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 I: INCREMENTAL ASSEMBLY, CONCEPTS AND CONSEQUENCES

Assemblers 1
Batch, Conversational and Incremental Systems 2
Basic Definitions 2
Batch Environment 2
Conversational Concepts 3
Incremental System Overview 5
Incremental Execution 7
Summary 8
Considerations for Interactive Programming 9
Interactive Utilization to Users 11
Programming Process 12
Conclusions 13

CHAPTER II: IMPLEMENTATION - BASIC CONCEPTS 14

Introduction 14
Standard Assembly 15
Simple Incremental Assembly 16
Forward References 17
Defined Memory Reference Instructions 18
Introductory Text 18
System Directives 18
:DUMP 19
:EDIT 19
CHAPTER III: ASSEMBLER IMPLEMENTATION

Introduction
Source Program Assembly
Mnemonics and Pseudo Operations
Assembler Control Statement
Instruction Modifications
Assembler Tables
Instruction Table
User Program Tables
Main Symbol Table
Special Symbol Table
Program Location Counter Table
The Source Code Block
Free Space Table
User Program Areas
Instruction Assembly
Forward References
Program Segments
Error Message Processor
Subroutine ERROR

CHAPTER IV: INITIALIZATION

Introduction
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NWLNS</td>
<td>59</td>
</tr>
<tr>
<td>DATIN</td>
<td>59</td>
</tr>
<tr>
<td>TTY.I</td>
<td>60</td>
</tr>
<tr>
<td>PROCS</td>
<td>60</td>
</tr>
</tbody>
</table>

Input Control

### Binary to Ascii Conversion

### Subroutines CNOCT and CNDEC

### CHAPTER VI: LEXICAL SCAN AND NUMBER MANIPULATION

Lexical Scan

Introduction

Subroutine LEX

Introduction

Source Statement Scan

Program Modifications

Character Manipulation Subroutines

- Subroutine BCKSP
- Subroutine GETCR
- Subroutine NTBLK
- Subroutine RDCOM
- Subroutine TRMCK

Lexical Support Routines

- Subroutine LABRD
- Subroutine LETPR
- Subroutine LOKUP

viii
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND</td>
<td>76</td>
</tr>
<tr>
<td>MNEM</td>
<td>77</td>
</tr>
<tr>
<td>RANGE</td>
<td>79</td>
</tr>
<tr>
<td>OPREC</td>
<td>79</td>
</tr>
<tr>
<td>STDAT</td>
<td>79</td>
</tr>
<tr>
<td>LABCK</td>
<td>80</td>
</tr>
<tr>
<td>DATRG</td>
<td>80</td>
</tr>
<tr>
<td>VAL</td>
<td>80</td>
</tr>
<tr>
<td>Number Manipulation</td>
<td>82</td>
</tr>
<tr>
<td>Introduction</td>
<td>82</td>
</tr>
<tr>
<td>OCTIN</td>
<td></td>
</tr>
<tr>
<td>OCTCK</td>
<td>83</td>
</tr>
<tr>
<td>NUMBR</td>
<td>83</td>
</tr>
<tr>
<td>DECHK</td>
<td>84</td>
</tr>
<tr>
<td>Dec Pseudo Op</td>
<td>85</td>
</tr>
<tr>
<td>CONST</td>
<td>85</td>
</tr>
<tr>
<td>NUMCK</td>
<td>85</td>
</tr>
<tr>
<td>Decimal Integers</td>
<td>86</td>
</tr>
<tr>
<td>TYPCK</td>
<td>86</td>
</tr>
<tr>
<td>IFIX</td>
<td>87</td>
</tr>
<tr>
<td>GTNUM</td>
<td>87</td>
</tr>
<tr>
<td>TWINT</td>
<td>87</td>
</tr>
</tbody>
</table>

Summary

CHAPTER VII: ASSEMBLY AND STORAGE 100

Introduction ix
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Assembly</td>
<td>100</td>
</tr>
<tr>
<td>Subroutine SETCD</td>
<td>100</td>
</tr>
<tr>
<td>Data Definitions</td>
<td>101</td>
</tr>
<tr>
<td>Machine Instructions</td>
<td>101</td>
</tr>
<tr>
<td>Memory Reference Operand Evaluation</td>
<td>101</td>
</tr>
<tr>
<td>Assembly Routines</td>
<td>104</td>
</tr>
<tr>
<td>Subroutine DETLN</td>
<td>104</td>
</tr>
<tr>
<td>Subroutine STRCD</td>
<td>104</td>
</tr>
<tr>
<td>Subroutine DTSET</td>
<td>104</td>
</tr>
<tr>
<td>Subroutines STRCK and DATPL</td>
<td>104</td>
</tr>
<tr>
<td>Subroutine STLBL</td>
<td>105</td>
</tr>
<tr>
<td>Subroutine STPLC</td>
<td>105</td>
</tr>
<tr>
<td>Statement Storage</td>
<td>105</td>
</tr>
<tr>
<td>Introduction</td>
<td>106</td>
</tr>
<tr>
<td>Subroutine ASMBL</td>
<td>106</td>
</tr>
<tr>
<td>Subroutine STSCB</td>
<td>107</td>
</tr>
<tr>
<td>Subroutine LBDEF</td>
<td>108</td>
</tr>
<tr>
<td>Subroutine FWDRF</td>
<td>108</td>
</tr>
<tr>
<td>CHAPTER VIII: SYSTEM DIRECTIVES</td>
<td>110</td>
</tr>
<tr>
<td>Introduction</td>
<td>110</td>
</tr>
<tr>
<td>ABORT</td>
<td>110</td>
</tr>
<tr>
<td>DUMP</td>
<td>110</td>
</tr>
<tr>
<td>DUMP Subroutines</td>
<td>112</td>
</tr>
<tr>
<td>EDIT</td>
<td>112</td>
</tr>
<tr>
<td>HALT</td>
<td>112</td>
</tr>
</tbody>
</table>
CHAPTER IX: THE EDITOR

Introduction

Edit Instruction Scan

Overview

Source Program Edit

Subroutine DSCB

Subroutine ISCB

Subroutine RSCB

Data Edit Operations

Subroutine DTEDD

Subroutine DTEDI

Subroutine SCSYM

Machine Code Edit Operations
Introduction 132
Single and Multiple Delete 134
Single and Multiple Insert 136
Replace 137
Edit Subroutines 138
Subroutine PREPR 138
Subroutine DELTE 138
Subroutine CMOVE 139
Subroutine CASCD 139
Subroutine JMPAF and JMPBF 140
Subroutine JMPS 140
Subroutine JMPEL 140
Subroutine STFSP 140
Subroutine SNGDL 140
Subroutine XDEL 141
Subroutine XINS 141
Subroutine YINS 142
Subroutine MULIN 143
Subroutine ENDMI 143
Subroutine EDIPT 143
Edit Subsystems 144
Introduction 144
Single Delete 144
Multiple Delete 145
Single Insert 147
Multiple Insert 147
Replace 149
End 150
Conclusions 150

APPENDIX A: Assembler Machine Instructions and Pseudo ops 153
APPENDIX B: The Introductory Text 162
APPENDIX C: Direct Memory Access 194
APPENDIX D: Non-Interrupt Transfer Routines 197
APPENDIX E: Dump and List Output 200
APPENDIX F: Memory Map and Functional Unit Relation Chart 205
APPENDIX G: Source Program Listing 218
APPENDIX H: Bibliography 443
| Table 3.1 | The User Program Tables          | 41 |
| Table 3.2 | Forward Reference Linkage        | 42 |
| Table 3.3 | Base Page Error Messages         | 43 |
| Table 5.1 | Input/Output Subroutines in      | 62 |
|           | Functional Groups                |    |
| Table 6.1 | Lexical Error Messages           | 89 |
| Table 6.2 | Character Manipulation Subroutines | 91 |
| Table 6.3 | Lexical Support Routines         | 92 |
| Table 6.4 | Error Messages for Lexical       | 93 |
|           | Support Routines                 |    |
| Table 6.5 | Number Program Error Messages    | 94 |
| Table 7.1 | Auxiliary Assembly Subroutines   | 109|
| Table 8.1 | Dump Error Messages              | 122|
| Table 8.2 | List and Sequence Error Messages | 123|
| Table 9.1 | Editor Error Messages            | 152|
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>System Controller Flow Diagram</td>
<td>63</td>
</tr>
<tr>
<td>6.1</td>
<td>Subroutine Lex Flow Diagram</td>
<td>95</td>
</tr>
</tbody>
</table>
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 editing during assembly. Incremental mode, used normally conversationally, 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.
"An interactive system is only useful if 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.

Editing 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.
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 peripheral 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.
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 editing 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 suppressed 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 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

INTRODUCTION

The major design and implementation 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 Manual⁹ 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:

\[
(+) \text{ (symbol)}(\pm \text{ integer})(,I).
\]
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 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 definition 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 402₈ 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 editing 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 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.
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.

