for IBM-7040	1	2		1		1
Extendible operating systems	Write programs for IBM-7040	1	2		1		1

Table 10 Users of MSPSF and the Facilities They Use

The evaluation that follows is divided into two parts. First, specific comments are presented regarding the component processors of MSPSF. Second, some more general comments relating to the total system are given. Some of the comments concern the implementation of MSPSF, and while these are of little interest to the system architect they are included both for completeness and as a reminder that they are considered important by the users.

4.2 General Characteristics of the Use of MSPSF

For a period of about four months prior to this writing, the University of Pennsylvania Computer Center has been operating its IBM-7040 computer under the MSPSF system for four hours each day, two hours in the morning and two hours in the evening. During these four hours the computer was largely occupied with background work (jobs submitted on cards). The total time during which MSPSF has been active with remote users has been small. The number of remote terminals in use at any time did not usually exceed one, and never exceeded four.

There are two classes of programs which had to be excluded from running under MSPSF for technical reasons. First, some programs were too large, since the nucleus of MSPSF is larger than that of the regular version of IBSYS. Second, some programs made reference to absolute core locations which were different under the two versions of IBSYS.

Additionally, in order to provide adequate response for remote users, jobs with maximum run times greater than 10 minutes were supposed to have been excluded by the computer operator. Occasionally, the operator would start a 20 or 30 minute job causing the remote users to experience intolerable delays.

4.3 Evaluation of the Component Processors

4.3.1 Evaluation of the Input/Output Processor

There is general agreement among MSPSF users that the terminal control language is inadequate in its present form. It is too awkward for the average user and incomplete for the more advanced user.

The average user would prefer a smaller command vocabulary, where each command would accomplish more for him. This, however, would restrict those users who do make use of all the commands. The solution seems to be to provide a macro language similar to that of IXSYS. A user could then simply type, for example, his name, and the entire system could then appear to him as a special purpose computer designed to process his particular language.

The more advanced users have requested some new commands for the input/output processor. Most of these are editing commands for the input/output files. They range from simple ones, such as, moving output to the input file, to such complex operations as full text search and replacement operations. Here two macro operations could prove very useful.

4.3.2 Evaluation of the Special Computing Processor

The major complaint with the special computing processor is that it is too difficult to write worker programs. Under the present implementation of MSPSF worker programs must be written in assembly language and must be entirely self-contained except for calls on the storage and retrieval system. What is needed is a new linking loader capable of piecing together one large program from a library of small subroutines. The subroutines would be kept in the storage and retrieval file. Additionally, the calls on the storage and retrieval system routines need to be standardized to conform to those produced by the language processors. This would allow programs written in FORTRAN, for example, to call for data to be retrieved directly from the data file. Both of these changes would require additional modification to the dynamic storage allocation routines of MULTILANG.

One result of this difficulty is that there are not an adequate number of worker programs available. In addition to IXSYS only three other routines have been added to the MULTILANG worker program library to specifically support the remote terminal users. These programs perform the functions of saving and restoring input files for later use and punching saved files.

Several other such programs have been suggested by the users. An existing program which stores binary decks should be able to recover arbitrary punched output from the punch file of an IXSYS run. This same program or another should be able to actually punch the recovered decks if desired. Similarly, facilities should be provided to save printed output and to allow it to be printed on the high-speed line printer. Along these same lines, the program that saves input files should be expanded to include arbitrary sources such as magnetic tape.

There are, of course, an indefinite number of such programs which could be added to the MULTILANG program library. The point here is that a specific effort has to be made to provide special facilities for remote terminal users. Moreover, such an effort should be a continuing one. As in the case with virtually every other computer function, regardless of how elaborate a system gets, someone will think of some modification which will provide an additional capability.

4.3.3 Evaluation of the General Computing Processor

The use of an already existing operating system empowered MSPSF to handle the work that users had previously generated. Most users found that no modifications were required to run past programs. The exceptions involved programs that either were too large or involved absolute machine addresses.

On the other hand, difficulties arose from the incompatibility between the input/output formats used with the compilers, assemblers, etc., and the formats of the new remote terminals. Specifically, IBSYS input is expected to be 80 column card images and output is formatted for a 132 character line printer. Since remote input is 64 characters per line, spaces are added to the end to make 80. In most cases this is sufficient. The output format is more difficult to handle since, again, lines are limited to 64 characters. MSPSF solves this problem by simply breaking the 132 character line into two 64 character lines and ignoring the last four characters. A better solution, however, would be to use editing programs which allow the user to specify that portion of each line of current interest. This approach may apply generally as the best way to bridge the gap between pretested, in-use software and new hardware, particularly for remote accessed systems with a variety of terminal types and frequent hardware additions.

4.4 Evaluation of the Overall System

Three points of significant interest have been made by the users of MSPSF.

1) Reliability in both the total system and the component processors is critical. Those users who attempted to use MSPSF during the development and early production stages became so disheartened at

system failures that they were very hesitant to try again even when the problems were cleared up. This observation holds for both hardware and software; the user is clearly not concerned with why the system fails.

This does not mean that all components must be error free before the system can be used. Instead, every possibility for error detection and correction must be included. When the hardware fails, sufficiently explicit error messages should point out the difficulty to allow prompt diagnosis and correction.

2) Each level in the hierarchy has its own control language. The complexity of that language reflects the complexity of the operation of that level. Consequently, those users who use only the simpler levels of the hierarchy need only learn the simpler control languages.

The users of MSPSF who have benefited the most from this structure are those who are primarily interested in the special computing processor. These users have been able to use their programs without having to learn the complex operating system language of IBSYS.

3) The computer used at the computation levels should be a much faster one. On this point there is an essential difference between remote and batch operation. Most remote users of MSPSF found a delay of ten to twenty minutes for a run unbearable. As has been found elsewhere(26) remote users tend to be less careful in program preparation, and consequently, get less accomplished per run. He thus expects error messages of a trivial nature to come back to him with a minimum of delay. The 8 μsec. cycle time of the IBM-7040 does not permit an adequate amount of computation within the remote users tolerance limits.

(26) See Footnote 1.

5. CONCLUSIONS

5.1 The Hierarchical Approach to System Architecture

A new system architecture is proposed for time shared computing that alleviates the high overhead due to swapping and program size restrictions due to multiprogramming. It utilizes a hierarchy of processors, where each processor is assigned tasks on the basis of four factors: interactive requirements, frequency of use, execution time, and program length.

In order to study the hierarchical approach to system architecture, MSPSF was built and used. The study of the manner of operation of MSPSF and the reactions of the user has resulted in a number of conclusions which have defined and clarified the hierarchy of processors approach to system architecture.

1. The choice of three processors for the hierarchy of MSPSF is based on the minimum number that is considered sufficient. Separating tasks based on execution time is an essential feature of the hierarchical approach. This separation implies a minimum of two processors in addition to the input/output processor. More than three levels in the hierarchy are possible on the same basis.

It has been found advantageous that the input/output processor, the lowest in the hierarchy, be implemented using a separate computer. This is based on the consideration that the present state of the art of remote terminal devices is highly dynamic. Modifications to the system for handling new types of communications and terminals are to be expected. Such modifications are much easier to perfect and incorporate on a separate computer than within the environment of equipment also serving the other processors.

2. The processors in the hierarchy should be program compatible, and each should be general purpose. General purpose means that each processor is capable of running user programs. If the processors are program compatible, any program may be run at any level. This also allows the debugging of programs on a lower level processor, and then shifting to a higher level processor for production running.

Compatibility is also necessary for system maintenance. As the system grows it should be expected that functions provided by one level will be moved to another. Also, it may be necessary to debug system components at one level which are to be used at another.

3. The selection of the processor in which a user's program is to run should be either user directed or automatic. Both mechanisms should be provided. If a user wishes, he should be able to select a processor using the terminal interface language and one or more of the processor interface languages described in (4) below. When the user does not select the processor, the system must make estimates of the various factors which determine the proper processor for a task.

The four parameters which were used subjectively in assigning tasks to processors in MSPSF were amount of interaction required, frequency of use, program length, and execution time. In an automatic selection procedure (not incorporated in MSPSF), some formula, such as a weighted sum, comprised of these four factors would be computed, and the value of the formula would determine the processor chosen. For example, if

$$\Sigma = W_1F_1 + W_2F_2 + W_3F_3 + W_4F_4$$

then if

$$\Sigma < \Sigma_1, \text{ use processor 1}$$
$$\Sigma_1 \le \Sigma < \Sigma_2, \text{ use processor 2}$$
$$\Sigma_2 \le \Sigma, \text{ use processor 3.}$$

It is to be expected that conflicts should arise between a program's requirements and the resources of the processor to which it has been assigned. Again, the resolution of these conflicts should be either automatic or by user direction, and, again, program compatibility is required so that reassignment is possible.

4. A macro language similar to the IXSYS language is necessary for communication between every pair of processors, serving also for communication between the user and each of the processors.

The usual view of a macro facility is one of a mechanism for providing some user-defined subroutines to an assembly program. These subroutines may then be called from time to time during the course of assembling a particular program. The utility of such macros comes from their being called a number of times during the assembly process, thus saving the user from enumerating the lines generated by the macro.

The same approach applies to input stream macros. Instead of being used repeatedly in a single program, however, an input stream macro will be used to assemble a number of programs. Input stream macros have the ability to simplify communication between the user and the total system.

The IXSYS language differs from existing input stream macro languages in two ways. First, IXSYS macros may include statements in various levels of language, such as system, compiler and assembly languages, as well as data. Thus, the user does not need to rely on the logic of the system programmer to organize his work. This approach also allows parameter substitutions in statement of languages at all levels.

Second, IXSYS includes conditional statements which provide power and flexibility not found in other input stream macro languages. Such features are found in programming language macro processors, and there is no reason why input stream macro processors should not be just as powerful.

5. In MSPSF the information storage and retrieval system is part of the special computing processor. In order to better handle the needs of the total system, however, it is suggested that the storage and retrieval system be a separate special purpose processor, probably implemented on its own computer with links to each of the other three processors. There are three reasons for reaching this conclusion.

First, each of the processors in the hierarchy may require direct access to the storage and retrieval system.

Second, if the storage and retrieval system is accessible to each of the other processors directly, and if the several processors are implemented on the same or compatible hardware, then commonly used data and programs may be stored only once. For example, a compiler used on two levels of the hierarchy, depending on the length of the program to be compiled, could be maintained in the storage and retrieval facility and accessed by all processors using it.

The third reason for separating the storage and retrieval system from the other processors and, particularly, for implementing it on a separate computer is modularity. The same reasons apply here as for input/output devices in (1) above. New devices for storage and retrieval and new techniques for organizing data abound.

6. The block diagram of a computing system which satisfies the above five conclusions is shown in Fig. 13. Each of the three processors is shown. From any processor at level i to any other processor at level i + 1 there is a language L_{i+1}. Each of the processors may communicate directly with the storage and retrieval system.

The multiplexing factor at each level is represented by n, m, and ℓ on Fig. 13. That is, n users may use a level 1 processor simultaneously; m may use a level 2 processor simultaneously; ℓ may use a level 3 processor simultaneously. From these numbers it is possible to compute expected reaction times from each processor, provided adequate estimates are available for the distribution of execution times for each processor and the probability that a user of level i will move to level i + 1. These estimates depend upon the work load and job mix for each installation. For MSPSF, m and ℓ were each 1. Since the average execution time at level 3 was long and the probability of use of level 3 was high, reaction times were frequently very long.

From Fig. 13 it can be seen how expansion to more than three processors could be accomplished. The highest numbered processor could be split into two, the first being "Processor for somewhat longer and larger tasks" and the second "Processor for other tasks."

5.2 Suggestions for Future Research

1. The selection of processing units must be investigated in light of advances in computer design. Concepts which need further investigation within a hierarchical system architecture include multiprogramming, multiprocessing, virtual memories, etc. A system built with these advanced techniques will require additional work to determine the proper distribution of software resources among the various processors.

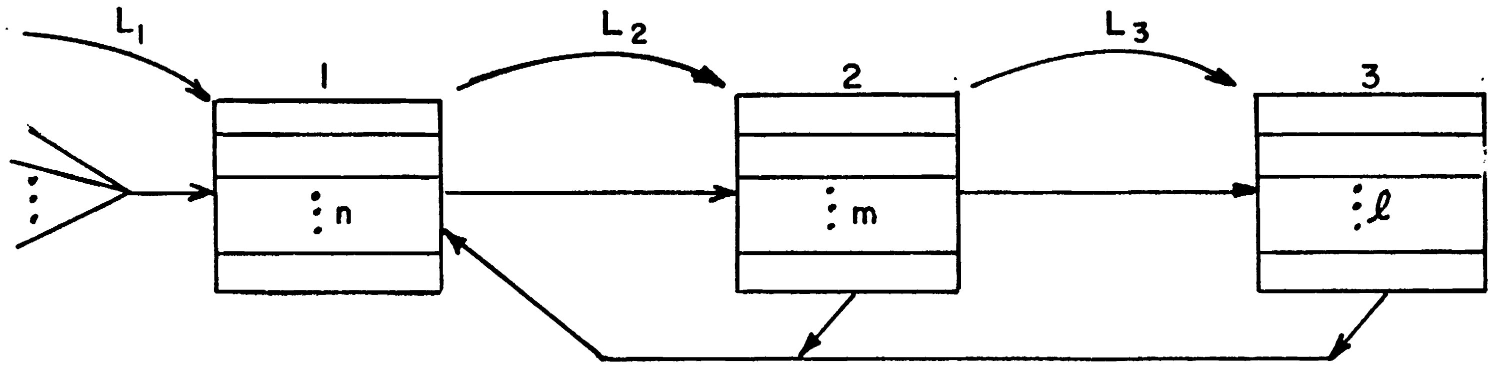


Figure 13 A Hierarchical Computing System

2. Additional studies, probably involving the simulation of a system, are needed to determine processor sizes in order to meet reaction time requirements.

3. An algorithm is necessary for handling automatic assignment of tasks to processors. Additional investigation is needed to determine (a) the selection of important factors, (b) the proper relationships between these factors, and (c) way of automatic estimation of the values of the factors for a task.

5.3 Implications for Future Applications

The system architecture described in this dissertation may be advantageous in some areas of future computer system research. The advantages of modularity for remote terminals and mass storage systems have already been pointed out. Software development may also profit from this modularity. Consider, for example, the development of a conversational program such as a teaching program or a conversational compiler. It is highly likely that such programs will be quite large and time consuming to debug. Consequently, such debugging would normally take place in the highest level of the hierarchy. Once developed, however, the critical factors in using such programs are high frequency of use and high interaction requirements. At this point, highly interactive programs may be moved to the lowest level processor for improved service to the users.

The development of other operating system resources may proceed along similar lines. Debugging runs may be carried out wherever they best fit into the job stream and wherever the best debugging facilities are available for the particular type of program. In fact, the highest level processor may be used for developing new operating systems without interfering with other users. This, in turn, has implications for future computer hardware development, such as in the area of privileged instructions.

The main characteristic of operating system experiments which makes them difficult to accommodate in present on-line computing systems is that they require complete control of the computer for the duration of the experiment. During this time the computer is thus unavailable to other users. The hierarchical approach allows one of the processors to be temporarily withdrawn from general use, but does not require the entire system to be closed to all other users.

Other application areas have also presented difficulties for present systems for the same reason of requiring complete system domination. Examples of this include some instances of on-line data collection and real-time process control where the data rates are sufficiently high. Here, again, one processor may be temporarily diverted to a special project while service to other users continues.

APPENDIX 1 THE IXSYS PROGRAM

The IXSYS program is actually two program decks: IXSYS handles input preparation, and IXOUT recovers output. For the rest of this discussion IXSYS will be used to refer only to the one deck. IXOUT is covered in Appendix 2.

A1.1 Basic IXSYS

Basic IXSYS refers to those portions of IXSYS which get used when an input file which contains no IXSYS commands is processed.

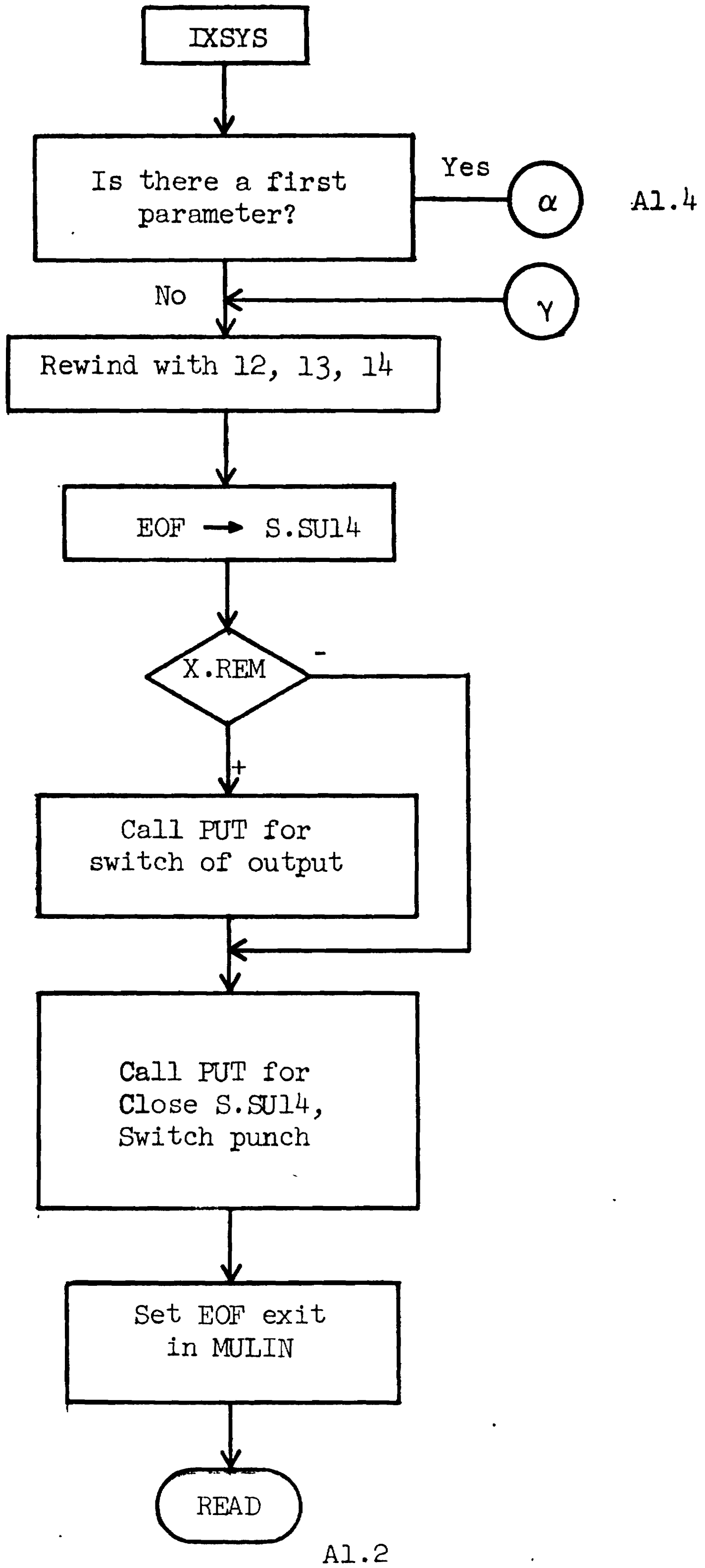
A1.1.1 Initialization

The primary responsibility of the initialization section of IXSYS is to set up the input, output, and punch utilities which are used during an IXSYS job. Each must be rewound, an End-of-File mark is written on the punch utility, and some initial IBSYS control cards are written on the input utility (see Fig. A1.1). Then, an end-of-file exit in the MULTILANG input routine is set in case a terminating $JOB card is not supplied. Lastly, if no condensed form macro call is pending control passes to the read routine. In the figure, exit α is taken if a condensed form for either a macro definition or call is given (Sect. 3.4.4.3). Exit β is taken if a macro is to be expanded before reading input.

A1.1.2 Reading and Writing

A card is read from the input file to see if it is a $JOB card (Fig. A1.2). If there is no more input, IXGET will exit to IXEOF. If a $JOB card is read, it is treated like an end-of-file and otherwise ignored.

