THE DESIGN AND IMPLEMENTATION
OF AN
INCREMENTAL ASSEMBLER

By

JAMES ALAN FORRESTER, B.Sc.

A Project
in Partial Fulfilment of the Requirements
for the Degree
Master of Science

McMaster University

May 1974

MASTER OF SCIENCE (1974)
(Computation)

McMASTER UNIVERSITY
Hamilton, Ontario.

TITLE: The Design and Implementation of a Simple Incremental Assembler on the Hewlett Packard 2100A Computer

AUTHOR: James Alan Forrester, B.Sc. (McMaster University)

SUPERVISOR: Dr. Nicholas Solntseff

NUMBER OF PAGES: xv, 444

ABSTRACT

The basic concepts of batch, conversational, and incremental computing are presented along with a brief discussion on their influence on computing.

The design and implementation consideration for the assembly language implementation of a simple incremental assembler is presented. An assembler, to accept simple assembly language programs which are scanned as they are received and assembled into machine code, has been implemented on the Hewlett Packard 2100A computer and is discussed in full detail. The assembler has been designed to execute incomplete programs such that debugging print out of registers and specified core locations is possible. The assembler also provides an editor to perform delete, insert and replace operations on user programs input to the assembler.

The assembler is oriented for the naive user, but it assumes the user has a small knowledge of assembly language programming. Important considerations in writing interactive programs are also discussed.

ACKNOWLEDGEMENTS

At this time I would like to thank Dr. Nicholas Solntseff for his patient and helpful guidance throughout the implementation of the project and for his many comments and suggestions regarding the form and content of this report.

I would like to express my appreciation to the Applied Mathematics Department at McMaster University for giving me the opportunity to attend graduate school and for the privilege of using the departmental computer for my Master's project.

I would also like to express my appreciation to the National Research Council of Canada for the research support.

I am grateful to Dr. G.L. Keech and Dr. R.A. Rink for reading my project.

Special thanks must be given to Mr. Chris Bryce whose advice and suggestions were very valuable in the implementation of the project, Dr. Khursheed Ahmend for the use of his Hewlett Packard cross-assembler during the implementation of the project, and to all the students and friends at McMaster University with whom I have been associated.

Lastly, I would like to thank Mrs. Jane Fabricius for the meticulous typing of the project report.

TABLE OF CONTENTS

CHAPTER V: THE SYSTEM CONTROLLER AND THE INPUT/OUTPUT PACKAGE 51

Page

Summary

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INCREMENTAL ASSEMBLY, CONCEPTS AND CONSEQUENCES

ASSEMBLERS

When computers first began to be used it was realized that programming in machine language was an extremely tedious process. One of the most important steps taken to make programming easier was to introduce mnemonic codes in place of machine operation codes and addresses. The use of mnemonic codes leads to a programming language almost equivalent to machine language but very much easier to read. A program for translating from such a language into the corresponding machine language is called an assembler.

The main task of an assembler is to translate assembly language instructions into machine language instructions that correspond almost one-to-one with what appears in the assembly language program. The assembler uses a table to determine the appropriate operation codes. Also it must assign and keep track of addresses as well as pseudo operation codes of the assembly language.

The advantage of an assembler arises when a program is being tested. It is often useful to output intermediate results, as well as the required answers, to follow the course of calculations in full detail. Extra output instructions must obviously be inserted to provide this information.

These additional instructions can be easily removed from the program once the program is working properly. The assembler can create a new machine language version without any further effort on the part of the programmer. On the other hand, to remove extra instructions directly from a machine language program and include the necessary adjustments is tedious and likely to introduce new program errors.

The difficulty in writing an assembler is not so much in developing one that translates assembly language programs correctly but in producing one that is able to handle incorrect programs in some sensible way.

BATCH, CONVERSATIONAL AND INCREMENTAL SYSTEMS

BASIC DEFINITIONS

Of prime importance are the definitions of source and object program. The source program is the program written by the programmer whether it is coded in symbolic form like punched cards or typed in at terminal. The object program is the assembled code which is recognized by the computer as executable instructions.

BATCH ENVIRONMENT

The term batch processing implies a programmer submitting his job and receiving his results at a later time. Several jobs are accumulated and the batch then presented to the computer system on an input tape. To the programmer the most important point is that he has no contact with his job between the time the job is submitted until he receives his output.

The most significant aspect of batch processing is that the entire source program is available initially and all output can be postponed until a later phase. Declarative statements are processed in an initial phase with storage allocated immediately. In the same pass statement labels are recognized and entered into the symbol table; then in a later phase decisions regarding statements using labels can be made immediately on the basis of table entries. In addition source program error diagnostics can be postponed and the object code may be suppressed.

CONVERSATIONAL CONCEPTS

Compared to the batch environment where the user has no contact with his job after submission a conversational environment provides the exact opposite. In a batch environment a user may have to make several runs to eliminate syntax and logic errors with the intervening time ranging from minutes to days. But in the conversational mode the user can interact with the computer to define his program on a statement by statement basis. After each statement has been entered the conversational assembler will respond to the user so that syntactic errors can be eliminated in one terminal session and execution time debugging is possible on a dynamic basis.

Conversational programming places a heavy load on the overall system; the magnitude of the load is reflected in the additions necessary to support the conversational environment. Basically the conversational assembler or compiler is very similar to the conventional batch processor containing special features for conventional, terminal-oriented operation. Conversational assembly involving two passes assembles each statement conditionally with the source program residing on external storage.

Conversational assembly offers significant advantages over batch processing which are inherent in the interactive mode of operation. The conversational mode is similar to the batch mode in that the entire source program must be defined before execution but differs from batch processing in that the user has control over the input/output functions in the conversational mode. Ultimately one would like the flexibility of a language interpreter with the performance of a batch or a conversational assembler.

The incremental mode of operation is a refinement of the conversational mode. Like the conversational mode, user-system interaction on a statement by statement entry is inherent to incremental assembly but the possibility of line by line execution or the execution of incomplete programs is inherent in an incremental system and not in batch or conversational operations.

INCREMENTAL SYSTEM OVERVIEW

An interactive programming environment should achieve the speed factors inherent in assembled programs and the flexibility of interpretive systems. Incremental systems are an attempt to achieve these goals.

In order to achieve such goals the following features are required:

1. The ability to execute a source program as it is being input;
2. The ability to edit prior statements without re-entry;
3. The ability to execute selected portions of a program;
4. The ability to function in the batch mode.

To achieve these above requirements a highly sophisticated operating system is required. Some of the features would possibly be:

1. A dynamic loader for hand coded subroutines;
2. A memory relocation feature for changing virtual addresses to actual machine addresses;
3. A high level language beyond standard FORTRAN or assembly language for implementation to enable a significant amount of computation per interaction.

Incremental assembly permits two modes: batch and incremental. The batch mode allows the user to assemble prestored source programs but does not allow program editting during assembly. Incremental mode, used normally conversationall permits execution and edit operations during assembly.

The incremental assembler accepts statements on a statement by statement basis with immediate assembly once the statement is received. Code generated is immediately available for execution with a link maintained between the source program statement and the assembled code to permit edit operations to both the source and assembled code. The user is able to assemble, modify and execute the program on a statement by statement basis otherwise only available with an interpreter. But with an interpreter each statement must be processed each time it is executed. In an incremental system the statement is processed once, when it is entered initially.

There exist two different types of control statements, transient statements and commands. A transient statement is a statement in the source language which is assembled and discarded immediately. This may allow the user perhaps to preset registers or core. Commands permit system activity outside the scope of the source language. An example would be the command to change statement sequencing.

Four basic blocks of any incremental system are:

Program Structure Routines: The program structure routines maintain the source program and manage a program structure table which contains an entry for each source statement. The Program Structure Table indicates the relationship of statements and the static properties of the program. Table elements are generated as the source language statements are processed.

Controller: To provide the interface between the user and the assembler and to direct control flow according to the input.

Execution Monitor: To control program execution as determined by the established mode of operation.

Command Controller: To analyse and dispatch command requests.

By the nature of incremental assembly and the Program Structure Table it is not always necessary that code reside in contiguous core locations. Although this is a conceptual difference it poses no serious problems.

Source statements available at entry to an incremental assembler may range from a single statement to a whole program. The source may also be a group of statements to be inserted into the existing program or replacement statements which must be incorporated into the Program Structure Table.

INCREMENTAL EXECUTION

Due to the incremental process there are four possible modes of execution:

1. Automatic:	Each statement is executed immediately after assembly.
2. Controlled:	Execution only occurs when explicitly requested.
3. Block Step:	Controller pauses for user intervention after the execution of each block or subroutine within the program.
4. Step:	Execution is suspended after each statement.

SUMMARY

Batch techniques were developed out of necessity and when these techniques gained acceptance the batch mode was the only operation procedure. Programming in the batch mode may not be the most natural or optimum method, but conversational techniques do not offer a complete solution in that partial program execution is not permitted. Clearly, language and syntax errors are quickly eliminated but if a programmer must fully develop an algorithm before assembly he might as well as assemble in batch mode and rely on execution time debugging.

Therefore some kind of incremental assembly seems necessary to develop algorithms in an interactive computing environment. To execute a program as it is being assembled is a natural way and may well be the optimum from a development point of view. Incremental interaction is useful when hunting for errors caused by mispunching or when exploring a family of algorithms. It remains to be seen if the gains justify the complexity of incremental assembly.

This report is concerned with an attempt to design and implement a simple incremental assembler for teaching assembly language programming. Before describing design considerations and implementation, considerations for interactive programming and the net effect of online utilization are discussed.

CONSIDERATIONS FOR INTERACTIVE PROGRAMMING

"An interactive system is only useful it it satisfies the users' needs."(7) Depending on the type of person for which the system is designed, various features can be implemented to achieve successful user orientation. The following list includes features used in the project and mentions others which could be used for similar programs.

The system should consist of smoothly linked steps. No gaps should occur in its flow which require the user to consult outside references. Ancillary information should be stored to be produced on request rather than routinely within the program unit.

All input should be completely checked, and both lexical and logical errors, if possible, should be flagged. Diagnostic messages should clearly indicate user remedial action. Errors can be reduced if the user can see his input after he enters it but before it is processed -- an echo check.

Responses to prompts should be as simple as possible so that control alternates frequently between the user and the program; although the computer accomplishes much more during its section of the input/output cycle, the user should feel he is participating as an equal.

If the occurrence of the user's response is more important than the contents of the response, e.g., if the response is simply a proceed command, then input checking can be relaxed; this prevents a delay when an unimportant spelling or other error is made.

It may be that the user should be forced to select an option rather than simply be given the opportunity to specify an option. (This is equivalent to requiring that every field on a control card be specified, even if zero; the chance of an option being forgotten is eliminated.) This feature is not used in this project but changes could be introduced to implement such a system feature.

The availability of a record of the user's experience with the system is helpful when the normal output device does not produce a hard copy.

It may be feasible to include two or more levels of complexity within a system. Once the beginner becomes acquainted with the rules and concepts he can step up to a more advanced system. Storage requirements could therefore be kept to a minimum until the functions and messages of the higher level are required; processing time might increase but user response time should decrease.

The user could earn the right to increased control over the program flow as he learns; he could skip certain steps which he no longer finds interesting or alter certain variables in the midst of execution.

Lastly, the system could be designed to accept criticism. Users would be asked to make comments or otherwise rate the program; on the basis of the response the program can be modified.

INTERACTIVE UTILIZATION TO USERS

The differences between batch and interactive programming lies in the "entire programming practice"(6). The user can direct the run without concern for optimum computer utilization. The interactive environment implies certain conditions different from those of a batch environment; the following is a brief list of some of these features.

A complete plan is not necessary; techniques of trial and error solutions requiring human assistance are all permissible. In program debugging one need not fear that a small omission causes a lost run as in a batch environment. In a good online system program errors should not cause any problem; immediate discovery and correction of program errors should be inherent in an interactive system.

Input/Output devices with the exception of display scopes are generally quite slow restricting the volume of output that can be presented in a given time period. Even if terminals were faster it is unlikely that a user would make much use of the speed for he does not always take the time to absorb much output.

In the interactive mode the user generally enters commands or programs by keyboard devices, which are not intended for rapid or high volume input. The means of expression must be concise to accomplish a maximum and minimize input errors.

Unlike batch or off-line processing the user is spending his own time during the entire programming practice. Some people would prefer to deliver their jobs and retire to their home or office until the job is run and collect their results at a later time. Most people feel their time is worth the gain of interactive programming but people become annoyed when some error such as a system malfunction causes lost time at a terminal.

PROGRAMMING PROCESS

One apparent difference is that interactive programming favours small program modules which can be connected to form large programs. Small routines are easily and quickly entered and tested for the rapid turnaround time far outweighs the time spent in finding few or no errors.

The language should provide concise powerful statements that allow a dialogue between the user and the program.

Editting techniques modify existing programs or merge keyboard input with other routines at assembly time. Such editors may edit lines by line number or by more advanced methods which edit by context rather than line number.

Lastly, interactive programming is valuable in permitting interaction between the user and the assembler; the assembler may query the user regarding error conditions permitting changes before the assembly is complete. This may be extended to compilers which include questions to aid the compiler to produce better code.

CONCLUSIONS

The most obvious advantage of interactive programming is the time saving. The whole process from coding to final execution can be repeated several times within a relatively short time span. But without the existence of support the mere existence of an interactive terminal will not assist the user very much. In providing such a system one must consider both the methods of operation forced on the user and those which should be present to take full advantage of the situation.

CHAPTER II

IMPLEMENTATION - BASIC CONCEPTS

INTRODUCTION

Initial considerations affecting the assembly language implementation of an incremental assembler are:

1. The basic inherent assumptions about the user;
2. The ultimate goal of the project;
3. To a much lesser extent the facilities of the installation.

The purpose of this project was to design and implement an incremental assembler on the Hewlett Packard 2100A computer to accept simple programs which are scanned as they are received and assembled into machine code. Appropriate error messages are output if necessary. It should be possible to execute parts of a program; debugging printout of registers and core locations should also be possible. In addition, an editor to delete, insert and replace source and object programs should be available.

An inherent basic assumption is that anyone using the assembler has a small knowledge of assembly language programming. The user who has not had experience with assembler languages may have some difficulty but a brief look at the assembler mnemonics in the Hewlett Packard 2100A Reference Manual(8) or the Assembler Manual(9) should provide the user with enough information to use the assembler. For anyone proficient in assembler language programming this assembler is too elementary.

The installation offers a Hewlett Packard 2100A computer with 12K (12288 words) of core, supported by perepheral I/O devices. Of interest are the Olivetti teletype machine and the Data Point 3300 terminal, hard and soft copy devices respectively which lend themselves to interactive input/output activity.

The core size is 12K but it should be pointed out that the last 100_8 words of core contains the hardware - protected basic binary loader and is not available for users' programs.

Since this assembler is an incremental assembler, assembly occurs immediately after statement entry. The assembler does not wait until the program is fully defined.

The remainder of this chapter briefly discusses the standard assembly process and mentions some of the important differences required to implement a simple incremental assembler. Also included is a very brief discussion of the introductory text and System Directives; neither of these are features of a standard assembler but have been included to acquaint the user with the system and to make the assembler more like an incremental system.

STANDARD ASSEMBLY

An assembler normally begins assembly once the program has been thoroughly defined. Such an assembler has two or three passes, if punch and list output are requested. In the first pass the assembler creates a symbol table from the names used in the source statements. It also checks for certain possible error conditions and generates diagnostic messages, if necessary.

During pass two the assembler again examines each statement in the source program along with the symbol table and produces the binary program and program listing. Additional diagnostic messages may also be produced. If both punch and list output are requested, the list function may be deferred to the third pass.

References to undefined instructions or data will cause the printing of diagnostic messages and may halt further system activity after assembly.

SIMPLE INCREMENTAL ASSEMBLY

After the lexical scan of each statement, the assembled instruction and any symbol table entry must both be stored in their appropriate location before reading in the next program statement. A program statement having a lexical error initiates the printing of an error message and a request to re-enter the statement. No attempt is made to assemble such a statement thus the program need not be reassembled for a lexical error.

Assembly time pseudo operations become meaningless in an incremental system. In particular, the Assembly Listing Control pseudo ops listed in the Hewlett Packard Assembler

Manual(9), allowing the user to control assembly listing during pass two or three of the assembly process, are meaningless.

Since the program is defined statement by statement, the program may be executed statement by statement, by specifying program execution after each statement entry. However, the assembler is intended for the inexperienced programmer to develop programs in steps and blocks. It seems reasonable that a user would enter his program in blocks or groups of statements and check out each block by program execution.

The most important difference between standard assembly and incremental assembly is the handling of forward references and the assembly of Memory Reference instructions.

FORWARD REFERENCES

During the first pass of a standard assembly, references to undefined instructions or data are referred to as forward references.

In a one pass system Memory Reference instructions having forward references, involving an undefined symbol in the operand, are retained by linking the undefined assembled code of the Memory Reference instruction to the symbol position in the Symbol Table by means of special pointers. The design and manipulation of forward reference pointers for direct and indirect Memory Reference operands are discussed in Chapter III and VII.

An undefined symbol in an Input/Output instruction operand causes the statement to be ignored; this is discussed fully in Chapter VI in the lexical scan of program statements. An undefined symbol in an ABS or BSS pseudo instruction operand is treated in an entirely different manner; operand handling in this case is explained in Chapter III under the topic of assembler mnemonics and in Chapter VI in the lexical scan of program statements.

DEFINED MEMORY REFERENCE INSTRUCTIONS

In order to distinguish Memory Reference instructions having defined operands from Memory Reference instructions having a forward reference we employ a special assembly of the instruction using one level of indirect addressing and a special table to hold Memory Reference operands.

Instruction assembly techniques used in this assembler are discussed fully in Chapter III following the discussion on program tables.

INTRODUCTORY TEXT

Eleven pages of introductory text are printed to provide some background information and acquaint the user with the system features, in particular the System Directives.

SYSTEM DIRECTIVES

There are seven System Directives all beginning with a colon and all are recognized by their first letter

:ABORT	Discontinue program entry, start over
:DUMP	Dump register contents
:EDIT	Edit the existing source and object program
:HALT	Halt the computer, press run to continue
:LIST	List all or part of the user program
:SEQUENCE	Change the sequencing, then list the program
:XECUTE	Execute the user's program

The commands resemble the control statements in the incremental system described in Chapter I, for they are intended to give the user control beyond the program level.

All but the Halt directive are presented to the user for a halt instruction is more important to someone exhibiting such a program rather than using it. Of these directives presented to the user all are explained in some detail with the exception of the Abort which is fully explained in a single statement, when listed with the others.

:DUMP

After execution register contents will be saved. It will be possible to dump these register contents as well as data address values as an alternative to using output instructions in the user program.

:EDIT

"The process of editting code online is considered by some to be the heart on an online system".(7) The editor is by far the most complicated feature of the program and will only be discussed briefly in this section.

The editor will allow the user

to delete any number of program statements,
to insert statements between any two program statements,
and to replace a single statement by another single statement.

Editor restrictions will be discussed in the section dealing with the detailed program description.

:LIST(,M(,N))

A list option is another inherent feature to permit listing of all or part of the program anytime, except during an edit.

M and N, if present, specify the first and last lines to be listed. If N is absent then all statements from M on are listed. If neither M nor N are present then the whole program is listed. It was decided that all listing would be surpressed if M was greater than N.

:SEQUENCE,M,N

Change the program sequencing such that M is the first statement number with N being the increment. Following completion, the whole program is listed.

Restrictions on M and N are that both are positive integers. M must not exceed 1000 while N must be greater than zero and not exceed 25. Some upper bounds on M and N were necessary and these seem reasonable in relation to more important user restrictions.

The sequence option may seem unnecessary but may be of great importance when inserting many statements between two successive statements or realigning statement numbers after a series of deletes or inserts.

:XECUTE

XECUTE is responsible for the execution of the user program. Incomplete programs may be partially executed but execution will immediately halt with a warning message printed for attempting to execute a machine instruction having a forward reference.

Immediately after successful execution or after encountering a forward reference the contents of the A, B, E, and O registers will be saved in special store variables.

CHAPTER III

ASSEMBLER IMPLEMENTATION

INTROCUTION

The major design and implentation considerations are presented in Chapter III. Also included is a discussion on program segments and error message handling.

SOURCE PROGRAM ASSEMBLY

The operating system of the Hewlett Packard 2100A, the Moving Head Disc Operating System (DOS-M), offers relocatable and absolute assembly options; relocatable assembly permits the user programs to take advantage of all operating system features such as external subroutine calls to library programs. One very obvious advantage is that relocatable assembly requires that the program be written dependent upon operating system features. To implement the assembler using relocatable assembly would require program segments all be dependent on the DOS-M system.

To avoid such dependence on the operating system the source program has been assembled as an absolute program. In an absolute program the addresses generated by the assembler are to be interpreted as absolute locations in memory.

One minor exception is the instructional text stored on the cartridge disc. This data has been stored on the disc using the DOS-M facility to write onto a user disc file

(EXEC Call, Request Code 15). Storing the data in this manner is for ease of programming.

Core normally occupied by system routines during execution after relocatable assembly will now be available to the assembler after absolute assembly. However, base page linkage, external subroutine calls, literals, or any other inherent feature of the relocatable assembler and loader are not available, nor will they be available in any user program input to the incremental assembler.

MNEMONICS AND PSEUDO OPERATIONS

All machine instructions and the arithmetic subroutine requests for hardware multiply/divide operations listed in the Hewlett Packard Assembler Manual(9) are available to the user but not floating point operations.

Scanning Hewlett Packard System listings for the frequency of Register Reference and Alter Skip multiple instructions, it was found that multiple instructions do not constitute a significant proportion of the overall instructions. The Reverse Skip Sense, RSS, instruction was the most common instruction involved in the multiple instructions. An inexperienced programmer may be aware of multiple instructions but will not have much use for them and consequently they will not be made available.

Memory Reference instruction operands have also been restricted to the form:

(+) (symbol)(± integer)(,I) .

A symbol may have one to five characters consisting of A through Z, 0 through 9 or a period; the first character cannot be 0 through 9. The symbol may be replaced by an asterisk (*) signalling the present program location. A symbol may be preceded by a positive sign or a blank.

The integer may be an octal or decimal value. If there is no symbol in the operand this value must be positive but not greater than 77_8; the user is allowed to access the first 100_8 words of base page. An integer and symbol together must not exceed the bounds of the user program area.

The indirect reference indicator causes the address value of the operand to access any other word in the user program which is taken as the new memory reference for the same instruction.

The introductory text warns the user that the assembler is restricted in size but does not discuss user program location. To the user the assembler is a virtual address program, the user is not aware of where and, in some cases, how his program is stored in memory. Thus, many of the pseudo operations instructions listed in the Hewlett Packard Assembler Manual(9) are excluded.

All Assembler control pseudo ops with the exception of the END pseudo-op are excluded. The REP pseudo op, to "repeat the statement immediately following by the number specified in the operand" is described as an Assembler Control pseudo op. Although it does not influence program positioning it has been excluded for it exists as a convenience to experienced programmers.

Object Program Linkage pseudo ops are concerned with relocatable assembly; accordingly, they have been excluded. As discussed in Chapter II the Assembler Listing Control pseudo ops have been excluded.

The Constant Definition pseudo ops ASC, DEC and OCT have been included and implemented in strict accordance with Hewlett Packard definition. Appendix A lists and defines all machine instructions and available pseudo ops.

The DEX pseudo op to generate extended precision constants has been excluded.

The BSS pseudo op for storage allocation has been included but its defintion has been altered. The format

BSS m

normally restricts m to be any expression that evaluates to a non-zero, positive integer. Due to space limitations an upper bound of 128 has been imposed. The definition has been expanded to initialize program storage to zero.

Address and Symbol Definition pseudo ops ABS, DEF, and EQU have been included. Operands for these instructions must evaluate to a value within the program data area bounds. For ABS and EQU pseudo ops the operand is of the form

(+)(symbol)(± integer) .

The operand may also evaluate to an address on the available base page area.

In the case of an EQU a label must precede the pseudo op and an undefined symbol may not be present in the operand. An undefined symbol in an ABS or BSS operand is permitted but will initiate a request to the user to enter a temporary value for the symbol. Further reference to this symbol will not necessarily yield this value.

The DEF pseudo op operand is restricted to a data address symbol and an optional indirect flag. Undefined operands will not be permitted during an edit, but during normal program definition the user is requested to define the symbol on the next statement entry. If the next data entry does not define the symbol or if a data edit operation alters the data area holding the DEF pseudo op, then the address value will be incorrect.

The END pseudo op has been redefined to halt program entry and advance to execute the user program. It will not be stored in the user program; any label preceding or any operand following is ignored. END will not be permitted during an edit operation.

Altogether there are 86 machine instructions and pseudo ops which have been divided up into fifteen different categories depending upon the instruction type and the operand expected. Appendix A has a list of:

1. The available machine code instructions and pseudo ops and their definition.
2. The instruction type number.
3. The machine instructions according to their instruction number.

ASSEMBLER CONTROL STATEMENT

The Assembler Control Statement normally beginning user programs has been excluded. Since the source program is in absolute format a user program input to the incremental assembler will then be an absolute program.

The program list option is meaningless but a list of the source program can be taken at almost any time using the List Directive. Other assembler options like binary output or a cross reference table will not be available or needed.

Since most of the options normally associated with the Assembler Control Statement have been excluded or redefined, the inexperienced user is not expected to enter an Assembler Control Statement.

INSTRUCTION MODIFICATIONS

Although the instruction set has been restricted, the user is expected to have only a small knowledge of assembler language programming. The available 86 mnemonics are ample for learning purposes.

Changes that could be made for an advanced user would be the inclusion of the REP pseudo op and floating point operations. These extra instructions would provide further assembler versatility. To include any other pseudo ops is questionable for the users' expectations are apt to change significantly. Once a user has mastered the techniques of assembler language programming, the pseudo ops should be easily understood.

It may be possible to include features like a cross reference table, conditional assembly or some other feature normally associated with the Assembler Control Statement. The user is apt to benefit from the inclusion of such changes but the overall influence of such program improvements on the user require serious consideration before implementation.

The remainder of Chapter III is a discussion on:

Assembler Tables,
Instruction Assembly,
Forward References,
Program Segments and a list of the Assembler Functional Units,
Error Message Handling.

This material is of particular interest to anyone wishing to alter or extend the assembler but not to those interested in understanding the basic concepts.

ASSEMBLER TABLES

Storage has been allocated for system and user tables beginning at address 15200 to the last available word in memory. These tables are as follows:

The Instruction Table,
The Main Symbol Table,
The Special Symbol Table,
The Program Location Counter Table,
The Free Space Table,
The Source Code Block,
The User Program Table for machine instructions and data.

INSTRUCTION TABLE

This is a system table for instruction look up. This table is not initialized for each new user program; all other tables are initialized for each new user program and set during program definition.

The 86 machine instructions and pseudo ops have been arranged alphabetically for a binary search table look up. The table 402_8 (3×86) words in length has been divided into three separate sections. The first section holds the first two of the three letters of the alphabetic list of mnemonics. Each word in the second section holds the third letter of the mnemonic and the instruction type number in the format:

Bits 0- 3 Instruction type number
8- 15 Third letter of mnemonic name

The third section holds the skeleton of the assembled instruction; the pseudo ops are assigned a (-1) minus one value in this section. The skeleton code of a pseudo instruction is ignored throughout the assembler.

USER PROGRAM TABLES

Unlike the Instruction Table these tables are initialized for each new program. The Main Symbol Table and Special Symbol Table must also be set with special pointers for direct and indirect forward references used by the assembled instructions.

With the exception of the Free Space Table an attempt to make an entry to a User Program Table will terminate all user-assembler activity with the user program being lost. However, all user tables, with the exception of the Free Space Table have a built in warning to the user if the table is about to overflow and a request to begin execution to obtain final program results before table overflow occurs.

MAIN SYMBOL TABLE

The Symbol Table can accommodate up to 125 different symbols, each symbol requiring six words of storage. The format for symbol storage is:

Word 1	First two characters of symbol name		
Word 2	Third and fourth characters		
Word 3	Bits	8- 15	Last character of symbol
	Bit	0 = 1	Defined symbol
		= 0	Undefined symbol

Word 4 and 5 have different uses depending on whether the symbol is defined or not.

Undefined	Word 4	Address of last direct forward reference
	Word 5	Address of last indirect forward reference
Defined	Word 4	Symbol address in assembled code
	Word 5	Symbol address in source code storage
	Word 6	Linkage to Special Symbol Table (see below)

Symbol positioning in the table will be determined by a hash code which takes the arithmetic sum of the words holding the symbol name and divides the value by 125. The remainder yields a relative position in the table to begin a linear search for the next free area to store the symbol. The hash code was tested and found to distribute the symbols throughout the table. This is the only table using a hashing function for all other tables use strictly a linear search and storage procedure.

Each entry to the Symbol Table will be counted by the subroutine for storing symbols while overflow will be determined by the subroutine that applies the hash code function to the symbol and finds the symbol position.

SPECIAL SYMBOL TABLE

The Special Symbol Table, SST, is for compound operands, i.e. Memory Reference operands having a symbol and an integer value. The SST will hold up to 75 different compound operands with each entry requiring four words as follows:

Word 1	The integer value
Word 2	Bits 0-14 Source code address of the instruction Bit 15 = 0 Direct reference = 1 Indirect reference
Word 3	Address of last forward reference
Word 4	Link to further entries in SST

For each Memory Reference operand combination an entry to the SST is made. Symbols having more than one entry in the SST will be linked by Word 4 with a zero in Word 4 terminating the list. Word 6 of the symbol entry in the Symbol Table will hold the address of the first SST entry.

Before actual user program execution special routines will scan the SST and the Program Location Counter Table, a table used to hold similar operands where the asterisk term replaces the symbol, to calculate operand addresses, provided such addresses are within the bounds of the program. This allows edit operations to occur after instruction entry and before execution in order to preserve operand addresses.

By initiating execution as many addresses as possible are defined; the table area used by these address pointers is cleared for further use. Further editting of these instructions after the address has been set is at the users' peril for the address cannot be altered.

PROGRAM LOCATION COUNTER TABLE

The PLC table will hold up to 50 memory reference operands involving the asterisk with the table format being:

Word 1	Bits 0-14	Source code storage address of statement
	Bit 15 = 1	Indirect reference
	= 0	Direct reference
Word 2	Integer value in operand	

The PLC table holds these operands until the user wishes to execute his program at which time the assembler will attempt to define all operand references in the PLC table.

THE SOURCE CODE BLOCK

All incremental systems should allow the user to make corrections to his program and list the updated source program. An incremental assembler can be implemented in several ways; the two means considered for this project were:

1. The user program could be assembled to some intermediate form from which the source program can be recreated.
2. The user program can be assembled into object code. Since the assembly process is not normally one-to-one, it is not usually possible to recreate the source program from the assembled version. The assembler must maintain two copies of the program, one in source form and one in assembled form.

The first approach offers the advantage of not having two copies of the program at the expense of slower running. Using the first approach it was felt that the user might be slightly alarmed if the interpreter were to remove redundant blanks and reformat his output for a list command. It was also found that the trade off between the simplicity in storing source code along with a simple listing program, and the complexity required in the implementation of an intermediate code algorithm from which the source or assembled code could be generated justified storing the source code along with the assembled code.

The Source Code Block, SCB, is 5700_8 words in length and will retain six words of information concerning each statement as well as the source statement. The format for a source statement entry is:

Word 1	Address of the next statement (0 for the last statement)	
Word 2	Address of the previous statement (-1 for the first statement)	
Word 3	Statement number	
Word 4	Bits 0- 7	Number of words in SCB entry
	Bits 8-15	Number of characters in source statement
Word 5	Bits 0-14	Address of assembly (0 for a comment statement)
	Bit 15 = 1	Data definition
	= 0	Machine code instruction
Word 6	Length of assembly	

The source program statement will be stored two characters per word beginning in the first character position (Bits 8-15) of the first word to follow Word 6 in the SCB.

Like the main Symbol Table space in this table cannot be reclaimed by an execution.

FREE SPACE TABLE

The Free Space Table holds the length and address of deletions from the SCB after an edit operation. Each deletion from the SCB will be recorded in two words in the Free Space Table in the following format:

Word 1	Length of the deletion,
Word 2	Address of the deletion.

Unlike the other tables, entries to a full table will not cause program termination. The entry will be retained, if the length of the deletion is larger than the smallest deletion and the smallest deletion will be discarded.

Before storing any statement the assembler will scan the FREE SPACE for an isolated SCB location before allocating the next free area in the SCB. This is a reclamation procedure to make use of all available SCB space for statement storage.

USER PROGRAM AREAS

The last two tables are the user program areas for data definitions and machine instructions having 400_8 and 340_8 words respectively for assembled code. The Dump Directive has been included as an alternative to using output instructions in the user program. For this reason the data area was set larger than the program area.

The overall program area could best be fitted into the last page where 1700_8 locations were available for the user program (700_8 words) and the data area (1000_8 words).

Table 3.1 lists the layout of the user program and data table areas.

The structure of both these tables is very inefficient and space consuming for each table requires a corresponding address field for each data and machine instruction location, i.e. two locations are required for each word of assembled instructions and each word of data definitions.

In the case of data definitions, the address block is necessary to maintain an address pointer to each data item for reference by a machine instruction and for shifting data on an edit operation.

INSTRUCTION ASSEMBLY

The Memory Reference instructions require the address field so that forward references can be easily distinguished from defined Memory Reference instructions.

All machine instructions other than Memory Reference instructions are assembled in much the same manner as in the standard assembly process. Memory Reference instructions use the address table to hold a 15 bit operand address.

Normally, assembly of a simple Memory Reference instruction has a 10-bit address, a current page bit and an indirect bit to be set according to the operand. The incremental assembler sets the 15-bit operand address in the program address table corresponding to the position of the instruction in the user program area. An indirect reference indication in the operand is handled by setting bit 15 of the operand address in the program address area. The Memory Reference instruction is set into the user program area with the 10-bit address pointing to the 15-bit address stored in the address table position. The current page and indirect bits are set so that the instruction involves an indirect reference to the address through the address table.

An Extended Arithmetic Memory Reference instruction assembles into two words; the second word of the assembly is a 15-bit address to the program address table with an indirect reference specified.

All defined Memory Reference instructions with the exception of valid user references to the base page will have the indirect bit and the current page bit set for simple Memory Reference instructions. Forward references will appear as a direct reference to base page.

The assembly is definitely no longer a one-to-one transformation from source to object code because of the particular means adopted for implementation. This is further justification for having two copies of the program.

FORWARD REFERENCES

Forward reference addresses are combined with the instruction skeleton on a Memory Reference instruction; the instruction will appear like a direct reference to base page. Such an address must be greater than 100_8 else the instruction is regarded as a valid user reference to the available base page area. For this reason the user program area was arranged with the program address table preceding the user program area.

From Table 3.1 Symbol Table entries pointing to user program instructions having forward references will be in the range 341_8 to 677_8.

During initialization Symbol Table entries for forward references were set to a value greater than 700_8. Forward reference indicators in the Symbol Table begin at 701_8 for direct references and 1076_8 for indirect references. Each symbol position has a separate pointer for direct and indirect references separated by 175_8 (125). The SST has its forward references beginning at 1273_8.

During program definition the forward reference indicator in the symbol tables is replaced by a pointer to the last forward reference. Forward references to the same operand are linked into a chain with a reference greater than 700_8 signalling the end of the chain and a pointer to the symbol tables.

Program location counter references in the user program are also treated as forward references. The PLC table is bounded by address XPLC, 17634 and YPLC, 17777 such that PLC forward references would range from 1634_8 to 1777_8 and not conflict with symbol table references.

No linkage techniques are used with the PLC table for each PLC reference is regarded as a separate forward reference.

Table 3.2 offers a diagram of forward reference linkage in the main Symbol Table.

PROGRAM SEGMENTS

Program Segments may be described in terms of functional units or segments of storage. In planning the overal program an attempt was made to design each segment as a self-contained program unit so that each functional unit could be regarded as a particular block of computer storage.

However, as the complexity of a program unit increases there is a tendency for the segment to become fragmented. A very obvious example is the editor; due to its complexities it became far too large to store on one page such that editor subroutines were allocated to three different pages of memory. It is also convenient for two different program functions to share common subroutines rather than permit duplication. In such a case program segments will not remain self-contained units. Sharing of common subroutines by several program units will conserve storage space and due to the limited storage size it was necessary for program units to share common subroutines rather than maintaining self-contained program units which may involve subprogram duplication.

In terms of functional units the program may be segmented as follows:

Initialization,
System Controller,
Input/Output Package,
Lexical Scan,
Number Manipulation Package,
Statement Assembly and Storage,
Systems Directives excepting the Editor,
Editor.

A description of program segmentation in relation to the dynamic storage allocation becomes difficult to follow or remember for the text becomes an enumeration of subroutines or program units followed by a brief discussion on each. Such a discussion is not presented but Appendix F does offer a listing of program units in relation to their storage with a brief program discussion.

Following a brief discussion on the error message processing the following six chapters offer a detailed program discussion of the functional segments.

ERROR MESSAGE PROCESSOR

Normally an error message follows the program which uncovered the error condition with the error message output programs resident on base page. There are some minor exceptions in the positioning of error messages; the most obvious exception in the presence of nine error messages on base page to avoid unnecessary duplication. These messages are listed in Table 3.3.

Since most error messages concern user input it seems that there should be an automatic return to the System Controller yet avoid duplication of return instructions. For this reason there is a base page entry point, label ERCAL, which initiates a jump to subroutine ERROR followed by an indirect jump to the System Controller. Any error condition followed by an input operation will initiate a jump to ERCAL.*

Subroutine ERROR

Calling Sequence

```
     LDA < Character length of the error message >
     LDB < Address of error message >
```

Subroutine ERROR calls subroutine BPLN to print the error message on a newline and subroutine REENT to print the re-entry request

PLEASE RE ENTER STATEMENT

on the next line following the error message. BPLN and REENT use the Input/Output package presented in Chapter V, to output the error messages.

---

* There are two exceptions.

Within subroutine DATIN, which prompts the input operation, a buffer overflow error message is printed if necessary but control does not leave DATIN.

On an input error in a sequence request the Sequence flag is set after calling ERROR and before returning to the System Controller.

TABLE 3.1 THE USER PROGRAM TABLES

ADDRESS	ADDRESS NAME	PURPOSE
026001		First address of program address table corresponding to first address of the user program area
026337		Last address of program address table corresponding to the last address of the user program area
026340	PROG	Entry/Exit point for executing the user program
026341	XUSRP	First address of user program area
026677	YUSRP	Last address of user program area
026700		Return jump from user program to calling point
026701	XDATA	First address of data address area
027277		Last address of data address area
027301		First address for data value storage
027677		Last address for data value storage

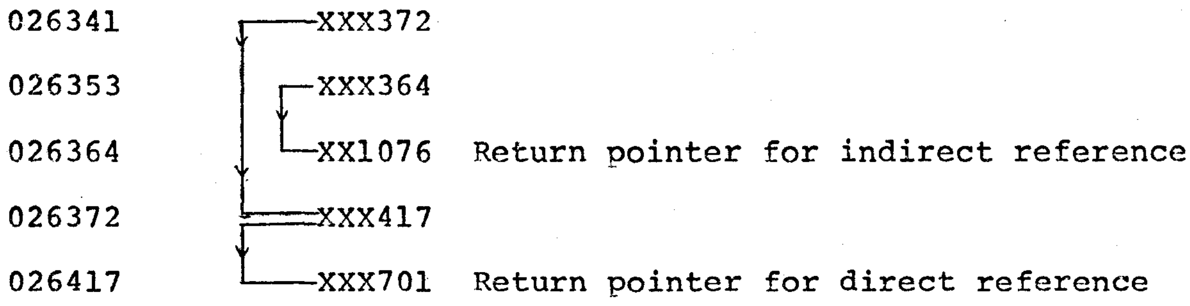
TABLE 3.2 FORWARD REFERENCE LINKAGE

This example of forward reference linkage uses the first symbol position of the Symbol Table having an undefined symbol with direct and indirect references to that symbol.

A diagram of the linkage of the forward references in the user program area shows the address pointer combined with XX, or XXX denoting the skeleton assembly of a Memory Reference instruction. The pointers linking back to the Symbol Table are also presented.

Symbol Table	Address Contents
Word 1	
Word 2	Symbol name stored as Ascii characters
Word 3	
Word 4	341 Page address of first direct and
Word 5	353 indirect forward references
Word 6	

Memory Address



026341 XXX372

026353 XXX364

026364 XX1076 Return pointer for indirect reference

026372 XXX417

026417 XXX701 Return pointer for direct reference

TABLE 3.3 BASE PAGE ERROR MESSAGES

LABEL	ERROR MESSAGE
ERR1	BAD DATA INPUT
ERR2	STATEMENT NUMBER OUT OF RANGE
ERR3	OPERAND VALUE OUT OF RANGE
ERR4	ILLEGAL OPERAND TERMINATION
ERR5	ILLEGAL CHARACTER BEGINS LABEL
ERR6	NO OPERAND FOUND
ERR7	OPERAND IS UNDEFINED
ERR8	UNDEFINED LABEL IN OPERAND
ERR9	NO LABEL FOUND

CHAPTER IV

INITIALIZATION

INTRODUCTION

The initialization program is called for each new user program after one of the following conditions.

Recognition of the Abort Directive
Abort request from the System Controller
Abnormal abort due to a program table overflow
Operator intervention by setting the Program Location Counter register on the computer front panel

PROGRAM INITIALIZATION

The first task is to turn off all I/O activity and enable the interrupt system for the assembler and user program use. A call to subroutine CNFIG will configure the input/output package to direct all user-system communication through the teletype machine for a hard copy output.

Besides the last 100_8 words holding the basic binary loader the first 100_8 words are also reserved locations. Though not considered core storage the A and B registers occupy the first two memory locations. Memory locations 00002 and 00003 are exit points if the A and B register contents should be used as executable instructions. The program was initially assembled with these locations holding indirect jumps to the forward reference warning program as part of the execution routines, if the user should attempt to execute the contents of A or B.

Location 00004 and 00005 are the Power fail and

Memory Protect/Parity Error interrupt locations each holding halt instructions.

All other main frame interrupt locations, address 00006 to 00025, are assembled to zero. Address 00026 to 00077 are the remaining interrupt locations; these addresses are not initialized. By giving the user access to the first 100_8 words allows the user to alter these locations; it is necessary to restore these locations for each new user program.

Into memory locations 00006, 00011 and 00012 are stored subroutine jump instructions to three base page interrupt subroutines used by the disc input driver.

Using the disc input driver the eleven pages of introductory information will be read in. Appendix B has a brief discussion on the text and a listing of the program to store this data as well as a listing of the actual text. Length and address pointers are stored in two tables following the initialization program.

All disc data input/output operations will be initiated by subroutine GRTIO which initializes the disc read, calls the disc input driver, and prints the data using the system I/O package. Disc input operations will be handled using Direct Memory Access, DMA, a facility to provide a direct data path software assignable between memory and a high speed peripheral output device. A full discussion of DMA is given in Appendix C.

After the first page has been printed the user is requested to type S to transfer all I/O activity to the CRT screen or C to continue. This is the first instance where input checking is relaxed for any response other than S is accepted as a continue command. Although, a particular character has been requested as a response to a prompt virtually any other character will be accepted to avoid the generation of an error message.

The S response will cause the I/O package to be configured for soft copy output on the Data Point 3300 CRT screen.

By default program statements are sequenced by beginning at ten and incrementing each statement by ten. The second page advises the user that he may specify alternate sequencing by typing S followed by the first statement number and an increment.

After printing the second page and before reading the user response, all system variables and user tables are initialized. It is not possible to initialize program tables before printing the second page for the length of the first two pages is greater than the length of the buffer area available to store the disc input. An attempt to store either of the first two pages in this area would overwrite part of the Instruction Table. The remaining pages of the introductory text will fit into this buffer area. The first two pages are stored in the core normally used by the program tables; once the second page has been printed, the user tables are initialized.

All user program tables are initialized to zero with the forward reference pointers stored in the symbol tables. All program control flags used in the System Controller and all system variables are set to their initial value. Temporary values used throughout the assembler will not be initialized.

One special variable which must be set is GRTFG, the program flag to signal that the program is in the initialization phase. GRTFG must be set before a user sequence request is read so that program control will return to the calling point within the initialization program rather than the System Controller on an error condition.

The third page offers an option. For the user aware of the assembly features program entry may begin immediately. Any response other than L, the learning option, for presentation of the remaining text is accepted as a signal to begin program entry.

After the last page has been output and before reading the first user program statement all main frame locations beginning at address 00006 to 00025 are cleared to zero along with the initialization flag, GRTFG. After the user entry has been read in program control transfers to the System Controller to call the main lexical routines.

INITIALIZATION SUBROUTINES

Three subroutines from the input/output package are called by the initialization program:

DATIN	Read user input,
I.OFF	Turn off output device interrupt,
TTY.P	Perform output operation.

These subroutines are presented as part of the I/O package in Chapter V.

Subroutine SQNCE reads in the statement numbers for the sequence request. SQNCE is also used for the Sequence Directive introduced in Chapter II; SQNCE is discussed in Chapter VIII with the discussion of System Directives.

The remaining subprograms CNFIG, GRTIO and the disc input driver are used strictly for initialization purposes.

Subroutine CNFIG

Calling Sequence

```
    LDB    < Channel number of I/O device >
```

CNFIG will configure the I/O package to direct all user-system communication through the device referenced by the channel number. All input/output instructions in the I/O package will be set with a new channel number. As well the Memory Reference instructions referring to the device interrupt location must have a new address to point to a new interrupt location.

Subroutine GRTIO

Calling Sequence

```
    LDA    < Disc address of input >
    LDB    < Input length (words) >
```

GRTIO will call the disc input driver to read in a page of the introductory text and call subroutine TTY.P to print the text.

DISC INPUT DRIVER

The disc input driver is comprised of eight subroutines: three interrupt service subroutines and five subroutines taken from the disc I/O driver used in the DOS-M System generator program. Minor changes were made to the five disc driver subroutines but the program structure is unchanged.

The interrupt subroutines are needed after a DMA interrupt to address 00006, a disc Data Channel interrupt to address 00011 and a disc Control Channel interrupt to address 00012. These service routines will clear the control flag of their respective channel and return program control to the location causing the interrupt.

The disc input program has been written by professional programmers understanding the interface between the disc controller and the computer. A program description of the disc driver could be presented but it was felt that such a description requires too much additional background information for a program which is not part of the assembly process.

This program is a tested program. Nevertheless, in order to trace most disc read problems that might arise, it was decided to include three halt conditions for:

Ten unsuccessful read attempts	(HLT 22B),
Address error, abnormal halt	(HLT 24B),
Disc not ready	(HLT 26B).

With the present implementation knowledge of the disc input driver would not be necessary for changing the overall program features. The disc driver is required to retrieve binary data to be printed as introductory text. Changing the assembler might require the disc to input assembler programs. Again the circumstances would not require that the mechanics of the disc be known, since the disc driver operates independently of the assembly process. However, changing the disc driver hardware unit would probably require a totally new disc driver program.

CHAPTER V

THE SYSTEM CONTROLLER AND THE INPUT/OUTPUT PACKAGE

THE SYSTEM CONTROLLER

INTRODUCTION

After initialization, program control is directed to the first of two secondary entry points to the System Controller at which point the input is treated as a source program statement entry. The initialization program is the only program to use this entry point to the System Controller. The other secondary entry point is a return from an editor insert or replace operation. Both these operations involve the inclusion of source statements in the program and the SCB storage of such statements is carried out in the System Controller.

Program control is directed to the main entry point of the System Controller for any program situation requiring user input, with the exception of:

the user responses when printing the introductory text,
the user responses to an edit-veto request.

After the input operation is complete the System Controller is intended to direct program control in any one of eight directions depending on the first character of the input and the status of five different system variables.

PROGRAM CONTROL TRANSFERS

After initialization any response beginning with an equal sign is interpretted as a request to abort the current user program and prepare for another user program.

If this test fails, interrupt mode on the output device is enabled, after being disabled for an input operation. Now five different system variables are examined; if one of these variables is set to a non-zero value, control will be transferred to the program unit requiring the input.

The first variable tested is the ABS/BSS flag. After a prompting message is printed, the ABS/BSS flag is set followed by a return to the System Controller. The user is expected to enter a temporary value to define an undefined symbol in an ABS or BSS instruction operand. Program control returns to the ABS/BSS routine, subroutine VAL, to examine the input.

If the ABS/BSS flag had not been set subroutine CLEAR is called to initialize all lexical variables in preparation of a source program statement either during an edit operation or normal program definition or in preparation of a data address for a Dump operation.

The Dump Directive offers an option of displaying data addresses; the user is requested to type in a response either to end the Dump operation or to dump data address contents. The Dump flag is set in anticipation of such a response to return control to the Dump routine.

Prior to setting the Sequence flag a user sequence request is not accepted. After an error message and re-entry request are printed, the Sequence flag is set to direct program control to the Sequence Directive routine with new statement sequencing data.

Two different system variables involved with edit operations are examined. The flag signalling source statement entries during an editor replace or insert operation will direct program control to subroutine EDIPT, which originally requested the input.

The other editor flag examined is the main edit flag, signalling an edit operation is in progress. Program control is directed to the edit instruction scan program to interpret and execute what should be an edit instruction request.

The seventh and last test is applied to the first character of the input; a colon beginning the entry signals a System Directive. After the colon has been recognized control branches to the program which interprets and channels the System Directives.

SOURCE PROGRAM ENTRY

Failure to satisfy any of the seven tests results in the assembler treating the input as a source program statement. It should be noted that this is the first secondary entry point to the System Controller at which point control branches to the main lexical scan routine, subroutine LEX. Following successful completion of the lexical scan control branches to subroutine ASMBL to allocate space in the SCB to store the program statement.

Data definitions and machine code instructions will be assembled into their appropriate location by subroutine SETCD while comment statements are ignored.

The next instruction, a call to subroutine STSCB to store all statements in the Source Code Block, is the last entry point to the System Controller. Edit operations involved with the insertion of source statements have already performed the lexical scan, the SCB space alloment and the assembled code storage independently of the System Controller.

After the statement has been stored in the SCB, symbols are defined and entered into the Symbol Table. In most cases, program control loops back to the beginning of the System Controller except during an insert involving the entry of more than one program statement where control will return to the insert subsystem.

SYSTEM CONTROLLER MODIFICATIONS

The System Controller is primarily intended to direct the input to the program unit requiring the input. The overall structure of the unit is very simple and could easily be expanded or modified to include transfers to different program units requiring user input.

Changes to source program definition or storage are more likely to be introduced in the subroutines called by the System Controller rather than within the System Controller.

SUBROUTINE REQUESTS

After entry to the System Controller and during examination of the different branch conditions the System Controller calls two I/O subroutines :

DATIN	Request and read user input,
I.ON	Turn on output device interrupt.

Both these subroutines are discussed in the following section on the I/O package.

One other subroutine called is subroutine CLEAR to initialize all variables used in the lexical scan of source program statements or in the scan of an address for a data address dump.

The subroutines called in the section on the source program entry are as follows:

LEX	The main lexical scan program,
ASMBL	Prepare SCB area for statement storage,
SETCD	The main assembly program,
STSCB	Store statement in SCB,
LBDEF	Define label beginning statement.

These subroutines will be discussed in their respective program unit in the next two chapters.

THE INPUT/OUTPUT PACKAGE

INTRODUCTION

The Input/Output Package is comprised of fifteen subroutines to perform five different interrelated input/output functions:

1. Request and read in an input string,
2. Output Ascii records,
3. Interrupt control and service routines,

4. Carriage control programs,
5. Binary to Ascii octal or decimal conversion.

These fifteen different subroutines, which are listed in Table 5.1 in their functional groups form a self-contained unit; program modifications would not likely involve changing the I/O package for it exists as a unit almost totally independent of other assembler features, yet used by almost all assembler features. Subroutine GETCR is normally used for scan purposes, but it is also called in DATIN to retrieve the first character from the input buffer to ensure at least one character has been read before returning from DATIN.

On scanning the program listing it may seem haphazard to arrange subroutines TTY.I, TTY.P, I.ON and I.OFF one after the other not according to functional group. This arrangement within the I/O package is convenient to subroutine CNFIG for all I/O machine instructions reside within these four subroutines.

With the exception of the binary to Ascii conversion all other I/O functions have been designed around the I/O facility of the Hewlett Packard Basic compiler for the 2100A computer; also they are in some way reliant on the output function. For this reason the output unit is discussed first.

OUTPUT CONTROL

The output function is called from various points throughout the program; Subroutine TTY.P is the main driver program calling subroutines INIT and GETCH.

Subroutine TTY.P

```
Calling Sequence
     LDA < Character length of output >
     LDB < The address of the output buffer >
```

On entry if

```
(A) > 0  then print (A) characters followed by a carriage
         return and line feed,
(A) = 0  then print only a carriage return and line feed,
(A) < 0  then print -(A) characters only.
```

TTY.P will output each character using the non-interrupt transfer routines discussed in Appendix D. By typing any key on the keyboard the user may interrupt his program if the interrupt mode had been enabled before the input operation. Interrupt mode is disabled during the printing of the introductory text. Output operations in non-interrupt mode cannot be interrupted. Interrupt mode is enabled in the System Controller after the Abort test. On an interrupt the control flag is cleared to turn off device activity before calling the interrupt service subroutine.

On a normal completion a carriage return and line feed are output if requested earlier.

Subroutine INIT

```
Calling Sequence
     LDA < Character length of output >
     LDB < The address of the output buffer >
```

INIT saves the register contents and sets a pointer depending on the sign of (A) on input to TTY.P.

Subroutine GETCH

```
Return  P+1  Buffer empty
        P+2  Character in (A)
```

GETCH retrieves the next character, removes the parity bit and returns the character in (A) to the second return address. The first return address indicates that the text has been output.

INTERRUPT CONTROL

An interrupt is a user initiated action to halt some present activity. For the purposes of the assembler the interrupt mode is used primarily to interrupt the printing of warning messages to the user.

The interrupt service subroutines are called from several locations in the assembler. Subroutine I.OFF and I.ON are both very straightforward and not apt to be altered. Subroutine I.STP uses a very simple handling of an interrupt condition. The subroutine could easily be changed to treat the interrupts in a different manner.

Subroutine I.OFF

I.OFF turns off the device interrupt mode by setting a NOP, a no operation instruction, into the device interrupt location and clears the device control flag to turn off read mode.

Subroutine I.ON

I.ON turns on the device interrupt by storing a jump to the interrupt service subroutine in the device interrupt location. The device is set to read mode and set to look for input.

Subroutine I.STP

I.STP is the actual interrupt service subroutine; it will call I.OFF to turn off interrupt mode and then call TTY.P to print STOP before returning to the System Controller.

CARRIAGE CONTROL

The carriage control calls are also called throughout the assembler; often they precede a call to the output function to print the output on a new line.

Subroutine CRLFD

CRLFD will clear the A register and call TTY.P to output a carriage return and line feed.

Subroutine NWLNS

Calling Sequence

```
    LDA < Two's complement number of CR-LF >
```

NWLNS will output the two's complement number of carriage return-line feeds as specified in (A) by successive calls to CRLFD.

INPUT CONTROL

Subroutine DATIN is the main input subroutine calling TTY.I to perform the input operation and PROCS to store each character in the input buffer.

Subroutine DATIN is primarily called from the System Controller but there are separate calls from the initialization program and for a response to the edit-veto request.

Subroutine DATIN

Return (A) First character of input

DATIN outputs the read prompt, the @ and the bell characters before calling subroutine TTY.I. On returning from TTY.I length and address pointers for character retrieval and statement storage are set. A call to subroutine GETCR will return the first character of the input in (A).

Subroutine TTY.I

```
Calling Sequence
    LDA < Length of the input buffer, 72 characters >
    LDB < Address of the input buffer >

Return   (A)   The number of characters input or -1 on
               buffer overflow
```

TTT.I saves the length and address pointers and sets the device to input mode. Using the non-interrupt request routines presented in Appendix D, each character is read in, immediately after each character is read in subroutine PROCS is called to store each character in the buffer.

Before returning to DATIN, TTY.I turns off the input device read mode.

Subroutine PROCS

```
Calling Sequence
    LDA < Character to be stored >

Return   P+1   Get next character
         P+2   (A)  Character count
               (B)  Minus one value on buffer overflow
```

PROCS will ignore superfluous characters, in particular the line feed and null character, and pack all valid characters into the input buffer. The back space character, the left arrow, permits the back up of one character. Any number of back space entries are permitted but multiple back spacing cannot backup beyond the original buffer address.

Buffer overflow will be flagged in PROCS but is not acted on. The second return address is set after recognition of a carriage return character to end the input string.

BINARY TO ASCII CONVERSION

Although, not directly related to the other I/O functions the binary to Ascii conversion facility is used in the List program to convert the statement number to Ascii characters and in the Dump program to convert the register contents, after execution.

Subroutines CNDEC, CNOCT, CNBIN, and DVUKN are all Hewlett Packard library programs which have been modified slightly to simplify storage and output.

Subroutines CNOCT and CNDEC

```
Calling Sequence
     LDA  < Value to be converted >

Return   (A)  The least two significant digits
         (B)  The address of the most significant digits
```

CNDEC and CNOCT specify ten and eight decimal, respectively for the conversion. The address returned in (B) will be used as input to subroutine TTY.P.