The last two tables are the user program areas for data definitions and machine instructions having 400\text{8} and 340\text{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\text{8} locations were available for the user program (700\text{8} words) and the data area (1000\text{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 may be described in terms of functional units or segments of storage. In planning the overall 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

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>026001</td>
<td></td>
<td>First address of program address table corresponding to first address of the user program area</td>
</tr>
<tr>
<td>026337</td>
<td></td>
<td>Last address of program address table corresponding to the last address of the user program area</td>
</tr>
<tr>
<td>026340</td>
<td>PROG</td>
<td>Entry/Exit point for executing the user program</td>
</tr>
<tr>
<td>026341</td>
<td>XUSR</td>
<td>First address of user program area</td>
</tr>
<tr>
<td>026677</td>
<td>USRF</td>
<td>Last address of user program area</td>
</tr>
<tr>
<td>026700</td>
<td></td>
<td>Return jump from user program to calling point</td>
</tr>
<tr>
<td>026701</td>
<td>XDATA</td>
<td>First address of data address area</td>
</tr>
<tr>
<td>027277</td>
<td></td>
<td>Last address of data address area</td>
</tr>
<tr>
<td>027301</td>
<td></td>
<td>First address for data value storage</td>
</tr>
<tr>
<td>027677</td>
<td></td>
<td>Last address for data value storage</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Symbol Table Address Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 1</td>
</tr>
<tr>
<td>Word 2 Symbol name stored as Ascii characters</td>
</tr>
<tr>
<td>Word 3</td>
</tr>
<tr>
<td>Word 4 341 Page address of first direct and</td>
</tr>
<tr>
<td>Word 5 353 indirect forward references</td>
</tr>
<tr>
<td>Word 6</td>
</tr>
</tbody>
</table>

Memory Address

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>026341 XXX372</th>
</tr>
</thead>
<tbody>
<tr>
<td>026353 XXX364</td>
<td></td>
</tr>
<tr>
<td>026364 XX1076</td>
<td>Return pointer for indirect reference</td>
</tr>
<tr>
<td>026372 XXX417</td>
<td></td>
</tr>
<tr>
<td>026417 XXX701</td>
<td>Return pointer for direct reference</td>
</tr>
</tbody>
</table>
## TABLE 3.3 BASE PAGE ERROR MESSAGES

<table>
<thead>
<tr>
<th>LABEL</th>
<th>ERROR MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR1</td>
<td>BAD DATA INPUT</td>
</tr>
<tr>
<td>ERR2</td>
<td>STATEMENT NUMBER OUT OF RANGE</td>
</tr>
<tr>
<td>ERR3</td>
<td>OPERAND VALUE OUT OF RANGE</td>
</tr>
<tr>
<td>ERR4</td>
<td>ILLEGAL OPERAND TERMINATION</td>
</tr>
<tr>
<td>ERR5</td>
<td>ILLEGAL CHARACTER BEGINS LABEL</td>
</tr>
<tr>
<td>ERR6</td>
<td>NO OPERAND FOUND</td>
</tr>
<tr>
<td>ERR7</td>
<td>OPERAND IS UNDEFINED</td>
</tr>
<tr>
<td>ERR8</td>
<td>UNDEFINED LABEL IN OPERAND</td>
</tr>
<tr>
<td>ERR9</td>
<td>NO LABEL FOUND</td>
</tr>
</tbody>
</table>
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 words holding the basic binary loader the first 100 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 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 $\text{SQNCE}$ reads in the statement numbers for the sequence request. $\text{SQNCE}$ is also used for the Sequence Directive introduced in Chapter II; $\text{SQNCE}$ is discussed in Chapter VIII with the discussion of System Directives.

The remaining subprograms $\text{CNFIG}$, $\text{GRTIO}$ and the disc input driver are used strictly for initialization purposes.

**Subroutine $\text{CNFIG}$**

**Calling Sequence**

$LDB < \text{Channel number of I/O device} >$

$\text{CNFIG}$ 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 $\text{GRTIO}$**

**Calling Sequence**

$LDA < \text{Disc address of input} >$

$LDB < \text{Input length (words)} >$

$\text{GRTIO}$ will call the disc input driver to read in a page of the introductory text and call subroutine $\text{TTY.P}$ 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 interpreted 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,
- **ASMRL** 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

TTY.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**

\[
\text{LDA } < \text{Value to be converted} >
\]

**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
   - **NWLS**: 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

MAIN ENTRY POINT

READ INPUT

ABORT PROGRAM

INITIALIZE PROGRAM

TURN ON INTERRUPT

ABS/BSS OPERAND DEFINITION

LEXICAL SCAN SUBROUTINE VAL

INITIALIZE LEXICAL VARIABLES

DUMP REQUEST

DUMP DIRECTIVE
ENTRY AFTER EDIT
REPLACE OR INSERT
OPERATION

STORAGE STATEMENT
IN SCB

STORAGE STATEMENT
LABEL

MULTIPLE INSERT
Y

MULTIPLE INSERT
PROGRAM

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.
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 definition 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)

LETTPR 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.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>ERROR MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LXR1</td>
<td>FIRST CHARACTER NOT FOUND</td>
</tr>
<tr>
<td>LXR2</td>
<td>ILLEGAL FIRST CHARACTER</td>
</tr>
<tr>
<td>LXR3</td>
<td>BAD DATA Follows LABEL</td>
</tr>
<tr>
<td>LXR4</td>
<td>DOUBLY DEFINED LABEL</td>
</tr>
<tr>
<td>LXR5</td>
<td>INSTRUCTION NOT FOUND</td>
</tr>
<tr>
<td>LXR6</td>
<td>NO OPERAND FOUND</td>
</tr>
<tr>
<td></td>
<td>(ERR6)</td>
</tr>
<tr>
<td>LXR7</td>
<td>BAD DATA Follows OP CODE</td>
</tr>
<tr>
<td>LXR8</td>
<td>BAD DATA IN OPERAND FIELD</td>
</tr>
<tr>
<td>LXR9</td>
<td>ILLEGAL CHARACTER BEGINS LABEL</td>
</tr>
<tr>
<td></td>
<td>(ERR5)</td>
</tr>
<tr>
<td>LXR10</td>
<td>UNDEFINED LABEL IN OPERAND</td>
</tr>
<tr>
<td></td>
<td>(ERR8)</td>
</tr>
<tr>
<td>LXR11</td>
<td>ILLEGAL OPERAND TERMINATION</td>
</tr>
<tr>
<td></td>
<td>(ERR4)</td>
</tr>
<tr>
<td>LXR12</td>
<td>ILLEGAL INSTRUCTION DURING EDIT</td>
</tr>
<tr>
<td>LXR13</td>
<td>OPERAND VALUE OUT OF RANGE</td>
</tr>
<tr>
<td></td>
<td>(ERR3)</td>
</tr>
<tr>
<td>LXR14</td>
<td>NO LABEL PRECEDES EQU PSEUDO OP</td>
</tr>
<tr>
<td>LXR15</td>
<td>ADDRESS MUST BE POSITIVE</td>
</tr>
<tr>
<td>LXR16</td>
<td>INSTRUCTION NOT FOUND</td>
</tr>
<tr>
<td>LXR17</td>
<td>OPERAND IS UNDEFINED</td>
</tr>
<tr>
<td></td>
<td>(ERR7)</td>
</tr>
<tr>
<td>LXR18</td>
<td>UNDEFINED LABEL NOT PERMITTED WITH DEF DURING EDIT</td>
</tr>
<tr>
<td>LABEL</td>
<td>ERROR MESSAGE</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>LXR19</td>
<td>OPERAND VALUE MUST BE GREATER THAN ZERO</td>
</tr>
</tbody>
</table>
**TABLE 6.2 CHARACTER MANIPULATION SUBROUTINES**

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCKSP</td>
<td>Back space one character in the input buffer</td>
</tr>
<tr>
<td>GETCR</td>
<td>Retrieve the next character from the input buffer</td>
</tr>
<tr>
<td>NTBLK</td>
<td>Get the next non-blank character from the input</td>
</tr>
<tr>
<td>RDCOM</td>
<td>Read up to a comma in the buffer</td>
</tr>
<tr>
<td>TRMCK</td>
<td>Check for a termination character</td>
</tr>
<tr>
<td>SUBROUTINE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>DATRC</td>
<td>Check for data address</td>
</tr>
<tr>
<td>FIND</td>
<td>Find Symbol Table address of symbol</td>
</tr>
<tr>
<td>LABCK</td>
<td>Read in operand, examine symbol</td>
</tr>
<tr>
<td>LABRD</td>
<td>Read a symbol</td>
</tr>
<tr>
<td>LETPR</td>
<td>Check for period or a letter</td>
</tr>
<tr>
<td>LOKUP</td>
<td>Look up Symbol Table address</td>
</tr>
<tr>
<td>MNEM</td>
<td>Find assembled instruction from mnemonic</td>
</tr>
<tr>
<td>OPREC</td>
<td>Read in and interpret operand</td>
</tr>
<tr>
<td>RANGE</td>
<td>Check Channel Number and Shift Count range</td>
</tr>
<tr>
<td>STDAT</td>
<td>Store data value in temporary data buffer</td>
</tr>
<tr>
<td>VAL</td>
<td>Prompt definition of undefined ABS or BSS symbol</td>
</tr>
<tr>
<td>SUBROUTINE</td>
<td>ERROR MESSAGE</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DATRG</td>
<td>ADDRESS BEYOND PROGRAM BOUNDS</td>
</tr>
<tr>
<td>FIND</td>
<td>SYMBOL TABLE OVERFLOW</td>
</tr>
<tr>
<td>LABRD</td>
<td>NO LABEL FOUND</td>
</tr>
<tr>
<td>MNEM</td>
<td>ILLEGAL ASSEMBLER INSTRUCTION</td>
</tr>
<tr>
<td>OPREC</td>
<td>OPERAND VALUE OUT OF RANGE</td>
</tr>
<tr>
<td></td>
<td>ILLEGAL OPERAND TERMINATION</td>
</tr>
<tr>
<td></td>
<td>MINUS SIGN PRECEDES LABEL</td>
</tr>
<tr>
<td></td>
<td>MINUS SIGN PRECEDES ASTERISK</td>
</tr>
<tr>
<td></td>
<td>INDIRECT REFERENCE PERMITTED ONLY WITH MEMORY REFERENCE AND DEF INSTRUCTIONS</td>
</tr>
<tr>
<td>RANGE</td>
<td>OPERAND VALUE OUT OF RANGE</td>
</tr>
<tr>
<td>STDAT</td>
<td>DATA INPUT EXCEEDS IMPOSED LIMIT</td>
</tr>
<tr>
<td>LABEL</td>
<td>ERROR MESSAGE</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>NUMR1</td>
<td>NO OPERAND DATA FOUND</td>
</tr>
<tr>
<td>NUMR2</td>
<td>SOLITARY SIGN</td>
</tr>
<tr>
<td>NUMR3</td>
<td>BAD DATA INPUT</td>
</tr>
<tr>
<td>(ERR1)</td>
<td>BAD DATA FOLLOWS INTEGER</td>
</tr>
<tr>
<td>NUMR4</td>
<td>ERROR IN EXPONENT</td>
</tr>
<tr>
<td>NUMR5</td>
<td>INTEGER OVERFLOW</td>
</tr>
<tr>
<td>NUMR6</td>
<td>POSITIVE INTEGER EXPECTED</td>
</tr>
<tr>
<td>NUMR7</td>
<td>BAD DATA FOLLOWS INTEGER</td>
</tr>
<tr>
<td>NUMR8</td>
<td>REAL NUMBER OUT OF RANGE</td>
</tr>
</tbody>
</table>
Lexical Errors have not been included. The term "INTR NUM" is used to represent Instruction Number.
A