If the card is not a $JOB card it is checked to see if it is an IXSYS command card. If not it is simply written out, and the next card is read. If an IXSYS command is read, the appropriate switches are


IXSYS
Is there a first parameter?
Yes
α
A1.4
No
γ
Rewind with 12, 13, 14
EOF → S.SU14
X.REM
-
+
Call PUT for switch of output
Call PUT for Close S.SU14, Switch punch
Set EOF exit in MULIN
READ

A1.2

Figure A1.1 Basic IXSYS Initialization

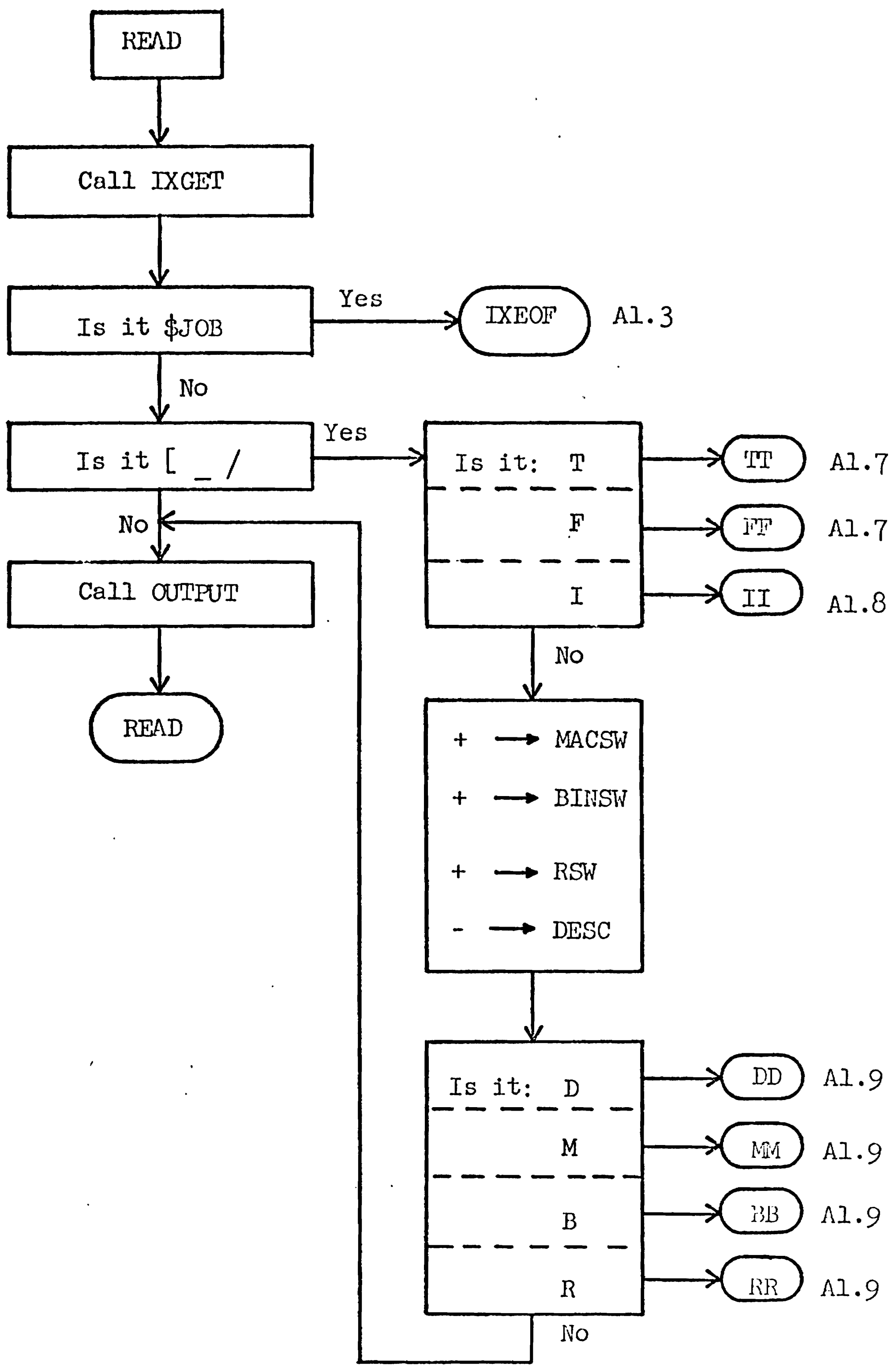


Figure A1.2 Basic Read Routine

initialized and control passes to the appropriate command routine.

A1.1.3 Terminating

After all the input has been processed, a check is made to determine if any IBSYS input was generated. If not, a message to that effect is printed and IXSYS returns to the MULTILANG monitor. Assuming that there is work for IBSYS, then several additional control cards are written out, including those necessary to reload MULTILANG, with some selectivity based on whether the job is for a remote user or a regular batch user. Just prior to passing control back to IBSYS, a card for switching the input to utility 12 is stored where IBSYS will find it. The exit from IXSYS is made in such a way that remote terminal output buffers will be closed.

A1.2 Condensed Forms

If the call to IXSYS has any parameters, then it must be a condensed form of either a macro definition or a macro call (Fig. A1.4). If it is a macro definition it is handled immediately by IXSYS. If it is a macro call, it is simply noted to be handled after the initialization is completed.

A1.3 Macro Definitions

A1.3.1 Macro Definition Item Initialization

A macro definition is a MULTILANG item and, as such, must conform to the prescribed format for items. The first half of the procedures which perform this task are illustrated in Fig. A1.5. The given flowchart actually represents two distinct sections of the program, one for condensed forms and one not. The coding of these two sections is completely parallel, but the data comes from different places. In the case of the condensed form it is obtained by calling for MULTILANG parameters. Otherwise, it initialized and control passes to the appropriate command routine.

A1.1.3 Terminating

After all the input has been processed, a check is made to determine if any IBSYS input was generated. If not, a message to that effect is printed and IXSYS returns to the MULTILANG monitor. Assuming that there is work for IBSYS, then several additional control cards are written out, including those necessary to reload MULTILANG, with some selectivity based on whether the job is for a remote user or a regular batch user. Just prior to passing control back to IBSYS, a card for switching the input to utility 12 is stored where IBSYS will find it. The exit from IXSYS is made in such a way that remote terminal output buffers will be closed.

A1.2 Condensed Forms

If the call to IXSYS has any parameters, then it must be a condensed form of either a macro definition or a macro call (Fig. A1.4). If it is a macro definition it is handled immediately by IXSYS. If it is a macro call, it is simply noted to be handled after the initialization is completed.

A1.3 Macro Definitions

A1.3.1 Macro Definition Item Initialization

A macro definition is a MULTILANG item and, as such, must conform to the prescribed format for items. The first half of the procedures which perform this task are illustrated in Fig. A1.5. The given flowchart actually represents two distinct sections of the program, one for condensed forms and one not. The coding of these two sections is completely parallel, but the data comes from different places. In the case of the condensed form it is obtained by calling for MULTILANG parameters. Otherwise, it

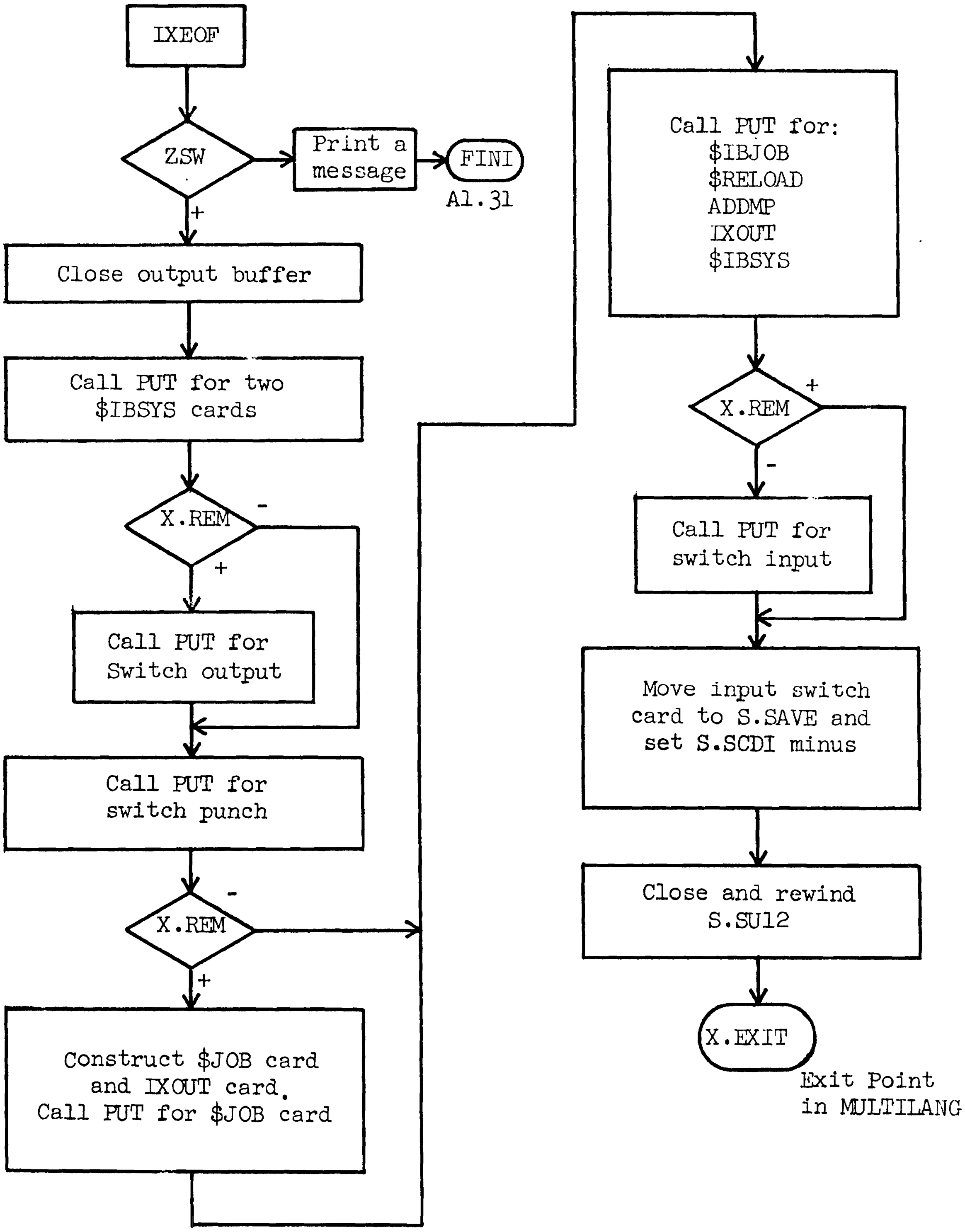

IXEOF
ZSW
Print a message
FINI
A1.31
+
Close output buffer
Call PUT for two $IBSYS cards
X.REM
-
+
Call PUT for Switch output
Call PUT for switch punch
X.REM
-
+
Construct $JOB card and IXOUT card. Call PUT for $JOB card
Call PUT for: $IBJOB $RELOAD ADDMP IXOUT $IBSYS
X.REM
+
-
Call PUT for switch input
Move input switch card to S.SAVE and set S.SCDI minus
Close and rewind S.SU12
X.EXIT
Exit Point in MULTILANG

Figure A1.3 Terminating Procedures for IXSYS

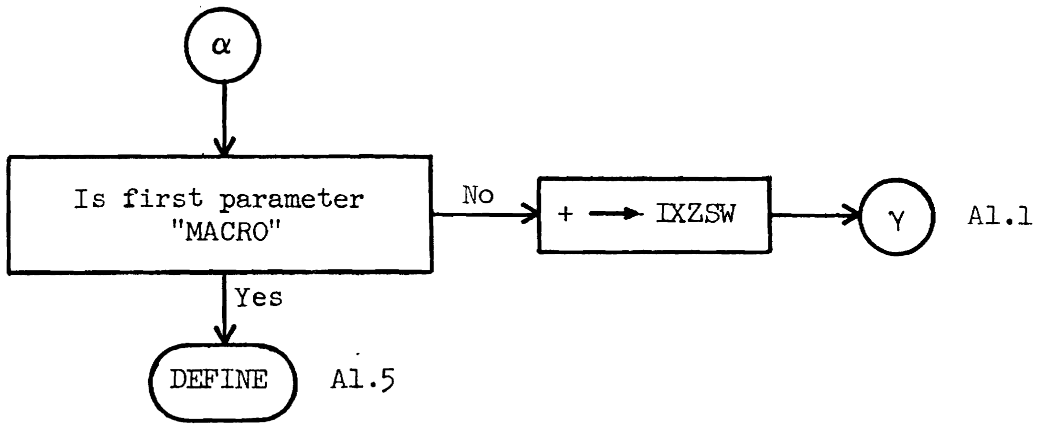


Figure A1.4 Condensed Form Determination

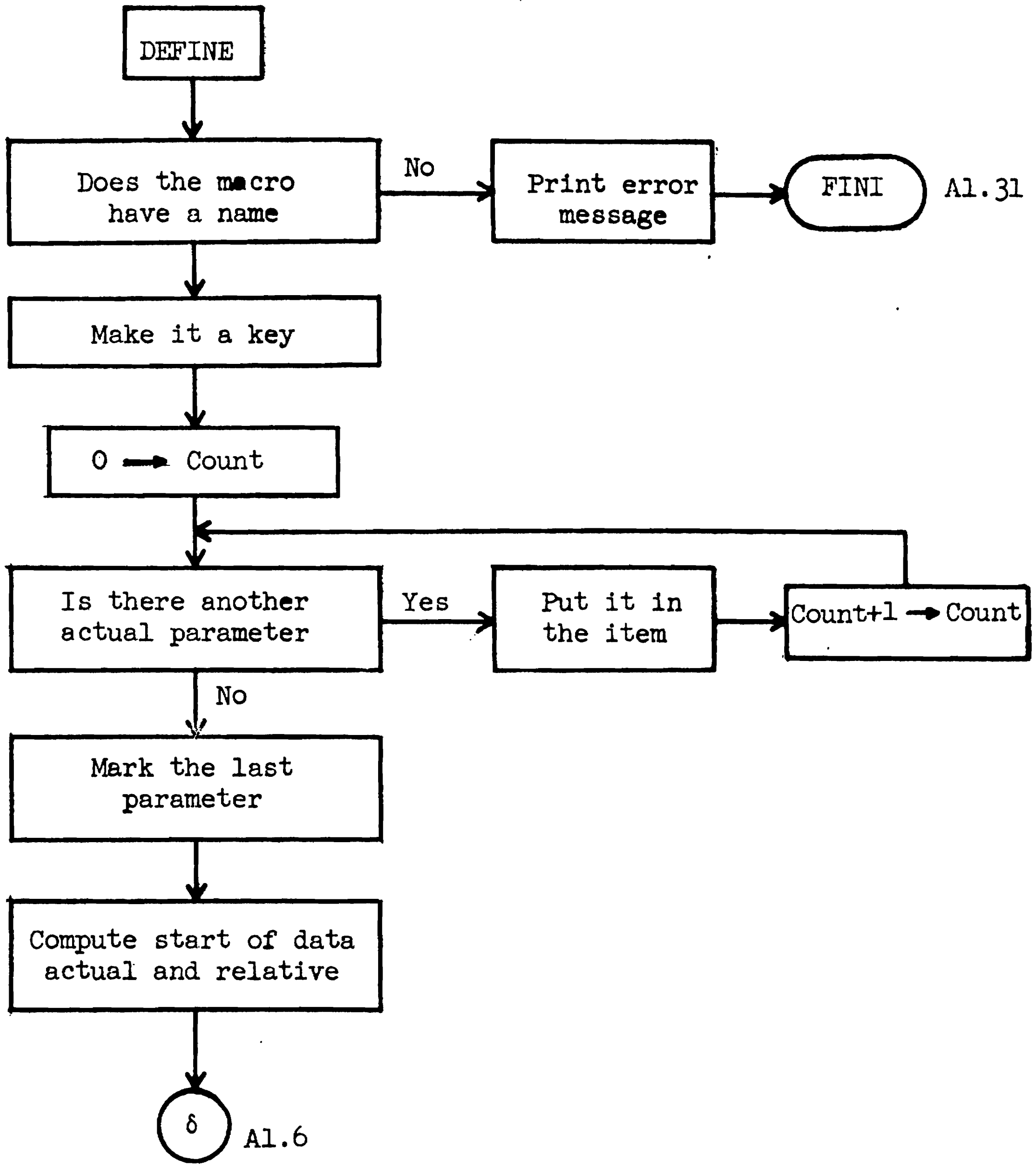


Figure A1.5 Macro Definition Item Initialization

is obtained from where the IXSYS command assembly routine puts it.

Clearly, a macro must have a name by which it is to be called. The macro name is used as a key by which it is to be stored and later retrieved. Each actual parameter is added to the item and the last one is tagged. The location where the prototype cards go is then computed from the number of actual parameters stored.

A1.3.2 Prototype Card Reading and Item Formation

Once the actual parameters have been added to the item, the prototype cards may be added. Each card is read and checked for the macro terminating card. Prototype cards are stored in the macro item and printed for the user's record. When the terminating card is encountered, the total length of the item, as well as the lengths of the various components, is known. From this data, the item header word, link word, and table of contents are computed and added to the item, thus, completing the item formation. It can then be stored by calling the MULTILANG item storage routine, and the user can be notified that the definition has been completed.

Termination from the definition routine depends on the form of the call. In the case where the condensed form was used, control return to the MULTILANG monitor. Otherwise, the space used for the item formation must be reclaimed since it is also used by the IXSYS command assembly routine. After this is done, control returns to the main read loop of IXSYS.

A1.4 Command Card Processing

The command card processing routines are divided into two groups, the input control command routines, and the retrieval and macro routines. The reason for this division is that the input control commands do not

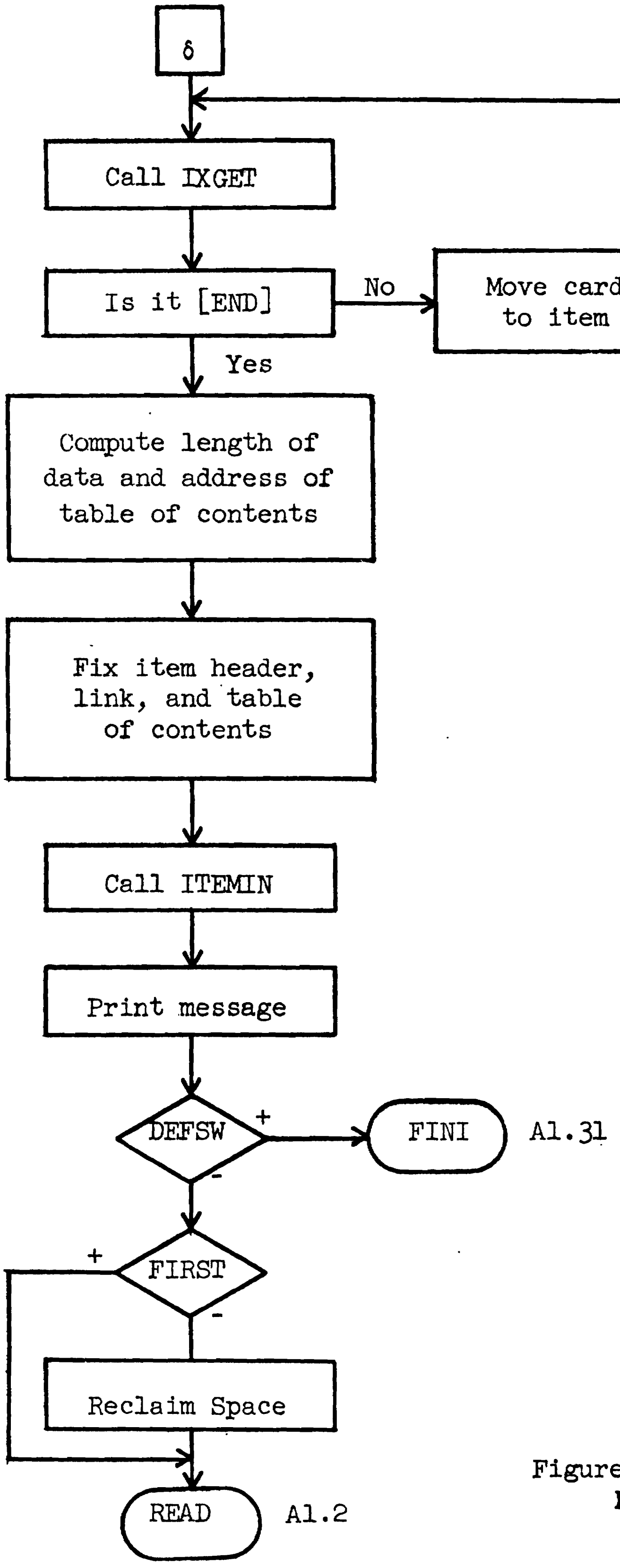


Figure A1.6 Macro Definitions Reading and Terminating

use the IXSYS command assembly routine and the others do. The input control command processing routines are all rather short, simple, and independent (except that If True and If False are combined). The others are generally longer, more complex, and highly interconnected.

A1.4.1 Input Control Command Routines

A1.4.1.1 If True and If False

The two commands If True and If False share a common processing routine. The general idea of this routine is as follows (see Fig. A1.7). First, set an indicator switch according to whether If True or If False is requested. Second, find the two components to be compared. Then, set another indicator depending on whether the two components are equal. Lastly, compare the two indicator switches for equivalence. If they are the same, skip a card; otherwise, do not skip.

More specifically, the GETCHR subroutine (Sect. A1.6.5) is called to get all the characters before the equal sign, but only up to six characters are retained for comparison. If no equal sign is encountered before the end of the card, the second component is assumed blank. Thus if the entire variable field of the card is blank, an equal condition will result.

When an equal sign is found, the next six characters are obtained for the second component. If there are less than six characters available an unequal condition is assumed, since the first component must have had more than six characters.