TABLE 5.1

INPUT/OUTPUT SUBROUTINES IN FUNCTIONAL GROUPS

1. INPUT:

DATIN	Request and read user input
TTY.I	Perform input operation
PROCS	Character processing for input

2. OUTPUT:

TTY.P	Perform output operation
GETCH	Character processing for output
INIT	Initialize for output

3. INTERRUPT CONTROL:

I.ON	Turn on interrupt
I.OFF	Turn off interrupt
I.STP	Interrupt service

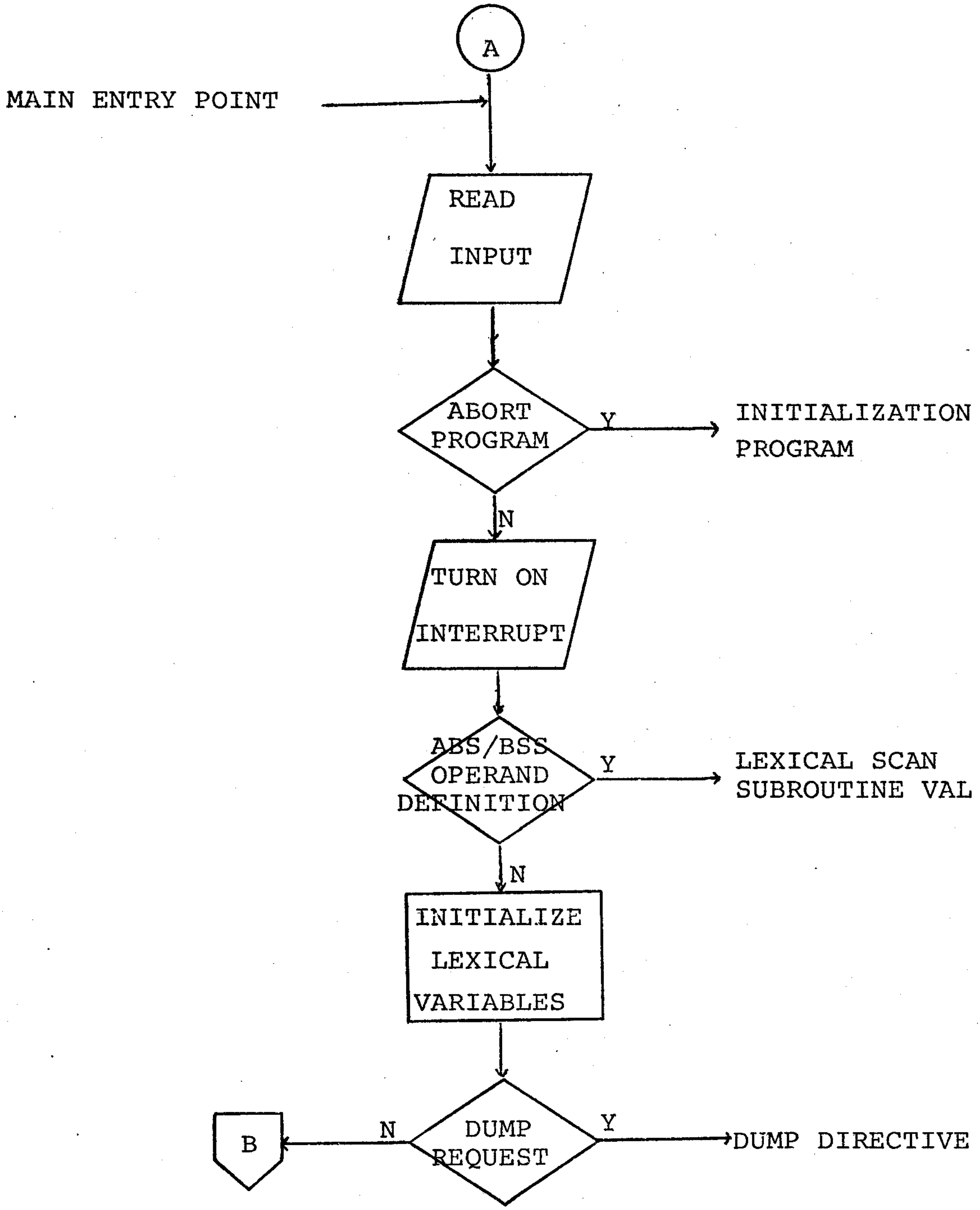
4. CARRIAGE CONTROL:

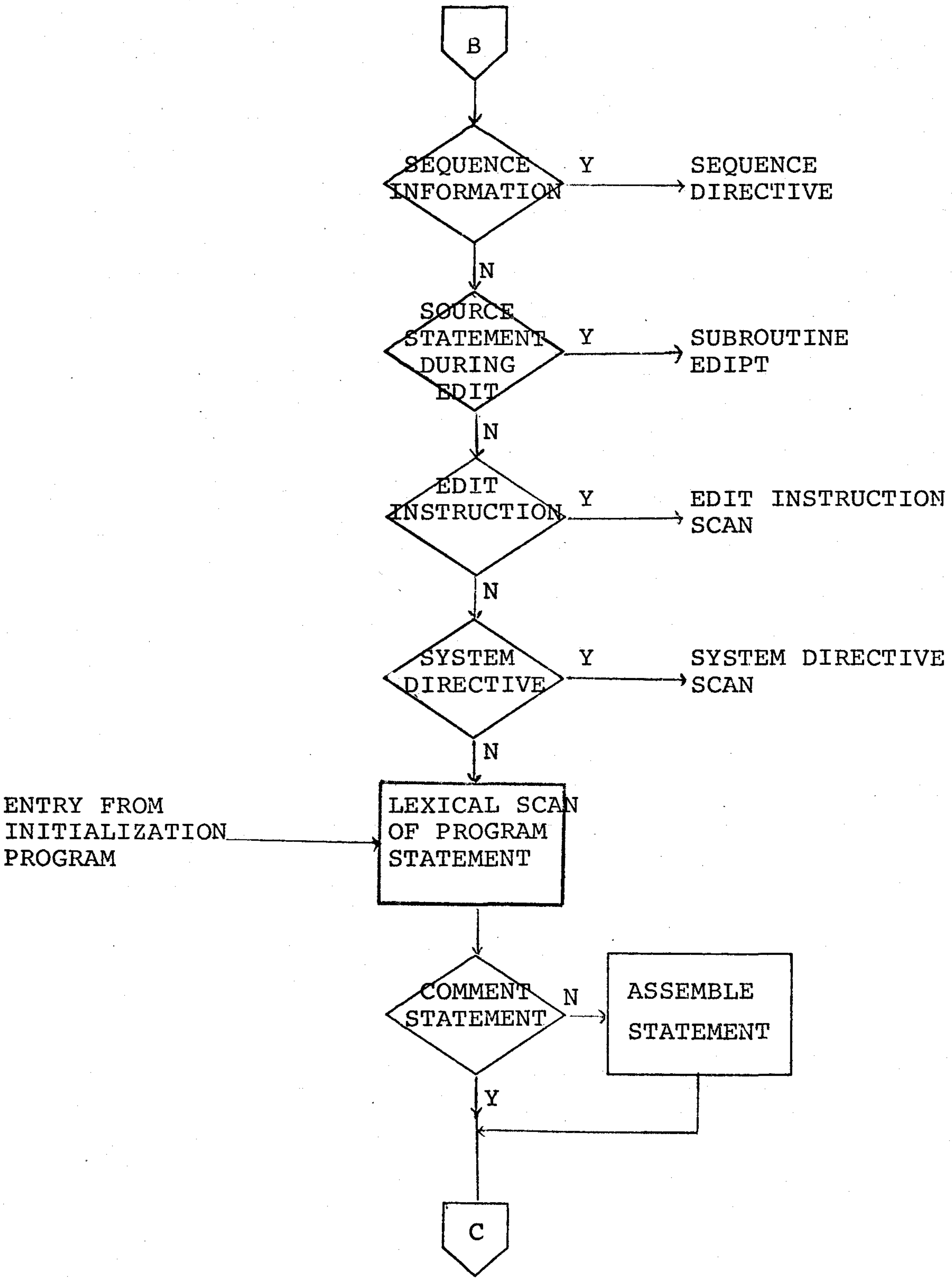
CRLFD	Output carriage return-line feed
NWLNS	Output multiple CR-LF

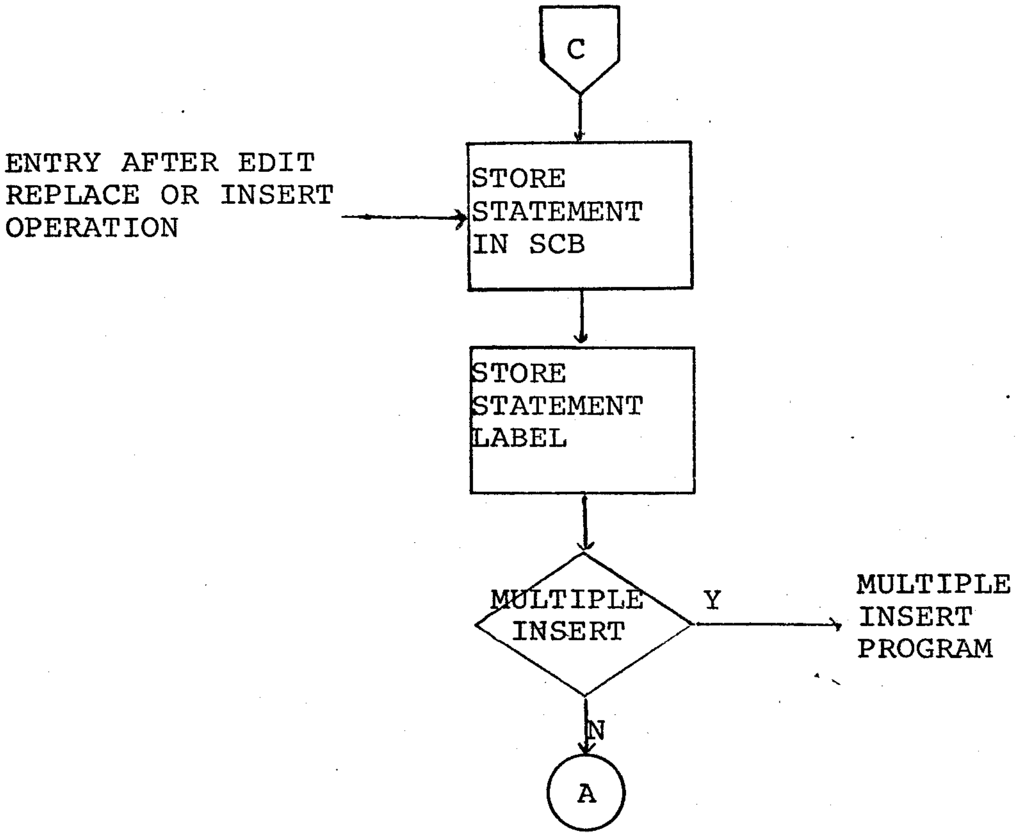
5. BINARY TO ASCII CONVERSION:

CNOCT	Convert to Ascii octal
CNDEC	Convert to Ascii decimal
CNBIN	Stored converted value
DVUKN	Divide value to be converted

FIGURE 5.1 SYSTEM CONTROLLER FLOW DIAGRAM




B
SEQUENCE INFORMATION
Y
SEQUENCE DIRECTIVE
N
SOURCE STATEMENT DURING EDIT
Y
SUBROUTINE EDIPT
N
EDIT INSTRUCTION
Y
EDIT INSTRUCTION SCAN
N
SYSTEM DIRECTIVE
Y
SYSTEM DIRECTIVE SCAN
N
ENTRY FROM INITIALIZATION PROGRAM
LEXICAL SCAN OF PROGRAM STATEMENT
COMMENT STATEMENT
N
ASSEMBLE STATEMENT
Y
C


C
ENTRY AFTER EDIT
REPLACE OR INSERT
OPERATION
STORE
STATEMENT
IN SCB
STORE
STATEMENT
LABEL
MULTIPLE
INSERT
Y
MULTIPLE
INSERT
PROGRAM
N
A

CHAPTER VI

LEXICAL SCAN AND NUMBER MANIPULATION

LEXICAL SCAN

INTRODUCTION

Subroutine LEX is the main lexical scan program used to analyse source program statements. LEX is called from three different locations in the assembler:

The System Controller,
Subroutine EDIPT,
Subroutine DELTE.

A call from the System Controller is for the analysis of source program statements entered during the normal program definition. Subroutine EDIPT will call LEX to scan source program statements involved in an edit insert or replace operation.

DELTE is an edit subroutine for deleting statements from the assembled program. On an edit operation involving the deletion or replacement of a program statement, the lexical scan is necessary to return statement label information and Memory Reference operand information. A label beginning a statement to be deleted is no longer defined after the edit operation; Subroutine LEX returns information used to locate the symbol in the Symbol Table. Operand analysis is unnecessary except for Memory Reference instructions; operand information must be returned to adjust forward reference pointers, if necessary, after an edit operation.

The section "Subroutine LEX" describes the lexical scan and emphasizes some of the changes required in the instruction scan for the instructions which are not implemented in accordance with the standard Hewlett Packard assembly language.

Following the section "Subroutine LEX" is a discussion on changes which could be implemented. The remainder of Chapter VI is a detailed discussion of the important lexical routines. This group of subroutines may be further divided into three groups, those involved with character manipulation, the lexical support routines used in instruction analysis and the number forming subroutines.

Subroutine LEX

INTRODUCTION

The available assembler instructions have been divided into fifteen different groups for operand analysis; these fifteen groups and their operands have been described in Appendix A. After the group type has been established the program falls through a logical cascade operation which eventually locates the value of the group number by comparing the group type value with all possible group number values. Following the comparison test for each group type is the program unit to interpret the operand for the particular operand type.

Excepting Memory Reference instructions, all operand recognition and evaluation is within the lexical programs. Memory Reference operands will be examined but not evaluated until the instruction is about to be assembled.

SOURCE STATEMENT SCAN

LEX begins a character by character scan to analyse the statement entry. The first character must be one of:

a blank,
a letter or a period,
an asterisk.

Any other character will result in a call to a lexical error message; all lexical error messages are listed in Table 6.1.

An asterisk signals a comment statement; no further scan is necessary. The assembly flag has been set for a comment statement by subroutine CLEAR; LEX returns to the calling program.

A blank signals that no label is present; the program continues by advancing to the next non-blank character in preparation for the instruction mnemonic.

An alphabetic character or a period signals a label is present. Using subroutine LABRD the label is read into the temporary buffer for statement labels. Conventional Hewlett Packard assembly will truncate any label greater than five characters and issue a warning message. For this assembler at least one blank terminator character must follow the fifth or last label character or an error message will be printed with the statement being ignored.

A label flag is set for the presence of a statement label with an error message being printed for a doubly defined label and the statement again being ignored.

The instruction mnemonic is packed into a two-word buffer to facilitate instruction look up by subroutine MNEM.

After returning from MNEM, the program begins the logical cascade of the different instruction types.

On matching the instruction type number, operand analysis may begin. Generally, the scan of machine code instructions adheres to standard Hewlett Packard definition. The restrictions pertaining to Memory Reference instructions have already been discussed. One further deviation from standard assembly is the use of a symbol in an Input/Output operand in the place of a channel number value.

Normally the channel number is in the range 0 to 63 but it may be equated to a symbol such that a symbol replaces the integer in the operand. It was decided that an I/O instruction with an undefined operand would not be accepted.

This is the first instance of statements with undefined operands not being accepted. Memory reference instructions having undefined operands will be accepted and retained for the symbol tables have been specially designed to hold such references. The Memory Reference instruction offers a 10 bit address field to link forward references while an I/O instruction has only a six bit field for the channel number. This is intended to discourage the use of I/O instructions for the user program area is restricted in size; it should encourage the use of the Dump Directive after execution.

On recognition of the END instruction control branches to the execution programs, except during an edit operation which must be completed before beginning execution.

Data definitions have been discussed in Chapter III in the section on mnemonics and pseudo ops. One important restriction is that the data definition may be no longer than 28 words in length. The only exception is the BSS pseudo op which may be 128 words.

Before scanning any data defintion the 28-word data buffer is cleared. As the instruction is scanned each data value is stored in the buffer; this is particularly relevant to the ASC, DEC, and OCT pseudo ops which may involve more than one word in the definition. An error in the data entry will cause the whole statement to be ignored. LEX will call subroutines to input numeric terms for OCT and DEC but the terminator character after each value is checked within LEX.

The remaining pseudo ops are at most a one-word entry to the buffer. The BSS and EQU pseudo ops do not use the data buffer.

Any symbol in a pseudo op operand is restricted to a data address symbol. This is important in the scan of the ABS, BSS, EQU and DEF pseudo ops. The ABS and BSS pseudo ops have been discussed in Chapter III and Appendix A and need not be dealt with any further.

The EQU pseudo op is regarded as a data definition of length zero but an assembly address must be set to store a Symbol Table address for the label which must precede the instruction. The operand address is stored in the last position of the data table area with the assembly address corresponding to this location. Before returning, the upper bound of the data table is decremented to prevent an overwrite of this instruction.

The EQU instructions is another instance of a statement being ignored due to an undefined operand symbol but in this case it is in accordance of the Hewlett Packard definition.

An undefined symbol in a DEF pseudo op operand is again handled in a different manner as presented in Chapter III.

The DEF pseudo op is the last instruction type. Failure by the program to match the instruction type number within LEX signals a program error. An error message is printed followed by a computer halt (HLT 33B); a re-entry request is not presented. Operator intervention is required to correct the program fault. This intervention would probably involve referring to an assembler listing of the program to determine core addresses of the variables involved in the lexical scan and examining actual core locations to determine the error. To correct the program fault, it would probably be necessary to change some memory locations to restore their proper value and reset the program location counter either to continue assembler activity on the current user program or to abort the current program and initialize for a new user program.

PROGRAM MODIFICATIONS

In considering the implementation of any changes the overall program changes must be weighed against what advantages could be gained.

The DEC and OCT pseudo ops instructions are totally rejected if any part of the statement is in error. Changes could be made strictly within LEX to ignore any data item in error and print a warning message pointing to the ignored value. To ignore the data item in error is trivial and presumably to point to the data item in error is also trivial. But would such a change be advantageous?

A user entering several data values in one statement usually would not want an item excluded due to an error. With the present implementation a user has greater control over the program structure by the rejection of the statement on a single error.

It, therefore, seems best to assume that changes to the lexical scan would have to be implemented as a result of expanding the set of available instructions or relaxing the restrictions on the present instruction set.

Relaxing some user program restrictions would definitely be significant within LEX. Operands for the DEF pseudo op could be expanded to resemble a Memory Reference operand or undefined references during an edit operation may be permitted.

Changes regarding Memory Reference operands or undefined symbols in I/O instructions could be considered.

However, the program modifications necessary would probably far outweigh the advantages of such changes.

Expanding the instruction set to include the REP pseudo op or floating point arithmetic requests would require changes throughout the assembler. Allowing the user to enter multiple instructions would require a much more thorough scan. Such a change would necessarily involve a distinction between Alter Skip, and Shift Rotate instructions in the Instruction Table and a provision for the instructions which belong to both instruction groups. Subroutine LEX would be responsible for scanning these instructions and forming the multiple instruction.

Seemingly storage allocation would have to be rearranged. The available storage size does not permit these inclusions without usage of the disc input driver to load either ancillary subroutines or program segments as needed. It would probably be best to leave all assembler and program tables in memory at all times and rely on the controller unit to manage disc transfers of program segments.

In the long run, the advantages of such changes should far outweigh the work involved in implementing such a change. Such changes would probably be beneficial to a more experienced user without defeating the original purpose of the assembler.

CHARACTER MANIPULATION SUBROUTINES

The remainder of Chapter VI is devoted to the discussion of the different subroutines used in the lexical scan and for number handling purposes. Some of these subroutines have important uses outside the lexical scan but their primary function is as part of the lexical scan.

The subroutines involved with character manipulation are listed in Table 6.2 and will be discussed first.

Subroutine BCKSP

BCKSP will back up the scan of the input buffer by one character by adjusting the one's complement word count and the address word to the next character in the buffer. No check is needed for backing up beyond the original buffer address for the situation never occurs.

Subroutine GETCR

```
Return  P+1  Buffer empty
        P+2  Next character from input buffer in (A)
```

GETCR is the only subroutine to retrieve a character from the input buffer. For each call to GETCR the one's complement character count is incremented; when this value goes to zero the buffer has been fully scanned. The second return address returns the character in (A).

Subroutine NTBLK

```
Return  P+1  Non-blank character not found
        P+2  Next non-blank character in (A)
```

Using GETCR, NTBLK will search for the next non-blank character in the buffer.

Subroutine RDCOM

```
Return  P+1  No comma found in buffer
        P+2  Comma read
```

Using GETCR, RDCOM will position the buffer pointers to retrieve the first character after the comma on the next call to GETCR.

Subroutine TRMCK

```
Return  P+1  Valid termination
        P+2  Invalid termination, character in (A)
```

TRMCK uses GETCR, but it has a different function in that it is examining the character to be a terminator, either the blank character or the end of line condition. The first return address signals valid termination; the second return exits with the character in (A) for further analysis.

LEXICAL SUPPORT RETURNS

The lexical support subroutines will be described in their approximate order of occurrence in LEX. Table 6.3 lists these subroutines; error messages associated with these subroutines are listed in Table 6.4.

Subroutine LABRD

```
Calling Sequence
     LDA < First character of symbol, (A) > 0 >
         < First character not read, (A) < 0 >
     LDB < Address of symbol buffer >

Return  P+1  First character not a letter or a period,
             character in (A)
        P+2  Symbol read
```

LABRD is the symbol reading subroutine for reading statement labels and operand symbols. The first return address is applicable if on entry (A) signals that the first character has not been read. Normally, no error message is generated unless nothing was read.

Ordinarily LABRD will read up to five characters into the symbol buffer. Numeric characters will be stored as Ascii characters so that these characters can be output if the symbol must be printed separately.

Subroutine LETPR

Calling Sequence
    LDA < character to be examined >

Return	P+1	Character in (A) not alphabetic or a period
	P+2	Alphabetic or period character in (A)

LETRP is called by LEX and LABRD to examine a character to be alphabetic or a period.

Subroutine LOKUP

Calling Sequence
    LDB < Address of the symbol buffer >

Return	(A) > 0	The program address of the symbol
	(A) = 0	Symbol not found in Symbol Table
	(A) < 0	Undefined symbol
	(B)	Symbol Table address of symbol

Given the symbol buffer address LOKUP calls subroutine FIND to locate the symbol position in the Symbol Table. An undefined symbol has had previous references but has not been defined as a statement label.

Subroutine FIND

Calling Sequence
    LDB < Address of the symbol buffer >

Return	(A) = 0	Symbol not in Symbol Table
	(B)	Symbol Table address of symbol

FIND applies the hashing function to yield the relative table position to begin a linear search. The relative table position is converted to an actual storage address to begin the search for the next free area to store the symbol or the symbol position in the table.

If the table area is not occupied, the symbol has not been previously entered; control returns to LOKUP. A symbol entry in this location will be checked word by word with the symbol being sought.

Reaching the end of the table will immediately cause the search to continue at the beginning of the table in a circular fashion. Failure to find the symbol or a free position for the symbol indicates the Symbol Table is full and results in an abnormal program abort.

Subroutine MNEM

Subroutine MNEM finds the assembly skeleton of the instruction mnemonic from the Instruction Table. Using the mnemonic which has been packed into a two-word buffer by subroutine LEX, MNEM performs a binary search with the first section of the Instruction Table for the first two characters of the mnemonic.

After finding the instruction position in the first section of the Instruction Table, this position pointer is adjusted to reference the corresponding position in the second section of the Instruction Table.

Further corrections may be included to the position pointer if there is more than one mnemonic in the Instruction Table beginning with the same first two letters. The pointer is set to reference the position of the first mnemonic is such a case.

Using the position information, a linear search is set to match the third character of the mnemonic with the characters stored in the second section of the Instruction Table. Since six different mnemonics may begin with the same two letters, the linear search is attempted six times.

Failure to match either the first two characters or the third character of the mnemonic with the appropriate entry in the Instruction Table will signal an undefined mnemonic which results in an error message and return to the System Controller.

On successful recognition, the instruction number and skeleton assembly code are retrieved from the Instruction Table.

For the simple task of determining the type of assembly an assembly flag variable is used rather than making reference to the assembly skeleton. Initialized to zero by subroutine CLEAR, the assembly flag is used to denote:

pseudo operation (data definition)	(-1),
comment statement	(0),
machine code instruction	(1).

Subroutine RANGE

```
Calling Sequence
     LDA < Value in operand >
     LDB < Two's complement of upper bound value >

Return  P+1  Valid termination
        P+2  Invalid termination
```

RANGE is intended to examine the operand values for the Input/Output and Extended Arithmetic Register Reference instructions. RANGE checks the operand value to be positive and within range and includes the operand value with the assembly skeleton.

Subroutine TRMCK is called to check for valid termination; RANGE uses the two return addresses depending on TRMCK.

Subroutine OPREC

All Memory Reference operands, some pseudo-op operands and data addresses to be output by the Dump Directive will be read in and retained. OPREC calls BSKSP, TRMCK, LABRD, and NUMBR. NUMBR reads in decimal or octal integers. OPREC does not rely on RANGE to check operand values for RANGE will include the operand value with the assembly skeleton and include a separate call to TRMCK.

Subroutine STDAT

```
Calling Sequence
     LDA < Data value to be stored > .
```

Before any data definition is scanned, the data buffer is cleared and a counter is set. STDAT will store data values from the data buffer during the scan of the pseudo op.

Data definitions using the buffer have an imposed bound of 28 words since this is only a temporary buffer. Failure to comply with this restriction results in a warning message with the statement being ignored. This data is held in the buffer to be assembled after the lexical scan.

Subroutine LABCK

Return	P+1	No operand symbol
	P+2	Operand symbol is not defined
	P+3	Operand symbol defined, address in (A)

Using OPREC, LABCK will read in the operand for pseudo ops having address operands and data addresses for the Dump Directive. With three different return addresses operand recognition and analysis for the different instruction types is easier.

Subroutine DATRG

Calling Sequence

```
    LDA < Address to be examined >
```

DATRG checks the address to be within bounds of the program data area or the available base page area. DATRG is primarily a lexical support routine but is also required by the Dump Directive.

Subroutine VAL

After a prompt from VAL the user is to type in a temporary value for an undefined symbol in an ABS or BSS operand.

The ABS/BSS flag is set followed by a return to the System Controller to input a value. The System Controller will return program control to VAL to clear the ABS/BSS flag and substitute the value for the undefined symbol.

Reading in a value as such requires several precautionary steps; the original statement entry resides in the input buffer and the statement length in a special variable. Both of these must be retained if the statement is to be stored in the Source Code Block after assembly.

After each input operation the character length of the input is stored in a special input variable. Before reading in a temporary value the character length of the original program statement must be stored in a temporary location, not involved with an input operation so that this value may be retrieved after the temporary value is input; the input buffer address is altered so that an auxiliary buffer is used to input the value. Pointers must be retained to scan the buffer. After the input operation is complete the input buffer address and the statement length are then restored to their proper variable.

An error in the entry of a temporary value results in the original program statement being ignored.

NUMBER MANIPULATION

INTRODUCTION

The number handling subroutines are used throughout the assembler but are primarily called by the lexical routines. There are four major categories with which number usage is associated:

Octal integers for the OCT pseudo op,
Octal and decimal integers for operand expressions,
Floating point numbers and decimal integers for the DEC pseudo op,
Decimal integers generally involved with statement numbers.

Before discussing the four different number types it should be pointed out that there are eight error messages, listed in Table 6.5, shared by the number forming subprograms. In the event of an error, subroutine ERROR is called to print the error message and re-entry request. During initialization program control returns to the calling point but normally control passes to the System Controller.

OCTAL INTEGERS - Subroutine OCTIN

Return (A) Octal integer

Subroutine OCTIN is called strictly by LEX to form octal integers for the OCT pseudo op. The next non-blank character is examined to be a sign with the sign flag set accordingly. Failure to locate any data or a solitary sign necessitates a branch to the appropriate error routine.

Initially a zero value is set into a temporary variable. While constructing the value each new digit will be added into the previous value after the value has been shifted three times to the left. The shift used is a left circular shift with overflow checked after each shift by examining bit 0.

On finding a character which is not an octal digit OCTIN checks that at least one valid octal digit has been input. If so, OCTIN assumes that this character is the terminator and that the value has been defined. Like all other number routines a terminator is returned to the buffer and not checked in OCTIN.

Before returning one last check for a negative sign is taken with the two's complement value returned if necessary.

If no valid octal digits were input before encountering the terminator an error message is output.

Subroutine OCTCK

```
Calling Sequence
   LDA < Character to be examined >

Return  P+1  Character in (A)
        P+2  Octal digit in (A)
```

OCTCK is the only subroutine called by OCTIN to examine each character to be an octal digit.

OPERAND INTEGERS - Subroutine NUMBR

```
Return  P+1  First character not a number
        P+2  Decimal or octal integer in (A)
```

Subroutine NUMBR is called to read in operand integers, either decimal integers or octal integers flagged by a B, immediately following the value. NUMBR will form an octal and decimal value from the input until it can determine which value to return.

Like OCTIN, NUMBR will check for no operand data, a solitary sign and retain sign information. Each character will be examined by subroutine DECHK to be a decimal digit but a separate internal check is required to test a decimal digit to be an octal digit as well.

Before including a new decimal digit the previous value is multiplied by ten using shifts and additions. A valid octal digit is included after three shifts. In each case overflow will be checked before accepting the new digit.

Any character which does not satisfy the octal digit test results in an error flag being set; the scan must continue for this number is apt to be a decimal value. The first character rejected by DECHK is tested to be the character B signalling an octal digit. If this character is a B and the octal error flag is clear, the octal value is returned, but if the error flag is set there will be an error message.

Any character other than B is assumed to be a terminator and is returned to the buffer; a decimal value is returned.

Subroutine DECHK

```
Return  P+1  Character in (A)
        P+2  Decimal digit in (A)
```

All number forming subroutines involved with decimal values will use DECHK to check each character being scanned. DECHK examines the character to be in the range of decimal digit characters and returns the character if the test fails.

DEC PSEUDO OP

The DEC pseudo op may have floating point, or decimal integer operand values even though floating point arithmetic is not available. Subroutine CONST will initiate the input of floating point constants.

Subroutine CONST

Return (A) and (B) Floating point constant

CONST advances up to the next non-blank character, sets the sign flag and checks for a solitary sign. CONST calls NUMCK which controls the Ascii to binary conversion.

Subroutine NUMCK

Return (A) and (B) Floating point constant

NUMCK is very similar to the subroutine NUMCK is the Hewlett Packard Basic compiler for Ascii to binary conversion of floating point numbers. Changes have been made to ignore leading zeros in an exponent term and error handling has been altered. As part of the number input NUMCK calls:

.PACK	To normalize and pack a floating point constant,
NORML	To normalize a value with its exponent,
MPY	To multiply an unpacked number by ten,
DBY	To divide an unpacked number by ten,
MPY	To multiply an integer by ten.

The program logic has not been changed from the program listings of the Hewlett Packard Basic compiler. Since these programs are available in Hewlett Packard system listings and since they exist as support programs they will not be discussed any further.

DECIMAL INTEGERS

The DEC pseudo op, by definition, may have decimal integer operand values. Rather than write an additional program for strictly decimal integer input it became necessary to provide a real to integer conversion.

The presence of subroutine IFIX in the Hewlett Packard system listings provided the necessary conversion as well as a check on the exponent of a floating point number.

All that remained was to write a simple subroutine to determine a real or integer value from the floating point number stored in (A) and (B). Two variables DPFLG, the decimal point flag and EFLG, the exponent flag, have the format.

```
DPFLG =  0  No decimal point present
      =  1  Decimal point present
EFLG  = -1  No exponent term
      =  0  Exponent term
```

Subroutine TYPCK

```
Calling Sequence
     LDA < Floating point number >
     LDB < Floating point number >

Return  P+1  Floating point number in (A) and (B)
        P+2  Integer in (A)
```

TYPCK examines the decimal point flag and the exponent flag and will call subroutine IFIX if neither of these variables were set in NUMCK.

Subroutine IFIX

Calling Sequence

```
LDA < Floating point number >
LDB < Floating point number >
```

Return (A) Integer value

IFIX converts the floating point value to a single word integer.

Subroutine GTNUM

GTNUM calls CONST to input a positive decimal integer value. GTNUM will not accept negative or real number values.

Subroutine TWINT

```
Return  P+1  One integer    valid termination
        P+2  One integer  invalid termination
        P+3  Two integers   valid termination
        P+4  Two integers invalid termination
```

TWINT is set to call GTNUM twice to input one or two positive integers. The different return conditions are important when examining the veto flag on an edit request. Normally, the third return is the only acceptable return for statement number input. Termination is checked by TRMCK and as in all other cases the terminating character is returned to the buffer by BCKSP.

SUMMARY

The number handling subroutines and the main features of the lexical scan have been presented. Programs to input and store floating point numbers have been successfully implemented. Further implementation of floating point arithmetic subroutine requests is definitely possible.

Once the lexical scan is completed control returns to the calling program. In the case of a call from the System Controller statement assembly and storage follow immediately.

TABLE 6.1 LEXICAL ERROR MESSAGES

Error messages with an alternate label, i.e., (ERR6), signal base page error messages.

LABEL	ERROR MESSAGE
LXR1	FIRST CHARACTER NOT FOUND
LXR2	ILLEGAL FIRST CHARACTER
LXR3	BAD DATA FOLLOWS LABEL
LXR4	DOUBLY DEFINED LABEL
LXR5	INSTRUCTION NOT FOUND
LXR6 (ERR6)	NO OPERAND FOUND
LXR7	BAD DATA FOLLOWS OP CODE
LXR8	BAD DATA IN OPERAND FIELD
LXR9 (ERR5)	ILLEGAL CHARACTER BEGINS LABEL
LXR10 (ERR8)	UNDEFINED LABEL IN OPERAND
LXR11 (ERR4)	ILLEGAL OPERAND TERMINATION
LXR12	ILLEGAL INSTRUCTION DURING EDIT
LXR13 (ERR3)	OPERAND VALUE OUT OF RANGE
LXR14	NO LABEL PRECEDES EQU PSEUDO OP
LXR15	ADDRESS MUST BE POSITIVE
LXR16	INSTRUCTION NOT FOUND
LXR17 (ERR7)	OPERAND IS UNDEFINED
LXR18	UNDEFINED LABEL NOT PERMITTED WITH DEF DURING EDIT

LABEL	ERROR MESSAGE
LXR19	OPERAND VALUE MUST BE GREATER THAN ZERO

TABLE 6.2 CHARACTER MANIPULATION SUBROUTINES

SUBROUTINE	FUNCTION
BCKSP	Back space one character in the input buffer
GETCR	Retrieve the next character from the input buffer
NTBLK	Get the next non-blank character from the input
RDCOM	Read up to a comma in the buffer
TRMCK	Check for a termination character

TABLE 6.3 LEXICAL SUPPORT ROUTINES

SUBROUTINE	FUNCTION
DATRG	Check for data address
FIND	Find Symbol Table address of symbol
LABCK	Read in operand, examine symbol
LABRD	Read a symbol
LETPR	Check for period or a letter
LOKUP	Look up Symbol Table address
MNEM	Find assembled instruction from mnemonic
OPREC	Read in and interpret operand
RANGE	Check Channel Number and Shift Count range
STDAT	Store data value in temporary data buffer
VAL	Prompt definition of undefined ABS or BSS symbol

TABLE 6.4 ERROR MESSAGES FOR LEXICAL SUPPORT ROUTINES

SUBROUTINE	ERROR MESSAGE
DATRG	ADDRESS BEYOND PROGRAM BOUNDS
FIND	SYMBOL TABLE OVERFLOW
LABRD	NO LABEL FOUND
MNEM	ILLEGAL ASSEMBLER INSTRUCTION
OPREC	OPERAND VALUE OUT OF RANGE
	ILLEGAL OPERAND TERMINATION
	MINUS SIGN PRECEDES LABEL
	MINUS SIGN PRECEDES ASTERISK
	INDIRECT REFERENCE PERMITTED ONLY WITH MEMORY REFERENCE AND DEF INSTRUCTIONS
RANGE	OPERAND VALUE OUT OF RANGE
STDAT	DATA INPUT EXCEEDS IMPOSED LIMIT

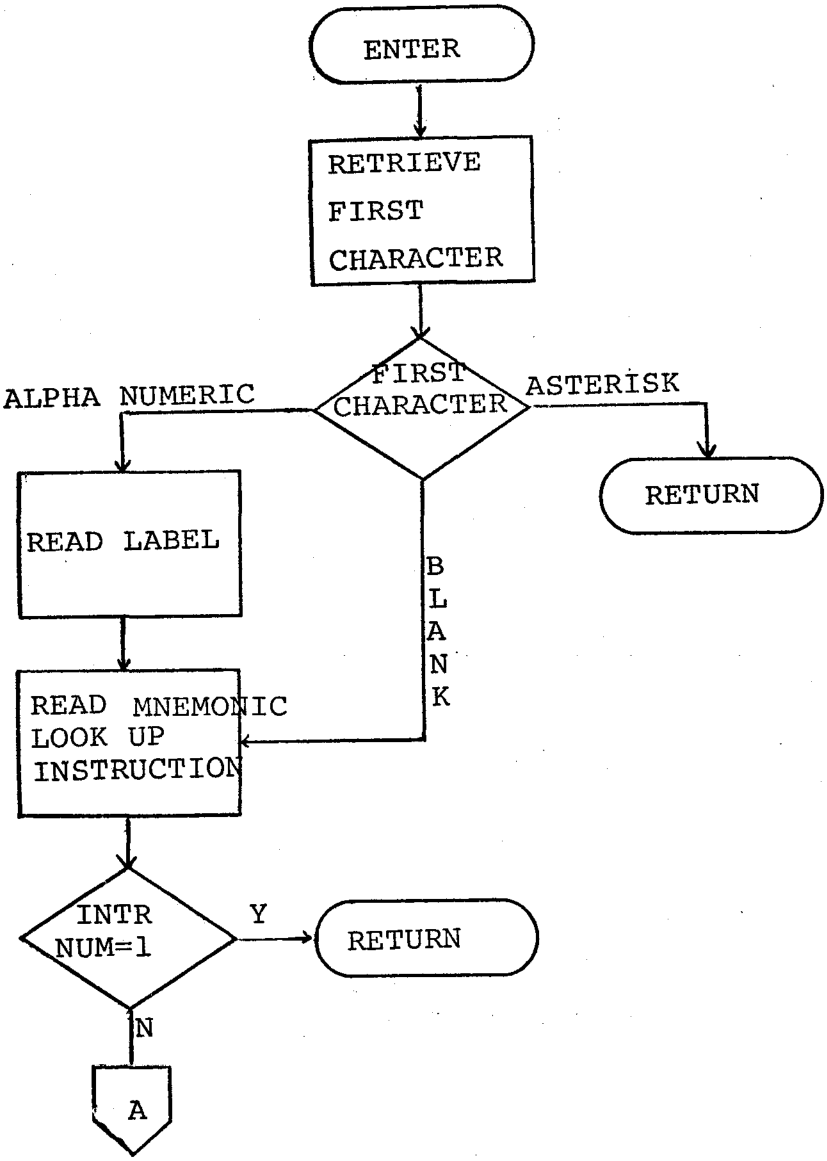
TABLE 6.5 NUMBER PROGRAM ERROR MESSAGES

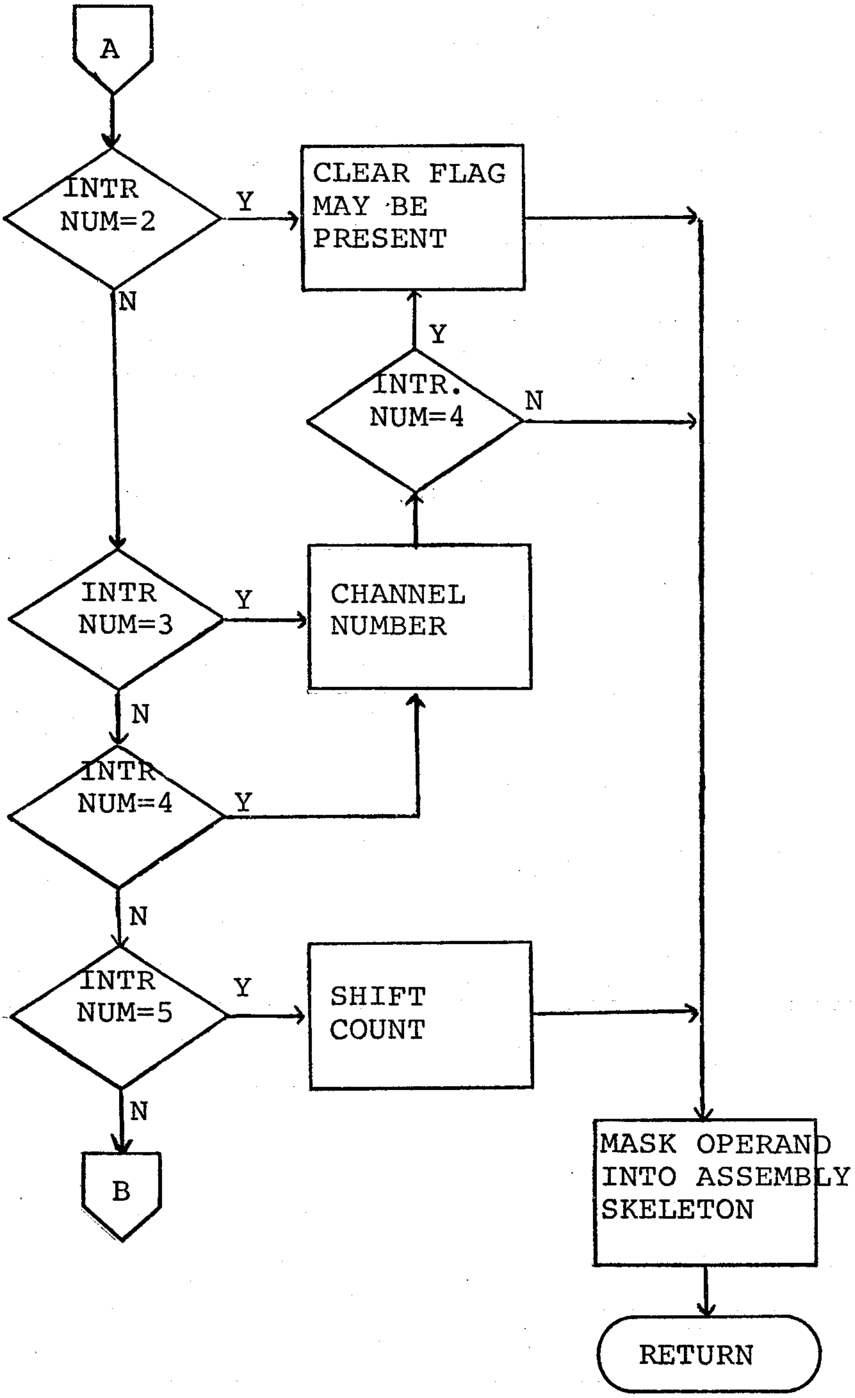
LABEL	ERROR MESSAGE
NUMR1	NO OPERAND DATA FOUND
NUMR2	SOLITARY SIGN
NUMR3 (ERR1)	BAD DATA INPUT
NUMR4	ERROR IN EXPONENT
NUMR5	INTEGER OVERFLOW
NUMR6	POSITIVE INTEGER EXPECTED
NUMR7	BAD DATA FOLLOWS INTEGER
NUMR8	REAL NUMBER OUT OF RANGE

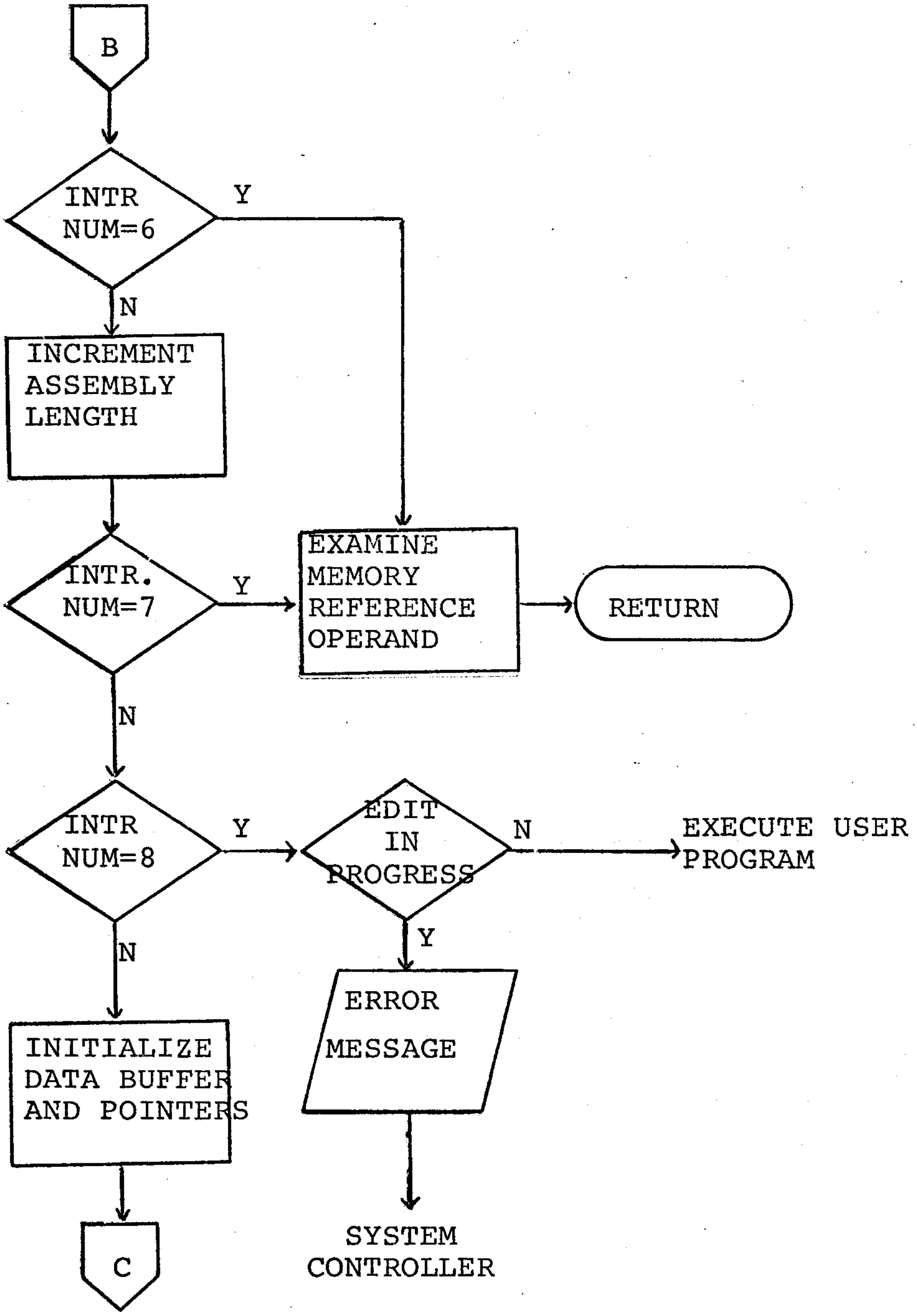
FIGURE 6.1

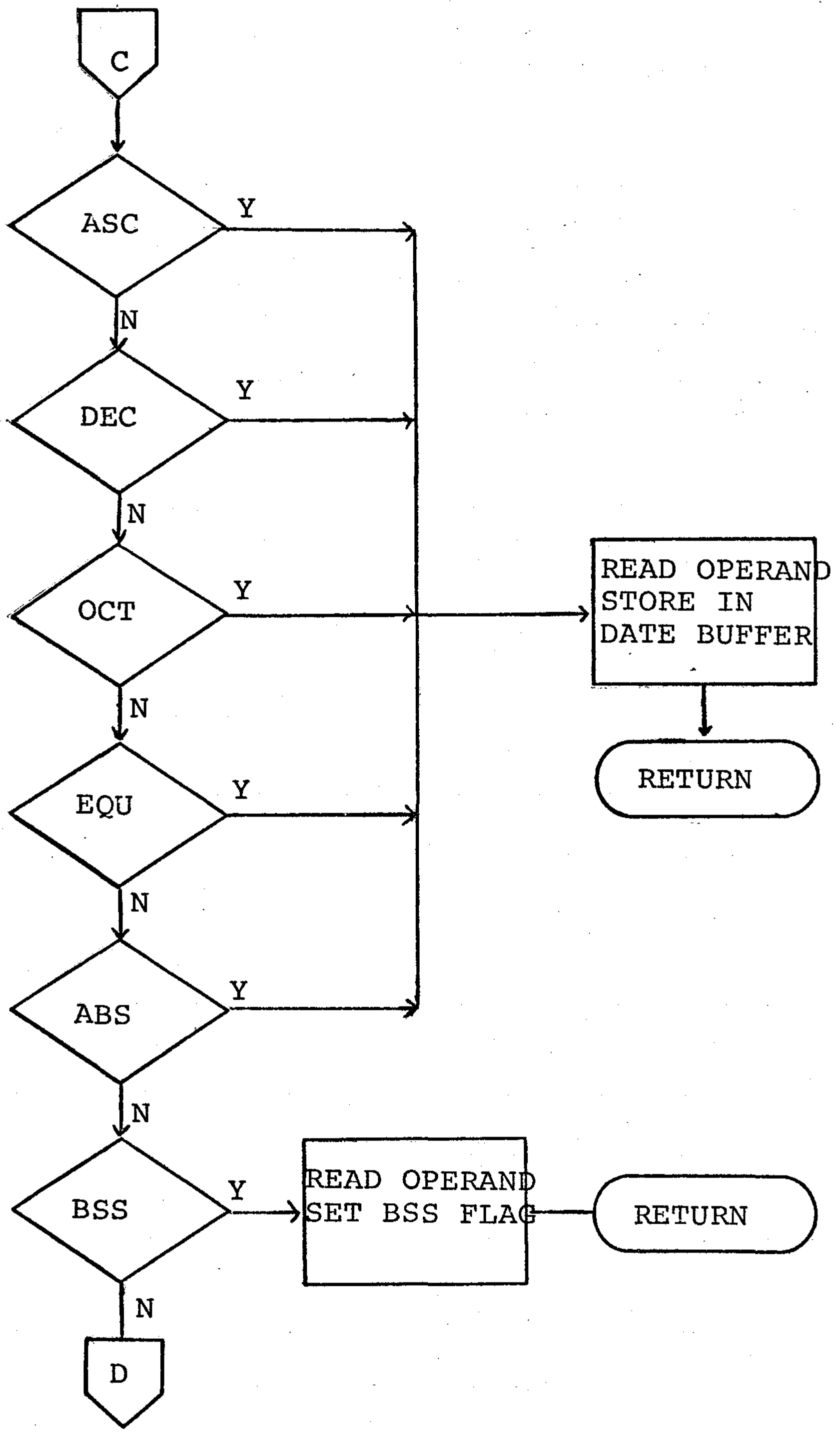
SUBROUTINE LEX FLOW DIAGRAM

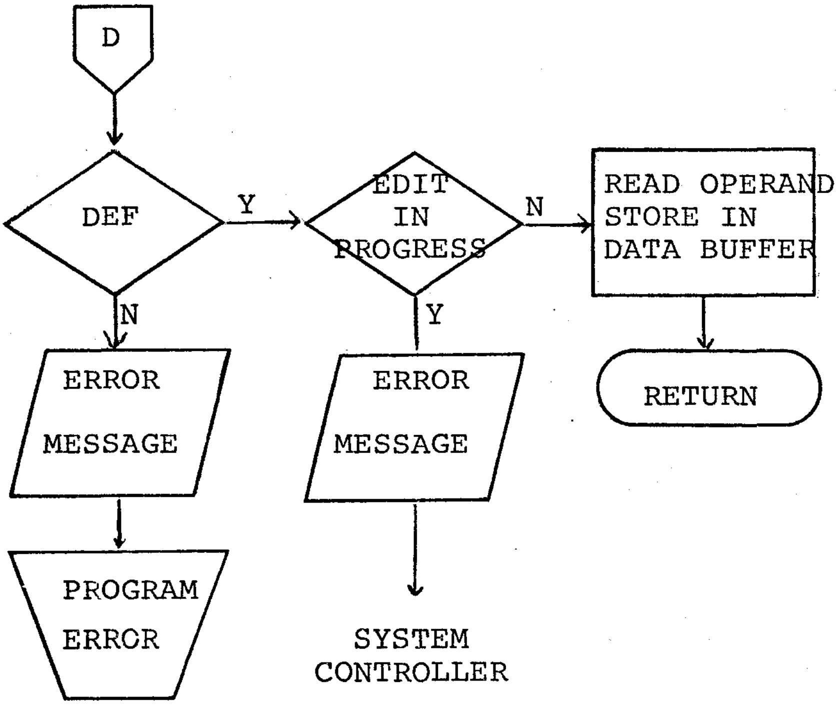
Lexical Errors have not been included. The term "INTR NUM" is used to represent Instruction Number.




A
INTR NUM=2
Y
N
CLEAR FLAG MAY BE PRESENT
Y
INTR. NUM=4
N
INTR NUM=3
Y
CHANNEL NUMBER
N
INTR NUM=4
Y
N
INTR NUM=5
Y
SHIFT COUNT
N
B
MASK OPERAND INTO ASSEMBLY SKELETON
RETURN


B
INTR NUM=6
Y
N
INCREMENT ASSEMBLY LENGTH
INTR. NUM=7
Y
EXAMINE MEMORY REFERENCE OPERAND
RETURN
N
INTR NUM=8
Y
EDIT IN PROGRESS
N
EXECUTE USER PROGRAM
Y
ERROR MESSAGE
N
INITIALIZE DATA BUFFER AND POINTERS
C
SYSTEM CONTROLLER


C
ASC
Y
N
DEC
Y
N
OCT
Y
N
EQU
Y
N
ABS
Y
N
BSS
Y
N
D
READ OPERAND
STORE IN
DATE BUFFER
RETURN
READ OPERAND
SET BSS FLAG
RETURN


D
DEF
Y
N
EDIT IN PROGRESS
N
Y
READ OPERAND STORE IN DATA BUFFER
ERROR MESSAGE
ERROR MESSAGE
RETURN
PROGRAM ERROR
SYSTEM CONTROLLER

CHAPTER VII

ASSEMBLY AND STORAGE

INTRODUCTION

During program definition after control returns from subroutine LEX, subroutine ASMBL is called to prepare pointers and allot space to store the statement in the Source Code Block. During an edit operation EDIPT calls ASMBL.

INSTRUCTION ASSEMBLY

After calling ASMBL the System Controller loads the assembly flag into the A register and will call SETCD unless (A) is a zero, a comment statement. SETCD is also called from various places in the editor for storing edit entries.

Table 7.1 lists seven of the auxiliary assembly subroutines called by SETCD. Of that list subroutine DETLN, DTSET and STRCD do not have error conditions while subroutines STLBL, STRCK, DATFL and STPLC have error messages. These subroutines will be discussed following the discussion on subroutine SETCD.

Subroutine SETCD

Calling Sequence

```
    LDA < Assembly flag >
```

The assembly flag is examined and control branches accordingly; data definitions were treated first for they are less complicated than machine code entries.

DATA DEFINITIONS

For an EQU pseudo op the assembly address has been set; SETCD returns to the calling program.

Otherwise, the assembly address is set the next free area in the data table; data table pointers are set to check for a data table overflow, by calling subroutine DATFL. Once it has been ascertained that the data table will not overflow, the data is assembled before returning to the calling program.

MACHINE INSTRUCTIONS

All machine instructions other than Memory Reference instructions have already been scanned and prepared for assembly and will be immediately stored in the next location in the user program area.

Memory Reference operand evaluation and storage now become the sole function of SETCD.

MEMORY REFERENCE OPERAND EVALUATION

Memory Reference operands not having a symbol or an asterisk term are evaluated first. The operand integer becomes the address by a simple addition to the assembly skeleton. After checking for an indirect reference the instruction is stored in the user program area.

Operands involving the PLC symbol, the asterisk, are assembled next. The SCB address of the program statement with a bit flag for an indirect reference and the operand integer are stored in the PLC table. The relative page address of the entry to the PLC table becomes the forward reference pointer to the instruction.

The remaining operand expressions involve symbols either defined or undefined with or without integer terms. For each symbol there is a call to subroutine LOKUP returning the Symbol Table address as well as a pointer for a defined or undefined symbol or a symbol not found in the Symbol Table.

A defined symbol without an operand integer can be assembled immediately. A data address must be adjusted for the address to reference the data value and not the data address location. The instruction is assembled as discussed in Chapter III by referring to the operand through one level of indirect addressing.

Symbols which were not found in the Symbol Table are entered into the table by a call to STLBL. These symbols can now be regarded as undefined symbols.

The address of the last direct and indirect forward reference will be held in Word 4 and 5 respectively in the Symbol Table entry of undefined symbols. The address of the last reference will be taken from the Symbol Table and combined with the assembly skeleton to be stored with the user program. The instruction will appear as a direct reference to base page but will be recognized as a forward reference using the address-linkage techniques presented in Chapter III.

Each Memory Reference operand having a symbol and an integer is referred to as a compound operand and will be stored in the Special Symbol Table, SST. Word 6 of the Symbol Table entry for any symbol is a link to the first compound operand for that symbol. Word 4 for each compound operand entry is a link to further compound operands with a zero in Word 4 being the terminator.

For each different operand combination there will be a new entry to the SST. Second and subsequent entries of identical compound operands will not require a new entry but will be linked in the same manner as forward references for undefined symbols.

A zero in Word 6 of the symbol involved in the compound operand necessitates a linear search through the SST until the next free area is found. Entries to the SST have been presented in Chapter III. The address of the SST entry is set into Word 6 of the Symbol Table entry. The instruction is stored like any forward reference; in this case, the address term is a pointer to the SST.

If Word 6 has a link to the SST then each SST entry associated with the symbol will be examined for an identical compound operand. Failure to find a match requires a linear search through the SST for the next free area. In this case, the link pointer is set to Word 4 of the last SST entry.

Within SETCD there is a check for SST overflow or the approach to overflow conditions with the appropriate warnings.

ASSEMBLY ROUTINES

Subroutine DETLN

```
Return   (A) = 0  Two-word assembly
         (A) ≠ 0  One-word assembly
```

DETLN determines the assembly length of Memory Reference instructions. On a two-word assembly the first word is stored in the user program area. For a one-word assembly the assembly skeleton is returned in (A).

Subroutine STRCD

```
Calling Sequence
    LDA < Assembled instruction >
```

Each instruction is stored in the next free location in the user program area. The pointer to the user program area is advanced by one and a call to subroutine STRCK will check for program area overflow.

Subroutine DTSET

```
Calling Sequence
    LDB < Address for first data term >
```

DTSET will prepare the address pointers and store the data addresses and values. The BSS instruction uses an indirect reference to non-existant memory to return a zero to be used as the data value; all other pseudo ops, excepting EQU, have the data values stored in a temporary data buffer.

Subroutines STRCK and DATFL

These subroutines simply check the position pointers of the user program area and the data area respectively. Overflow of either table will result in an abnormal abort condition. A warning message is printed if either table approaches an overflow condition.

Subroutine STLBL

Calling Sequence

```
LDA < A > 0, Defined symbol >
    < A = 0, Symbol not in table >

LDB < Symbol Table address of symbol >
```

STLBL will copy the symbol name into the Symbol Table, for a defined symbol the program address and the SCB address will also be included. For undefined symbols the forward reference pointers have already been set.

STLBL also counts the number of symbol entries to the Symbol Table and will print a warning message if the table is nearly full. Overflow is detected in subroutine FIND when a symbol cannot be stored or located in the table.

Subroutine STPLC

Calling Sequence

```
LDA < SCB address of statement >
```

All PLC references are stored in the PLC table. No attempt will be made to define such references until execution. Like the other program tables a warning is presented if the table is nearly full or the user program will be lost if the table is allowed to overflow.

STATEMENT STORAGE

INTRODUCTION

Four different subroutines are responsible for statement storage in the Source Code Block and the definition of statement labels in the user program. They are:

ASMBL	To allocate SCB space to store a program statement,
STSCB	To store the program statement,
LBDEF	To define a statement label,
FWDRF	To define previous references to a statement label.

ASMBL is called from the System Controller and subroutine EDIPT. STSCB and LBDEF are strictly called from the System Controller. Subroutine FWDRF is called from LBDEF and the XECUTE Directive.

These subroutines are called in the order presented and once complete the System Controller loops back to its main entry point or to the multiple insert module if a multiple insert operation is in progress.

The remainder of Chapter VII is a discussion of these four subroutines.

Subroutine ASMBL

The character length of the program statement and the word length of the entry to the SCB will be saved in a temporary variable. The Free Space table is scanned for an area large enough to hold the statement entry.

The scan of the Free Space table will cease when the first area large enough to hold the SCB entry is found. The table entry may be deemed large enough to hold further entries;

it was arbitrarily decided that any isolated area in the SCB larger than twelve words in length would be retained after part of this isolated area had been allocated for the current statement; remaining entries smaller than twelve words would be ignored.

Failure to find an entry in the Free Space table large enough to hold the statement entry requires that the next available area in the SCB be allotted. The SCB address is retained for statement storage after assembly.

A test is made for the SCB table being full or nearly full with the appropriate action taken in each case.

Subroutine STSCB

STSCB stores the six words of information pertaining to each statement along with the source statement in the SCB buffer. Edit instructions involving source statement entries will handle the storage of the address of the previous and next statements as well as statement numbers but require STSCB to complete the SCB entry.

During program definition the address of the next and previous statements are readily set but a back up must be included if the instruction should be stored in an area that was referenced by the Free Space table. A correction must be introduced to link the previous instruction with the current instruction.

The statement number is easily calculated and saved. Word 4 becomes the temporary, set in ASMBL, holding the character-length of the statement and the word-length of the SCB entry.

Word 5 is the assembly address with bit 15 set to one for a data definition. A comment statement is represented by a zero value. Word 6 is the assembly length of the statement.

Beginning with the first word to follow Word 6 the source statement is copied into the Source Code Block.

Subroutine LBDEF

Subroutine LBDEF initiates Symbol Table definition of all statement labels. If there has not been previous reference to the symbol STLBL is called to store the symbol in the table and signify that the symbol has been defined.

A symbol having had a previous reference has forward references associated with it. By checking the direct and indirect forward reference pointers any value less than 700_8 signals a forward reference. By setting a flag for either a direct or indirect reference these forward references will be defined by a call to subroutine FWDRF.

Subroutine FWDRF

Each forward reference is split into the assembly skeleton and the address pointer. Using the assembly skeleton and the assembly address, each instruction will be defined in the same manner as a Memory Reference instruction having an operand symbol. Once the address pointer becomes greater than 700_8, all forward references have been defined; FWDRF may return to the calling program.

TABLE 7.1 AUXILIARY ASSEMBLY SUBROUTINES

SUBROUTINE	FUNCTION
DATFL	Check data table area for overflow
DETLN	Determine assembly length of Memory Reference instruction
DTSET	Assemble data definition
STLBL	Store symbol in Symbol Table
STPLC	Store program location counter reference
STRCD	Store assembled instruction in user program area
STRCK	Check user program area for overflow

CHAPTER VIII

SYSTEM DIRECTIVES

INTRODUCTION

After the colon, signalling a System Directive, is recognized there is a transfer to the program module to interpret and channel the System Directives. The next non-blank character following the colon is required for directive identification. Failure to find a non-blank character or failure to match the character to one of A, D, E, H, L, S, or X will result in an error message and a return to the System Controller.

Using a logical cascade the character is tested with the above characters in the order presented until a match is found.

ABORT

The Abort Directive will initiate an unconditional jump to the initialization program.

DUMP

On recognition of a D, program control branches to the Dump program to print the register names and contents in octal and decimal format. Dump will print the register contents as they appeared after the previous execution by using the special store variables holding such values.

The binary to Ascii section of the I/O package is used solely to convert the register values to Ascii characters for output. One further feature is the binary to Ascii decimal subroutine ASCDC which will convert binary to Ascii using subroutine CNDEC but also include a minus sign preceding the value if negative.

After the register contents have been presented a request is presented to the users to type either R to return or D followed by a data address to be output. The Dump flag is set before program control returns to the System Controller to input the user response. Program control returns to the next location in the Dump program.

Any response beginning with a character other than a D is accepted as a request to terminate all Dump operations. The Dump flag is cleared and control passes back to the System Controller.

Otherwise, the data address is read in by LABCK. The operand must have a data address symbol and be within data table bounds. Failure to satisfy these conditions generates an error message and a re-entry request; Dump error messages are listed in Table 8.1. It should be noted that these are base page error messages used for operand errors elsewhere in the assembler.

Successful entry of a valid data address will result in the corresponding value being printed first as a decimal value then as an octal value. The message requesting a data address dump will be presented after each address dump until the user signals he is finished.

Dump output is presented in Appendix E.

DUMP SUBROUTINES

There are five subroutines called strictly within the Dump Directive.

EODMP	To prepare to display (E) or (O)
RGDP1	To display (A) or (B)
RGDP2	To display (E) or (O)
RGDP3	To print the register name
ASCDC	To convert binary to Ascii decimal with a minus sign preceding a negative value

These subroutines rely on the binary to Ascii conversion facility in the I/O package to prepare the values before calling subroutine TTY.P.

Subroutine ASCDC calls subroutine CNDEC in the I/O package; a negative number is converted to a positive before calling CNDEC and a minus sign character will be stored in the buffer holding the Ascii output.

Three lexical scan subroutines are required to read in and examine the data address.

RDCOM	Read up to the comma before the data address
LABCK	Read the operand and examine the symbol term
DATRG	Check for a data address

EDIT

Even though Edit is the next directive in the logical sequence of System Directives, it will not be discussed since Chapter IX is a detailed discussion of the editor.

HALT

Recognition of the H character will halt the computer with instruction HLT 77B and 102077_8 will appear in the display register on the computer front panel. By pressing the run switch assembler operations may continue.

LIST

The List Directive will list the user program statement by statement. Unlike the System Directives already presented List requires a scan of the command to establish the presence of statement numbers.

The format for the list instruction is:

:L(IST)(,M(,N)).

M and N, if present, specify the first and last statements to be listed. If N is absent then all statements from M on are listed. If neither M nor N appear the whole program is listed.

If a comma is not encountered in the scan, it is assumed the whole program should be listed. The first and last statement numbers are set as parameters to subroutine LIST.

On recognition of a comma it is assumed that statement numbers follow. Subroutine TWINT will read in these statement numbers; the second and fourth return addresses from TWINT involve invalid termination and result in an error message warning.

If N is absent then statement number M is examined to be less than the largest number else an error message for statement numbers out of range. Statement number M and the last statement number will be set as parameters to subroutine LIST.

If both M and N are present, M and N will be the statement number parameters to LIST. N must be greater than the first statement number and M must be less than the last. If N is less than M no error warning is printed.

List error messages are presented in Table 8.2, these are base page error messages which are used by the Sequence Directive as well.

Sample LIST ouput is presented in Appendix E.

Subroutine LIST

Calling Sequence

```
    LDA < Positive value, call from System Directive >
        < -1, call from editor >
```

Beginning with the first statement entry in the SCB and continuing for all entries LIST will save the address of the next instruction.