INTR NUM=2

Y

CLEAR FLAG MAY BE PRESENT

N

Y

INTR NUM=4

N

CHANNEL NUMBER

INTR NUM=4

Y

SHIFT COUNT

INTR NUM=5

N

B

INTR NUM=2

Y

CLEAR FLAG MAY BE PRESENT

INTR NUM=4

N

CHANNEL NUMBER

INTR NUM=4

Y

SHIFT COUNT

INTR NUM=5

N

B

RETURN

MASK OPERAND INTO ASSEMBLY SKELETON
INCREMENT ASSEMBLY LENGTH

INTR. NUM=6

Y

INCREMENT ASSEMBLY LENGTH

N

INTR. NUM=7

Y

EXAMINE MEMORY REFERENCE OPERAND

RETURN

N

EDIT IN PROGRESS

EXECUTE USER PROGRAM

Y

ERROR MESSAGE

SYSTEM CONTROLLER

N

INITIALIZE DATA BUFFER AND POINTERS

C
DEF

Y

READ Operand STORE IN DATA BUFFER

N

ERROR MESSAGE

ERROR MESSAGE

RETURN

EDIT IN PROGRESS

SYSTEM CONTROLLER

PROGRAM ERROR

D
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, DTSFT 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 EOU 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 SCR 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 7008 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 7008, all forward references have been defined; FWDRF may return to the calling program.
<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATFL</td>
<td>Check data table area for overflow</td>
</tr>
<tr>
<td>DETLM</td>
<td>Determine assembly length of Memory Reference instruction</td>
</tr>
<tr>
<td>DTSET</td>
<td>Assemble data definition</td>
</tr>
<tr>
<td>STLBL</td>
<td>Store symbol in Symbol Table</td>
</tr>
<tr>
<td>STPLC</td>
<td>Store program location counter reference</td>
</tr>
<tr>
<td>STRCD</td>
<td>Store assembled instruction in user program area</td>
</tr>
<tr>
<td>STRCK</td>
<td>Check user program area for overflow</td>
</tr>
</tbody>
</table>
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:

:LIST(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 output is presented in Appendix E.

Subroutine LIST

Calling Sequence

\[
\text{LDA} \ < \text{Positive value, call from System Directive } > \\
< \text{-1, call from editor } >
\]

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 suppressed.

**SEQUENCE**

The format for the Sequence Directive is:

:S(EOUENCE),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 SONCE, 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 SONCE**

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

Calling subroutine TWINT, SONCE 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 FWDREF 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 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

<table>
<thead>
<tr>
<th>LABEL</th>
<th>ERROR MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPER1</td>
<td>NO OPERAND FOUND</td>
</tr>
<tr>
<td>DPER2</td>
<td>NO LABEL FOUND</td>
</tr>
<tr>
<td>DPER3</td>
<td>UNDEFINED LABEL IN OPERAND</td>
</tr>
<tr>
<td>DPER4</td>
<td>OPERAND IS UNDEFINED</td>
</tr>
<tr>
<td>LABEL</td>
<td>ERROR MESSAGE</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ERR1</td>
<td>BAD DATA INPUT</td>
</tr>
<tr>
<td>ERR2</td>
<td>STATEMENT NUMBERS OUT OF RANGE</td>
</tr>
</tbody>
</table>
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:

\[
/R(\text{EPLACE}),M(,V).
\]

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:

\[
/E(\text{ND}).
\]

**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 edited. 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.

Editing 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 follows 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 around 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 edited code.

Subroutine JMPE1

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

JMPE1 inserts one jump instruction to link the edited 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 JMPFL 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 inadvertently 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 definitions 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.
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.
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**

<table>
<thead>
<tr>
<th>LABEL</th>
<th>ERROR MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDR1</td>
<td>ILLEGAL DATA PRECEDES EDIT INSTRUCTION</td>
</tr>
<tr>
<td>EDR2</td>
<td>UNDEFINED EDIT INSTRUCTION</td>
</tr>
<tr>
<td>EDR3</td>
<td>BAD DATA Follows EDIT INSTRUCTION</td>
</tr>
<tr>
<td>EDR4</td>
<td>VETO NOT PERMITTED ON AN INSERT</td>
</tr>
<tr>
<td>EDR5</td>
<td>STATEMENT NUMBER OUT OF RANGE</td>
</tr>
<tr>
<td>EDR6</td>
<td>ILLEGAL SOURCE TYPE ENTRY DURING EDIT</td>
</tr>
<tr>
<td>EDR7</td>
<td>STATEMENT NUMBER ALREADY DEFINED</td>
</tr>
<tr>
<td>EDR8</td>
<td>STATEMENT NUMBERS MUST ACCOMPANY EDIT INSTRUCTION</td>
</tr>
<tr>
<td>EDR9</td>
<td>STATEMENT NUMBER IS NOT DEFINED</td>
</tr>
</tbody>
</table>
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
MTB  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  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 (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

<table>
<thead>
<tr>
<th>GROUP NUMBER</th>
<th>INSTRUCTION TYPE</th>
<th>OPERAND REQUIRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALTER SKIP</td>
<td>NO OPERAND REQUIRED</td>
</tr>
<tr>
<td></td>
<td>REGISTER REFERENCE</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INPUT/OUTPUT</td>
<td>CLEAR FLAG may BE PRESENT</td>
</tr>
<tr>
<td>3</td>
<td>INPUT/OUTPUT</td>
<td>CHANNEL NUMBER EXPECTED</td>
</tr>
<tr>
<td>4</td>
<td>INPUT/OUTPUT</td>
<td>CHANNEL NUMBER EXPECTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLEAR FLAG MAY BE PRESENT</td>
</tr>
<tr>
<td>5</td>
<td>EXTENDED ARITHMETIC</td>
<td>NUMBER OF SHIFTS</td>
</tr>
<tr>
<td></td>
<td>REGISTER REFERENCES</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MEMORY REFERENCE</td>
<td>SYMBOL (ASTERISK)</td>
</tr>
<tr>
<td>7</td>
<td>EXTENDED ARITHMETIC</td>
<td>INTEGER</td>
</tr>
<tr>
<td></td>
<td>MEMORY REFERENCE</td>
<td>INDIRECT FLAG</td>
</tr>
</tbody>
</table>

**PSEUDO OPS**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>END</td>
<td>NO OPERAND REQUIRED</td>
</tr>
<tr>
<td>9</td>
<td>ASC</td>
<td>LENGTH AND LIST OF ASCII DATA</td>
</tr>
<tr>
<td>10</td>
<td>DEC</td>
<td>REALS OR DECIMAL INTEGERS</td>
</tr>
<tr>
<td>11</td>
<td>OCT</td>
<td>OCTAL INTEGERS</td>
</tr>
<tr>
<td>12</td>
<td>EQU</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>13</td>
<td>ABS</td>
<td>ADDRESS VALUE</td>
</tr>
<tr>
<td>14</td>
<td>BSS</td>
<td>VALUE</td>
</tr>
<tr>
<td>15</td>
<td>DEF</td>
<td>ADDRESS DEFINITION</td>
</tr>
</tbody>
</table>
### MACHINE INSTRUCTIONS

#### MNEMONIC CLASSIFICATION BY GROUP NUMBER

<table>
<thead>
<tr>
<th>GROUP 1</th>
<th>ALF</th>
<th>ALR</th>
<th>ALS</th>
<th>ARS</th>
<th>BLF</th>
<th>BLR</th>
<th>BLS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BRS</td>
<td>CCA</td>
<td>CCB</td>
<td>CCE</td>
<td>CLA</td>
<td>CLB</td>
<td>CLE</td>
</tr>
<tr>
<td></td>
<td>CLO</td>
<td>CMA</td>
<td>CMB</td>
<td>CME</td>
<td>ELA</td>
<td>ELB</td>
<td>ERA</td>
</tr>
<tr>
<td></td>
<td>ERB</td>
<td>INA</td>
<td>INB</td>
<td>NOP</td>
<td>RAL</td>
<td>RAR</td>
<td>RBL</td>
</tr>
<tr>
<td></td>
<td>RBR</td>
<td>RSS</td>
<td>SEZ</td>
<td>SLA</td>
<td>SLB</td>
<td>SSA</td>
<td>SSB</td>
</tr>
<tr>
<td></td>
<td>STO</td>
<td>SWP</td>
<td>SZA</td>
<td>SZB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 2</td>
<td>SOC</td>
<td>SOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 3</td>
<td>CLF</td>
<td>SFS</td>
<td>SFS</td>
<td>STC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 4</td>
<td>CLC</td>
<td>HLT</td>
<td>LIA</td>
<td>LIB</td>
<td>MIA</td>
<td>MIB</td>
<td>OTA</td>
</tr>
<tr>
<td></td>
<td>OTB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 5</td>
<td>ASL</td>
<td>ASR</td>
<td>LSL</td>
<td>LSR</td>
<td>RRL</td>
<td>RRR</td>
<td></td>
</tr>
<tr>
<td>GROUP 6</td>
<td>ADA</td>
<td>ADB</td>
<td>AND</td>
<td>CPA</td>
<td>CPB</td>
<td>IOR</td>
<td>ISZ</td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>JSB</td>
<td>LDA</td>
<td>LDB</td>
<td>STA</td>
<td>STB</td>
<td>XOR</td>
</tr>
<tr>
<td>GROUP 7</td>
<td>DIV</td>
<td>DLD</td>
<td>DST</td>
<td></td>
<td></td>
<td></td>
<td>MPY</td>
</tr>
</tbody>
</table>
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 \( d_1[,d_2, ..., d_n] \)