After both components have been obtained, they are compared for equality. If they are equal the second indicator is set and compared against the first. The nature of the comparisons and indicator settings is such that one card will be skipped if either If True was called and

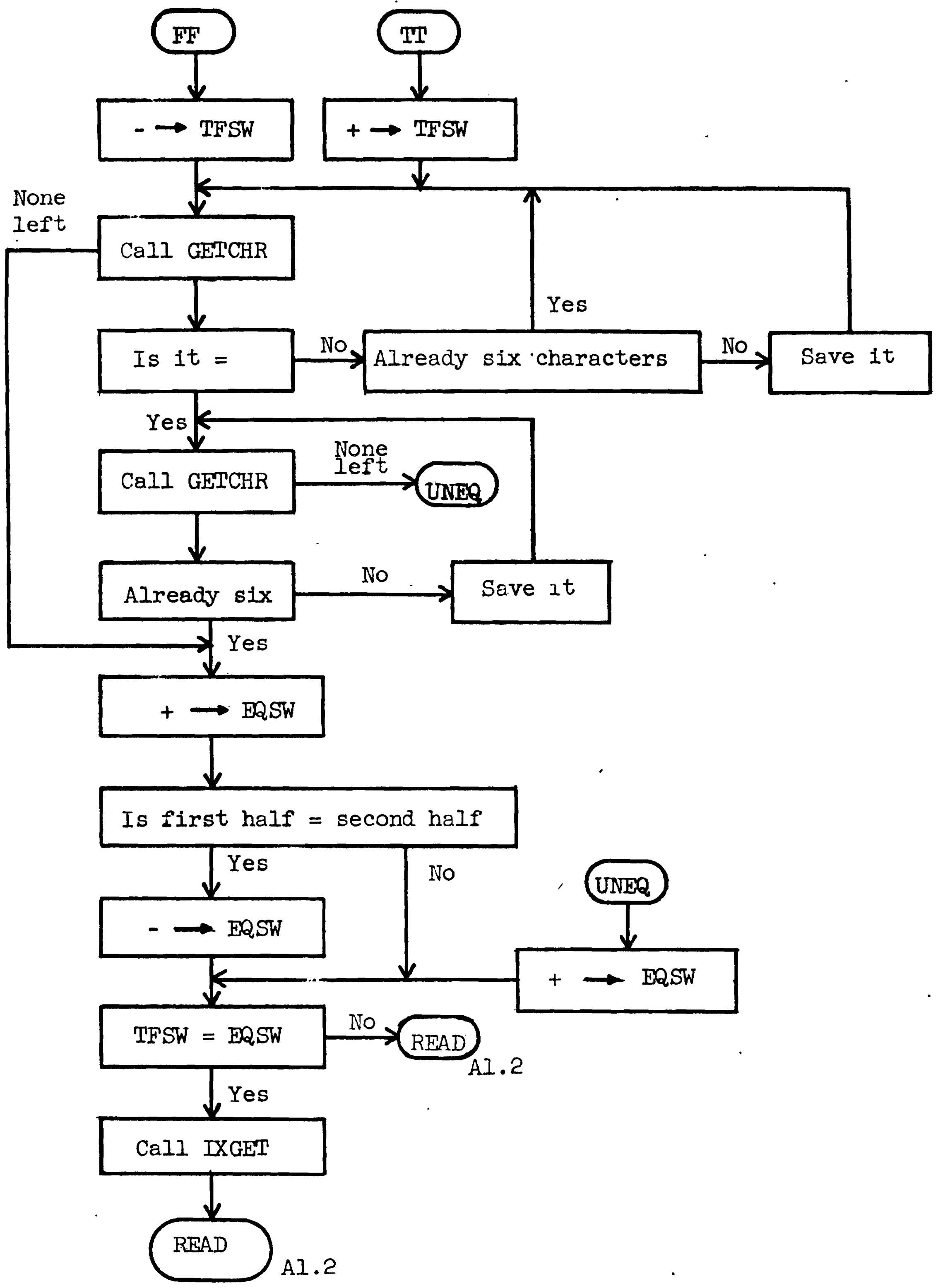


Figure A1.7 The IF TRUE/IF FALSE Routine

the components were unequal or If False was called and the components were equal.

In either case the If True/If False routine returns to the main read loop of IXSYS (Sect. A1.1.2).

A1.4.1.2 Input

The function of the Input command is quite simple - build an input control block (ICB) and put it on the pushdown list. There are two fields of the ICB which must be computed first. These are the record count and terminating mask. Both of these can be determined by considering the Input command parameter (See Fig. A1.8).

There are three possibilities for the parameter which can be directly translated into values for the record count and terminating mask. If the parameter is blank, the count is 32767 and the mask is $JOBbb. If the parameter is an octal number that number is used as the count and the mask is $JOBbb. If the parameter is anything else, the count is 32767 and the mask is the parameter.

A1.4.2 Retrieval and Macro Command Routines

The routines to be discussed in this section are distinguished by the fact that they all use the command assembly routine. In addition, they also follow a main path, which is the source retrieval routine, branching away at the appropriate points.

A1.4.2.1 Command Assembly Routine

The command assembly routine (Fig. A1.9) is an open subroutine whose responsibility it is to cause IXSYS commands containing complex parameters to be assembled into a more compact form. The actual assembly is performed by the regular MULTILANG assembler, MASS, called as a subroutine to the command assembly routine.

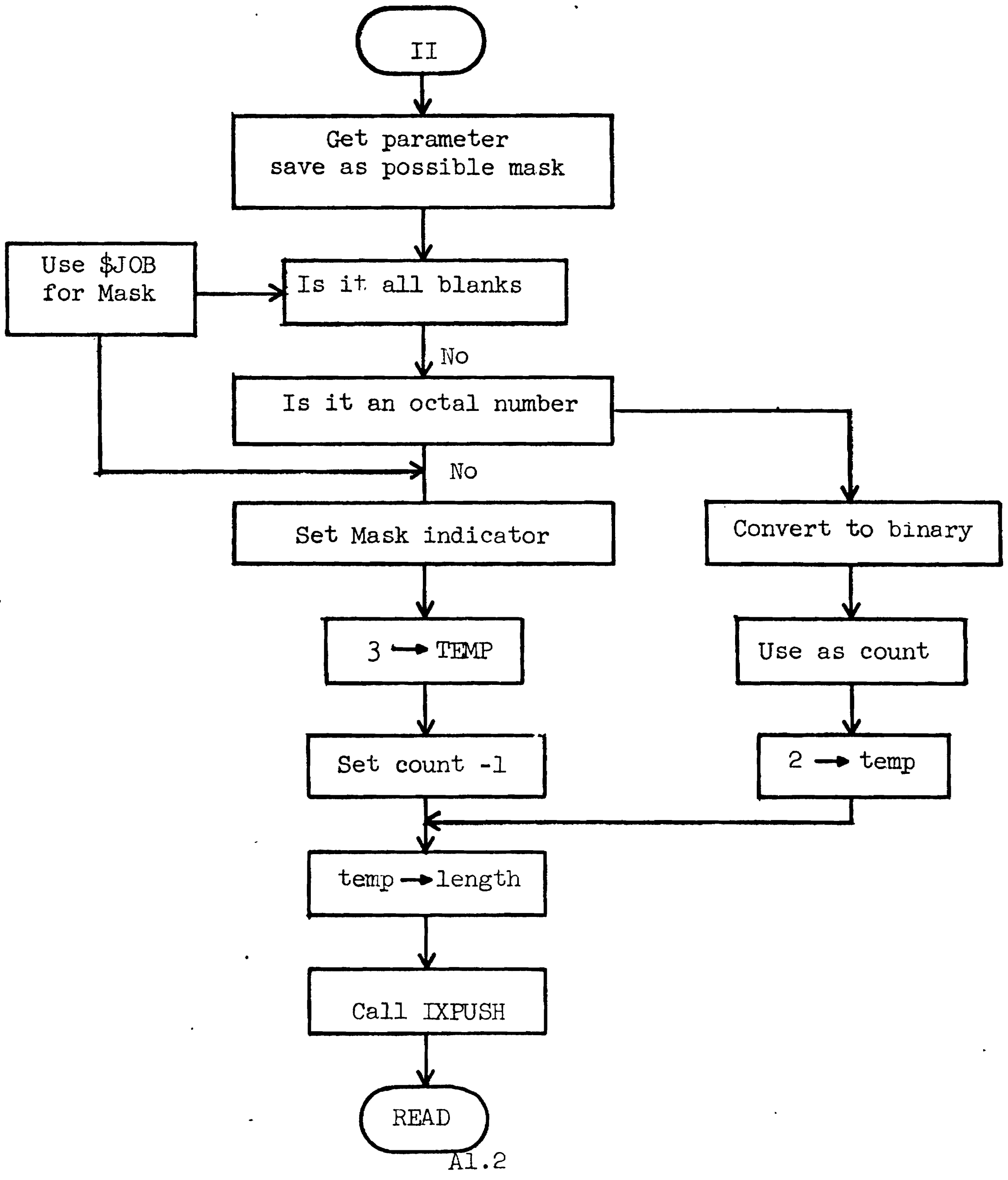


Figure A1.8 The Input Command Routine

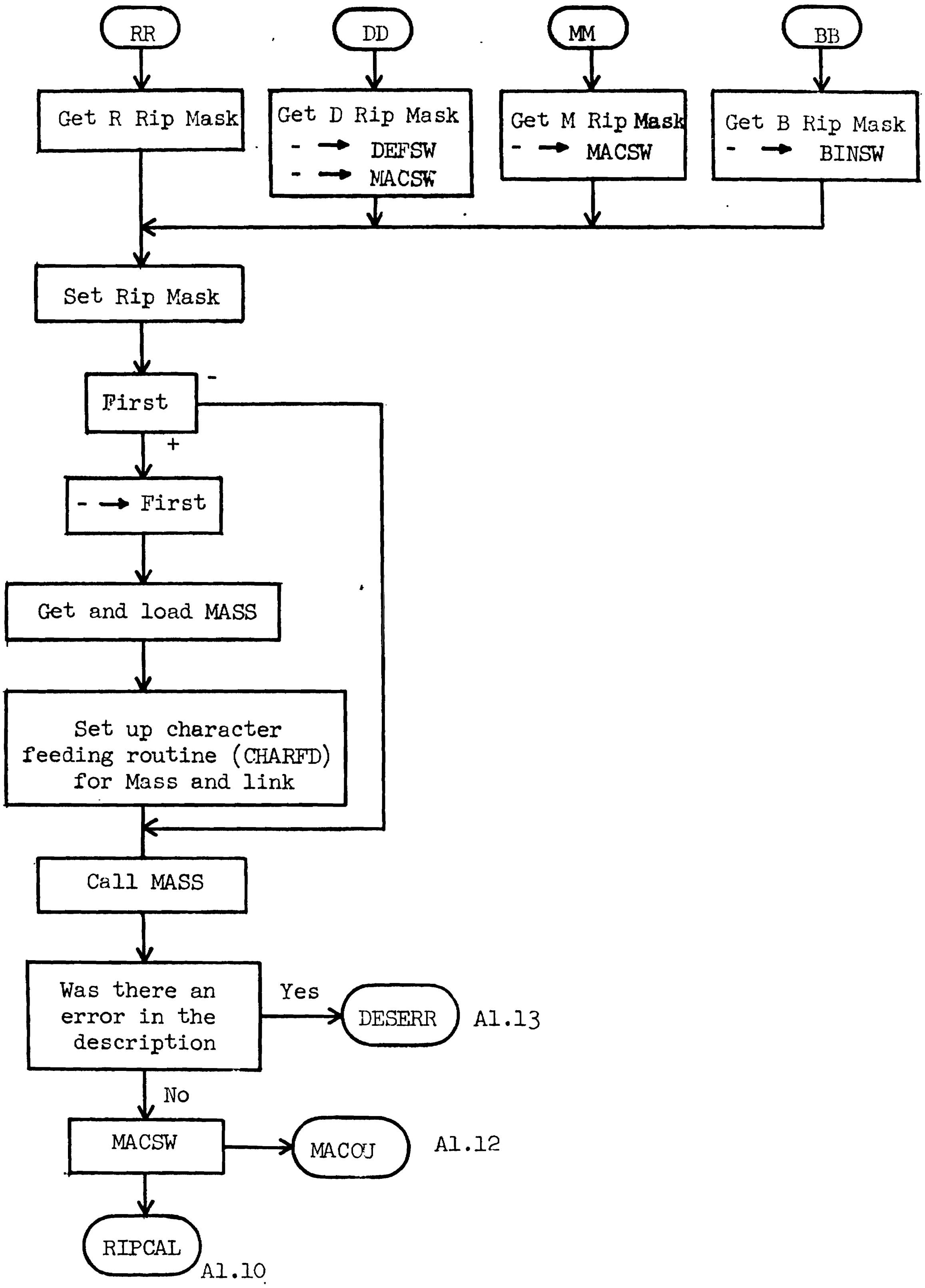


Figure A1.9 Command Assembly Routine

The command assembly routine first checks whether MASS has already been loaded once. If it has it uses it without reloading. Otherwise, the storage and retrieval system is called to retrieve MASS, and the MULTILANG loader, MLDR, is used to relocate MASS. Once MASS is in, it must be linked to routine CHARFD (Sect. A1.7) which provides the character string which MASS is to assemble. Then, MASS assembles the IXSYS command and a check is made for errors detected by MASS in the description formats.

A1.4.2.2 The Source Retrieval Routine

The source retrieval routine, in addition to executing retrieval commands, which is its primary purpose, also performs several useful functions for the other command routines described below. First, it retrieves items by descriptions supplied from the command assembly routine (Sect. A1.4.2.1), and verifies that there in fact was at least one item found with that description. It also verifies that the item has data in the proper place and computes the address and length of that data. Lastly, it outputs the data, one line at a time and returns to the top of the routine to look for more items matching the given description.

A1.4.2.3 The Binary Retrieval Routine

The binary retrieval routine follows the source retrieval routine as far as the verifying that an item has been retrieved (see Fig. A1.10). It must then verify that the data retrieved is indeed a binary deck. After this it need only write the deck out, but this process is complicated by the fact that a binary deck includes both binary and BCD cards. The BCD cards are distinguished by the fact that they all start with a dollar sign. Note also that binary decks are not stored with the usual visible sequence punches. Consequently, blank sequence columns must be provided to insure proper card length.

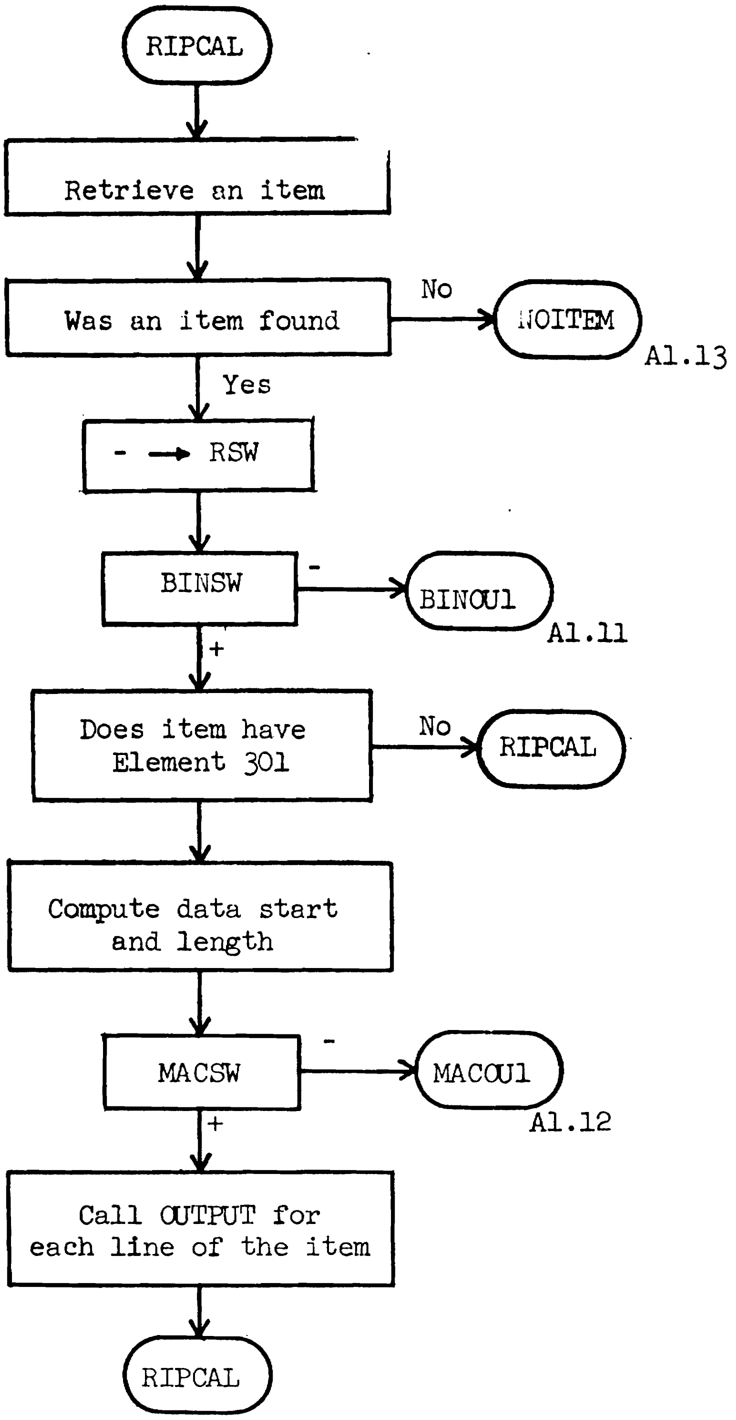


Figure A1.10 Source Retrieval Routine

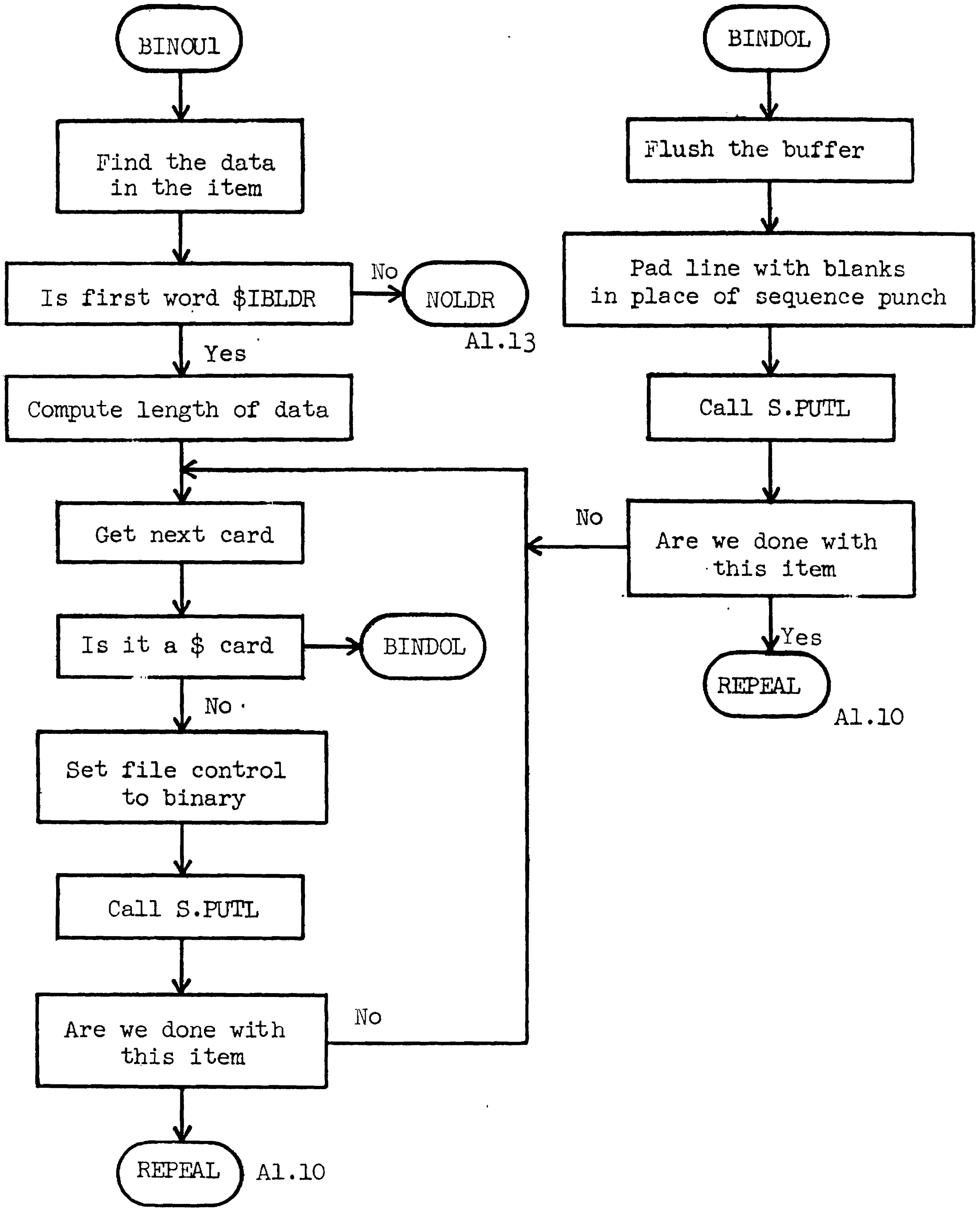


Figure A1.11 Binary Retrieval Routine

A1.4.2.4 The Macro Retrieval Routine

Unlike the source and binary retrieval routines, the macro retrieval routine does not actually output any card images. Rather, after the macro has been retrieved, an input control block (ICB) is constructed which will cause the macro to be read whenever IXGET is called.

All the activity of the macro retrieval routine is directed toward constructing the ICB which will be used to read the macro. There are two essential types of data which must go into the ICB, information concerning the address and length of the macro and information defining any actual parameters.