For each statement LIST scans it retrieves the statement number, Word 3 of the SCB entry. LIST is looking for the first statement number not less than the first statement number parameter. But before any statement will be printed the statement number must also be less than or equal to the second statement number parameter.

Using the binary to Ascii subroutine CNDEC the statement number is converted to Ascii data and printed with leading zeros. A blank character is then printed. Word 4 of the SCB entry holds the length of the source statement; now with the SCB address of the statement the source statement can be listed.

When either statement number bounds or the terminator in the SCB are encountered all listing ceases. On a call from the System Directives module the message *LIST ENDS* is presented. On a call from the editor the message is surpressed.

SEQUENCE

The format for the Sequence Directive is:

:S(EQUENCE),M,N.

Sequence will change statement sequencing such that M becomes the first user program statement number and N is the increment for successive statement numbers. Following completion the whole program will be listed by a call to subroutine LIST.

Subroutine SQNCE, called for sequencing information in the initialization program, is also called by the Sequence Directive.

Bad input data or a range error will cause the Sequence flag to be set before returning to the System Controller for new values of M and N. Once the Sequence Directive has been requested, and an error has occurred valid data must be entered before the Sequence flag will be cleared.

With the new sequencing information there is a cascade through the SCB with a new statement number assigned to each statement.

Subroutine SQNCE

Return	P+1	Error, Re-enter statement
	P+2	Statement numbers accepted and stored

Calling subroutine TWINT, SQNCE will read in two statement numbers, two integer values for M and N. M is restricted to be a positive value less than or equal to 1000 while N must be positive, non-zero and less than or equal to 25.

On a data input or range error the error message is printed before program control is directed to first return address.

If both numbers are in range the values are stored and program control returns through the second return address .

XECUTE

Before beginning the execution of a user program, XECUTE subroutines PLCDF and SSTDF will attempt to define all PLC references and entries to the SST table.

Subroutine CDSCN will scan the user program and replace the first 99 forward references with a jump to the XECUTE warning message regarding undefined forward references.

The main input buffer, the auxiliary input buffer for the temporary definition of undefined ABS or BSS operand symbols and the data store buffer together form a 100-word buffer. CDSCN will clear this buffer area to zero and store the first 99 forward references. Even though the buffer can hold up to 100 forward references only the first 99 are held so that a zero will signal the last forward reference.

It is definitely possible that there may be more than 99 forward references and it is definitely possible to define a program which will skip around the first 99 forward references and yield incorrect results by executing instructions which are forward reference indicators.

But if these conditions should arise the user is not using the assembler as it was intended and/or the user's requirements are beyond the scope of the assembler.

The assembler was intended for inexperienced programmers to develop programs in steps and blocks so that the user can check his program by executing and dumping the results. To accumulate over 99 forward references shows that the user is entering a long complicated program without testing it in steps, in which case the user is probably too experienced to benefit from using the assembler. But if these 99 forward references are such that they are intended to reference an address beyond the bounds of the program, because of an operand integer term, then the user is being foolish and wasting his time for he should know that the assembler is restricted in program size.

Thus, it seemed reasonable to stop at 99 forward references being replaced by a jump instruction to the forward reference warning.

This special jump instruction has also been placed in locations 00002 and 00003 if the user should attempt to execute the contents of the A or B registers.

The user program may now be executed. After successful execution the register contents are specially saved by subroutine SAVR and all forward references are returned to the program.

The user program is scanned for the occurrence of the particular jump to the forward reference warning. Each of these jumps will be replaced by the next forward reference stored in the buffer before execution. Once a zero is encountered all forward references have been restored to the program; control returns to the System Controller.

During the execution of a user program if control should pass to the forward reference warning execution of the user program will cease at that point; register contents will be saved for dump purposes.

Before printing the warning message, the interrupt facility on the output device must be disabled. This is extremely important for once the warning message is printed the forward references are returned to the program. Interrupting the printing of the warning message will return control to the System Controller before the forward references can be restored to the user program.

XECUTE SUBROUTINES

There are five execution subroutines, PLCDF, SSTDF, FNDAD, CNSCN and SAVR which are all strictly called from the XECUTE routine.

Subroutine PLCDF

PLCDF will make a linear search through the PLC table to define as many PLC references as possible. Given the SCB address of the PLC reference and the integer value in the operand, PLCDF calls subroutine FNDAD to calculate the address referenced.

FNDAD returns the address in (A) or sets (A) to -1 if the address referenced by the operand expression is beyond program range. If this address is out of range the PLC reference will not be removed from the table.

A defined address will be retained. Using the SCB address the assembly address is retrieved and retained; also the corresponding address in the address table is required. The forward reference pointer is separated from the instruction skeleton and using this data the instruction is assembled like any other Memory Reference instruction.

Using the forward reference pointer, the forward reference area in the PLC table is reclaimed for further use by clearing the address area to zero.

Subroutine SSTDF

SST will attempt to define compound operands. The Symbol Table is examined for defined symbols with references to the SST.

Taking the SCB address of the symbol and the integer value SSTDF calls FNDAD to calculate the address of the compound operand. Using subroutine FWDRF to advance through the forward references, all forward references will be defined with new addresses.

All compound operand references for each defined symbol will be input to FNDAD. After all SST entries for any one symbol have been tested the links between the Symbol Table entry and all remaining SST entries must be adjusted. After an address is defined the SST entry will be cleared to reclaim table area for further use. The relative position of the entry in the table is found and used to calculate a forward reference pointer to be placed in its appropriate table-entry position.

Subroutine FNDAD

```
Calling Sequence
     LDA < Operand integer value >
     LDB < SCB address of symbol or PLC reference >

Return   (A) = -1, Address not found
             ≠ -1, Address calculated
```

Starting at the SCB address in the B register on input, FNDAD will scan through the SCB using the assembly length of each statement to find the operand address.

FNDAD will have to search in two directions for positive and negative operand integers. In scanning through the SCB, program termination must be checked for each statement. On a search backwards, due to a negative integer value, program termination is flagged by a -1 value in Word 2 of the SCB entry. For each search ahead, program termination is established when Word 1 of the SCB entry points to the next free address in the SCB.

As the program advances ahead the assembly length of each statement is subtracted from the integer until the value is zero or less than zero. The assembly address of this instruction is the address sought with a correction term included if the integer value is less than zero. A search involving a negative value is similar for the operand integer is converted to a positive value.

In either case the address is returned in the A register with a -1 value returned if the terminator was encountered.

Subroutine CDSCN

Subroutine CDSCN clears the 100-word buffer area to zero and stores the first 99 forward references in the buffer.

Since the first 100_8 words of base page are available to the user, Memory Reference instructions making reference to this area must not be regarded as forward references. All forward reference pointers will be removed and replaced by an unconditional jump to the forward reference warning program.

Extended Arithmetic Memory Reference instructions must not be confused with I/O instructions or Extended Arithmetic Register Reference Instructions. In such a case the first word of the two-word assembly is replaced.

Subroutine SAVR

SAVR will save the contents of the A, B, E and O registers in special store variables after execution.

CONCLUSIONS

With the exception of the XECUTE Directive all System Directives discussed are all fairly straightforward and would probably not require further modifications.

The XECUTE program could be expanded to resemble a totally incremental system. Specifically, this would entail the provision for user defined single or multiple step execution options to be implemented using micro-programming.

TABLE 8.1 DUMP ERROR MESSAGES

LABEL	ERROR MESSAGE
DPER1	NO OPERAND FOUND
DPER2	NO LABEL FOUND
DPER3	UNDEFINED LABEL IN OPERAND
DPER4	OPERAND IS UNDEFINED

TABLE 8.2 LIST AND SEQUENCE ERROR MESSAGES

LABEL	ERROR MESSAGE
ERR1	BAD DATA INPUT
ERR2	STATEMENT NUMBERS OUT OF RANGE

CHAPTER IX

THE EDITOR

INTRODUCTION

After recognition of the Edit Directive and before returning to the System Controller in anticipation of an edit instruction, the edit flag and address pointers are set. A message requesting the user to begin edit operations is printed.

The Editor will allow the user to:

Delete any number of statements in the program,
Insert statements between successive statements,
Replace any statement with another statement.

The following instruction causes statements M through N to be deleted:

```
/D(ELETE),M(,N)(,V).
```

If only M is specified only that one statement will be deleted. If M is greater than N the instruction is ignored.

V is the veto flag. When specified, all statements involved in the edit are printed; the user is prompted to respond:

Y(ES) to delete the program statements.

Any other response causes the instruction to be ignored.

The following instruction permits insertions between successive statements:

```
/I(NSERT),M(,N).
```

If only M is specified, then only statement M will be inserted. N is a statement number increment for more than one insertion between successive statements.

On a multiple insert, N is defined and greater than zero, it is not possible to enter both data and machine code type statements. A multiple insertion will be automatically ended if the statement number of the statement to be inserted exceeds the statement number of the instruction which follows the insert.

To replace a single statement the edit instruction is:

/R(EPLACE),M(,V).

A machine instruction statement cannot be replaced by data nor can data be replaced by a machine instruction. However, it is possible either to replace a data definition or a machine instruction by a comment statement or to replace a comment statement by a machine instruction or data definition.

A multiple replace operation is not permitted since sequencing information is not available.

To end the current edit operation, the instruction format is:

/E(ND).

EDIT INSTRUCTION SCAN

All edit operations begin with a slash and the first non-blank character is used to identify the edit instruction. All following characters up to the comma are ignored.

Failure to detect a slash in the first character position will result in an error message; a list of all editor error messages is presented on Table 9.1. If a multiple insert has just been completed a call to subroutine ENDMI must be made to end the multiple insert at the assembled code level. All edit variables are initialized by subroutine EDCLR.

The program performs a logical cascade on the next non-blank character to test for the characters D, E, I, or R and set an instruction number for each except for E which transfers control to finish the edit operation.

Using subroutine TWINT the statement numbers will be read in. The second and fourth return from TWINT signal an illegal terminal character. On such a condition subroutine VETCK will continue the scan for a veto request. If the terminal characters are a comma immediately followed by a V, the veto flag is set; any other combination results in the instruction being ignored and an error message being printed.

The third and fourth return from TWINT signal a multiple edit operation. A multiple delete or insert is valid but a multiple replace results in an error message being printed. There are now five different edit instructions:

1. Single Delete,
2. Multiple Delete,
3. Single Insert,
4. Multiple Insert,
5. Replace.

The number preceding the instruction corresponds to the edit instruction number.

Before the edit operation can begin, several program checks and further preparatory work are required. The value of statement number M must obviously be within the bounds of the user program.

The Source Code Block address of the statements immediately preceding and following the statements involved in the edit must be found by a search through the SCB. Delete instructions require special attention: A delete instruction referring to the last statement in the user program has a special flag set. A multiple delete instruction requires an extended search through the SCB to find the SCB address of the statement following the last deletion.

A multiple insert will allow several statements to be inserted between successive statements. The sum of M and N must not be greater than or equal to the statement number following the insert.

If this is the case, the multiple insert is converted to a single insert instruction by changing the edit instruction number from four to three and by printing a warning message to the user.

If M and N are within range, the first statement number is prepared for the first and subsequent entries by subtracting the value of N from M so that each statement number of the multiple insert can simply be calculated by adding N to the new value of M.

The veto flag, if set, requests the printing of all statements involved in the edit. Statement numbers of the lines to be listed are parameters to subroutine LIST. As well the address of the statement before the edit will also be set as a special variable used by LIST to scan only those statements involved in the edit. Immediately following, the user is asked if these are the statements to be editted. The lexical scan of the response is relaxed and only the first character is examined. Any response other than Y(ES) is regarded as a signal to veto the edit operation.

Subroutine ASMAD retrieves the assembly address of the instruction preceding and following the edit instruction and the assembly address of the instruction involved in a delete or replace operation.

OVERVIEW

The Introduction and the Edit Instruction Scan sections introduce the editor operations but only offer a brief discussion on part of the edit operation.

Before discussing each of the edit subsystems further background information is required to understand edit operations.

SOURCE PROGRAM EDIT

Since two copies of the user program, the source and object program, are maintained by the assembler both must be treated separately by an edit operation. For each of the three operations it was necessary to write separate subroutines to manage next and previous statement pointers as well as the statement number entry in the SCB, Word 1, 2, and 3 of each statement entry in the SCB. Subroutines DSCB, ISCB, and RSCB were written to handle the case of delete, insert and replace operations.

Subroutine DSCB

If the whole program is to be deleted then the system pointer to the first statement is set to the next free area in the SCB while the system address of the previous statement is reinitialized to negative one. If the first program statement is to be deleted the system pointer to the first statement is altered and the SCB address of the previous instruction for the new first statement must be set to the terminator value, -1. On deleting the last statement the system address pointer of the previous statement is reset.

For a deletion preceded and followed by program statements, the successor address pointer of the statement before the delete must point to the first statement after the delete and the previous address pointer of the statement after the delete must be reset to point to the statement preceding the delete.

Subroutine ISCB

By definition an insert is an inclusion between successive statements such that no program check is required for operations involving the first or last statement. The appropriate pointers of the statements following and preceding the insert must be reset. The next and previous pointers as well as the statement number of the insertion are set by ISCB.

On a multiple insert, each inserted statement can be included so that the multiple insert can be terminated after any number of insertions.

Subroutine RSCB

On replace operation not involving the first or last statement RSCB calls subroutine ISCB to link up the edit entry. Replacements involving the first or last statement require special attention.

On replacing the first program statement the first three pointers of the edit entry to the Source Code Block must be set. The system variable pointing to the address of the first statement is set to point to the new first statement.

On replacing the last statement the first three pointers of the SCB entry are set. As well the successor address of the previous statement must be changed and the system variable pointing to the previous statement must now point to the replacement.

DATA EDIT OPERATIONS

Editor operations at the assembly code level manage data and machine instructions separately. To edit a machine code instruction is a far more complicated procedure than a data edit operation. There are three subroutines, DTEDD, DTEDI and SCSYM, directly involved with the manipulation of the user program and data area on a data edit operation.

Subroutine DTEDD

With the length and address of the data to be deleted DTEDD shifts the data area by moving data addresses and data values to fill the gap left by the deleted data. Actually, there is no gap for the deleted data is overwritten; afterwards vacated data areas are cleared. For each data address moved the address area in the data table must be altered to compensate for the shift.

No data shift is necessary when an EQU pseudo op is deleted since the reference will be cleared in the Symbol Table such that the symbol is flagged as undefined for future references.

Subroutine DTEDI

Data insert operations also shift the data table to insert the data in its proper position. Beginning with the last data item and continuing to the first data item after the insert both the data address and value are moved with the address pointer adjusted to compensate for the shift. The program checks for data table overflow before calling subroutine DTSET to store the data.

EQU instructions, having had their assembled code address set during the lexical scan, do not require data shifting.

Shifting data will upset the program address of the shifted data. DTEDI as well as DTEDD call subroutine SCSYM to adjust data addresses after a shift operation.

Subroutine SCSYM

Calling Sequence

```
     LDA < Correction value for address >
     LDB < Test address >
```

The A register holds a correction term to be added to any address greater than or equal to the test address in the B register. Program area addresses will not be altered for the core location of the data table follows the program area, hence the data addresses will always be greater than any address referring to the user program area.

Subroutine SCSYM first scans the Symbol Table for defined symbols and compares the test address with the assembly code address, Word 4 of the symbol entry. The correction term is added to all addresses greater than or equal to the test address, but a special check is set to ignore EQU instructions which are stored at the end of the data table.

The user program address area is next scanned for data addresses. The test address is adjusted so that this address points to the data value rather than the data address. The same test is applied using the address of the data value.

Lastly the data definition instructions in the Source Code Block which follow the insert must have the assembly address adjusted to compensate for the edit. Again, EQU instructions in the SCB must not have the assembly address changed. An EQU instruction in the SCB is recognized as a data definition with an assembly length of zero.

MACHINE CODE EDIT OPERATIONS

INTRODUCTION

Before discussing the edit of machine code instructions in full detail an understanding of the basic concepts involved in a machine code edit is needed.

Editting the assembled machine code entails moving assembled code involved in the edit operation and the use of unconditional jump instructions to link together the edit entries and the existing user program. It was decided to place these edit entries immediately after the existing user program. However, once all edit operations are complete, program definition must be able to continue such that the main user program defined before the edit operation is linked with the program entered after all edit operations are complete.

A two-word buffer is used to separate the first edit entry from the existing program. After all edit operations have been completed these two locations are used to hold two unconditional jump instructions to the next two free areas in the user program area for program definition. These two jump instructions will maintain the link between the program entered before and after the edit.

This technique in using two jump instructions is used in linking the program and most of the edit entries.

It would seem that only one jump instruction is required to link the program units but two jump instructions are required due to skip instructions.

To avoid using two jump instructions would require a bit pattern check on the assembled instruction which immediately precedes the jump instructions. Such a bit pattern test to seek out all the different skip instructions is apt to be a fairly large program. It was believed that the difficulty in implementing such a feature would far outweigh the apparent gain.

With these concepts in mind the machine code edit operations are discussed.

SINGLE AND MULTIPLE DELETE

All instructions being deleted must be examined for a Memory Reference instruction with a forward reference pointer. All other instructions, including Memory Reference instructions with defined operands may be deleted immediately.

Instructions with a reference to the PLC table must first clear the entry to the PLC table before being deleted. But for instructions with forward references pointing to the symbol tables or linking to references which point to the symbol tables, it is necessary to adjust such pointers to exclude the reference.

A machine code delete operation depends upon the length of the deleted code. If more than one word of the assembly code is to be deleted the assembly code involved is cleared to zero. Two jump instructions are placed after the assembly code which precedes the delete to point to the instructions which immediately follow the delete. A delete operation involving only one word of assembly code may not simply be cleared to zero. If a skip instruction should proceed the assembled instruction to be deleted the program logic will be altered by simply clearing the instruction to be deleted.

In the location occupied by the single word to be deleted a jump instruction is set to point to the next free program area for storing the edit entry. Since two jumps must be used to link all edit entries the next assembly instruction must be moved into the next free program area.

Moving an assembled instruction involves some of the problems similar to deleting. Changing the assembly address in the Source Code Block is simple enough but instructions having forward references must have the list, linking the forward references, changed to point to the new position of the forward reference.

In the place of the assembled instruction following the deletion is stored the second jump. Two jumps following the edit entry will link the edit entry back to the next assembled instruction in the program.

If no assembled instructions follow the deletion, the address of the delete becomes the address used to hold jumps linking the user program, entered before the edit operation, to the next free program location, after all edit operations are complete.

SINGLE AND MULTIPLE INSERT

An assembly code insert preceded and followed by assembled instructions is fairly straightforward. The instruction which precedes the insert is moved to the next free program area; the assembly code to be inserted is stored immediately following. The assembly instruction which logically follws the insert is moved to the next program area. Jumps are appropriately placed to link the program and edit entry.

Complications develop if there is no assembly code which either precedes and/or follows the insert.

If no assembled code precedes, then all insertions will be stored in the next free program area. On completion, the instruction occupying the first location in the user program is moved and stored immediately after the insert. In the place formerly occupied by the first instruction is stored a single jump instruction to the insertion. Two jumps following the insert will link the insert to the instructions which logically follow.

If no assembly code follows the insert the program handles the situation similar to the case where no assembly code follows an instruction to be deleted. In this case the two locations following the insert will be used to link the program with the next free program location after all edit operations are complete.

Should assembly code neither precede nor follow the insertion the program pointers must be arranged so that the pointers, normally used to link the program to the next free program area once an edit operation is ended, are not going to branch arount the insertion. Once the insert is complete program pointers will be set to reference the insertion as the main user program and treat any further edit entries appropriately.

REPLACE

A one-word machine code instruction can be replaced by a one-word instruction in the same storage location. The same is true for a two-word assembly being replaced by another two-word assembly instruction.

Replacing a two-word assembly by a one-word assembly requires that the replacement be stored in the next free program area with jumps in the position of the deleted two-word assembly pointing to the edit entry and jumps from the edit entry back to the user program.

A one-word assembly replaced by a two-word assembly is similar to a delete for the replacement is stored in the next available program area. The next instruction in the assembled code is moved to be stored after the replacement entry with the appropriate linkage provided.

A machine code instruction replaced by a comment is treated as a single delete while a comment statement replaced by a machine code instruction is treated as a single insert at the assembly code level.

EDIT SUBROUTINES

With an understanding of the basic edit operations it is now possible to discuss the subroutines concerned with machine code edit operations. These subroutines are presented in the approximate order in which they are called.

Subroutine PREPR

Calling Sequence
    LDB < Address of statement to be deleted >

Return (A) Assembly flag/Assembly address of instruction to be deleted

Subroutine PREPR prepares some pointers before scanning an instruction to be deleted.

Subroutine DELTE

Calling Sequence
    LDB < Address of statement to be deleted >

DELTE initiates the lexical scan of the statement to be deleted and after the scan is complete, DELTE begins analysis of the results to delete the statement.

If a statement label is present, the symbol involved is set as undefined in the Symbol Table. Using the symbol address, forward reference pointers are calculated and stored in their appropriate Symbol Table position.

On a data delete operation subroutine DTEDD is called but a machine code deletion is handled within DELTE.

Machine instructions excluding Memory Reference instructions with forward reference pointers may be deleted immediately. Instructions involving PLC references can be deleted once the PLC reference is cleared from the PLC table. The remaining instructions will be Memory Reference instructions involving references to the symbol tables. The address pointer of the deleted instruction will be set as input to subroutine CASCD to remove the forward reference from the linked list of forward references.

Subroutine CMOVE

Calling Sequence
    LDA < Assembly address of instruction to be moved >
    LDB < SCB address of instruction to be moved >

CMOVE is needed to moved assembled instructions before and after instructions involved in an edit operation.

Before moving the assembled code CMOVE will change the assembly address location in the Source Code Block to account for the move. The assembled code is moved into the next free area of the user program area; the words which previously held the instruction area cleared. After moving each instruction there is call to subroutine STRCK to check for program overflow.

If a moved instruction has a forward reference pointer to the symbol tables, address pointers are set as input to subroutine CASCD to change the forward reference of the instruction pointing to the moved instruction.

Subroutine CASCD

CASCD performs a cascade through the forward references beginning at an address specified by an input variable until the required pointer is found. The forward reference pointer is changed to compensate either for a deleted instruction or for the movement of an instruction with a forward reference.

Failure to find the forward reference signals a program error. A warning message is printed followed by a halt (HLT 33B).

Subroutines JMPAF and JMPBF

JMPAF and JMPBF both call subroutine JMPS to place jump instructions to link the edit entry with the user program and to link the user program with the edit entry respectively.

Subroutine JMPS

```
Calling Sequence
     LDA < Address where jump references >
     LDB < Address to store jump instruction >
```

JMPS forms the jump instructions from the address reference and the instruction skeleton and stores two jump instructions to link the editted code.

Subroutine JMPE1

```
Calling Sequence
     LDA < Address where jump reference >
     LDB < Address to store jump instruction >
```

JMPE1 inserts one jump instruction to link the editted code.

Subroutine STFSP

For every deletion STFSP is called to clear the entry from the Source Code Block and store the length and address of the deletion in the Free Space Table.

Subroutine SNGDL

SNGDL is strictly a delete subroutine to delete a single machine code instruction. Subroutine SVPSN is called to find the next free program area to store the edit entry. Subroutine DELTE will examine the statement to be deleted. Subroutine XDEL will find the location of the instruction after the deletion, to be moved by CMOVE. Subroutines JMPAF and JMPBF will place jumps to link the edit entry.

Subroutine XDEL

Return	(A)	Assembly address of instruction after deletion
	(B)	SCB address of instruction after deletion

XDEL is strictly a delete subroutine to find the first machine instruction after a deletion. Using information from the instruction scan and beginning with the instruction after the delete, the SCB address and assembly address of the next machine code instruction will be returned.

If no assembly code follows the delete then the program pointers are set to link the user program with the next free program area after the edit operations.

Subroutine XINS

XINS is an insert subroutine, for a single insert instruction, to find the SCB and assembly addresses of the machine instruction which logically precedes an insert.

Failing to find any machine code before the insert, XINS calls subroutine YINS to find the instruction in the assembled code which logically follows the insert.

If assembled code neither precedes nor follows the insert, XINS stores the assembled code insert and resets program pointers to treat the entry like the user main program. For a multiple insert, subroutine MULIN will handle this situation.

If machine code instructions follow but do not precede the insert, the insert is stored and the assembly instruction, which logically follows the insert, is moved and placed after the insert. Using JMPE1 one jump is set to point to the insert entry and JMPAF stores two jumps back to the main user program.

Subroutine YINS

Return	P+1	Edit entry linked with program	
	P+2	(A)	Assembly address of instruction after insert
		(B)	SCB address of instruction after insert

By scanning through the SCB, YINS returns the SCB and assembly addresses of the instructions which logically follows the insert.

If the insert follows the last machine code instruction, program activity varies depending on the calling program: On a call from XINS, YINS returns such information to XINS. Usually, the inserted code is linked with the main program. YINS returns to the first return address.

There is one other secondary call to YINS for a machine code replacement of a one-word assembly by a two-word assembly. Normally, YINS will return the SCB and assembly address of the instruction which follows the replacement but if no assembly code follows the replacement, YINS sets up the linkage of the two-word replacement to the user program and advances the program location counter to include the replacement.

Subroutine MULIN

Like XINS and YINS, MULIN scans the Source Code Block for the SCB and assembly addresses which precede and follow a multiple insert operation with the appropriate pointers set.

MULIN initiates storage of the first statement to be inserted and branches to the last entry point to the System Controller to finish statement storage.

Subroutine ENDMI

A multiple insert operation can be terminated any time by the user entering a new edit instruction; termination may also occur on a statement number violation. Using the pointers set in MULIN, ENDMI stores the appropriate jump instructions to link the multiple insert and ENDMI clears all the multiple insert pointers.

Subroutine EDIPT

EDIPT handles the input of source program statements during an edit. The special flag for source statement input is set before jumping to the System Controller.

The System Controller returns control to EDIPT to examine the input. If a slash begins the input it is assumed the slash signals an edit instruction and in such cases a multiple insert is terminated. If the user inadvertantly enters the slash the multiple insert will still be terminated. The program branches to scan the instruction.

For a source statement entry subroutine LEX is called to scan the input. Any lexical errors are treated in the usual manner with control returning to the System Controller. Subsequent statement re-entry returns control to EDIPT for the edit input flag has not been cleared.

Input for replace operations is examined for an assembly flag match between the deleted and the replacement statement; comment statements do not require an assembly flag match.

The statement number for a multiple insert is calculated. On a statement number error, the calculated statement is greater than that of the next statement; the multiple insert is terminated by a call to ENDMI. A warning message is printed and the edit input flag is cleared before returning to the System Controller.

If the statement number is in range, the edit input flag is cleared and subroutine ASMBL is called to allocate space to store the statement in the SCB.

EDIT SUBSYSTEMS

INTRODUCTION

After gathering all information that is requested from the instruction scan, the editor uses the instruction number in a logical cascade to find the appropriate edit subsystem.

SINGLE DELETE

An undefined statement number in the edit instruction results with the instruction being ignored but a warning message is printed.

Otherwise subroutine DSCB handles the delete of the source program. PREPR prepares some pointers in anticipation of an assembled code edit and returns the assembly flag/assembly address word before scanning the instruction to be deleted.

A comment statement being deleted does not require a lexical scan of the statement; the Source Code Block length and address of the delete are retained in the Free Space Table by calling subroutine STFSP.

For both data and machine instructions subroutine DELTE is called; DELTE calls DTEDD to delete a data definition or DELTE returns information on a machine code instruction and if necessary adjusts forward reference pointers to exclude the deleted instruction.

Using the assembly length of the deleted machine code the deleted area is replaced by jump instructions for a two-word assembly or subroutine SVPSN is called to delete a single-word assembly.

Before returning to the System Controller a record of the deletion is stored in the Free Space Table by subroutine STFSP.

MULTIPLE DELETE

A multiple delete is somewhat more complicated than a single delete. A counter is first set to hold the assembly length of all deleted machine code instructions. DSCB is called to perform the edit on the source program.

For each statement being deleted not only is the SCB address of the statement retained but also the link to the next statement else it will be lost calling subroutine STFSP.

Like the single delete there is a call to PREPR for each statement to be deleted. For both data defintions and machine instructions code subroutines DELTE and STFSP are called; for a comment statement only subroutine STFSP need be called. The deletion of a comment or data definition is complete; the next statement may now be deleted.

On a machine code delete the address of the first machine code deleted must be retained. The address of the last machine code instruction deleted is advanced for each delete with the deleted area cleared. The second word of a two-word assembly must also be cleared; the length of the deleted code is advanced by the assembly length for each deletion.

After scanning all statements to be deleted, the length of the deleted assembly code is examined. If no assembly statements have been deleted, the multiple delete is finished. If only one word in the assembled code is to be deleted then the situation resembles a single delete at the machine code level; subroutine SNGDL is called to perform a single machine code delete. If more than one word in the assembled code is to be deleted, then a pair of jumps stored in the first two words beginning the delete point to the first two assembled instructions after the delete.

SINGLE INSERT

If the statement number specified by the insert instruction is a defined statement, the error message labelled EDR7 is printed with the re-entry request.

Before beginning a single insert, subroutine EDIPT will input the statement to be inserted and examine the assembly flag to determine the nature of the insert.

Regardless of the assembly the SCB pointers must be set by a call to ISCB. For a comment statement program control may branch to the last entry point of the System Controller to complete statement storage in the SCB. For a data insert subroutine DTEDI is called to store the data in its appropriate data table position before returning to the System Controller.

On a machine code insert the assembly code before and after the insert is sought; the insert is stored depending upon its logical position in the assembled program.

MULTIPLE INSERT

Like the single insert there is a call to error message EDR7 for a defined statement number on an insert operation.

Otherwise, the multiple insert flag is set. All source statements in the insert are input by a call to EDIPT. After a statement has been fully stored in the SCB in the System Controller, program control returns to the multiple insert program. This call to EDIPT, in the multiple insert program is the return point from the System Controller for further input.

Since both data and machine code cannot both be entered interchangeably the assembly flag of each statement to be inserted is compared with the flag denoting either a data or machine code insert. On an assembly flag clash the edit flag signalling source statement entry is set before printing an error message so that control will return to EDIPT following statement re-entry.

A comment statement requires a call to ISCB. A data definition requires calls to DTEDI and ISCB. On the first machine code instruction to be inserted a call to MULIN prepares address pointers and stores the first machine code insert. A flag is set to signal the second and subsequent machine code entries which are stored in the next user program area similar to any other assembled instruction.

The multiple insert operation is terminated by a call to subroutine ENDMI from the instruction scan section of the editor on recognition of a new edit instruction or from EDIPT on a statement number violation.

REPLACE

Using the delete subroutines PREPR, DELTE and STFSP the instruction to be replaced is deleted. EDIPT inputs the replacement statement and checks for an assembly flag clash between the deleted and replacement statements. RSCB sets the SCB pointers before storing the instruction.

For machine code instructions replaced by machine code instructions of the same assembly length the replacement is stored in the deleted area. To store the replacement it is necessary to save the user program location pointer in a temporary variable. The program location of the replacement is set as the program area pointer used by SETCD, to store the replacement instruction. After the replacement has been stored the user program location counter is restored.

Any other machine code replace operations have already been discussed in the section on machine code replace operations.

Data deletions are handled in DELTE. Data replacements are easily included by calling DTEDI.

After all replacement operations are complete control returns to the last entry point of the System Controller to complete SCB entries for the replacement.

END

The End request first adjusts the SCB successor address pointer of the last program statement to point to the next free location in the SCB. The successor address pointer of the last program statement may point to edit entries in the SCB which have been stored immediately after the last program statement. Changing the successor address pointer will by bypass any possible edit entries in the SCB and maintain the proper source program linkage.

Two jump instructions are set to link the main user program with the next free program area in the user program area. These jumps are to reside in the two words set aside after recognition of the Edit Directive.

Lastly, the main edit flag is cleared before returning to the System Controller.

CONCLUSIONS

The Editor is restricted to the three main edit operations which are adequate for a beginner's use. Multiple skip instructions or subroutine calls which pick up arguments from subsequent locations would not be handled correctly. Fortunately, multiple skip instructions are not available; the people for whom the assembler is intended are not expected to employ such argument linkage techniques, but the possibility exists. The only alternative seems to be complete reassembly which defeats the purpose of the assembler.

However, the Editor will handle patches made over patches; although the object program may come to look rather peculiar, the source program will always be readable. Before changing the editor serious consideration should be given to all editor features in the light if possible changes to any other assembler features.

TABLE 9.1 EDITOR ERROR MESSAGES

LABEL	ERROR MESSAGE
EDR1	ILLEGAL DATA PRECEDES EDIT INSTRUCTION
EDR2	UNDEFINED EDIT INSTRUCTION
EDR3	BAD DATA FOLLOWS EDIT INSTRUCTION
EDR4	VETO NOT PERMITTED ON AN INSERT
EDR5 (ERR2)	STATEMENT NUMBER OUT OF RANGE
EDR6	ILLEGAL SOURCE TYPE ENTRY DURING EDIT
EDR7	STATEMENT NUMBER ALREADY DEFINED
EDR8	STATEMENT NUMBERS MUST ACCOMPANY EDIT INSTRUCTION
EDR9	STATEMENT NUMBER IS NOT DEFINED

APPENDIX A

ASSEMBLER MACHINE INSTRUCTIONS AND PSEUDO OPS

Assembler machine code instructions are:

ADA	Add to (A)
ADB	Add to (B)
ALF	Rotate (A) left 4
ALR	Shift (A) left 1, clear sign
ALS	Shift (A) left 1
AND	And to (A)
ARS	Shift (A) right 1, carry sign
ASL	Arithmetic long shift left
ASR	Arithmetic long shift right
BLF	Rotate (B) left 4
BLR	Shift (B) left 1, clear sign
BLS	Shift (B) left 1
BRS	Shift (B) right 1, carry sign
CCA	Clear and complement (A)
CCB	Clear and complement (B)
CCE	Clear and complement (E) set (E) = 1
CLA	Clear (A)
CLB	Clear (B)
CLC	Clear I/O control bit
CLE	Clear (E)
CLF	Clear I/O flag
CLO	Clear overflow bit
CMA	Complement (A)
CMB	Complement (B)
CME	Complement (E)
CPA	Compare to (A), skip is unequal
CPB	Compare to (B), skip if unequal
DIV	Divide
DLD	Double load
DST	Double store
ELA	Rotate (E) and (A) left 1
ELB	Rotate (E) and (B) left 1
ERA	Rotate (E) and (A) right 1
ERB	Rotate (E) and (B) right 1
HLT	Halt
INA	Increment (A) by 1
INB	Increment (B) by 1
IOR	Inclusive or into (A)
ISZ	Increment, then skip if zero
JMP	Jump
JSB	Jump to subroutine
LDA	Load into (A)
LDB	Load into (B)
LIA	Load into (A) from I/O channel
LIB	Load into (B) from I/O channel
LSR	Logical long shift right
MIA	Merge (or) into (A) from I/O channel
MIB	Merge (or) into (B) from I/O channel
MPY	Multiply
NOP	No operation
LSL	Logical long shift left

OTA	Output from (A) to I/O channel
OTB	Ouptut from (B) to I/O channel
RAL	Rotate (A) left 1
RAR	Rotate (A) right 1
RBL	Rotate (B) left 1
RBR	Rotate (B) right 1
RRL	Rotate (A) and (B) left
RRR	Rotate (A) and (B) right
RSS	Reverse skip sense
SEZ	Skip if (E) = 0
SFC	Skip if I/O flag = 0 (clear)
SFS	Skip if I/O flag = 1 (set)
SLA	Skip if LSB of (A) is zero
SLB	Skip is LSB of (B) is zero
SOC	Skip if overflow bit = 0 (clear)
SOS	Skip if overflow bit = 1 (set)
SSA	Skip if sign bit of (A) = 0
SSB	Skip if sign bit of (A) = 0
STA	Store (A)
STB	Store (B)
STC	Set I/O control bit
STF	Set I/O control flag
STO	Set overflow bit
SWP	Switch (A) and (B)
SZA	Skip if (A) = 0
SZB	Skip if (B) = 0
XOR	Exclusive or to (A)

Assembler Pseudo Operation instructions are limited to:

ABS	Define absolute value
ASC	Generate Ascii characters
BSS	Reserve Block of storage
DEC	Define decimal constants
DEF	Define address
END	Terminate program (begin execution)
EQU	Equate symbol
OCT	Define octal constants

ASSEMBLER INSTRUCTIONS

LEXICAL GROUP NUMBER CLASSIFICATION

GROUP NUMBER	INSTRUCTION TYPE	OPERAND REQUIRED
1	ALTER SKIP REGISTER REFERENCE	NO OPERAND REQUIRED
2	INPUT/OUTPUT	CLEAR FLAG may BE PRESENT
3	INPUT/OUTPUT	CHANNEL NUMBER EXPECTED
4	INPUT/OUTPUT	CHANNEL NUMBER EXPECTED CLEAR FLAG MAY BE PRESENT
5	EXTENDED ARITHMETIC REGISTER REFERENCES	NUMBER OF SHIFTS
6	MEMORY REFERENCE	SYMBOL (ASTERISK)
7	EXTENDED ARITHMETIC MEMORY REFERENCE	INTEGER INDIRECT FLAG
	PSEUDO OPS	
8	END	NO OPERAND REQUIRED
9	ASC	LENGTH AND LIST OF ASCII DATA
10	DEC	REALS OR DECIMAL INTEGERS
11	OCT	OCTAL INTEGERS
12	EQU	ADDRESS
13	ABS	ADDRESS VALUE
14	BSS	VALUE
15	DEF	ADDRESS DEFINITION

MACHINE INSTRUCTIONS

MNEMONIC CLASSIFICATION BY GROUP NUMBER

GROUP 1	ALF	ALR	ALS	ARS	BLF	BLR	BLS
	BRS	CCA	CCB	CCE	CLA	CLB	CLE
	CLO	CMA	CMB	CME	ELA	ELB	ERA
	ERB	INA	INB	NOP	RAL	RAR	RBL
	RBR	RSS	SEZ	SLA	SLB	SSA	SSB
	STO	SWP	SZA	SZB			
GROUP 2	SOC	SOS					
GROUP 3	CLF	SFS	SFS	STC			
GROUP 4	CLC	HLT	LIA	LIB	MIA	MIB	OTA
	OTB						
GROUP 5	ASL	ASR	LSL	LSR	RRL	RRR	
GROUP 6	ADA	ADB	AND	CPA	CPB	IOR	ISZ
	JMP	JSB	LDA	LDB	STA	STB	XOR
GROUP 7	DIV	DLD	DST	MPY			

MACHINE INSTRUCTION OPERAND TYPES

GROUP 2 SOC (C)

The clear flag if present will clear the overflow bit after execution of the instruction.

GROUP 3 CLF (+)integer
SFS (+)symbol

The integer must be a positive value less than 64 signifying the channel number to make the instruction apply to one of up to 64 I/O devices or functions. The operand may also be a symbol which has been equated to an I/O channel address by an EQU pseudo op. An optional plus sign may precede the channel number.

GROUP 4 CLC (+)integer(,C)
HLT (+)symbol(,C)

Group 4 instruction operands are similar to Group 3 except that they may be followed by, C to clear the device flag after execution of the instruction.

GROUP 5 ASL (+)integer

The integer operand must be a positive value from one to sixteen to specify the number of shifts on the combined contents of (B) and (A).

GROUP 6 ADA (+)(symbol)(±integer)(,I)

GROUP 7 DIV (+)(symbol)(±integer)(,I)

The memory reference operand has been restricted to a symbol, integer and indirect flag combination. The symbol may be preceded by a blank or a + sign; any other character will generate an error message.
An integer operand without a symbol must be a positive integer less than 64 for reference to the base page; any other value will not be accepted. A symbol-integer combination must be within bounds of the user's program area.
The indirect flag allows the value of the operand to access another word in the user program area which is taken as the new memory reference for the instruction.

PSEUDO OPERATIONS

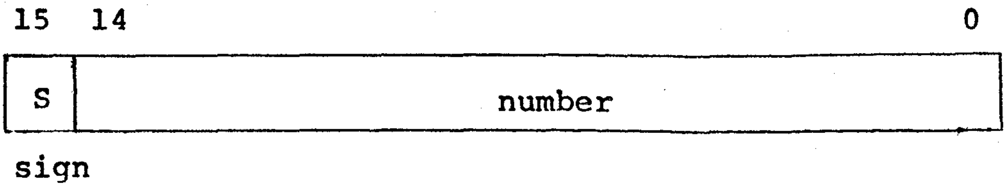
The ASC, DEC and OCT data definitions have been implemented in accordance with Hewlett Packard definition.

ASC n, < 2n characters >

ASC generates a string of 2n alphanumeric characters in Ascii code into n consecutive words. One character is right justified in each 8 bits; the most significant bit is zero. n must be a positive decimal integer in the range 1 to 28*. If any number less than 2n characters are present before the end of the statement, the remaining characters are assumed to be blanks and stored as such. Anything after 2n characters in the operand field is treated as a comment.
To enter the code for Ascii symbols which perform some action like carriage return or line feed, the OCT pseudo op must be used.
A label preceding represents the address of the first two characters.

DEC $d_1[,d_2, \ldots, d_n]$

DEC records a string of decimal constants into consecutive words. The constants may be integer or real (floating point) and positive or negative. If no sign is specified, positive is assumed. The decimal number is converted into its binary equivalent by the assembler. The label, if present, serves as the address of the first word occupied by the constant.
A decimal integer must be in the range 0 to $2^{15}-1$ (32767) which may assume positive, negative or zero values. It is converted into one binary word and appears as follows.



A floating point number has two components a fraction and an exponent which specifies the power of ten by which the fraction is multiplied. The fraction is a signed or unsigned number which may be written with or without a decimal point.

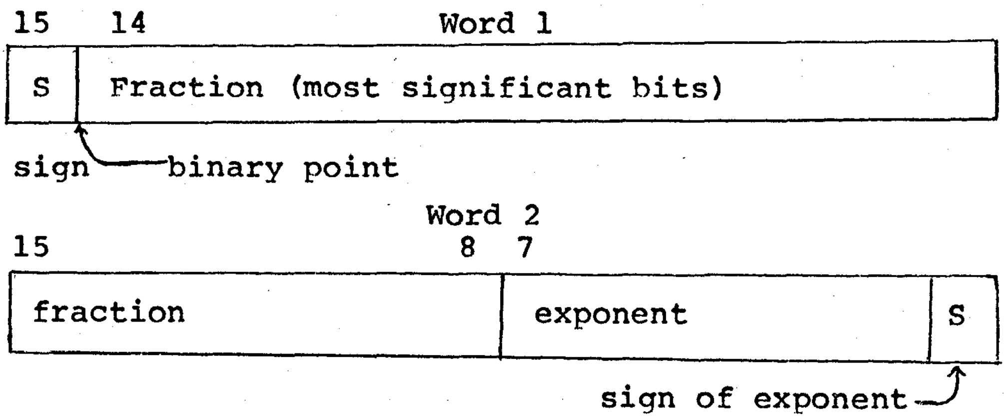
---

* By Hewlett Packard definition n may be any expression resulting in a decimal value in the range 1 to 28 but the implementation has been restricted to strictly decimal integers.

The exponent is indicated by the letter E and precedes a signed or unsigned decimal integer. A floating point number may have any of the following formats:

±n.n, ±n., ±.n, ±n.E±e, ±n.nE±e, ±n.E±e, ±.nE±e

The number is converted to binary, normalized and stored in two computer words. If either of the fraction or the exponent is negative that part is stored in two's complement form.



The floating point number is made up of a seven bit exponent with a sign bit and a 23 bit fraction with a sign bit. The number must be in the approximate range of 10^{-38} to zero.

OCT $o_1[,o_2, \ldots, o_n]$

OCT stores one or more octal constants in consecutive words of the object program. Each constant consists of one to six octal digits (0 to 17777). If no sign is given the sign is assumed to be positive. If the sign is negative, the two's complement binary equivalent is stored. The constants are separated by commas with the last constant terminated by a space. If less than six digits are specified for a constant the data is right justified in the word. The letter B must not be used after the constant.

The remainder of the pseudo operations, ABS, BSS, DEF, END, and EQU have been altered from the Hewlett Packard definition.

ABS ±(symbol)(±integer)

ABS will define a data address or a base page address within the user program bounds. Undefined symbols in the operand will be accepted but a temporary value must be entered to define the symbol

BSS (+)(symbol)(±integer)

BSS advances the program location counter according to the value of the operand and initializes the data area to zero. The operand value has been restricted to the range of 1 to 128. As undefined symbol in the operand will be accepted but a value must be entered to define temporarily the symbol.

DEF symbol(,I)

DEF generates one word of core as a 15 bit data area address which may be used as the object of an indirect address found elsewhere into the source program. The address may be referenced indirectly through the label preceding. The operand field must be a data symbol which may be followed by an indirect flag.

END

END does not require an operand for it is a command to begin execution of the user's program.

EQU (+)(symbol)(±integer)

EQU assigns to a symbol a value other than one normally assigned by the program location counter. A label must precede the EQU pseudo op to be assigned the value represented by the operand field.
The operand must be an address in the user program data area or in the base page area available to the user. A symbol in the operand must have been previously defined.

APPENDIX B

THE INTRODUCTORY TEXT

THE INTRODUCTORY TEXT

The data has been stored as binary data packed two characters per word beginning on the first sector of the first track of a removable cartridge disc by the DOS -M System facility to write onto user files, EXEC call, Request code 15. Every page of information starts on a disc sector boundary but no page of information will be allowed to cross a track boundary. This restriction is imposed by the disc controller which requires additional head positioning and read commands to read across a track boundary. The special positioning of each page has been incorporated into the disc address table, in the initialization program, according to the format:

Bits 0 - 7 Sector number,
8 - 15 Track number.

This arrangement of the introductory text removes the necessity for using a disc file directory or search program.

The following is a list of the page names used in the program to store the text on disc and the names used in the address table in the initialization program.

PAGE 1	Introductory information
PAGE 2	Introductory information
PAGE 3	User option to begin program entry or continue presentation of text
PAGE 4	List of the System Directives excluding the Halt Directive
DUMP	Explanation of Dump Directive
LIST	Explanation of List Directive
SEQUENCE	Explanation of Sequence Directive

XECUTE	Explanation of Xecute instruction
EDIT 1	Explanation of Editor and edit instructions
EDIT 2	Explanation of Editor and edit instructions
LAST	Warning to user about program size and prompt to begin

The remainder of Appendix B is a listing of the program used to store the text on disc followed by a listing of the eleven pages of the text.

PROGRAM TO STORE INTRODUCTORY TEXT ON DISC

```
ASMB,R,L
      NAM JF2
      EXT EXEC
*
*
* STORE READ ONLY INTRODUCTARY TEXT ON DISC CARTRIDGE
*
*
START NOP
*
* PAGE 1
*
      LDB =B1111
      STB BUFL         LENGTH OF DATA
      LDA PAGE1        PROGRAM ADDRESS OF DATA
      CLB              RELATIVE SECTOR NUMBER
      JSB DWRIT
*
* PAGE 2
*
      LDB =B750
      STB BUFL
      LDA PAGE2
      LDB =D5
      JSB DWRIT
```

```
*
* PAGE 3
*
      LDB =B212
      STB BUFL
      LDA PAGE3
      LDB =D9
      JSB DWRIT
*
* PAGE 4
*
      LDB =B457
      STB BUFL
      LDA PAGE4
      LDB =D11
      JSB DWRIT
*
* PAGE 5 DUMP
*
      LDB =B745
     STB BUFL
      LDA PAGE5
      LDB =D14
      JSB DWRIT
```

```
*
* PAGE 6 LIST
*
      LDB =B312
      STB BUFL
      LDA PAGE6
      LDB =D16
      JSB DWRIT
*
* PAGE 7 SEQUENCE
*
      LDB =B416
      STB BUFL
      LDA PAGE7
      LDB =D18
      JSB DWRIT
*
* PAGE 8 XECUTE
*
      LDB =B345
      STB BUFL
      LDA PAGE8
      LDB =D21
      JSB DWRIT
```

```
*
* PAGE 9 EDIT 1
*
      LDB =B542
      STB BUFL
      LDA PAGE9
      LDB =D24
      JSB DWRIT
*
* PAGE 10 EIDT 2
*
      LDB =B447
      STB BUFL
      LDA PGE10
      LDB =D27
      JSB DWRIT
*
* PAGE 11 LAST
*
      LDB =B553
      STB BUFL
      LDA PGE11
      LDB =D30
      JSB DWRIT
*
* STOP PROGRAM
*
      JSB EXEC
      DEF *+2
      DEF *+2
      NOP
      DEC 6
```

```
*
*
* COPY BINARY DATA ONTO DISC
*
* ENTER (A) PROGRAM ADDRESS OF DATA
*       (B) RELATIVE SECTOR NUMBER
*
*
DWRIT NOP
      STA ADDR
      STB SECTR
      JSB EXEC
      DEF *+7
      DEF RCODE    REQUEST CODE 15 FOR DISC WRITE
      DEF CONWD
      DEF ADDR,I   PROGRAM ADDRESS
      DEF BUFL     LENGTH OF BUFFER
      DEF FNAME    FILE NAME
      DEF SECTR    RELATIVE SECTOR
      JMP DWRIT,I
*
*
ADDR  BSS 1
SECTR BSS 1
RCODE DEC 15
CONWD OCT 102
BUFL  BSS 1
FNAME ASC 3,JFCAD  BINARY FILE ON USER DISC AREA
*
*
      SUP
```

```
PAGE1 DEF *+1
*
* PAGE 1 INTRODUCTION
*
      ASC 16,     YOU ARE COMMUNICATING WITH
      ASC 12,A HEWLETT PACKARD 2100A
      OCT 106612
      ASC 16,COMPUTER THAT HAS BEEN PREPARED
      ASC 12,TO READ IN AND ASSEMBLE
      OCT 106612
      ASC 17,COMPUTER PROGRAMS WHICH YOU ENTER.
      OCT 106612,106612
      ASC 18,     A COMPUTER PROGRAM IS A SERIES
      ASC 11,OF COMMANDS TO DIRECT
      OCT 106612
      ASC 15,THE COMPUTER IN A STEP BY STEP
      ASC 14, PROBLEM SOLVING PROCEDURE.
      OCT 106612,106612
      ASC 28,     SUCH COMMANDS RECOGNIZED BY THE COMPUTER ARE IN THE
      OCT 106612
      ASC 26,FORM OF MACHINE LANGUAGE, BUT PROGRAMMING IN MACHINE
      OCT 106612
      ASC 15,LANGUAGE IS A TEDIOUS PROCESS
      ASC 15,AND ONE OF THE MOST IMPORTANT
      OCT 106612
      ASC 18,STEPS IN TRYING TO MAKE PROGRAMMING
      ASC 11,EASIER IS TO INTRODUCE
      OCT 106612
```

```
ASC 15,INSTRUCTION CODES IN PLACE OF
ASC 15,MACHINE CODES AND ADDRESSES.
OCT 106612
ASC 14,THE USE OF INSTRUCTION CODES
ASC 16, LEADS TO A PROGRAMMING LANGUAGE
OCT 106612
ASC 28,ALMOST EQUIVALENT TO MACHINE CODE BUT EASIER TO READ.  A
OCT 106612
ASC 18,PROGRAM TO TRANSLATE SUCH A LANGUAGE
ASC 12, INTO THE CORRESPONDING
OCT 106612
ASC 20,MACHINE LANGUAGE IS CALLED AN ASSEMBLER.
OCT 106612,106612
ASC 27,    THE TASK OF AN ASSEMBLER IS TO TRANSLATE ASSEMBLY
OCT 106612
ASC 17,INSTRUCTIONS INTO MACHINE LANGUAGE
ASC 14, INSTRUCTIONS CORRESPONDING
OCT 106612
ASC 26,WITH WHAT APPEARS IN THE ASSEMBLY LANGUAGE PROGRAM.
OCT 106612,106612
ASC 18,    IT IS NOW POSSIBLE TO TRANSFER
ASC 13,CONTROL TO THE CRT SCREEN.
OCT 106612,106612
ASC 21,    TYPE S TO PRINT OUTPUT ON CRT SCREEN
OCT 106612,106612
ASC 17,    OTHERWISE TYPE C TO CONTINUE
OCT 106612,106612
ASC 23,    (TYPE RETURN KEY TO ENTER ALL RESPONSES)

NOP
```

```
PAGE2 DEF *+1
*
* PAGE 2 INTRODUCTION
*
      OCT 116637
      ASC 17,      AN ASSEMBLER NORMALLY BEGINS
      ASC 13,ASSEMBLY ONCE THE PROGRAM
      OCT 106612
      ASC 18,HAS BEEN FULLY DEFINED.  REFERENCES
      ASC 13,TO UNDEFFINED INSTRUCTIONS
      OCT 106612
      ASC 18,OR DATA WILL TERMINATE THE ASSEMBLY
      ASC 11,OR HALT FURTHER SYSTEM
      OCT 106612
      ASC 12,ACTIVITY AFTER ASSEMBLY.
      OCT 106612,106612
      ASC 19,      THIS ASSEMBLER IS AN INCREMENTAL
      ASC 11,ASSEMBLER FOR ASSEMBLY
      OCT 106612
      ASC 28,OCCURS IMMEDIATELY AFTER STATEMENT ENTRY.  THE ASSEMBLER
      OCT 106612
      ASC 16,DOES NOT WAIT UNTIL THE PROGRAM
      ASC 14,IS FULLY DEFINED.  UNDEFINED
      OCT 106612
      ASC 24,REFERENCES ARE RETAINED UNTIL DEFINITTION OCCURS.
```

```
OCT 106612,106612
ASC 16,     EACH STATEMENT IS SEQUENCED
ASC 13, AND ASSIGNED A STATEMENT
OCT 106612
ASC 15,NUMBER.  BY DEFAULT THE FIRST
ASC 16,STATEMENT NUMBER IS 10 WITH EACH
OCT 106612
ASC 23,SUCCESSIVE STATEMENT NUMBER INCREMENTED BY 10.
OCT 106612,106612
ASC 18,     TO SPECIFY ALTERNATE SEQUENCING
ASC 12, TYPE S FOLLOWED BY THE
OCT 106612
ASC 28,FIRST STATEMENT NUMBER THEN A VALUE FOR THE INCREMENT.
OCT 106612
ASC 26,USE COMMAS (,) TO SEPARATE THE S AND THE TWO VALUES.
OCT 106612,106612
ASC 12,     FOR EXAMPLE:S,12,6
OCT 106612
ASC 17,RESULTS WITH THE FIRST INSTRUCTION
ASC 13, ASSIGNED THE STATEMENT
OCT 106612
ASC 28,NUMBER 12 WITH THE FOLLOWING STATEMENTS ADVANCED BY 6.
OCT 106612,106612
ASC 13,     OR TYPE C TO CONTINUE
NOP
```

```
PAGE3 DEF *+1
*
* PAGE 3 INTROCDUCTION TO USERS
*
      OCT 115637
      OCT 106612
      OCT 105212,105212
      OCT 105212,105212
      ASC 15,      IF YOUR ARE FAMILIAR WITH
      ASC 15, THE FEATURES OF THE ASSEMBLER
      OCT 106612
      ASC 27,YOU MAY BEGIN ENTRY OF AN ASSEMBLY LANGUAGE PROGRAM.
      OCT 106612,106612
      ASC 12,      TYPE C TO CONTINUE
      OCT 106612,106612
      ASC 15,      WAIT FOR SYSTEM RESPONSE
      OCT 106612,106612
      ASC 12,      BEGIN PROGRAM ENTRY
      OCT 106612,106612
      ASC 27,ELSE TYPE L TO LEARN ABOUT THE VARIOUS SYSTEM FEATURES
      NOP
```

```
PAGE4 DEF *+1
*
*
* PAGE 4
*
      OCT 116637,105212
      ASC 17,     THERE ARE 6 SYSTEM DIRECTIVES
      ASC 13, WHICH MAY BE ENTERED ANY
      OCT 106612
      ASC 28,TIME WHILE DEFINING YOUR PROGRAM, EXCEPT DURING AN EDIT.
      OCT 106612
      ASC 15,THEY ALLOW YOU GREATER CONTROL
      ASC 14, OVER THE ASSEMBLER AND THE
      OCT 106612
      ASC 12,DESIGN OF YOUR PROGRAM.
      OCT 106612,106612
      ASC 24,THESE DIRECTIVES ARE ALL PRECEDED BY A COLON (:).
      OCT 106612,106612
      ASC 27,     :ABORT     DISCONTINUE PROGRAM ENTRY, BEGIN AGAIN
      OCT 106612,106612
      ASC 19,     :DUMP      DUMP REGISTER CONTENTS
      OCT 106612,106612
      ASC 21,     :EDIT      EDIT THE EXISTING PROGRAM
      OCT 106612,106612
      ASC 24,     :LIST      LIST ALL OR PART OF YOUR PROGRAM
      OCT 106612,106612
      ASC 19,     :SEQUENCE  CHANGE THE SEQUENCING,
      ASC 11, THEN LIST THE PROGRAM
      OCT 106612,106612
      ASC 18,     :XECUTE    EXECUTE YOUR PROGRAM
      OCT 106612,106612,106612
      ASC 19,                    TYPE C TO CONTINUE
      NOP
```

```
PAGE5 DEF *+1
*
* DUMP
*
      OCT 116637
      OCT 106612,105212,105212
      ASC 28,    AFTER EXECUTION THE CONTENTS OF THE A, B, E AND O
      OCT 106612
      ASC 12,REGISTERS WILL BE SAVED.
      OCT 106612,106612
      ASC 15,                     :DUMP
      OCT 106612,106612
      ASC 15,WILL DISPLAY THE REGISTERS AS
      ASC 13,OCTAL AND DECIMAL VALUES.
      OCT 106612
      ASC 18,INSTRUCTIONS WILL ALSO BE PRESENTED
      ASC 12,TO DISPLAY DATA ADDRESS
      OCT 106612
      ASC 5,CONTENTS.
      OCT 106612,106612
      ASC 17,AS AN ALTERNATIVE TO USING OUTPUT
      ASC 12,INSTRUCTIONS WITHIN YOUR
      OCT 106612
      ASC 16,       PROGRAM, RESULTS CAN BE
      ASC 14, STORED IN THE REGISTERS OF
      OCT 106612
      ASC 25,       AS DATA AND THEN DUMPED AFTER EXECUTION.
      OCT 106612,105212
      ASC 9,TYPE C TO CONTINUE
      NOP
```

```
PAGE6 DEF *+1
*
* LIST
*
      OCT 116637,106612
      OCT 105212,105212
      ASC 19,                        *LIST(,M(,N))
      OCT 106612,106612
      ASC 28,TO LIST YOUR PROGRAM SEQUENTIALLY STATEMENT BY STATEMENT
      OCT 106612,106612
      ASC 28,M AND N, IF PRESENT SPECIFY THE FIRST AND LAST STATEMENT
      OCT 106612
      ASC 27,         TO BE LISTED.  IF N IS ABSENT THEN ALL STATE-
      OCT 106612
      ASC 28,         MENTS FROM M ON ARE LISTED.  IF NEITHER APPEAR
      OCT 106612
      ASC 21,         THEN THE WHOLE PROGRAM IS LISTED.
      OCT 106612,106612
      ASC 27,         BUT IF N IS LESS THAN M LISTING IS SUPPRESSED.
      OCT 106612,105212
      ASC 9,TYPE C TO CONTINUE
      NOP
```

```
PAGE7 DEF *+1
*
* SEQUENCE
*
      OCT 116637,105212,106612
      ASC 28,     WHILE ENTERING YOUR PROGRAM YOU MAY WANT TO CHANGE
      OCT 106612
      ASC 22,STATEMENT SEQUENCING.
      OCT 106612,106612
      ASC 18,                       :SEQUENCE,M,N
      OCT 106612,106612
      ASC 16,IS VERY SIMILAR TO THE SEQUENCE
      ASC 14,OPTION PRESENTED EARLIER FOR
      OCT 106612
      ASC 22,M AND N ARE TWO POSITIVE INTEGERS SUCH THAT
      OCT 106612,106612
      ASC 24,         M    BECOMES THE FIRST STATEMENT NUMBER
      OCT 106612
      ASC 15,         N    IS THE INCREMENT
      ASC 14, FOR SUCCESSIVE STATEMENTS.
      OCT 106612,106612
      ASC 22,ON COMPLETION, THE WHOLE PROGRAM IS LISTED.
      OCT 106612,106612
      ASC 16,RESTRICTIONS ON M AND N ARE THAT
      ASC 14, M MUST NOT EXCEED 1000 AND
      OCT 106612
      ASC 17,             N MUST NOT EXCEED 25.
      OCT 106612,105212
      ASC 9,TYPE C TO CONTINUE
      NOP
```

```
PAGE8 DEF *+1
*
* XECUTE
*
      OCT 116637,105212,105212
      ASC 17,                         !XECUTE
      OCT 106612,106612
      ASC 28,WILL INITIATE THE EXECUTION OF YOUR PROGRAM.  INCOMPLETE
      OCT 106612
      ASC 15,PROGRAMS MAY ALSO BE EXECUTED
      ASC 15,BUT EXECUTION WILL IMMEDIATELY
      OCT 106612
      ASC 14,HALT, WITH A WARNING MESSAGE
      ASC 16, PRINTED, IF THERE IS A MACHINE
      OCT 106612
      ASC 20,INSTRUCTION HAVING A FORWARD REFERENCE.
      OCT 106612,106612
      ASC 28,     IMMEDIATELY AFTER EXECUTION OR AFTER ENCOUNTERING A
      OCT 106612
      ASC 15,FORWARD REFERENCE THE CONTENTS
      ASC 16, OF THE A, B, E AND O REGISTERS
      OCT 106612
      ASC 7,WILL BE SAVED.
      OCT 106612,105212
      ASC 24,                             TYPE C TO CONTINUE
      NOP
```

```
PAGE9 DEF *+1
BUFF  OCT 116637
      ASC 16,                         :EDIT
      OCT 106612,106612
      ASC 9,WILL ALLOW YOU TO
      OCT 106612,106612
      ASC 27,     DELETE  ANY NUMBER OF STATEMENTS IN YOUR PROGRAM
      OCT 106612
      ASC 21,     INSERT  BETWEEN SUCCESSIVE STATEMENTS
      OCT 106612
      ASC 14,     REPLACE ANY STATEMENT.
      OCT 106612,106612
      ASC 23,ALL EDIT INSTRUCTIONS BEGIN WITH A SLASH (/).
      OCT 106612,106612
      ASC 28,     THE FOLLOWING OPERATION CAUSES STATEMENTS M THROUGH
      OCT 106612
      ASC 14,N, INCLUSIVE, TO BE DELETED
```

```
OCT 106612,106612
ASC 14,          /DELETE,M(,N)(,V)
OCT 106612,106612
ASC 27,IF ONLY M IS SPECIFIED ONLY THAT ONE STATEMENT WILL BE
OCT 106612
ASC 4,DELETED.
OCT 106612,106612
ASC 16,V, THE VETO FLAG, WHEN SPECIFIED
ASC 13, INITIATES THE PRINTING OF
OCT 106612
ASC 27,                  ALL STATEMENTS INVOLVED IN THE EDIT.
OCT 106612
ASC 26,                  TYPE IN  YES  TO CONTINUE THE EDIT
OCT 106612
ASC 27,                  OR   NO  TO VETO THE EDIT OPERATION.
OCT 106612,106612
ASC 24,                              TYPE C TO CONTINUE
NOP
```

```
PGE10 DEF *+1
*
* EDIT 2
*
      OCT 116637,106612
      ASC 22,     TO INSERT BETWEEN SUCCESSIVE STATEMENTS
      OCT 106612,106612
      ASC 12,           /INSERT,M(,N)
      OCT 106612,106612
      ASC 14,IF ONLY M IS SPECIFIED ONLY
      ASC 15,STATEMENT M WILL BE INSERTED.
      OCT 106612
      ASC 27,N IS AN INCREMENT FOR MORE THAN ONE INSERTION BETWEEN
      OCT 106612
      ASC 11,SUCCESSIVE STATEMENTS.
      OCT 106612,106612
      ASC 17,     BY MEANS OF AN EDIT OPERATION
```

```
ASC 10, STATEMENT M CAN BE
OCT 106612
ASC 15,REPLACED BY A SINGLE STATEMENT
OCT 106612,106612
ASC 12,      /REPLACE,M(,V)
OCT 106612,106612
ASC 17,A MACHINE CODE INSTRUCTION CANNOT
ASC 12,BE REPLACED BY DATA NOR
OCT 106612
ASC 16,CAN A DATA STATEMENT BE REPLACED
ASC 13, BY A MACHINE INSTRUCTION.
OCT 106612,106612
ASC 7,      /END
OCT 106612,106612
ASC 15,THE END INSTRUCTION TERMINATES
ASC 14, THE CURRENT EDIT OPERATION.
OCT 106612,105212
ASC 24,                     TYPE C TO CONTINUE
NOP
```

```
PGE11 DEF *+1
*
* LAST PAGE
*
      OCT 116677
      OCT 105212
      ASC 28,     NOTE THAT THIS IS A SMALL ASSEMBLER NOT CAPABLE OF
      OCT 106612
      ASC 27,HANDLING LARGE PROGRAMS.  PROGRAM AREA OVERFLOW WILL
      OCT 106612
      ASC 17,TERMINATE ALL ASSEMBLY.  PAY CLOSE
      ASC 12, ATTENTION FOR OVERFLOW
      OCT 106612
      ASC 9,WARNING MESSAGES.
      OCT 106612,106612
      ASC 27,     ONE IMPORTANT PROGRAMMING CONSIDERATION INVOLVES
      OCT 106612
      ASC 18,THE DEF PSEUDO OP USED FOR DEFINING
```

```
ASC 11,ADDRESSES.  ITS USAGE
OCT 106612
ASC 16,IS RESTRICTED TO DATA ADDRESSES.
OCT 106612,106612
ASC 28,    MORE IMPORTANTLY, THE DEF PSEUDO OP SHOULD PRECEDE
OCT 106612
ASC 15,ALL DATA WHICH MAY BE INVOLVED
ASC 14, IN ANY DATA EDIT OPERATIONS
OCT 106612
ASC 28,OR FOLLOW ALL DATA DEFINITIONS AFTER THE LAST DATA EDIT
OCT 106612
ASC 28,OPERATION.  FAILURE TO DO SO MAY RESULT IN AN INCORRECT
OCT 106612
ASC 25,ADDRESS REFERENCE AND MEANINGLESS PROGRAM RESULTS.
OCT 106612,106612
ASC 21,        YOU MAY NOW BEGIN PROGRAM ENTRY
OCT 106612,106612
ASC 19,        TYPE IN YOUR FIRST STATEMENT
OCT 106612
NOP
END START
```

PAGE 1

```
     YOU ARE COMMUNICATING WITH A HEWLETT PACKARD 2100A
COMPUTER THAT HAS BEEN PREPARED TO READ IN AND ASSEMBLE
COMPUTER PROGRAMS WHICH YOU ENTER.

     A COMPUTER PROGRAM IS A SERIES OF COMMANDS TO DIRECT
THE COMPUTER IN A STEP BY STEP PROBLEM SOLVING PROCEDURE.

     SUCH COMMANDS RECOGNIZED BY THE COMPUTER ARE IN THE
FORM OF MACHINE LANGUAGE, BUT PROGRAMMING IN MACHINE
LANGUAGE IS A TEDIOUS PROCESS AND ONE OF THE MOST IMPORTANT
STEPS IN TRYING TO MAKE PROGRAMMING EASIER IS TO INTRODUCE
INSTRUCTION CODES IN PLACE OF MACHINE CODES AND ADDRESSES.
THE USE OF INSTRUCTION CODES LEADS TO A PROGRAMMING LANGUAGE
ALMOST EQUIVALENT TO MACHINE CODE BUT EASIER TO READ.  A
PROGRAM TO TRANSLATE SUCH A LANGUAGE INTO THE CORRESPONDING
MACHINE LANGUAGE IS CALLED AN ASSEMBLER.

     THE TASK OF AN ASSEMBLER IS TO TRANSLATE ASSEMBLY
INSTRUCTIONS INTO MACHINE LANGUAGE INSTRUCTIONS CORRESPONDING
WITH WHAT APPEARS IN THE ASSEMBLY LANGUAGE PROGRAM.

     IT IS NOW POSSIBLE TO TRANSFER CONTROL TO THE CRT SCREEN.

     TYPE S TO PRINT OUTPUT ON CRT SCREEN

     OTHERWISE TYPE C TO CONTINUE

     (TYPE RETURN KEY TO ENTER ALL RESPONSES)
```

PAGE 2

```
     AN ASSEMBLER NORMALLY BEGINS ASSEMBLY ONCE THE PROGRAM
HAS BEEN FULLY DEFINED.  REFERENCES TO UNDEFINED INSTRUCTIONS
OR DATA WILL TERMINATE THE ASSEMBLY OR HALT FURTHER SYSTEM
ACTIVITY AFTER ASSEMBLY.

     THIS ASSEMBLER IS AN INCREMENTAL ASSEMBLER FOR ASSEMBLY
OCCURS IMMEDIATELY AFTER STATEMENT ENTRY.  THE ASSEMBLER
DOES NOT WAIT UNTIL THE PROGRAM IS FULLY DEFINED.  UNDEFINED
REFERENCES ARE RETAINED UNTIL DEFINITION OCCURS.

     EACH STATEMENT IS SEQUENCED AND ASSIGNED A STATEMENT
NUMBER.  BY DEFAULT THE FIRST STATEMENT NUMBER IS 10 WITH EACH
SUCCESSIVE STATEMENT NUMBER INCREMENTED BY 10.

     TO SPECIFY ALTERNATE SEQUENCING TYPE S FOLLOWED BY THE
FIRST STATEMENT NUMBER THEN A VALUE FOR THE INCREMENT.
USE COMMAS (,) TO SEPARATE THE S AND THE TWO VALUES.

     FOR EXAMPLE:S,12,6

RESULTS WITH THE FIRST INSTRUCTION ASSIGNED THE STATEMENT
NUMBER 12 WITH THE FOLLOWING STATEMENTS ADVANCED BY 6.

     OR TYPE C TO CONTINUE
```

PAGE 3

IF YOUR ARE FAMILIAR WITH THE FEATURES OF THE ASSEMBLER YOU MAY BEGIN ENTRY OF AN ASSEMBLY LANGUAGE PROGRAM.