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.

\[
\begin{array}{cccc}
15 & 14 & & 0 \\
S & number & & 0
\end{array}
\]

sign

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:

\[ \pm n.n, \pm n., \pm .n, \pm n.E \pm e, \pm n.nE \pm e, \pm n.E \pm e, \pm .nE \pm e \]

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.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>Word 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Fraction (most significant bits)</td>
<td></td>
</tr>
</tbody>
</table>

**Word 2**

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>fraction</td>
<td>exponent</td>
<td>S</td>
</tr>
</tbody>
</table>

sign of exponent

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** \[0_1, 0_2, \ldots, 0_n\]

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.
Page 1

START

LD B = 1111
LENGTH OF DATA
PROGRAM ADDRESS
OF DATA

LD A = PAGE 1
LDA PAGE 1
JSR DWT

USE ONLY IN PRODUCCTARY TEXT ON NTSC CARTRIDGE

PAGE 2

Программа запись интроductory text on disc

165
### PAGE 6 LIST

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LDB = 0312</td>
<td>STR RUEL</td>
</tr>
<tr>
<td>LDA PAGE6</td>
<td>LDB = 016</td>
</tr>
<tr>
<td></td>
<td>JSR DW0IT</td>
</tr>
</tbody>
</table>

### PAGE 7 SEQUENCE

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LDB = 0316</td>
<td>STR RUEL</td>
</tr>
<tr>
<td>LDA PAGE7</td>
<td>LDB = 018</td>
</tr>
<tr>
<td></td>
<td>JSR DW0IT</td>
</tr>
</tbody>
</table>

### PAGE 8 XECUTE

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LDB = 0345</td>
<td>STR RUEL</td>
</tr>
<tr>
<td>LDA PAGE8</td>
<td>LDB = 021</td>
</tr>
<tr>
<td></td>
<td>JSR DW0IT</td>
</tr>
</tbody>
</table>
* PAGE 9 EDIT 1

LDB =8542
STB RUEL
LDA PAGE9
LDB =024
JSR DWRT

* PAGE 10 EDIT 2

LDB =8447
STB RUEL
LDA PGE10
LDB =037
JSR DWRT

* PAGE 11 LAST

LDB =8557
STB RUEL
LDA PGE11
LDB =038
JSR DWRT

* STOP PROGRAM

JSR 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
STA SECTR
JSP EXEG
DEF *+7
DEF RCODE REQUEST CODE 15 FOR DISC WRITE
DEF CONWD
DEF ADDR, I PROGRAM ADDRESS
DEF DUEL LENGTH OF BUFFER
DEF FNAME FILE NAME
DEF SECT, RELATIVE SECTOR
JMP DWRIT, I

ADD I R8S 1
SECTR R8S 1
ROCODE DEC 15
CONWD OCT 102
DUEL R8S 1
FNAME ASC 'JFCAD' BINARY FILE ON USER DISC AREA

* * *
SUP
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 FIDIOUS 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 16, MACHINE CODES AND ADDRESSES.
OCT 106612

ASC 14, THE USE OF INSTRUCTION CODES
ASC 16, LEADS TO A PROGRAMMING LANGUAGE
OCT 106612

ASC 23, ALMOST EQUIVALENT TO MACHINE CODE BUT EASIER TO READ.
OCT 106612

ASC 13, 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 13, 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)
NCP
AN ASSEMBLER NORMALLY BEGINS
ASSEMBLY ONCE THE PROGRAM
HAS BEEN FULLY DEFINED. REFERENCES
TO UNDEFINED INSTRUCTIONS
DATA WILL TERMINATE THE ASSEMBLY
OR HALT FURTHER SYSTEM
ACTIVITY AFTER ASSEMBLY.
T HIS 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.
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
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 17, 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
PAGE 3 OFF *+1

* PAGE 3 INTRODUCTION TO USERS

OCT 116677
OCT 106612
OCT 105212, 105212
OCT 105212, 105212
ASC 15, IF YOU ARE FAMILIAR WITH
ASC 15, THE FEATURES OF THE ASSEMBLED
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
ASC 17, THERE ARE 6 SYSTEM DIRECTIVES
ASC 17, WHICH MAY BE ENTERED ANY
ASC 28, TIME WHILE DEFINING YOUR PROGRAM, EXCEPT DURING AN EDIT.

ASC 15, THEY ALLOW YOU GREATER CONTROL
ASC 14, OVER THE ASSEMBLER AND THE
ASC 12, DESIGN OF YOUR PROGRAM.

ASC 24, THESE DIRECTIVES ARE ALL PRECEDED BY A COLON (:).

ASC 27, DISCONTINUE PROGRAM ENTRY, BEGIN AGAIN

ASC 29, DUMP REGISTER CONTENTS

ASC 31, EDIT THE EXISTING PROGRAM

ASC 34, LIST ALL OR PART OF YOUR PROGRAM

ASC 39, CHANGE THE SEQUENCING

ASC 11, THEN LIST THE PROGRAM

ASC 18, EXECUTE YOUR PROGRAM

ASC 19, TYPE C TO CONTINUE
OCT 116637
OCT 106612,105212,105212
OCT 106612
ASC 12, REGISTERS WILL BE SAVED.
OCT 106612,106612
ASC 16, DUMP
OCT 106612,106612
ASC 15, WILL DISPLAY THE REGISTERS AS
OCT 106612
ASC 13, OCTAL AND DECIMAL VALUES.
OCT 106612
ASC 18, INSTRUCTIONS WILL ALSO BE PRESENTED
OCT 106612
ASC 12, TO DISPLAY DATA ADDRESS
OCT 106612
ASC 5, CONTENTS.
OCT 106612,106612
ASC 17, AS AN ALTERNATIVE TO USING OUTPUT
OCT 106612
ASC 12, INSTRUCTIONS WITHIN YOUR
OCT 106612
ASC 16, PROGRAM, RESULTS CAN BE
OCT 106612
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
PAGE 6 DEF *+1

* LIST

OCT 116637, 106612
OCT 106212, 106212
ASC 19.
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 37.
OCT 106612
ASC 28.
OCT 106612
ASC 21.
TO BE LISTED. IF N IS ABSENT THEN ALL STATE-
MENTS FROM M ON ARE LISTED. IF NEITHER APPEAR
OCT 106612, 106612
ASC 27, THEN THE WHOLE PROGRAM IS LISTED.
OCT 106612, 106612
ASC 27, BUT IF N IS LESS THAN M LISTING IS SUPPRESSED.
OCT 106612, 106212
ASC 9, TYPE C TO CONTINUE
NOR
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 15, IS VERY SIMILAR TO THE SEQUENCE
ASC 14, OPTION PRESENTED EARLIER FOR
OCT 106612
ASC 27, M AND N ARE TWO POSITIVE INTEGERS SUCH THAT
OCT 106612,106612
ASC 14, M BECOMES THE FIRST STATEMENT NUMBER
OCT 106612
ASC 15, N IS THE INCREMENT
ASC 14, FOR SUCCESSIVE STATEMENTS.
OCT 106612,106612
ASC 27, 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 1999 AND
OCT 106612
ASC 17, N MUST NOT EXCEED 25.
OCT 106612,105212
ASC 9, TYPE 'C TO CONTINUE
NOP
PAGE 8  DEF *+1
* EXECUTE
*
OCT 116627, 105212, 105212
ASC 17, EXECUTE
OCT 106612, 106612
ASC 28, WILL INITIATE THE EXECUTION OF YOUR PROGRAM. INCOMPLETE
OCT 106612
ASC 15, PROGRAMS MAY ALSO BE EXECUTED
OCT 106612
ASC 15, BUT EXECUTION WILL IMMEDIATELY
OCT 106612
ASC 14, HALT, WITH A WARNING MESSAGE
OCT 106612
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
OCT 106612
ASC 15, OF THE A, R, E AND O REGISTERS
OCT 106612
ASC 7, WILL BE SAVED.
OCT 106612, 105212
ASC 24, TYPE C TO CONTINUE
NOP
<table>
<thead>
<tr>
<th>PAGE</th>
<th>REF</th>
<th>OCT</th>
<th>ASC</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td>106612,106612</td>
<td>106612</td>
</tr>
<tr>
<td>9</td>
<td>\textit{WILL ALLOW YOU TO}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>\textit{DELETE ANY NUMBER OF STATEMENTS IN YOUR PROGRAM}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>\textit{INSERT BETWEEN SUCCESSIVE STATEMENTS}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>\textit{REPLACE ANY STATEMENT.}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>\textit{ALL EDIT INSTRUCTIONS BEGIN WITH A SLASH (/).}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>\textit{THE FOLLOWING OPERATION CAUSES STATEMENTS M THROUGH}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14,N</td>
<td>\textit{INCLUSIVE, TO BE DELETED}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
OCT 106612
ASC 13, INITIATES THE PRINTING OF
OCT 106612
ASC 27, ALL STATEMENTS INVOLVED IN THE EDIT.
OCT 106612
ASC 26.
OCT 106612
ASC 27.
OCT 106612,106612
ASC 24, TYPE IN YES TO CONTINUE THE EDIT
NOP
OCT 106612
ASC 27.
OCT 106612
ASC 27.
OCT 106612,106612
ASC 24, TYPE C TO CONTINUE
NOP
PGE10 DEF *+1