After the macro command has been assembled by the command assembly routine (Sect. A1.4.2.1), the assembled form, which contains the actual parameters plus a local table of contents (LTC), is put into an ICB prototype buffer. This buffer, which contains space for the additional required data, is then put on the ICB pushdown list. After the macro has been retrieved the necessary addresses and record count are added to the ICB on the pushdown list.

A1.4.2.5 Error Routines for Retrieval Command Routines

There are three errors which can result from an improper description in a retrieval command (See Fig. A1.13). First, something may be wrong with the punctuation, choice of characters, etc., which will result in MASS not being able to assemble the description (DESERR). Second, there may be no items which fit the given description (NOITEM). Third, a binary item may not be a relocatable binary dec (NOLDR). For each of these errors an appropriate message is printed for the user, and the error is otherwise ignored.

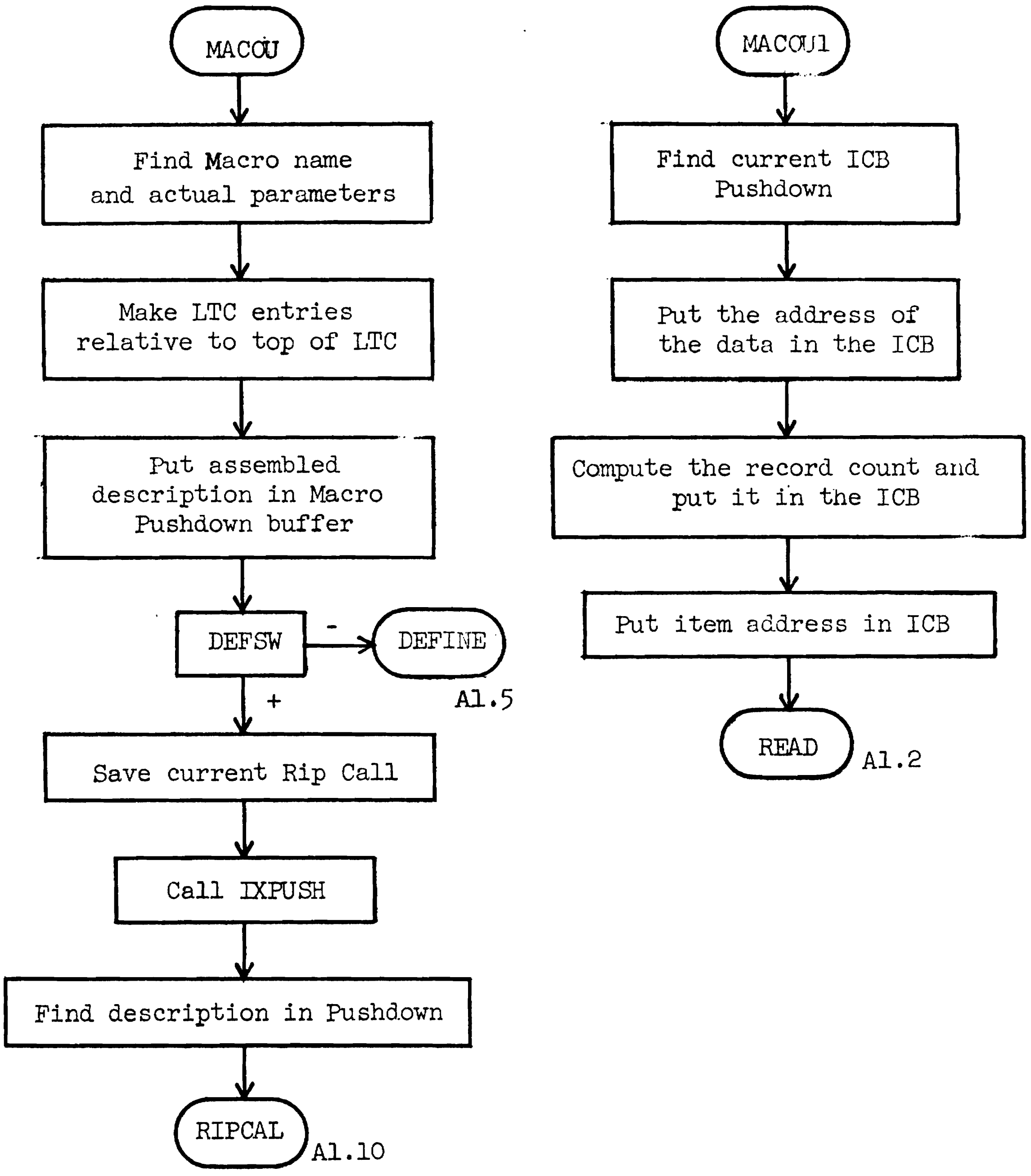

MACOU
Find Macro name and actual parameters
Make LTC entries relative to top of LTC
Put assembled description in Macro Pushdown buffer
DEFSW
-
DEFINE
A1.5
+
Save current Rip Call
Call IXPUSH
Find description in Pushdown
RIPCAL
A1.10
MACOU1
Find current ICB Pushdown
Put the address of the data in the ICB
Compute the record count and put it in the ICB
Put item address in ICB
READ
A1.2

Figure A1.12 Macro Retrieval Routine

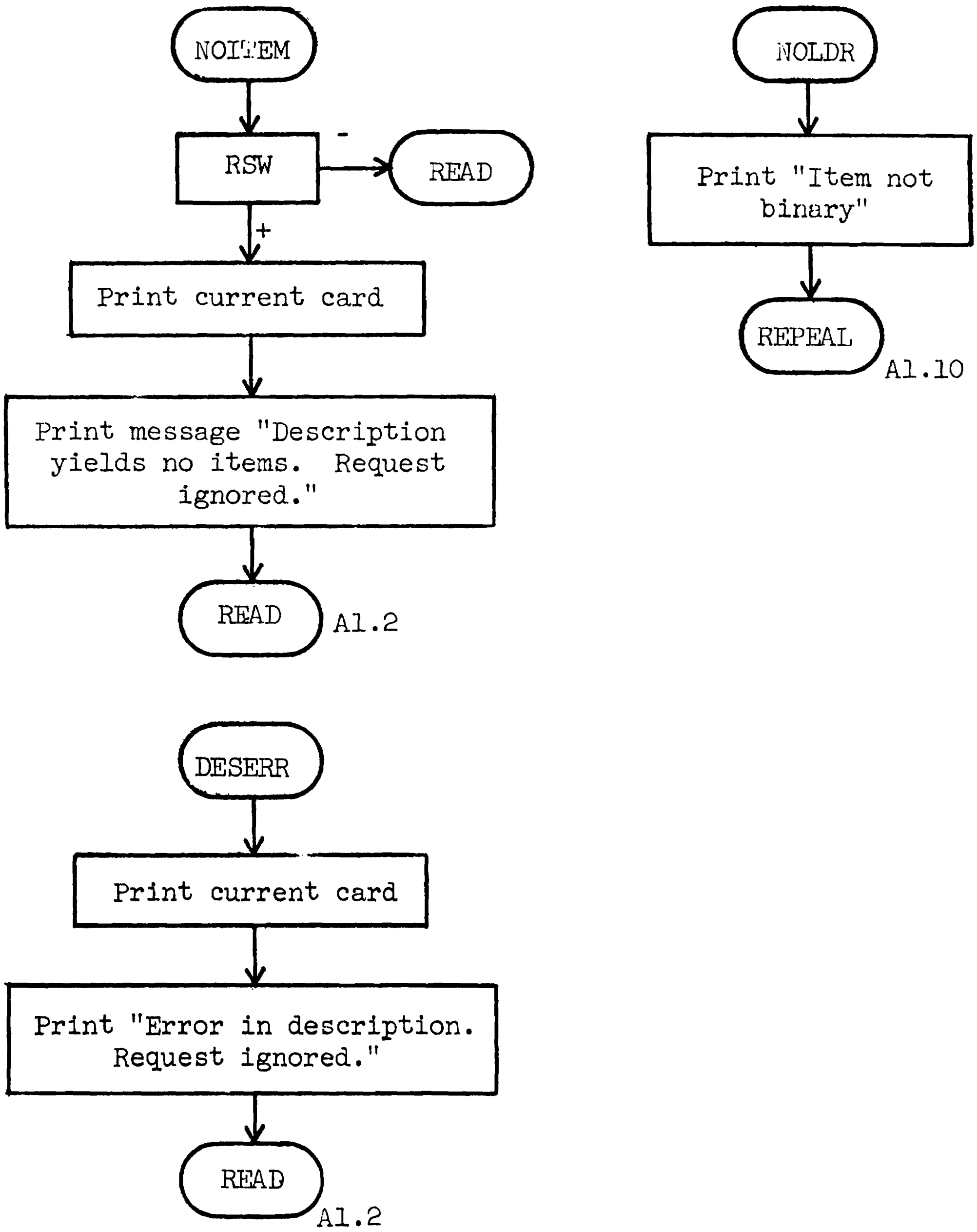


Figure A1.13 Error Routines for Retrieval Command Routines

A1.5 The Generalized Input Control Routines

Because of the possibility of nesting IXSYS macros and interlacing them with Input commands, it is necessary that some central input routine be responsible for handling the sequencing of the several sources of input. This sequencing control is made possible with the use of a group of devices.

Central to this group is the Input Control Block (ICB) pushdown list. Each time a new source of input is called for, an ICB is constructed and placed on the pushdown. Within the ICB is all the information required to allow reading of the new input source, including location of data, number of card images to be read, and a control mask to allow a variable number of card images. Thus input can be read according to this ICB until all of it has been read. Then the pushdown is popped and reading continues from the previous source. The pushdown is initialized to start reading the user's input file.

Input from an input file and input from a macro must be read from different sources. Also, macro input requires preprocessing for parameter substitution before control returns from the main input routine. In order to account for these differences, each ICB contains a pointer to a select routine which is responsible for knowing how to handle the particular input source.

Similarly, when an input source has been exhausted, an end of file routine must be called which knows how to handle the situation. Each ICB, thus, contains a pointer to the appropriate end of file routine.

A1.5.1 IXGET

The main section of the generalized input control routine is called IXGET. IXGET must first locate the ICB at the top of the pushdown and get the current record count from it (See Fig. A1.14). If the record count has gone to zero the end of file exit is taken. Otherwise, the count is reduced by one. Next the address of the select routine is found in the ICB and control is transferred to it. When the select routine returns to IXGET, a check is made to determine if the select routine detected an end of file. If so, the end of file exit is taken. If not, the ICB is checked to determine if a mask check is called for. If so the first six characters of the card read are compared with the mask found in the ICB. If a match is found, this condition is treated like an end of file. If either no mask check is called for or a match does not result, then the card image is moved to a common buffer where all other routines will operate upon it. If the select routine indicated that the card is to be checked for parameter substitution, this is done next. IXGET finally exits by returning to the calling program.

A1.5.2 The Pushdown Control Routines

There are two subroutines, IXPUSH and IXPOP, which are responsible for maintaining the ICB pushdown list. (See Fig. A1.15) There are three locations which these routines must maintain. The current pushdown pointer is used by many routines to find the current ICB. The current ICB length is used by IXPUSH to determine where a new ICB goes in the pushdown. The previous pushdown pointer is stored in the new ICB to allow IXPOP to restore it when deleting an ICB from the pushdown. When adding an ICB to the pushdown its length must be supplied to IXPUSH by the caller. When popping IXPOP computes the new current length from the difference between

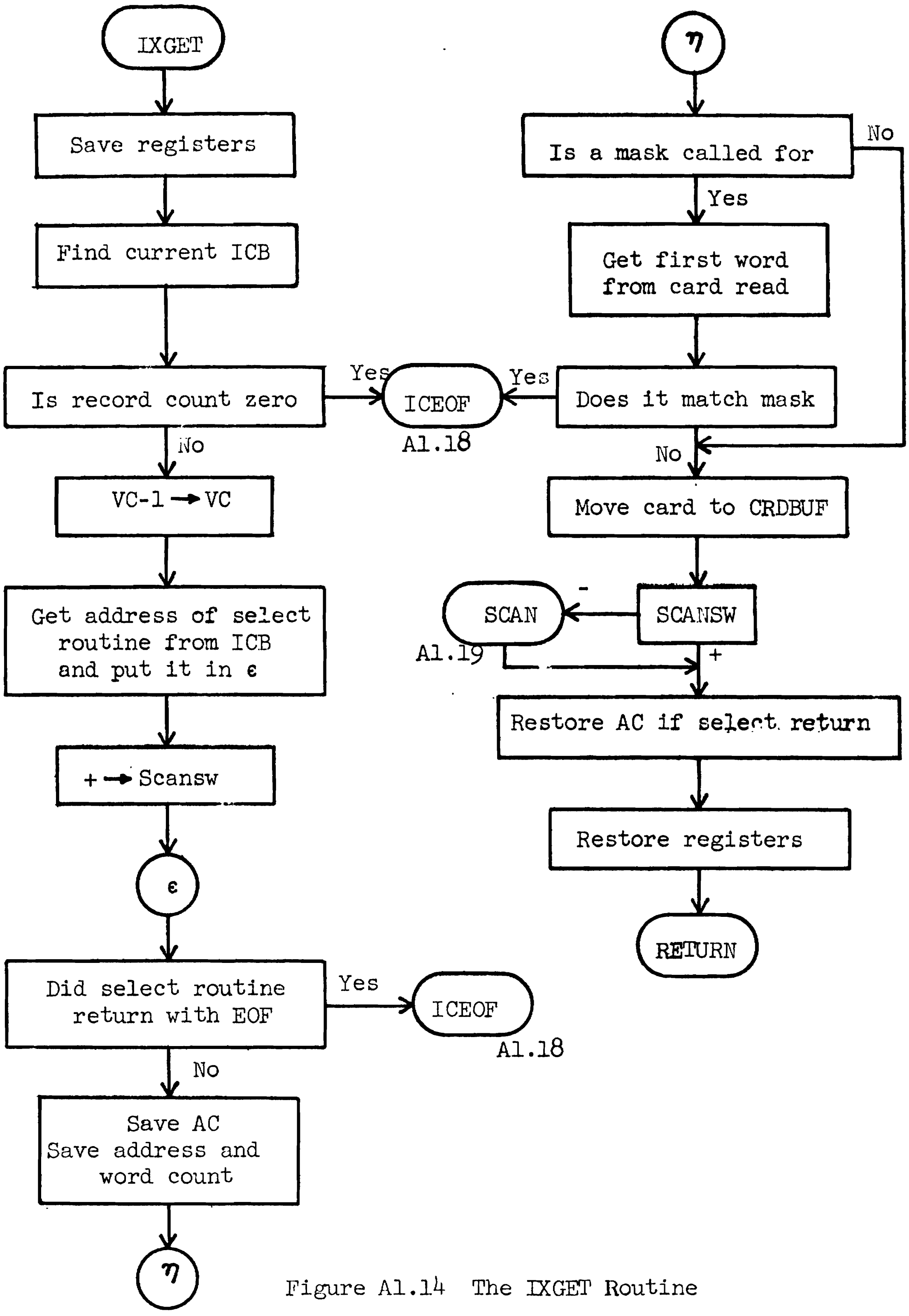


Figure A1.14 The IXGET Routine

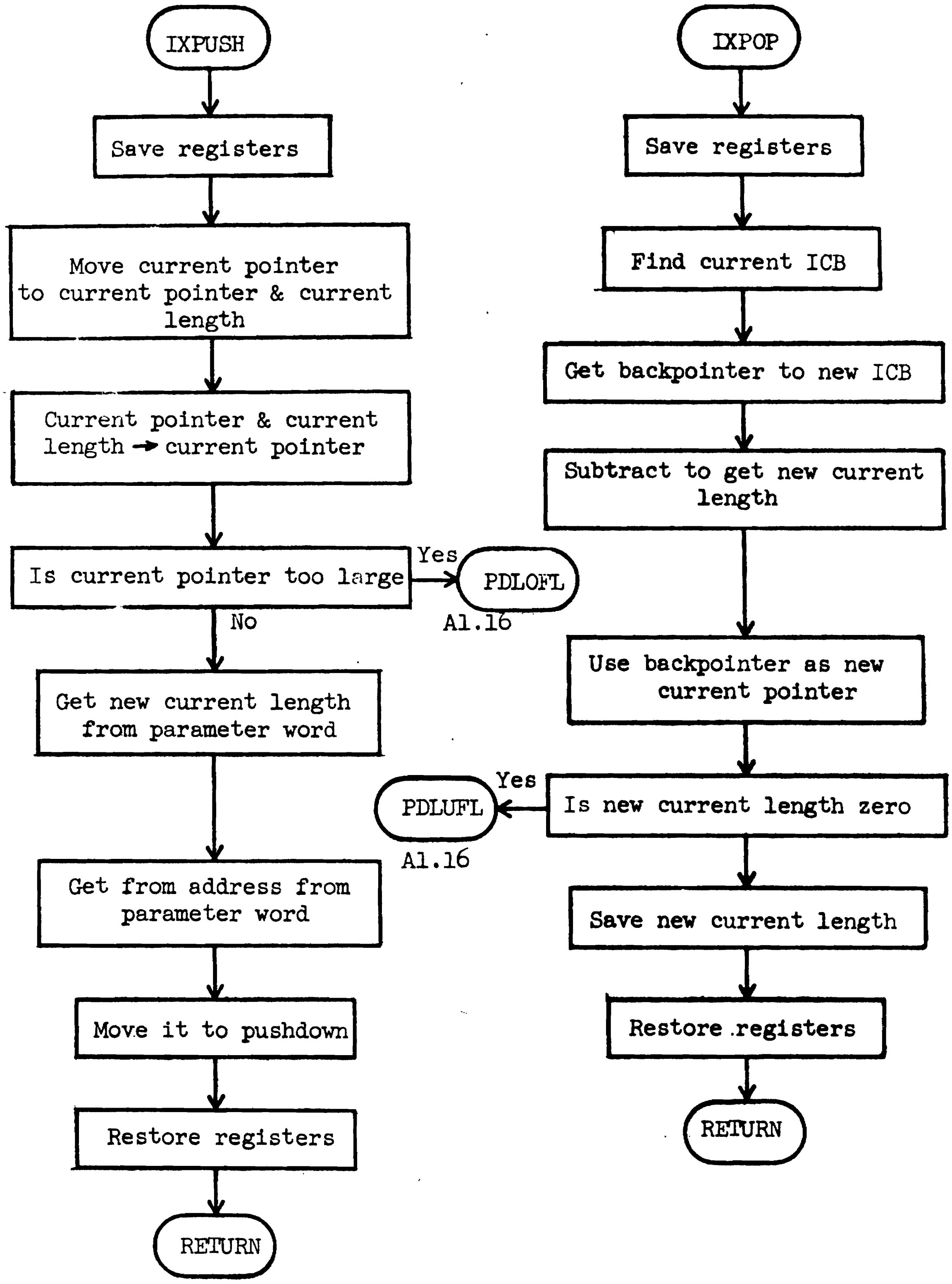

IXPUSH
Save registers
Move current pointer to current pointer & current length
Current pointer & current length → current pointer
Is current pointer too large
Yes
PDLOFL
A1.16
No
Get new current length from parameter word
Get from address from parameter word
Move it to pushdown
Restore registers
RETURN
IXPOP
Save registers
Find current ICB
Get backpointer to new ICB
Subtract to get new current length
Use backpointer as new current pointer
Is new current length zero
Yes
PDLUFL
A1.16
Save new current length
Restore registers
RETURN

Figure A1.15 The Pushdown Control Routines

the old and new pushdown pointers.

A1.5.3 Select Routines

The select routines are called by IXGET and are responsible for reading cards, or otherwise locating them, and informing IXGET of cards address and word count. They may also indicate that instead of a card being read, that an end of file was read. The select routine is also responsible for indicating that a card is to be scanned for macro parameter substitutions.

Two select routines, MACSEL for macros and MULSEL for input files, are shown in Fig. A1.17. MACSEL finds cards in a macro item, updating the current record address each time it gets a card. MULSEL gets cards by calling the MULTILANG input routine, MULIN. Note that if MULIN returns an end of file indicator to MULSEL, MULSF" will pass this on to IXGET and will always indicate an end of file thereafter. This is necessary because calling MULIN after it has returned an end of file will result in the job being terminated immediately.

A1.5.4 End of File Routines

When a select routine returns to IXGET with an end of file indication, or when an ICB record count is zero, or when a card is read which matches an ICB end of file mask, IXGET takes the end of file exit. First, control transfers to IXEOF (see Fig. A1.19). ICEOF must then find the pointer to the appropriate end of file routine in the current ICB.

If the pushdown has only one ICB on it (the original for reading input) when an end of file is encountered, then IXSYS is all finished reading input so the end of file routine for this ICB is IXEOF (Fig. A1.3).