TYPE C TO CONTINUE

WAIT FOR SYSTEM RESPONSE

BEGIN PROGRAM ENTRY

ELSE TYPE L TO LEARN ABOUT THE VARIOUS SYSTEM FEATURES

PAGE 4

THERE ARE 6 SYSTEM DIRECTIVES WHICH MAY BE ENTERED ANY TIME WHILE DEFINING YOUR PROGRAM, EXCEPT DURING AN EDIT. THEY ALLOW YOU GREATER CONTROL OVER THE ASSEMBLER AND THE DESIGN OF YOUR PROGRAM.

THESE DIRECTIVES ARE ALL PRECEDED BY A COLON (:)

:ABORT	DISCONTINUE PROGRAM ENTRY, BEGIN AGAIN
:DUMP	DUMP REGISTER CONTENTS
:EDIT	EDIT THE EXISTING PROGRAM
:LIST	LIST ALL OR PART OF YOUR PROGRAM
:SEQUENCE	CHANGE THE SEQUENCING, THEN LIST THE PROGRAM
:XECUTE	EXECUTE YOUR PROGRAM

TYPE C TO CONTINUE

DUMP

AFTER EXECUTION THE CONTENTS OF THE A, B, E AND O REGISTERS WILL BE SAVED.

```
:DUMP
```

WILL DISPLAY THE REGISTERS AS OCTAL AND DECIMAL VALUES. INSTRUCTIONS WILL ALSO BE PRESENTED TO DISPLAY DATA ADDRESS CONTENTS.

AS AN ALTERNATIVE TO USING OUTPUT INSTRUCTIONS WITHIN YOUR PROGRAM, RESULTS CAN BE STORED IN THE REGISTERS AS DATA AND THEN DUMPED AFTER EXECUTION.

TYPE C TO CONTINUE

LIST

```
:LIST(,M(,N))
```

TO LIST YOUR PROGRAM SEQUENTIALLY STATEMENT BY STATEMENT

M AND N, IF PRESENT SPECIFY THE FIRST AND LAST STATEMENT TO BE LISTED. IF N IS ABSENT THEN ALL STATEMENTS FROM M ON ARE LISTED. IF NEITHER APPEAR THEN THE WHOLE PROGRAM IS LISTED.

BUT IF N IS LESS THAN M LISTING IS SUPRESSED.

TYPE C TO CONTINUE

XECUTE

```
                              :XECUTE

WILL INITIATE THE EXECUTION OF YOUR PROGRAM.  INCOMPLETE
PROGRAMS MAY ALSO BE EXECUTED BUT EXECUTION WILL IMMEDIATELY
HALT, WITH A WARNING MESSAGE PRINTED, IF THERE IS A MACHINE
INSTRUCTION HAVING A FORWARD REFERENCE.

     IMMEDIATELY AFTER EXECUTION OR AFTER ENCOUNTERING A
FORWARD REFERENCE THE CONTENTS OF THE A, B, E AND O REGISTERS
WILL BE SAVED.

                                  TYPE C TO CONTINUE
```

SEQUENCE

```
     WHILE ENTERING YOUR PROGRAM YOU MAY WANT TO CHANGE
STATEMENT SEQUENCING.

                           :SEQUENCE,M,N

IS VERY SIMILAR TO THE SEQUENCE OPTION PRESENTED EARLIER FOR
M AND N ARE TWO POSITIVE INTEGERS SUCH THAT

          M    BECOMES THE FIRST STATEMENT NUMBER
          N    IS THE INCREMENT FOR SUCCESSIVE STATEMENTS.

ON COMPLETION, THE WHOLE PROGRAM IS LISTED.

RESTRICTIONS ON M AND N ARE THAT M MUST NOT EXCEED 1000 AND
             N MUST NOT EXCEED 25.


TYPE C TO CONTINUE
```

EDIT 1

```
                         :EDIT

WILL ALLOW YOU TO

     DELETE  ANY NUMBER OF STATEMENTS IN YOUR PROGRAM
     INSERT  BETWEEN SUCCESSIVE STATEMENTS
     REPLACE ANY STATEMENT.

ALL EDIT INSTRUCTIONS BEGIN WITH A SLASH (/).

     THE FOLLOWING OPERATION CAUSES STATEMENTS M THROUGH
N, INCLUSIVE, TO BE DELETED

          /DELETE,M(,N)(,V)

IF ONLY M IS SPECIFIED ONLY THAT ONE STATEMENT WILL BE
DELETED.

V, THE VETO FLAG, WHEN SPECIFIED INITIATES THE PRINTING OF
                  ALL STATEMENTS INVOLVED IN THE EDIT.
                  TYPE IN  YES  TO CONTINUE THE EDIT
                  OR   NO  TO VETO THE EDIT OPERATION.

                              TYPE C TO CONTINUE
```

EDIT 2

```
     TO INSERT BETWEEN SUCCESSIVE STATEMENTS

          /INSERT.M(.N)

IF ONLY M IS SPECIFIED ONLY STATEMENT M WILL BE INSERTED.
N IS AN INCREMENT FOR MORE THAN ONE INSERTION BETWEEN
SUCCESSIVE STATEMENTS.

     BY MEANS OF AN EDIT OPERATION STATEMENT M CAN BE
REPLACED BY A SINGLE STATEMENT

          /REPLACE.M(.V)

A MACHINE CODE INSTRUCTION CANNOT BE REPLACED BY DATA NOR
CAN A DATA STATEMENT BE REPLACED BY A MACHINE INSTRUCTION.

          /END

THE END INSTRUCTION TERMINATES THE CURRENT EDIT OPERATION.

                              TYPE C TO CONTINUE
```

LAST

```
     NOTE THAT THIS IS A SMALL ASSEMBLER NOT CAPABLE OF
HANDLING LARGE PROGRAMS.  PROGRAM AREA OVERFLOW WILL
TERMINATE ALL ASSEMBLY.  PAY CLOSE ATTENTION FOR OVERFLOW
WARNING MESSAGES.

     ONE IMPORTANT PROGRAMMING CONSIDERATION INVOLVES
THE DEF PSEUDO OP USED FOR DEFINING ADDRESSES.  ITS USAGE
IS RESTRICTED TO DATA ADDRESSES.

     MORE IMPORTANTLY, THE DEF PSEUDO OP SHOULD PRECEDE
ALL DATA WHICH MAY BE INVOLVED IN ANY DATA EDIT OPERATIONS
OR FOLLOW ALL DATA DEFINITIONS AFTER THE LAST DATA EDIT
OPERATION.  FAILURE TO DO SO MAY RESULT IN AN INCORRECT
ADDRESS REFERENCE AND MEANINGLESS PROGRAM RESULTS.

          YOU MAY NOW BEGIN PROGRAM ENTRY

          TYPE IN YOUR FIRST STATEMENT
```

APPENDIX C

DIRECT MEMORY ACCESS

DIRECT MEMORY ACCESS

Disc input operations will be handled by Direct Memory Access, DMA, a facility to provide a direct data path software assignable between memory and a high speed peripheral device.

DMA transfers are accomplished in blocks which are initiated by an initialization routine and from then on operation is under automatic control of the hardware. The initialization tells DMA which direction to transfer the data, which I/O channel is involved and how much data to transfer. Completion will be signalled by an interrupt to the DMA channel address, address 00006.

The information required to initialize DMA is given by the control words which must be specifically addressed to the DMA interface card.

Control Word 1 identifies the I/O channel in bits 0 - 5 and offers two options

Bit 15 = 1 Give STC to I/O channel at end of each DMA cycle (except last cycle if input operation)

= 0 No STC

Bit 13 = 1 Give CLC to I/O channel at end of block transfer

= 0 No CLC

The disc data channel specified on Control Word 1 is 11_8; the disc command channel is 12_8. Both STC and CLC options were selected.

Control Word 2 gives the starting memory address for the block transfer. Bit 15 determines whether the data is to go into memory (=1) or out of memory (=0).

Control Word 3 is the two's complement of the number of words to be transferred into or out from memory. The disc controller will transfer the data in 128 word blocks but this is not intended to imply that DMA transfers must be in multiples of 128. DMA may transfer any number of words within the bounds of available memory. Any buffer less than 128 words will be zero filled.

One important difference should be noted when doing a DMA input operation from a disc. Due to the asynchronous nature of disc storage and the design of the interface, the order of staring must be reversed, thus start the DMA first then the disc.

APPENDIX D

NON-INTERRUPT TRANSFER ROUTINES

NON-INTERRUPT TRANSFER ROUTINES

It is possible to transfer data without using the interrupt system which involves a "wait-for-flag" method in which the computer commands the device to operate and then waits for the completion response. It is assumed that computer time is relatively unimportant.

INPUT

The operation begins with a program instruction to set the control and clear the flag on the addressed interface card. In this example, it will be assumed that the interface card is in the slot for select code 16, thus the instruction STC 16,C. The computer goes into a waiting loop, repeatedly checking the status of the flag bit. If the flag is not set the JMP *-1 instruction causes a jump back to the SFS instruction. When the flag is set the skip condition for a SFS is met and the JMP instruction is skipped. The computer thus exists from the waiting loop and the LIB 16 loads the device input data into (B).

```
INSTRUCTIONS              COMMENTS

 STC 16,C        Start device
 SFS 16          Is input ready
 JMP *-1         No, repeat previous instruction
 LIB 16          Yes, load input into (B)
```

OUTPUT

The first step is to transfer the output to the interface buffer; the OTB 16 instruction does this. Then STC 16,C commands the device to operate and accept the data. The computer then goes into the waiting loop, the same as described for an input operation. When the flag is set indicating the device has accepted the data, the computer exits from the loop. (In the example, the final NOP is for illustration purposes only).

```
INSTRUCTIONS                 COMMENTS

 OTB 16           Output (B) to buffer
 STC 16,C         Start device
 SFS 16           Has device accepted data
 JMP *-1          No, repeat previous instruction
 NOP              Yes, proceed
```

APPENDIX E

DUMP AND LIST OUTPUT

```
:LIST PROGRAM




000010 *
000020 * SAMPLE PROGRAM FOR LIST AND DUMP OUTPUT
000030 *
000040       CLA            CLEAR A REGISTER
000050       CCB            CLEAR AND COMPLEMENT B REGISTER
000060       STO            SET OVERFLOW REGISTER

 *LIST ENDS*



@:XECUTE PROGRAM
@:DUMP PROGRAM RESULTS

A REGISTER OCTAL   000000
           DECIMAL 000000


B REGISTER OCTAL   177777
           DECIMAL -00001


E REGISTER 1


O REGISTER 1


TYPE R TO RETURN
ELSE TYPE D, FOLLOWED BY OPERAND TO BE DUMPED

@R
```

```
:L(IST)
```

```
000005 *
000010 * SAMPLE PROGRAM FOR LIST AND DUMP OUTPUT
000015 *
000020       CLA            CLEAR A REGISTER
000025       CCB            CLEAR AND COMPLEMENT B REGISTER
000030       STO            SET OVERFLOW REGISTER
000035       LDA ALPHA+1    LOAD A AND B REGISTERS
000040       LDB BETA
000045 *
000050 ALPHA DEC  11,12,13 DECIMAL CONSTANTS
000055 BETA  OCT  11,12,13 OCTAL   CONSTANTS
000060 *

 *LIST ENDS*
```

```
@:X(ECUTE)
@:D(UMP)

A REGISTER OCTAL    000014
           DECIMAL  000012


B REGISTER OCTAL    000011
           DECIMAL  000009


E REGISTER 1


O REGISTER 1


TYPE R TO RETURN
ELSE TYPE D, FOLLOWED BY OPERAND TO BE DUMPED

@D,ALPHA
 DECIMAL 000011
 OCTAL   000013


TYPE R TO RETURN
ELSE TYPE D, FOLLOWED BY OPERAND TO BE DUMPED

@D,BETA-1
 DECIMAL 000013
 OCTAL   000015


TYPE R TO RETURN
ELSE TYPE D, FOLLOWED BY OPERAND TO BE DUMPED

@D,BETA+1
 DECIMAL 000010
 OCTAL   000012


TYPE R TO RETURN
ELSE TYPE D, FOLLOWED BY OPERAND TO BE DUMPED

@R
```

```
@:LIST,5,30
```

```
000005 *
000010 * SAMPLE PROGRAM FOR LIST AND DUMP OUTPUT
000015 *
000020       CLA              CLEAR A REGISTER
000025       CCB              CLEAR AND COMPLEMENT B REGISTER
000030       STO              SET OVERFLOW REGISTER

 *LIST ENDS*
```

```
@:LIST,28,32
```

```
000030       STO              SET OVERFLOW REGISTER

 *LIST ENDS*
```

```
@:LIST,35
```

```
000035        LDA ALPHA+1  LOAD A AND B REGISTERS
000040        LDB BETA
000045 *
000050 ALPHA DEC 11,12,13 DECIMAL CONSTANTS
000055 BETA  OCT 11,12,13 OCTAL   CONSTANTS
000060 *

 *LIST ENDS*
```

APPENDIX F

MEMORY MAP AND FUNCTIONAL UNIT RELATION CHART

INTRODUCTION

The Memory Map offers a through listing of all the program units. The address of almost every subroutine as well as a brief description of the subroutine has been included.

Immediately following the Memory Map is a chart to display the relationship between the program units on each page. For each program unit there is a list of the units called and also a list of the different program units which call each particular unit. The number following each entry in the chart refers to the page on which the unit resides.

MEMORY MAP

PAGE 0

ADDRESS	
00000	A REGISTER
00001	B REGISTER
00002 00003	EXIT SEQUENCE TO FORWARD REFERENCE WARNING IF A AND B CONTENTS ARE USED AS EXECUTABLE INSTRUCTIONS
00004	POWER FAIL INTERRUPT HALT
00005	MEMORY PROTECT/PARITY ERROR HALT
00006	DIRECT MEMORY ACCESS CHANNEL
00011	DISC DATA CHANNEL
00012	DISC CONTROL CHANNEL
00101	JUMP TO INITIALIZATION
00103	BASE PAGE LINKAGE OF SYSTEM SUBROUTINES
00172	ASSEMBLER TABLE ADDRESSES
	CONSTANTS
00211	Decimal constants
00313	Octal constants
00343	Alphabetic constants
	VARIABLES
00365	System variables
00416	Temporary variables
00427	Edit variables
00511	CONSTANTS AND VARIABLES FOR DISC INPUT DRIVER
00516	CHARACTER CONSTANTS
	BUFFERS
00532	Input buffer
00576	Auxiliary input buffer
00642	Data store buffer
00677	OCTAL CONSTANTS
00714	INTERRUPT HALTS

00717	INTERRUPT SERVICE SUBROUTINE CALLS	
00727	ERROR MESSAGE OUTPUT	
	ERROR MESSAGE SUBROUTINES	
00724	ERROR	Call BPLN and REENT
00730	REENT	Print re-entry request
00753	BPLN	Print error message
00763	BASE PAGE ERROR MESSAGES	
01131	TABLE OVERFLOW WARING	
	INTERRUPT SERVICE SUBROUTINES	
01154	DMASS	Clear control flag on DMA channel
01157	DCSS	Clear control flag on disc data channel
01162	CCSS	Clear control flag on disc control channel
	INITIALIZATION SUBROUTINE	
01165	CNFIG	Configure I/O package
	LEXICAL SCAN SUBROUTINES	
01234	GETCR	Get next character from input buffer
01253	NTBLK	Get next non blank character
01262	RDCOM	Read upto a comma in buffer
01272	BCKSP	Back up one character in buffer
01304	TRMCK	Check for valid terminator character
01313	SAVEE	Save present contents of (E)
01317	RSTRE	Resore contents of (E)
	ASSEMBLY SUBROUTINES	
01323	WMOVE	Move N words
01340	DATAD	Adjust address for data address
01350	IDRCT	Mask on indirect reference bit
	EXECUTION SUBROUTINE	
01355	SAVR	Save register contents after execution
	EDIT SUBROUTINES	
01365	EDTAD	Prepare address pointer for edit
01374	PREPR	Prepare to scan editted text
01411	DSCB	Delete from Source Code Block
01450	SNGDL	Delete a single machine code instruction
01457	XDEL	Find assembled instruction after deletion
01504	SVPSN	Save user program position before edit
01510	JMPE1	Insert one jump during edit
01515	JMPAF	Place return after edit entry
01525	JMPBF	Place link to edit entry
01532	JMPS	Store two jump instructions to link edit entry

DISC INPUT DRIVER SUBROUTINES

01543	DISKI	Disc input controller
01562	DISKD	Disc input driver
01607	SEEK	Output disc head positioning commands
01652	RSEEK	Output disc seek after ten read errors
01662	STAT	Retrieve disc status word

PAGE 1

ADDRESS

02000 SYSTEM CONTROLLER

INPUT/OUTPUT PACKAGE

02041	DATIN	Request input
02103	TTY.I	Preform input operation
02122	TTY.P	Preform output operation
02165	I.OFF	Turn off interrupt mode
02172	I.ON	Turn on interrupt mode
02202	PROCS	Character processing for input
02252	GETCH	Character processing for output
02266	INIT	Initialize for output
02300	I.STP	Interrupt service
02312	NWLNS	Output multiple carriage return line feed
02320	CRLFD	Output carriage return line feed
02324	CNDEC	Binary to Ascii decimal
02330	CNOCT	Binary to Ascii octal
02334	CNBIN	Store converted value
02370	DVUKN	Divide value to be converted

STATEMENT STORE

02410	STSCB	Store statement in Source Code Block
02457	LBDEF	Define label beginning statement

02526 SYSTEMS DIRECTIVE CONTROLLER

02530	ABORT	Abort program
02632	DUMP	Branch to Dump routine
02534	EDIT	Prepare for an edit operation
02567	HALT	Halt the computer
02574	LIST	Interpret and execute List request
02643	SEQUENCE	Branch to sequence routine
02645	XECUTE	Branch to execute user program

02676 SEQUENCE DIRECTIVE EXECUTION

DUMP DIRECTIVE EXECUTION

02721	Dump register contents
02756	Dump data address contents

	DUMP SUBROUTINES	
03030	EODMP	Prepare to dump either (E) or (O)
03040	RGDP1	Dump (A) or (B)
03061	RGDP2	Dump (E) or (O)
03072	RGDP3	Print register name
03103	ASCDC	Convert binary to Ascii decimal with minus sign if needed
03123	TEXT FOR DUMP OUTPUT	
03211	DUMP ERROR MESSAGES	
03231	USER PROGRAM EXECUTION	
03257	FORWARD REFERENCE EXECUTION WARNING	
	EXECUTION SUBROUTINES	
03334	SSTDF	Define compound operands
03446	PLCDF	Define Program Location Counter (PLC) references
03535	FNDAD	Find address for PLC or compound operands
03625	FWDRF	Define forward references
03671	LIST SUBROUTINE	

PAGE 2

ADDRESS		
	LEXICAL SCAN	
04000	LEX	Main lexical scan subroutine to scan all source program statements
04517	LEXICAL ERROR MESSAGES	
	LEXICAL SUBROUTINES	
05174	RANGE	Check range of operand value
05212	STDAT	Store data in temporary buffer
05245	VAL	Input temporary value for undefined symbol
05237	LABCK	Read in and examine operand for data definition
05350	CLEAR	Initialize all variables in lexical scan
05401	LOKUP	Symbol Table look up
05416	FIND	Find symbol address in Symbol Table
05550	MNEM	Look up mnemonic in Instruction Table
05672	DATFL	Check for data table overflow

PAGE 3

ADDRESS		
	NUMBER MANIPULATION SUBROUTINES	
06000	CONST	Input a decimal constant
06020	NUMCK	Fetch number and convert to binary
06227	.PACK	Normalize and pack floating point number
06302	NORML	Normalize value and exponent
06336	MBY10	Multiply unpacked number by ten
06367	DBY10	Divide unpacked number by ten
06423	MPY	Multiply integer in (A)
06461	DECHK	Examine character to be decimal digit
06500	TYPCK	Determine real or integer
06515	IFIX	Convert real to integer
06553	TWINT	Input one or two decimal integers
06607	GTNUM	Input a positive decimal integer
06616	OCTIN	Input an octal integer
06662	OCTCK	Examine decimal or octal operand integer
07000	ERROR MESSAGES FOR NUMBER ROUTINES	
	LEXICAL AND DUMP SUBROUTINES	
07155	OPREC	Read in operand
07435	LABRD	Read a symbol
07501	LETPR	Check character to be alphabetic or period
07516	DATRG	Check address to be in program data table range
	EXECUTION SUBROUTINE	
07561	CDSCN	Scan user program for forward references
	SEQUENCE SUBROUTINE	
07657	SQNCE	Read in user defined statement numbers

PAGE 4

ADDRESS		
	INSTRUCTION ASSEMBLY	
	ASSEMBLY SUBROUTINES	
10000	SETCD	Set and store instructions in appropriate program area Evaluate and store all memory reference operands
10327	DETLN	Determine assembly length for a Memory Reference instruction
10336	ASMBL	Allocate space in Source Code Block for storing statement

10511	DTSET	Store data definition in program data area
10535	STLBL	Store symbol in Symbol table
10622	STRCD	Store instruction in program area
10627	STRCK	Check user program area for overflow
10664	STPLC	Store Program Location Counter reference

EDIT SUBROUTINES

11000	CMOVE	Move assembled code
11066	CASCD	Adjust forward reference pointers of statements involved in an edit
11210	DELTE	Delete statement from assembled code
11332	DTEDD	Delete data definition
11405	DTEDI	Insert data definition
11475	SCSYM	Adjust data address after an edit
11623	STFSP	Store length and address of deletion from Source Code Block
11727	ASMAD	Retrieve assembly addresses of instructions involved in an edit

PAGE 5

ADDRESS

12000 EDIT CONTROLLER (INSTRUCTION SCAN)

EDIT SUBSYSTEMS

12267	Single Delete
12323	Multiple Delete
12437	Single Insert
12476	Multiple Insert
12542	Replace
12651	End

EDIT SUBROUTINES

12661	EDCLR	Initialize edit variables
12701	VETCK	Check for a veto request

12726 EDITOR ERROR MESSAGES

EDIT SUBROUTINES

13207	EDIPT	Source code input during an edit operation
13305	ISCB	Link insert with Source Code Block
13325	XINS	Find assembled instruction which precedes insert
13412	YINS	Find assembled instruction which follows insert
13462	MULIN	Prepare for and begin machine code multiple insert

13544	ENDMI	End a multiple insert operation
13603	RSCB	Link replacement with Source Code Block

PAGE 6

ADDRESS	
14000	INITIALIZATION PROGRAM
14340	DISC INPUT STORE BUFFER

ASSEMBLER TABLES

ADDRESS	
15200	INSTRUCTION TABLE
15602	SYMBOL TABLE
17160	SPECIAL SYMBOL TABLE (SST)
17634	PROGRAM LOCATION COUNTER TABLE
20000	SOURCE CODE BLOCK (SCB)
25700	FREE SPACE TABLE
26001	USER PROGRAM AREA
26701	PROGRAM DATA TABLE

PROGRAM UNIT INTERRELATION

PAGE 0	CALLING PROGRAM	PROGRAM CALLED
ERROR MESSAGE PROCESSOR	THROUGHOUT THE PROGRAM	I/O PACKAGE (1)
INTERRUPT SERVICE SUBROUTINES	DISC INPUT DRIVER (0)	
INITIALIZATION SUBROUTINE	INITIALIZATION PROGRAM (6)	
LEXICAL SCAN SUBROUTINES	LEXICAL SCAN (2) SYSTEM DIRECTIVE CONTROLLER (1) EDIT CONTROLLER (5)	
ASSEMBLY SUBROUTINES	STATEMENT ASSEMBLY (4)	
EXECUTION SUBROUTINE	XECUTE DIRECTIVE (1)	
EDIT SUBROUTINES	EDIT SUBSYSTEMS (5) EDIT SUBROUTINES (5) EDIT DIRECTIVE (1)	
DISC INPUT DRIVER	INITIALIZATION (6)	INTERRUPT SERVICE SUBROUTINES (0)

PAGE 1	CALLING PROGRAM	PROGRAM CALLED
SYSTEM CONTROLLER		I/O PACKAGE (1) LEXICAL ROUTINES (0) STATEMENT ASSEMBLY (4) STATEMENT STORAGE (1)
I/O PACKAGE	THROUGHOUT THE PROGRAM	
STATEMENT STORAGE	SYSTEM CONTROLLER (1)	
SYSTEM DIRECTIVE CONTROLLER (SDC)	SYSTEM CONTROLLER (1)	LEXICAL ROUTINE (0)

PAGE 1	CALLING PROGRAM	PROGRAM CALLED
DUMP	SDC (1)	DUMP SUBROUTINES (1) LEXICAL AND DUMP SUBROUTINES (2,3)
EDIT	SDC (1)	EDIT SUBROUTINES (0)
LIST	SDC (1) SEQUENCE DIRECTIVE (1) EDIT CONTROLLER (5)	LIST SUBROUTINE (1) NUMBER MANIPULATION SUBROUTINES (3)
SEQUENCE	SDC (1)	STATEMENT NUMBER SUBROUTINE (3) LIST DIRECTIVE (1)
XECUTE	SDC (1) LEXICAL SCAN (2)	XECUTE SUBROUTINES (1,3)

PAGE 2	CALLING PROGRAM	PROGRAM CALLED
MAIN LEXICAL SCAN SUBROUTINE	SYSTEM CONTROLLER (1) EDIT SUBROUTINES (4,5)	LEXICAL ROUTINES (0,2,3) NUMBER MANIPULATION ROUTINES (3)
LEXICAL SCAN SUBROUTINES	MAIN LEXICAL SCAN SUBROUTINE (2)	LEXICAL SUBROUTINES (0) NUMBER MANIPULATION ROUTINES (3)

PAGE 3	CALLING PROGRAM	PROGRAM CALLED
NUMBER MANIPULATION ROUTINES	LEXICAL SCAN SUBROUTINES (2,3) EDIT CONTROLLER (5) SDC (1)	LEXICAL SUBROUTINES (0)
LEXICAL AND DUMP SUBROUTINES	LEXICAL SCAN (2) DUMP DIRECTIVE (1)	LEXICAL SCAN (0) NUMBER MANIPULATION ROUTINES (3)
EXECUTION SUBROUTINE	XECUTE DIRECTIVE (1)	

PAGE 3	CALLING PROGRAM	PROGRAM CALLED
SEQUENCE SUBROUTINE (STATEMENT NUMBER INPUT)	SEQUENCE DIRECTIVE (1) INITIALIZATION (6)	NUMBER MANIPULATION ROUTINES (3)

PAGE 4	CALLING PROGRAM	PROGRAM CALLED
INSTRUCTION ASSEMBLY	SYSTEM CONTROLLER (1) EDIT SUBSYSTEMS (5) EDIT SUBROUTINES (4,5)	ASSEMBLY SUBROUTINES (0)
EDIT SUBROUTINES	EDIT CONTROLLER (5) EDIT SUBSYSTEMS (5)	LEXICAL SCAN (2) ASSEMBLY SUBROUTINE (4)

PAGE 5	CALLING PROGRAM	PROGRAM CALLED
EDIT CONTROLLER	SYSTEM CONTROLLER (1)	EDIT SUBSYSTEMS (5) EDIT SUBROUTINES (4,5) LEXICAL SUBROUTINES (0)
EDIT SUBSYSTEMS	EDIT CONTROLLER (5)	EDIT SUBROUTINES (0,4,5)

PAGE 6	CALLING PROGRAM	PROGRAM CALLED
INITIALIZATION	SYSTEM CONTROLLER (1)	INITIALIZATION SUBROUTINE (0) DISC INPUT DRIVER (0) STATEMENT NUMBER INPUT (3)