*  EDIT 2

OCT 116612,106612
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
OCT 106612,106612
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 105612
ASC 12, REPLACED BY A SINGLE STATEMENT
OCT 105612, 105612
ASC 12, /REPLACE,M(.V)
OCT 105612, 105612
ASC 17, A MACHINE CODE INSTRUCTION CANNOT
ASC 12, BE REPLACED BY DATA NOR
OCT 105612
ASC 15, CAN A DATA STATEMENT BE REPLACED
ASC 13, BY A MACHINE INSTRUCTION.
OCT 105612, 105612
ASC 7, /END
OCT 105612, 105612
ASC 15, THE END INSTRUCTION TERMINATES
ASC 14, THE CURRENT EDIT OPERATION.
OCT 105612, 105312
ASC 24, TYPE C TO CONTINUE
NOP
PGE11 DEF *+1

* LAST PAGE

OCT 116677
OCT 106512
ASC 28, NOTE THAT THIS IS A SMALL ASSEMBLER NOT CAPABLE OF
OCT 106512
ASC 27, HANDLING LARGE PROGRAMS. PROGRAM AREA OVERFLOW WILL
OCT 106512
ASC 17, TERMINATE ALL ASSEMBLY, PAY CLOSE
ASC 12, ATTENTION FOR OVERFLOW
OCT 106512
ASC 9, WARNING MESSAGES.
OCT 106612, 106512
ASC 27, ONE IMPORTANT PROGRAMMING CONSIDERATION INVOLVES
OCT 106512
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, MOST IMPORTANTLY, THE DEF PSEUDO OP SHOULD PRECEDE
OCT 106612
ASC 15, ALL DATA WHICH MAY BE INVOLVED
OCT 106612
ASC 14, IN ANY DATA EDIT OPERATIONS
OCT 106612
ASC 28, OP 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
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)
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
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

---

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
:XEXECUTE     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
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

\[ \begin{align*}
\text{Bit 15} = 1 & \quad \text{Give STC to I/O channel at end of each DMA cycle (except last cycle if input operation)} \\
& \quad = 0 \quad \text{No STC} \\
\text{Bit 13} = 1 & \quad \text{Give CLC to I/O channel at end of block transfer} \\
& \quad = 0 \quad \text{No CLC}
\end{align*} \]

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 starting 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).

<table>
<thead>
<tr>
<th>INSTRUCTIONS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>STC 16,C</td>
<td>Start device</td>
</tr>
<tr>
<td>SFS 16</td>
<td>Is input ready</td>
</tr>
<tr>
<td>JMP *-1</td>
<td>No, repeat previous instruction</td>
</tr>
<tr>
<td>LIB 16</td>
<td>Yes, load input into (B)</td>
</tr>
</tbody>
</table>

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).

<table>
<thead>
<tr>
<th>INSTRUCTIONS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTB 16</td>
<td>Output (B) to buffer</td>
</tr>
<tr>
<td>STC 16,C</td>
<td>Start device</td>
</tr>
<tr>
<td>SFS 16</td>
<td>Has device accepted data</td>
</tr>
<tr>
<td>JMP *-1</td>
<td>No, repeat previous instruction</td>
</tr>
<tr>
<td>NOP</td>
<td>Yes, proceed</td>
</tr>
</tbody>
</table>
APPENDIX E

DUMP AND LIST OUTPUT
LIST PROGRAM

0000010 *
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*

@:EXECUTE 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
SAMPLE PROGRAM FOR LIST AND DUMP OUTPUT

CLA CLEAR A REGISTER
CCB CLEAR AND COMPLEMENT B REGISTER
STD SET OVERFLOW REGISTER
LOA ALPHA+1 LOAD A AND B REGISTERS
LOB BETA

ALPHA DEC 11,12,13 DECIMAL CONSTANTS
BETA OCT 11,12,13 OCTAL CONSTANTS

*LIST ENDS*
@:X(ecute)
@:D(ump)

A REGISTER
OCTAL  000014
DECIMAL  000012

B REGISTER
OCTAL  000011
DECIMAL  000009

E REGISTER

O REGISTER

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
SAMPLE PROGRAM FOR LIST AND DUMP OUTPUT

CLA CLEAR A REGISTER
CCB CLEAR AND COMPLEMENT B REGISTER
STO SET OVERFLOW REGISTER

*LIST ENDS*
205

@: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 thorough 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

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>A REGISTER</td>
</tr>
<tr>
<td>00001</td>
<td>B REGISTER</td>
</tr>
<tr>
<td>00002</td>
<td>EXIST SEQUENCE TO FORWARD REFERENCE WARNING IF A AND B CONTENTS ARE USED AS EXECUTABLE INSTRUCTIONS</td>
</tr>
<tr>
<td>00004</td>
<td>POWER FAIL INTERRUPT HALT</td>
</tr>
<tr>
<td>00005</td>
<td>MEMORY PROTECT/PARITY ERROR HALT</td>
</tr>
<tr>
<td>00006</td>
<td>DIRECT MEMORY ACCESS CHANNEL</td>
</tr>
<tr>
<td>00011</td>
<td>DISC DATA CHANNEL</td>
</tr>
<tr>
<td>00012</td>
<td>DISC CONTROL CHANNEL</td>
</tr>
<tr>
<td>00101</td>
<td>JUMP TO INITIALIZATION</td>
</tr>
<tr>
<td>00103</td>
<td>BASE PAGE LINKAGE OF SYSTEM SUBROUTINES</td>
</tr>
<tr>
<td>00172</td>
<td>ASSEMBLER TABLE ADDRESSES</td>
</tr>
</tbody>
</table>

### CONSTANTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00211</td>
<td>Decimal constants</td>
</tr>
<tr>
<td>00313</td>
<td>Octal constants</td>
</tr>
<tr>
<td>00343</td>
<td>Alphabetic constants</td>
</tr>
</tbody>
</table>

### VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00365</td>
<td>System variables</td>
</tr>
<tr>
<td>00416</td>
<td>Temporary variables</td>
</tr>
<tr>
<td>00427</td>
<td>Edit variables</td>
</tr>
</tbody>
</table>

### CONSTANTS AND VARIABLES FOR DISC INPUT DRIVER

### CHARACTER CONSTANTS

### BUFFERS

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00532</td>
<td>Input buffer</td>
</tr>
<tr>
<td>00576</td>
<td>Auxiliary input buffer</td>
</tr>
<tr>
<td>00642</td>
<td>Data store buffer</td>
</tr>
</tbody>
</table>

### OCTAL CONSTANTS

### INTERRUPT HALTS
209

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 Restore 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 edited 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 JMPEl 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

ADDRESS

02000 SYSTEM CONTROLLER

INPUT/OUTPUT PACKAGE

02041 DATIN Request input
02103 TTY.I Perform input operation
02122 TTY.P Perform output operation
02165 I.OFF Turn off interrupt mode
02172 I.ON Turn on interrupt mode
02202 PROCIS 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

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

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
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
06533 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

ERROR MESSAGES FOR NUMBER ROUTINES

LEXICAL AND DUMP SUBROUTINES

07155 OPRREC Read in operand
07435 LABRD Read a symbol
07501 LETPR Check character to be alphabetic or period
07516 DATRGR 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