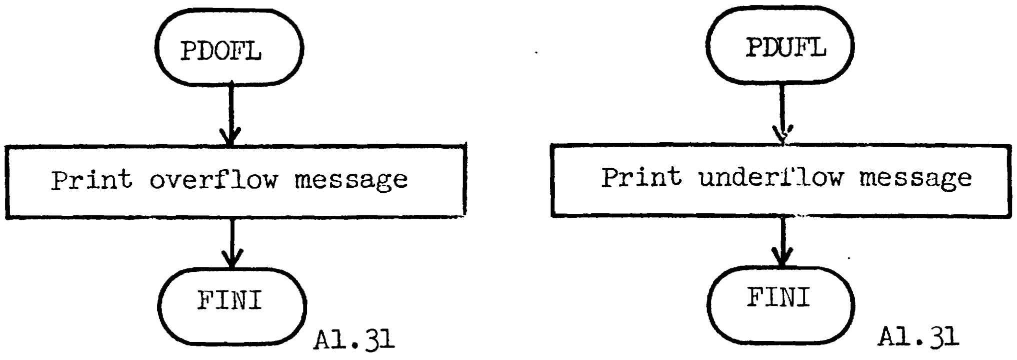


Figure A1.16 Pushdown Error Routines

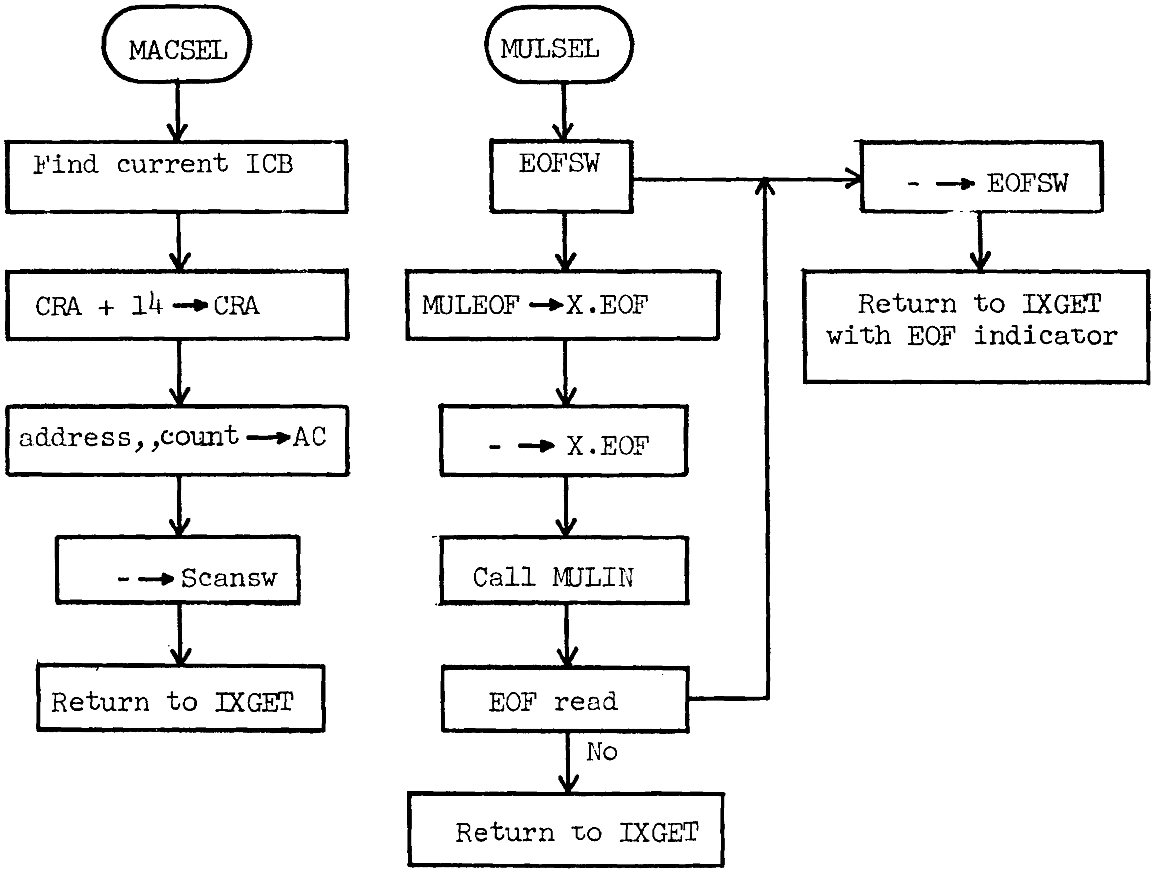


Figure A1.17 Select Routines for Macros and Input Files

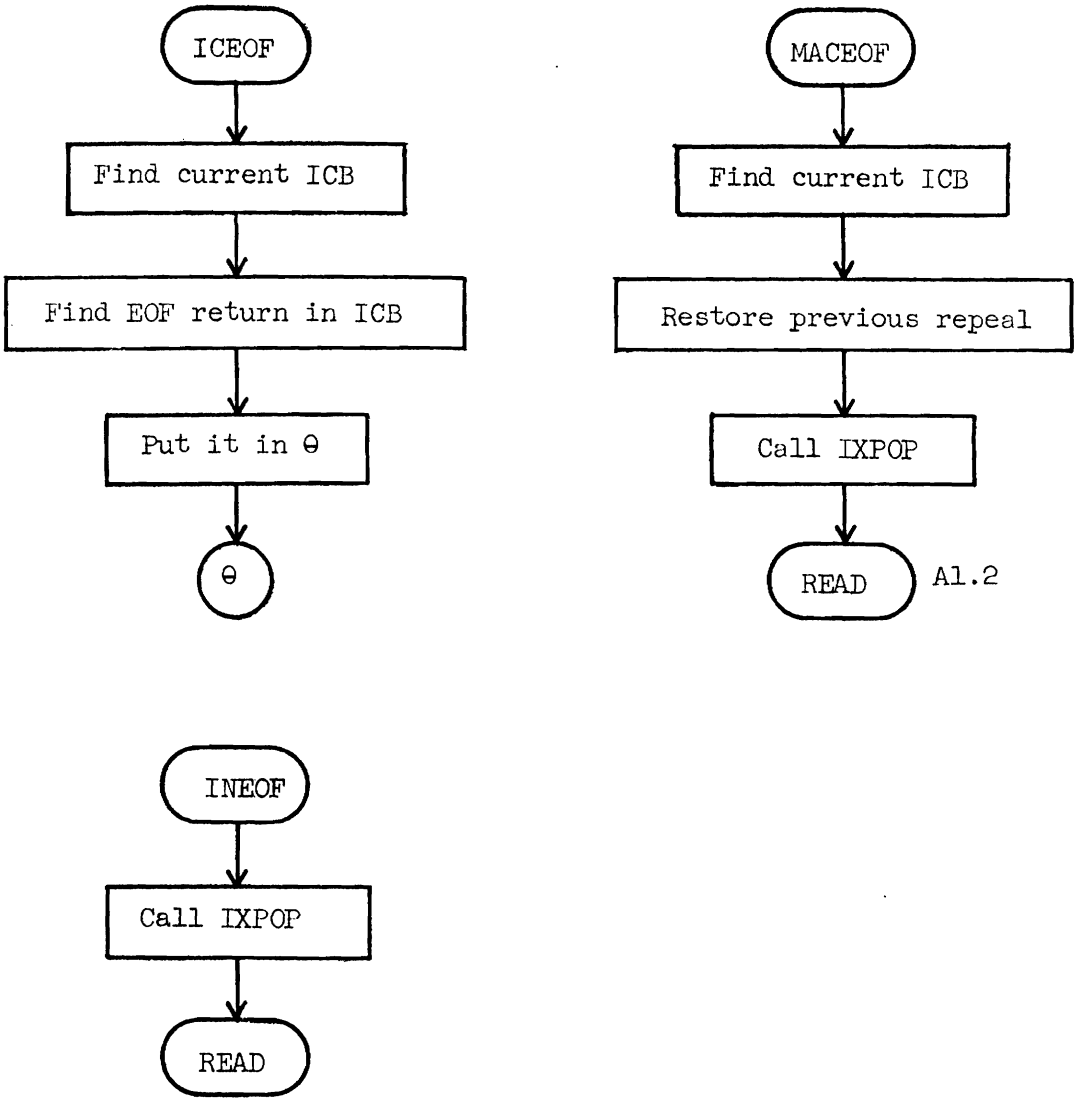


Figure A1.18 End of File Routines

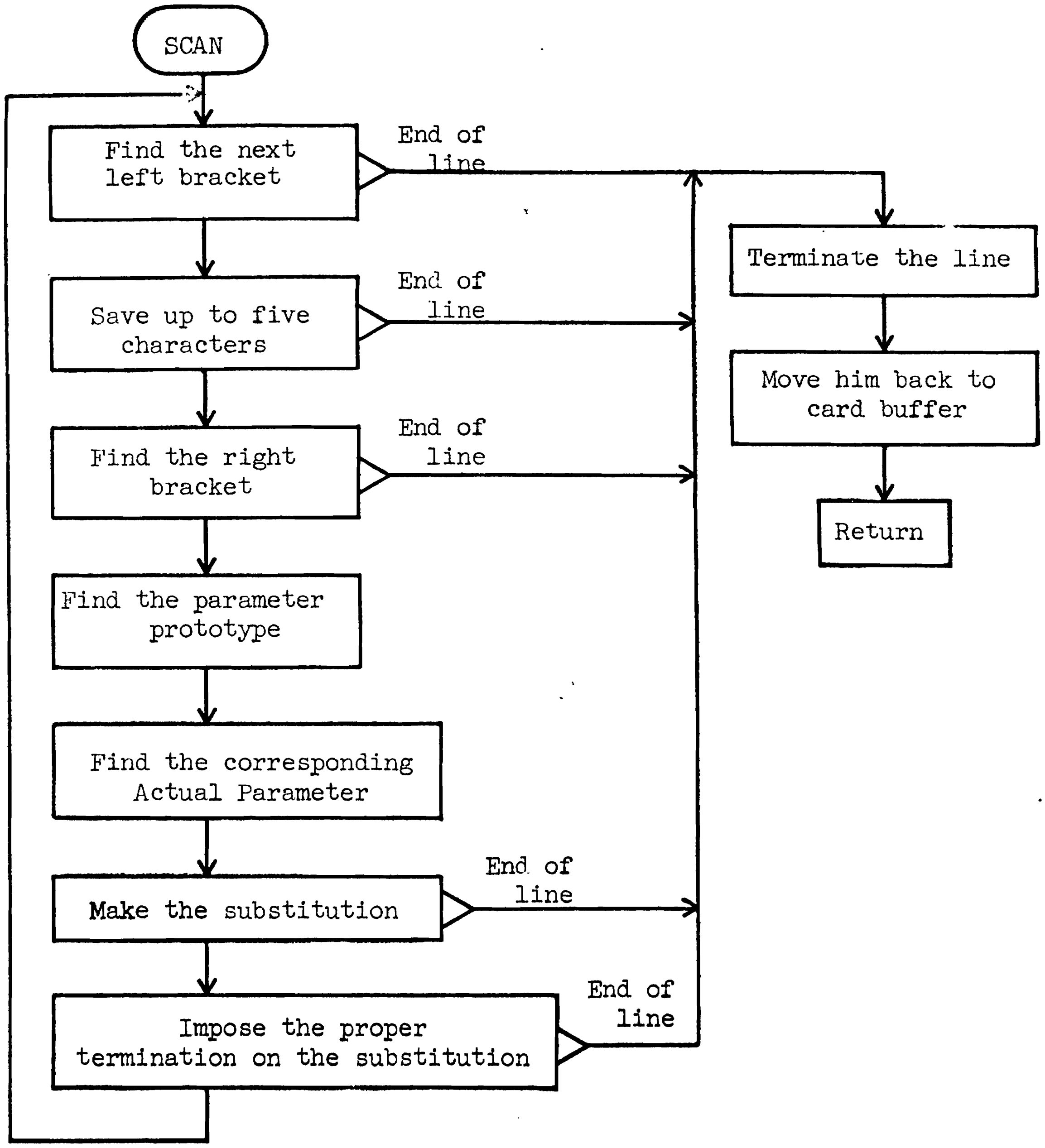

SCAN
Find the next left bracket
End of line
Save up to five characters
End of line
Find the right bracket
End of line
Find the parameter prototype
Find the corresponding Actual Parameter
Make the substitution
End of line
Impose the proper termination on the substitution
End of line
Terminate the line
Move him back to card buffer
Return

Figure A1.19 Flow Diagram of Macro Parameter Substitution Routine

Otherwise, if either an input ICB or a macro ICB has been placed on the pushdown, control will pass to either INEOF or MACEOF (Fig. A1.18).

A1.6 The Macro Parameter Substitution Routine

The macro parameter substitution routine is the largest single section of IXSYS. A general flow diagram for this routine, called SCAN, is given in Fig. A1.19. The basic process of substituting a parameter may be broken down into the following four steps:

1) Locate a possible candidate. This means find a left bracket and a closing right bracket and save up to five of the initial characters of the enclosed string.
2) Verify and identify the parameter by finding the identical string on the formal parameter list for the macro and noting which one it is.
3) Find the corresponding actual parameter. The actual parameters are saved in the ICB pushdown list. This process also calls for identifying null actual parameters and calling for inclusion of the formal parameter.
4) Make the substitution. This includes accounting for fixed length fields.

The complete process consists of looping on the four basic steps until some point in the sequence encounters the end of a line, either the given line or the formed line.

A1.6.1 Locating a Possible Formal Parameter

The line in which parameter substitutions are to be made is scanned from left to right with each character being moved to a merge buffer, until a left bracket is encountered (see Fig. A1.20). A formal parameter must be enclosed in brackets, so if a left bracket is found its position on

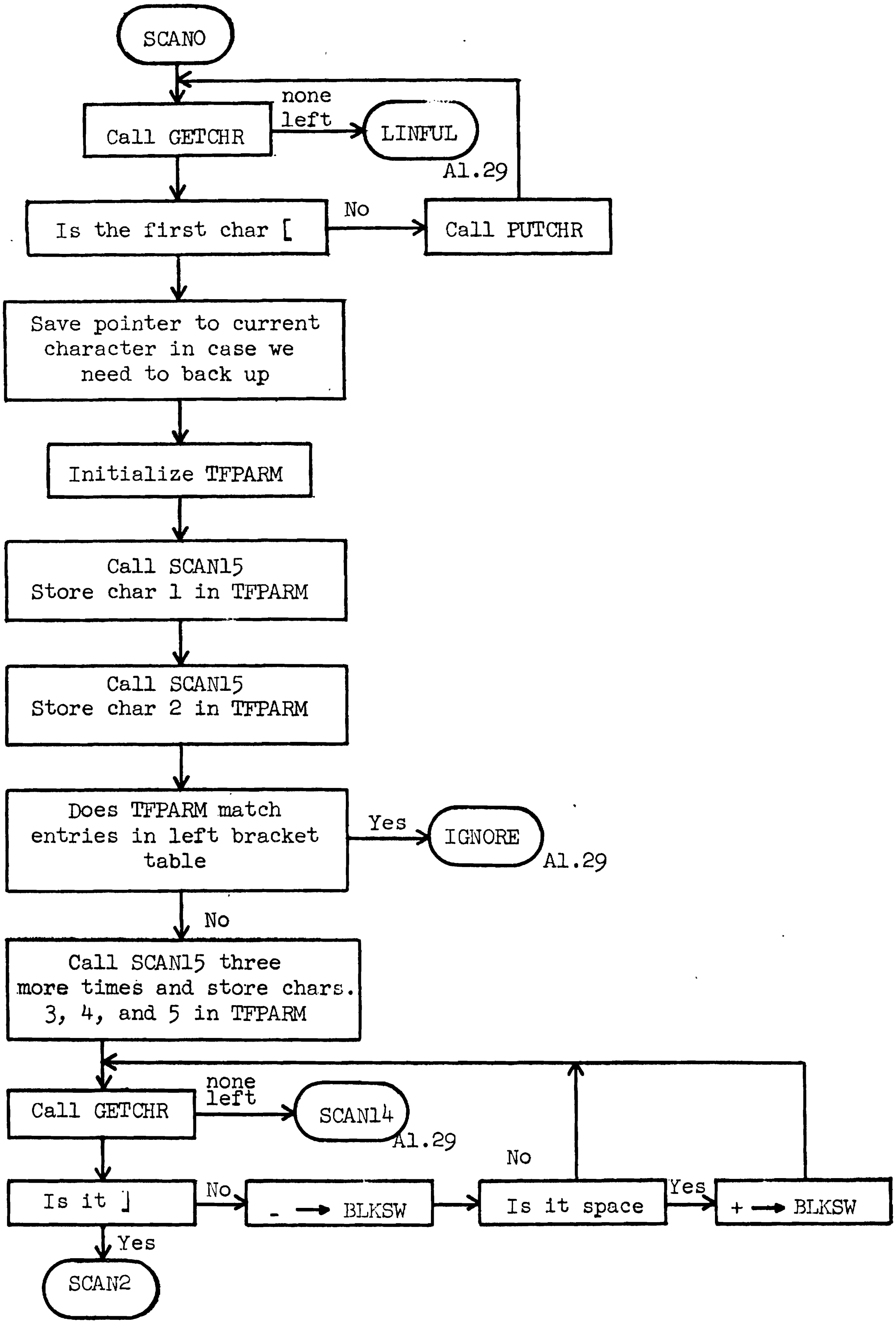


Figure A1.20 Finding a Formal Parameter Candidate

the line is noted and a right bracket is sought. As additional characters are fetched they are saved until five have been accumulated. The first two characters are checked against a list of IXSYS command forms, and the scan for that parameter terminates if a match is found.

When the right bracket is located note is made as to whether it was preceded by a blank to identify a fixed length field.

The characters accumulated between the brackets (up to five) are then saved as the formal parameter candidate.

A1.6.2 Identifying a Formal Parameter

A string of characters enclosed in brackets on a macro prototype card is not necessarily a formal parameter. It must also match some string on the formal parameter list included in the macro definition. This list is contained in the macro item, the address of which is saved on the ICB pushdown (see Fig. A1.21). Once the list of formal parameters is located the parameter candidate is compared against each member of the list until a match is found. The position of the matching member on the list is noted since this position will identify the corresponding actual parameter.

A1.6.3 Finding the Corresponding Actual Parameter

There are two ways to locate actual parameters in a macro call, depending on the form of the call. For a regular call, the parameters have been saved in the ICB on the pushdown (see Fig. A1.22). In this case the parameters are pointed to by a local table of contents (LTC) which is scanned from top to bottom to make sure that there are enough actual parameters. Once the proper LTC entry is obtained it must be examined for the proper type. The only allowed types are prefix five (a block of constants) and prefix six (a single constant), except that prefix zero is