APPENDIX G

SOURCE PROGRAM LISTING

```
*
ASMB,A,L
*
*
*  PURPOSE
*  -------
*
* TO ASSEMBLE AND EXECUTE HEWLETT PACKARD ASSEMBLER
* LANGUAGE PROGRAMS
*
* INPUT SOURCE CODE WILL BE ASSEMBLED IMMEDIATELY AFTER
* ENTRY  FORWARD REFERENCES WILL BE RETAINED UNTIL
* DEFINITION.  EXECUTION MAY BE SPECIFIED ANY NUMBER
* OF TIMES ONCE ALL OR PART OF THE PROGRAM HAS BEEN
* DEFINED.
*
*
*
*  IMPLEMENTATION       HEWLETT PACKARD 2100A
*  --------------
*
*
*                       COMPUTER LABORATORY
*                       DEPARTMENT OF APPLIED MATHEMATICS
*                       MCMASTER UNIVERSITY
*
*                       DIRECTOR:  DR. N. SOLNTSEFF
*
*
*
*  STORAGE REQUIREMENTS
*  ------- ------------
*
* THE PROGRAM USES ALL AVAILABLE COMPUTER STORAGE.
* (12K OR 12288 WORDS)
*
* IN ADDITION, TWO TRACKS BEGINNING A DISC CARTRIDGE HAVE
* BEEN ALLOCATED TO HOLD READ ONLY DATA.  THE DATA IS SYSTEM
* INFORMATION PRESENTED TO THE USER.
*
* EACH TRACK IS 3072 WORDS UNDER THE PRESENT DOS M OPERATING
* SYSTEM.
*
```

```
*
*
*  PROGRAM RESTRICTIONS
*  ------- ------------
*
* ASSEMBLER FEATURES ARE LIMITED BY THE STORAGE CAPACITY OF
* THE COMPUTER.  ALL ASSEMBLER PROGRAMS AND TABLES ARE CORE
* RESIDENT WHICH LIMITS THE AVAILABLE AREA FOR USER PROGRAM
* TABLES (THIS IS DISCUSSED AT LENTH IN THE SECTION ON USER
* PROGRAM RESTRICTIONS).
*
*
* TWO TRACKS ON A CARTRIDGE DISC HOLD READ ONLY DATA FOR
* DISPLAY AS INTRODUCTARY INFORMATION TO THE USER. THERE ARE
* ELEVEN PAGES OF INFORMATION STORED SO THAT NO PAGE OF
* INFORMATION CROSSES A TRACK BOUNDARY. THIS IS PARTICULARLY
* IMPORTANT FOR THE DISC INPUT DRIVER USED TO INPUT THIS DATA
* CANNOT CROSS TRACK BOUNDS.
* PROGRAM ADDRESS TABLES ARE SET FOR THE DATA BEGINNING ON
* THE FIRST SECTOR OF THE FIRST TRACK.  THE DATA WAS STORED
* USING THE MOVING HEAD DISC OPERATING SYSTEM (DOS M)
* FACILITY TO WRITE ONTO USER FILES (EXEC CALL, REQUEST CODE
* 15)
*
* MOVING THE DISC RESIDENT DATA REQUIRES THAT THE ADDRESS
* TABLE (LAST TABLE IN LISTING) BE UPDATED TO COMPENSATE FOR
* THIS CHANGE.
*
*
```

```
*
*  PROGRAMMING LANGUAGE
*  ----------- --------
*
* HEWLETT PACKARD ASSEMBLY LANGUAGE FOR THE 2100 SERIES
* OF COMPUTERS (ABSOLUTE ASSEMBLY)
*
*
*
*  PRIMARY STORAGE
*  ------- -------
*
* XOPCD   OPERATION CODE TABLE FOR INSTRUCTION LOOK UP
*         (SYSTEM TABLE  NOT ACCESIBLE BY THE USER)
*
* XSTBL   MAIN SYMBOL TABLE
*
* XSST    SPECIAL TABLE FOR COMPOUND OPERANDS
*         (OPERAND WITH A LABEL AND NUMERIC VALUE)
*
* XPLC    PROGRAM LOCATION COUNTER TABLE
*         (HOLD ALL PLC REFERENCES TO BE DEFINED
*          IMMEDIATELY BEFORE EXECUTION)
*
* XSCB    SOURCE CODE TABLE
*         (STORE SOURCE PROGRAM ALONG WITH ALL
*          NECESSARY INFORMATION)
*
* XFRSP   FREE SPACE IN SOURCE CODE BLOCK
*         (STORE LENGTH AND ADDRESS OF DELETIONS FROM SCB)
*
* XUSRP   USER PROGRAM AREA FOR MACHINE CODE INSTRUCTIONS
*
* XDATA   USER PROGRAM AREA FOR DATA
*
*
*
*  AUTHOR    JAMES FORRESTER
*  ------
*            MASTER S DEGREE PROJECT
*            MCMASTER UNIVERSITY, HAMILTON ONTARIO
*
*            NOVEMBER, 1973
*
*
```

```
*
*  USER PROGRAM RESTRICTIONS
*  ---- ------- ------------
*
* 1  ASSEMBLER CONTROL STATEMENT
*
* IS NOT NECESSARY AND ANY ATTEMPT TO ENTER ONE WILL
* ONLY RESULT IN A LEXICAL ERROR.
*
* THE ASSEMBLER IS DESIGNED TO ASSEMBLE A PROGRAM
* ASSUMING   ASMB,A,L  WERE TO BE THE ASSEMBLER CONTROL
* STATEMENT.
*
* OTHER FEATURES LIKE BINARY OUTPUT OR A CCROSS REFERENCE
* TABLE ARE NOT AVAILABLE.
*
*
* 2 PROGRAM SIZE
*
* PROGRAMS ARE RESTRICTED TO SMALL LEARNING PROGRAMS FOR
* STORAGE BUFFERS ARE NOT LARGE.
* OVERFLOW BY ANY OF THE FOLLOWING USER TABLES
*
*        THE MAIN SYMBOL TABLE
*        THE SPECIAL SYMBOL TABLE
*        THE PROGRAM LOCATION COUNTER TABLE
*        THE SOURCE CODE BLOCK
*     OR THE USER PROGRAM AREAS (EITHER DATA OR MACHINE
*         INSTRUCTION)
*
* WILL IMMEDIATELY HALT ASSEMBLY WITH NO RECOVERY
* PROCEDURE.  WITH THE EXCEPTION OF THE SYMBOL TABLE, A
* WARNING IS PRINTED IF A TABLE IS CLOSE TO OVERFLOWING
* WITH INSTRUCTIONS TO BEGIN EXECUTION.
```

```
*
* BEGINNING EXECUTION IS APT TO FREE AREA IN THE PLC TABLE
* AND THE SST AREA AS OPERANDS ARE DEFINED BEFORE EXECUTION.
* SPACE IN THE OTHER TABLES CANNOT BE REPRIEVED.
*
*
* 3  PROGRAM STRUCTURE
*
* THE USER PROGRAM WLL BE TREATED AS AN ABSOLUTE PROGRAM
*
*THERE WILL NOT BE ANY
*
*          LITERALS
*          EXTERNAL SUBROUTINE CALLS
*       OR ANY FEATURES AVAILABLE USING THE OPERATING SYSTEM
*           OR RELOCATABLE ASSEMBLY
*
*
* MULTIPLE INSTRUCTIONS ARE NOT PERMITTED
*
* THE OPERAND TERM FOR MEMORY REFERENCE OR EXTENDED ARITH
* MEMORY REFERENCE INSTRUCTIONS IS LIMITED TO
*
*          (+LABEL)(+/-VALUE)(,I)
*
* THE PROGRAM LOCATION COUNTER REFERENCE (*) MAY REPLACE
* THE LABEL.
*
*
* SEVERAL PSEUDO OPS ARE NOT AVAILABLE
* (A LIST OF AVAILABLE ASSEMBLER INSTRUCTIONS AND PSEUDO OPS
* FOLLOWS)
*
```

```
*
* FOUR OF THE AVAILABLE PSEUDO OPS HAVE BEEN ALTERED FROM
* THE STANDARD HEWLETT PACKARD DEFINITION.
*
*    ABS    ADDRESS DEFINITION MUST BE WITHIN BOUNDS OF THE
*           USER PROGRAM AREA OR THE FIRST 100 (OCTAL)
*           WORDS OF COMPUTER STORAGE.
*
*           WILL INITIALIZE N (0<N≤128) STORAGE LOCATIONS TO
*           ZERO AS WELL AS ADVANCE THE LOCATION COUNTER N
*           TIMES.
*
*    DEF    IS STRICTLY RESTRICTED TO DATA ADDRESS DEFINITION
*           AN UNDEFINED OPERAND IS PERMITTED EXCEPT DURING
*           AN EDIT, BUT THE USER WILL IMMEDIATELY BE
*           REQUESTED TO DEFINE THE LABEL ON NEXT ENTRY.
*
*           DEF INSTRUCTIONS SHOULD NOT BE WITH EDIT
*           OPERATIONS NOR SHOULD THEY FOLLOW DATA INVOLVED
*           IN AN EDIT OPERATION OTHERWISE THE DEF POINTER
*           WILL BE ALTERED.
*
*    END    WILL SIGNAL END OF PROGRAM AND ADVANCE TO EXECUTION
*           ROUTINES.
*           IT WILL NOT BE STORED WITH THE USER PROGRAM
*           IN THE SOURCE CODE BLOCK AND ANY LABEL PRECEDING
*           OR ANY OPERAND FOLLOWING WILL BE IGNORED.
*           END IS NOT PERMITTED DURING AN EDIT OPERATION.
*
```

```
*
*
* ASSEMBLER MACHINE CODE INSTRUCTIONS ARE:
*
*          ADA        ADD TO (A)
*          ADB        ADD TO (B)
*          ALF        ROTATE (A) LEFT 4
*          ALR        SHIFT (A) LEFT 1, CLEAR SIGN
*          ALS        SHIFT (A) LEFT 1
*          AND        AND TO (A)
*          ARS        SHIFT (A) RIGHT 1, CARRY SIGN
*          ASL        ARITHMETIC LONG SHIFT LEFT
*          ASR        ARITHMETIC LONG SHIFT RIGHT
*          BLF        ROTATE (B) LEFT 4
*          BLR        SHIFT (B) LEFT 1, CLEAR SIGN
*          BLS        SHIFT (B) LEFT 1
*          BRS        SHIFT (B) RIGHT 1, CARRY SIGN
*          CCA        CLEAR AND COMPLEMENT (A)
*          CCB        CLEAR AND COMPLEMENT (B)
*          CCE        CLEAR AND COMPLEMENT (E) SET (E) = 1
*          CLA        CLEAR (A)
*          CLB        CLEAR (B)
*          CLC        CLEAR I/O CONTROL BIT
*          CLE        CLEAR (E)
*          CLF        CLEAR I/O FLAG
*          CLO        CLEAR OVERFLOW BIT
*          CMA        COMPLEMENT (A)
*          CMB        COMPLEMENT (B)
```

```
*        CME        COMPLEMENT (E)
*        CPA        COMPARE TO (A), SKIP IF UNEQUAL
*        CPB        COMPARE TO (B), SKIP IF UNEQUAL
*        DIV        DIVIDE
*        DLD        DOUBLE LOAD
*        DST        DOUBLE STORE
*        ELA        ROTATE (E) AND (A) LEFT 1
*        ELB        ROTATE (E) AND (B) LEFT 1
*        ERA        ROTATE (E) AND (A) RIGHT 1
*        ERB        ROTATE (E) AND (B) RIGHT 1
*        HLT        HALT
*        INA        INCREMENT (A) BY 1
*        INB        INCREMENT (B) BY 1
*        IOR        INCLUSIVE OR INTO (A)
*        ISZ        INCREMENT, THEN SKIP IF ZERO
*        JMP        JUMP
*        JSB        JUMP TO SUBROUTINE
*        LDA        LOAD INTO (A)
*        LDB        LOAD INTO (B)
*        LIA        LOAD INTO (A) FROM I/O CHANNEL
*        LIB        LOAD INTO (B) FROM I/O CHANNEL
*        LSR        LOGICAL LONG SHIFT RIGHT
*        MIA        MERGE (OR) INTO (A) FROM I/O CHANNEL
*        MIB        MERGE (OR) INTO (B) FROM I/O CHANNEL
*        MPY        MULTIPLY
*        NOP        NO OPERATION
*        OTA        OUTPUT FROM (A) TO I/O CHANNEL
*        OTB        OUTPUT FROM (B) TO I/O CHANNEL
```

```
*        RAL      ROTATE (A) LEFT 1
*        RAR      ROTATE (A) RIGHT 1
*        RBL      ROTATE (B) LEFT 1
*        RBR      ROTATE (B) RIGHT 1
*        RRL      ROTATE (A) AND (B) LEFT
*        RRR      ROTATE (A) AND (B) RIGHT
*        RSS      REVERSE SKIP SENSE
*        SEZ      SKIP IF (E) = 0
*        SFC      SKIP IF I/O FLAG = 0 (CLEAR)
*        SFS      SKIP IF I/O FLAG = 1 (SET)
*        SLA      SKIP IF LSB OF (A) IS ZERO
*        SLB      SKIP IF LSB OF (B) IS ZERO
*        SOC      SKIP IF OVERFLOW BIT = 0 (LCEAR)
*        SOS      SKIP IF OVERFLOW BIT = 1 (SET)
*        SSA      SKIP IF SIGN BIT OF (A) = 0
*        SSB      SKIP IF SIGN BIT OF (B) = 0
*        STA      STORE (A)
*        STB      STORE (B)
*        STC      SET I/O CONTROL BIT
*        STF      SET I/O CONTROL FLAG
*        STO      SET OVERFLOW BIT
*        SWP      SWITCH (A) AND (B)
*        SZA      SKIP IF (A) = 0
*        SZB      SKIP IF (B) = 0
*        XOR      EXCLUSIVE OR TO (A)
*
```

```
*
* ASSEMBLER PSEUDO OP INSTRUCTIONS ARE LIMITED TO:
*
*        ABS       DEFINE ABSOLUTE VALUE
*        ASC       GENERATE ASCII CHARACTERS
*        BSS       RESERVE BLOCK OF STORAGE
*        DEC       DEFINE DECIMAL CONSTANTS
*        DEF       DEFINE ADDRESS
*        END       TERMINATE PROGRAM (BEGIN EXECUTION)
*        EQU       EQUATE SYMBOL
*        OCT       DEFINE OCTAL CONSTANTS
*
*
      ORG 2
      SUP PRESS LISTING OF EXTENDED CODE LINES
      JMP MPPE,I   UNDEFINED OPERAND IN USER PROGRAM
      JMP MPPE,I
      HLT 4,C      HALT ON A POWER FAIL
      HLT 5        MEMORY PROTECT/ PARITY ERROR HALT
*
* NOP ALL MAIN FRAME INTERRUPT LOCATIONS
*
      OCT 0,0,0,0,0,0,0,0
      OCT 0,0,0,0,0,0,0,0
*
* FIRST 100 (OCTAL) LOCATIONS AVAILABLE TO USER
*
      ORG 101B
*
* JUMP TO INITIALIZATION
*
START JMP *+1,I
      DEF GREET
*
```

```
*
* BASE PAGE LINKAGE OF SYSTEM SUBROUTINES
*
WRITE DEF TTY.P    TTY OUTPUT LINK
STOP  DEF I.STP    STOP SERVICE LINK
ASSM  DEF ASMBL    FIND SOURCE CODE BLOCK ADDRESS
ASMD  DEF ASMAD    ADDR OF ASSEMBLED CODE IN EDIT
CLER  DEF CLEAR    INITIALIZE VARIABLES FOR LEXICAL SCAN
CMVE  DEF CMOVE    MOVE ASSEMBLED CODE
CNST  DEF CONST    REAL OR DECIMAL INTEGER
DATN  DEF DATIN    READ INPUT, RETURN FIRST CHARACTER
DLTE  DEF DELTE    DELETE ASSEMBLED CODE
DTED  DEF DTEDD    DELETE DATA DURING EDIT
DTDI  DEF DTEDI    INSERT DATA DURING EDIT
DTFL  DEF DATFL    CHECK DATA AREA OVERFLOW
DTRG  DEF DATRG    CHECK ADDRESS RANGE IN BUFFER
DTST  DEF DTSET    STORE DATA
GTNM  DEF GTNUM    INPUT POSITIVE INTEGER
IMON  DEF I.ON     TURN ON TTY INTERRUPT
IOFF  DEF I.OFF    TURN OFF INTERRUPT
ISTP  DEF I.STP    INTERRUPT SERVICE
LBCK  DEF LABCK    READ IN OPERAND CHECK LABEL
LBRD  DEF LABRD    READ LABEL
LEXI  DEF LEX      LEXICAL SCAN OF SOURCE CODE
LISTI DEF LIST     LIST PROGRAM
LOKP  DEF LOKUP
LTPR  DEF LETPR    LETTER OR PERIOD CHECK
NMBR  DEF NUMBR    INPUT DECIMAL OR OCTAL INTEGER
NWLN  DEF CRLFD    OUTPUT CR-LF
NWLS  DEF NWLNS    OUTUT MULTIPLE CR-LF
OCTN  DEF OCTIN    OCTAL INTEGER INPUT
OPRC  DEF OPREC    OPERAND RECOGNITION
```

```
SCNCD  DEF CDSCN     SCAN ASSEMBLED CODE FOR FWD REF
SFSP   DEF STFSP     LENGTH AND ADDR OF DELETE IN FR SP
SLBL   DEF STLBL     STORE LABLE IN SYMBOL TABLE
SQNC   DEF SQNCE     READ IN STATEMENT NUMBERS
STCD   DEF SETCD     SET AND STORE CODE
STCK   DEF STRCK     CHECK PROGRAM AREA OVERFLOW
TWNT   DEF TWINT     INPUT TWO POSITIVE INTEGERS
TPCK   DEF TYPCK     DETERMINE INTEGER OR REAL
*
ASME   DEF ASMEO     EDIT VARIABLE ADDRESSES
SCBE   DEF SCBEO
*
ABSSR  DEF LXRTN     RETURN TO ABS/BSS PROGRAM
CNTRL  DEF CMAND     LINK TO SYSTEM CONTROLLER
DMPRT  DEF DMP2      RETURN TO DUMP AFTER USER INSTR
EDTR   DEF EDIT      LINK TO EDIT SUPERVISOR
EDLEX  DEF EDXRT     LINK TO EDIT FOR SOURCE INPUT
GRTER  DEF GRT8
I.O    DEF TI.1      ADDR OF FIRST I/O INSTR
INT1   DEF I.OFF+2
INT2   DEF I.ON+2
INT3   DEF TP.3-1
INT4   DEF TTY.P+2
LXANL  DEF LXSCN     LINK TO LEXICAL ROUTINE
MIRTI  DEF MIRT      RETURN DURING MULTIPLE INSERT
MPPE   DEF MPPET     WARNING ABOUT UNDEFINED OPERANDS
SCBI   DEF EDRTN     RETURN FROM EDIT TO STORE IN SCB
XEQ    DEF XEQI      LINK TO EXECUTE ROUTINE
*
*
*
```

```
*
*
A     EQU 0        REGISTER REFERENCE ADDRESSES
B     EQU 1
DC    EQU 11B      DISC DATA CHANNEL
CC    EQU 12B      DISC CONTROL CHANNEL
TTY   EQU 17B      CHANNEL NUMBER I/O DEVICE
XOPCD DEF 15200B   OPCODE TABLE
XSTBL DEF 15602B   SYMBOL TABLE
XSST  DEF 17160B   AUXILIARY SYMBOL TABLE
XPLC  DEF 17634B   UNDEFINED PLC REFERENCE STORE
XSCB  DEF 20000B   SOURCE CODE BLOC
XFRSP DEF 25700B   TABLE OF FREE SPACE
XUSRP DEF 26341B   USER PROGRAM
XDATA DEF 26701B   PROGRAM DATA
*
*
YSTBL DEF 17157B
YSST  DEF 17633B
YPLC  DEF 17777B
YSCB  DEF 25677B
YFRSP DEF 25777B
YUSRP DEF 26677B
*
PROG  DEF 026340B  ADDR FOR SUBR JUMP TO USER PRGRM
```

```
*
* DECIMAL CONSTANTS
*
*
ZERO  DEC 0
.1    DEC 1
.2    DEC 2
.3    DEC 3
.4    DEC 4
.5    DEC 5
.6    DEC 6
.7    DEC 7
.8    DEC 8
.9    DEC 9
.10   DEC 10
.11   DEC 11
.12   DEC 12
.13   DEC 13
.14   DEC 14
.15   DEC 15
.16   DEC 16
.18   DEC 18
.20   DEC 20
.22   DEC 22
.24   DEC 24
.26   DEC 26
.28   DEC 28
.30   DEC 30
.32   DEC 32
.34   DEC 34
.38   DEC 38
.40   DEC 40
```

```
.48    DEC 48         ASCII ZERO
.50    DEC 50
.52    DEC 52
.64    DEC 64
.72    DEC 72
.75    DEC 75
.86    DEC 86
.115   DEC 115
.125   DEC 125
M1     DEC -1
M2     DEC -2
M3     DEC -3
M4     DEC -4
M5     DEC -5
M6     DEC -6
M7     DEC -7
M8     DEC -8
M9     DEC -9
M10    DEC -10
M12    DEC -12
M16    DEC -16
M19    DEC -19
M20    DEC -20
M25    DEC -25
M26    DEC -26
M28    DEC -28
M29    DEC -29
M72    DEC -72
M86    DEC -86
M75    DEC -75
M100   DEC -100
M125   DEC -125
M129   DEC -129
M256   DEC -256
M750   DEC -750
M1001  DEC -1001
```

```
*
*
* OCTAL CONSTANTS
*
*
B137   OCT 137
B177   OCT 177
B200   OCT 200
B337   OCT 337
B376   OCT 376
B377   OCT 377
B400   OCT 400
B700   OCT 700
B701   OCT 701
B1000  OCT 1000
B1200  OCT 1200
B1273  OCT 1273
B1600  OCT 1600
B1777  OCT 1777
B2000  OCT 2000
B2400  OCT 2400
B0700  OCT 070000
B1760  OCT 176000
*
*
*
D70    OCT -70
D72    OCT -72
D100   OCT -100
D133   OCT -133
D337   OCT -337
D340   OCT -340
D700   OCT -700
D701   OCT -701
```

```
*
*
* ALPHABETIC CONSTANTS
*
*
AY      OCT 101
BE      OCT 102
C       OCT 103
D       OCT 104
E       OCT 105
H       OCT 110
I       OCT 111
L       OCT 114
N       OCT 116
O       OCT 117
R       OCT 122
S       OCT 123
T       OCT 124
V       OCT 126
X       OCT 130
Y       OCT 131
```

```
*
*
* VARIABLES
*
*
ABSSF BSS 1        ABS/BSS PSEUDO OP FLAG
BADDR BSS 1        CURRENT BUFFER ADDRESS
CCNT  BSS 1        CHARACTER COUNT
COUNT BSS 1        HOLDS RECORD LENGTH
CUSTN BSS 1        CURRENT USER STATEMENT NUMBER
DMPFG BSS 1        DUMP FLAG
EDTFG BSS 1        EDIT FLAG
FIRST BSS 1        FIRST ENTRY IN SOURCE CODE BLOCK
FSTMT BSS 1        FIRST STATEMENT NUMBER
GRTFG BSS 1        FLAG SET DURING INTRODUCTARY TEXT
LBCNT BSS 1        LABEL COUNTER IN SYMBOL TABLE
NEXT  BSS 1        ADDR OF NEXT ENTRY IN SOURCE CODE
PREV  BSS 1        PREVIOUS ENTRY IN SOURCE CODE BLOCK
SAVA  BSS 1        STORAGE FOR (A)
SAVB  BSS 1        STORAGE FOR (B)
SAVEO BSS 1        STORAGE FOR (E) AND (O)
SEQFG BSS 1        SEQUENCE DIRECTIVE INDICATOR
SRCNT BSS 1        BUFFER LENGTH FOR CODE STORAGE
STINC BSS 1        STATEMENT NUMBER INCREMENT
TEMPI NOP          TTY INTERRUPT STORE
YDATA BSS 1        UPPER BOUND OF USER DATA AREA
ZDATA BSS 1        NEXT LOCATION IN DATA AREA
ZFRSP BSS 1        NEXT OPENING IN FREE SPACE
ZPLC  BSS 1        NEXT LOCATION FOR UNDEF PLC REFERENCE
ZUSRP BSS 1        NEXT LOCATION IN USER PROGRAM
*
```

```
*
ADDR1 BSS 1          ADDRESS IN SOURCE CODE BLOCK
ADDR2 BSS 1          TEMPORARY STORAGE VARIABLE
ADDR3 BSS 1
ASMBY BSS 1          SKELETON OF ASSEM INTRUCTION
ASMFG BSS 1          ASSEMBLY FLAG
DATPT BSS 1          DATA BUFFER POINTER
DPFLG BSS 1
EDINT BSS 1          EDIT INSTRUCTION TYPE
EFLG  BSS 1          EXPONENT E FLAG
EXP   BSS 1
EXPON BSS 1
INSNM BSS 1          INSTRUCTION NUMBER
LBLAD BSS 1          LABEL ADDRESS
LBLFG BSS 1          LABEL FLAG
LENTH BSS 1          LENGTH OF ASSEMBLY
LMTFG BSS 1          CONTROL IN SYM TBL SEARCH
LNTH2 BSS 1          DATA COUNTER
MANT1 BSS 1          MANTISSA TERMS, TEMPORARY STORAGE
MANT2 BSS 1
NUMFG BSS 1          OPERAND NUMBER FLAG
NUM1  BSS 1          HOLD NUMBERS, TEMPORARY STORAGE
NUM2  BSS 1
OPLBL BSS 1          OPERAND LABEL
OPNUM BSS 1          OPERAND INTEGER VALUE
SIGN  BSS 1          NUMBER SIGN
STORE BSS 1
TEMP  BSS 1
```

```
TEMP1 BSS 1
TEMP2 BSS 1
TEMP3 BSS 1
TEMP4 BSS 1          TEMPORARY STORAGE
TEMP5 BSS 1
TEMP6 BSS 1
TEMP7 BSS 1
ZADD  BSS 1          ADDRESS IN ASSEMBLED CODE
*
LAB1  DEF LABL1      ADDR OF LABEL BEGINNING STATEMENT
LAB2  DEF LABL2      ADDR OF OPERAND LABEL
LABL1 EQU TEMP5
LABL2 EQU TEMP
MNMNC DEF MNC        ADDRESS OF MNEMONIC BUFFER
*
ADDR  EQU EXP        TEMPORARY USED IN BUFFER STOREAGE
ENDFG EQU ADDR3
HOLDA EQU MANT1
HOLDB EQU MANT2
IDRCT EQU EFLG       INDIRECT BIT
LINK  EQU EXPON      LINK FOR COMPOUND OPERANDS
LWRBD EQU NUM1       LOWER BOUND IN MNEMONIC SEARCH
MNC   EQU MANT1      OP CODE BUFFER
MORG  EQU EXPON      MEMORY ORIGIN
OPADD EQU NUM1       OP CODE ADDRESS STORE
SORCE EQU STORE      SOURCE ADDRES OF DATA TO BE MOVED
STNUM EQU EFLG       STATEMENT NUMBER IN LIST OPERATION
UNDEF EQU SIGN       UNDEFINED POINTER
UPRBD EQU NUM2       UPPER BOUND IN MNEMONIC SEARCH
```

```
*
* EDIT VARIABLES
*
ASME0 BSS 1
ASME1 BSS 1        ASSEMBLY ADDRESSES
ASME2 BSS 1
DADR1 BSS 1        ASSEMBLY CODE ADDRESSES ON A
DADR2 BSS 1        MULTIPLE DELETE OPERATION
DLTLN BSS 1        DELETE LAST LINE
EDLMT BSS 1        STAT NUM LIMIT ON MULT INSERT
EDNUM BSS 1        INSTRUCTION NUMBER
EDTSV BSS 1        ADDRESS FOR MOVING CODE
ELNTH BSS 1        LENGTH OF DELETED CODE
ENEXT BSS 1        NEXT FREE AREA IN SCB BEFORE EDIT
ENM1  BSS 1        STATEMENT NUMBER
ENM2  BSS 1
EUSRP BSS 1        LOCATION STORE TO LINK EDIT
EXPEC BSS 1        INPUT EXPECTATION FLAG
MIIP  BSS 1        MULTIPLE INSERT IN PROGRESS
MCMIP BSS 1        MACHINE CODE MULT INSERT
SCBE0 BSS 1
SCBE1 BSS 1        SOURCE CODE BLCK ADDRESSES
SCBE2 BSS 1
VETO  BSS 1        VETO FLAG
*
AHEAD EQU ASME0    LOOK AHEAD POINTER IN SCB
BACK  EQU ASME1    LOKK BACK POINTER IN SCB
EDLX  EQU DADR1    SOURCE INPUT FLAG DURING EDIT
LKPSN EQU DADR2    LINK POSITION
LNTH3 EQU EDNUM    LENGTH OF ASSEMBLY
POSN  EQU ENEXT    POSITION OF SEACRCH IN SST
SSTAD EQU ENM1     SST ADDRESS
SUCAD EQU VETO     POINTER FOR LISTING PURPOSES
VALUE EQU SCBE0    TEMPORARY TO HOLD NUMBER IN OPERAND
```

```
*
* DISC INPUT DRIVER VARIABLES
*
DREAD OCT 020000   DISC READ COMMAND
SEEKX OCT 030000
TR202 OCT 145000   DISC ADDR OF LAST TRACK
*
LSTAC DEC 204      LAST TRACK ACCESSED
*
DSIPT OCT 14340    MEMORY ADDRESS FOR DISC INPUT
*
DCMND EQU EFLG     DISC ADDRESS
DOTA  EQU EXP      READ COMMAND
DSTAT EQU EXPON    DISC STATUS
HDMSK EQU INSNM    DISC HEAD MASK
MADDR EQU LBLFG    MEMORY ADDRESS FOR INPUT
*
*
* CHARACTER CONSTANTS
*
*
BLANK OCT 40       BLANK
COLON OCT 72       COLON  PRECEDES SYSTEM DIRECTIVES
COMMA OCT 54       COMMA
EQUAL OCT 75       EQUAL SIGN, UNIVERSAL ABORT
MINUS OCT 55       MINUS SIGN
PLUS  OCT 53       PLUS SIGN
PRIOD OCT 56       PERIOD
SLASH OCT 57       SLASH  PRECEDES EDIT DIRECTIVES
STAR  OCT 52       ASTERISK
*
* INPUT STORE BUFFERS
*
BUFA  DEF *+3      INPUT BUFFER
BUFB  DEF *+38     AUXILIARY INPUT BUFFER
DATBF DEF *+73     DATA STORE BUFFER
      BSS 100
      BSS 1        DATA OVERFLOW BUFFER
```

```
*
*
*
* OCTAL CONSTANTS
*
CH1   OCT 177400   FIRST CHARACTER POSITION
CLRTB OCT -12500   CLEAR TABLES
CPIB  OCT 102000   CURRENT PAGE  INDIRECT BIT
DMACW OCT 120011   DMA CONTROL WORD
IMODE OCT 160000   INPUT MODE FLAG FOR TTY
LMODE OCT 120000   OUTPUT FLAG ON TTY
MSIGN OCT 026400   MINUS SIGN FOR ASCII OUTPUT
JMP   OCT 026000   JUMP INSTRUCTION SKELETON
MSK4  OCT 77600
MNEG  OCT 100000   BIT 15 FOR INDIRECT REFERENCES
TENTH OCT 63146
YDAT  OCT 27300    UPPER LIMIT OF USER DATA AREA
XRTRN OCT 126340   EXECUTION RETURN
*
* INTERRUPT HALTS
*
HLT4  HLT 4,C      HALT ON A POWER FAIL
HLT5  HLT 5        PARITY ERROR / MEMORY PROTECT
*
MPPEX JMP MPPE,I   USER WARNING FOR FORWARD REFERENCES
*
* INTERRUPT SERVICE SUBROUTINE CALLS
*
DMAI  JSB DMASS
DCI   JSB DCSS
CCI   JSB CCSS
```

```
*
* CALL TO ERROR MESSAGE OUTPUT
*
*
ERCAL JSB ERROR    PRINT ERROR MESSAGE
      JMP CNTRL,I
*
*
* SUBROUTINE TO PRINT ERROR MESSAGES
*
*
*
ERROR NOP
      JSB BPLN
      JSB REENT    REQUEST RE-ENTRY
      JMP ERROR,I
*
*
* PRINT MESSAGE REQUESTING USER RE-ENTER STATEMENT AFTER ERROR
*
*
REENT NOP
      LDA .26      MESSAGE LENGTH
      LDB RENT
      JSB WRITE,I  PRINT MESSAGE
      JMP REENT,I
*
RENT  DEF *+1      MESSAGE TO REQUEST RE-ENTRY
      ASC 13,PLEASE RE-ENTER STATEMENT
*
*
* PRINT MESSAGE ON NEW LINE
*
*
BPLN  NOP
      STA HOLDA    PRESERVE POINTERS TO ERROR MESSAGE
      STB HOLDB
      JSB NWLN,I   OUTPUT CR-LF
      LDA HOLDA    RESTORE (A) AND (B)
      LDB HOLDB
      JSB WRITE,I  PRINT MESSAGE
      JMP BPLN,I
```

```
*
*
* BASE PAGE ERROR MESSAGES
*
*
ERR1   ASC 7,BAD DATA INPUT
ERR2   ASC 15 STATEMENT NUMBER OUT OF RANGE
ERR3   ASC 13,OPERAND VALUE OUT OF RANGE
ERR4   ASC 14,ILLEGAL OPERAND TERMINATION
ERR5   ASC 15,ILLEGAL CHARACTER BEGINS LABEL
ERR6   ASC 8,NO OPERAND FOUND
ERR7   ASC 10,OPERAND IS UNDEFINED
ERR8   ASC 13,UNDEFINED LABEL IN OPERAND
ERR9   ASC 7,NO LABEL FOUND
*
*
* PRINT MESSAGE ON TABLE OVERFLOW WITH RESTART INSTRUCTIONS
*
TBLOV JSB BPLN      NEW LINE ERROR MESSAGE
      LDA .24
      LDB *+4
      JSB WRITE,I
      HLT 55B
      JMP START
*
      DEF *+1
      ASC 12,PRESS RUN TO START AGAIN
*
*
* INTERRUPT SERVICE SUBROUTINES
*
*
* DMA INTERRUPT SERVICE ROUTINE
*
*
DMASS NOP
      CLC 6         CLEAR CONTROL AFTER DMA TRANSFER COMPLETE
      JMP DMASS,I
```

```
*
*
* DATA CHANNEL INTERRUPT
*
*
DCSS   NOP
       CLC DC       CLEAR CONTROL ON DATA CHANNEL
       JMP DCSS,I
*
*
* CONTROL CHANNEL INTERRUPT
*
*
CCSS   NOP
       CLC CC       CLEAR CONTROL ON CONTROL CHANNEL
       JMP CCSS,I
*
*
* CONFIGURE I/O SUBROUTINES
*
*
* ENTER (B) CHANNEL NUMBER OF I/O DEVICE
*
*
CNFIG  NOP
       LDA D72
       STA TEMP2
       LDA I.0      ADDR OF FIRST I/O INSTR
       STA TEMP1
CNFG1  LDA TEMP1,I  INSTRUCTION IN (A)
       STA TEMP3
       SSA,RSS      BIT 15 SET
       JMP CNFG2    NO
       AND B0700    MEMORY REFERENCE
       SZA
       JMP CNFG2    YES
       LDA TEMP3    RESTORE INSTRUCTION
```

```
      AND B2000     YES BIT 10 SET
      SSA
      JMP CNFG2     NO
      LDA TEMP3     YES, RETRIEVE INSTRUCTION
      JSB CNFG3
      STA TEMP1,I
CNFG2 ISZ TEMP1     ADVANCE ADDRESSES
      ISZ TEMP2
      JMP CNFG1
      LDA INT1,I
      JSB CNFG3     CHANGE ADDRESSES FOR STORING
      STA INT1,I    AND CLEARING INTERRUPT LOCATIONS
      LDA INT2,I
      JSB CNFG3
      STA INT2,I
      LDA INT3,I
      JSB CNFG3
      STA INT3,I
      LDA INT4,I
      JSB CNFG3
      STA INT4,I
      JMP CNFIG,I
*
*
* REMOVE CHANNEL NUMBER AND REPLACE WITH NEW ONE
*
*
CNFG3 NOP
      AND D100
      IOR B         ADD IN NEW VALUE
      JMP CNFG3,I
```

```
*
*
* GET NEXT CHARACTER FOM INPUT BUFFER
*
* RETURN P+1 ON EOL
*        P+2 CHARACTER IN A
*
*
GETCR NOP
      ISZ CCNT     ANY CHARACTERS LEFT
      RSS
      JMP GETCR,I  NO, END OF FILE EXIT
      JSB SAVEE    SAVE (E) REGISTER
      LDB BADDR    LOAD BUFFER ADDRESS
      ISZ BADDR    UPDATE FOR NEXT TIME
      CLE,ERB      SET CHARACTER FLAG
      LDA B,I      LOAD CURRENT BUFFER WORD
      SEZ,RSS      FIRST CHARACTER
      ALF,ALF      YES, POSITION IT
      AND B177     MASK EXTRANEOUS BITS
      JSB RSTRE    RESTORE (E) REGISTER
      ISZ GETCR    UPDATE RETURN ADDRESS
      JMP GETCR,I
*
*
* GET NEXT NON BLANK CHARACTER
*
* RETURN P+1 ON EOL
*        P+2 NON BLANK CHAR IN A
*
*
NTBLK NOP
NTBL1 JSB GETCR
      JMP NTBLK,I
      CPA BLANK    CHARACTER BLANK
      JMP NTBL1    YES, GET NEXT CHARACTER
      ISZ NTBLK
      JMP NTBLK,I  RETURN
```

```
*
*
* READ UP TO COMMA IN BUFFER
*
* RETURN P+1 NO COMMA FOUND
*        P+2 COMMA READ
*
*
RDCOM NOP
      JSB GETCR
      JMP RDCOM,I
      CPA COMMA
      RSS
      JMP *-4
      ISZ RDCOM
      JMP RDCOM,I
*
*
* BACKSPACE OVER ONE CHARACTER
*
*
BCKSP NOP
      JSB SAVEE     SAVE (E)
      CCA
      ADA CCNT      BACKSPACE OVER LAST
      STA CCNT      CHARACTER IN INPUT BUFFER
      CCA
      ADA BADDR
      STA BADDR
      JSB RSTRE     RESTORE (E)
      JMP BCKSP,I
```

```
*
*
* CHECK TERMINATOR OF INPUT STRING
*
* RETURN P+1 VALID TERMINATOR
*          P+2 NON TERMINAL CHARACTER
*
*
TRMCK NOP
      JSB GETCR
      RSS            END LF LINE
      CPA BLANK      BLANK CHARACTER
      JMP TRMCK,I    YES, RETURN VALID TERMINATOR
      ISZ TRMCK      NO, NON TERMINAL CHARACTER
      JMP TRMCK,I
*
*
* SAVE AND RESTORE CONTENTS OF (E)
*
*
SAVEE NOP
      ERB            SHIFT (E) INTO (B)
      STB ERROR      STORE (B)
      JMP SAVEE,I
RSTRE NOP
      LDB ERROR
      CLE,ELB        CLEAR THEN RESTORE (E)
      JMP RSTRE,I
```

```
*
*
* MOVE N WORDS FROM (A) TO (B)
*
* ENTER (A) = FWA OF ORIGIN
*       (B) = FWA OF DESTINATION
*
*
WMOVE NOP
      STA MORG     SET FWA OF ORIGIN
      LDA SORCE    WORD COUNT
      CMA,INA
      STA TEMP4
      LDA MORG,I
      STA B,I      STORE A WORD
      INB
      ISZ MORG     ADVANCE COUNTERS
      ISZ TEMP4
      JMP *-5
      ADB M1       REFERENCE LAST WORD MOVED
      JMP WMOVE,I
*
*
* DETERMINE DATA OR MACHINE INSTRUCTION ADDRESS
* AND MAKE CORRECTION FOR DATA ADDRESS
*
* ENTER (A) ADDRESS TO BE EXAMINED
*
* RETURN MACHINE CODE ADDRESS OR UPDATED DATA ADDRESS
*
*
DATAD NOP
      LDB XDATA    FIRST ADDRESS IN DATA AREA
      CMB,INB
      ADB A
      SSB
      JMP DATAD,I  MACHINE INSTRUCTION ADDRESS
      LDA A,I      DATA ADDRESS
      JMP DATAD,I  RETRIEVE ADDRESS REFERENCE
```

```
*
*
* MASK ON INDIRECT BIT IF REQUESTED
*
* ENTER (A) INSTRUCTION OR ADDRESS
*
*
IDIRT NOP
      LDB IDRCT     INDIRECT FLAG
      SZB
      IOR MNEG      MASK ON BIT 15
      JMP IDIRT,I
*
*
* SAVE REGISTER CONTENTS AFTER EXECUTION
*
*
SAVR  NOP
      STA SAVA      SAVE (A)
      STB SAVB      SAVE (B)
      ERA,ALS       SHIFT (E) INTO (A), CLEAR BIT 0
      SOC
      INA           SET BIT 0 IF OVERFLOW SET
      STA SAVEO     SAVE (E) AND (O)
      JMP SAVR,I
*
*
* PREPARE ADDRESS POINTERS FOR EDIT OPERATION
*
*
EDTAD NOP
      LDA ZUSRP     NEXT FREE AREA IN PROGRAM
      STA EUSRP     SAVE FOR EDIT LINK PURPOSES
      ADA .2        ADVANCE FOR EDIT ENTRIES
      STA ZUSRP
      JSB STCK,I    CHECK FOR PROGRAM AREA OVERFLOW
      JMP EDTAD,I
```

```
*
*
* PREPARE SOME POINTERS FOR SCAN OF SOURCE CODE TEXT
*
*
* ENTER (B) SCB ADDRESS OF INSTRUCTION TO BE DELETED
*
* RETURN (A) ASSEMBLY FLAG, ADDRESS OF ASSEMBLY OF
*                INSTRUCTION TO BE DELETED
*
*
PREPR NOP
      ADB .3          ADDR OF LENGTH
      LDA B,I
      AND B177        NUMBER OF WORDS IN SCB ENTRY
      STA CNFG3
      LDA B,I
      ALF,ALF
      AND B177        NUMBER OF CHARACTERS
      CMA             CONTROL VARIABLE USED IN GETTING
      STA CCNT        NEXT CHARACTER FROM BUFFER
      INB
      LDA B,I         ASSEM FLAG, ADDR OF ASSEMBLY
      JMP PREPR,I
```

```
*
*
* CLEAR UP LINKAGE IN SOURCE CODE BLOCK ON A DELETE
* OPERATION
*
*
DSCB  NOP
      LDA SCBE0
      SZA           DELETE FIRST LINE
      JMP DSCB2     NO
      LDB DLTLN     YES, DELETE LAST LINE
      SZB,RSS
      JMP DSCB1     NO
      CCA           YES, DELETE WHOLE PROGRAM
      STA PREV
      LDA NEXT      NEXT AREA IN SCB WILL BE ADDR
      STA FIRST     OF FIRST STATEMENT IN SCB
      JMP DSCB,I
*
DSCB1 LDA SCBE2     DELETE FIRST LINE
      STA FIRST
      CCB           SET TERMINATOR IN PREVIOUS
      LDA SCBE2
      INA
      STB A,I
      JMP DSCB,I
DSCB2 LDA DLTLN     DELETE LAST LINE
      SZA,RSS
      JMP DSCB3     NO
      LDB SCBE0     ADDR OF INSTR BEFORE DELETION
      STB PREV      RESET POSITION OF LAST INSTR
      JMP DSCB,I    BEFORE EDIT
DSCB3 LDA SCBE0
      LDB SCBE2
      STB SCBE0,I   STORE SUCC ADDR IN PREV INSTR
      INB
      STA B,I       SET PREV ADDR IN SUCC INSTR
      JMP DSCB,I
```

```
*
*
* SINGLE MACHINE CODE INSTRUCTION DELETE
*
* DELETE A MACHINE CODE INSTR OF LENGTH ONE WORD
*
*
SNGDL NOP            SINGLE DELETE
      JSB SVPSN      SAVE NEXT LOCATION IN PROGRAM
      JSB XDEL       MOVE CODE AFTER DELETED CODE
      JSB CMVE,I
*
* PLACE JUMPS TO LINK PROGRAM AND EDIT ENTRIES
*
      JSB JMPBF
*
      JSB JMPAF
      JMP SNGDL,I
*
*
* FIND NEXT MACHINE CODE INSTRUCTION IN ASSEMBLED
* PROGRAM TO FINISH DELETE OPERATION
*
*
XDEL  NOP
      LDB SCBE2
      RSS
XDEL1 LDB B,I        ADDR OF NEXT ENTRY IN SCB
      CPB ENEXT      END OF SOURCE CODE BLOCK
      JMP XDEL2      YES
      ADB .4         ADD OF ASSEM ADDR, ASSEM FLAG
      LDA B,I
      ADB M4         RESTORE SCB ADDRESS
      SSA            ASSEMBLY
      JMP XDEL1      DATA
      SZA,RSS
      JMP XDEL1      COMMENT
      STA ASME2
      JMP XDEL,I
```

```
*
* DELETE LAST LINE
*
XDEL2 LDA ASME1     ADDR OF DELETED WORD SNGL DLTE
      LDB EDNUM     EDIT INSTRUCTION NUMBER
      CPB .2        MULTIPLE DELETE
      LDA DADR1     YES, ADDR OF SINGLE DELETE
      STA EUSRP
      JMP CNTRL,I
*
*
* SAVE POSITION IN USER PROGRAM AREA FORPOSITIONING
* LINK INSTRUCTIONS AFTER AN EDIT OPERATION
*
*
SVPSN NOP           SAVE POSITION
      LDB ZUSRP     NEXT LOCATION IN PROGRAM
      STB EDTSV     SAVE POSITION
      JMP SVPSN,I
*
*
* INSERT A SINGLE JUMP DURING EDIT
*
* ENTER (A) ADDRESS WHERE JUMP RESULTS
*       (B) ADDRESS WHERE JUMP ORIGINATES
*
*
JMPE1 NOP
      AND B1777     GET RELATIVE ADDRESS
      ADA JMP
      STA B,I       STORE JUMP
      JMP JMPE1,I
```

```
*
*
* PLACE JUMP AFTER EDIT ENTRY
*
*
JMPAF NOP               JUMP AFTER
      LDB ZUSRP         NEXT AREA IN USER PROGRAM
      LDA ASME2
      INA
      JSB JMPS          PLACE JUMPS
      STB ZUSRP
      JSB STCK,I        PROGRAM AREA OVERFLOW
      JMP JMPAF,I
*
*
* PLACE JUMPS TO CONNECT MAIN USER PROGRAM WITH
* BEGINNING OF EDIT ENTRY
*
*
JMPBF NOP               JUMP BEFORE
      LDB ASME1         ADDR WHERE JUMP ORIGINATES
      LDA EDTSV         ADDR WHERE JUMP RESULTS
      JSB JMPS          PLACE TWO JUMP INSTRUCTIONS
      JMP JMPBF,I
*
*
* STORE JUMPS TO LINK EDITTED CODE
*
* ENTER (A) ADDRESS WHERE JUMP RESULTS
*       (B) ADDRESS WHERE JUMP ORIGINATES
*
*
JMPS  NOP
      CLE
      AND B1777         GET ADDRESS
      ADA JMP           ADD IN JMP INSTRUCTION SKELETON
      STA B,I           STORE
      INA               ADVANCE POINTERS TO INCLUDE
      SEZ,CME,INB,RSS             SECOND JUMP
      JMP *-5
      JMP JMPS,I
```

```
*
*
* SUBROUTINE DISKI CONTROLS INPUT FROM THE DISC.  IT ADDS
* THE DIRECTION BIT (BIT 15=1) TO THE CORE ADDRESS AND HAS
* AN ERROR RECOVERY PROCEDURE IF READ PARITY OR DECODE
* ERRORS ARE DETECTED.  FOLLOWING DETECTION OF SUCH AN
* ERROR, 9 ADDITIONAL ATTEMPTS WILL BE MADE.  IF THESE FAIL
* THE DISC ADDRESS AND THE DISC STATUS ARE DESPLAYED IN (A)
* AND (B) THE THE COMPUTER HALTS BY PRESSING RUN, 10
* ADDITIONAL READS WILL BE ATTEMPTED.
*
* ENTER (A) DISC ADDRESS
*       (B) CORE ADDRESS
*
*
DISKI NOP
      ADB MNEG     DIRECTION FOR READ
      STA DCMND    SAVE DISC ADDRESS
      STB MADDR    MEMORY ADDRESS
DISK1 LDA M10      DISC READ ERROR COUNT
      STA TEMP     ERROR COUNTER
DISK2 JSB DISKD    INPUT FROM DISC
      JMP DISKI,I  RETURN
      ISZ TEMP     ADVANCE COUNTER
      JMP DISK2    TRY AGAIN
      LDA DCMND    DISC ADDRESS
      LDB DSTAT    DISC STATUS
      HLT 22B
      JSB RSEEK
      JMP DISK1    TRY AGAIN 10 MORE TIMES
```

```
*
*
* SUBROUTINE DISKD IS THE DISC INPUT DRIVER.  IT SETS UP THE
* MEMORY ADDRESS REGISTER, THE WORD COUNT REGISTER, AND THE
* DISC ADDRESS.  FOLLOWING THESE IT INITIATES THE TRANSFER,
* AND WAITS UNTIL THE TRANSFER IS COMPLETE (BY CHECKING THE
* DISC STATUS WORD).  READ PARITY AND DECODE ERRORS WILL BE
* TESTED.
*
* RETURN P+1 SUCCESSFUL READ
*        P+2 ERROR IN READ
*
*
DISKD NOP
      LDB MADDR     CORE ADDRESS
      CLC 2         PREPARE TO SET MEMORY ADDR REG
      OTB 2         SET MEM ADDRESS IN MAR
      STC 2         PREPARE TO SET WORD COUNT REGISTER
      LDB LENTH     NEGATIVE WORD COUNT
      OTB 2         SET WORD COUNT IN WCR
      LDA DCMND     DISC ADDRESS
      JSB SEEK
      JMP DSKD1
      LDA DOTA      DISC READ COMMAND
      OTA CC        OUTPUT TO COMMAND CHANNEL
      STC DC,C      SET CONTROL ON DATA CHANNEL
      CLC CC
      STC 6,C       INITIATE DMA
      STC CC,C      INITIATE DMA TRANSFER
      SFS CC        WAIT FOR
      JMP *-1       COMPLETE TRANSFER
      JSB STAT      CHECK STATUS
DSKD1 ISZ DISKD
      JMP DISKD,I
```

```
*
*
* OUTPUT SEEK COMMAND ALONG WITH TRACK AND SECTOR NUMBER TO
* THE DISC
*
* ENTER (A) DISC ADDRESS
*           BITS 0- 8 SECTOR NUMBER
*           BITS 8-15 TRACK NUMBER
*
* RETURN P+1 STATUS ERROR
*        P+2 DISC READY, INITIATE DATA TRANSFER
*
SEEK  NOP
      ALF,ALF      ROTATE TRACK NUMBER TO LOW BITS
      AND B377     ISOLATE TRACK NUMBER
      OTA DC       OUTPUT TRACK NUMBER
      STC DC,C     TO DATA CHANNEL
      LDB SEEKX    SEEK COMMAND
      CPA LSTAC    CURRENT TRACK = LAST TRACK ACCESSED
      ADA MNEG     YES, ALTER TO ADDRESS COMMAND
      STA LSTAC    UPDATE LAST TRACK ACCESSED
      CLC CC
      OTB CC       OUTPUT SEEK ADDR COMMAND
      STC CC,C     TO COMMAND CHANNEL
      LDA DREAD    READ COMMAND
      STA DOTA     SAVE READ COMMAND
      LDA DCMND    DISC ADDRESS
      AND B377     ISOLATE SECTOR
*
* COMPUTE PHYSICAL HEAD/SECTOR FROM LOGICAL SECTOR
* NUMBER AND HEAD MASK
*
      CLB,RSS
      INB
      ADA M12
      SSA,RSS
      JMP *-3
      ADA .12      12 SECOTRS PER TRACK
      BLF,BLF
```

```
      ADB HDMSK
      IOR B
      SFS DC       WAIT FOR DMA TO ACCEPT TRACK NUMBER
      JMP *-1
      OTA DC       OUTPUT HEAD SECTOR
      STC DC,C     TO DATA CHANNEL
      SFS CC       WAIT FOR SEEK COMPLETION
      JMP *-1
      JSB STAT     CHECK STATUS
      RSS
      ISZ SEEK
      JMP SEEK,I
*
*
*
* OUTPUT SEEK COMMANDS TO FIRST AND LAST TRACKS ON DISC
* FOLLOWING 10 UNSUCCESSFUL READ ATTEMPTS.
*
*
RSEEK NOP
      CLA
      JSB SEEK
      NOP
      LDA TR202
      JSB SEEK
      NOP
      JMP RSEEK,I
```

```
*
*
*
* CHECK DISC STATUS BEFORE AND AFTER DATA TRANSFER
* (CHECK FOR COMPLETION WITH DISC STATUS WORD)
*
* RETURN P+1 DISC STAUS ERROR
*        P+2 SUCCESSFUL STATUS CHECK
*
*
STAT  NOP
      STF 6
      LIB 2
      STC DC,C
      CLA          DRIVE NUMBER, STATUS CHECK
      CLC CC
      OTA CC       OUTPUT STATUS COMMAND
      STC CC,C     TO COMMAND CHANNEL
      SFS DC       WAIT UNTIL DATA
      JMP *-1      CHANNEL CLEAR
      CLC CC       CLEAR COMMAND CHANNEL
      LIA DC       LOAD STAUS FROM DATA CHANNEL
      STA DSTAT    DISC STATUS
      CLE,SLA,RSS  ERROR
      JMP STAT1    NO
      RAL,ARS
      SSA,RSS      FIRST SEEK
      RAR,SLA,RAR  DATA ERROR
```

```
      JMP STAT1+1   YES
      RAL,SLA
      JMP STAT2     FALG, CYLINDER ERROR
      JMP STAT3     DISC NOT READY
STAT1 ISZ STAT
      JMP STAT,I
*
STAT2 RAL,SLA
      JMP *-2
*
* WRITE ERROR   ABNORMAL HALT
*
      HLT 24B
      JMP DISK1     BEGIN READ AGAIN
*
STAT3 JSB NWLN,I
      LDA .14
      LDB STATR     DISC NOT READ MESSAGE
      JSB WRITE,I
      HLT 26B
      JMP DISK1     BEGIN READ CYCLE AGAIN
*
STATR DEF *+1
      ASC 7,DISC NOT READ
*
```

```
*
*
      ORG 2000B
*
* SYSTEM CONTROLLER
*
*
* THE SYSTEM CONTROLLER DIRECTS THE PROGRAM IN ANY ONE
* OF EIGHT DIRECTIONS DEPENDING ON THE FIRST CHARACTER
* OF THE USER RESPONSE AND/OR FIVE SYSTEM VARIABLES.
*
* ALL INPUT OPERATIONS WILL BE HANDLED WITHIN THE SYSTEM
* CONTROLLER WITH THE EXCEPTION OF:
*
*        USER RESPONSES WHEN PRINTING INTRODUCTARY TEXT
*
*        USER RESPONSE TO AN EDIT VETO OPERATION.
*
*
* THERE ARE UP TO SEVEN DIFFERENT TESTS TO DIRECT USER
* ENTRIES TO THE APPROPRIATE PROGRAM LOGIC.
*
* ONE:      ANY RESPONSE BEGINNING WITH AN EQUAL SIGN IS
*           INTERPRETED AS A REQUEST TO ABORT THE PROGRAM.
*
*           RETRUN TO THE INITIALIZATION ROUTINE IF AN
*           EQUAL SIGN BEGINS THE RESPONSE
*
*
* TESTS TWO TO SIX INVOLVE EXAMINING SYSTEM VARIABLES TO
* BE SET (=-1) TO TRANSFER PROGRAM CONTROL.
*
*
* TWO:      ABS/BSS FLAG (ABSSF)
*
*           RETURN TO ABS/BSS ROUTINE FOLLOWING USER
*           RESPONSE TO PROMPT FOR TEMPORARY DEFINITION
*           OF UNDEFINED ABS OR BSS OPERAND.
*
```

```
*
* THREE:   DUMP FLAG (DMPFG)
*
*          RETURN TO DUMP ROUTINE WITH USER RESPONSE
*          EITHER TO END THE DUMP OPERATION OR DUMP
*          DATA ADDRESS CONTENTS.
*
*
* FOUR:    SEQUENCE FLAG (SEQFG)
*
*          RETURN TO SEQUENCE ROUTINE WITH STATEMENT
*          NUMBER DATA.
*
*
* FIVE:    EDIT SOURCE CODE INPUT FLAG (EDLX)
*
*          RETURN WITH SOURCE INPUT DURING EDIT OPERATION.
*
*
* SIX:     EDIT FLAG (EDTFG)
*
*          RETURN TO MAIN EDITOR ROUTINE TO INTERPRET
*          AND EXECUTE EDIT REQUEST.
*
* SEVEN:   A COLON BEGINNING A USER ENTRY SIGNALS A
*          SYSTEM DIRECTIVE.  AFTER RECOGNIZING A COLON
*          BRANCH TO THE ROUTINE TO INTERPRET AND
*          CHANNEL SYSTEM DIRECTIVES.
*
*
* FAILURE TO SATISFY ANY ONE OF THESE TESTS RESULTS IN
* THE ASSEMBLER TREATING THE INPUT AS A SOURCE PROGRAM
* STATEMENT
* THE CODING WILL FALL THROUGH TO THE MAIN LEXICAL
* ROUTINE
*
*
```

```
CMAND JSB DATIN    READ INPUT, FIRST CHAR IN (A)
      CPA EQUAL    ABORT
      JMP START    YES
      JSB I.ON     NO, ENABLE INTERRUPT
      LDB ABSSF    ABS/BSS FLAG
      SZB
      JMP ABSSR,I  RETURN TO LEXICAL ROUTINE
      JSB CLER,I   CLEAR LEXICAL POINTERS
      LDB DMPFG    DUMP FLAG
      SZB
      JMP DMP2     RETURN TO DUMP ROUTINE
      LDB SEQFG    SEQUENCE FLAG
      SZB
      JMP SEQ      RETURN TO SEQUENCE ROUTINE
      LDB EDLX     SOURCE INPUT DURING EDT
      SZB
      JMP EDLEX,I  RETURN TO EDIT INPUT CONTROL
      LDB EDTFG    EDIT FLAG, EDIT INSTRUCTION
      SZB
      JMP EDTR,I   PROCESS EDIT COMMAND
      CPA COLON    COLON PRECEDES SYSTEM COMMANDS
      JMP SYSTM
*
* ENTRY POINT TO MAIN PROGRAM AFTER INITIALIZATION
*
LXSCN JSB LEXI,I   LEXICAL ANALYSIS
      JSB ASSM,I   PREPARE FOR STORAGE
      LDA ASMFG    COMMENT STATEMENT
      SZA          YES
      JSB STCD,I   STORE CODE
*
* RETURN AFTER COMPLETION OF AN EDIT OPERATION
* INVOLVING PROGRAM INPUT
*
EDRTN JSB STSCB    STORE IN SOURCE CODE BLOCK
      JSB LBDEF    DEFINE LABEL IF PRESENT
      LDB MIIP     MULTIPLE INSERT
      SZB,RSS
      JMP CMAND
      JMP MIRTI,I  RETURN TO MULTIPLE INSERT
```

```
*
*
* SUBROUTINE TO REQUEST INPUT AND CALL INPUT ROUTINE
*
*
DATIN NOP
      JSB CRLFD    OUTPUT CR/LF
      LDA M2
      LDB RDSYM    OUTPUT DATA REQUEST PROMPT
      JSB TTY.P
      JSB I.OFF    TURN OFF TTY
      LDA .72      BUFFER LENGTH
      LDB BUFA     BUFFER ADDRESS
      JSB TTY.I    READ
      STA SRCNT    LENGTH FOR SOURCE CODE RETENTION
      CMA,SSA,RSS  CHECK FOR BUFFER OVERFLOW
      JMP DAT1     RECORD TOO LONG
      STA CCNT     RETAIN NUMBER OF CHARACTERS
      LDA BUFA
      CLE,ELA      SHIFT BUFFER ADDRESS LEFT
      STA BADDR    ODD/EVEN WORD
      JSB GETCR    RETURN CHARACTER IN (A)
      JMP DATIN+1  REQUEST RE-ENTRY
      JMP DATIN,I
*
RDSYM DEF *+1
      OCT 40007    INPUT PROMPT
*
* MESSAGE ON BUFFER OVERFLOW
*
DAT1  LDA .16
      LDB DAT2
      JSB ERROR
      JMP DATIN+1
*
DAT2  DEF *+1
      ASC 8,BUFFER OVERFLOW
```

```
*
*
* INPUTS FROM TELETYPE OR CRT SCREEN
*
*   (A) = MAXIMUM NUMBER OF CHARACTERS IN RECORD
*   (B) = BUFFER STARTING ADDRESS
*
*
* RETURN (A) = NUMBER OF CHARACTERS IN RECORD
*            = -1 ON BUFFER OVERFLOW
*
*
* THE CHARACTERS ARE PACKED TWO TO A WORD IN THE BUFFER.
*
* ALL RECORDS MUST BE TERMINATED WITH A LINE FEED.
* THE NULL AND CARRIAGE RETURN CHARACTERS ARE IGNORED.
*
* THE LEFT ARROW(S) DELETE THE PREVIOUS CHARACTER(S).
*
*
TTY.I NOP
      STA COUNT     SAVE LENGTH
      STB BADDR     SET BUFFER ADDRESS
      CLB           SET CHARACTER COUNTER
      LDA IMODE
TI.1  OTA TTY       SET TTY TO INPUT MODE
TI.2  STC TTY,C     REQUEST CHARACTER
      SFS TTY
      JMP *-1       WAIT FOR CHARACTER INPUT
      LIA TTY       LOAD CHARACTER
      JSB PROCS     PROCESS CHARACTER
      JMP TI.2      GET NEXT CHARACTER
      CLC TTY
      JMP TTY.I,I   RECORD COMPLETE RETURN
```

```
*
*
*
*
* OUTPUT ASCII RECORS THROUGH THE TELETYPE PRINTER
*
*   (A) = NUMBER OF CHARACTERS TO BE OUTPUT
*   (B) = STARTING ADDRESS OF BUFFER
*
*
* IF (A) >= 0 THEN OUTPUT (A) CHARACTERS
* FOLLOWED BY A CARRIAGE RETURN LINE FEED.
*
*
*  IF (A) < 0 THEN PRINT -(A) CHARACTERS ONLY.
*
* BUFFER CONTAINS ASCII CHARACTERS PACKED TWO PER WORD.
*
* IF (A) = 0 ON ENTRY ONLY A CR/LF IS OUTPUT.
*
*
TTY.P NOP
      JSB INIT       INITIALIZE AND SET UP
      LDA TTY        SAVE TTY INTERRUPT INSTR
      STA TEMPI
      LDA LMODE
      OTA TTY        SET TTY TO OUTPUT MODE
      STB TTY        PUT NOP INTO INTERRUPT CELL
TP.3  JSB GETCH      GET NEXT CHARACTER
      JMP TP.8       BUFFER EMPTY
      OTA TTY        LOAD TTY BOARD BUFFER
      STC TTY,C      GIVE PRINT COMMAND
      SFS TTY        WAIT FOR FLAG
      JMP *-1
      LDA TEMPI
      SZA,RSS        IS INTERRUPT ENABLED
      JMP TP.3       NO
```

```
*
* THIS SECTION CHECKS IF A CHARACTER HAS BEEN TYPED FROM THE
* KEYBOARD DURING OUTPUT ON TELETYPE.
*
      LIA TTY      LOAD FROM BOARD BUFFER
      CMA          FIRST 8 BITS SHOULD BE ONES
      AND B177
      SZA,RSS
      JMP TP.3     NO KEY STRUCK, CONTINUE
      LDA IORI
      STA FINSH+1  RESTORE IOR INSTRUCTION
      CLC TTY      TURN OFF TTY
      JSB I.STP    GO TO STOP
TP.8  CLC TTY      TURN OFF TTY
      LDA TEMPI
      CPA TT.II    IS INTERRUPT MODE SET
      JSB I.ON     YES, RE-ENABLE KEYBOARD
      LDA IORI
      STA FINSH+1  RESTORE IOR INSTRUCTION
      SEZ,CLE,RSS  RECORD COMPLETE  CLEAR E
      JMP TTY.P,I  (E) = 0   RECORD OUTPUT COMPLETE
      LDA M2       (E) = 1  ADD A RETURN AND LINE FEED
      LDB CRLFA    LOAD ADDRESS OF CR AND LF
      JMP TTY.P+1  DO CR/LF
*
*
* THIS ROUTINE TURNS OFF THE TELETYPE INTERRUPT MODE
*
I.OFF NOP
      CLA
      STA TTY      SET NOP INTO INTERRUPT CELL
      CLC TTY      TURN OFF READ MODE
      JMP I.OFF,I  RETURN
*
* THIS ROUTINE TURNS ON THE TELETYPE INTERRUPT MODE
*
I.ON  NOP
      LDB TT.II
      STB TTY      SET JSB INTO INTERRUPT CELL
      LDB IMODE
      OTB TTY      SET TTY TO INPUT MODE
      STC TTY,C    SET TTY TO LOOK FOR INPUT
      JMP I.ON,I
```

```
*
IT.II JSB ISTP,I   INTERRUPT LOCATION CODE
*
*
*
* CHARACTER PROCESSING SECTION FOR TTY
*
*
* ENTER (A) HOLDS CHARACTER
*
* RETURN P+1 GET NEXT CHARACTER
*        P+2 RECORD COMPLETE
*
*
PROCS NOP
      AND B177     STRIP BIT 7
      SZA,RSS      NULL
      JMP PROCS,I  YES, IGNORE
      CPA LNFD     NO, LINEFEED
      JMP PROCS,I  YES, IGNORE
      CPA CRTN     NO, CARRIAGE RETURN
      JMP CMPLT    YES, COMPLETE RECORD
      CPA B177
      JMP TI.2
      CPB COUNT    NO, BUFFER OVERFLOW
      CCB          YES, LOOK FOR CARRIAGE RETURN
      SSB          LOOKING FOR CARRIAGE RETURN
      JMP PROCS,I  YES, RETURN
      CPA LFTAR    NO, LEFT ARROW
      JMP DLETE    YES, DELETE PREVIOUS CHARACTER
      SLB,INB      NO, CHECK ODD/EVEN FLAG
      JMP PROC2    B0 = 0, EVEN CHARACTER
PROC1 ALF,ALF      B0 = 1,ODD CHARACTER
      STA BADDR,I
      JMP PROCS,I  RECORD HIGH CHARACTER AND RETURN
PROC2 IOR BADDR,I  PACK TWO CHARACTERS
      STA BADDR,I  PUT IN BUFFER
      ISZ BADDR    INDEX BUFFER ADDRESS POINTER
      JMP PROCS,I
```

```
*
* THIS SECTION DELETES PREVIOUS CHARACTER(S)
*
DLETE SZB,RSS      IS BUFFER EMPTY
      JMP PROCS,I  YES, RETURN
      CCA          NO
      ADB A        DECREMENT CHARACTER COUNT
      SLB,RSS      LOW CHARACTER
      JMP PROCS,I  YES
      ADA BADDR    NO DECREMENT ADDRESS POINTER
      STA BADDR
      LDA BADDR,I  GET LAST TWO CHARACTERS
      ALF,ALF
      AND B177     BELETE LAST CHARACTER
      JMP PROC1    STORE NEXT-TO-LAST CHARACTER
*
* THIS SECTION PUTS COUNT IN A AND RETURNS TO P+2
*
CMPLT LDA B        PUT CHARACTER COUNT IN (A)
      ISZ PROCS
      JMP PROCS,I
*
*
*
* SUBROUTINE GETCH
*
* RETURN P+1 BUFFER EMPTY
*        P+2 CHARACTER IN (A)
*
*
GETCH NOP
      CPB COUNT
      JMP GETCH,I  BIFFER EMPTY, P+1 RETURN
      LDA BADDR,I  GET TWO CHARACTERS
      SLB,RSS
      ALF,ALF      (B) EVEN, POSITION CHAR RIGHT
      SLB,INB      CHECK O/E, AND INDEX COUNT
      ISZ BADDR    (B) ODD, INCREMENT ADDR POINTER
FINSH AND B177     STRIP LEFT CHARACTER
      IOR B200     ADD BIT 7
      ISZ GETCH
      JMP GETCH,I  RETURN ON P+2
```

```
*
*
* INITIALIZES FOR OUTPUTTING A RECORD
*
*
INIT   NOP
       CCE,SSA      SET (E) = 1, CHECK FOR (A) < 0
       CMA,CLE,INA  SET (E) = 0, (A) =-(A)
       STA COUNT    SAVE CHARACTER COUNT
       STB BADDR    SET BUFFER STARTING ADDRESS
       CLB          INITIALIZE OUTPUT COUNT
       JMP INIT,I
*
*
*
LNFD   EQU .10      LINE FEED
CRTN   EQU .13      CARRIAGE RETURN
LFTAR  EQU B137
CLAI   EQU B2400
IORI   IOR B200     ADD IN BIT 7
*
CRLFA  DEF CRLF
CRLF   OCT 106612
*
*
*
*
* STOP COMMAND SERVICE
*
*
I.STP  NOP
       JSB I.OFF    TURN OFF KEYBOARD INTERRUPT
       JSB CRLFD    NEW LINE
       LDA .4
       LDB STOPA    PRINT STOP
       JSB TTY.P
       JMP CMAND
*
STOPA  DEF *+1
       ASC 2,STOP
*
```

```
*
*
* TO OUTPUT MULTIPLE CR-LF
*
* ENTRY -(A) CONTAINS THE NUMBER OF CR-LF TO BE OUTPUT
*
*
NWLNS NOP
      STA TEMP
      JSB CRLFD
      ISZ TEMP
      JMP *-2
      JMP NWLNS,I
*
*
* SUBROUTINE TO OUTPUT CARRIAGE RETURN - LINE FEED
*
*
CRLFD NOP
      CLA
      JSB TTY.P    OUTPUT CR-LF
      JMP CRLFD,I
*
*
* CONVERT BINARY TO ASCII OCTAL OR DECIMAL
*
* ENTER (A) = VALUE TO BE CONVERTED
*
* RETURN (A) CONTAINS LEAST TWO SIGNIFICANT DIGITS
*        (B) POINTS TO ADDRESS OF MOST SIGNIFICANT DIGITS
*
*
CNDEC NOP          BINARY TO DECIMAL ASCII
      LDB M10
      JSB CNBIN
      JMP CNDEC,I
*
*
CNOCT NOP          BINARY TO OCTAL ASCII
      LDB M8
      JSB CNBIN
      JMP CNOCT,I
```

```
*
*
CNBIN NOP
      STB TEMP5
      LDB A00
      STB TEMP
      STB TEMP1
      STB TEMP2
      LDB CNMBR
      STB TEMP3
CNBN1 JSB DVUKN     DIVIDE BY 8 OR 10
      ADB TEMP3,I
      STB TEMP3,I
      SZA,RSS
      JMP CNBN2
      JSB DVUKN     DIVIDE BY 8 OR 10
      BLF,BLF
      ADB TEMP3,I
      STB TEMP3,I
      ISZ TEMP3
      SZA
      JMP CNBN1
CNBN2 LDA TEMP
      LDB TEMP2
      STB TEMP      SWAP FOR OUTPUT PURPOSES
      STA TEMP2
      LDB CNMBR
      JMP CNBIN,I
*
*
A00   ASC 1,00
CNMBR DEF TEMP
```

```
*
*
DVUKN NOP
      CLB                CLEAR LOOP COUNTER = QUOTIENT + 1
      STB TEMP4
DVUK1 STA B
DVUK2 ADA TEMP5          DIVIDE BY SUCCESSIVE SUBTRACTION
      ISZ TEMP4
      SSA,RSS            DONE IF (A) IS NEG AND (B) IS POS
      JMP DVUK1          CLEAR (B) TO ALLOW EXIT
      SSB                EIXIT IF POSITIVE
      JMP DVUK2          ORIG NUMBER TO CONVERT WAS NEG
      LDB TEMP5          DONE
      CMB,INB
      ADB A              REMAINDER TO (B)
      LDA TEMP4
      ADA M1
      JMP DVUKN,I
```

```
*
*
*
*
* SET SOURCE CODE BLOCK ENTRIES
*
*
* BESIDES STORING THE STATEMENT ENTRY, SIX WORDS HOLDING
* NECESSARY INFORMATION ABOUT THE STATEMENT MUST BE SET.
*
* THE FORMAT FOR THESE SIX WORDS IS:
*
* WORD 1   ADDRESS OF NEXT STATEMENT ENTRY IN SCB
*
* WORD 2   ADDRESS OF PREVIOUS STATEMENT ENTRY
*          (-1 FOR THE FIRST STATEMENT)
*
* WORD 3   STATEMENT NUMBER
*
* WORD 4   BITS 0- 7 NUMBER OF WORDS IN SCB ENTRY
*          BITS 8-15 NUMBER OF CHARACTERS IN SOURCE INPUT
*
* WORD 5   BITS 0-14 ADDRESS OF ASSEMBLY
*                    (0 FOR A COMMENT STATEMENT)
*          BIT   15 0 MACHINE CODE INSTRUCTION
*                   1 DATA DEFINITION
*
* WORD 6   LENGTH OF ASSEMBLY
*
*
* THE USER SOURCE STATEMENTS WILL BE STORED TWO CHARS
* PER WORD BEGINNING IN THE FIRST CHARACTER POSITION
* (BITS 8-15) OF THE FIRST WORD TO FOLLOW WORD 6 IN THE
* SOURCE CODE BLOCK TABLE.
*
*
```

```
STSCB NOP
      LDA EDTFG    EDIT OPERATION
      SZA
      JMP SCB1     YES
      LDB ADDR1    NO, ADDRESS OF ENTRY IN SCB
      STB PREV,I
      LDA NEXT     SUCCESSOR ADDRESS
      STA B,I      STORE SUCCESSOR ADDRESS
      INB
      LDA PREV     ADDRESS OF PREVIOUS INSTRUCTION
      STA B,I
      LDA CUSTN    PREVIOUS STATEMENT NUMBER
      ADA STINC    STATEMENT NUMBER INCREMENT
      STA CUSTN    CURRENT USER STATEMENT NUMBER
      INB
      STA B,I      STORE STATEMENT NUMBER
      LDB ADDR1
      STB PREV     ADDR OF PREVIOUS FOR NEXT ENTRY
SCB1  LDB ADDR1
      ADB .3
      LDA LNTH2    WORD HOLDING LENGTHS
      STA B,I
      INB
      LDA ASMFG    ASSEMBLY FLAG
      SZA,RSS
      JMP SCB2     COMMENT STATEMENT
      CLE,ELA      STORE ASSEMBLY INFORMATION IN (E)
      LDA ZADD     ADDRESS OF ASSEMBLY
      RAL,ERA      ASSEMBLY INFORMATION IN BIT 15
SCB2  STA B,I
      INB
      LDA LENTH    LENGTH OF ASSEMBLY
      STA B,I
      INB
      LDA SRCNT    NUMBER OF WORDS IN SOURCE INPUT
      STA STORE
      LDA BUFA     INPUT BUFFER ADDRESS
      JSB WMOVE    MOVE INTO SCB
      JMP STSCB,I
```

```
*
*
* DEFINE LABEL PRECEDING MNEMONIC
*
*
LBDEF NOP
      LDB LBLFG    LABEL FLAG
      SZB,RSS      LABEL PRESENT
      JMP LBDEF,I  NO, RETURN
      LDB LBLAD    LABEL ADDRESS
      LDA B,I      PREVIOUS REFERENCE TO LABEL
      SZA
      JMP LBDF1    YES
      LDA ZADD     NO
      CLE
      JSB SLBL,I   STORE LABEL IN SYMBOL TABLE
      JMP LBDEF,I
*
```

```
LBDF1 CLA
      STA IDRCT     DIRECT REFERENCE
      ADB .3
      STB RSTRE
      LDA B,I       UNDEFINED DIRECT REFERENCE
      ADA D700
      SSA           FORWARD REFERENCE
      JSB FWDRF     YES, CLEAR UP ALL DIRECT REFS
      ISZ RSTRE
      LDA RSTRE,I   LOOK FOR INDIRECT FWD REFS
      STA IDRCT     SET INDIRECT POINTER
      ADA D700
      SSA           FORWARD REFERENCES
      JSB FWDRF     YES
      LDB LBLAD     LABEL ADDRESS
      ADB .2
      LDA B,I       LABEL INFORMATION
      AND CH1       SAVE LAST CHARACTER OF LABEL
      ADA .1        DEFINED LABEL
      STA B,I
      INB
      LDA ZADD      ADDRESS IN ASSEMBLED CODE
      STA B,I       STORE WITH LABEL
      INB           ADVANCE ADDRESS
      LDA ADDR1     ADDRESS IN SOURCE CODE
      STA B,I
      JMP LBDEF,I
```

```
*
*
* INTERPRET AND CHANNEL SYSTEM DIRECTIVES
*
*
* THERE ARE SEVEN SYSTEM DIRECTIVES WHICH MAY BE ENTERED
* ANY TIME EXCEPT, DURING AN EDIT.  THESE DIRECTIVES
* WITH THE EXCEPTION OF THE HALT ARE PRESENTED TO THE
* USER IN THE INTRODUCTARY TEXT.
*
*
*       :ABORT      DISCONTINUE PROGRAM ENTRY,START AGAIN
*       :DUMP       DUMP REGISTER CONTENTS
*       :EDIT       EDIT THE EXISTING PROGRAM
*       :HALT       HALT THE COMPUTER
*                   PRESS RUN TO START AGAIN
*       :LIST       LIST ALL OR PART OF USER PROGRAM
*       :SEQUENCE   CHANGE THE SEQUENCING
*                   THEN LIST THE PROGRAM
*       :XECUTE     EXECUTE USER PROGRAM
*
*
* THE COLON FLAGGING DIRECTIVES HAS BEEN RECOGNIZED
*
*
SYSTM JSB NTBLK    NEXT NON BLANK CHARACTER
      JMP SYST5    NO CHARACTER FOUND
      CPA AY       ABORT DIRECTIVE
      JMP START    YES, BEGIN PROGRAM ONCE AGAIN
*
*
*       :D(UMP)
*
* WILL DISPLAY THE REGISTERS AS OCTAL AND DECIMAL
* VALUES.  INSTRUCTIONS WILL ALSO BE PRESENTED TO
* DISPLAY DATA DDRESS CONTENTS
*
*
      CPA D        NO, DUMP DIRECTIVE
      JMP DUMP     YES
```

```
*
*
*        :E(DIT)
*
* PREPARE SOME POINTERS AND PROMPT USER TO BEGIN
*
*
      CPA E        NO, EDIT DIRECTIVE
      CCB,RSS      YES
      JMP SYST1    NO
      STB EDTFG    SET EDIT FLAG
      JSB EDTAD    SET ADDRESS POINTERS
      LDA NEXT
      STA ENEXT
      ISZ NEXT     ADVANCE FOR TEST ADDR FOR INSERTS
      LDA M8
      JSB NWLNS    OUTPUT 8 CR-LF
      LDA .20
      LDB EDMSG
      JSB TTY.P    PRINT EDIT PROMPT
      LDA M2
      JSB NWLNS
      JMP CMAND    RETURN TO CONTROLLER
*
EDMSG DEF *+1
      ASC 10,BEGIN EDIT OPERATION
*
*
*        :H(ALT)
*
* STOP THE COMPUTER
*
*
SYST1 CPA H        HALT DIRECTIVE
      RSS          YES
      JMP *+3      NO
      HLT 77B      YES, STOP
      JMP CMAND    PRESS RUN TO CONTINUE
```

```
*
*
*        :L(IST)(,M(,N))
*
* LIST THE PROGRAM SEQUENTIALLY STATEMENT BY STATEMENT
*
* M AND N, IF PRESENT SPECIFY THE FIRST AND LAST STATEMENT
*          TO BE LISTED.. IF N IS ABSENT THEN ALL STATEMENTS
*          FROM M ON ARE LISTED.. IF NEITHER APPEAR THE
*          WHOLE PROGRAM IS LISTED.
*
*          IF N < M LISTING IS SUPPRESSED
*
*
      CPA L        LIST DIRECTIVE
      RSS          YES
      JMP SYST4    NO
      JSB RDCOM    READ UP TO COMMA
      JMP SYST3    LIST WHOLE PROGRAM
      JSB TWNT,I   READ STATEMENT NUMBERS
      JMP SYST2    ONE NUMBER FOUND
      RSS          ERROR
      RSS          TWO NUMBERS FOUND
      JMP SYST6    ERROR
      LDB NUM1
```

```
      CMB,INB
      ADB NUM2     FIRST NUMBER GREATER THAN SECOND
      SSB
      JMP CMAND    YES, IGNORE LIST INSTRUCTION
      LDA FSTMT    CHECK RANGE OF SECOND STATEMENT
      CMA,INA        NUMBER
      ADA NUM2
      SSA          TOO SMALL
      JMP SYST7    YES
      JMP *+3
SYST2 LDA CUSTN
      STA NUM2     SET TERMINATOR
      LDA NUM1
      CMA,INA
      ADA CUSTN    CHECK RANGE OF FIRST NUMBER
      SSA          IN RANGE
      JMP SYST7    NO, TOO BIG
      JMP *+5
SYST3 LDA FSTMT    SET PARAMETERS FOR COMPLETE LISTING
      STA NUM1
      LDA CUSTN
      STA NUM2
      LDA M10      OUTPUT 10 CR-LF
      JSB NWLNS
      JSB LIST     CALL LIST ROUTINE
      LDA M10
      JSB NWLNS
      JMP CMAND    RETURN TO CONTROLLER
```

```
*
*
*        :S(EQUENCE),M,N
*
* CHANGE PROGRAM SEQUENCING SUCH THAT
*
*        M BECOMES THE FIRST STATEMENT NUMBER
*
*        N IS THE THE INCREMENT FOR SUCCESSIVE STATEMENTS
*
* RESTRICTIONS ON M AND N ARE THAN M MUST NOT EXCEED 1000
*              AND N MUST NOT EXCEED 25.
*
*
SYST4 CPA S            SEQUENCE DIRECTIVE
      JMP SEQ          RESEQUENCE SOURCE CODE BLOCK
*
*
*        :X(ECUTE)
*
* WILL INITIATE THE EXECUTION OF THE USER PROGRAM.
*
*
      CPA X            NO, EXECUTE DIRECTIVE
      JMP XEQI         YES
*
*
* ERROR MESSAGES
*
*
SYST5 LDA .22
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 11,UNDEFINED INSTRUCTION
*
```

```
*
SYST6 LDA .14
      LDB *+2
      JMP ERCAL
*
       DEF ERR1    BAD DATA INPUT
*
*
SYST7 LDA .30
      LDB *+2
      JMP ERCAL
*
      DEF ERR2     STATEMENT NUMBERS OUT OF RANGE
*
*
SEQ   JSB SQNC,I
      CCA,RSS      STATEMENT NUMBER INPUT ERROR
      JMP SEQ1
*
* SET SEQUENCE FLAG
* RETURN TO SYSTEM CONTROLLER FOR INPUT
*
      STA SEQFG    SET SEQUENCE FLAG
      JMP CMAND
*
SEQ1  CLA
      STA SEQFG    CLEAR SEQUENCE FLAG
      LDB FIRST    ADDRESS OF FIRST STATEMENT
SEQ2  LDA B,I      ADDRESS OF NEXT STATEMENT
      STA AHEAD    SAVE ADDRESS
      ADB .2       ADDRESS OF STATEMENT NUMBER
      LDA CUSTN    CURRENT STATEMENT NUMBER
      ADA STINC    ADD INCREMENT
      STA CUSTN
      STA B,I
      LDB AHEAD
      CPB NEXT     TERMINATION
      JMP SYST3    LIST THE PROGRAM
      JMP SEQ2     NO, CONTINUE
```

```
*
*
* DUMP REGISTER CONTENTS AND DATA ADDRESSES
*
*
DUMP  LDA SAVA
      LDB DUMP1
      JSB RGDP1    DUMP (A)
      LDA SAVB
      LDB DUMP1+1
      JSB RGDP1    DUMP (B)
      LDA SAVEO
      LDB DUMP1+2
      JSB EODMP    DUMP (E)
      LDA SAVEO
      RAR          POSITION FOR (O)
      LDB DUMP1+3  ADDRESS OF REGISTER NAME
      JSB EODMP    DUMP (O)
*
DMP1  LDA M2
      JSB NWLNS
      LDA .16
      LDB RGDM4    RETURN INFORMATION
      JSB TTY.P
      LDA .46
      LDB RGDM5    OPERAND DUMP INSTRUCTION
      JSB TTY.P
*
* SET FLAG, JUMP TO SYSTEM CONTROLLER
*
      CCB
      STB DMPFG    SET DUMP FLAG
      JMP CMAND    READ RESPONSE
*
* RETURN POINT FROM SYSTEM CONTROLLER
*
DMP2  CPA D        OPERAND DUMP DIRECTIVE
      JMP *+4      YES
      CLB          NO
      STB DMPFG    CLEAR FLAG
      JMP CMAND    RETURN TO CONTROLLER
```

```
*
* DUMP DATA ADDRESS CONTENTS
*
      JSB RDCOM     READ UP TO COMMA
      JMP DPER1     NO COMMA, ERROR IN INSTRUCTION
      JSB LBCK,I    READ IN AND CHECK LABEL
      JMP DPER2     NO OPERAND LABEL
      JMP DPER3     LABEL IS UNDEFINED
      JSB DTRG,I    CHECK LABEL RANGE
      ADA OPNUM     OPERAND NUMBER IF PRESENT
      JSB DTRG,I    CHECK RANGE
      LDB ZDATA     NEXT FREE DATA AREA
      CMB,INB
      ADB A         ADD IN DATA ADDRESS BEING SOUGHT
      SSB,RSS
      JMP DPER4     ADDRESS UNDEFINED
      LDB A,I       CONTENTS OF ADDRESS
      LDA B,I       VALUE
      STA TEMP7
      JSB CRLFD
      LDB RGDM3
      LDA M5
      STA TEMP1
      INB           CHANGE MESSAGE ADDRESS
      ISZ TEMP1     TO IGNORE BLANKS
      JMP *-2
      LDA M9
      JSB TTY.P
      JSB ASCDC     BINARY TO ASCII DECIMAL
      LDA M1
      LDB RGDM3
      JSB TTY.P
      LDA M8
      LDB RGDM2
      JSB TTY.P
      LDA TEMP7
      JSB CNOCT     BINARY TO ASCII OCTAL
      LDA .6
      JSB TTY.P     OUTPUT OCTAL
      JMP DMP1      PRINT PROMPT
```

```
*
*
DUMP1 DEF AY
      DEF BE         ADDRESS OF REGISTER NAMES
      DEF E
      DEF O
DUMP2 DEF TEMP7
*
*
* PREPARE TO DUMP EITHER (E) OR (O)
*
* ENTER (A) REGISTER STATUS IN BIT 15
*       (B) ADDRES  OF REGISTER NAME
*
*
EODMP NOP
      CLE,ELA        MOVE BIT FLAG IN TO (E)
      LDA .48        ASCII ZERO
      SEZ
      INA
      ALF,ALF        SHIFT TO FIRST CHARACTER POSITION
      JSB RGDP2      DUMP REGISTER
      JMP EODMP,I
```

```
*
*
* DUMP (A) OR (B)
*
* ENTER (A) VALUE TO BE DUMPED
*       (B) ADDRESS OF REGISTER NAME
*
*
RGDP1  NOP
       JSB RGDP3     PRINT REGISTER NAME
       LDA M10
       LDB RGDM1
       JSB TTY.P
       LDA M8
       LDB RGDM2
       JSB TTY.P
       LDA TEMP7
       JSB CNOCT     BINARY TO OCTAL ASCII
       LDA .6
       JSB TTY.P     PRINT OCTAL VALUE
       LDA M19
       LDB RGDM3
       JSB TTY.P
       JSB ASCDC     CONVERT BINARY TO ASCII DECIMAL
       JMP RGDP1,I
*
*
* DUMP (E) OR (O)
*
*
RGDP2  NOP
       JSB RGDP3     PRINT REGISTER NAME
       LDA M10
       LDB RGDM1
       JSB TTY.P
       LDA .1
       LDB DUMP2     PRINT REGISTER
       JSB TTY.P
       JMP RGDP2,I
```

```
*
*
* PRINT REGISTER NAME
*
*
RGDP3 NOP
      STA TEMP7     VALUE TO BE CONVERTED
      STB TEMP6     ADDRESS OF REGISTER NAME
      LDA M2        TWO NEW LINES
      JSB NWLNS
      LDA M2
      LDB TEMP6
      JSB TTY.P     PRINT REGISTER NAME
      JMP RGDP3,I
*
*
* CONVERT BINARY TO ASCII DECIMAL WITH MINUS SIGN
* PRECEDING VALUE (WHEN NEEDED) AND PRINT VALUE
*
*
ASCDC NOP
      LDA TEMP7     VALUE TO BE CONVERTED
      SSA,RSS       NEGATIVE VALUE
      JMP ASCD1     NO
      CMA,INA       CONVERT TO POSITIVE INTEGER
      JSB CNDEC     BINARY TO ASCII DECIMAL
      LDA B,I
      AND B177      SAVE MOST SIGNIFICANT CHAR
      IOR MSIGN     INCLUDE MINUS SIGN
      STA B,I
      RSS
ASCD1 JSB CNDEC     CONVERT POSITIVE NUMBER TO ASCII
      LDA .6
      JSB TTY.P     PRINT DECIMAL VALUE
      JMP ASCDC,I
*
```

```
*
*
RGDM1 DEF *+1
      ASC 5, REGISTER
*
*
RGDM2 DEF *+1
      ASC 4,OCTAL
*
*
RGDM3 DEF *+1
      ASC 10,           DECIMAL
*
*
RGDM4 DEF *+1
      ASC 8,TYPE R TO RETURN
*
*
RGDM5 DEF *+1
      ASC 23,ELSE TYPE D, FOLLOWED BY OPERAND TO BE DUMPED
*
* DUMP ERROR MESSAGES
*
DPER1 LDA .16
      LDB *+2
      JMP ERCAL
*
      DEF ERR6     NO OPERAND FOUND
```

```
*
*
DPER2 LDA .14
      LDB *+2
      JMP ERCAL
*
      DEF ERR9     NO LABEL FOUND
*
*
DPER3 LDA .26
      LDB *+2
      JMP ERCAL
*
      DEF ERR8     UNDEFINED LABEL IN OPERAND
*
*
DPER4 LDA .20
      LDB *+2
      JMP ERCAL
*
      DEF ERR7     OPERAND IS UNDEFINED
```

```
*
*
* EXECUTE USER PROGRAM
*
*
XEQI   JSB PLCDF     DEFINE PLC REFERENCES
       JSB SSTDF     DEFINE SST ENTRIES
       JSB SCNCD,I   SCAN CODE FOR FORWARD REFERENCES
       JSB PROG,I    EXECUTE USER PROGRAM
       JSB SAVR      SAVE REGISTER CONTENTS
*
* RESTORE FORWARD REFERENCES TO USER PROGRAM
*
XEQ1   LDA XUSRP     FIRST LOCATION IN USER PROGRAM
       STA TEMP
       LDB BUFA      ADDRESS OF UNDEFINED REFERENCES
       STB TEMP1
XEQ2   LDB TEMP1,I
       CPB ZERO      ALL UNDEF REF RETURNED TO PROGRAM
       JMP XEQ4      YES
XEQ3   LDA TEMP,I    NO
       CPA MPPEX     SPECIAL TERM TO INTERRUPT EXECUTION
       JMP *+3       YES
       ISZ TEMP      NO, NEXT LOCATION IN PROGRAM
       JMP XEQ3
       STB TEMP,I    RETURN FORWARD REFERENCE TO
       ISZ TEMP      USER PROGRAM
       ISZ TEMP1
       JMP XEQ2
XEQ4   JMP CMAND     RETURN TO CONTROLLER
```

```
*
*
*
*
*
* UNDEFINED (FORWARD REFERENCE) WARNING
*
*
MPPET JSB SAVR      SAVE REGISTER CONTENTS
*
* PREVENT INTERRUPT BEFORE PROGRAM IS RESTORED
*
      JSB I.OFF     TURN OFF INTERRUPT
      LDA .28       MEMORY PROTECT ERROR
      LDB MPT1
      JSB BPLN      PRINT EXPLANATION OF ERROR
      LDA .40       TO USER
      LDB MPT2
      JSB TTY.P
      JMP XEQ1
*
*
MPT1  DEF *+1
      ASC 14,UNDEFINED OPERAND IN PROGRAM
*
*
MPT2  DEF *+1
      ASC 20,EXECUTION CEASES, CONTINUE PROGRAM ENTRY
```

```
*
*
* DEFINE COMPOUND OPERAND REFERENCES
*
*
SSTDF NOP
      LDA XSTBL    ADDRRSS OF SYMBOL TABLE
      JMP *+3
SST1  LDA RSTRE    RETRIEVE ADDRESS
      ADA .6
      STA RSTRE    SAVE PRESENT POSITION IN SYM TBL
      LDB YSTBL    UPPER BOUND OF SYMBOL TABLE
      CMB,INB
      ADB A
      SSB,RSS      WHOLE TABLE SCANNNED
      JMP SSTDF,I  YES, RETURN
      ADA .2       EXAMINE LABEL INFORMATION
      LDB A,I
      CLE,ERB
      SEZ,RSS      LABEL DEFINED
      JMP SST1     NO
      ADA .2       YES
      LDB A,I      ADDRESS IN SOURCE CODE BLOCK
      STB ADDR1    SAVE ADDRESS
      ADA .1
      STA LKPSN    SAVE LINK POSITION
      LDB A,I      LINK TO SST
      SZB,RSS      SST ENTRIES
```

```
       JMP SST1       NO, EXAMINE NEXT AREA IN SYM TBL
SST2   STB SSTAD      YES, SAVE ADDRESS IN SST
       LDA B,I
       STA OPNUM
       INB
       LDA B,I        LINK BACK TO SYMBOL TABLE
       SSA            INDIRECT BIT SET
       CCA,RSS        YES