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

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

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
<table>
<thead>
<tr>
<th>PAGE 0</th>
<th>CALLING PROGRAM</th>
<th>PROGRAM CALLED</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR MESSAGE</td>
<td>THROUGHOUT THE PROGRAM</td>
<td>I/O PACKAGE (1)</td>
</tr>
<tr>
<td>PROCESSOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERRUPT SERVICE</td>
<td>DISC INPUT</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINES</td>
<td>DRIVER (0)</td>
<td></td>
</tr>
<tr>
<td>INITIALIZE</td>
<td>INITIALIZE</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINE</td>
<td>PROGRAM (6)</td>
<td></td>
</tr>
<tr>
<td>LEXICAL SCAN</td>
<td>LEXICAL SCAN (2)</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINES</td>
<td>SYSTEM DIRECTIVE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONTROLLER (1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDIT CONTROLLER (5)</td>
<td></td>
</tr>
<tr>
<td>ASSEMBLY</td>
<td>STATEMENT ASSEMBLY (4)</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXECUTION</td>
<td>XECUTE DIRECTIVE (1)</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDIT</td>
<td>EDIT SUBSYSTEMS (5)</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINES</td>
<td>EDIT SUBROUTINES (5)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDIT DIRECTIVE (1)</td>
<td></td>
</tr>
<tr>
<td>DISC INPUT</td>
<td>INITIALIZATION (6)</td>
<td></td>
</tr>
<tr>
<td>DRIVER</td>
<td>INTERRUPT SERVICE SUBROUTINES (0)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAGE 1</th>
<th>CALLING PROGRAM</th>
<th>PROGRAM CALLED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM CONTROLLER</td>
<td>I/O PACKAGE (1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LEXICAL ROUTINES (0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATEMENT ASSEMBLY (4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATEMENT STORAGE (1)</td>
<td></td>
</tr>
<tr>
<td>I/O PACKAGE</td>
<td>THROUGHOUT THE PROGRAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATEMENT STORAGE</td>
<td>SYSTEM CONTROLLER (1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM DIRECTIVE CONTROLLER (SDC)</td>
<td>SYSTEM CONTROLLER (1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LEXICAL ROUTINE (0)</td>
<td></td>
</tr>
<tr>
<td>PAGE 1</td>
<td>CALLING PROGRAM</td>
<td>PROGRAM CALLED</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>DUMP</td>
<td>SDC (1)</td>
<td>DUMP SUBROUTINES (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LEXICAL AND DUMP SUBROUTINES (2,3)</td>
</tr>
<tr>
<td>EDIT</td>
<td>SDC (1)</td>
<td>EDIT SUBROUTINES (0)</td>
</tr>
<tr>
<td>LIST</td>
<td>SDC (1)</td>
<td>LIST SUBROUTINE (1)</td>
</tr>
<tr>
<td></td>
<td>SEQUENCE DIRECTIVE</td>
<td>NUMBER MANIPULATION SUBROUTINES (3)</td>
</tr>
<tr>
<td></td>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDIT CONTROLLER (5)</td>
<td></td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>SDC (1)</td>
<td>STATEMENT NUMBER SUBROUTINE (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LIST DIRECTIVE (1)</td>
</tr>
<tr>
<td>XECUTE</td>
<td>SDC (1)</td>
<td>XECUTE SUBROUTINES (1,3)</td>
</tr>
<tr>
<td></td>
<td>LEXICAL SCAN (2)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAGE 2</th>
<th>CALLING PROGRAM</th>
<th>PROGRAM CALLED</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN LEXICAL</td>
<td>SYSTEM CONTROLLER</td>
<td>LEXICAL ROUTINES (0,2,3)</td>
</tr>
<tr>
<td>SCAN SUBROUTINE</td>
<td>(1)</td>
<td>NUMBER MANIPULATION ROUTINES (3)</td>
</tr>
<tr>
<td></td>
<td>EDIT SUBROUTINES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(4,5)</td>
<td></td>
</tr>
<tr>
<td>LEXICAL SCAN</td>
<td>MAIN LEXICAL SCAN</td>
<td>LEXICAL SUBROUTINES (0)</td>
</tr>
<tr>
<td>SUBROUTINES</td>
<td>SUBROUTINE (2)</td>
<td>NUMBER MANIPULATION ROUTINES (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAGE 3</th>
<th>CALLING PROGRAM</th>
<th>PROGRAM CALLED</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER MANIPULATION ROUTINES</td>
<td>LEXICAL SCAN SUBROUTINES (2,3)</td>
<td>LEXICAL SUBROUTINES (0)</td>
</tr>
<tr>
<td></td>
<td>EDIT CONTROLLER (5)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SDC (1)</td>
<td></td>
</tr>
<tr>
<td>LEXICAL AND DUMP SUBROUTINES</td>
<td>LEXICAL SCAN (2)</td>
<td>LEXICAL SCAN (0)</td>
</tr>
<tr>
<td></td>
<td>DUMP DIRECTIVE (1)</td>
<td>NUMBER MANIPULATION ROUTINES (3)</td>
</tr>
<tr>
<td>EXECUTION SUBROUTINE</td>
<td>XECUTE DIRECTIVE (1)</td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>Calling Program</td>
<td>Program Called</td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>SEQUENCE SUBROUTINE (STATEMENT NUMBER INPUT)</td>
<td>SEQUENCE DIRECTIVE (1) NUMBER MANIPULATION ROUTINES (3)</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>INSTRUCTION ASSEMBLY</td>
<td>SYSTEM CONTROLLER (1) ASSEMBLY SUBROUTINES (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDIT SUBSYSTEMS (5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDIT SUBROUTINES (4,5)</td>
</tr>
<tr>
<td></td>
<td>EDIT SUBROUTINES</td>
<td>EDIT CONTROLLER (5) LEXICAL SCAN (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDIT SUBSYSTEMS (5) ASSEMBLY SUBROUTINES (4)</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td>EDIT CONTROLLER</td>
<td>SYSTEM CONTROLLER (1) EDIT SUBSYSTEMS (5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDIT SUBROUTINES (4,5) LEXICAL SUBROUTINES (0)</td>
</tr>
<tr>
<td></td>
<td>EDIT SUBSYSTEMS</td>
<td>EDIT CONTROLLER (5) EDIT SUBROUTINES (0,4,5)</td>
</tr>
<tr>
<td><strong>6</strong></td>
<td>INITIALIZATION</td>
<td>SYSTEM CONTROLLER (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INITIALIZATION SUBROUTINE (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISC INPUT DRIVER (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STATEMENT NUMBER INPUT (3)</td>
</tr>
</tbody>
</table>
APPENDIX G

SOURCE PROGRAM LISTING
**ASMB,AL**

**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 LENGTH IN THE SECTION ON USER PROGRAM RESTRICTIONS).

TWO TRACKS ON A CARTRIDGE DISC HOLD READ ONLY DATA FOR DISPLAY AS INTRODUCTORY 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 4).

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 ACCESSIBLE 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)

XUSR  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 ASM3,A,L WERE TO BE THE ASSEMBLER CONTROL STATEMENT.

* OTHER FEATURES LIKE BINARY OUTPUT OR A CROSS 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 WILL 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:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A to (A)</td>
<td>ADD TO (A)</td>
</tr>
<tr>
<td>ADD B to (B)</td>
<td>ADD TO (B)</td>
</tr>
<tr>
<td>Rotate (A) left 4</td>
<td>ROTATE (A) LEFT 4</td>
</tr>
<tr>
<td>Shift (A) left 1, clear sign</td>
<td>SHIFT (A) LEFT 1, CLEAR SIGN</td>
</tr>
<tr>
<td>Shift (A) left 1</td>
<td>SHIFT (A) LEFT 1</td>
</tr>
<tr>
<td>AND (A) to (A)</td>
<td>AND TO (A)</td>
</tr>
<tr>
<td>Rotate (A) right 1, carry sign</td>
<td>ROTATE (A) RIGHT 1, CARRY SIGN</td>
</tr>
<tr>
<td>Arithmetic long shift left</td>
<td>ARITHMETIC LONG SHIFT LEFT</td>
</tr>
<tr>
<td>Arithmetic long shift right</td>
<td>ARITHMETIC LONG SHIFT RIGHT</td>
</tr>
<tr>
<td>Rotate (B) left 4</td>
<td>ROTATE (B) LEFT 4</td>
</tr>
<tr>
<td>Shift (B) left 1, clear sign</td>
<td>SHIFT (B) LEFT 1, CLEAR SIGN</td>
</tr>
<tr>
<td>Shift (B) left 1</td>
<td>SHIFT (B) LEFT 1</td>
</tr>
<tr>
<td>Shift (B) right 1, carry sign</td>
<td>SHIFT (B) RIGHT 1, CARRY SIGN</td>
</tr>
<tr>
<td>Clear and complement (A)</td>
<td>CLEAR AND COMPLEMENT (A)</td>
</tr>
<tr>
<td>Clear and complement (B)</td>
<td>CLEAR AND COMPLEMENT (B)</td>
</tr>
<tr>
<td>Clear and complement (E) set (E) = 1</td>
<td>CLEAR AND COMPLEMENT (E) SET (E) = 1</td>
</tr>
<tr>
<td>Clear (A)</td>
<td>CLEAR (A)</td>
</tr>
<tr>
<td>Clear (B)</td>
<td>CLEAR (B)</td>
</tr>
<tr>
<td>Clear I/O control bit</td>
<td>CLEAR I/O CONTROL BIT</td>
</tr>
<tr>
<td>Clear (E)</td>
<td>CLEAR (E)</td>
</tr>
<tr>
<td>Clear I/O flag</td>
<td>CLEAR I/O FLAG</td>
</tr>
<tr>
<td>Clear overflow bit</td>
<td>CLEAR OVERFLOW BIT</td>
</tr>
<tr>
<td>Complement (A)</td>
<td>COMPLEMENT (A)</td>
</tr>
<tr>
<td>Complement (B)</td>
<td>COMPLEMENT (B)</td>
</tr>
</tbody>
</table>
* 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
* LOA    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
* SFZ  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 (CLEAR)
* 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
* SNP  SWITCH (A) AND (B)
* SZA  SKIP IF (A) = 0
* SZA  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**  0,0,0,0,0,0,0,0,0,0,0,0,0,0
- **OCT**  0,0,0,0,0,0,0

FIRST 100 (OCTAL) LOCATIONS AVAILABLE TO USER

```
ORG 1018
```

* 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.SIP 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
OTED DEF OTEDD DELETE DATA DURING EDIT
DIDI DEF DTEDI INSERT DATA DURING EDIT
OTFL DEF DTFL Check DATA AREA OVERFLOW
DTRG DEF DTTRG CHECK ADDRESS RANGE IN BUFFER
OTST DEF OTSST STORE DATA
GTMN 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 LTPPR LETTER OR PERIOD CHECK
NMBR DEF NUMBR INPUT DECIMAL OR OCTAL INTEGER
NWLN DEF CRFD OUTPUT CR-LF
NWLS DEF NWLNS OUTPUT MULTIPLE CR-LF
OCIN DEF OCTIN OCTAL INTEGER INPUT
OPRC DEF OPREC OPERAND RECOGNITION
SCNCOD DEF CDSCN  SCAN ASSEMBLED CODE FOR FWD REF
SFSP DEF STFSPEC LENGTH AND ADDR OF DELETE IN FR SP
SLBL DEF STLBL  STORE LABEL 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 ASMED  EDIT VARIABLE ADDRESSES
SCBE DEF SCBEO

ABSSR DEF LXRIN  RETURN TO ABSS/BSB PROGRAM
CNTRL DEF CHAND  LINK TO SYSTEM CONTROLLER
DMPRT DEF DMP2  RETURN TO DUMP AFTER USER INSTR
EDTR DEF EDIT  LINK TO EDIT SUPERVISOR
EDLXEX DEF EDXRT  LINK TO EDIT FOR SOURCE INPUT
GRTER DEF GRT8  ADDR OF FIRST I/O INSTR
I.0 DEF I.1  ADDR OF FIRST I/O INSTR
INT1 DEF I.OFF+2
INT2 DEF I.1N+2
INT3 DEF TP.3-1
INT4 DEF TTY.P+2
LXANL DEF LXSCN  LINK TO LEXICAL ROUTINE
MIRTL DEF MIRTE  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 118  DISC DATA CHANNEL
CC  EQU 128  DISC CONTROL CHANNEL
TTY  EQU 179  CHANNEL NUMBER I/O DEVICE
XOPOCD  DEF 152008  OPCODE TABLE
XSTBL  DEF 156028  SYMBOL TABLE
XSSST  DEF 171608  AUXILIARY SYMBOL TABLE
XPLC  DEF 176348  UNDEFINED PLC REFERENCE STORE
XSCB  DEF 200008  SOURCE CODE BLOC
XFRSP  DEF 257009  TABLE OF FREE SPACE
XUSRP  DEF 263418  USER PROGRAM
XDATA  DEF 267018  PROGRAM DATA
*  *  *