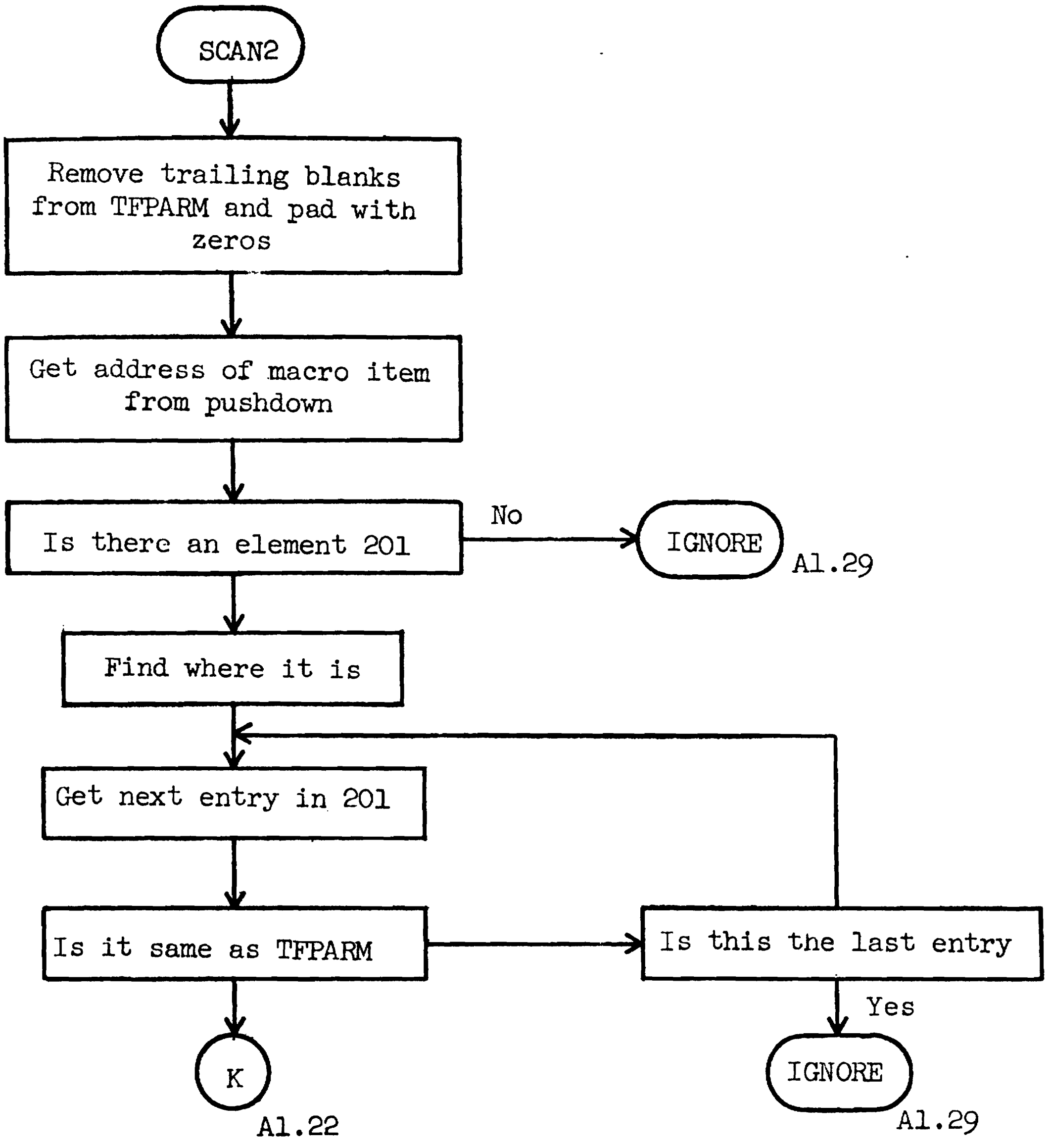


Figure A1.21 Identifying a Formal Parameter

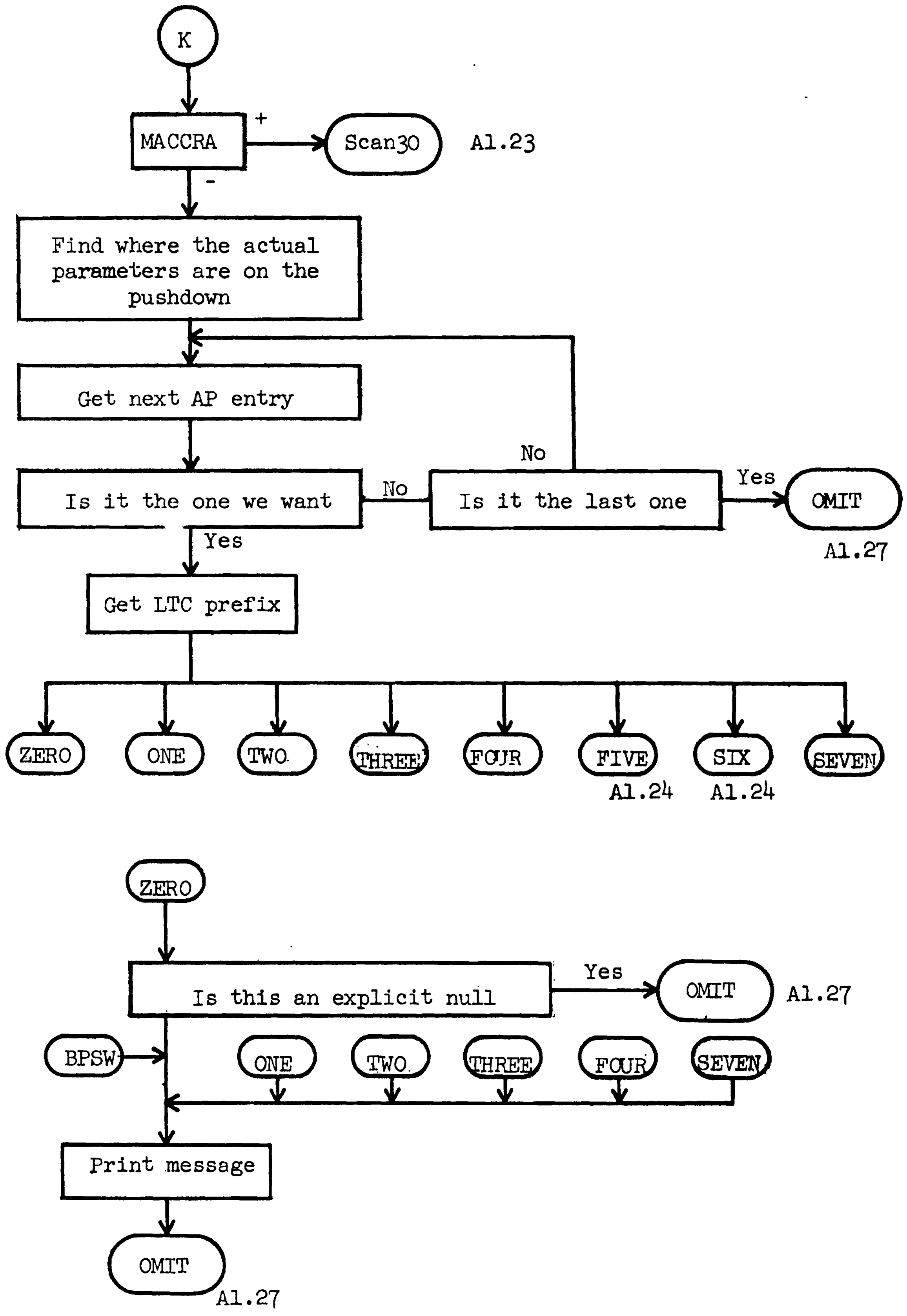


Figure A1.22 Finding The Corresponding Actual Parameter

acceptable provided the parameter is explicitly null. All other prefixes are considered errors and are treated as null.

For a condensed form macro call, the parameter list is maintained by the MULTILANG retrieval initiation program (RIP). The parameter number desired is used directly in the call to RIP (see Fig. A1.23), and RIP return with all necessary indicators and pointers to be used directly.

A1.6.4 Making the Substitution

Once the actual parameter has been obtained, it must be moved to the merge buffer. A subroutine SCNGET is called (see Fig. A1.24) to get one character from the actual parameter and put it in the merge buffer. As each character is moved in this way a character of the formal parameter is checked until a right bracket is found. An indicator keeps track of which runs out first, the formal parameter or the actual parameter.

After the right bracket of the formal parameter is reached the question of whether or not it was preceded by a blank arises (Fig. A1.25). If not, then if more actual parameter characters need to be moved to the merge buffer, they are. If the right bracket is preceded by a blank, then the field must be right justified to the column of the right bracket. This may mean either moving the merge buffer pointer back to the column of the right bracket, or getting more actual parameter characters or blanks to fill the field.

A1.6.5 Miscellaneous SCAN Subroutines

SCNGET

The SCNGET subroutine (Fig. A1.26) is responsible for moving actual parameter characters to the merge buffer. It must be primed with the address and word count of the actual parameter. It keeps track of where it is and each time it is called it gets one more character and puts it

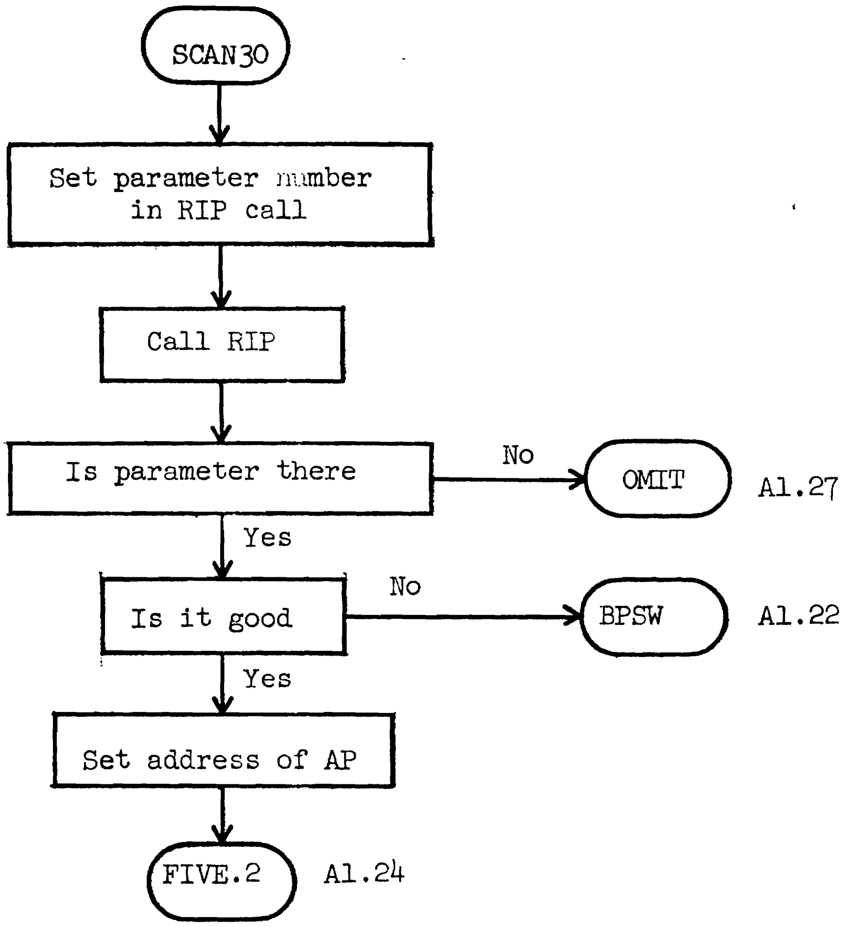


Figure A1.23 Sources of Actual Parameters In A Condensed Form Macro Command

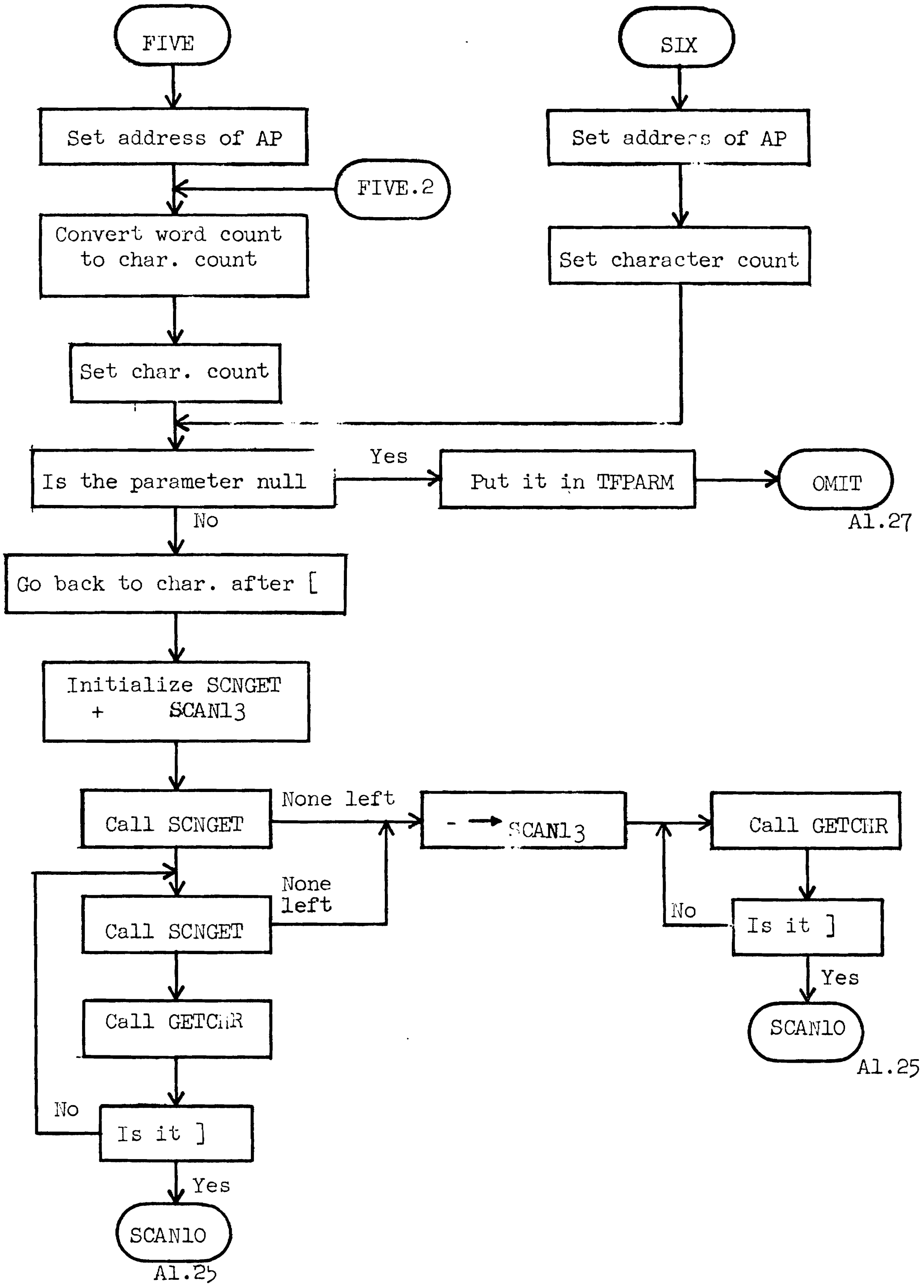


Figure A1.24 Parameter Substitution Phase

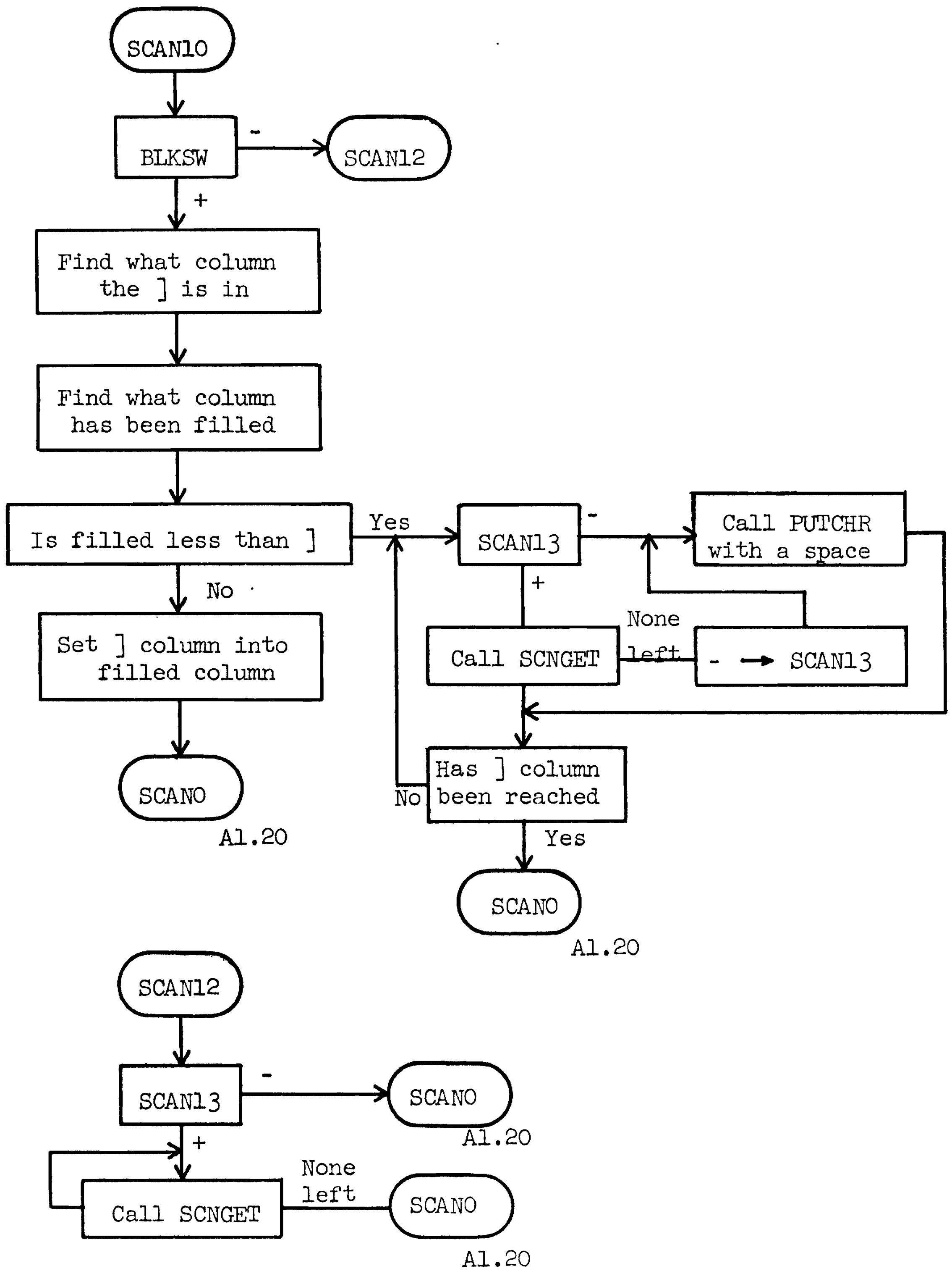


Figure A1.25 Parameter Termination Routines

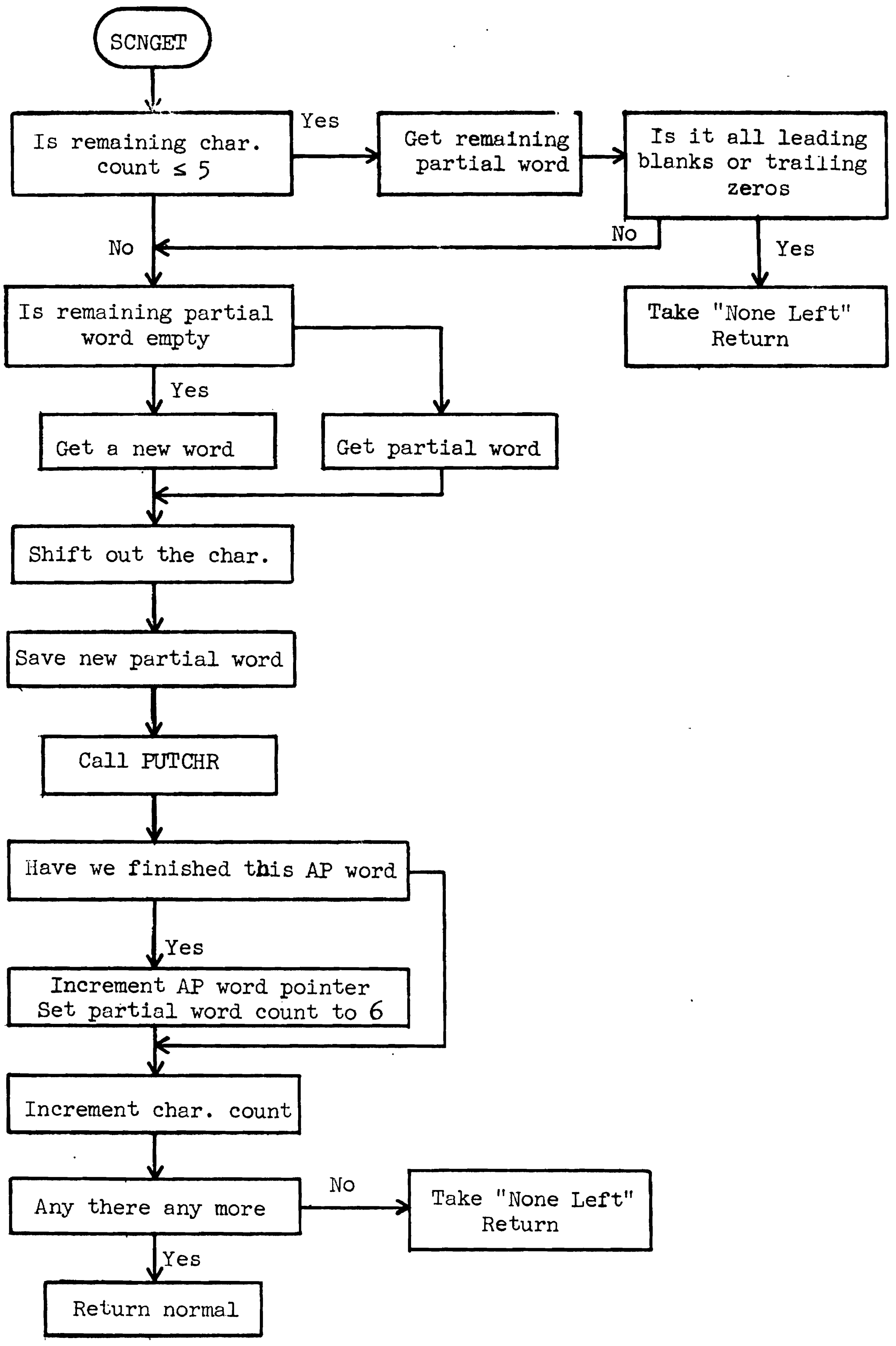

SCNGET
Is remaining char. count ≤ 5
Yes
Get remaining partial word
Is it all leading blanks or trailing zeros
No
Yes
Take "None Left" Return
No
Is remaining partial word empty
Yes
Get a new word
Get partial word
Shift out the char.
Save new partial word
Call PUTCHR
Have we finished this AP word
Yes
Increment AP word pointer Set partial word count to 6
Increment char. count
Any there any more
No
Take "None Left" Return
Yes
Return normal

Figure A1.26 SCNGET Routine

in the merge buffer. When the character count gets to five or less, SCNGET checks to see if all remaining characters are blank. If so a special return to the calling program is taken indicating that all actual parameter characters have been moved.

OMIT

This subroutine is called when an actual parameter has been omitted, signalling that the formal parameter should be used (see Fig. A1.27). It is also called when the actual parameter is an explicit null. In this case, the explicit null overlays the formal parameter before the call to OMIT.

GETCHR

GETCHR is a simple character feeding routine (Fig. A1.28). It must be initialized with the buffer address from which characters are to be obtained and the character number of the first character of the first word. Subsequently, every time it is called it fetches one more character from the buffer. Two exits are provided, a normal one when a character has been fetched, and a "none left" exit when the end of the buffer is reached. GETCHR assumes that the buffer used is the standard card buffer, but the starting address may be anywhere within this buffer.

PUTCHR

This subroutine is called to put one character in the merge buffer (Fig. A1.28). When the merge buffer is filled, PUTCHR automatically exits to LINFUL (Fig. A1.29).