       CLA            NO
       STA IDRCT      SET INDICATION
       STB POSN       SAVE PRESENT POSITION IN SST
       LDA OPNUM      OPERAND NUMBER VALUE
       LDB ADDR1      SCB ADDRRSS OF LABEL
       JSB FNDAD      FIND ADDRESS
       SSA            ADDRRSS FOUND
       JMP SST4       NO
       STA ZADD       YES, SAVE ADDRRSS
       ISZ POSN       NEXT LOCATION IN SST
       LDA POSN,I
       ADA D700       FORWARD REFERENCES
       SSA
       JSB FWDRF      YES, CLEAR UP FWD REFS
       ISZ POSN       ADDRESS OF LINK IN SST
       LDB POSN,I     VALUE OF LINK
       STB LKPSN,I    STORE IN NEW LOCATION
       LDA SSTAD      ADDRRSS OF ENTRY IN SST
       STA TEMP       SAVE ADDRESS
       LDA M4
```

```
      STA TEMP1
      CLA
      STA TEMP,I   CLEAR ENTRY IN SST
      ISZ TEMP
      ISZ TEMP1    ADVANCE ADDRESS POINTER
      JMP *-3
      LDA M2
      ADA TEMP
      STA TEMP     ADDRESS OF FORWARD REFERENCE
      LDA XSST     BASE ADDRESS OF SST
      CMA,INA
      ADA SSTAD
      ARS,ARS
     ADA B1273     RESTORE FORWARD REF
      STA TEMP,I
SST3  SZB          LINK TO FURTHER ENTRIES
      JMP SST2     YES
      JMP SST1     NO, EXAMINE NEXT LABEL
*
*
* ADDRESS NOT FOUND FOR SST ENTRY
*
SST4  LDA POSN
      ADA .2       POSITION OF LINK
      STA LKPSN    NEW LINK ADDRESS POINTER
      LDB A,I      EXAMINE LINK
      JMP SST3
```

```
*
*
* DEFINE PLC REFERENCES BEFORE BEGINNING EXECUTION
*
*
* EACH PLC REFERENCE IS STORED IN TWO WORDS IN THE PLC
* TABLE
*
* WORD 1    SCB ADDRESS WITH BIT 15 SET FOR INDIRECT
*           REFERENCE
*
* WORD 2    NUMERIC VALUE IN OPERAND
*
* NO ATTEMPT WILL BE MADE TO DEFINE THE PLC REFERENCE
* UNTIL EXECUTION.  BEFORE EXECUTION THE PLC TABLE
* WILL BE SCANNED AND ALL POSSIBLE REFERENCES WILL BE
* DEFINED.  THE SPACE OCCUPIED BY THE ADDRESS WILL BE
* CLEARED TO ZERO.
*
* A WARNING IS PRESENTED IF THE PLC TABLE IS NEARLY FULL
* THE EXISTING USER PROGRAM IS LOST IF THE TABLE IS
* ALLOWED TO OVERFLOW.
*
*
```

```
PLCDF NOP
      LDB XPLC      BASE ADDRESS OF PLC TABLE
      JMP *+3
PLC1  LDB STORE
      ADB .2        NEXT AREA IN PLC TABLE
      STB STORE     SAVE ADDRESS
      LDA YPLC      UPPER BOUND OF TABLE
      CMA,INA
      ADA B
      SSA,RSS       TABLE FULLY SCANNED
      JMP PLCDF,I   YES, RETURN
      LDA B,I       ENTRY
      SZA,RSS
      JMP PLC1      NO, LOOK AT NEXT AREA IN PLC TABLE
      SSA           YES, INDIRECT REFERENCES
      CCA,RSS       YES
      CLA           NO
      STA IDRCT     SET POINTER
      LDA B,I       RESTORE ADDRESS
      ELA,CLE,ERA   CLEAR BIT 15
      STA ADDR1     SAVE ADDRESS
      INB
      LDA B,I       OPERAND NUMBER VALUE
      LDB ADDR1     SCB ADDRESS
      JSB FNDAD     FIND ADDRESS
      SSA           ADDRRSS FOUND
      JMP PLC1      NO
      STA ZADD      YES, SAVE ADDRESS
```

```
      LDA ADDR1     SCB ADDRESS
      ADA .4        ADDRESS OF ASSEMBLY
      LDB A,I
      STB ADDR1     SAVE ADDRESS
      ADB D340      CORRESPONDING ADDRRSS IN
      STB ADDR2     ADDRESS BLOCK
      LDA ZADD
      JSB DATAD     DETERMINE ADDRESS TYPE
      JSB IDIRT     CHECK FOR INDIRECT REFERENCE
      STA ADDR2,I   STORE ADDRESS
      LDB ADDR1,I   INSTRUCTION SKELETON
      SSB           TWO WORD ASSEMBLY
      JMP PLC3      YES
      LDA ADDR2
      AND B1777     GET RELATIVE ADDRESS
      IOR CPIB      CURRENT PAGE INDIRECT BIT
      SWP           STORE ADDRESS IN (B)
      AND B1760     SAVE INSTRUCTION SKELETON
      ADA B
PLC2  STA ADDR1,I   RETURN INSTRUCTION
      CLA
      STA STORE,I   CLEAR AREA IN PLC TABLE STORE
      JMP PLC1
*
* TWO WORD ASSEMBLY
*
PLC3  ISZ ADDR1
      LDA ADDR2     POSITION IN ADDRESS BLOCK
      IOR MNEG      INDIRECT BIT
      JMP PLC2
```

```
*
*
* FIND ADDRESS FOR COMPOUND OPERAND
*
* ENTER (A) OPERAND NUMBER VALUE
*       (B) SOURCE CODE BLOCK ADDRESS OF LABEL
*
*
* RETURN (A) = -1 ADDRESS NOT FOUND
*             ADDRESS IN ASSEMBLED CODE
*
*
FNDAD NOP
      STA VALUE
      SSA               DETERMINE DIRECTION OF SEARCH
      JMP FNDD5
FNDD1 LDA B,I           SEARCH AHEAD
      STA AHEAD         ADDRESS OF NEXT ENTRY
      ADB .5
      LDA B,I           LENGTH OF ASSEMBLY
      STA LNTH3         SAVE LENGTH OF ASSEMBLY
      CMA,INA
      ADA VALUE
      SSA
      JMP FNDD3
      CLE,SZA,RSS
      CCE
      STA VALUE         RETAIN NEW VALUE
      LDB AHEAD         ADDRESS OF NEXT ENTRY
      CPB NEXT          TERMINATION
FNDD2 CCA,RSS           YES
      RSS
      JMP FNDAD,I       RETURN ADDRESS NOT FOUND
      SEZ,RSS
      JMP FNDD1
      INB               BACK UP IN SCAN OF SCB
      LDB B,I           TO RETRIEVE ADDRESS OF ASSEMBLY
      ADB .5            FOR PREVIOUS INSTRUCTION
```

```
*
* ADDRESS FOUND
*
FNDD3 ADA LNTH3    LENGTH OF ASSEMBLY
FNDD4 ADB M1       ADDRESS IN ASSEMBLED CODE
      LDB B,I
      ELB,CLE,ERB  CLEAR BIT 15 IF NECESSARY
      ADA B        RETAIN ADDRESS IN (A)
      JMP FNDAD,I
*
* SEARCH BACKWARD IN SOURCE CODE BLOCK
*
FNDD5 CMA,INA
      STA VALUE    CONVERT CONSTANT TO POSITIVE
FNDD6 INB
      LDB B,I      ADDRESS OF PREVIOUS INSTRUCTION
      CPB M1       TERMINATION
      JMP FNDD2    YES
      INB
      LDA B,I      RETAIN ADDRESS OF PREVIOUS INSTR
      STA BACK
      ADB .4
      LDA B,I      LENGTH OF ASSEMBLY
      CMA,INA      SUBTRACT LENGTH OF ASSEMBLY
      ADA VALUE    FROM CONSTANT
      SSA
      JMP FNDD7    POSITION FOUND
      SZA,RSS
      JMP FNDD7
      STA VALUE    SAVE NEW VALUE
      LDB BACK     ADDR OF PREV IN SCB ENTRIES
      CPB M1       TERMINATOR
      JMP FNDD2    YES, ADDRESS NOT FOUND
      JMP FNDD6+4  NO
*
FNDD7 CMA,INA
      JMP FNDD4
```

```
*
*
* DEFINE FORWARD REFERENCES
*
*
FWDRF NOP
FWDR1 ADA B700        ADDRESS OF FIRST REFERENCE
      ADA JMP         ACTUAL ADDRESS
      STA WMOVE
      LDA WMOVE,I     RETTRIEVE INSTRUCTION
      AND B1777       RELATIVE ADDR OF NEXT INSTR
      STA SAVEE       RETAIN ADDRESS
      LDA WMOVE,I     RETRIEVE INSTRUCTION
      AND B1760       REMOVE POINTER TO NEXT REFERENCE
      STA WMOVE,I
      AND B0700       LENGTH OF ASSEMBLY
      SZA
      CLB,INB,RSS     ONE WORD ASSEMBLY
      CCB             TWO WORD ASSEMBLY
      STB LENTH
      LDA WMOVE       ADDRESS OF INSTRUCTION
      ADA D340        ADDR IN ADDR BLOCK
      STA ADDR3
      LDA ZADD        ASSEMBLY ADDRESS
      JSB DATAD       UPDATE DATA ADDRESS
      JSB IDIRT
      STA ADDR3,I     STORE ADDRESS
      LDA ADDR3
      LDB LENTH       GET ADDRESS IN ADDRESS BLOCK
      SSB,RSS         FOR INSTRUCTION
      AND B1777       REL ADDR FOR 1 WORD ASSEMBLY
      STA TEMP
      LDA WMOVE,I     RETRIEVE UNDEF INSTRUCTION
      ADA TEMP        ADDRESS
      IOR CPIB        CURRENT PAGE INDIRECT BIT
      STA WMOVE,I     STORE INSTRUCTION
      LDA SAVEE       POINTER TO NEXT INSTRUCTION
      ADA D700        RETURN TO SYMBOL TABLE
      SSA
      JMP FWDR1       NO, FORWARD REFERENCE
      JMP FWDRF,I
```

```
*
*
* SYSTEM LIST ROUTINE
*
* ENTER (A) > 0 SYSTEM DIRECTIVE
*       (A) < 0 CALL FROM EDITOR
*
*
LIST   NOP
       STA ENDFG       PRINT FOR END MESSAGE
       CLE,SSA         CLEAR PRINT FLAG
       JMP LST1        CALL FROM EDIT
       LDA FIRST       FIRST ENTRY IN SCB
       RSS
LST1   LDA SUCAD       GET PREVIOUS SUCCESSOR ADDRESS
       CPA NEXT        TERMINATION
       JMP LST3        YES
       LDB A,I         ADDRESS OF NEXT STATEMENT
       SZB,RSS         END OF USER PROGRAM
       JMP LST3
       STB SUCAD       NO, SAVE NEXT ADDRESS
       ADA .2
       STA ADDR
       LDA A,I         GET STATEMENT NUMBER
       STA STNUM
       SEZ             PRINT FLAG
       JMP LST2        SET
       LDB NUM1
       CMB,INB
       ADB STNUM       CHECK RANGE
       CLE,SSB
       JMP LST1        GET NEXT STATEMENT
LST2   CMA,INA         -STATEMENT NUMBER
```

```
      ADA NUM2
      SSA          IN RANGE
      JMP LST3     NO
      CCE          YES, SET PRINT FLAG
      LDA STNUM    STATEMENT NUMBER
      JSB CNDEC    BINARY TO ASCII DECIMAL
      LDA M6
      JSB TTY.P    PRINT STATEMENT NUMBER
      LDA M1
      LDB RGDM3    ADDRESS OF 1 BLANK
      JSB TTY.P
      LDB ADDR
      INB
      LDA B,I      LENGTH
      ALF,ALF
      AND B177     NUMBER OF CHARS IN SOURCE INPUT
      ADB .3       ADDRESS OF SOURCE LINE
      JSB TTY.P    PRINT LINE, AND CR-LF
      JMP LST1     GET NEXT LINE
*
LST3  LDB ENDFG    END MESSAGE
      SSB
      JMP LIST,I   NO, RETIRN
      LDA M2
      JSB NWLNS
      LDA .12
      LDB LSTMG
      JSB TTY.P    PRINT -LIST ENDS-
      JMP LIST,I
*
*
LSTMG DEF *+1
      ASC 6, *LIST ENDS*
```

```
*
*
      ORG 4000B
*
*
* MAIN LEXICAL SUBROUTINE TO SCAN INPUT SOURCE CODE
* ALONG WITH CODE INVOLVED IN EDIT OPERATIONS
*
*
* THE INSTRUCTION SET HAS BEEN DIVIDED INTO 15 GROUPS
* DEPENDING UPON THE INSTRUCTION TYPE AND THE OPERAND
* REQUIRED
*
* GROUP    INSTRUCTION TYPE          OPERAND REQUIRED
* NUMBER
*
*  1                             NO OPERAND REQUIRED
*
*  2      INPUT / OUTPUT         CLEAR FLAG MAY BE PRESENT
*
*  3      INPUT / OUTPUT         CHANNEL NUMBER EXPECTED
*
*  4      INPUT / OUTPUT         CHANNEL NUMBER EXPECTED
*                                CLEAR FLAG MAY BE SPECIFIED
*
*  5      EXTENDED ARITH         NUMBER OF SHIFTS
*         REGISTER REFERENCE
*
*  6      MEMORY REFERENCE       LABEL
*                                NUMBER
*  7      EXTENDED ARITH         ASTERISK
*         MEMORY REFERENCE       INDIRECT FLAG
```

```
*
*
*          PSEUDO OP
*
*  8          END             NO OPERAND REQUIRED
*
*  9          ASC             LENGTH AND LIST OF ASCII
*                             DATA
*
* 10          DEC             REAL OR DECIMAL INTEGER
*
* 11          OCT             OCTAL INTEGER VALUES
*
* 12          EQU             ADDRESS
*
* 13          ABS             ADDRESS VALUE
*
* 14          BSS             VALUE
*
* 15          DEF             ADDRESS DEFINITION
*
*
* EXCEPT FOR THE MEMORY REFERENCE INSTRUCTIONS ALL
* OPERAND RECOGNITION AND HANDLING WILL BE WITHIN THE
* LEXICAL SUBROUTINES.  MEMORY REFERENCE OPERANDS WILL
* BE EXAMINED BUT NOT PROCESSED UNTIL THE INSTRUCTION
* IS ABOUT TO BE STORED.
*
*
*
```

```
LEX    NOP
       LDB EDINT     REPLACE OR DELETE
       SLB,RSS
       JMP *+3       NO
       JSB GETCR     GET FIRST CHARACTER
       JMP LXR1      FIRST CHARACTER NOT FOUND
       CPA STAR      WHOLE LINE A COMMENT
       JMP LEX,I     YES, RETURN
       CPA BLANK     BLANK, NO LABEL
       JMP LEX2      YES
       JSB LTPR,I    NO, LETTER OR PERIOD
       JMP LXR2      NO, ILLEGAL FIRST CHARCTER
       LDB LAB1      YES, ADDRESS FOR FIRST LABEL
       JSB LBRD,I    READ LABEL
       JMP LXR9      ILLEGAL CHARACTER BEGINS LABEL
       JSB GETCR
       JMP LXR3      ILLEGAL TERMINATION AFTER LABEL
       CPA BLANK     BLANK
       RSS           YES, VALID TERMINATION
       JMP LXR3
       LDB LAB1      MEMORY ADDRESS OF LABEL
       JSB LOKUP     SYMBOL TABLE LOOK UP
       STB LBLAD     SAVE SYMBOL TABLE ADDRESS
       LDB EDINT     EDIT INSTRUCTION FLAG
       SLB           REPLACE OR DELETE
       JMP LEX1      YES
       SZA           NO, LABEL EXIST IN TABLE
       JMP *+4       YES
       LDA .2
       STA LBLFG     VALUE FOR NON EXISTANT LABEL
      JMP LEX2
       SSA,RSS       LABEL DEFINED
       JMP LXR4      DOUBLY DEFINED LABEL
```

```
*
* STORE NEGATIVE VALUE FOR UNDEFINED LABEL
*
* RETAIN ADDRESS OF DEFINED LABEL ON EDIT OPERATION
*
LEX1  STA LBLFG     ADDRESS IN ASSEMBLED CODE
LEX2  JSB NTBLK     NEXT NON BLANK CHARACTER
      JMP LXR5      NO OPCODE FOUND
      JSB BCKSP     RETURN LAST CHAR TO BUFFER
      LDA M3
      STA TEMP3     READ THREE CHARACTERS
      CLE
LEX3  JSB GETCR     READ CHARACTER
      JMP LXR5      MNEMONIC NOT FOUND
      SEZ,RSS
      ALF,ALF       SHIFT ALTERNATE CHAR
      IOR OPADD,I
      STA OPADD,I   STORE CHAR IN OPCODE BUFFER
      SEZ,CME
      ISZ OPADD     ADVANCE BUFFER ADDRESS
      ISZ TEMP3     CHARACTER COUNT
      JMP LEX3      READ NEXT CHAR
      JSB MNEM      LOOK UP OP CODE NAME
      LDB EDINT     EDIT INSTR FLAG
      SLB,RSS       DELETE OR REPLACE
      JMP LEX4      NO
      LDA INSNM     YES, MEMORY REFERENCE INSTR
      CPA .6
      JMP LEX12     YES
      CPA .7        EXTENDED ARITH MEM REF
      JMP LEX12-1   YES
      CCB
      ADA M8        MACHINE CODE OR DATA EDIT
      SSA,RSS
      STB ASMFG     DATA
      JMP LEX,I
```

```
*
LEX4   JSB GETCR      GET TERMINATOR CHAR
       RSS
       JMP LEX5
*
* END OF LINE   CHECK INSTRUCTION NUMBER
*
       LDA INSNM
       CPA .8         END PSEUDO OP DOES NOT REQUIRE
       JMP LEX12+2    OPERAND
       ADA M3
       SSA,RSS        OPERAND EXPECTED
       JMP LXR6       YES, ERROR NO OPERAND FOUND
       JMP LEX,I      RETURN VALID INSTRUCTION
LEX5   CPA BLANK      VALID TERMINATOR AFTER OPCODE
       RSS            YES
       JMP LXR8       NO
       LDA INSNM      INSTRUCTION OPERAND NUMBER
       CPA .1         FIRST OPERAND TYPE
       JMP LEX,I      YES, RETURN NO OPERAND EXPECTED
       CPA .2         NO, SECOND OPERAND TYPE
       RSS            YES, CLEAR FLAG EXPECTED
       JMP LEX7       NO
       JSB GETCR
       JMP LEX,I      CLEAR FLAG NOT FOUND
       CPA C          CLEAR FLAG
       JMP LEX6
       CPA BLANK      BLANK TERMINATOR
       JMP LEX,I      YES, RETURN
       JMP LXR8       NO, ERROR ILLEGAL CHAR IN OPER
LEX6   LDA ASMBY
       IOR B1000      MASK IN CLEAR FLAG BIT
       STA ASMBY
       JSB TRMCK      CHECK TERMINATION
       JMP LEX,I
       JMP LXR8       ILLEGAL CHAR IN OPERAND
LEX7   CPA .3         THIRD OPERAND TYPE
       RSS            YES, READ CHANNEL NUMBER
       CPA .4         FOURTH OPERAND TYPE
       RSS            YES, READ IN CHANNEL NUMBER
```

```
*
LEX31 CPA .14      BSS PSEUDO OP
      RSS
      JMP LEX35
      JSB LABCK
      JMP LEX32    NO LABEL
      JMP LEX33    UNDEF LABEL/DOES NOT EXIST
      JSB DTRG,I   DEFINED LABEL
      ADA OPNUM
      JSB DTRG,I
      LDA A,I      RETRIEVE ADDRESS
      SZA,RSS
      JMP LXR17    UNDEFINED OPERAND
      LDA A,I      RETRIEVE VALUE
      JMP *+3
LEX32 CLA
      ADA OPNUM
      SZA,RSS      OPERAND VALUE ZERO
      JMP LXR19
      SSA
      JMP LXR19
      LDB M129     CHECK RANGE
      ADB A
      SSB,RSS
      JMP LXR13
      STA LENTH    SAVE LENGTH
      STA IDRCT    FLAG TO SIGNAL BSS
      JMP LEX,I
*
LEX33 JSB VAL
      JMP LEX32+1
```

```
        JMP LEX10     NO, ADVANCE TO REGISTER REFERENCE
        JSB NMBR,I    READ CHANNEL NUMBER
        JMP LEX9      FIRST CHAR NOT A NUMBER
LEX8    LDB D100
        JSB RANGE     CHECK RANGE OF OPERAND
        JMP LEX,I
        LDB INSNM     INSTRUCTION
        CPB .3        TYPE 3 OPERAND
        JMP LXR8      YES, ILLEGAL CHAR IN OPERAND
        CPA COMMA     COMMA BOFORE CLEAR FLAG
        RSS           YES
        JMP LXR8
        JSB GETCR     NEXT CHARACTER
        JMP *-2
        CPA C         CLEAR FLAG
        JMP LEX6      YES, MASK IN
        JMP LXR8      NO
LEX9    JSB LTPR,I    LETTER OR PERIOD
        JMP LXR8      NO, BAD DATA IN OPERAND
        LDB LAB2      YES, ADDRESS OF LABEL
        JSB LBRD,I    READ LABEL
        JMP LXR9      ILLEGAL CHAR BEGINS LABEL
        LDB LAB2      MEMORY ADDRESS OF LABEL
        JSB LOKUP     SYMBOL TBLE LOOK UP
        SZA,RSS       RETURN POSITIVE ADDRESS IN
        JMP LXR10     ASSEMBLED CODE
        SSA           POSITIVE ADDRESS
        JMP LXR10     NO, UNDEFINED LABEL
        LDA A,I       RETRIEVE VALUE
        JMP LEX8
LEX10   CPA .5        TYPE 5 OPERAND
        RSS
        JMP LEX11
        JSB NMBR,I    READ IN OPERAND VALUE
        JMP LXR8
        SZA,RSS       ZERO VALUE
        JMP LXR13     YES, ERROR
        LDB M16
        JSB RANGE     CHECK RANGE OF VALUE
        JMP LEX,I
        JMP LXR11     ILLEGAL CHAR AFTER OPERAND
```

```
*
* MEMORY REFERENCE TYPE INSTRUCTIONS
* ALL OPERAND EVALUATION IS HANDLED OUTSIDE LEXICAL SUBROUTINE
*
LEX11 CPA .6
      JMP LEX12
      CPA .7         EXTENDED ARITH MEMORY REFERENCE
      RSS
      JMP LEX13      NO
      ISZ LENTH      TWO WORD ASSEMBLY
LEX12 JSB OPRC,I
      JMP LEX,I
*
* END PSEUDO OP BRANCHES TO EXECUTE ROUTINE
*
      LDB EDTFG      EDIT OPERATION
      SZB
      JMP LXR12      ILLEGAL OP CODE DURING EDIT
      JMP XEQ,I      YES BEGIN EXECUTION
*
* THE REMAINDER OF THE INSTRCTIONS ARE FOR DATA DEFINITION PURPOSES
*
*
LEX13 LDB M29
      STB LNTH2
      INB
      STB TEMP3
      LDB DATBF      ADDRESS OF DATA BUFFER
      STB DATPT      RETAIN ADDR OF DATA BUFFER
      STB TEMP4      CLEAR DATA BUFFER TO ZERO
      CLB
      STB TEMP4,I
      ISZ TEMP4
      ISZ TEMP3
      JMP *-3
*
      STB LENTH      LENGTH OF DATA ENTRY
      STB TEMP3      CLEAR FOR ASC PSEUDO OP
```

```
*
*
*
       CPA .9         ASC PSEUDO OP
       RSS            YES
       JMP LEX16      NO
       JSB GTNM,I     INPUT POSITIVE INTEGER
       STA OPNUM
       SZA,RSS        VALUE ZERO
       JMP LXR13      YES, ERROR
       LDB M29        CHECK RANGE
       ADB A
       SSB,RSS
       JMP LXR13      VALUE OUT OF RANGE
       JSB NTBLK      NEXT NON BLANK CHARACTER
       JMP LXR11      NO OPERAND FOUND
       CPA COMMA      COMMA BEFORE DATA
       RSS            YES
       JMP LXR8       NO, ERROR
       LDB OPNUM
       BLS            MULTIPLY BY 2 FOR CHARACTER COUNT
       CMB,INB
       STB TEMP
       CLO
       CLE
LEX14  SOC
       JMP LEX15
       JSB GETCR      NEXT CHARACTER
       RSS            NONE FOUND
       JMP LEX15+1
       STO            SET IF NO CHAR FOUND
LEX15  LDA BLANK      LOAD BLANK
       SEZ,RSS
       ALF,ALF        SHIFT CHARACTER
       IOR TEMP3      MASK IN CHARACTER
       STA TEMP3
       SEZ,CME
       JSB STDAT      STORE DATA IN BUFFER
       STA TEMP3      CLEAR STORE WORD
       ISZ TEMP
       JMP LEX14
       JMP LEX,I
```

```
*
*
LEX16 CPA .10        DEC PSEUDO OP
      RSS            YES
      JMP LEX22
LEX17 JSB CNST,I     REAL OR DECIMAL INTEGER
      JSB TPCK,I     INTEGER OR REAL
      JMP LEX20      REAL
LEX19 JSB STDAT      INTEGER IN (A)
      JSB TRMCK
      JMP LEX,I      RETURN
      CPA COMMA     COMMA SEPARATING DATA
      JMP LEX17      YES, READ NEXT NUMBER
      JMP LXR8       NO, BAD CHARACTER
LEX20 JSB STDAT      STORE FIRST WORD OF REAL
      LDA TEMP2      SECOND WORD OF REAL
      JMP LEX19
*
*
LEX22 CPA .11        OCT PSEUDO OP
      RSS
      JMP LEX24
LEX23 JSB OCTN,I     READ IN OCTAL INTEGER
      JSB STDAT      STORE IN DATA BUFFER
      JSB TRMCK      CHECK TERMINATION
      JMP LEX,I
      CPA COMMA      COMMA SEPARATING DATA
      JMP LEX23      YES, READ NEXT INTEGER
      JMP LXR8       NO, ERROR
*
```

```
*
LEX24 CPA .12       EQU PSEUDO OP
      RSS
      JMP LEX28
      LDB LBLFG    LABEL FLAG
      SZB,RSS
      JMP LXR14     NO LABEL PRECEDES EQU
      JSB LABCK     READ IN AND EXAMIN OPERAND
      JMP LEX26     LABEL NOT FOUND
      JMP LXR10     LABEL IS UNDEFINED
      ADA OPNUM     ADD IN CONSTANT
      JSB DTRG,I    CHECK RANGE OF ADDRESS
      STA ZADD      ADDRESS IN ASSEMBLED CODE
      JSB TRMCK     CHECK TERMINATION
      JMP LEX,I
      JMP LXR11     BAD DATA FOLLOWS OPERAND
*
*
* STORE OPERAND VALUE IN LAST POSITION OF DATA BLOCK
* IF LABEL NOT PRESENT IN OPERAND
*
LEX26 LDA OPNUM
      SSA
      JMP LXR15     OPERAND MUST BE POSITIVE
      LDB D100
      ADB A
      SSB,RSS
      JMP LXR13     OPERAND VALUE TOO LARGE
      LDB YDATA     LAST LOCATION IN DATA BLK
      STA YDATA,I   STORE ADDRESS
      STB ZADD      RETAIN ADDRESS IN ASSEMBLED CODE
      ADB M1
      STB YDATA     UPPER BOUND OF DATA BLOCK
      JMP LEX,I
*
```

```
*
LEX28 CPA .13       ABS PSEUDO OP
      RSS
      JMP LEX31
      JSB LABCK     READ IN OPERAND
      JMP LEX29     NO LABEL
      JMP LEX30     UNDEFINED LABEL
      JSB DTRG,I    DEFINED LABEL
      ADA OPNUM     INCLUDE CONSTANT
      JSB DTRG,I    CHECK ADDRESS RANGE
      LDA A,I       RETRIEVE DATA ADDRESS
      SZA
      JMP LXR17     UNDEFINED OPERAND
      JSB STDAT     STORE DATA
      JMP LEX,I
*
LEX29 CLA
      ADA OPNUM
      SSA           NEGATIVE
      JMP LXR13     YES, ERROR
      LDB D100
      ADB A
      SSB,RSS       CHECK RANGE OF NUMERIC
      JMP LXR13
      JSB STDAT     STORE DATA VALUE IN BUFFER
      JMP LEX,I
*
LEX30 CCB
      JSB VAL       REQUEST USER ENTRY
      JMP LEX29+1
*
```

```
*
*
*  DEF PSEUDO OP
*
* THE FORMAT FOR THE DEF INSTRUCTION IS:
*
* (LABEL) DEF LABEL(,I)
*
* THE OPERAND IS FURTHER RESTRICTED THAN A MEMORY REF
* INSTRUCTION.  FOR THAT REASON SUBROUTINE OPREC WHICH
* NORMALLY READS OPERANDS WILL NOT BE USED FOR THE DEF
* OPERAND.  INSTEAD THE LABEL READING SUBROUTINE, LABRD,
* WILL BE USED WITH A SPEARATE CHECK FOR THE INDIRECT
* FLAG.
*
*
*  UNDEFINED OPERANDS
*
* DURING AN EDIT OPERATION THE INSTRUCTION WILL NOT BE
* PERMITTED.
* DURING NORMAL PROGRAM DEFINITION A REQUEST TO DEFINE
* THE UNDEFINED LABEL ON THE NEXT ENTRY IS PRESENTED.
* FAILURE TO DO SO WILL RESULT IN A MEANINGLESS ADDRESS
* DEFINITION.
*
*
* ENTERING A DEF INSTRUCTION PRIOR TO PROGRAM COMPLETION
* MAY LEAD TO UNEXPECTED RESULTS IF A DATA EDIT OPERATION
* OCCURS
*
* DATA EDITS INVOLVE SHIFTING OF DATA TO MAKE SPACE FOR
* AN INSERTION OR TO FILL A GAP LEFT BY A DELETION.
* INSUCH CASES SHIFTING WILL ALTER A DEF POINTER.
*
*
* IT IS STRONGLY RECOMMENDED THAT ALL DEF INSTRUCTIONS
* BE THE LAST DATA ENTRIES BEFORE BEGINNING FINAL PROGRAM
* EXECUTION AND AFTER ALL DATA EDIT OPERATIONS OR
* THAT ANY DATA DEFINITIONS REFERENCES BY A DEF BE THE
* FIRST DATA ENTRIES AND ALL DATA EDIT OPERATIOND
* REFERENCE SUBSEQUENT DATA ENTRIES
```

```
*
*
LEX35 CPA .15      DEF PSEUDO OP
      RSS
      JMP LXR16
      CCA
      LDB LAB2     READ LABEL FIRST CHAR NOT READ
      JSB LBRD,I
      JMP LXR9     ILLEGAL CHAR BEGINS LABEL
      JSB TRMCK    CHECK TERMINATION
      JMP LEX36    VALID TERMINATION
      CPA COMMA
      RSS
      JMP LXR11    ERROR
      JSB GETCR    NEXT CHARACTER
      JMP LXR11
      CPA I        INDIRECT BIT
      RSS          YES
      JMP LXR11
      STA IDRCT    SET INDIRECT FLAG
LEX36 LDB LAB2     ADDRESS OF LABEL
      JSB LOKUP    SYMBOL TABLE LOOK UP
      SZA,RSS      LABEL EXIST
      JMP LEX37
      SSA          LABEL DEFINED
      JMP LEX37    NO
      ADB .3       YES
```

```
      LDA B,I
      JMP LEX38
*
LEX37 LDB EDTFG    UNDEFINED LABEL NOT PERMITTED
      SZB          ON AN EDIT OPERATION
      JMP LXR18
      CCB
      STB UNDEF
      LDA ZDATA    NEXT LOCATION IN DATA AREA
      INA
LEX38 JSB IDIRT    MASK ON INDIRECT BIT IF NECESSARY
      JSB STDAT    STORE DATA IN BUFFER
      ISZ UNDEF    UNDEFINED LABEL
      JMP LEX,I
      JSB NWLN,I
      LDA M8
      LDB LXMS2
      JSB WRITE,I
      LDA M6
      LDB LAB2
      JSB WRITE,I  PRINT LABEL NAME
      LDA .14
      LDB LXMS3
      JSB WRITE,I  PROMPT TO DEFINE LABEL
      JMP LEX,I
*
```

```
*
LXMS2 DEF *+1
      ASC 4, DEFINE
*
*
LXMS3 DEF *+1
      ASC 7, ON NEXT ENTRY
*
*
* LEXICAL ERROR MESSAGES
*
LXR1  LDA .26
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 13,FIRST CHARACTER NOT FOUND
*
*
LXR2  LDA .24
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 12,ILLEGAL FIRST CHARACTER
*
*
LXR3  LDA .22
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 11,BAD DATA FOLLOWS LABEL
*
*
LXR4  LDA .20
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 10,DOUBLY DEFINED LABEL
```

```
*
*
LXR5  LDA .22
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 11,INSTRUCTION NOT FOUND
*
*
LXR6  LDA .16
      LDB *+2
      JMP ERCAL
*
      DEF ERR6     NO OPERAND FOUND
*
*
LXR7  LDA .24
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 12,BAD DATA FOLLOWS OP CODE
*
*
LXR8  LDA .26
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 13,BAD DATA IN OPERAND FIELD
*
*
LXR9  LDA .30
      LDB *+2
      JMP ERCAL
*
      DEF ERR5     ILLEGAL CHARACTER BEGINS LABEL
*
*
```

```
LXR10 LDA .26
      LDB *+2
      JMP ERCAL
*
      DEF ERR8      UNDEFINED LABEL IN OPERAND
*
*
LXR11 LDA .28
      LDB *+2
      JMP ERCAL
*
      DEF ERR4      ILLEGAL OPERAND TERMINATION
*
*
LXR12 LDA .32
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 16,ILLEGAL INSTRUCTION DURING EDIT
*
*
LXR13 LDA .26
      LDB *+2
      JMP ERCAL
*
      DEF ERR3      OPERAND VALUE OUT OF RANGE
*
*
LXR14 LDA .32
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 16,NO LABEL PRECEDES EQU PSEUDO OP.
```

```
*
*
LXR15 LDA .24
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 12,ADDRESS MUST BE POSITIVE
*
*
LXR16 LDA .22
      LDB *+3
      JSB BPLN      PRINT ERROR MESSAGE
      HLT 33B       HALT, PROGRAM ERROR
*
      DEF *+1
      ASC 11,INSTRUCTION NOT FOUND
*
LXR17 LDA .20
      LDB *+2
      JMP ERCAL
*
      DEF ERR7      OPERAND IS UNDEFINED
*
*
LXR18 LDA .50
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 25,UNDEFINED LABEL NOT PERMITTED WITH DEF DURING EDIT
*
*
LXR19 LDA .40
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 20,OPERAND VALUE MUST BE GREATER THAN ZERO
*
```

```
*
*
* CHECK RANGE OF OPERAND VALUE
*
* ENTER (A) VALUE IN OPERAND
*       (B) UPPER BOUND OF OPERAND VALUE
*
*
RANGE NOP
      STA OPNUM    CHANNEL NUMBER/NUMBER OF SHIFTS
      SSA          POSITIVE
      JMP LXR13    NO, VALUE OUT OF RANGE
      ADA B
      SSA,RSS      TOO LARGE
      JMP LXR13    YES, VALUE OUT OF RANGE
      LDA ASMBY
      IOR OPNUM    MASK IN OPERAND
      STA ASMBY    RESTORE
      JSB TRMCK
      JMP RANGE,I  RETURN  VALID TERMINATION
      ISZ RANGE
      JMP RANGE,I
*
*
* STORE DATA IN SPECIAL STORE BUFFER DURING LEXICAL SCAN
*
* ENTER (A) DATA ITEM TO STORED IN BUFFER
*
*
STDAT NOP
      STA DATPT,I  STORE DATA IN BUFFER
      CLA
      ISZ DATPT    ADVANCE POINTER
      ISZ LENTH    COUNT LENGTH
      ISZ LNTH2    DATA BUFFER OVERFLOW
      JMP STDAT,I  NO
      LDA .32
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 16,DATA INPUT EXCEEDS IMPOSED LIMIT
```

```
*
*
*
* INPUT TEMPORARY VALUE FOR UNDEFINED LABEL
*
*
VAL    NOP
       STB ABSSF     SET ASB/BSS FLAG
       JSB LXNTY     REQUEST USER UNTERVENTION
       LDB SRCNT     SAVE LENGTH OF INPUT
       STB RDCOM
       LDA BUFA      INPUT BUFFER ADDRESS
       LDB BUFB      AUXILIARY BUFFER
       STA WMOVE     SAVE PRIMARY BUFFER ADDRESS
       STB BUFA      USE AUXILIARY BUFFER FOR INPUT
*
* JUMP TO SYSTEM CONTROLLER TO READ INPUT
*
       JMP CNTRL,I
*
* RETURN POINT FROM CONTROLLER
*
LXRTN  JSB BCKSP     RETURN FIRST CHAR TO BUFFER
       LDB RDCOM     RESTORE LENGTH OF INPUT BUFFER
       STB SRCNT
       LDA WMOVE
       STA BUFA      RESTORE MAIN BUFFER ADDRESS
       CLB
       STB ABSSF     CLEAR ABS/BSS FLAG
       JSB NMBR,I    READ IN INTEGER
       JMP LXR8      FIRST CHAR NOT A DIGIT
       JMP VAL,I
```

```
*
*
* PRINT PROMPT TO INPUT A VALUE FOR UNDEFINED LABEL
*
*
LXNTY NOP
      JSB NWLN,I   OUTPUT CR-LF
      LDB LAB2
      LDA M6
      JSB WRITE,I  PRINT LABEL
      LDA .40
      LDB LXMS1
      JSB WRITE,I
      JMP LXNTY,I
*
*
LXMS1 DEF *+1
      ASC 20, IS UNDEFINED TYPE IN A TEMPORARY VALUE
*
```

```
*
*
* READ IN AND EXAMINE OPERAND FOR DATA DEFINITION INSTRUCTIONS
*
* RETURN P+1 NO OPERAND LABEL
*        P+2 OPERAND LABEL IS UNDEFINED
*        P+3 LABEL DEFINED ADDRESS IN (A)
*
*
LABCK NOP
      JSB OPRC,I
      LDB OPLBL      OPERAND LABEL
      SZB,RSS
      JMP LABCK,I    LABEL NOT FOUND
      SSB
      JMP LXR8       PLC IS NOT VALID
      LDB LAB2       ADDRESS OF LABEL
      JSB LOKUP
      SSA
      JMP LABC1      UNDEFINEC LABEL
      SZA,RSS
      JMP LABC1      LABEL DOES NOT EXIST
      JSB DTRG,I     CHECK FOR DATA LABEL
      ISZ LABCK
LABC1 ISZ LABCK
      JMP LABCK,I
```

```
*
*
* CHECK FOR OVERFLOW IN DATA TABLE
*
*
DATFL NOP
      LDA ZDATA     NEXT FREE DATA AREA
      CMA,INA
      ADA YDATA     UPPER BOUND OF DATA AREA
      SSA,RSS       OVERFLOW
      JMP DTFL1     NO
      LDA .30
      LDB *+2
      JMP TBLOV     TABLE OVERFLOW
*
      DEF *+1
      ASC 15,OVERFLOW IN PROGRAM DATA TABLE
*
*
DTFL1 ADA M10
      SSA,RSS       DATA TABLE NEAR OVERFLOW
      JMP DATFL,I   NO
      JSB NWLN,I    NEW LINE
      LDA .40
      LDB *+3       PRINT WARNING MESSAGE
      JSB WRITE,I
      JMP DATFL,I
*
      DEF *+1
      ASC 20,DATA TABLE NEARLY FULL, BEGIN EXECUTION
```

```
*
*
* SUBROUTINE CLEAR TO INITIALIZE VARIABLES USED IN THE LEXICAL SCAN
*
*
CLEAR NOP
      CLB
      STB LABL1
      STB LABL1+1
      STB LABL1+2  CLEAR LABEL BUFFERS
      STB LABL2
      STB LABL2+1
      STB LABL2+2
*
      STB ASMBY    SKELETON OF ASSEMBLED INSTRUCTION
      STB ASMFG    ASSEMBLY FLAG
      STB IDRCT    INDIRECT FLAG
      STB INSNM    INSTRUCTION NUMBER
      STB LBLAD    LABEL ADDRESS
      STB LBLFG    LABEL FLAG
      STB MNC      MNEMONIC BUFFER
      STB MNC+1
      STB NUMFG    OPERAND NUMBER FLAG
      STB OPLBL    OPERAND LABEL
      STB OPNUM    NUMERIC VALUE IN OPERAND
      STB ZADD     ADRESS IN ASSEMBLED CODE
      INB
      STB LENTH    LENGTH OF ASSEMBLY
      LDB MNMNC    OPERAND ADDRESS STORE
      STB OPADD
      JMP CLEAR,I
```

```
*
*
* SYMBOL TABLE LOOKUP
*
* ENTER (B) = ADDRESS OF LABEL
*
* RETURN (A) > 0 ADDRESS OF LABEL IN PROGRAM
*        (A) = 0 LABEL DOESNOT EXIST
*        (A) < 0 UNDEFINED LABEL
*
*        (B) SYMBOL TABLE ADDRESS OF LABEL
*
*
LOKUP NOP
      JSB FIND     FIND LABLE IN SYMBOL TABLE
      SZA,RSS      LABEL EXISTS
      JMP LOKUP,I  NO, LABEL NOT IN TABLE
      ADB .2       YES, GET INFO ON LABEL
      LDA B,I
      CLE,ERA
      INB
      LDA B,I      ADDRESS IN ASSEMBLED CODE
      SEZ,RSS      UNDEFINED REFERENCE
      CMA,INA      YES
      ADB M3       RESTORE LABEL ADDRESS
      JMP LOKUP,I
```

```
*
*
* FIND LABEL IN SYMBOL TABLE
*
*
* THE SYMBOL TABLE HAS BEEN IMPLEMENTED TO HOLD NO MORE
* THAN 125 LABELS.  AN ATTEMPT TO INTRODUCE MORE THAN
* 125 WILL CAUSE THE ASSEMBLER TO HALT WITH THE USER S
* PROGRAM LOST
*
* EACH SYMBOL TABLE ENTRY IS SIX WORDS IN LENGTH
*
* WORD 1   FIRST TWO CHARACTERS OF LABEL
*
* WORD 2   THIRD AND FOURTH CHARACTER IN LABEL
*
* WORD 3   BITS 8-15 LAST CHARACTER
*          BIT  0 = 0 UNDEFINED LABEL
*                   1   DEFINED LABEL
*
*
* WORD 4 AND 5 HAVE DIFFERENT USES IF THE LABEL IS OR
* IS NOT DEFINED
*
* UNDEFINED WORD 4  ADDRESS TO LAST DIRECT FORWARD REF
*           WORD 5  ADDRESS TO LAST INDIRECT FORWARD REF
*
*   DEFINED WORD 4  LABEL ADDRESS IN ASSEMBLED CODE
*           WORD 5  LABEL ADDRESS IN SCB
*
* WORD 6   LINK TO SPECIAL SYMBOL TABLE FOR COMPOUND
*          OPERANDS
*
* ENTER (B) ADDRESS OF LABEL BUFFER
*
* RETURN (B) SYMBOL TABLE ADDRESS OF LABEL
*        (A) = 0 LABEL NOT IN TABLE
*
*
```

```
FIND   NOP
       STB ADDR3     RETAIN ADDRESS OF LABEL
       STB ADDR2
       CLB
       STB LMTFG     CONTROL SEARCH OF SYM TBL
       STB TEMP4
       LDA M3
       STA TEMP3
       CLB,RSS
       ISZ ADDR3
FIND1  LDA ADDR3,I
       ADA TEMP4     SUM ELEMENTS OF LABEL
       STA TEMP4
       ISZ TEMP3
       JMP FIND1-1
       DIV .125      REMAINDER IN (B)  0-124
       BLS           MULTIPLY REMAINDER BY 6
       STB A
       BLS
       ADB A
       ADB XSTBL     BASE ADDRESS OF SYMBOL TABLE
       STB TEMP3
FIND2  LDA B,I
       SZA           LABEL CELL EMPTY
       JMP FIND6     NO, SOMETHING IN SYMBOL TABLE
```

```
*
* EITHER LABEL NOT IN TABLE OR LOCATION FREE TO STORE LABEL
*
      JMP FIND,I
FIND4 ADB .6
      LDA LMTFG
      SZA,RSS
      JMP FIND5
      LDA TEMP3
      CMA,INA
      ADA B
      SSA,RSS       TABLE OVERFLOW
      JMP FNDER     YES
      JSB FND1
      JMP FIND2
FIND5 LDA YSTBL     UPPR BND OF SYM TBL
      CMA,INA
      ADA B
      SSA           TABLE BOUND EXCEEDED
      JMP *+3       NO
      ADB M750      YES, SEARCH BEGINNING OF TABLE
      STB LMTFG     SEARCH OTHER SIDE OF TABLE
      JSB FND1
      JMP FIND2
FIND6 JSB FND1
      STB TEMP4     RETAIN SYM TBL ADDR
      CLE
      JMP *+4
FIND7 CCE
      INB           ADVANCE ADDRESSES
      ISZ ADDR3
      LDA B,I
      CPA ADDR3,I   MATCH
```

```
      JMP *+2      YES
FIND8 JMP FIND9    NO
      SEZ,RSS
      JMP FIND7
      INB          ADVANCD ADDRESS
      ISZ ADDR3
      LDA B,I
      AND CH1      MASK OFF UPPER 8 BITS
      CPA ADDR3,I  MATCH
      CCA,RSS      SET (A) NE 0
      JMP FIND9
      LDB TEMP4    RETRIEVE LABL ADDR
      JMP FIND,I
FIND9 LDB TEMP4    GET PREVIOUS ADDRESS
      JMP FIND4    CHECK NEXT ENTRY
*
*
FNDER LDA .22
      LDB *+2
      JMP TBLOV
*
      DEF *+1
      ASC 11,SYMBOL TABLE OVERFLOW
*
*
* RETRIEVE ADDRESS OF LABEL
*
*
FND1  NOP
      LDA ADDR2
      STA ADDR3    RESTORE ADDRESS OF LABEL
      JMP FND1,I
```

```
*
*
* LOOK UP MNEMONIC IN TABLE
*
* RETRIEVE INSTRUCTION NUMBER AND INSTRUCTION SKELETON
*
*
MNEM  NOP
      CLE
      LDA XOPCD    ADDR OF OP CLDE TABLE
      STA LWRBD    LOWER BOUND IN SEARCH
      LDB .86
      ADB A
      STB UPRBD    UPPERBOUND
      ADB LWRBD
      JMP MNEM1
      LDA LWRBD
      CMA          COMPLEMENT LOWER BOUND
      ADA UPRBD
      CLE,SZA,RSS  CHECK FOR CONVERGENCE
      CCE
MNEM1 BRS          DIVIDE BY TWO
      LDA B,I
      CPA MNC      FIRST TWO CHARACTERS MATCH
      JMP MNEM3    YES
      SEZ          NO
      JMP MNER     MNEMONIC NOT IN TABLE
      CMA,INA      HALVE INTERVAL
      ADA MNC
      SSA
      JMP MNEM2
      STB LWRBD    SET NEW LOWER BOUND
      ADB UPRBD
      JMP MNEM1-5
MNEM2 STB UPRBD    SET NEW UPPER BOUND
      ADB LWRBD
      JMP MNEM1-5
MNEM3 ADB M1       BACK UP FOR SEVERAL MNEMONICS
      LDA B,I      BEGIN WITH THE SAME TWO LETTERS
```

```
      CPA MNC
      JMP *-3
      ADB .86
      LDA M6
      STA TEMP3
MNEM4 INB               ADVANCE ADRES TO NEXT ENTRY
      LDA B,I           TEST FOR THIRD CHARACTER
      AND CH1           MASK OUT FIRST CHARACTER POSITION
      CPA MNC+1         THIRD CHARACTER MATCH
      JMP MNEM5         YES
      ISZ TEMP3         NO
      JMP MNEM4         LOOK AT NEXT OPCODE
      JMP MNER          OPCODE NOT FOUND
MNEM5 ADB M86           BACK UP TO CHECK FIRST
      LDA B,I           TWO CHARACTERS
      CPA MNC
      RSS
      JMP MNER          ERROR
      ADB .86
      LDA B,I
      AND B177          GET INSTRUCTION NUMBER
      STA INSNM         RETAIN
      ADA M8
      SSA
      CLA,INA,RSS       MACHINE INSTRUCTION
      CCA               DATA
      STA ASMFG         ASSEMBLY FLAG
      ADB .86
      LDA B,I
      STA ASMBY         SKELETON OF ASSEMBLED INSTRUCTION
      JMP MNEM,I
*
* INSTRUCTION NOT FOUND
*
MNER  LDA .30
      LDB *+2
      JMP ERCAL
      DEF *+1
      ASC 15,ILLEGAL ASSEMBLER INSTRUCTION
```

```
*
*
*
      ORG 6000B
*
* INPUT A CONSTANT
*
*
* RETURN P+1 VALID DATA IN (A) AND (B)
*
* THE TERMINATOR WILL BE RETURNED TO THE INPUT STRING
*
*
CONST NOP
      JSB NTBLK    NEXT NON BLANK CHARACTER
      JMP NUMR1    NO DATA FOUND
      CLB
      STB SIGN     SET SIGN POSITIVE
      INB
      CPA PLUS     POSITIVE SIGN
      JMP CONS1    YES
      CPA MINUS    NO, NEGATIVE SIGN
      CCB,RSS      YES
      JMP CONS2    NO
CONS1 STB SIGN     RECORD SIGN
      JSB GETCR    FETCH NEXT CHARACTER
      JMP NUMR2    SOLITARY SIGN
CONS2 JSB NUMCK    FETCH CONSTANT
      JMP CONST,I
```

```
*
*
* FETCH NUMBER AND CONVERT TO BINARY
*
* RETURN P+1 VALID DATA RETURNED IN (A) AND (B)
*
*
NUMCK NOP
      CLB
      STB EXP
      STB MANT1     ZERO ALL COMPONENTS OF NUMBER
      STB MANT2
      STB EXPON
      STB TEMP3     SET NUMBER FLAG FALSE
      CCB
      STB DPFLG     SET DECIMAL POINT FLAG FALSE
      STB EFLG      SET EXPONENT FLAG FALSE
NUMC1 CPA PRIOD     DECIMAL POINT
      ISZ DPFLG     YES, SET FLAG TRUE
      JMP NUMC2     NO
      CLA           INITIALIZE POST DECIMAL DIGIT
      STA EXPON     DIGIT COUNTER TO ZERO
      JMP NUMC3+1   FETCH A CHARACTER
NUMC2 JSB DECHK
      JMP NUMC7
      ISZ EXPON     YES  COUNT DIGIT
      ALF,ALF
      ALF,RAR       LEFT JUSTIFY DIGIT AND SAVE IT
      STA TEMP4
      JSB MBY10     MULTIPLY PREVIOUS NUMBER BY 10
      LDB EXP
      SZB           ZERO EXPONENT
      JMP NUMC4     NO
      LDA .4        YES  SET EXPONENT TO 4
      STA EXP
      LDA TEMP4     LOAD NUMBER
```

```
        CLB
NUMC3   JSB NORML       NORMALIZE THE NUMBER
        ISZ TEMP3       SET NUMBER OCCURRED FLAG
        JSB GETCR
        JMP NUM12
        JMP NUMC1       NEW CHARACTER
NUMC4   ADB M4
        CMB             COMPUTE EXPONENT BIAS AND SAVE IT
        LDA TEMP4
        STB TEMP4
        CLB
NUMC5   ISZ TEMP4       DIGIT POSITIONED
        JMP NUMC6       NO
        CLE             YES  ADD IN LOW PART OF NUMBER
        ADB MANT2
        CLO
        SEZ             OVERFLOW
        INA               YES ADVANCE A
        ADA MANT1       ADD IN HIGH PART OF NUMBER
        SOS             OVERFLOW
        JMP NUMC3         NO
        CLE,ERA         YES ROTATE DOWN AND BUMP
        ERB             EXPONENT
        ISZ EXP
        NOP
        JMP NUMC3
NUMC6   CLE,ERA
        ERB             SHIFT DIGIT RIGHT
        JMP NUMC5
NUMC7   CLB             DECIMAL POINT
        STB TEMP4
        CPB TEMP3       OR DIGIT FOUND
```

```
       JMP NUMR3     NO, BAD INPUT DATA
       CPA E         YES, E
       ISZ EFLG      YES
       JMP NUM12     NO, NO EXPONENT PART
       JSB GETCR
       JMP NUMR4
       CPA PLUS
       JMP NUMC8
       CPA MINUS
       CCA,RSS
       JMP NUMC9
       STA TEMP4     NOTE MINUS SIGN
       STA TEMP3
NUMC8  JSB GETCR
       RSS
NUMC9  JSB DECHK     DIGIT
       RSS           NO
       JMP NUMCA
       CLA           CHECK FOR ZERO EXPONENT
       CPA TEMP3     ZERO EXPONENT
       JMP NUM10     YES
       JMP NUMR4     NO
NUMCA  CPA ZERO      LEADING ZERO
       JMP NUMC8-1   YES, SET EXPONENT ZERO
       STA TEMP3     NO, SAVE IT
       JSB GETCR
       JMP NUM10     SECOND DIGIT
       JSB DECHK
       JMP NUM10     NO
       LDB TEMP3     YES
       BLS,BLS
       ADB TEMP3     MULTIPLY PRIOR DIGIT BY 10
       BLS
       ADA B         ADD NEW DIGIT
       STA TEMP3     SAVE EXPONENT
       JSB GETCR
```

```
      JMP NUM10    THIRD DIGIT
      JSB DECHK
      RSS
      JMP NUMR4    EXPONENT TOO LONG
NUM10 LDA TEMP3    RETRIEVE EXPONENT
      ISZ TEMP4    POSITIVE
      CMA,INA      YES, COMPLEMENT IT
      RSS          NO
NUM12 CLA
      ISZ DPFLG    DECIMAL POINT
      ADA EXPON     YES, CORRECT EXPONENT
      SZA,RSS      ZERO EXPONENT
      JMP NUM14    YES
      SSA          NO, NEGATIVE EXPONENT
      JMP NUM13    NO
      CMA,INA      YES, SET COUNTER
      STA EXPON
      JSB DBY10    DIVIDE NUMBER BY 10
      ISZ EXPON    FINI
      JMP *-2      NO
      JMP NUM14    YES
NUM13 STA EXPON    SET COUNTER
      JSB MBY10    MULTIPLY BY 10
      ISZ EXPON    FINI
      JMP *-2      NO
NUM14 LDA MANT1    YES, LOAD
      LDB MANT2    NUMBER
      ISZ SIGN     POSITIVE
      JMP NUM15    YES
      CMA          NO, COMPLEMENT IT
      CMB,INB,SZB,RSS
      INA
NUM15 JSB .PACK      PACK NUMBER INTO (A) AND (B)
      STA TEMP1
      STB TEMP2
      JSB BCKSP    RETURN TERMINATOR TO BUFFER
      LDA TEMP1    RESTORE (A)
      JMP NUMCK,I
```

```
*
*
* NORMALIZE AND PACK FLOATING POINT NUMBER
*
*
.PACK NOP          MANTISSA IN (A) AND (B)
      JSB NORML    EXPONENT IN EXP, (E) CLEARED
      CLE,SZA,RSS  ZERO RESULT
      JMP .PACK,I  YES
      ADB B177     NO, ROUND
      SSA,RSS      POSITIVE NUMBER
      INB          YES, FINISH ROUND
      CLO
      SEZ          OVERFLOW FROM (B)
      CLE,INA      YES , ADVANCE (A)
      SOS          OVERFLOW (A) = 100000, (B) = 0
      RAL
      SSA,SLA,RSS  TWO HIGH BITS 1, (A) = 140000
      JMP PACK1    NO
      CCE          YES
      ARS,SLA,ALS  SET (A) = 100000 AND SKIP
PACK1 RAR          COUNTERPART TO *-5
      STA MBY10
      LDA 1
```

```
AND M256     DELETE 8 LOW ORDER BITS OF MANTISSA
STA 1        SAVE LOW ORDER MANTISSA
LDA EXP      FETCH EXPONENT
SEZ          DECREMENT EXPONENT
ADA M1       YES
SOC          NO, PRIOR OVERFLOW
INA          YES, INCREMENT EXPONENT
ADA B200     NO
SSA          EXPONENT UNDERFLOW
JMP NUMR8    YES, ERROR
ADA M256     NO
SSA,RSS      EXPONENT OVERFLOW
JMP NUMR8    YES,ERROR
ADA B200     NO, RESTORE EXPONENT, POSITION SIGN
RAL
AND B377
ADB A        MASK TO 8 BITS, AND COMBINE WITH
LDA MBY10    LOW MANTISSA, RETRIEVE HIGH
CPA MNEG     MANTISSA
RSS          NEGATIVE
JMP .PACK,I
CPB B376     OVERFLOW
JMP NUMR8    YES
JMP .PACK,I  NO
```

```
*
*
* NORMALIZE A, B, AND EXP
*
*
NORML NOP
      STA MBY10    SET LEFT-SHIFT
      CLA
      STA MPY      COUNTER TO ZERO
      LDA MBY10
      SZA,RSS
      SZB          ON ZERO CLEAR EVERYTHING
      JMP NORM3
      STA EXP
      STA MANT1
NORM1 STB MANT2    STORE MANTISSA AND RETURN
      JMP NORML,I
NORM2 ISZ MPY      COUNT LEFT SHIFTS
NORM3 CLE,ELB      ROTATE (A) AND (B) LEFT INTO (E)
      ELA
      SEZ,SSA,RSS  TWO HIGHEST BITS 0
      JMP NORM2    YES, + UNNORMALIZED
      SEZ,SSA      NO, TWO HIGHEST BITS 1
      JMP NORM2    YES, - UNNORMALIZED
      ERA
      ERB,CLE      SHIFT TO NORMALIZE MANTISSA
      STA MANT1    NO, COMPUTE CORRECTED EXPONENT
      LDA MPY
      CMA,INA
      ADA EXP
      STA EXP
      LDA MANT1
      JMP NORM1
```

```
*
*
* MULTIPLY UNPACKED NUMBER BY 10
*
*
MBY10 NOP
      LDA MANT1
      SZA,RSS      RETURN ON ZERO  MANTISSA
      JMP MBY10,I
      LDB EXP
      ADB .3       MULTIPLY BY 8
      STB EXP
      LDB MANT2    LOAD MANTISSA
      CLE,ERA
      ERB          DIVIDE BY 4
      CLE,ERA
      ERB,CLE
      ADB MANT2
      SEZ          DOUBLE ADD TO PRODUCE 1.25 $
      INA          MANTISSA
      ADA MANT1
      SSA,RSS      CORREXT ON OVERFLOW
      JMP *+5
      CLE,ERA
      ERB
      ISZ EXP
      NOP
      STA MANT1
      STB MANT2
      JMP MBY10,I
```

```
*
*
* DIVIDE UNPACKED NUMBER BY 10
*
*
DBY10 NOP          MULTIPLY BY DOUBLE-LENGTH TENTH
      LDA MANT1
      SZA,RSS      RETURN ON ZERO MANTISSA
      JMP DBY10,I
      LDB M2
      ADB EXP      ADD EXPONENT OF TENTH TO  MANTISSA EXPONENT
      STB EXP
      LDA MANT2
      CLE,ERA      JUSTIFY LOWER MANTISSA
      JSB MPY      MULTIPLY BY ONE TENTH
      DEF TENTH
      CLE,ELA      SHIFT BACK
      ELB,CLE
      ADA B        ADD IN LOW ORDER MANTISSA
      SEZ
      INB          TENTH**2-16 AND ROUND  TO 16 BITS
      STB MANT2
      LDA MANT1
      JSB MPY      DO SAME FOR HIGH MANTISSA
      DEF TENTH
      CLE
      ADA B
      ADA MANT2    EFFECTIVELY SUM DOUBLE LENGTH PRODUCTS
      SEZ
      INB
      SWP          EXCHANGE (A) AND (B)
      JSB NORML    NORMALIZE RESULT
      JMP DBY10,I
```

```
*
*
* MULTIPLY INTEGER IN A
*
*
MPY   NOP          ADDRESS OF MULTIPLIER IN MPY,I
      LDB M2
      STB MBY10    SET -2 IN SIGN TEMP
      LDB MPY,I
      LDB B,I      LOAD MULTIPLIER
      CLE,SSA      (A) NEGATIVE
      CMA,CME,INA  YES, COMPLEMENT (A) AND (E)
      SSB          (B) NEGATIVE
      CMB,CME,INB  YES, COMPLEMENT (B) AND (E)
      SEZ          (E) = 0
      ISZ MBY10    NO, SET SIGN OF RESULT NEGATIVE
      STB NORML    SAVE MULTIPLIER
      LDB M16
      STB TEMP1    SET COUNTER
      CLB          ZERO PRODUCT
      ELA          BIAS A TO LEFT
MPY1  ERA,CLE,SLA
      ADB NORML    SHIFT,TEST, AND ADD UPON NON- ZERO BIT
      ERB
      ISZ TEMP1    DONE
      JMP MPY1     NO
      ERA,CLE      YES, ADJUST FINAL RESULT
      ISZ MBY10    NEGATIVE RESULT
      JMP MPY2     NO
      CMB
      CMA,INA,SZA,RSS    YES,COMPLEMENT RESULT
      INB
MPY2  CLO
      ISZ MPY
      JMP MPY,I
```

```
*
*
* EXAMINE CHARACTER TO BE DECIMAL VALUE
*
* ENTER CHARACTER IN (A)
*
* RETURN P+1 CHARACTER IN (A)
*        P+2 DIGIT IN (A)
*
*
DECHK NOP
      STB STORE     SAVE (B)
      JSB SAVEE     SAVE (E) REGISTER
      LDB D72
      ADB A         CHARACTER IN (A)
      SSB,RSS       ASCII 72B OR GREATER
      JMP *+6       YES RETURN WITH CHARACTER
      ADB .10       NO, ASCII 60B OR GREATER
      SSB
      JMP *+3
      LDA B         COPY DIGIT INTO (A)
      ISZ DECHK
      JSB RSTRE     RESTORE (E)
      LDB STORE     RESTORE (B)
      JMP DECHK,I
```

```
*
*
*  SUBROUTINE TO DETERMINE REAL OR INTEGER
*
*  ENTER NUMBER STORED IN (A) AND (B)
*
*    DPFLG =  0 NO DECIMAL POINT
*          =  1 DECIMAL POINT
*
*    EFLG  = -1 NO E RECOGNIZED
*          =  0 E RECOGNIZED
*
*  RETURN P+1 REAL IN (A) AND (B)
*         P+2 INTEGER IN (A)
*
*
TYPCK NOP
      STA TEMP      SAVE A REGISTER
      LDA DPFLG
      CPA ZERO      COMPARE DEC. PT. FLAG FOR ZERO
      JMP TYPC1     NO DECIMAL POINT
      LDA TEMP      RESTORE (A)
      JMP TYPCK,I   RETURN WITH REAL NUMBER
TYPC1 LDA TEMP
      ISZ EFLG
      JMP TYPCK,I   REAL NUMBER
      JSB IFIX      CONVERT TO INTEGER
      ISZ TYPCK
      JMP TYPCK,I
```

```
*
*
* INTEGERIZE FLOATING POINT NUMBER
*
* ENTER NUMBER IN (A) AND (B)
*
* RETURN P+1 INTEGER IN (A)
*
*
IFIX   NOP
       STA MPY       SAVE (A)
       JSB FLUN
       CLO
       SSA           IF EXP NEGATIVE ERROR
       JMP NUMR5
       ADA M16       COMPUTE SHIFT COUNT
       SSA,RSS       IF EXP 16 OR MORE OVERFLOW
       JMP NUMR5
       CLE,SZB       SET (E) = 0 IF (B) = 0
       CME
       STA B         SAVE SHIFT COUNT IN (B)
       LDA MPY
IFX1   ISZ B         ANY MORE SHIFTS
       JMP IFX2      YES
       SEZ,SSA       IF NUMB LT 0 AND FRACT NOT 0
       INA           BUMP RESULT
       JMP IFIX,I
IFX2   SLA,ARS       SHIFT RIGHT AND TEST BIT LOST
       CCE
       JMP IFX1
*
*
* UNPACK LOW WORD OF NUMBER
*
*
FLUN   NOP           WORD IN (B)
       LDA B         (A) = (B)
       AND B377      EXTRACT EXPONENT IN (A)
       CMB           SUBTRACT OFF EXPONENT FROM
       ADB A         MANTISSA IN (B)
       CMB
       SLA,RAR       NEGATIVE EXPONENT
       IOR MSK4      YES, FILL IN LEADING BITS
       JMP FLUN,I    NO
```

```
*
*
* SUBROUTINE TWINT READS IN ONE OR TWO POSITIVE INTEGERS
*
* RETURN P+1 ONE INTEGER,  VALID TERMINATOR
*        P+2 ONE INTEGER,  INVALID TERMINATOR
*        P+3 TWO INTEGERS, VALID TERMINATOR
*        P+4 TWO INTEGERS, INVALID TERMINATOR
*
*
TWINT NOP
      CLA
      STA NUM1      INITIALIZE TWO INTEGERS TO ZERO
      STA NUM2
      JSB GTNUM     GET FIRST INTEGER
      STA NUM1      STORE VALUE
      JSB TRMCK     CHECK TERMINATOR
      JMP TWINT,I   FIRST RETRUN CONDITION
      CPA COMMA     COMMA
      RSS           YES
      JMP NUMR7     NO, BAD DATA
      JSB NTBLK     NEXT NON BLANK CHARACTER
      JMP NUMR7
      JSB DECHK     CHECK FOR DIGIT
      RSS           NO
      JMP TWIN1     YES, READ SECOND INTEGER
      JSB BCKSP     RETURN  CHARACTER TO BUFFER
      JMP TWIN2+1   SECOND RETURN
TWIN1 JSB BCKSP     RETURN CHARACTER TO BUFFER
      JSB GTNUM     READ INTEGER
      STA NUM2
      JSB TRMCK
      JMP TWIN2     THIRD RETURN CONDITION
      JSB BCKSP     RETURN CHARACTER TO BUFFER
      ISZ TWINT
TWIN2 ISZ TWINT
      ISZ TWINT
      JMP TWINT,I
```

```
*
*
* SUBROUTINE GTNUM CALLED BY TWINT TO INPUT AN INTEGER
*
* RETURN P+1 POSITIVE INTEGER IN (A)
*
*
GTNUM NOP
      JSB CONST      INPUT A CONSTANT
      SSA            NEGATIVE NUMBER
      JMP NUMR6      YES
      JSB TYPCK      NO, REAL OR INTEGER
      JMP NUMR6      REAL
      JMP GTNUM,I
*
*
* READ IN OCTAL INTEGER
*
* RETURN (A) OCTAL INTEGER
*
*
OCTIN NOP
      JSB NTBLK      NEXT NON BLANK CHARACTER
      JMP NUMR1      NO DATA FOUND
      CLB,CLE
      STB NUM1       INITIALIZE
      INB
      STB SIGN       SET SIGN POSITIVE
```

```
      CPA PLUS      POSITIVE SIGN
      JMP OCTN1     YES
      CPA MINUS     NO, MINUS SIGN
      CCB,RSS       YES
      JMP OCTN2     NO
      STB SIGN      RECORD MINUS SIGN
OCTN1 JSB GETCR
      JMP OCTN3
OCTN2 JSB OCTCK     OCTAL DIGIT
      JMP OCTN3     NO
      CCE
      LDB M3
      STB TEMP3
      LDB NUM1
      RBL,SLB       CHECK FOR OVERFLOW
      JMP NUMR5     OVERFLOW
      ISZ TEMP3
      JMP *-3
      ADA B         ACCEPT VALUE  ADD NEW DIGIT
      STA NUM1
      JMP OCTN1     GET NEXT CHARACTER
OCTN3 SEZ,RSS
      JMP NUMR2     SOLITARY SIGN
      JSB BCKSP     RETURN CHAR TO BUFFER
      LDA NUM1
      LDB SIGN
      SSB           NEGATIVE SIGN
      CMA,INA       YES, COMPLEMENT
      JMP OCTIN,I
```

```
*
*
* SUBROUTINE OCTCK TO CHECK FOR OCTAL DIGIT
*
* ENTER CHARACTER IN (A)
*
* RETURN P+1 CHARACTER IN (A)
*        P+2 OCTAL DIGIT IN (A)
*
*
OCTCK NOP
      JSB SAVEE    SAVE (E)
      LDB D70
      ADB A        CHARACTER IN (A)
      SSB,RSS      CHARACTER 70B OR GREATER
      JMP *+6      YES, RETURN WITH CHARACTER
      ADB .8       NO, ASCII 60B OR GREATER
      SSB
      JMP *+3      NO
      LDA B        YES LOAD DIGIT INTO (A)
      ISZ OCTCK
      JSB RSTRE    RESTORE (E)
      JMP OCTCK,I
```

```
*
*
* INPUT DECIMAL INTEGER OR OCTAL INTEGER FOLLOWED BY A B
*
* RETURN P+1 FIRST CHARACTER NOT A NUMBER
*        P+2 INTEGER IN (A)
*
*
NUMBR NOP
      JSB NTBLK    NEXT NON BLANK CHAR
      JMP NUMR1    NO DATA FOUND
      CLB,INB
      STB SIGN     SET SIGN POSITIVE
      CPA PLUS     POSITIVE SIGN
      JMP NUMB1    YES
      CPA MINUS    NO, NEGATIVE SIGN
      CCB,RSS      YES
      JMP NUMB2
      STB SIGN     RECORD SIGN
NUMB1 JSB GETCR
      JMP NUMR2    SOLITARY SIGN
NUMB2 JSB DECHK    DECIMAL DIGIT
      JMP NUMBR,I  FIRST RETURN
      CLO
      CLB
      STB NUM1     DECIMAL
      STB NUM2     OCTAL
      STB TEMP4    OCTAL ERROR FLAG
NUMB3 ADB A
      SOC          DECIMAL OVERFLOW
      JMP NUMR5    YES
      ADB NUM1     NO, ADD IN DIGIT
      STB NUM1
      LDB M8       CHECK FOR OCTAL DIGIT
      ADB A
      SSB,RSS      OCTAL DIGIT
      ISZ TEMP4    NO, RECORD ERROR
```

```
      ADA NUM2
      STA NUM2
      SOC
      JMP NUMR5
      JSB GETCR
      JMP NUMB6    END OF LINE TERMINATION
      JSB DECHK    DECIMAL DIGIT
      JMP NUMB4    NO
      LDB M3       YES, SHIFT OCTAL RIGHT 3 PLACES
      STB TEMP3    TEMPORARY COUNTER
      LDB NUM2
      RBL,SL3      OCTAL OVERFLOW
      JMP NUMR5    YES
      ISZ TEMP3
      JMP *-3
      STB NUM2
      LDB NUM1     MULTIPLY DECIMAL BY 10 USING
      BLS,BLS      SHIFTS AND ADDITION
      ADB NUM1
      STB NUM1
      JMP NUMB3
NUMB4 CPA BE       OCTAL FLAG
      RSS          YES
      JMP NUMB6    NO
      LDA TEMP4
      SZA          OCTAL ERROR
      JMP NUMR5    YES
      LDA NUM2     RETURN OCTAL
      JMP NUMB7
NUMB6 JSB BCKSP    RETURN TERMINATOR TO BUFFER
      LDA NUM1     RETURN DECIMAL
NUMB7 LDB SIGN
      SSB          NEGATIVE SIGN
      CMA,INA      YES, COMPLEMENT
      ISZ NUMBR
      JMP NUMBR,I
```

```
*
*
*
* ERROR MESSAGES
*
*
*
NUMR1 LDA .22
      LDB *+2
      JMP NUMER
*
      DEF *+1
      ASC 11,NO OPERAND DATA FOUND
*
*
NUMR2 LDA .14
      LDB *+2
      JMP NUMER
*
      DEF *+1
      ASC 7,SOLITARY SIGN
*
*
NUMR3 LDA .14
      LDB *+2
      JMP NUMER
*
      DEF ERR1      BAD DATA INPUT
*
*
NUMR4 LDA .18
      LDB *+2
      JMP NUMER
*
      DEF *+1
      ASC 9,ERROR IN EXPONENT
*
*
NUMR5 LDA .16
      LDB *+2
      JMP NUMER
```

```
*
      DEF *+1
      ASC 8,INTEGER OVERLFOW
*
*
NUMR6 LDA .26
      LDB *+2
      JMP NUMER
*
      DEF *+1
      ASC 13,POSITIVE INTEGER EXPECTED
*
*
NUMR7 LDA .24
      LDB *+2
      JMP NUMER
*
      DEF *+1
      ASC 12,BAD DATA FOLLOWS INTEGER
*
*
NUMR8 LDA .24
      LDB *+2
      JMP NUMER
*
      DEF *+1
      ASC 12,REAL NUMBER OUT OF RANGE
*
*
* PRINT ERROR MESSAGE AND RE ENTRY REQUEST
*
* DURING INITIALIZATION RETURN TO CALLING ROUTINE
* OTHERWISE RETURN TO SYSTEM CONTROLLER
*
*
NUMER JSB ERROR
      LDA GRTFG
      SSA                GREET FLAG
      JMP GRTER,I        JUMP INTO GREET ROUTINE
      JMP CNTRL,I
```

```
*
*
* SUBROUTINE TO READ MEMORY REFERENCE OPERANDS AS WELL FOR OTHER
* ASSOCIATED FUNCTIONS
*
* A MEMORY REFERENCE OPERAND IS RESTRICTED TO
*
*        (+LABEL)(+/-VALUE)(,I)
*
* THE LABEL MAY BE SUBSTITUTED BY THE PROGRAM LOCATION
* COUNTER SYMBOL (*)
*
*
OPREC NOP
OPRC1 JSB NUMBR     READ IN INTEGER
      JMP OPRC4     FIRST CHAR NOT A NUMBER
      CCB
      STB NUMFG     SET OPERAND NUMBER FLAG
      STA OPNUM     STORE VALUE
OPRC2 JSB TRMCK     TERMINATION
      JMP OPRC8     YES, CHECK FOR LABEL
      CPA PLUS      NO, POSITIVE SIGN
      JMP OPRC3     YES, SET SIGN
      CPA MINUS     NO, MINUS SIGN
      RSS           YES
      JMP OPRC7
      CCB,RSS
OPRC3 CLB,INB
      STB SIGN      SET SIGN
      LDB OPLBL     LABEL FLAG
      SZB,RSS       SET
      JMP OPRC5     NO, READ LABEL
      LDB NUMFG
      SZB
      JMP OPER1     ERROR IN OPERAND
      JSB BCKSP     RETURN SIGN TO BUFFER
      JMP OPRC1     READ INTEGER
```

```
OPRC4 JSB LETPR    LETTER OR PERIOD
      JMP OPRC6    NO, SPECIAL CHARACTER
      RSS          YES, VALID CHARACTER
OPRC5 LDA M1       NO CHARACTER PREVIOUSLY READ
      LDB SIGN     SIGN BEFORE LABEL
      SSB          NEGATIVE SIGN
      JMP OPER2    YES, ERROR
      LDB LAB2     STORE ADDRESS FOR LABEL
      JSB LABRD
      JMP OPRC6    ILLEGAL CHARACTER BEGINS LABEL
      CLB,INB
      STB OPLBL    SET OPERAND LABEL FLAG
      JMP OPRC2
OPRC6 LDB DMPFG    DUMP FLAG
      SZB          DUMP OPERATION
      JMP OPER1    YES, ERROR
      CPA STAR     NO, ASTERISK
      CCA,RSS      YES
      JMP OPER1    NO, ERROR
      LDB SIGN
      SSB
      JMP OPER3    MINUS SIGN PRECEDES ASTERISK
      STA OPLBL    SET P.L.C. INDICATOR
      JMP OPRC2
OPRC7 LDB DMPFG
      SZB
      JMP OPER1
      CPA COMMA    COMMA
      RSS          YES
      JMP OPER1    NOP,ERROR
      JSB GETCR
      JMP OPER1    ILLEGAL CHAR IN OPERAND
      CPA I        INDIRECT FLAG
      CCA,RSS      YES
      JMP OPER1    ILLEGAL CHAR IN OPERAND
      STA IDRCT
```

```
      JSB TRMCK    END OF OPERAND
      RSS
      JMP OPER1    ILLEGAL TERMINATION
OPRC8 LDA IDRCT    INDIRECT FLAG SET
      SZA
      JMP OPR10    YES
      LDA OPLBL    CHECK FOR LABEL
      SZA          LABEL FOUND
      JMP OPREC,I  YES, RETURN
      LDA INSNM    NO LABEL FOUND
      CPA ZERO
      JMP OPREC,I
      ADA M8       MEMORY REFERENCE TYPE INSTRUCTION
      SSA,RSS
      JMP OPREC,I  NO
      LDA OPNUM    YES, CHECK RANGE
      SSA
      JMP OPRC9    NEGATIVE
      ADA D100
      SSA
      JMP OPREC,I  RETURN VALUE IN RANGE
OPRC9 LDA .26
      LDB *+2
      JMP ERCAL
*
      DEF ERR3     OPERAND VALUE OUT OF RANGE
*
OPR10 LDB INSNM    INSTRUCTION NUMBER
      ADB M8       MEMORY REFERENCE
      SSB
      JMP OPRC8+3  YES
      LDA .38      NO, INDRCT REFERENCE NOT ALLOWED
      LDB OPRM1
      JSB BPLN
```

```
       LDA .38
       LDB OPRM2
       JSB WRITE,I  PRINT ERROR MESSAGE
       JSB REENT    RE ENTRY REQUEST
       JMP CNTRL,I  RETURN TO CONTROLLER
*
OPRM1  DEF *+1
       ASC 19,INDIRECT REFERENCE PERMITTED ONLY WITH
*
OPRM2  DEF *+1
       ASC 19,MEMORY REFERENCE AND DEF INSTRUCTIONS
*
*
OPER1  LDA .28
       LDB *+2
       JMP ERCAL
*
       DEF ERR4     ILLEGAL OPERAND TERMINATION
*
*
OPER2  LDA .26
       LDB *+2
       JMP ERCAL
*
       DEF *+1
       ASC 13,MINUS SIGN PRECEDES LABEL
*
*
OPER3  LDA .28
       LDB *+2
       JMP ERCAL
*
       DEF *+1
       ASC 14,MINUS SIGN PRECEDES ASTERISK
```

```
*
*
*
*
* READ A LABEL
*
* A LABEL MAY HAVE ONE TO FIVE CHARACTERS CONSISTING
* OF A THROUGH Z, 0 THROUGH 9, AND THE PERIOD. THE
* FIRST CHARACTER MUST BE ALPHABETIC OR THE PERIOD.
*
* ENTER (A) ≥ 0 FIRST CHARACTER IN (A)
*       (A) < 0 FIRST CHARACTER NOT READ
*
*       (B) ADDRESS FOR LABEL
*
* RETURN P+1 FIRST CHARACTER NOT VALID
*        P+2 LABEL SUCCESSFULLY READ
*
LABRD NOP
      STB ADDR      SAVE ADDRESS
      LDB M5
      STB TEMP3     CHARACTER COUNT
      CLE,SSA,RSS
      JMP LABR1
      JSB NTBLK     FIRST NON BLANK CHARACTER
      JMP LBER1     RETURN NO LABEL FOUND
      JSB LETPR     LETTER-PERIOD CHECK
      JMP LABR2+1   BAD CHARACTER BEGINNING LABEL
      SEZ,RSS
LABR1 ALF,ALF       SHIFT CHARACTER
      IOR ADDR,I
      STA ADDR,I    STORE CHARACTER
      SEZ,CME
```

```
      ISZ ADDR     ADVANCE BUFFER POINTER
      ISZ TEMP3
      RSS
      JMP LABR2    FIVE CHARACTERS READ
      JSB GETCR    NEXT CHARACTER
      JMP LABR3
      JSB LETPR    LETTER-PERIOD
      RSS          NO
      JMP LABR1-1  YES, STORE CHARACTER
      JSB DECHK    DECIMAL NUMBER
      JMP LABR3    NO
      ADA .48      YES, BUT RETAIN AS CHARACTER
      JMP LABR1-1
LABR2 ISZ LABRD
      JMP LABRD,I
LABR3 JSB BCKSP    RETURN TERMINATOR TO BUFFER
      JMP LABR2
*
*
LBER1 LDA .14
      LDB *+2
      JMP ERCAL
*
      DEF ERR9     NO LABEL FOUND
```

```
*
*
*
* CHECK FOR LETTER OR PERIOD
*
* RETURN P+1 CHARACTER IN (A)
*        P+2 LETTER OR PERIOD IN (A)
*
*
LETPR NOP
      JSB SAVEE    SAVE (E) REGISTER
      CPA PRIOD    PERIOD
      JMP *+7      YES
      LDB A        NO
      ADB D133
      SSB,RSS      ASCII 133B OR GREATER
      JMP *+4
      ADB .26      NO ASCII 101B
      SSB,RSS      OR GREATER
      ISZ LETPR    YES
      JSB RSTRE    RESTORE (E) REGISTER
      JMP LETPR,I
```

```
*
*
*
*
* CHECK ADDRESS RANGE IN DATA BUFFER AREA
*
* ENTER (A) = ADDRESS TO BE CHECKED
*
* RETURN ADDRESS IN (A)
*
*
DATRG NOP
      LDB XDATA     LOWER BOUND OF DATA AREA
      CMB,INB
      ADB A
      SSB           LOWER BOUND ERROR
      JMP DTRG1
      LDB YDATA     UPPER BOUND OF DATA AREA
      CMB
      ADB A
      SSB,RSS       UPPER BOUND ERROR
      JMP DTER1
      JMP DATRG,I
DTRG1 LDB D100
      ADB A
      SSB
      JMP DATRG,I
*
*
DTER1 LDA .30
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 15,ADDRESS BEYOND PROGRAM BOUNDS
```

```
*
*
*
*
* SCAN USER PROGRAM FOR FORWARD REFERENCES
*
* STORE THE FIRST 99 FORWARD REFERENCES IN THE INPUT
* AND DATA STORE BUFFERS.  REPLACE THE FORWARD REFERENCES
* BY A JUMP TO A ROUTINE THAT TERMINATES EXECUTION AND
* WARNS THE USER ABOUT FORWARD REFERENCES
*
*
CDSCN NOP
      LDA M100
      STA TEMP
      STA TEMP3
      LDB BUFA      ADDR OF BUFFER TO HOLD FWD REF
      STB TEMP1
      STB TEMP2
      CLB
      STB TEMP1,I   CLEAR BUFFER
      ISZ TEMP1
      ISZ TEMP
      JMP *-3
      ISZ TEMP3
*
      LDA XUSRP     FIRST LOCATION IN PROGRAM AREA
      STA TEMP
*
CDSN1 LDA TEMP,I    RETRIEVE INSTRUCTION
      SSA,RSS       BIT 15 SET
      JMP CDSN2     NO
      AND B2000     YES, I/O INSTRUCTION
      SZA
      JMP CDSN4     YES
      LDA TEMP,I
      AND B700      REGISTER REFERENCE
      SZA,RSS
      JMP CDSN4
```

```
      ISZ TEMP       EXTENDED ARITH MEMORY REF
      LDA TEMP,I     EXAMINE ADDRESS
      SSA            INDIRECT BIT SET
      JMP CDSN4      YES, DEFINED REFERENCE
      ADA D100
      CLE
      JMP CDSN3
*
CDSN2 AND B0700      MEMORY REFERENCE
      SZA,RSS
      JMP CDSN4      NO
      LDA TEMP,I     YES, RETRIEVE INSTRUCTION
      AND B2000      CURENT PAGE BIT SET
      SZA
      JMP CDSN4      YES
      LDA TEMP,I
      AND B1777
      ADA D100
      CCE
CDSN3 SSA            FORWARD REFERENCE
      JMP CDSN4      NO
      CLA,SEZ,RSS    YES
      CCA
      ADA TEMP
      STA TEMP1      ADDRESS OF FORWARD REF
      LDA TEMP1,I    GET INSTRUCTION
      STA TEMP2,I    SAVE INSTRUCTION
      LDA MPPEX      FORCE PRINTING OF WARNING MESSAGE
      STA TEMP1,I    DURING EXECUTION
      ISZ TEMP2
      ISZ TEMP3
      RSS
      JMP CDSCN,I    FIRST 99 FWD REF SAVED
CDSN4 ISZ TEMP       NEXT ADDRESS
      LDA TEMP
      CPA ZUSRP      END OF USER PROGRAM
      JMP CDSCN,I    YES, RETURN
      JMP CDSN1      NO
```

```
*
*
* READ IN USER DEFINED STATEMENT NUMBERS
*
* RETURN P+1 ERROR, RE ENTRY NECESSARY
*        P+2 STATEMENT NUMBERS ACCEPTED AND STORED
*
*
SQNCE NOP
      JSB RDCOM
      RSS           NOTHING ENTERED
      JSB TWINT
      NOP
      RSS           BAD DATA INPUT
      RSS           TWO POSITIVE INTEGERS READ IN
      JMP SQER1
      LDB M1001
      ADB NUM1      CHECK RANGE OF FIRST
      SSB,RSS
      JMP SQER2     TOO LARGE
      LDB NUM2      IN RANGE
      SZB,RSS       ZERO
      JMP SQER2     YES, ERROR
      ADB M26       NO
      SSB,RSS       TOO LARGE
      JMP SQER2     YES
      LDA NUM1      BOTH NUMBERS IN RANGE
      STA FSTMT     FIRST STATEMENT NUMBER
      LDB NUM2
      STB STINC     STATEMENT NUMBER INCREMENT
      CMB,INB
      ADA B
      STA CUSTN     CURRENT USER STATEMENT NUMBER
      ISZ SQNCE
      JMP SQNCE,I
```

```
*
*
SQER1 LDA .14
      LDB *+2
      JMP SQER3
*
      DEF ERR1      BAD DATA INPUT
*
*
SQER2 LDA .30
      LDB *+3
SQER3 JSB ERROR
      JMP SQNCE,I  RETURN ON  ERROR
*
      DEF ERR2      STATEMENT NUMBER OUT OF RANGE
```

```
*
*
*
      ORG 10000B
*
*
* SET AND STORE INSTRUCTION (DATA OR MACHINE CODE)
* IN APPROPRIATE PROGRAM AREA
*
* EVALUATE ALL MEMORY REFERENCE OPERANDS
*
* ENTER (A) > 0 MACHINE INSTRUCTION
*       (A) < 0 DATA
*
*
SETCD NOP
      SSA,RSS        TYPE OF ASSEMBLY
      JMP STCD1      MACHINE INSTRUCTION
      LDA ZADD       DATA, ADDRESS OF ASSEMBLY
      SZA            ADDRESS ALREADY SET
      JMP SETCD,I    YES, RETURN
      LDA ZDATA      NO, NEXT FREE AREA IN DATA TABLE
      STA ZADD       ADDRESS IN ASSEMBLED CODE
      ADA LENTH      LENGTH OF ASSEMBLY
      STA ZDATA      RESET DATA TABLE POINTER
      JSB DTFL,I     CHECK FOR DATA TABLE OVERFLOW
      LDB ZADD       ADDRESS FOR DATA STORE
      JSB DTSET      MOVE DATA INTO DATA AREA
      JMP SETCD,I
*
*
*
* STORE ASSEMBLED MACHINE INSTRUCTIONS
*
*
*
STCD1 LDB ZUSRP      NEXT FREE AREA IN ASSEM CODE
      STB ZADD       SET ADDRESS IN ASSEM CODE
      LDA INSNM      INSTRUCTION NUMBER
      ADA M6
      SSA,RSS        MEMORY REFERENCE INSTRUCTION
      JMP STCD3      YES
```

```
STCD2 LDA ASMBY     NO, GET ASSEMBLED CODE
      JSB STRCD     STORE CODE
      JMP SETCD,I
*
* CLEAR UP OPERAND FOR MEMORY REFENCE INSTRUCTIONS
*
STCD3 LDA OPLBL     OPERAND LABEL PRESENT
      SZA           YES
      JMP STCD4
      JSB DETLN     NO, GET INSTR SKELETON
      ADA OPNUM     ADD OPERAND IF PRESENT
      JSB IDIRT     CHECK FOR INDIRECT FLAG
      JMP STCD2+1   STORE ASSEMBLED INSTRUCTION
*
* LABEL OR ASTERISK IS PRESENT
*
STCD4 SSA,RSS
      JMP STCD5     LABEL
      LDA ADDR1     ASTERISK, PLC REFERENCE
      JSB IDIRT     ADDRESS IN SCB
      JSB STPLC     STORE PLC REFERENCE
      JSB DETLN     GET INSTRUCTION
      ADA WMOVE     ADD TERM TO SIGNAL FORWARD REFERENCE
      JMP STCD2+1
*
*
* EXAMINE LABEL
*
*
STCD5 LDB LAB2      RETRIEVE LABEL ADDRESS
      JSB LOKP,I    SYMBOL TABLE LOOK UP
      STB BCKSP     SAVE SYMBOL TABLE ADDR
      SZA,RSS       LABEL EXIST
      JMP STCD8     NO
      SSA           LABEL DEFINED
      JMP STCD9     NO
      LDB OPNUM     OPERAND NUMBER
      SZB
      JMP TCD11     STORE OPERAND IN SST