YSTBL  DEF 171578
YSST  DEF 176338
YPLC  DEF 177778
YSGB  DEF 256778
YFRSP  DEF 257778
YUSRP  DEF 266778
*  *

PROG  DEF 0263408  ADDR FOR SUBR JUMP TO USER PRGRM
## Decimal Constants

<table>
<thead>
<tr>
<th></th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DEC 0</td>
</tr>
<tr>
<td>1</td>
<td>DEC 1</td>
</tr>
<tr>
<td>2</td>
<td>DEC 2</td>
</tr>
<tr>
<td>3</td>
<td>DEC 3</td>
</tr>
<tr>
<td>4</td>
<td>DEC 4</td>
</tr>
<tr>
<td>5</td>
<td>DEC 5</td>
</tr>
<tr>
<td>6</td>
<td>DEC 6</td>
</tr>
<tr>
<td>7</td>
<td>DEC 7</td>
</tr>
<tr>
<td>8</td>
<td>DEC 8</td>
</tr>
<tr>
<td>9</td>
<td>DEC 9</td>
</tr>
<tr>
<td>10</td>
<td>DEC 10</td>
</tr>
<tr>
<td>11</td>
<td>DEC 11</td>
</tr>
<tr>
<td>12</td>
<td>DEC 12</td>
</tr>
<tr>
<td>13</td>
<td>DEC 13</td>
</tr>
<tr>
<td>14</td>
<td>DEC 14</td>
</tr>
<tr>
<td>15</td>
<td>DEC 15</td>
</tr>
<tr>
<td>16</td>
<td>DEC 16</td>
</tr>
<tr>
<td>18</td>
<td>DEC 18</td>
</tr>
<tr>
<td>20</td>
<td>DEC 20</td>
</tr>
<tr>
<td>22</td>
<td>DEC 22</td>
</tr>
<tr>
<td>24</td>
<td>DEC 24</td>
</tr>
<tr>
<td>26</td>
<td>DEC 26</td>
</tr>
<tr>
<td>28</td>
<td>DEC 28</td>
</tr>
<tr>
<td>30</td>
<td>DEC 30</td>
</tr>
<tr>
<td>32</td>
<td>DEC 32</td>
</tr>
<tr>
<td>34</td>
<td>DEC 34</td>
</tr>
<tr>
<td>38</td>
<td>DEC 38</td>
</tr>
<tr>
<td>40</td>
<td>DEC 40</td>
</tr>
<tr>
<td>DEC</td>
<td>ASCII ZERO</td>
</tr>
<tr>
<td>-----</td>
<td>------------</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>M1</td>
<td>-1</td>
</tr>
<tr>
<td>M2</td>
<td>-2</td>
</tr>
<tr>
<td>M3</td>
<td>-3</td>
</tr>
<tr>
<td>M4</td>
<td>-4</td>
</tr>
<tr>
<td>M5</td>
<td>-5</td>
</tr>
<tr>
<td>M6</td>
<td>-6</td>
</tr>
<tr>
<td>M7</td>
<td>-7</td>
</tr>
<tr>
<td>M8</td>
<td>-8</td>
</tr>
<tr>
<td>M9</td>
<td>-9</td>
</tr>
<tr>
<td>M10</td>
<td>-10</td>
</tr>
<tr>
<td>M12</td>
<td>-12</td>
</tr>
<tr>
<td>M16</td>
<td>-16</td>
</tr>
<tr>
<td>M19</td>
<td>-19</td>
</tr>
<tr>
<td>M20</td>
<td>-20</td>
</tr>
<tr>
<td>M25</td>
<td>-25</td>
</tr>
<tr>
<td>M26</td>
<td>-26</td>
</tr>
<tr>
<td>M28</td>
<td>-28</td>
</tr>
<tr>
<td>M29</td>
<td>-29</td>
</tr>
<tr>
<td>M32</td>
<td>-32</td>
</tr>
<tr>
<td>M37</td>
<td>-37</td>
</tr>
<tr>
<td>M46</td>
<td>-46</td>
</tr>
<tr>
<td>M48</td>
<td>-48</td>
</tr>
<tr>
<td>M52</td>
<td>-52</td>
</tr>
<tr>
<td>M55</td>
<td>-55</td>
</tr>
<tr>
<td>M62</td>
<td>-62</td>
</tr>
<tr>
<td>M75</td>
<td>-75</td>
</tr>
<tr>
<td>M100</td>
<td>-100</td>
</tr>
<tr>
<td>M125</td>
<td>-125</td>
</tr>
<tr>
<td>M129</td>
<td>-129</td>
</tr>
<tr>
<td>M256</td>
<td>-256</td>
</tr>
<tr>
<td>M750</td>
<td>-750</td>
</tr>
<tr>
<td>M1001</td>
<td>-1001</td>
</tr>
</tbody>
</table>
## OCTAL CONSTANTS

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>8137</td>
<td>137</td>
</tr>
<tr>
<td>8177</td>
<td>177</td>
</tr>
<tr>
<td>8200</td>
<td>200</td>
</tr>
<tr>
<td>8337</td>
<td>337</td>
</tr>
<tr>
<td>8376</td>
<td>376</td>
</tr>
<tr>
<td>8377</td>
<td>377</td>
</tr>
<tr>
<td>8400</td>
<td>400</td>
</tr>
<tr>
<td>8700</td>
<td>700</td>
</tr>
<tr>
<td>8701</td>
<td>701</td>
</tr>
<tr>
<td>8100</td>
<td>1000</td>
</tr>
<tr>
<td>8120</td>
<td>1200</td>
</tr>
<tr>
<td>81273</td>
<td>1273</td>
</tr>
<tr>
<td>81600</td>
<td>1600</td>
</tr>
<tr>
<td>81777</td>
<td>1777</td>
</tr>
<tr>
<td>82000</td>
<td>2000</td>
</tr>
<tr>
<td>82400</td>
<td>2400</td>
</tr>
<tr>
<td>80700</td>
<td>70000</td>
</tr>
<tr>
<td>81760</td>
<td>176000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>070</td>
<td>-70</td>
</tr>
<tr>
<td>072</td>
<td>-72</td>
</tr>
<tr>
<td>0100</td>
<td>-100</td>
</tr>
<tr>
<td>0133</td>
<td>-133</td>
</tr>
<tr>
<td>0337</td>
<td>-337</td>
</tr>
<tr>
<td>0340</td>
<td>-340</td>
</tr>
<tr>
<td>0700</td>
<td>-700</td>
</tr>
<tr>
<td>0701</td>
<td>-701</td>
</tr>
<tr>
<td>Letter</td>
<td>Octal Value</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>A</td>
<td>OCT 101</td>
</tr>
<tr>
<td>B</td>
<td>OCT 102</td>
</tr>
<tr>
<td>C</td>
<td>OCT 103</td>
</tr>
<tr>
<td>D</td>
<td>OCT 104</td>
</tr>
<tr>
<td>E</td>
<td>OCT 105</td>
</tr>
<tr>
<td>F</td>
<td>OCT 110</td>
</tr>
<tr>
<td>G</td>
<td>OCT 111</td>
</tr>
<tr>
<td>H</td>
<td>OCT 114</td>
</tr>
<tr>
<td>I</td>
<td>OCT 116</td>
</tr>
<tr>
<td>J</td>
<td>OCT 117</td>
</tr>
<tr>
<td>K</td>
<td>OCT 122</td>
</tr>
<tr>
<td>L</td>
<td>OCT 123</td>
</tr>
<tr>
<td>M</td>
<td>OCT 124</td>
</tr>
<tr>
<td>N</td>
<td>OCT 126</td>
</tr>
<tr>
<td>O</td>
<td>OCT 130</td>
</tr>
<tr>
<td>P</td>
<td>OCT 131</td>
</tr>
</tbody>
</table>
### VARIABLES