SCAN14

SCAN14 (Fig. A1.29) is a terminal subroutine of SCAN which is called if it is determined that the current line being scanned could not possibly have any more formal parameter, for example, because there are

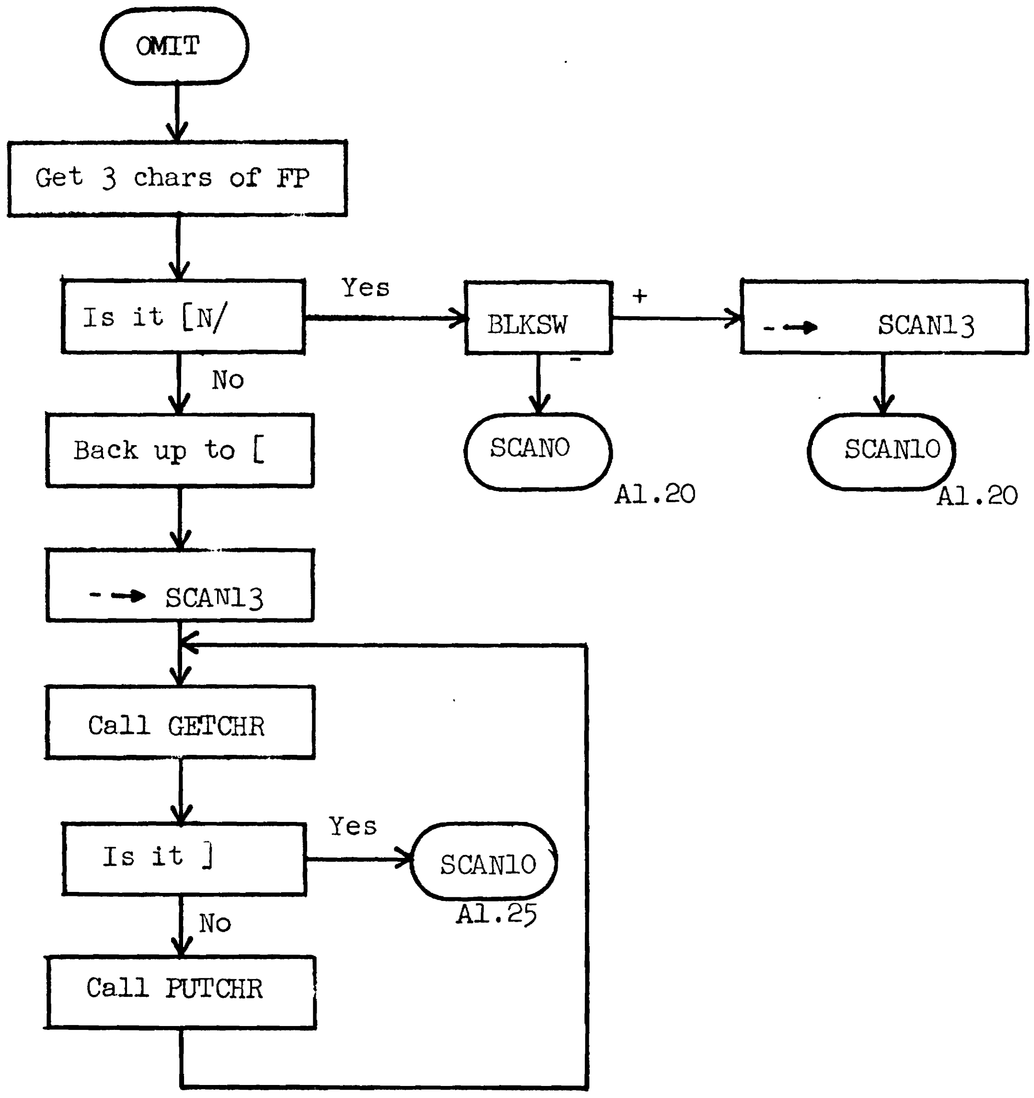


Figure A1.27 Null Parameter Routine

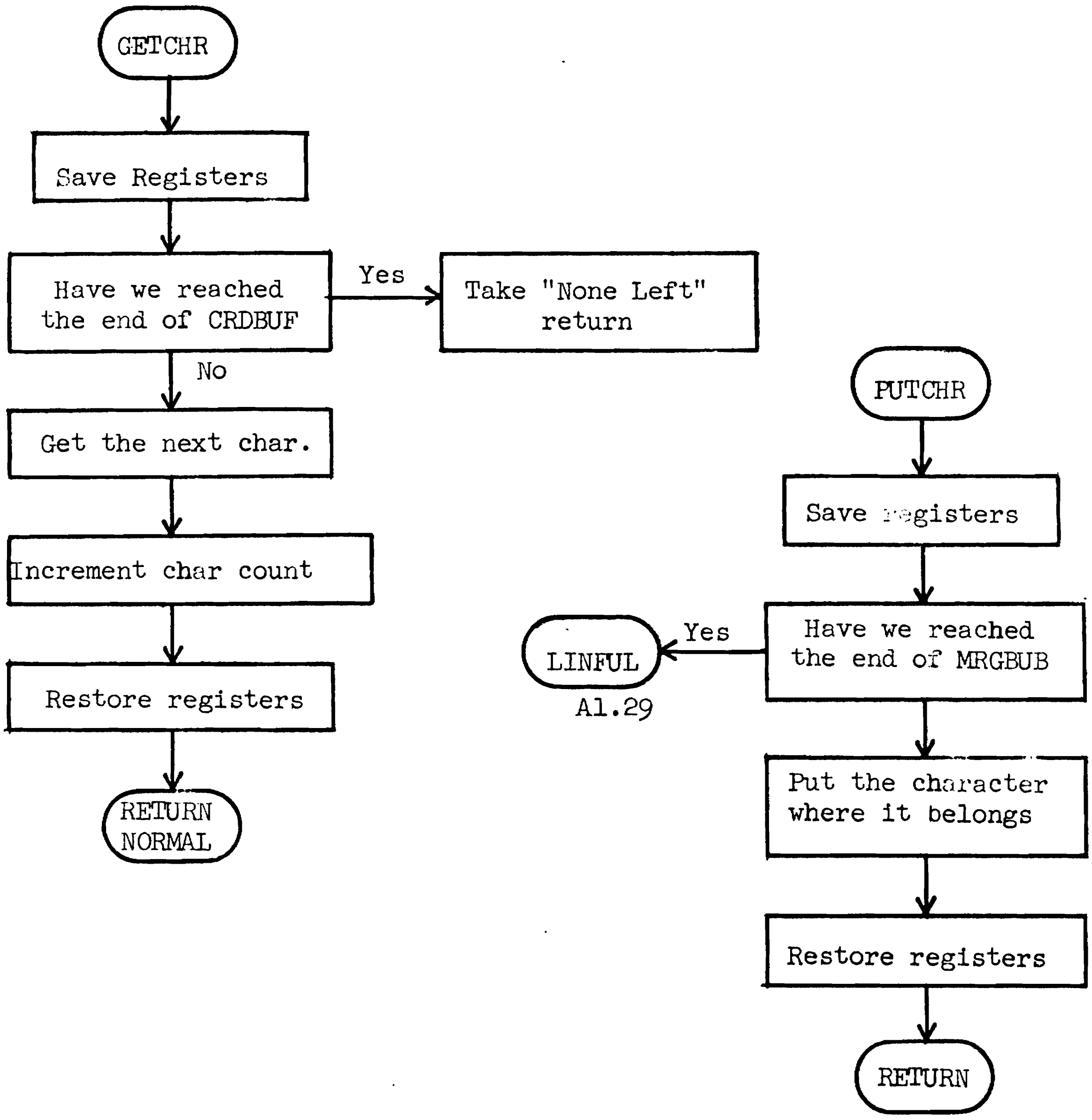


Figure A1.28 GETCHR and PUTCHR

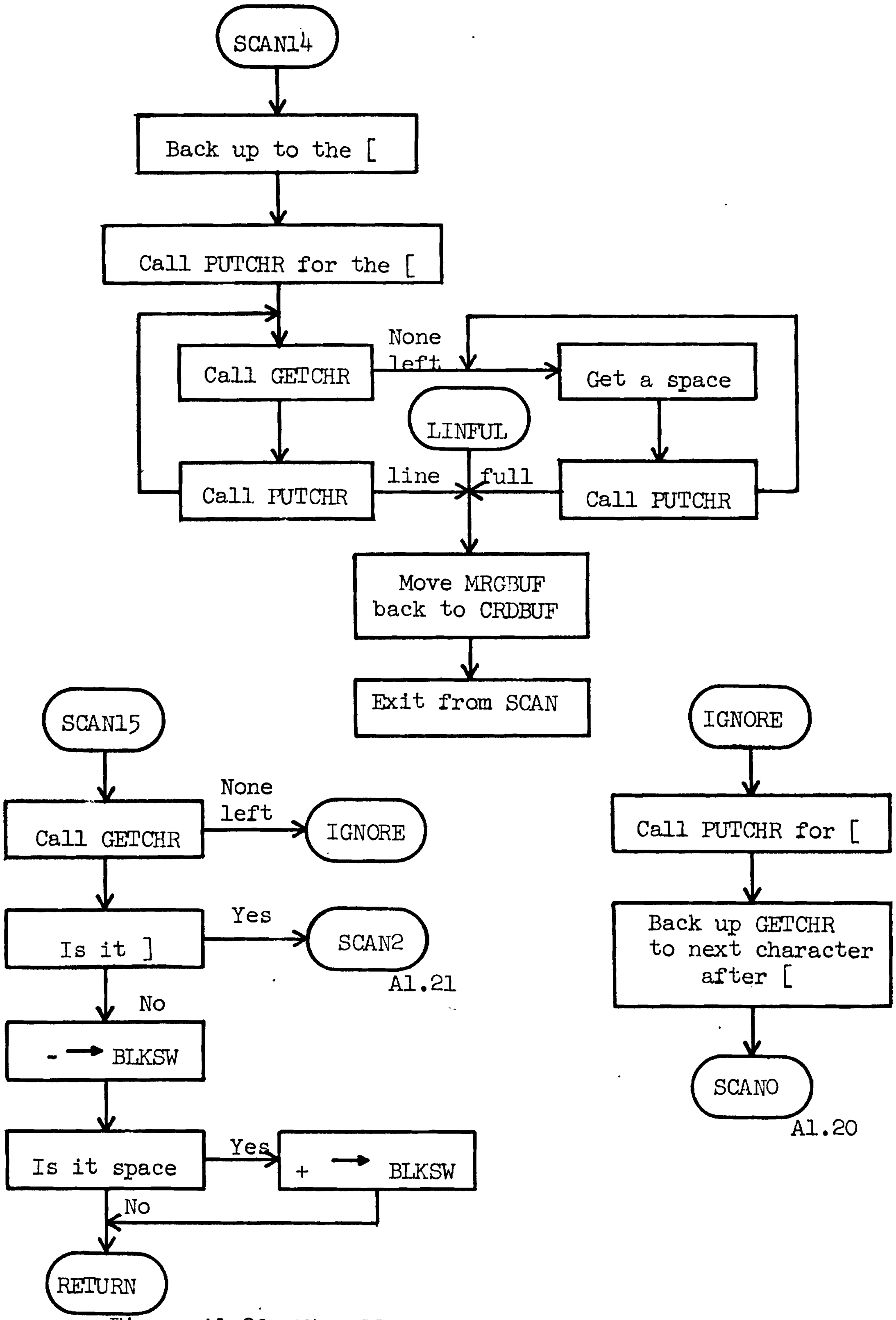


Figure A1.29 Miscellaneous SCAN Subroutines

no more right brackets. LINFUL is also an entry point of this routine which is called when the merge buffer is full.

SCAN15

This subroutine fetches one character from the line being scanned and checks for a right bracket. It also keeps track of whether or not the right bracket is preceded by a blank (see Fig. A1.29).

IGNORE

When a formal parameter candidate does not match a member of the formal parameter list, it is ignored, and scanning continues at the next character afte the left bracket (see Fig. A1.29).

A1.7 Miscellaneous IXSYS Subroutines

CHARFD

CHARFD is the subroutine which supplies characters to the MULTILANG assembler MASS when it assembles IXSYS commands (see Fig. A1.30). The first time it is called CHARFD is expected to feed six blanks to MASS. Thereafter, it feeds the number of characters requested by MASS. If the end of a line is encountered and MASS is still asking for more characters, the subroutine MORE (Fig. A1.31) is called to read another line and return to CHARFD.

PUT

PUT is a subroutine used to output dollar sign cards onto the input scratch IXSYS is making up for IBSYS (Fig. A1.31). PUT expects the file UTLFB1 to be opened as an output file when it is called.

OUTPUT

This subroutine is called to write card images on UTLFB1 (see Fig. A1.31). If the card is a dollar sign card, the buffers are flushed and PUT is called.

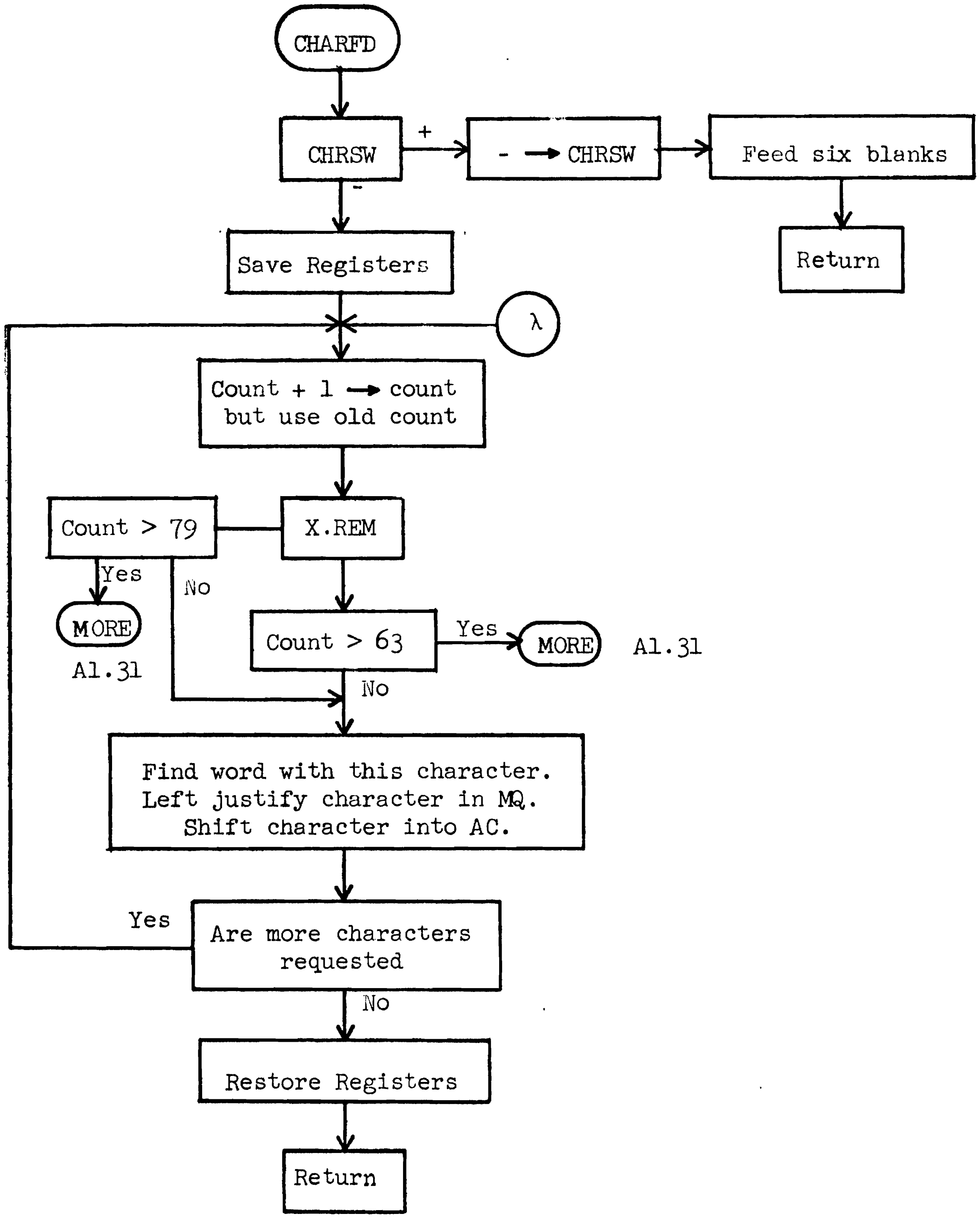

CHARFD
CHRSW
+
-
- ⟶ CHRSW
Feed six blanks
Return
Save Registers
λ
Count + 1 ⟶ count but use old count
Count > 79
X.REM
Yes
No
MORE
A1.31
Count > 63
Yes
MORE
A1.31
No
Find word with this character. Left justify character in MQ. Shift character into AC.
Yes
Are more characters requested
No
Restore Registers
Return

Figure A1.30 Character Feeding Routine Used With MASS

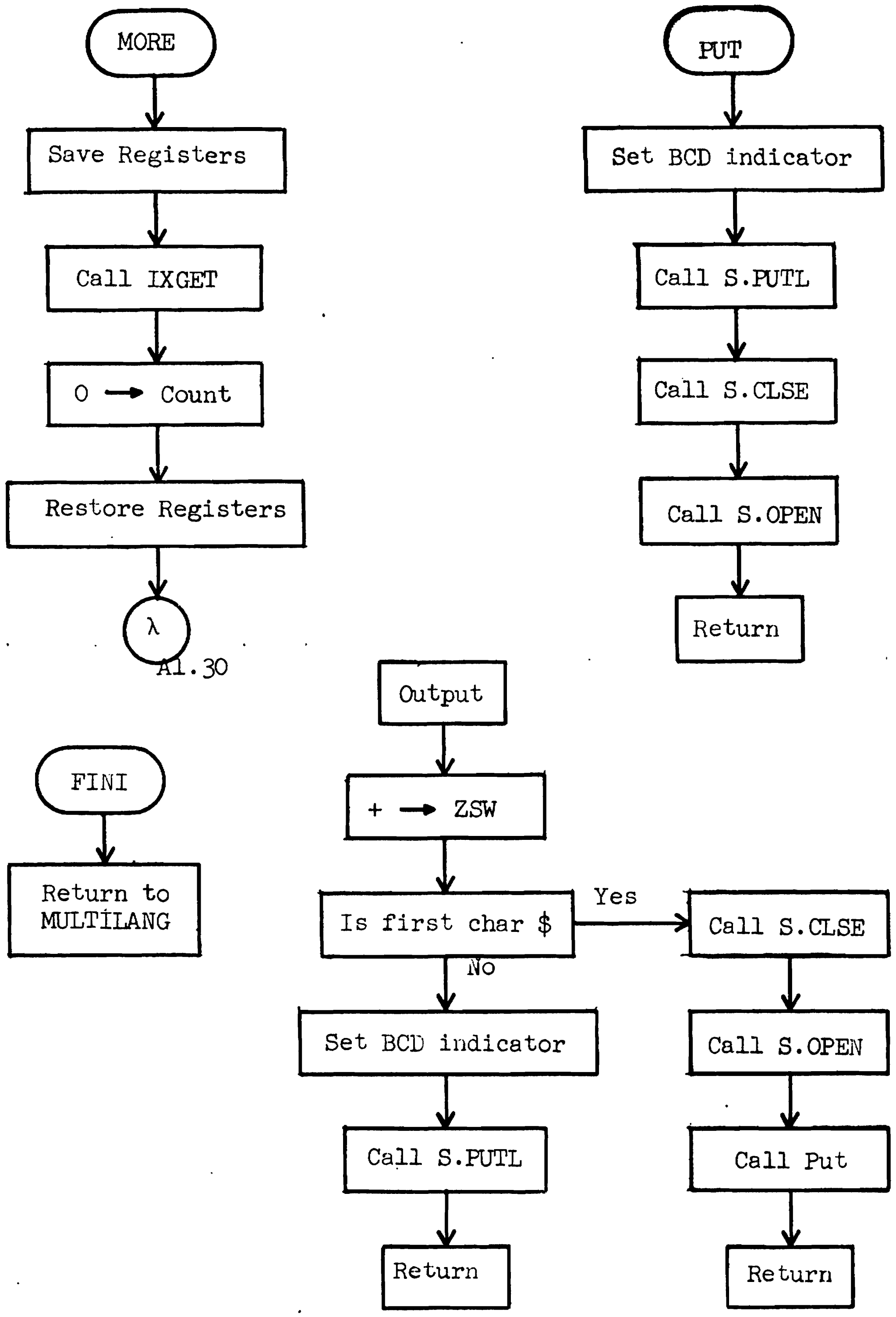


Figure A1.31 Miscellaneous IXSYS Subroutines

FINI

If, for some reason, IXSYS does not choose to transfer control to IBSYS, but instead returns to MULTILANG, it does so via FINI (Fig. A1.31). There are three reasons why this might happen: 1) a catastrophic error, 2) no data has been transferred to the IBSYS input tape (for example, if a description failed to retrieve anything), and 3) if a macro is defined using the condensed form for the definition.

APPENDIX 2 IXOUT

The output recovery phase of the IXSYS procedure is performed by a program called IXOUT. There are three steps to the IXOUT function. First, parameters to IXOUT that have been set up by IXSYS must be processed. These parameters identify the terminal to which the output belongs and the line MULTILANG is to start reading after IXOUT finishes (the line after the last one read by IXSYS). Then, the IBSYS system input unit must be switched back and the file IXOUT is to read must be opened. Lastly, the output data must be read from the intermediate unit, reformatted, and written on the appropriate terminal output file.