      JSB DATAD     UPDATE FOR DATA ADDRESS
```

```
       JSB IDIRT
       LDB ZUSRP     ADDRESS FOR INSTRUCTION
       ADB D340      ADDR POSITION IN ADDR BLOCK
       STA B,I       SET ADDRESS IN ADDR AREA
       STB ADDR3     SAVE ADDR IN ADDR BLOCK
       JSB DETLN     LENGTH OF ASSEMBLY
       SEZ
       JMP STCD6     TWO WORD ASSEMBLY
       SWP           ONE WORD ASSEMBLY
       AND B1777     GET RELATIVE ADDRESS
       ADA B
       ADA CPIB      CURRENT PAGE  INDIRECT BIT
       JMP STCD2+1
STCD6  ADA ADDR3     OPERAND ADDRESS
       IOR MNEG      INDIRECT BIT
       JMP STCD2+1
*
* LABEL DOES NOT EXIST
*
STCD8  CCE
       JSB STLBL     STORE LABEL IN SYM TBL
STCD9  LDA OPNUM     OPERAND NUMBER
       SZA
       JMP TCD11     YES
       LDB BCKSP     SYM TBL ADDR OF LABEL
       ADB .3
       LDA IDRCT     CHECK FOR INDIRECT REFERENCE
       SZA
       INB
       LDA B,I       ADDRESS OF LAST REFERENCE
       STA RDCOM
       STB BCKSP     SAVE ADDR IN SYM TBL
       JSB DETLN     DETERMINE LENGTH OF ASSEMBLY
       STA ASMBY     SAVE ASSEMBLED INSTRUCTION
       LDA ZUSRP
       AND B1777     GET RELATIVE ADDRESS
       STA BCKSP,I   SET FORWARD REF IN SYM TBL
       LDA ASMBY     SKELETON INSTRUCTION
       ADA RDCOM     ADD PREV UNDEF REFERENCE
       JMP STCD2+1
```

```
*
*
* LABEL WITH OPERAND NUMBER
*
*
TCD11 CLA             VARIABLE TO CONTROL PRINTING
      STA GETCR       OF WARNING MESSAGE
      STA LINK        LINK FLAG FOR SST
      LDB BCKSP       SYM TBL ADDR OF LABEL
      ADB .5          LINK TO SST
      STB RDCOM       SAVE LINK CHARACTER
      LDA B,I         PREVIOUS SST ENTRIES
      SZA
      JMP TCD16       YES
      LDB XSST        ADDRESS OF SST
      RSS
TCD12 ADB .4
      LDA YSST        UPPER BOUND OF SST
      CMA,INA
      ADA B
      SSA             TABLE OVERFLOW
      JMP TCD13       NO
      LDA .32         YES
      LDB TCDR1
      JMP TBLOV
TCD13 ADA .40
      SSA             TABLE NEAR OVERFLOW
      JMP TCD14       NO
      LDA GETCR       YES, WARNING PREVIOUSLY PRINTED
      SZA
      JMP TCD14       YES
      LDA .48         PRINT WARNING TO USER REGARDING
      LDB TCDR2       TABLE OVERFLOW
      JSB BPLN
      STA GETCR       SET FLAG TO SUPPRESS MESSAGE
TCD14 LDA B,I         AREA OCCUPIED
```

```
       SZA
       JMP TCD12     YES
       LDA LINK      NO, LINK SET
       SZA
       JMP *+3
       STB RDCOM,I   SET LINK ADDR IN SYM TBL
       RSS
       STB ADDR2,I   LINK TO PREV SST BLOCK
       LDA OPNUM     SET OPERAND VALUE IN SST
       STA B,I
       STB ADDR3     SAVE ADDRESS
       LDA  BCKSP    ADDR OF SYM TBL ENTRY
       JSB IDIRT     SET INDIRECT FLAG WITH THIS ADDR
       ISZ ADDR3     ADVANCE ADDRESS
       STA ADDR3,I   SET LINK TO SYMBOL TABLE
       ISZ ADDR3     ADDRESS OF LAST FORWARD REF
       LDA ADDR3,I
       STA GETCR     VALUE OF LAST FORWARD REF
TCD15  JSB DETLN
       STA ASMBY
       LDA ZUSRP
       AND B1777     RELATIVE ADDRESS
       STA ADDR3,I   SET ADDRESS IN SST
       LDA GETCR     VALUE OF LAST FORWARD REFERENCE
       ADA ASMBY
       JMP STCD2+1
*
* PREVIOUS ENTRIES FOR THIS LABEL
*
*         (A) CONTAINS LINK FROM SYMBOL TABLE
*
TCD16  LDB A,I       VALUE IN SST
       CPB OPNUM     MATCH
       RSS
       JMP TCD18     NO
       STA REENT     SAVE ADDRESS
```

```
      INA
      LDB A,I      GET WORD HOLDING INDIRECT FLAG
      CLE,ELB      INDIRECT BIT FLAG IN (E)
      LDB IDRCT    INDIECT FLAG ON OPERAND
      BLS          CLEAR BIT 0
      SEZ
      INB
      SZB,RSS
      JMP TCD17
      SSB,SLB,RSS  MATCH
      JMP TCD18-1  NO MATCH
TCD17 ADA .1
      LDB A,I      ADDR OF PREV REF
      STA ADDR3    SAVE ADDRESS IN SST
      STB GETCR    SAVE VALUE OF PREV REFERENCE
      JMP TCD15
*
* EXAMINE NEXT LINK IN SST
*
      LDA REENT
TCD18 ADA .3       ADDR OF LINK WORD
      STA ADDR2    SAVE ADDRESS
      LDA A,I      GET LINK ADDR
      SZA          FURTHER ENTRIES
      JMP TCD16    YES, LOOK AT NEXT ENTRY
      CCA          NO  SET UP LINK FOR SST
      STA LINK
      JMP TCD12-2
*
*
TCDR1 DEF *+1
      ASC 16,COMPOUND OPERAND TABLE OVERFLOW
*
*
TCDR2 DEF *+1
      ASC 24,COMPOUND OPERAND TABLE NEAR OVERFLOW, LIMIT USE
```

```
*
*
* DETERMINE LENGTH OF ASSEMBLY FOR MEMORY REFERENCE
* INSTRUCTION
*
*
DETLN NOP
      LDA ASMBY    RETRIEVE INSTR SKELETON
      CLE,SSA,RSS  TWO WORD ASSEMBLY
      JMP *+3      NO
      JSB STRCD    YES, STORE WORD
      CLA,CCE      SET INDICATOR
      JMP DETLN,I
*
*
* ALLOCATE STORAGE SPACE FOR STORING PROGRAM
* STATEMENT IN SOURCE CODE BLOCK
*
*
ASMBL NOP
      LDA SRCNT    CHAR LENGTH OF INPUT STRING
      STA B
      BLF,BLF      SHIFT CHAR COUNT
      INA
      ARS          NUMBER OF WORDS
      STA SRCNT    NUM OF WORDS TO BE MOVED TO SCB
      ADA .6       LENGTH OF ENTRY TO SCB
      STA TEMP3    RETAIN NUMBER OF WORDS
      ADB A
      STB LNTH2    INPUT LENGTH FOR SCB
```

```
*
* SCAN FREE SPACE AREA FIRST BEFORE ALLOCATING NEXT
* AREA IN SCB
*
      LDB XFRSP
ASMB1 LDA B,I
      SZA             ENTRY
      JMP ASMB4       YES
      ADB .2          NO
      LDA YFRSP       UPPER BOUND OF FREE SPACE AREA
      CMA,INA
      ADA B
      SSA             OVERFLOW IN FREE SPACE AREA
      JMP ASMB1       NO
ASMB2 LDA NEXT        YES, NEXT LOCATION IN SCB
      LDB TEMP3       NUMBER OF WORDS IN SCB ENTRY
      ADB A
      STB NEXT        PREPARE FOR NEXT ENTRY
      ADB M1          CHECK FOR TABLE OVERFLOW
      CMB,INB
      ADB YSCB        UPPER BOUND OF SOURCE CODE BLOCK
      SSB,RSS         OVERFLOW
      JMP ASMB3       NO
      LDA .30         YES
      LDB ASMR1
      JMP TBLOV
*
ASMB3 ADB M125        TABLE NEAR OVERFLOW
      SSB,RSS
      JMP ASMB5
      LDA .52
      LDB ASMR2
      JSB BPLN
      JMP ASMB5
```

```
*
ASMB4 CMA,INA       BLOCK IN FREE SPACE
      ADA TEMP3     LARGE ENOUGH TO HOLD EDIT ENTRY
      SSA,RSS
      JMP ASMB1+3   NO
      ADA .12       AREA REMAINING LARGE ENOUGH
      CCE,SSA,RSS   TO HOLD FURTHER ENTRIES
      CLA,CLE       NO
      STA B,I       CLEAR ENTRY FROM FREE SPACE AREA
      INB           YES
      LDA B,I       GET ADDRESS IN SCB
      RSS           SKIP NEXT INSTR (E) MAY BE SET
ASMB5 CLE           INHIBITS OPERATION OF FREE SPACE
      STA ADDR1     SAVE ADDR IN SCB
      SEZ,RSS       CHANGE REQUIRED IN FREE SPACE
      JMP ASMBL,I   NO, RETURN
      LDA TEMP3     LENGTH OF ENTRY IN SCB
      ADA B,I       ADD ADDRESS IN FREE SPACE
      STA B,I       STORE NEW ADDRESS
      ADB M1        BACK UP ADDRESS
      LDA TEMP3
      CMA,INA       AVAILABLE SPACE
      ADA B,I
      STA B,I       STORE NEW LENGTH
      JMP ASMBL,I
*
*
ASMR1 DEF *+1
      ASC 15,SOURCE PROGRAM TABLE OVERFLOW
*
*
ASMR2 DEF *+1
      ASC 26,PROGRAM APPROACHES IMPOSED LIMIT, BEGIN EXECUTION
```

```
*
*
* STORE DATA BUFFER IN PROGRAM DATA AREA
*
* ENTER (B) = ADDRESS FOR DATA STORAGE
*
*
DTSET NOP
      LDA DATBF     ADDR OF DATA BUFFER
      STB RDCOM     ADDRESS FOR DATA
      LDB IDRCT     BSS INSTRUCTION
      SZB
      LDA B0700     YES, ADDRESS IN NON EXISTNAT MEMORY
      STA TEMP3     SAVE BUFFER ADDRESS
      LDA LENTH     LENGTH OF ASSEMBLY
      CMA,INA
      STA TEMP4
DTST1 LDA RDCOM     FETCH ADDRESS
      ADA B400      ADD ADDRESS POINTER
      STA RDCOM,I   STORE ADDRESS
      LDB TEMP3,I   RETIREVE VALUE
      STB A,I       STORE VALUE AT APPROPRIATE ADDR
      ISZ TEMP3
      ISZ RDCOM     ADVANCE BUFFER POINTERS
      ISZ TEMP4
      JMP DTST1
      JMP DTSET,I
```

```
*
*
* STORE LABEL IN SYMBOL TABLE
*
*
* ENTER (A) GT 0 ADDRESS IN ASSEMBLED CODE
*           = 0 NON EXISTANT LABEL
*       (B) = ADDRESS OF LABEL IN SYMBOL TABLE
*       (E) ADDRESS OF BUFFER HOLDING LABEL
*           = 0 LAB1
*           = 1 LAB2
*
*
* THE SYMBOL TABLE HAS BEEN IMPLEMENTED TO HOLD NO MORE
* THAN 125 LABELS.  AN ATTEMPT TO INTRODUCE MORE THAN
* 125 WILL CAUSE THE ASSEMBLER TO HALT WITH THE USER S
* PROGRAM LOST
*
* EACH SYMBOL TABLE ENTRY IS SIX WORDS IN LENGTH
*
* WORD 1   FIRST TWO CHARACTERS OF LABEL
*
* WORD 2   THIRD AND FOURTH CHARACTER IN LABEL
*
* WORD 3   BITS 8-15 LAST CHARACTER
*          BIT  0 = 0 UNDEFINED LABEL
*                   1   DEFINED LABEL
*
*
* WORD 4 AND 5 HAVE DIFFERENT USES IF THE LABEL IS OR
* IS NOT DEFINED
*
* UNDEFINED WORD 4  ADDRESS TO LAST DIRECT FORWARD REF
*           WORD 5  ADDRESS TO LAST INDIRECT FORWARD REF
*
*   DEFINED WORD 4  LABEL ADDRESS IN ASSEMBLED CODE
*           WORD 5  LABEL ADDRESS IN SCB
*
* WORD 6   LINK TO SPECIAL SYMBOL TABLE FOR COMPOUND
*          OPERANDS
*
*
```

```
STLBL NOP
      STA TEMP3      SAVE (A)
      ISZ LBCNT      INCREMENT LABEL COUNT
      LDA .115
      CMA,INA
      ADA LBCNT
      SSA            SYMBOL TABLE NEARLY FULL
      JMP STBL1      NO
      LDA .42
      LDB STBLR
      JSB BPLN
STBL1 LDA .3
      STA SORCE      NUMBER OF WRODS TO BE MOVED
      LDA LAB1
      SEZ
      LDA LAB2       GET PROPER LABEL ADDRESS
      JSB WMOVE      MOVE THE LABEL
      LDA TEMP3      LABEL DEFINED
      SZA,RSS
      JMP STLBL,I    NO, RETURN SET NO FLAGS
      STA TEMP3      YES
      LDA B,I
      ADA .1         DEFINE LABEL/ DEFINED REFERENCE
      STA B,I
      INB
      LDA TEMP3      ADDR IN ASSEMBLED CODE
      STA B,I        STORE ADDR IN ASSEM CODE
      INB
      LDA ADDR1      ADDRESS IN SCB
      STA B,I
      JMP STLBL,I
*
STBLR DEF *+1
      ASC 21,SYMBOL TABLE NEARLY FULL, BEGIN EXECUTION
```

```
*
*
* STORE INSTRUCTION IN PROGRAM AREA
*
* ENTER (A) ASSEMBLED INSTRUCTION
*
*
STRCD NOP
      STA ZUSRP,I   STORE INSTRUCTION
      ISZ ZUSRP     NEXT LOCATION PROGRAM AREA
      JSB STRCK
      JMP STRCD,I
*
*
* CHECK USER PROGRAM AREA FOR OVERFLOW
*
*
STRCK NOP
      LDB YUSRP     UPPER BOUND OF PROGRAM AREA
      CMB,INB
      ADB ZUSRP
      SSB           OVERFLOW
      JMP STRC1     NO
      LDA .24       YES
      LDB STRER
      JMP TBLOV
*
* PROMPT USER IF PROGRAM AREA IS ABOUT TO OVERFLOW
*
STRC1 ADB .15
      SSB
      JMP STRCK,I
      LDA .52
      LDB ASMR2
      JSB BPLN
      JMP STRCK,I
*
*
STRER DEF *+1
      ASC 12,PROGRAM BUFFER OVERFLOW
```

```
*
*
* STORE PLC REFERENCE
*
* ENTER (A) SCB ADDRESS WITH INDIRECT BIT SET IF NEEDED
*
*
* EACH PLC REFERENCE IS STORED IN TWO WORDS IN THE PLC
* TABLE
*
* WORD 1   SCB ADDRESS WITH BIT 15 SET FOR INDIRECT
*          REFERENCE
*
* WORD 2   NUMERIC VALUE IN OPERAND
*
* NO ATTEMPT WILL BE MADE TO DEFINE THE PLC REFERENCE
* UNTIL EXECUTION.  BEFORE EXECUTION THE PLC TABLE
* WILL BE SCANNED AND ALL POSSIBLE REFERENCES WILL BE
* DEFINED.  THE SPACE OCCUPIED BY THE ADDRESS WILL BE
* CLEARED TO ZERO.
*
* A WARNING IS PRESENTED IF THE PLC TABLE IS NEARLY FULL
* THE EXISTING USER PROGRAM IS LOST IF THE TABLE IS
* ALLOWED TO OVERFLOW.
*
*
STPLC NOP
      STA HOLDA     SAVE (A)
      CLB
      STB SRCFG     CLEAR SEARCH FLAG
      LDA XPLC      BASE ADDRRSS OF PLC TABLE
      JMP *+3
STPL1 LDA ZPLC      RETRIEVE ADDRESS
      ADA .2        ADVANCE TO NEXT POSITION IN TABLE
      STA ZPLC      RETAIN POSITION IN TABLE
      JMP STPL2     CHECK FOR TABLE OVERFLOW
      LDA ZPLC
      LDB A,I
      SZB           AREA OCCUPIED
      JMP STPL1     YES
```

```
      AND B1777     NO SAVE ADDRESS
      STA WMOVE
      LDA HOLDA
      STA ZPLC,I
      ISZ ZPLC
      LDA OPNUM     OPERAND NUMBER
      STA ZPLC,I
      JMP STPLC,I   RETURN
STPL2 LDB YPLC      UPPER BOUND OF PLC ARE
      CMB,INB
      ADB ZPLC
      SSB           OVERFLOW
      JMP STPL3     NO
      LDA .24
      LDB PLCR1
      JMP TBLOV
STPL3 ADB .10
      SSB           TABLE NEARLY FULL
      JMP STPL1+4   NO
      LDA SRCFG     YES
      SZA           MESSAGE ALREADY PRINTED
      JMP STPL1+4   YES
      LDA .42       NO
      LDB PLCR2
      JSB BPLN
      STA SRCFG     SET SEARCH FLAG
      JMP STPL1+4
*
PLCR1 DEF *+1
      ASC 12, * LABEL TABLE OVERFLOW
*
*
PLCR2 DEF *+1
      ASC 21,BEGIN EXECUTION TO PREVENT TABLE OVERFLOW
*
*
CYCFG EQU DPFLG     DEFINE TEMPORARY STORAGE
SRCFG EQU EXP
```

```
*
*
*
*
* MOVE ASSEMBLED CODE WHICH PRECEDES OR FOLLOWS MACHINE
* CODE INSTRUCTIONS INVOLVED IN AN EDIT
*
* ENTER (A) ADDRESS IN ASSEMBLED CODE
*       (B) ADDRESS IN SOURCE CODE BLOCK
*
*
CMOVE NOP
      STA HOLDA
      ADB .4        ASSEM ADDR IN SCB
      LDA ZUSRP     ADDR WHERE CODE WILL RESIDE
      STA B,I       REDIFINE ASSEM ADDR IN SCB
      STB ADDR2     SAVE ADDRESS
      JSB CMVE3
      ISZ ADDR2     ASSEMBLY LENGTH ADDRESS
      LDB ADDR2,I   ASSEMBLY LENGTH
      CPB .2        TWO WORD ASSEMBLY
      JMP CMVE2     YES
      LDA HOLDB     NO
      AND B0700
      SZA,RSS       MEMORY REFERENCE INSTRUCTION
      JMP CMOVE,I   NO, RETURN
      LDA HOLDB
      AND B2000     CURRENT PAGE BIT
      SZA
      JMP CMOVE,I   CORRENT PAGE BIT SET, RETURN
CMVE1 LDA HOLDB     RESTORE INSTRUCTION
      SSA           DEFINED, INDIRECT BIT SET
      JMP CMOVE,I   YES, RETURN
      AND B1777     VALID REFERENCE TO BASE PAGE
      ADA D100      GET ADDRESS
      SSA
```

```
      JMP CMOVE,I  YES REFERENCE TO (A) OR (B)
      ADA .64      NO, INVALID REFERENCE TO BASE PAGE
      STA HOLDB
      LDB B1600    CHECK FOR PLC REFERENCE
      CMB,INB
      ADB A
      SSB,RSS      UNDEFINED PLC REFERENCE
      JMP CMOVE,I  YES, RETURN
      LDA HOLDA
      AND B1777    ADDRESS BEING SOUGHT
      STA GETCR    NO, SAVE ADDRESS
      LDA ZUSRP
      ADA M1
      AND B1777    ADDRESS TO BE INCLUDED DUE TO
      STA RDCOM    REPLACEMENT
      JSB CASCD
      JMP CMOVE,I
*
* TWO WORD ASSEMBLY
*
CMVE2 ISZ HOLDA
      JSB CMVE3
      JMP CMVE1
*
* RETRIEVE ASSEMBLED INSTRUCTION
*
CMVE3 NOP
      LDA HOLDA,I  RETIRIEVE ASSEMBLED CODE
      STA HOLDB
      STA ZUSRP,I  MOVE CODE INTO NEW LOCATION
      CLA
      STA HOLDA,I  PLACE NOP IN VACATED AREA
      ISZ ZUSRP
      JSB STCK,I   LOOK FOR OVERFLOW IN USER PROG
      JMP CMVE3,I
```

```
*
*
* ADVANCE THROUGH LINKED LIST OF FORWARD REFERENCES
* TO CHANGE POINTERS CAUSED BY A DELETE OR CODE
* BEING MOVED
*
*
CASCD NOP
      CLA
      STA IDRCT
      STA CSDFG
CSCD1 LDA HOLDB
      LDB D701
      ADB A          POINTER TO SYMBOL TABLE
      SSB,RSS
      JMP CSCD3      YES
      ADA JMP        CALCULATE ADDR OF NEXT REFERENCE
      STA ADDR2
      LDA ADDR2,I
      AND B1777      ADDRESS OF NEXT REFERENCE
CSCD2 STA HOLDB
      CPA GETCR      ADDRESS BEING SOUGHT
      RSS            YES
      JMP CSCD1      NO
      LDA ADDR2,I    RETRIEVE INSTRUCTION
      AND B1760      SAVE INSTRCUTION SKELETON
      ADA RDCOM      ADD IN NEW ADDRESS
      STA ADDR2,I
      JMP CASCD,I
*
* EXAMINE SYMBOL TABLE FOR FORWARD REFERENCES
*
CSCD3 LDA CSDFG
      SZA
      JMP CSDER      ERROR, CANNOT FIND FWD REF IN TBL
      LDA .125
      CMA,INA
      ADA B
      SSA
      JMP CSCD4      DIRECT REFERENCE IN SYM TBL
      ADB M125
```

```
      LDA .125
      CMA,INA
      ADA B
      SSA,RSS
      JMP CSCD5
      STA IDRCT        INDIRECT REFERENCE IN SYM TBL
CSCD4 BLS
      STB A            MULTIPLY BY 6 FOR SYMBOL TABLE
      BLS              LOOK UP
      ADB A
      ADB XSTBL        BASE ADDRESS OF SYMBOL TABLE
      ADB .3
      LDA IDRCT        INDIRECT REFERENCE
      SZA
      INB              YES, ADVANCE ADDRESS
      JMP CSCD6
CSCD5 ADB M125
      LDA .75
      CMA,INA
      ADA B
      SSA,RSS
      JMP CSDER        ADDRESS NOT IN SYMBOL TABLES
      BLS,BLS          MULTIPLY BY 4
      ADB XSST         BASE ADDRESS OF SST
      ADB .2
CSCD6 STB ADDR2
      LDA B,I
      STA CSDFG
      JMP CSCD2
*
CSDER LDA .34
      LDB *+3
      JSB BPLN
      HLT 55B          STOP    ERROR IN PROGRAM
*
      DEF *+1
      ASC 17,ADDRESS NOT LOCATED-PROGRAM ERROR
*
*
CSDFG EQU WMOVE
```

```
*
*
* DELETE STATEMENT FROM ASSEMBLED CODE
*
* ENTER (B) ADDRESS OF CODE TO BE DELETED
*
*
DELTE NOP
      CLE,ELB
      STB BADDR    ADDRESS POINTER
      CCA          FLAG TO DENOTE LEXICAL SCAN
      STA EDINT    OF CODE TO BE DELETED
      JSB CLER,I
      JSB LEXI,I
      CLA
      STA EDINT    CLEAR LEX-EDIT FLAG
*
      LDA LBLFG    LABEL FLAG FROM SOURCE CODE
      SZA,RSS      LABEL PRESENT
      JMP DELT1    NO
      LDB LBLAD    LABEL ADDR IN SYMBOL TABLE
      ADB .2       ADDRESS OF LABEL INFORMATION
      LDA B,I
      AND CH1      SAVE LAST CHARACTER IN LABEL
      STA B,I
      INB
      STB SAVR     SAVE SYMBOL TABLE ADDRESS
      LDA XSTBL    BASE ADDRESS OFSYMBOL TABLE
      CMA,INA
      ADA LBLAD    ADDRESS OF DELETED LABEL
      CLB
      DIV .6       RELATIVE POSITION OF LABEL
      ADA B701     SYMBOL TBL POSITION POINTER
```

```
       STA SAVR,I     INDICATING UNDEFINED LABELS
       ADA .125       STOTE IN SYMBOL TABLE TO BE USED
       ISZ SAVR
       STA SAVR,I
*
DELT1  LDA ASMFG      ASSEMBLY FLAG
       SSA,RSS
       JMP DELT2      MACHINE INSTRUCTION
       LDA SCBE1      SCB ADDR OF DATA TO BE DELETED
       ADA .5         ADDR OF LENGTH OF ASSEMBLY
       LDB A,I        LENGTH OF ASSEMBLY
       STB LENTH
       JSB DTEDD      EDIT DATA
       JMP DELTE,I
*
DELT2  LDA INSNM      INSTRUCTION NUMBER
       ADA M6
       SSA            MEMORY REFERENCE
       JMP DELTE,I    NO, RETURN
       SZA
       ISZ ELNTH      ADVANCE LENGTH OF DELETED CODE
       LDB OPLBL      YES, OPERAND LABEL PRESENT
       SZB,RSS
       JMP DELTE,I    NO, RETURN DEFINED REFERENCE
       SSB            PROGRAM LOCATION COUNTER REF
       JMP DELT3      YES
       LDB LAB2       NO, DO A SYMBOL TABLE LOOK UP
       JSB LOKP,I
       SSA,RSS        DEFINED REFERENCE
       JMP DELTE,I    YES, RETURN
       LDA ASME1      ADDR OF ASSEMBLED INSTRUCTION
       LDB ASMBY      TWO WORD ASSEMBLY
       SSB
```

```
       INA             YES
       INA
       STA  TEMP3      SAVE ADDRESS OF INSTRUCTION
       AND  B1777      GET RELATIVE ADDRESS OF
       STA  GETCR      UNDEFINED REFERENCE
       LDA  TEMP3,I    GET ASSEMBLED INSTRUCTION
       AND  B1777      ADDR OF NEXT FORWARD REFERENCE
       ADA  D100
       SSA             VALID REFERENCE TO BASE PAGE
       JMP  DELTE,I    YES
       ADA  .64        NO
       STA  RDCOM      SAVE ADDRESS TO BE MOVED
       STA  HOLDB
       JSB  CASCD      CASCADE THROUGH CODE TO UPDATE
       JMP  DELTE,I    ALTERED FORWARD REFERENCES
*
* CLEAR PLC REFERENCE IF INSTRUCTION IS DELETED
*
DELT3  LDB  ASME1      ADDRESS OF ASSEMBLED INSTR
       LDA  ASMBY      INSTRUCTION SKELETON
       SSA             TWO WORD ASSEMBLY
       INB
       LDA  B,I
       AND  B1777      GET RELATIVE ADDRESS
       ADA  B1600      ADDRESS  IN PLC TABLE
       CLB
       STB  A,I        CLEAR ENTRY IN PLC STORE TABLE
       JMP  DELTE,I
```

```
*
*
* DATA DELETE
*
* SHFIT DATA AND DATA ADDRESSES TO FILL GAP LEFT BY
* DELETED DATA
*
* NO DELETE IS NECESSARY WHEN AN EQU PSEUDO OP IS
* DELETED SINCE THE REFERENCE IS CLEARED IN THE SYMBOL
* TABLE
*
*
DTEDD NOP
      LDA LENTH     LENGTH OF DATA TO BE DELETED
      SZA,RSS       LENGTH ZERO
      JMP DTEDD,I   EQU PSEUDO OP, NO OPERATION NEEDED
      CMA,INA
      STA TEMP7
      LDA ASME1     ADDR OF FIRST WORD TO BE DELETED
      ELA,CLE,ERA   CLEAR BIT 15
      STA HOLDA
      ADA LENTH
      STA ASME2
      STA HOLDB
      LDA ZDATA     NEXT FREE DATA LOCATION
      CMA,INA
      ADA ASME2
      STA TEMP6     -NUM OF DATA INTEMS TO BE MOVED
      SZA,RSS
```

```
      JMP DTDD2-3  NO DATA ITEMS TO BE MOVED
DTDD1 LDA HOLDB,I  GET ADDRESS
      LDB A,I      GET VALUE
      ADA TEMP7    ADD DISPLACEMENT TO ADDR
      STB A,I      STORE VALUE IN NEW ADDR
      STA HOLDA,I  STORE ADDR IN NEW POSITION
      ISZ HOLDA
      ISZ HOLDB    ADVANCE ADDR POINTERS
      ISZ TEMP6
      JMP DTDD1
      LDB ASME2    PARAMETERS TO RESET SYMBOL TABLE
      LDA TEMP7    PROGRAM ADDR AREA AND SCB ADDR
      JSB SCSYM
      LDA TEMP7
      STA TEMP6
      CLA          CLEAR VACATED DATA AREA TO ZERO
DTDD2 LDB HOLDA,I
      STA B,I
      STA HOLDA,I
      ISZ HOLDA    ADVANCE ADDRESS POINTER
      ISZ TEMP6
      JMP DTDD2
      LDA ZDATA    RESET NEXT FREE AREA IN DATA
      ADA TEMP7    AREA AFTER DATA DELETE
      STA ZDATA
      JMP DTEDD,I
```

```
*
*
* INSERT DATA
*
* SHIFT DATA AND DATA ADDRESSES WHICH LOGICALLY FOLLOW
* INSERTTHEN STORE INSERTED DATA
*
* NO INSERT INVOLVED WITH EQU PSEUDO OP FOR ENTRY WILL
* BE SET IN SYMBOL TABLE
*
*
DTEDI NOP
      LDB LENTH    EQU PSEUDO OP LENGTH IS ZERO
      SZB,RSS
      JMP DTEDI,I
      LDA SCBE2
      JMP DTEI2
DTEI1 ADA M4
      LDA A,I
DTEI2 LDB ZDATA    NEXT FREE AREA IN DATA AREA
      CPA ENEXT    TERMINATOR
      JMP DTEI3    YES
      ADA .4       ADDR OF ASSEM FLAG, ASSEM ADDR
      LDB A,I      EXAMINE ASSEMBLY FLAG
      SSB,RSS      DATA
      JMP DTEI1    NO
      ELB,CLE,ERB  YES, CLEAR BIT 15
DTEI3 STB ASME2
      STB ZADD     ADDR IN ASSEMBLED CODE
      CPB ZDATA    NO ITEMS TO BE MOVED
      RSS          YES
      JMP DTEI5
DTEI4 LDA LENTH    LENGTH OF INSERT
      ADA ZDATA
      STA ZDATA    CHECK FOR OVERFLOW  IN DATA
      JSB DTFL,I   TABLE
      LDB ZADD     POSITION OF FIRST ENTRY
      JSB DTST,I   INSERT DATA
      JMP DTEDI,I
*
DTEI5 LDA ZDATA    NEXT FREE AREA IN DATA TABLE
      STA HOLDA
      CMA,INA
      ADA ASME2    LOCATION OF FIRST INSERT
```

```
        STA TEMP3     -NUMBER OF WORDS TO BE MOVED
DTEI6   LDA HOLDA
        ADA M1
        STA HOLDA     FIRST DATA ENTRY TO BE MOVED
        LDB A,I       GET ADDRESS
        STB HOLDB
        LDA B,I       GET VALUE
        ADB LENTH     ADD DISPLACEMENT
        STA B,I       STORE VALUE
         LDA HOLDA
        ADA LENTH
        STB A,I       STORE ADDRESS
        ISZ TEMP3
        JMP DTEI6
        LDB SCBE2     POINTER FOR SCAN THROUGH SCB
        ADB .1
        LDB B,I
        STB SCBE1
        LDA LENTH     SCAN THROUGH SYMBOL TABLE,
        LDB ASME2     PROGRAM DATA AREAS AND SCB
        JSB SCSYM     TO CLEAR UP ADDRESS CHANGES
        LDB ASME2     ADDR WHERE DATA WILL BE INSERTED
        STB ZADD      ADDR IN ASSEMBLED CODE
        JMP DTEI4     INSERT DATA
```

```
*
*
* SCAN SYMBOL TABLE, USER PROGRAM ADDRESS AREA AND
* SOURCE CODE BLOCK TO UPDATE LABELS AFTER AN EDIT
* OPERATION INVOLVING DATA
*
* ENTER (A) CORRECTION VALUE TO ADDRESSES
*       (B) ADDRESS VALUE USED TO WHICH ADDRESSES
*           NEED BE CHANGED
*
*
*
SCSYM NOP
      STA NUM1      VALUE
      CMB,INB
      STB NUM2      ADDRESS
      LDA XSTBL     ADDRESS OF SYMBOL TABLE
      RSS
SCSM1 ADA .6        NEXT ENTRY IN SYMBOL TABLE
      LDB YSTBL     UPPER BOUND OF SYM TBL
      CMB,INB
      ADB A
      SSB,RSS       OVERFLOW
      JMP SCSM4
      LDB A,I       NO, CONTENTS OF ADDRRSS
      SZB,RSS
      JMP SCSM1     NO ENTRIES
      STA BCKSP     ENTRY, SAVE ADDRESS
      ADA .2
      LDB A,I       GET LABEL INFORMATION
      CLE,ERB
      SEZ           LABEL DEFINED
      JMP SCSM3     YES
SCSM2 LDA BCKSP     NO
      JMP SCSM1
SCSM3 ADA .1
      STA ADDR2
      LDB A,I        ADDR IN ASSEMBLED CODE
```

```
      LDA NUM2      TEST ADDRESS
      ADA B         ADD IN ADDRESS
      SSA           TOO SMALL
      JMP SCSM2     YES
      LDA B,I       RETRIEVE ADDR POINTER
      ADA XDATA     EQU ADDR DEFINITION
      SSA
      JMP SCSM2     YES
      ADB NUM1      NO, REDEFINE LABEL
      STB ADDR2,I   SET VALUE IN SYMBOL TABLE
      JMP SCSM2     CONTINUE
*
* CHECK FOR DATA OR MACHINE CODE LABEL
*
SCSM4 LDA NUM2      TEST LABEL ADDRESS
      CMA,INA       CONVERT TO POSITIVE
      JSB DATAD     CORRECTION IF DATA ADDRESS
      CMA,INA       CONVERT TO NEGATIVE VALUE
      STA NUM2
*
* EXAMINE LABEL AREA IN PROGRAM
*
      LDA D337
      STA TEMP3
      LDA JMP
      STA TEMP4
SCSM5 ISZ TEMP4
      CLB
      LDA TEMP4,I   LOAD ADDRESS
      SSA
      CCB
      STB IDRCT     SET INDIRECT FLAG
      ELA,CLE,ERA   CLEAR BIT 15
      LDB NUM2      TEST ADDRESS
      ADB A
      SSB           CORRECTION REQUIRED
```

```
      JMP *+6     NO
      LDB IDRCT   YES
      ADA NUM1    ADD IN CORRECTION
      SSB
      IOR MNEG    MASK ON INDIRECT BIT
      STA TEMP4,I RETURN ADDRESS
      ISZ TEMP3
      JMP SCSM5
*
* SATISFY SCB REFERENCES WITH DATA
*
      LDB SCBE1,I GET ADDRESS
      RSS
SCSM6 LDB B,I     ADDRESS OF NEXT STATEMENT
      CPB ENEXT   FINISHED
      JMP SCSYM,I RETURN
      ADB .4      ADDR OF ASSEM FLAG, ASSEM ADDR
      LDA B,I
      SSA,RSS     DATA
      JMP SCSM7   NO
      STA STFSP   SAVE (A)
      INB         ADDR OF LENGTH OF ASSEMBLY
      LDA B,I     ASSEMBLY LENGTH IN (A)
      ADB M1      DECREMENT (B)
      SZA,RSS     EQU PSEUDO OP
      JMP SCSM7   YES
      LDA STFSP   NO, RESTORE (A)
      ELA,CLE,ERA REMOVE BIT 15
      ADA NUM1    ADD CORRECTION TERM TO DATA ADDR
      IOR MNEG    RESTORE BIT 15
      STA B,I
SCSM7 ADB M4      RESET (B)
      JMP SCSM6
```

```
*
*
* STORE LENGTH AND ADDRESS OF DELETION FROM SOURCE CODE
* BLOCK IN FREE SPACE AREA
*
*
* EACH DELETION FROM THE SOURCE CODE BLOCK WILL BE
* RECORDED IN TWO WORDS IN THE FREE SPACE TABLE WITH
* THE DELETED AREA IN THE SCB CLEARED TO ZERO
*
* WORD 1   LENGTH OF DELETION
*
* WORD 2   SCB ADDRESS OF DELETION
*
*
* FULL TABLE  SUBSEQUENT ENTRIES, LARGER THAN THE SMALLEST
*             WILL REPLACE THE SMALLEST.  ENTRIES SMALLER
*             THAN THE SMALLEST WILL BE IGNORED.
*
*
*
STFSP NOP
      CLA
      LDB SCBE1     ADDRESS OF DELETION
      STB TEMP5
      LDB CNFG3     LENGTH OF DELETION
      CMB,INB
      STB TEMP6
FSP1  STA TEMP5,I   CLEAR DELETED AREA
      ISZ TEMP5
      ISZ TEMP6     ADVANCE POINTERS
      JMP FSP1
      LDA XFRSP     ADDRESS OF TABLE
      RSS
FSP2  ADA =2        ADVANCE TO NEXT POSITION IN TABLE
      STA ZFRSP     SAVE PRESENT POSTION
      CMA,INA
      ADA YFRSP     UPPER BOUND OF TABLE
      SSA           IF TABLE FULL FIND SMALLER ENTRY
      JMP FSP3
      LDA ZFRSP     RETRIEVE PRESENT POSITION
      LDB ZFRSP,I   AREA OCCUPIED
      SZB
      JMP FSP2      YES, LOOK AT NEXT POSITION
      LDA CNFG3     NO, GET LENGTH OF DELETION
```

```
      STA ZFRSP,I  STORE LENGTH
      ISZ ZFRSP
      LDA SCBE1    ADDRESS OF DELETION
      STA ZFRSP,I  STORE ADDRESS
      JMP STFSP,I
FSP3  CLA
      STA TEMP2
      LDA XFRSP    BASE ADDRESS OF FREE SPACE
      JMP *+3
FSP4  LDA TEMP1
      ADA .2
      STA TEMP1    ADDRESS OF NEXT BLOCK IN FSP
      LDB YFRSP    UPPER BOUND OF FREE SPACE
      CMB,INB
      ADB A
      SSB,RSS      TABLE FULLY SCANNED
      JMP FSP6     YES
      LDB TEMP1,I  NO, GET LENGTH OF DELETION
      CMB,INB
      ADB CNFG3    GREATER THAN PRESENT DELETE
      SSB
      JMP FSP4     NO
      SZB,RSS
      JMP FSP4     NO
      LDB TEMP2
      SZB
      JMP FSP5
      STA TEMP2    ADDRESS IN FREE SPACE
      JMP FSP4
FSP5  LDB TEMP2,I
      CMB,INB
      ADB A,I
      SSB
      STA TEMP2
      JMP FSP4
FSP6  LDA TEMP2
      SZA,RSS
      JMP STFSP,I
      LDB CNFG3    LENGTH OF DELETION
      STB A,I
      INA
      LDB SCBE1    ADDRESS OF DELETION
      STB A,I
      JMP STFSP,I
```

```
*
*
* RETRIEVE ASSEMBLED CODE ADDRESSES OF INSTRUCTIONS
* INVOLVED IN THE EDIT OPERATION
*
*
ASMAD NOP
      LDA SCBE       ADDRESS OF SCB ADDRESS
      STA TEMP1
      LDB ASME       ADDR OF ASSEMBLED CODE ADDR STORE
      STB TEMP2
      LDA M3
      STA TEMP3
ASMD1 LDA TEMP1,I    ADDRESS IN SCB
      SZA,RSS        CHECK FOR UNDEFINED REFERENCE
      JMP ASMD2
      ADA .4         ADDRESS OF ASSEMBLY
      LDB A,I        ADDRESS IN ASSEMBLED CODE
      STB TEMP2,I
ASMD2 ISZ TEMP1
      ISZ TEMP2      ADVANCE ADDRESS POINTERS
      ISZ TEMP3
      JMP ASMD1
      JMP ASMAD,I
```

```
         ORG 12000B
*
*
*
* INTERPRET AND EXECUTE EDIT INSTRUCTIONS
*
*
* EDIT WILL ALLOW THE USER TO
*
*              DELETE  ANY NUMBER OF STATEMENTS IN THE PROGRAM
*              INSERT  BETWEEN SUCCESSIVE STATEMENTS
*              REPLACE ANY STATEMENT
*
*
* THE FOLLOWING OPERATION CAUSES STATEMENTS M THROUGH
* N, INCLUSIVE, TO BE DELETED
*
*              /D(ELETE),M(,N)(,V)
*
* IF ONLY M IS SPECIFIED ONLY THAT STATEMENT WILL BE
* DELETED.
* IF M > N THE INSTRUCTION WILL BE IGNORED
*
*
* V IS THE VETO FLAG
*
* WHEN SPECIFIED, STATEMENT(S) REFERENCED BY THE EDIT
* INSTRUCTION WILL BE PRINTED.  A MESSAGE WILL ASK THE
* USER IF THIS IS THE CODE TO BE EDITTED.
* A RESPONSE OF Y(ES) WILL CONTINUE THE EIDT INSTRUCTION
* WITH ANY OTHER RESPONSE CAUSING THE EDIT INSTRUCTION
* TO BE IGNORED.
*
*
* TO INSERT BETWEEN SUCCESSIVE STATEMENTS
*
*              /I(NSERT),M(,N)
*
* IF ONLY M IS SPECIFIED ONLY STATEMENT M WILL BE
* INSERTED.  N IS AN INCREMENT FOR MORE THAN ONE
* INSERTION BETWEEN SUCCESSIVE STATEMENTS.
*
```

```
*
* RESTRICTIONS ON AN INSERT
*
* 1  ON A MULTIPLE INSERT (N>0), IT WILL NOT BE
*    POSSIBLE TO ENTER BOTH DATA AND MACHINE CODE
*    TYPE STATEMENTS.
* 2  A MULTIPLE INSERTION WILL BE AUTOMATICALLY ENDED
*    IF THE STATEMENT NUMBER OF THE WOULD BE INSERT
*    EXCEEDS THE NEXT STATEMENT NUMBER IN THE PROGRAM.
*
*
*
* TO REPLACE A SINGLE STATEMENT
*
*        /R(EPLACE),M(,V)
*
* A MACHINE CODE INSTRUCTION CANNOT BE REPLACED BY DATA
* NOR CAN A DATA STATEMENT BE REPLACED BY A MACHINE
* INSTRUCTION.
*
* THERE IS NO MULTIPLE REPLACE BECAUSE SEQUENCING
* INFORMATION IS NOT AVAILABLE
*
*
* THE END INSTRUCTION WILL TERMINATE THE CURRENT EDIT
* OPERATION.
*
*        /E(ND)
*
*
EDIT   CPA SLASH     SLASH PRECEDING EDIT OPERATION
       RSS
       JMP EDR1      NO, ERROR
EDIT1  LDB MIIP
       SZB           MULTIPLE INSERT NOW COMPLETE
       JSB ENDMI     CLEAR UP MULT INSERT
       JSB EDCLR     CLEAR EDIT VARIABLES
       JSB NTBLK     NEXT NON BLANK CHARACTER
       JMP EDR2      NO INSTRUCTION
       LDB EDNUM     EDIT INSTRUCTION NUMBER
       CPA D         DELETE REQUEST
```

```
      JMP EDIT2+1
      ADB .2       NO, ADVANCE INSTR NUMBER
      CPA E        END REQUEST
      JMP EDT40    YES
      CPA I        NO, INSERT REQUEST
      JMP EDIT2    YES
      ADB .2       NO ADVANCE INSTR NUMBER
      CPA R        REPLACE REQUEST
      RSS          YES
      JMP EDR2     NO, UNDEFFINED EDIT OPERATION
EDIT2 STB EDNUM    SAVE INSTRUCTION NUMBER
      JSB RDCOM    READ UPTO COMMA
      JMP EDR6
      JSB TWNT,I   READ IN STATEMENT NUMBERS
      JMP EDIT4+1
      JMP EDIT3
      JMP EDIT4
      LDB EDNUM    EDIT INSTRUCTION NUMBER
      CPB .5       REPLACE
      JMP EDR3     YES, ERROR
      CLE,RSS
EDIT3 CCE
      JSB VETCK    LOOK FOR VETO FLAG
      SEZ          MULTIPLE OPERATION
      JMP EDIT4+1  NO
      LDB EDNUM    YES, CHECK FOR REPLACE
      CPB .5
      JMP EDR3
EDIT4 ISZ EDNUM    ADVANCE INSTRUCTION NUMBER
      LDA NUM1
*
* CHECK RANGE OF FIRST NUMBER
*
      LDB FSTMT    FIRST STATEMENT NUMBER IN PROGRAM
      CMB,INB
      ADB A        ADD FIRST EDIT STATEMENT NUMBER
      SSB
      JMP EDR5
      LDB CUSTN    LAST STATEMENT NUMBER IN PROGRAM
      CMB
      ADB A
      SSB,RSS
      JMP EDR5     FIRST NUMBER TOO LARGE
```

```
*
* ADVANCE THROUGH SOURCE CODE BLOCK FOR ADDRESSES OF
* CODE INVOLVED IN EDIT
*
      LDA FIRST     ADDR OF FIRST ENTRY IN SCB
      RSS
EDIT5 LDA A,I       ADDRESS OF NEXT ENTRY
      CPA ENEXT     END OF PROGRAM
      JMP EDIT7     YES
      ADA .2        ADDRESS OF STATEMENT NUMBER
      LDB A,I       STATEMENT NUMBER
      ADA M2
      STB STORE
      CMB,INB
      ADB NUM1      FIRST EDIT STATEMENT NUMBER
      SZB,RSS
      JMP EDIT6
      SSB
      JMP EDIT7
      STA SCBE0
      JMP EDIT5
EDIT6 STA SCBE1
      JMP EDIT5
EDIT7 LDB EDNUM     EDIT INSTRUCTION NUMBER
      CPB .2        MULTIPLE DELETE
      JMP EDIT8     YES
      CPB .1        SINGLE DELETE
      JMP EDIT9+1
      STA SCBE2     SAVE ADDRESS OF INSTRUCTION
      JMP EDT10     WHICH FOLLOWS EDIT OPERATION
EDIT8 LDB STORE     RETRIEVE STATEMENT NUMBER
      CPB NUM2      LAST STAEMENT TO BE DELETED
      JMP EDIT9     YES
      CPA ENEXT     TERMINATION
      JMP EDIT9+2   YES
      CMB,INB
      ADB NUM2      FIRST STATEMENT NUMBER AFTER
      SSB,RSS       MULTIPLE DELETE
      JMP EDIT5     NO
      RSS           YES
EDIT9 LDA A,I       ADDR OF NEXT STATEMENT
      CPA ENEXT     TERMINATOR IN SCB
```

```
      STA DLTLN    DELETE LAST LINE
      STA SCBE2
      JMP EDT12
*
* CHECK FOR MULTIPLE INSERT
*
EDT10 CPB .4       MULTIPLE INSERT
      RSS          YES
      JMP EDT12    NO
      LDA NUM2
      SZA,RSS      ZERO INCREMENT
      JMP EDR5     YES, ERROR
      LDB SCBE2    INSTRUCTION AFTER INSERT
      ADB .2
      LDB B,I      STATEMENT NUMBER
*
* UPPER LIMIT OF STATEMENT NUMBER ON A MULTIPLE INSERT
*
      STB EDLMT
      CMB,INB
      ADB NUM1
      ADB NUM2     STATEMENT NUMBER INCREMENT
      SSB,RSS      TOO LARGE
      JMP EDT11    YES, CONVERT TO SINGLE INSERT
      LDB NUM2     PREPARE STATEMENT NUMBERS
      CMB,INB      FOR FIRST ENTRY OF MULTIPLE
      ADB NUM1     INSERT
      STB NUM1
      JMP EDT13
*
EDT11 LDA .3       CONVERT TO AS SINGLE INSERT
      STA EDNUM
      LDA .40      WARNING TO USERS
      LDB EDM1
      JSB BPLN
      JMP EDT13
*
EDM1  DEF *+1
      ASC 20,MULTIPLE INSERT CHANGED TO SINGLE INSERT
*
```

```
*
*
* EXAMINE VETO FLAG
*
EDT12 LDA VETO
      SZA,RSS      VETO FLAG SET
      JMP EDT13    NO
      LDA NUM1     YES, PRINT INSTR INVOLVED IN EDIT
      LDB NUM2
      CPB ZERO     PRINT 1 LINE
      STA NUM2     YES, SET VARIABLE FOR LISTING
      LDA SCBED    ADDR OF STATEMENT BEFORE EDIT
      LDB FIRST    ADDR OF FIRST STATEMENT
      SZA          EDIT INVOLVE FIRST STATEMENT
      LDB A,I      NO
      STB SUCAD
      JSB NWLN,I
      CCA          SET FLAG FOR EDIT CALL
      JSB LISTI,I  LIST
      LDA .30
      LDB VETRQ    VETO REQUEST
      JSB BPLN
      JSB DATN,I   READ RESPONSE
      CPA Y        YES CONTINUE
      JMP EDT13
      JMP CNTRL,I  NO, ENTER NEW EDIT INSTRUCTION
*
VETRQ DEF *+1
      ASC 15,DO YOU WISH TO EDIT THIS CODE
*
*
EDT13 LDA NUM1
      LDB NUM2
      STA ENM1
      STB ENM2
*
* GET ASSEMBLED CODE ADDRESSES OF INSTRUCTIONS
* INVOLVED IN THE EDIT
*
      JSB ASMD,I
*
```

```
      LDA EDNUM
      CPA .1       SINGLE DELETE
      RSS          YES
      JMP EDT16    NO
      JSB DSCB     SET SCB REFERENCES FOR A DELETE
      LDB SCBE1    ADDR OF STATEMENT TO BE DELETED
      SZB,RSS
      JMP EDR9
      JSB PREPR    PREPARE FOR SCAN OF STORED CODE
      SZA,RSS      COMMENT STATEMENT
      JMP EDT14    YES,
      STA VETO     SAVE ASSEM FLAG, ADDR OF ASSEM
      ADB .2       ADDR OF CODE TO BE DELETED
      JSB DLTE,I   DELETE
      LDA VETO     ASSEM FLAG, ADDR OF ASSEMBLY
      SSA
      JMP EDT14    DATA
      LDB LENTH    LENGTH OF ASSEMBLY
      CPB .2       TWO WORD ASSEMBLY
      RSS
      JMP EDT15    NO, ONE WORD ASSEMBLY
      LDB ASME1    STORE JUMPS IN DELETED INSTRUCTION
      LDA .2
      ADA B        ADDR WHERE JUMP POINTS
      JSB JMPS     REPLACE TWO WORD ASSEMBLY BY JUMPS
EDT14 JSB SFSP,I
      JMP CNTRL,I  RETURN TO CONTROLLER
*
EDT15 JSB SNGDL    SINGLE DELETE
      JMP EDT14
*
* MULTIPLE DELETE INSTRUCTION
*
EDT16 CPA .2       MULTIPLE DELETE
      CLB,RSS      YES
      JMP EDT21    NO
      STB VETCK
      LDB NUM1
      CMB,INB
      ADB NUM2     CHECK THAT FIRST STATEMENT NUMBER
      SSB          IS LESS THAN SECOND
      JMP EDR5     NO, ERROR
```

```
      LDB FIRST    ADDR OF FIRST STATEMENT
      LDA SCBE0
      SZA          DELETE FIRST LINE
      LDB A,I      NO, ADDR OF INSTR BEFORE DELETE
      STB SCBE1    ADDR OF FIRST DELETION
      JSB DSCB     SATISFY SCB REFERENCES
      LDB SCBE1
      JMP *+3
EDT17 LDB SAVR
      STB SCBE1    ADDR OF NEXT DELETION
      LDA B,I      RETAIN POINTER TO NEXT STATEMENT
      STA SAVR
      CPB SCBE2    END OF DELETIONS
      JMP EDT19    YES
      JSB PREPR    NO, PREPARE SOME LEXICAL POINTERS
      STA VETO     SAVE (A)
      SZA,RSS
      JMP EDT18+2  COMMENT STATEMENT
      CLE,ELA      GET ADDRESS IN ASSEMBLED CODE
      RAR
      STA ASME1    CLEAR BIT 15 IF NECESSARY
      RAL
      ERA,CLE      THEN RESTORE BIT 15
      SSA          DATA
      JMP EDT18
      STA DADR2    ADDR OF LAST M C DELETE
*
* CLEAR LOCATION INVOLVED IN EDIT
* (FIRST WORD IN A TWO WORD ASSEMBLY)
*
      CLA
      STA DADR2,I
      LDA DADR1    ADDRESS OF FIRST M C DELETE
      SZA
      JMP EDT18
      LDA DADR2
      STA DADR1
EDT18 ADB .2       ADDRESS OF SOURCE CODE
      JSB DLTE,I
      JSB SFSP,I
```

```
      LDA VETO     ASSEM FLAG, ADDR OF ASSEMBLY
      SZA,RSS
      JMP EDT17    COMMENT
      SSA
      JMP EDT17
      LDA ELNTH    LENGTH OF DELETED CODE
      CPA .2       TWO WORD DELETE
      RSS          YES
      JMP *+4
*
* CLEAR SECOND WORD IN A TWO WORD DELETE
*
      ISZ DADR2
      CLA
      STA DADR2,I  CLEAR DELETION
      LDB VETCK    MACHINE INSTRUCTION
      ADB LENTH    SAVE LENGTH OF DELETED
      STB VETCK    MACHINE CODE
      JMP EDT17
EDT19 LDB VETCK    NUMBER OF M C WORDS DELETED
      SZB,RSS
      JMP CNTRL,I  NO M C INSTRUCTION DELETED
      CPB .1       ONE M C WORD TO BE DELETED
      RSS          YES
      JMP EDT20    NO
      LDA DADR1    ADDR OF WORD TO BE DELETED
      STA ASME1
      JSB SNGDL    DELETE ONE MACHINE INSTR
      JMP CNTRL,I  RETURN TO CONTROLLER
*
EDT20 LDB DADR1    ADDR WHERE JUMP ORIGINATES
      LDA DADR2    ADDR WHERE JUMP RESULTS
      INA          ADDR OF NEXT INSTR IN ASSEM CODE
      JSB JMPS     INSERT JUMPS TO FINISH MULT DLTE
      JMP CNTRL,I  RETURN TO CONTROLLER
```

```
*
* SINGLE INSERT
*
EDT21 CPA .3          SINGLE INSERT
      RSS              YES
      JMP EDT24       NO
      LDB SCBE1
      SZB
      JMP EDR7        INSTR EXISTS AT POSITION OF INSRT
      JSB EDIPT       EDITOR SOURCE INPUT
      LDA ASMFG       ASSEMBLY FLAG
      SZA,RSS         COMMENT
      JMP EDT22       YES
      SSA,RSS
      JMP EDT23       MACHINE INSTRUCTION
      JSB DTDI,I      DATA
EDT22 JSB ISCB
      JMP SCBI,I
EDT23 JSB ISCB        INSERT INTO SCB
      JSB XINS        FIND ASSEM CODE BEFORE INSERT
      JSB SVPSN       HOLD NEXT FREE POSN IN PROGRAM
      LDA ASMEO       ADDR IN ASSEM CODE
      LDB REENT
      JSB CMVE,I      MOVE CODE BEFORE INSERT
      CLA,INA         FLAG FOR M C TO BE STORED
      JSB STCD,I      STORE INSERTED CODE
      JSB YINS        NEXT INSTR IN ASSEM CODE
      JMP SCBI,I
      JSB CMVE,I      MOVE ASSEMBLED CODE AFTER INSERT
*
* INSERT JUMP TO LINK EDIT ENTRY
*
      LDB ASMEO
      LDA EDTSV
      JSB JMPS
      JSB JMPAF
      JMP SCBI,I
```

```
*
* MULTIPLE INSERT
*
EDT24 CPA .4          MULTIPLE INSERT
      RSS             YES
      JMP EDT29       NO
      LDB SCBE1       STATEMENT NUMBER ALREADY
      SZB             DEFINED
      JMP EDR7
      CCB
      STB MIIP        MULTIPLE INSERT IN PROGRESS
*
* RETURN FROM SYSTEM CONTROLLER
*
MIRT  JSB EDIPT       ENTRY POINT DURING MULT INSERT
      LDA ASMFG       ASSEMBLY FLAG
      SZA,RSS
      JMP EDT26       COMMENT
      LDB EXPEC       DATA OR M C EXPECTED
      SZB
      JMP *+3         YES
      STA EXPEC
      JMP EDT25
      CPA B           MATCH BETWEEN ENTRY AND PREV
      JMP EDT25       YES
      CCB
      STB EDLX        EDIT INPUT REQUEST FLAG
      JMP EDR6
EDT25 SSA,RSS
      JMP EDT27       MACHINE INSTRUCTION
      JSB DTDI,I      DATA INSERT
EDT26 JSB ISCB        SATISFY SCB REFERENCES
      JMP SCBI,I
EDT27 LDB MCMIP       MACHINE CODE MULTIPLE INSERT
      SZB             FLAG
      JMP EDT28
      CCB
      STB MCMIP       SET FLAG
      JSB MULIN       PREPARE FOR MULTIPLE INSERT
EDT28 JSB ISCB
      JSB STCD,I      STORE CODE
      JMP SCBI,I
```

```
*
* REPLACE
*
EDT29 LDB SCBE1     ADDR OF LINE TO BE REPLACED
      SZB,RSS       UNDEFINED STATEMENT
      JMP EDR9      ERROR
      JSB PREPR     PREPARE FOR SCAN OF SOURCE CODE
      SZA,RSS       COMMENT
      JMP EDT30
      SSA           NO, DATA OR MACHINE CODE
      CCA,RSS       DATA
      CLA,INA       MACHINE INSTRUCTION
      STA EXPEC
      ADB .2        ADDR OF CODE TO BE REPLACED
      JSB DLTE,I
EDT30 JSB SFSP,I
      JSB EDIPT     EDITOR SOURCE CODE INPUT
      JSB RSCB      SATISFY SCB REFERENCES
      LDA EXPEC     FLAG FOR EXPECTED INPUT
      LDB ASMFG     INPUT ASSEMBLY FLAG
      CPA .1        MACHINE CODE DELETED
      RSS           YES
      JMP EDT35     NO, DATA OR COMMENT
      CPA B         MACHINE CODE INSERT
      RSS           YES
      JMP EDT34     NO, COMMENT INSERT
      LDA ASME1     ADDR IN ASSEM CODE OF DELETION
      LDB ELNTH     LENGTH OF DELETED CODE
      CPB .2        TWO WORD ASSEMBLY
      RSS           YES
      JMP EDT33     NO
      LDB LENTH     LENGTH OF ASSEM REPLACEMENT CODE
      CPB .2        TWO WORD ASSEMBLY
      JMP EDT32     YES
      JSB STCD,I    NO, ONE WORD
      JSB JMPBF     JMP TO EDIT ENTRY
      LDB ZUSRP     JUMPS AFTER EDIT ENTRY
      LDA ASME2
      SZA,RSS
      LDA EUSRP
      JSB JMPS
      JMP SCBI,I
```

```
EDT32 LDB ZUSRP     SAVE PROGRAM POINTER
      STB CNFG3
      STA ZUSRP     TEMP VALUE OF PROG POINTER
      JSB STCD,I    SET AND STORE CODE
      LDA CNFG3
      STA ZUSRP     RESTORE PROGRAM POINTER
      JMP SCBI,I
*
EDT33 LDB LENTH     ONE WORD DELETION
      CPB .1        REPLACE BY
      JMP EDT32     ONE WORD ASSEMBLY
      JSB SVPSN     TWO WORD ASSEMBLY
      JSB STCD,I    SET AND STORE CODE
      JSB YINS      GET NEXT ASSEMBLED INSTR
      JMP SCBI,I
      JSB CMVE,I    MOVE CODE FOLLOWING INSERT
       JSB JMPBF
      JSB JMPAF     AND EDIT CHANGES
      JMP SCBI,I
*
* MACHINE CODE REPLACED BY A COMMENT
*
EDT34 LDA ASME1
      JSB SNGDL     SINGLE DELETE
      JMP SCBI,I
*
* COMMENT DELETED
*
EDT35 CPA ZERO      COMMENT DELETED
      RSS           YES
      JMP EDT36     NO, DATA DELETE
      CPA B         COMMENT INSERTED
      JMP SCBI,I    YES
      CPB .1        NO, MACHINE CODE INSERTED
      JMP EDT23+1
      JMP EDT36+1   NO, INSERT DATA
*
* DATA DELETED
*
EDT36 CPA B         DATA INSERT
      JSB DTDI,I    YES
      JMP SCBI,I
```

```
*
* END REQUEST
*
EDT40 LDA NEXT
      STA PREV,I    CLEAR UP REFERENCES IN SCB
      LDB EUSRP     SET JUMP TO LINK EXISTING PROGRAM
      LDA ZUSRP     WITH REMAINING PROGRAM AREA
      JSB JMPS
      CLB
      STB EDTFG     CLEAR EDIT FLAG
      JMP CNTRL,I
*
*
* CLEAR EDIT VARIABLES
*
*
EDCLR NOP
      CLB
      STB ASME0
      STB ASME1     ASSEMBLY ADDRESSES
      STB ASME2
      STB DADR1     ASSEMBLY CODE ADDRESSES ON A
      STB DADR2     MULTIPLE DELETE OPERATION
      STB DLTLN     DELETE LAST LINE
      STB EDINT     EDIT VARIABLE FOR LEXICAL SCAN
      STB EDLMT     STAT NUM LIMIT ON MULT INSERT
      STB EDTSV     ADDR FOR MOVING CODE
      STB EXPEC     INPUT EXPECTATION FLAG
      STB MIIP      MULTIPLE INSERT IN PROGRESS
      STB MCMIP     MACHINE CODE MULT INSERT
      STB SCBE0
      STB SCBE1     SOURCE CODE BLOCK ADDRESSES
      STB SCBE2
      STB VETO      VETO FLAG
      INB
      STB EDNUM     INSTRUCTION NUMBER
      STB ELNTH     LENGTH OF DELETED CODE
      JMP EDCLR,I
```

```
*
*
* CHECK FOR VETO FLAG ON AN EDIT OPERATION
*
*
* ENTER (E) = 0 MULTIPLE INSTRUCTION
*       (E) = 1 SINGLE INSTRUCTION
*
*
VETCK NOP
      SEZ
      JMP *+3         SINGLE INSTRUCTION
      JSB RDCOM       READ UPTO COMMA
      JMP EDR3        BAD DATA FOLLOWS EDIT STATEMENT
      JSB NTBLK       NEXT NON BLANK CHAR
      JMP EDR3
      CPA V           VETO FLAG
      RSS             YES
      JMP EDR3        NO
      LDB EDNUM       INSTRUCTION NUMBER
      CPB .3          INSERT
      JMP EDR4        YES, ERROR
      STA VETO        NO, SET VETO FLAG
      JMP VETCK,I
*
*
EDR1  LDA .38
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 19,ILLEGAL DATA PRECEDES EDIT INSTRUCTION
*
*
EDR2  LDA .26
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 13,UNDEFINED EDIT INSTRUCTION
```

```
*
*
EDR3  LDA .34
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 17,BAD DATA FOLLOWS EDIT INSTRUCTION
*
*
EDR4  LDA .32
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 16,VETO NOT PERMITTED ON AN INSERT
*
*
EDR5  LDA .30
      LDB *+2
      JMP ERCAL
*
      DEF EPR2      STATEMENT NUMBER OUT OF RANGE
*
*
EDR6  LDA .38
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 19,ILLEGAL SOURCE TYPE ENTRY DURING EDIT
*
```

```
*
EDR7  LDA .32
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 16,STATEMENT NUMBER ALREADY DEFINED
*
*
EDR8  LDA .50
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 25,STATEMENT NUMBERS MUST ACCOMPANY EDIT INSTRUCTION
*
*
EDR9  LDA .32
      LDB *+2
      JMP ERCAL
*
      DEF *+1
      ASC 16,STATEMENT NUMBER IS NOT DEFINED
```

```
*
*
*
* SOURCE CODE INPUT DURING AN EDIT OPERATION
*
*
EDIPT NOP
*
* SOURCE CODE INPUT DURING EDIT OPERATION
* JUMP TO SYSTEM CONTROLLER TO READ ENTRY
*
      CCB
      STB EDLX     FLAG SOURCE CODE DURING EDIT
      JMP CNTRL,I  READ INPUT
*
*
EDXRT CPA SLASH    RETURN AFTER READ
      JMP EDIT1    EDIT DIRECTIVE
      JSB LEXI,I   SCAN INPUT TEXT
      LDA EDNUM    EDIT INSTRUCTION TYPE
      CPA .5       REPLACE
      RSS          YES
      JMP EDPT1
*
* LOOK FOR VALID ENTRY DURING REPLACE
*
      LDA EXPEC    TYPE OF INPUT EXPECTED
      LDB ASMFG
      CPA B        ASSEMBLY FLAGS MATCH
      JMP EDPT2    YES, VALID REPLACEMENT
      ADB A
      SZB,RSS
      JMP EDR6     ERROR, TYPE CLASH ON ENTRY
```

```
EDPT1 LDB MIIP     MULTIPLE INSERT IN PROGRESS
      SZB,RSS
      JMP EDPT2
      LDB ENM1
      ADB ENM2     ADD INCREMENT
      STB ENM1     NEW STATEMENT NUMBER
*
* CHECK FOR STATEMENT NUMBER RANGE
*
      CMB
      ADB EDLMT    UPPER LIMIT OF STATEMENT NUMBER
      SSB,RSS      IN RANGE
      JMP EDPT2    YES
      JSB ENDMI    NO, END MULTIPLE INSERT
      STA EDLX     CLEAR SOURCE CODE FLAG
      LDA .46
      LDB EDPTM
      JSB BPLN
      JMP CNTRL,I  RETURN TO CONTROLLER
*
*
EDPTM DEF *+1
      ASC 23,STATEMENT IGNORED, MULTIPLE INSERT TERMINATED
*
*
*
* CLEAR CONTROL FLAG
*
EDPT2 CLB
      STB EDLX
      JSB ASSM,I   GET SCB ADDRESS
      JMP EDIPT,I
```

```
*
*
* LINK INSERT WITH EXISTING SOURCE CODE BLOCK ENTRIES
*
*
ISCB   NOP
       LDA ADDR1      ADDR OF INSERT IN SCB
       STA SCBE0,I    ADDR OF NEXT IN PREV INSTR
        LDB SCBE2
       STB A,I        ADDR OF NEXT IN NEW INSTR
       INB
       STA B,I        ADDR OF PREV IN NEXT INSTR
       INA
       LDB SCBE0      SET ADDR OF PREV IN NEW INSTR
       STB A,I
       INA
       LDB ENM1       STATEMENT NUMBER
       STB A,I        STORE STATEMENT NUMBER
       LDA ADDR1      SAVE ADDRESS OF INSERT ON A
       STA SCBE0      MULTIPLE INSERT OPERATION
       JMP ISCB,I
```

```
*
*
* FIND INSTRUCTION IN PROGRAM WHICH LOGICALLY PRECEDES
* INSERTED MACHINE CODE
*
*
XINS  NOP
      LDA ASMEO    ADDR IN ASSEM CODE
      LDB SCBEO    ADDR IN SOURCE CODE
      SZB,RSS      PRECEDING STATEMENT NOT FOUND
      JMP XINS3
XINS1 STB REENT    SAVE ADDRESS
      SZA,RSS
      JMP XINS2    COMMENT
      SSA
      JMP XINS2    DATA
      STA ASMEO    MACHINE INSTRUCTION
      JMP XINS,I
XINS2 INB
      LDB B,I      ADDR OF PREVIOUS INSTR
      CPB M1       TERMINATOR
      JMP XINS3    YES
      ADB .4
      LDA B,I      ASSEM FLAG, ASSEM ADDR
      ADB M4       RESTORE ADDR
      JMP XINS1
*
* NO MACHINE INSTRUCTION PRECEDES INSERT
*
XINS3 CCB
      STB CNFIG    FIND NEXT INSTR IN ASSEMBLED CODE
      JSB YINS     AFTER INSERT
      JMP XINS5
      STB CNFIG    SAVE SCB ADDRESS
      JSB STCD,I   STORE CODE
```

```
      LDA ZADD
      STA SAVR     SAVE ASSEMBLY ADDRESS
      LDA ASME2    INSTRUCTION AFTER INSERT
      CPA XUSRP    INSTR RESIDE IN FIRST LOCATION
      JMP XINS6    YES
      CCA          NO, SUBTRACT 1 FROM ADDR OF
      ADA ASME2    NEXT STATEMENT IN PROGRAM
      STA ASME2
      JSB JMPAF    INSERT JUMPS
XINS4 LDA SAVR     ADDRESS TO LINK EDIT WITH
      LDB XUSRP    BEGINNING OF PROGRAM
      JSB JMPE1
      LDA EDNUM    MULTIPLE INSERT
      CPA .4
      JMP ENDM3    RETURN TO APROPRIATE PROGRAM
      JMP SCBI,I
*
* NO MACHINE CODE PRECEDES OR FOLLOWS INSERT
*
XINS5 CLB
      STB CNFIG
      LDA EUSRP    RETRIEVE EDIT POINTER
      STA ZUSRP
      JSB STCD,I   STORE CODE
      JSB EDTAD    RESET EDIT POINTERS
      JMP SCBI,I   STORE IN SCB
*
*
* MOVE CODE IF FIRST AREA IN PROGRAM MUST BE RETAINED
*
XINS6 LDB CNFIG    SCB ADDRESS
      JSB CMVE,I   MOVE THE CODE
      JSB JMPAF    INSERT JUMPS TO LINK CHANGES
      JMP XINS4
```

```
*
*
* FIND NEXT INSTRUCTION IN ASSEMBLED PROGRAM AFTER
* AN INSERT
*
* RETURN P+1 EIDT TEXT LINKED WITH PROGRAM
*        P+2 EDIT TEXT NOT LINKED WITH PROGRAM
*
*
YINS  NOP
      LDA ASME2
      LDB SCBE2
     CPB ENEXT      END OF USER PROGRAM
      JMP YINS3
YINS1 SZA,RSS
      JMP YINS2     COMMENT
      SSA
      JMP YINS2     DATA
      ISZ YINS
      JMP YINS,I    MACHINE INSTRUCTION
*
YINS2 LDB B,I
      CPB ENEXT     END OF PROGRAM
      JMP YINS3     YES
      ADB .4
      LDA B,I
      ADB M4
      STA ASME2     SAVE FOR INSERTING JUMPS
      JMP YINS1
```

```
*
* INSERT FOLLOWS LAST MACHINE CODE STATEMENT IN THE PROGRAM
*
YINS3 LDB CNFIG
      CPB M1        CALL FROM SUBROUTINE XINS
      JMP YINS,I    YES
      LDB EDNUM     REPLACE OPERATION
      CPB .5
      RSS           YES
      JMP YINS4     NO
      LDA ZADD
      LDB ASME1     LINK PROGRAM WITH REPLACEMENT
      JSB JMPE1
      LDA EUSRP
      LDB ZUSRP     LINK REPLACEMENT TO PROGRAM
      JSB JMPE1
      ISZ ZUSRP     ADVANCE PROGRAM POINTER
      JSB STCK,I    CHECK FOR OVERFLOW
       JMP SCBI,I
*
YINS4 LDB ASME0     ADDR WHERE JUMP ORIGINATES
      LDA EDTSV     ADDR WHERE JUMP RESULTS
      JSB JMPS
*
      JSB EDTAD     UPDATE EDIT LINK POINTERS
      JMP YINS,I
```

```
*
*
* PREPARE FOR AND BEGIN MACHINE CODE MULTIPLE INSERT
*
*
MULIN NOP
      CLA
      STA EDCLR
      STA VETCK
      LDB SCBE0
      JMP *+3
MLN1  ADB M3
      LDB B,I
      STB SAVR     RETAIN SCB ADDRESS
      CPB M1       TERMINATION
      JMP MLN2
      ADB .4       ASSEMBLY ADDRESS
      LDA B,I
      SZA,RSS
      JMP MLN1     COMMENT
      SSA
      JMP MLN1     DATA
      STA ASME0    SAVE ASSEMBLY ADDR
      RSS
MLN2  STB EDCLR    NO ASSEMBLED CODE PRECEDES INSERT
      LDB SCBE2
      RSS
MLN3  LDB B,I      ADDR OF NEXT SCB ENTRY
      STB CNFIG    SAVE ADDRESS
```

```
      CPB ENEXT    TERMINATION
      JMP MLN4     YES
      ADB .4
      LDA B,I      ADDRESS OF ASSEMBLY
      ADB M4
      SZA,RSS
      JMP MLN3
      SSA
      JMP MLN3
      STA ASME2
      RSS
MLN4  STB VETCK    NO ASSEMBLED CODE FOLLOWS INSERT
      JSB ISCB     CLEAR UP SCB REFERENCES
      LDB EDCLR    ASSEMBLED CODE PRECEDE INSERT
      SZB
      JMP MLN5     NO
      JSB SVPSN
      LDA ASME0
      LDB SAVR     MOVE CODE BEFORE INSERT
      JSB CMVE,I
      JSB STCD,I   STORE INSERTED CODE
      JMP SCBI,I
*
* NO ASSEMBLED CODE PRECEDES INSERT
*
MLN5  JSB STCD,I   STORE CODE
      LDA ZADD     SAVE POSITION
      STA SAVR     SAVE POSITION
      JMP SCBI,I
```

```
*
*
* * END A MULTIPLE INSERT OPERATION
*
*
ENDMI NOP
      LDB MCMIP     M C MULTIPLE INSERT
      SZB,RSS
      JMP ENDM3
      LDA EDCLR     ASSEMBLED CODE PRECEDE INSERT
      SZA,RSS
      JMP ENDM1     YES
      LDA VETCK     NO, ASSEMBLE CODE FOLLOW INSERT
      SZA,RSS
      JMP XINS4-6   YES
      JSB EDTAD     NO
      JMP ENDM3
ENDM1 LDA VETCK     ASSEMBLED CODE FOLLOW INSERT
      SZA
      JMP ENDM2     NO
      LDA ASME2
      LDB CNFIG     MOVE CODE FOLLOWING INSERT
      JSB CMVE,I
      LDA EDTSV
      LDB ASME0
      JSB JMPS
      JSB JMPAF     LINK INSERT BACK INTO PROGRAM
      JMP ENDM3
*
ENDM2 LDA EDTSV
      LDB ASME0     STORE JUMP TO LINK INSERTED CODE
      JSB JMPS
      JSB EDTAD
ENDM3 CLA
      STA MIIP      CLEAR MULTIPLE INSERT FLAGS
      STA MCMIP
      JMP ENDMI,I
```

```
*
*
* SET SOURCE CODE BLOCK POINTERS FOR A REPLACE OPERATION
*
*
RSCB   NOP
       LDB ENM1
       CPB FSTMT      REPLACE FIRST STATEMENT
       RSS            YES
       JMP RSCB1      NO
       LDA ADDR1
       STA FIRST      POINTER TO FIRST STATEMENT
       LDB SCBE2      SUCCESSOR STATEMENT
       STB A,I        ADDR OF NEXT IN NEW STATEMENT
       INB
       STA B,I        ADDR OF PREV IN NEXT STATEMENT
       CCB            -1 TERMINATOR FOR BEGINNING
       INA            OF SCB
       STB A,I        STORE TERMINATOR
       JMP RSCB3
*
RSCB1  LDA ENM1       REPLACE LAST STATEMENT
       CPA CUSTN
       JMP RSCB2      YES
       JSB ISCB       NO
       JMP RSCB,I
*
RSCB2  LDA ADDR1      REPLACE LAST STATEMENT
       STA PREV       LAST STATEMENT AFTER EDIT
       STA SCBE0,I    ADDR OF NEXT IN PREV INSTR
       LDB SCBE0      ADDR OF PREV STAMENENT
       INA
       STB A,I        STORE ADDR OF PREV
RSCB3  LDB ENM1       STATEMENT NUMBER
       INA
       STB A,I        STORE STATEMENT NUMBER
       JMP RSCB,I
```

```
      ORG 14000B
*
*
* INITIALIZE STORAGE FOR EACH NEW PROGRAM
*
*
*
*
*     THE FOLLOWING TABLES WILL BE INITIALIZED
*
* THE SOURCE CODE BLOCK (SCB) FOR STORING USER SOURCE
*        PROGRAMS
*
* THE MAIN SYMBOL TABLE
*
* THE SPECIAL SYMBOL TABLE (SST) FOR COMPOUND OPERANDS
*
* THE PROGRAM LOCATION COUNTER (PLC) TABLE FOR UNDEFINED
*        PLC REFERENCES
*
* THE FREE SPACE TABLE FOR HOLDING ADDRESSES AND
*        LENGTHS OF DELETIONS FROM THE SCB
*
* THE USER PROGRAM AREA FOR BOTH MACHINE INSTRUCTIONS
*        AND DATA DEFINITIONS
*
*
GREET CLC 0,C        TURN OFF ALL I/O
      STF 0          TURN ON INTERRUPT SYSTEM
*
* CONFIGURE I/O SUBROUTINES
*
      LDA .15        PREPARE I/O SUBROUTINES FOR
      JSB CNFIG      I/O THROUGH TTY
      JSB IOFF,I     TURN OFF INTERRUPT
```

```
*
* SET MAIN FRAME INTERRUPT LOCATIONS FOR EACH NEW
* USER PROOGRAM
*
      LDB .2         FIRST ADDRESS TO BE SET
      LDA MPPEX      JUMP TO FORWARD REFERENCE WARNING
      STA B,I
      INB
      STA B,I
      INB            ADVANCE ADDRESS
      LDA HLT4       POWER FAIL HALT
      STA B,I
      INB
      LDA HLT5       MEMORY PROTECT / PARITY ERROR HALT
      STA B,I
      INB
      LDA DMAI       JUMP TO DMA SERVICE ROUTINE
      STA B,I
      LDB .9
      LDA DCI        JUMP TO DATA CHANNEL SERVICE ROUTINE
      STA B,I
      INB
      LDA CCI        CONTROL SERVICE ROUTINE
      STA B,I
*
* INITIALIZE LENGTH AND ADDRESS POINTERS FOR INPUT FROM DISC
*
      LDA TRACK      DISC ADDRESS OF DATA
      STA TEMP6
      LDA BUFL       BUFFER LENGTHS FOR OUTPUT
      STA TEMP7
      LDA XSTBL      MEMORY ADDR TO STORE INPUT FROM DISC
      STA ADDR1
*
* PREPARE TO PRINT FIRST PAGE OF INTRODUCTARY TEXT
*
      LDA TEMP6,I    DISC ADDRESS
      LDB TEMP7,I    LENGTH OF INPUT
```

```
*
* READ DATA FROM DISC AND OUTPUT TO USER
*
* USER MAY SPECIFY OPTIONAL I/O DEVICE
*
      JSB GRTIO     READ FROM DISC, THEN PRINT
      JSB DATN,I    READ RESPONSE, RETURN FIRST CHAR
      CPA S         OUTPUT TO CRT SCREEN
      RSS           YES
      JMP GRT6      NO
      LDB .11
      JSB CNFIG     CONFIGURE I/O SUBROUTINES
      JSB IOFF,I
*
* PRINT SECOND PAGE OF INTRODUCTION
*
* OPTIONAL SEQUENCING RESPONSE AVAILABLE
*
GRT6  LDA XSTBL     MEMORY ADDR TO STORE INPUT
      STA ADDR1
      ISZ TEMP6
      ISZ TEMP7
      LDA TEMP6,I   DISC ADDRESS
      LDB TEMP7,I   LENGTH OF INPUT
      JSB GRTIO     READ THEN PRINT DISC INPUT
*
* CLEAR USER PROGRAM TABLES BEFORE READING USER RESPONSE
*
      LDA CLRTB
      STA TEMP
      LDA XSTBL     STARTING ADDR OF SYMBOL TABLE
      CLB,RSS
      INA           ADVANCE TO NEXT LOCATION
      STB A,I
      ISZ TEMP
      JMP *-3
      LDA M125
      STA TEMP
      LDA XSTBL
      ADA .3
```

```
       STA ADDR1
       LDA B700      UNDEF FORWARD REF INDICATOR
       LDB B700      UNDEF INDIRECT FORWARD REFERENCE
       ADB .125      INDICATOR
       JMP *+6
GRT1   STA SAVA      SAVE (A)
       LDA ADDR1     ADVANCE ADDRESS IN SYMBOL TABLE
       ADA .5
       STA ADDR1
       LDA SAVA      RESTORE (A)
       INA
       INB
*
* STORE FORWARD REFERENCE POINTERS FOR DIRECT AND
* INDIRECT REFERENCES IN MAIN SYMBOL TABLE
*
       STA ADDR1,I   STORE APPROPRIATE SYMBOL TABLE
       ISZ ADDR1     REFERENCE FLAG
       STB ADDR1,I
       ISZ TEMP
       JMP GRT1
       LDA M75
       STA TEMP
       LDA XSST      BASE ADDR OF SPECIAL SYM TBL
       ADA .2
       RSS
GRT2   ADA .4
*
* FORWARD REFERENCE INDICATOR FOR SST
*
       INB           STORE SPECIAL SYMBOL TABLE
       STB A,I       INDICATOR FOR UNDEFINED REF
       ISZ TEMP
       JMP GRT2
       LDA YUSRP     UPPER BOUND OF USER PROGRAM
       INA
       LDB XRTRN     RETURN FROM EXECUTION
       STB A,I       STORE RETURN FROM EXECUTION
```

```
*
* INITIALIZE SYSTEM VARIABLES
*
      CCA
      STA GRTFG    SET GREET FLAG
      STA PREV     PREVIOUS ENTRY SET AS -1
      LDA XSCB
      STA FIRST    FIRST ENTRY IN SOURCE CODE BLOCK
      STA NEXT     NEXT ENTRY IN SOURCE CODE BLOCK
      LDA XUSRP
      STA ZUSRP    NEXT LOCATION USER PROG AREA
      LDA XDATA
      STA ZDATA    NEXT LOCATION IN PROG DATA AREA
      LDB YDAT
      STB YDATA
      CLB          INITIALIZE VARIABLES
      STB ABSSF    ABS/BSS PSEUDO OP FLAG
      STB DMPFG    DUMP FLAG
      STB EDINT    EDIT INPUT REQUEST
      STB EDLX     SOURCE DURING EDIT
      STB EDTFG    EDIT FLAG
      STB LBCNT    COUNT SYMBOL TABLE ENTRIES
      STB MCMIP    CLEAR MULTIPLE INSERT FLAGS
      STB MIIP
      STB SAVA
      STB SAVB     DUMP VARIABLES
      STB SAVEO
      STB SEQFG    SEQUENCE DIRECTIVE FLAG
*
* RESPONSE TO SEQUENCE REQUEST
*
*
GRT8  JSB DATN,I   READ RESPONSE
      CPA S        STATEMENT NUMBER REQUEST
      JMP GRT10    YES
      CLA          NO
      STA CUSTN    CURRENT USER STATEMENT NUMBER
      LDB .10
      STB FSTMT    FIRST STATEMENT NUMBER
      STB STINC    STATEMENT NUMBER INCREMENT
      JMP GRT12
```

```
*
*
*
GRT10 JSB SQNC,I
      JMP GRT8      ERROR
*
*
* THE THIRD PAGE OF USER OUTPUT OFFERS THE OPTION:
*
*    TO THOSE FAMILIAR WITH THE ASSEMBLER PROGRAM
*    ENTRY MAY BEGIN
*
*    ELSE INSTRUCTIONAL TEXT CAN BE PRESENTED TO
*    AQUAINT THE INEXPERIENCED WITH THE SYSTEM
*
* READ RESPONSE C TO CONTINUE
*               L TO LEARN
*
*
GRT12 LDA DSIPT     MEMORY ADDR FOR FURTHER DISC INPUT
      STA ADDR1
      ISZ TEMP6
      ISZ TEMP7
      LDA TEMP6,I   DISC ADDRESS
      LDB TEMP7,I   INPUT LENGTH
      JSB GRTIO
      JSB DATN,I
      CPA L         PRINT INSTRUCTIONAL TEXT
      RSS           YES
      JMP GRT20     NO
```

```
*
*
* PRINT INSTRUCTIONAL PAGES
*
*
* READ RESPONSE C TO CONTINUE
*               S TO START
*
*
      LDA M8
      STA TEMP5
      LDA DSIPT    MEMORY ADDR FOR DISC INPUT
      STA ADDR1
GRT14 ISZ TEMP6
      ISZ TEMP7
      LDA TEMP6,I  DISC ADDR
      LDB TEMP7,I  INPUT LENGTH
      JSB GRTIO
      ISZ TEMP5
      RSS
      JMP GRT20    ALL TEXT PRINTED
      JSB DATN,I
      CPA S        START
      RSS          YES
      JMP GRT14
*
* CLEAR MAIN FRAME INTERRUPT LOCATIONS
*
GRT20 LDA M16
      STA TEMP
      LDB .5
      CLA
      INB
      STA B,I
      ISZ TEMP
      JMP *-3
      STA GRTFG    CLEAR GREET FLAG
*
* READ FIRST SOURCE PROGRAM STATEMENT
*
```

```
      JSB DATN,I
      JSB IMON,I   TURN ON INTERRUPT
      JSB CLER,I   CLEAR LEXICAL VARIABLES
      JMP LXANL,I  JUMP TO LEXICAL SCAN
*
*
* READ AND PRINT INTRODUCTARY TEXT FROM DISC
*
* ENTER (A) DISC ADDRESS
*       (B) LENGTH OF INPUT (WORDS)
*
*
GRTIO NOP
      STB TEMP1
      CMB,INB
      STB LENTH    NEGATIVE WORD COUNT FOR DMA
      LDB DMACW    OUTPUT FIRST DMA CONTROL WORD
      OTB 6
      CLB
      STB HDMSK    DISC HEAD MASK
      LDB ADDR1    MEMORY ADDRESS FOR INPUT
      JSB DISKI
      LDA M12
      JSB NWLS,I
      LDA TEMP1    LENGTH OF INPUT (WORDS)
      ALS          LENGTH OF INPUT (CHARACTERS)
      LDB ADDR1
      JSB WRITE,I
      JMP GRTIO,I
```

```
*
* PAGE LENGTH (WORDS) OF TEXT
*
BUFL  DEF *+1
      OCT 1111      PAGE 1
      OCT 746       PAGE 2
      OCT 212       PAGE 3
      OCT 457       PAGE 4
      OCT 345       DUMP
      OCT 312       LIST
      OCT 416       SEQUENCE
      OCT 345       XECUTE
      OCT 542       EDIT 1
      OCT 447       EDIT 2
      OCT 553       LAST
*
* DISC ADDRESS OF INTRODUCTARY TEXT
*
* DATA BEGINS ON FIRST SECTOR OF FIRST TRACK ON CARTRIGDE
* DISC
*
TRACK DEF *+1
      OCT 400       PAGE 1
      OCT 405       PAGE 2
      OCT 411       PAGE 3
      OCT 413       PAGE 4
      OCT 416       DUMP
      OCT 420       LIST
      OCT 422       SEQUENCE
      OCT 425       XECUTE
      OCT 1000      EDIT 1
      OCT 1003      EDIT 2
      OCT 1006      LAST
*
*
      END
```

```
      ORG 15200B
*
* MNEMONIC TABLE
*
* FIRST TWO LETTERS OF MNEMONIC
*
      ASC 2,ABAD
      ASC 7,ADALALALANARAS
      ASC 7,ASASBLBLBLBRBS
      ASC 7,CCCCCCCLCLCLCL
      ASC 7,CLCLCMCMCMCPCP
      ASC 7,DEDEDIDLDSELEL
      ASC 7,ENEQEREPHLININ
      ASC 7,IOISJMJSLDLDLI
      ASC 7,LILSLSMIMIMPNO
      ASC 7,OCOTOTRARARBRB
      ASC 7,RRRRRSSESFSFSL
      ASC 7,SLSOSOSSSSSTST
      ASC 7,STSTSTSWSZSZXO
*
* THIRD LETTER OF MNEMONIC AND INSTRUCTION NUMBER
*
      OCT 051415,040406,041006,043001
      OCT 051001,051401,042006,051401
      OCT 041411,046005
      OCT 051005,043001,051001,051401
      OCT 051401,051416,040401,041001
      OCT 042401,040401,041001,041404
      OCT 042401,043003,047401,040401
      OCT 041001,042401,040406,041006
      OCT 041412,043017,053007,042007
      OCT 052007,040401,041001,042010
      OCT 052414,040401,041001,052004
      OCT 040401,041001,051006,055006
```

```
      OCT 050006,041006,040406,041006
      OCT 040404,041004,046005,051005
      OCT 040404,041004,054407,050001
      OCT 052013,040404,041004,046001
      OCT 051001,046001,051001,046005
      OCT 051005,051401,055001,041403
      OCT 051403,040401,041001,041402
      OCT 051402,040401,041001,040406
      OCT 041006,041404,043003,047401
      OCT 050001,040401,041001,051006
*
* SKELETON OF ASSEMBLED CODE
*
      OCT 177777,040000,044000,001700
      OCT 001400,001000,010000,001100
      OCT 177777,100020
      OCT 101020,005700,005400,005000
      OCT 005100,177777,003400,007400
      OCT 002300,002400,006400,106700
      OCT 002100,103100,103101,003000
      OCT 007000,002200,050000,054000
      OCT 177777,177777,100400,104200
      OCT 104400,001600,005600,177777
      OCT 177777,001500,005500,102000
      OCT 002004,006004,030000,034000
      OCT 024000,014000,060000,064000
      OCT 102500,106500,100040,101040
      OCT 102400,106400,100200,000000
      OCT 177777,102600,106600,001200
      OCT 001300,005200,005300,100100
      OCT 101100,002001,002040,102200
      OCT 102300,000010,004010,102201
      OCT 102301,002020,006020,070000
      OCT 074000,102700,102100,102101
      OCT 101100,002002,006002,020000
```

APPENDIX H

BIBLIOGRAPHY

BIBLIOGRAPHY

(1) BROWN, P.J., Recreation of Source Code from Reverse Polish Form, Software - Practice and Experience, Vol 2, 275-278, 1972, John Wiley and Sons, New York.

(2) HULL, T.E., DAY, D.D.F., Computers and Problem Solving Addison Weslex, Don Mills, 1970.

(3) KATZAN, HARRY, Batch, Conversational and Incremental Compilers, Proc. APIFS 1969 SJCC, Vol 34, 47-56, APIFS Press.

(4) LAMPSON, B., Interactive Machine Language Programming Proc. AFIPS 1965 FJCC, Vol. 27 part 1, 473-482, Macmillan and Co., London.

(5) LOCK, K., Structuring Programs for Multi-Program Time-Sharing On-Line Applications, Proc. AFIPS 1965 FJCC, Vol 27 part 1, 457-472, Macmillan and Co., London.

(6) SCHWARTZ, JULES I., On line Programming, CACM 9, No. 3, 199-202, March 1966.

(7) SMITH, L.B., The Use of Interactive Graphics to Solve Numerical Problems, CACM 13, No. 10, 625-634, October 1970.

(8) HEWLETT PACKARD COMPANY, 2100A Computer: Reference Manual, HP 02100-90001, December 1971, Hewlett Packard Company, Cupertino, California.

(9) HEWLETT PACKARD COMPANY, HP Assembler, HP 02116-9014, June 1971, Hewlett Packard Company, Cupetino, California.

(10) HEWLETT PACKARD COMPANY, Moving Head Disc Operating System, HP 02116-91779, March 1971, Hewlett Packard Company, Cupertino, California.