A2.1 Parameter Processing

IXSYS supplies two parameters to IXOUT. The first (see Fig. A2.1) is the terminal number of the user requesting the IXSYS job. If this number is larger than any terminal number it identifies the system input unit (a regular batch user). In this case a switch is set indicating that no output must be recovered and the second parameter is ignored. Otherwise, the second parameter is the input line number where processing is to continue when IXOUT returns to MULTILANG.

A2.2 Unit Control

There are three types of units which must be initialized by the unit control phase of IXOUT (Fig. A2.2). The MULTILANG units X.IN and X.OUT must be set to the input and output files of the user's terminal, and the input file must be positioned to the proper line. The IBSYS units S.SIN1 and S.SU12 must be switched back to their original positions. The IXOUT input file UTLFB2 must be closed, marked with an end of file mark, rewound, and opened so that the user's output may be read from it.

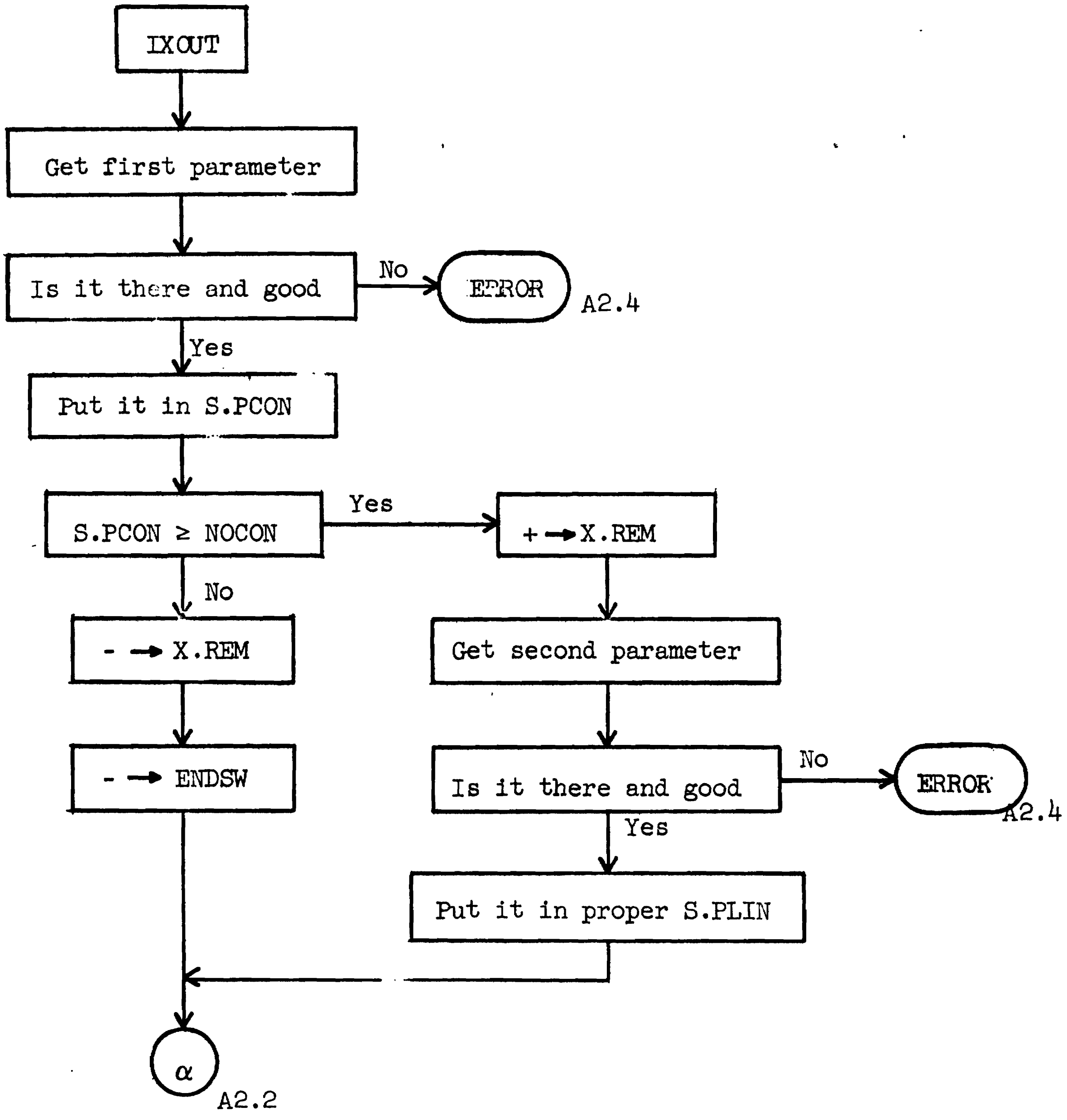


Figure A2.1 IXOUT Parameter Processing

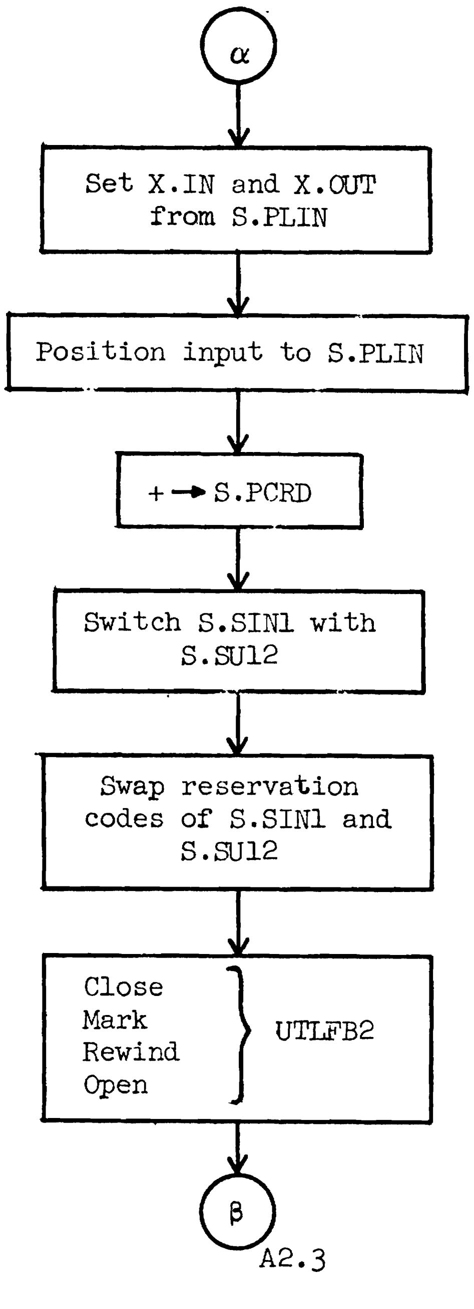

α
Set X.IN and X.OUT from S.PLIN
Position input to S.PLIN
+ → S.PCRD
Switch S.SIN1 with S.SU12
Swap reservation codes of S.SIN1 and S.SU12
Close
Mark
Rewind
Open
UTLFB2
β
A2.3

Figure A2.2 Unit Control in IXOUT

A2.3 Output Swapping

The flow diagram for the output swapping phase of IXOUT is given in Fig. A2.3. One logical record (output line) is read at a time and a check is performed to determine if it is the last line to be swapped. The carriage control character must be moved from the first position of the line to the calling sequence of the output routine, being converted in the process to the corresponding control character for output files of the MSPSF. Next, the output line must be split into two lines between characters 64 and 65. This requires the second half of the line to be shifted right two characters. Lastly, the two half lines are written on the user's output file using the MULTILANG output routine MULOUT.

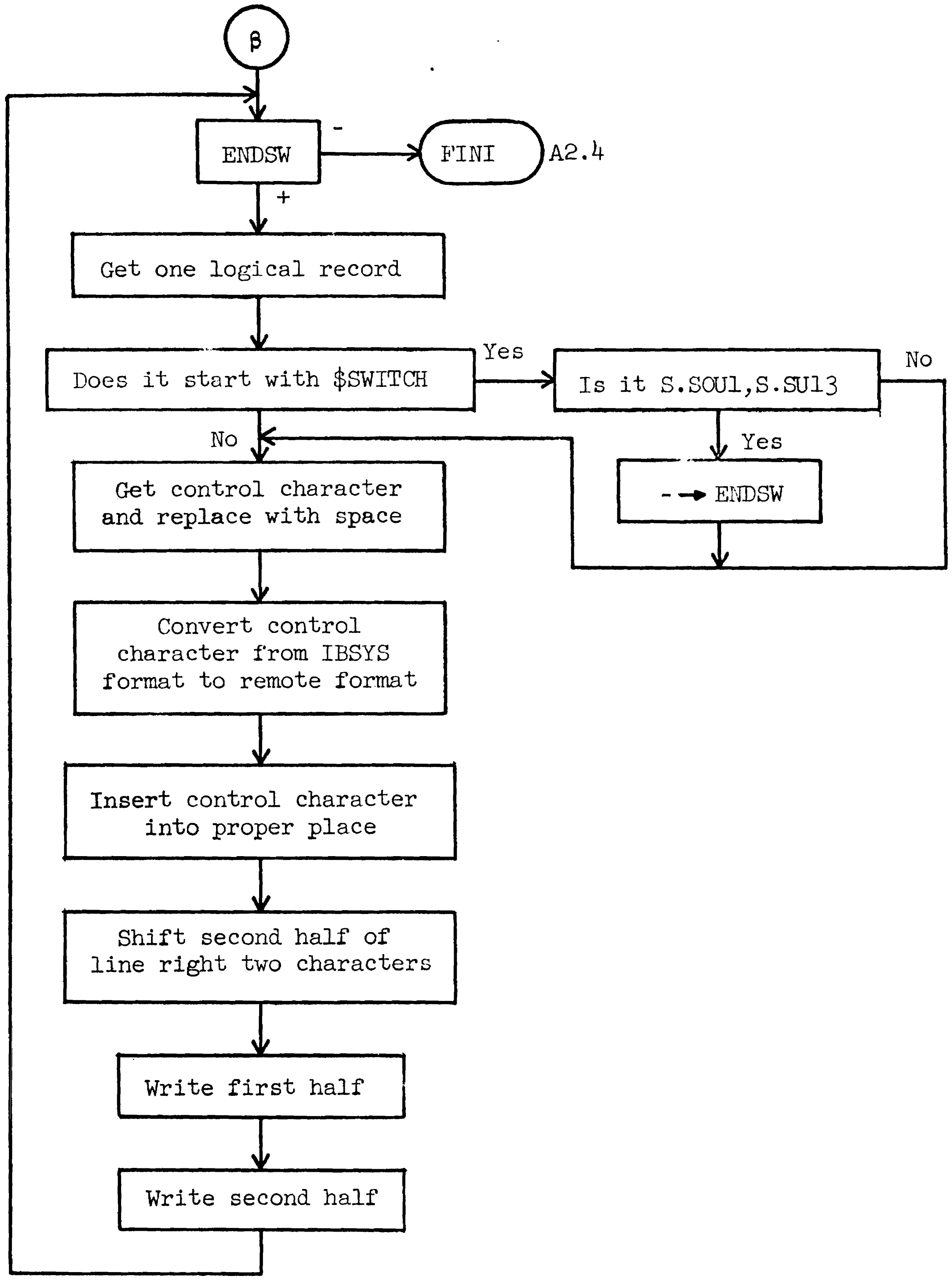

β
ENDSW
-
+
FINI
A2.4
Get one logical record
Does it start with $SWITCH
Yes
No
Is it S.SOU1,S.SU13
No
Yes
--→ ENDSW
Get control character and replace with space
Convert control character from IBSYS format to remote format
Insert control character into proper place
Shift second half of line right two characters
Write first half
Write second half

Figure A2.3 IXOUT Swapping Phase

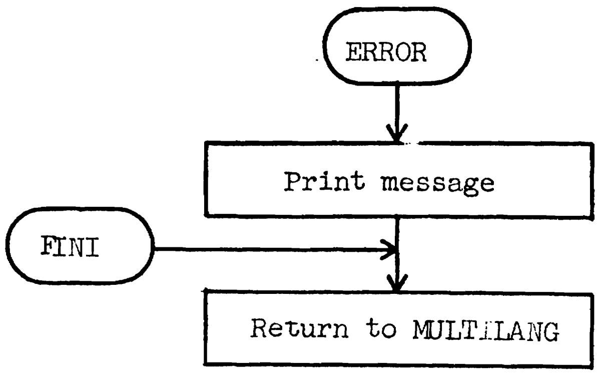


Figure A2.4 Terminating from IXOUT

APPENDIX 3 MISCELLANEOUS WORKER PROGRAMS

The worker programs described in this Appendix were written primarily for remote users of MSPSF, although they also have utility for batch users. Conversely, there are several worker programs used frequently by remote users which have not been described here because they are intended for far more general use.

A3.1 STORE

Purpose: To save source input in the data file.

Use: The format for a call on STORE is as follows:

```
STORE/key1/key2/.../keyn
```

where key1, key2, ..., keyn are the keys under which the input is to be saved. The key ST.RE is also added.

The input to be saved follows the call on STORE, and ends with the following card:

```
$STORE
```

Operation: First, all data items matching the description ST.RE&key1&key2&...&keyn are deleted from the data file. Next the input is read and built into an item for storing. Lastly, the item is stored using the MULTILANG routine ITEMIN.

A3.2 RESTORE

Purpose: To restore to an input file data which has been previously saved by STORE.

Use: The format for a call on RESTORE is as follows:

```
RESTORE/desc[/X]
```

where desc is a description by which the input to be restored may be retrieved, and X is any key. The second parameter is optional and signifies that an attempt is to be made to retrieve as many items which match the description as possible. If no second parameter is provided, only the first item which matches the description (and also contains ST.RE as a key) will be restored to the input file.

Operation: First, the output file pointer in the MULTILANG program MULOUT is reset to be the same as that for the input file. Then the first item matching the description is retrieved and checked for the additional key ST.RE. If no items are found a message to that effect is printed and the exit routine is taken. When a retrieval is successful, the data is written out using MULOUT. This results on the data for a remote user being placed on his input file, or for a background user, the data is simply printed.

After the item has been written the new length of the input file is computed and the append pointed is updated. Then tne existence of a second parameter is checked. If there is one, as many items as can be retrieved by the given description and which contain the key ST.RE are also written on the input file. If there is no second parameter or if there are no more items then the exit routine is taken. The exit routine flushes the MULOUT buffer, restores the output file pointer and prints a terminating message on the output file for the user.

A3.3 PUNCH

Purpose: To punch data which has been previously by STORE.

Use: The format for a call on PUNCH is as follows:

PUNCH/desc[/sequence number]

where desc is a description by which the items to be punched may be retrieved, and sequence number as the starting sequence number for punching. This parameter is optional, and if omitted, all 80 columns of the original data are punched. If the second parameter is included tne following two options:

(a) sequence number is less than or equal to six characters; ignore it and sequence from b̸b̸b̸b̸0000.

(b) sequence number is more than six characters; use first eight characters (left justify space fill) as the initial sequence number.

If sequencing is performed, it is by tens. If sequencing is performed and more than one item is retrieved, the sequencing of each successive item continues where the previous one ended.

Operation: The first item matching the given description is retrieved and checked for the additional key ST.RE. If none are found a message to that effect is printed. If an item is found which is to be punched, a separator card, which identifies the deck for dispatching purposes, is punched first. Then the second parameter is checked and the appropriate sequencing or lack of it is noted and initialized. Next, each card is obtained from the item, sequenced appropriately, and punched. After all decks matching the given description have been punched, a message to the user is printed informing him of the number of decks punched.

BIBLIOGRAPHY

Amdahl, Gene M.: Validity of the single processor approach to achieving large scale computing capabilities. Proc. AFIPS Spring Joint Computer Conf. New York: Spartan Books, Inc., 1967.

Auroux, A., and Bellino, J.: A 1401/7044 Time-Shared System in Batch Processing Mode and in Conversational Mode, Tr. by Michael S. Wolfberg. Unpublished.

Bursky, Philip, Churchill, William and Prywes, Noah S.: Description of a Man/Machine Competitive Game. University of Pennsylvania, Moore School of Electrical Engineering, M. S. Report 67-21, 1967.

Dartmouth Time Sharing System. Dartmouth College Computation Center, Hanover, New Hampshire, 1964.

Denning, Peter J.: Effects of scheduling on file memory operations. Proc. AFIPS Spring Joint Computer Conf. New York: Spartan Books, Inc. 1967.

Dennis, J. B.: Segmentation and the design of multiprogrammed computer systems, J. ACM, 12:4, 1965.

Engvold, K. J., and Hughes, J. L.: Teaching "Hands-On" Programming at a Display Terminal: The ABAC-II System. International Business Machines Corp., Internal Document, Poughkeepsie, 1966.

Engvold, K. J., and Hughes, J. L.: A General Purpose Display Processing and Tutorial System. International Business Machines Corp., TROO.1694, Poughkeepsie, 1968.

Estrin, G., and Kleinrock, L.: Measures, models, and measurements for time-shared computer utilities. Proc. ACM National Conf. Washington: Thompson Book Co., 1967.

Freedman, H.: A Storage and Retrieval System for Real-Time Problem Solving. University of Pennsylvania, Moore School of Electrical Engineering, M. S. Report 66-05, 1965.

Fuchel, Kurt, and Heler, Sidney: Considerations in the design of a multiple computer system with extended core storage. Comm. ACM, 11:5, 1968.

Fuller, R.H.: Associative parallel processing. Proc. AFIPS Spring Joint Computer Conf. New York: Spartan Books, Inc. 1967.

Harrison, M.C., and Schwartz, J.T.: SHARER, A time sharing system for the CDC 6600. Comm. ACM, 10:10, 1967.

Hollander, Gerhard L.: Architecture for large computer systems. Proc. AFIPS Spring Joint Computer Conf. New York: Spartan Books, Inc., 1967.

Hsiao, David K.: A File System for a Problem Solving Facility. University of Pennsylvania, Ph.D. Dissertation, 1968.

IBM 7040/7044 Remote Computing System. IBM System Reference Library No. 7040-25, Form C28-6800.

Kapps, Charles A.: SPRINT: A Direct Approach to List Processing Languages. Proc. AFIPS Spring Joint Computer Conf. New York: Spartan Books, Inc., 1967.

Kleinrock, Leonard: Time-shared Systems: A Theoretical Treatment. J. ACM, 14:2, 1967.

Morton, Richard P., and Wolfberg, Michael S.: The Input/Output and Control System of the Moore School Problem Solving Facility. University of Pennsylvania, Moore School of Electrical Engineering, Report 67-30, 1967.

The Multi-List System, Technical Report No. 1, Vols. I and II, University of Pennsylvania, Moore School of Electrical Engineering, M. S. Report 62-10, 1961.

Nielson, Norman R.: The Simulation of Time Sharing Systems. Comm. ACM, 10:7, 1967.

Oestreicher, M.D., Bailey, M.J., and Strauss, J.I.: GEORGE 3-A General Purpose Time Sharing and Operating System. Comm. ACM, 10:11, 1967.

Ostrand, T.J.: An Expanding Computer Operating System. University of Pennsylvania, Moore School of Electrical Engineering, M. S. Report 67-16, 1967.

Pike, Thomas N., Jr.: Time-Shared Computer Systems. Advances in Computers, Vol. 8. New York: Academic Press, Inc., 1967.

Prywes, N. S. and Gray, H. J.: Outline for a Multi-List Organized System, Proc. 14th ACM Meeting, 1959.

Reiter, Allen: A Resource-allocation Scheme for Multi-user On-Line Operation of a Small Computer. Proc. AFIPS Spring Joint Computer Conference. New York: Spartan Books, June, 1967.

Sackman, H., Erikson, W.J., and Grant, E.E.: Exploratory Experimental Studies Comparing Online and Offline Programming Performance. Comm. ACM, 11:1, 1968.

Schatzoff, M., Tsao, R., and Wiig, R.: An Experimental Comparison of Time Sharing and Batch Processing. Comm. ACM, 10:5, 1967.

Schwartz, Jules I., and Weissman, Clark: The SDC Time-Sharing System Revisited. Proc. ACM National Conference. New York: Thompson Book Co., 1967.

Slotnick, Daniel L.: Unconventional Systems. Proc. AFIPS Spring Joint Computer Conf. New York: Spartan Books, Inc., 1967.

Strachey, C.: A General Purpose Macrogenerator, Computer J., 8:3, 1965.

Varian, L.C., and Coffman, E.G.: An Empirical Study of the Behavior of Programs in a Paging Environment. Preprint for ACM Symposium on Operating Systems, 1967.

Vyssotsky, V.A., Corbato, F.J., and Graham, R.M.: Structure of the Multics Supervisor. Proc. AFIPS Fall Joint Computer Conf. New York: Spartan Books, 1965.

Weinberg, Paul R., and Wolfberg, Michael S.: The PDP-5 As a Satellite Processor. Proc. Spring DECUS Meeting. Maynard, Mass.: Digital Equipment Corp. Users Society, 1966.

West, George P.: The Best Approach to a Large Computing Capability. Proc. AFIPS Spring Joint Computer Conf. New York: Spartan Books, Inc., 1967.

Wexelblat, R.: The Development and Mechanization of a Problem Solving Facility. University of Pennsylvania, Moore School of Electrical Engineering, Dissertation, 1965.

Zimmerman, B., Lefkovitz, D., and Prywes, N.S.: The Naval Aviation Supply Office Inventory Retrieval System--A Case Study in File Automation. Management Science, 10:3, 1964.