<table>
<thead>
<tr>
<th>Variable</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSSF</td>
<td>ABS/BSS PSEUDO OP FLAG</td>
</tr>
<tr>
<td>BADDR</td>
<td>CURRENT BUFFER ADDRESS</td>
</tr>
<tr>
<td>CCNT</td>
<td>CHARACTER COUNT</td>
</tr>
<tr>
<td>COUNT</td>
<td>HOLDS RECORD LENGTH</td>
</tr>
<tr>
<td>CUSTN</td>
<td>CURRENT USER STATEMENT NUMBER</td>
</tr>
<tr>
<td>DMPFG</td>
<td>DUMP FLAG</td>
</tr>
<tr>
<td>EDTFG</td>
<td>EDIT FLAG</td>
</tr>
<tr>
<td>FIRST</td>
<td>FIRST ENTRY IN SOURCE CODE BLOCK</td>
</tr>
<tr>
<td>FSTMT</td>
<td>FIRST STATEMENT NUMBER</td>
</tr>
<tr>
<td>GRTFG</td>
<td>FLAG SET DURING INTRODUCTORY TEXT</td>
</tr>
<tr>
<td>LBCNT</td>
<td>LABEL COUNTER IN SYMBOL TABLE</td>
</tr>
<tr>
<td>NEXT</td>
<td>ADDR OF NEXT ENTRY IN SOURCE CODE</td>
</tr>
<tr>
<td>PREV</td>
<td>PREVIOUS ENTRY IN SOURCE CODE BLOCK</td>
</tr>
<tr>
<td>SAVA</td>
<td>STORAGE FOR (A)</td>
</tr>
<tr>
<td>SAVB</td>
<td>STORAGE FOR (B)</td>
</tr>
<tr>
<td>SAVEO</td>
<td>STORAGE FOR (E) AND (O)</td>
</tr>
<tr>
<td>SEQFG</td>
<td>SEQUENCE DIRECTIVE INDICATOR</td>
</tr>
<tr>
<td>SRCNT</td>
<td>BUFFER LENGTH FOR CODE STORAGE</td>
</tr>
<tr>
<td>STING</td>
<td>STATEMENT NUMBER INCREMENT</td>
</tr>
<tr>
<td>TEMPI</td>
<td>INTY INTERRUPT STORE</td>
</tr>
<tr>
<td>YDATA</td>
<td>UPPER BOUND OF USER DATA AREA</td>
</tr>
<tr>
<td>ZDATA</td>
<td>NEXT LOCATION IN DATA AREA</td>
</tr>
<tr>
<td>ZFRSP</td>
<td>NEXT OPENING IN FREE SPACE</td>
</tr>
<tr>
<td>ZPLC</td>
<td>NEXT LOCATION FOR UNDEF PLC REFERENCE</td>
</tr>
<tr>
<td>ZUSRPI</td>
<td>NEXT LOCATION IN USER PROGRAM</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ADDR1</td>
<td>Address in source code block</td>
</tr>
<tr>
<td>ADDR2</td>
<td>Temporary storage variable</td>
</tr>
<tr>
<td>ADDR3</td>
<td></td>
</tr>
<tr>
<td>ASMBY</td>
<td>Skeleton of assembly instruction</td>
</tr>
<tr>
<td>ASMFG</td>
<td>Assembly flag</td>
</tr>
<tr>
<td>DATPT</td>
<td>Data buffer pointer</td>
</tr>
<tr>
<td>DPFLG</td>
<td></td>
</tr>
<tr>
<td>EDINT</td>
<td>Edit instruction type</td>
</tr>
<tr>
<td>EFLG</td>
<td>Exponent e flag</td>
</tr>
<tr>
<td>EXP</td>
<td></td>
</tr>
<tr>
<td>EXPON</td>
<td></td>
</tr>
<tr>
<td>INSNM</td>
<td>Instruction number</td>
</tr>
<tr>
<td>LBLAD</td>
<td>Label address</td>
</tr>
<tr>
<td>LFLFG</td>
<td>Label flag</td>
</tr>
<tr>
<td>LENTH</td>
<td>Length of assembly</td>
</tr>
<tr>
<td>LMTFG</td>
<td>Control in sym tbl search</td>
</tr>
<tr>
<td>LNTH2</td>
<td>Data counter</td>
</tr>
<tr>
<td>MANT1</td>
<td>Mantissa terms, temporary storage</td>
</tr>
<tr>
<td>MANT2</td>
<td></td>
</tr>
<tr>
<td>NUMFG</td>
<td>Operand number flag</td>
</tr>
<tr>
<td>NUM1</td>
<td>Hold numbers, temporary storage</td>
</tr>
<tr>
<td>NUM2</td>
<td></td>
</tr>
<tr>
<td>OPLBL</td>
<td>Operand label</td>
</tr>
<tr>
<td>OPNUM</td>
<td>Operand integer value</td>
</tr>
<tr>
<td>SIGN</td>
<td>Number sign</td>
</tr>
<tr>
<td>STORE</td>
<td></td>
</tr>
<tr>
<td>TEMP</td>
<td></td>
</tr>
</tbody>
</table>
TEMP1 BSS 1
TEMP2 BSS 1
TEMP3 BSS 1
TEMP4 BSS 1
TEMP5 BSS 1
TEMP6 BSS 1
TEMP7 BSS 1
ZADD BSS 1

TEMPORARY STORAGE

TEMP1
TEMP2
TEMP3
TEMP4
TEMP5
TEMP6
TEMP7
ZADD

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

ADDRESS OF MNEMONIC BUFFER

MNC DEF MNC

ADDRESS OF LABEL BEGINNING STATEMENT

ADDR EQU EXP TEMPORARY USED IN BUFFER STORAGE
ENDF EQU ADDR3
HOLDA EQU MANT1
HOLDB EQU MANT2
IDRCT EQU EFLG INDIRECT BIT
LINK EQU EXPN LINK FOR COMPOUND OPERANDS

LOWER BOUND IN MNEMONIC SEARCH

LWRSO EQU NUM1
MNC EQU MANT1
MORG EQU EXPN
OPADD EQU NUM1
SORCE EQU STORE
STNUM EQU EFLG
UNDEF EQU SIGN
UPRBD EQU NUM2

UPPER BOUND IN MNEMONIC SEARCH
* EDIT VARIABLES

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASME0</td>
<td>BSS 1 Assembly Addresses</td>
</tr>
<tr>
<td>ASME1</td>
<td>BSS 1 Assembly Code Addresses on a</td>
</tr>
<tr>
<td>ASME2</td>
<td>BSS 1 Multiple Delete Operation</td>
</tr>
<tr>
<td>DADR1</td>
<td>BSS 1 Delete Last Line</td>
</tr>
<tr>
<td>DADR2</td>
<td>BSS 1 Stat Num Limit on Multi Insert</td>
</tr>
<tr>
<td>DLTLN</td>
<td>BSS 1 Instruction Number</td>
</tr>
<tr>
<td>EDLMT</td>
<td>BSS 1 Address for Moving Code</td>
</tr>
<tr>
<td>EDNUM</td>
<td>BSS 1 Length of Deleted Code</td>
</tr>
<tr>
<td>ENEXT</td>
<td>BSS 1 Next Free Area in SCB Before Edit</td>
</tr>
<tr>
<td>ENM1</td>
<td>BSS 1 Statement Number</td>
</tr>
<tr>
<td>ENM2</td>
<td>BSS 1 Location Store to Link Edit</td>
</tr>
<tr>
<td>EUSRP</td>
<td>BSS 1 Input Expectation Flag</td>
</tr>
<tr>
<td>EXPEC</td>
<td>BSS 1 Multi Insert in Progress</td>
</tr>
<tr>
<td>MCMIP</td>
<td>BSS 1 Machine Code Multi Insert</td>
</tr>
<tr>
<td>SCBEO</td>
<td>BSS 1 Source Code Block Addresses</td>
</tr>
<tr>
<td>SCBE1</td>
<td>BSS 1 Veto Flag</td>
</tr>
<tr>
<td>SCBE2</td>
<td>BSS 1 Source Code Block Addresses</td>
</tr>
<tr>
<td>VETO</td>
<td>BSS 1 Veto Flag</td>
</tr>
<tr>
<td>AHEAD</td>
<td>EQU ASME0 Look Ahead Pointer in SCB</td>
</tr>
<tr>
<td>BACK</td>
<td>EQU ASME1 Look Back Pointer in SCB</td>
</tr>
<tr>
<td>EDLX</td>
<td>EQU DADR1 Source Input Flag During Edit</td>
</tr>
<tr>
<td>LKPSN</td>
<td>EQU DADR2 Link Position</td>
</tr>
<tr>
<td>LNGTH3</td>
<td>EQU EDNUM Length of Assembly</td>
</tr>
<tr>
<td>POSN</td>
<td>EQU ENEXT Position of Search in SST</td>
</tr>
<tr>
<td>SSTAD</td>
<td>EQU ENM1 SST Address</td>
</tr>
<tr>
<td>SUCAD</td>
<td>EQU VETO Pointer for Listing Purposes</td>
</tr>
<tr>
<td>VALUE</td>
<td>EQU SCBEO Temporary to Hold Number in Operand</td>
</tr>
</tbody>
</table>
* DISC INPUT DRIVER VARIABLES

DREAD OCT 020000 DISC READ COMMAND
SFEXX OCT 030000
TR202 OCT 145000 DISC ADDR OF LAST TRACK
LSTAT DEC 204 LAST TRACK ACCESSED
DSIPT OCT 14340 MEMORY ADDRESS FOR DISC INPUT

DCMND EQU EFLG DISC ADDRESS
DATA EQU EXP READ COMMAND
DSTAT EQU EXPON DISC STATUS
HOMSK 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
PERIOD 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

<table>
<thead>
<tr>
<th>Octal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH1</td>
<td>OCT 177400</td>
</tr>
<tr>
<td>CLR1B</td>
<td>OCT -12500</td>
</tr>
<tr>
<td>CPIB</td>
<td>OCT 102000</td>
</tr>
<tr>
<td>DMACH</td>
<td>OCT 120011</td>
</tr>
<tr>
<td>IMODE</td>
<td>OCT 160000</td>
</tr>
<tr>
<td>LMODE</td>
<td>OCT 120000</td>
</tr>
<tr>
<td>MSIGN</td>
<td>OCT 026400</td>
</tr>
<tr>
<td>JMP</td>
<td>OCT 026000</td>
</tr>
<tr>
<td>MSK4</td>
<td>OCT 77600</td>
</tr>
<tr>
<td>MNEG</td>
<td>OCT 100000</td>
</tr>
<tr>
<td>TENTH</td>
<td>OCT 63145</td>
</tr>
<tr>
<td>YDAT</td>
<td>OCT 27300</td>
</tr>
<tr>
<td>XRTRN</td>
<td>OCT 126340</td>
</tr>
</tbody>
</table>

### INTERRUPT HALTS

<table>
<thead>
<tr>
<th>Octal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLT4</td>
<td>HLT 4, C</td>
</tr>
<tr>
<td>HLT5</td>
<td>HLT 5</td>
</tr>
</tbody>
</table>

### INTERRUPT SERVICE SUBROUTINE CALLS

<table>
<thead>
<tr>
<th>Octal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAI</td>
<td>JSB DMASS</td>
</tr>
<tr>
<td>DCI</td>
<td>JSB DCSS</td>
</tr>
<tr>
<td>CCI</td>
<td>JSB CCSS</td>
</tr>
</tbody>
</table>
CALL TO ERROR MESSAGE OUTPUT

ERCALL 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 NLNLN,I OUTPUT CR-LF
LDA HOLDA RESTORE (A) AND (B)
LDB HOLDB
JSB WRITE,I PRINT MESSAGE
JMP BPLN,I
**BASE PAGE ERROR MESSAGES**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR1</td>
<td>ASC 7, BAD DATA INPUT</td>
</tr>
<tr>
<td>ERR2</td>
<td>ASC 15, STATEMENT NUMBER OUT OF RANGE</td>
</tr>
<tr>
<td>ERR3</td>
<td>ASC 13, OPERAND VALUE OUT OF RANGE</td>
</tr>
<tr>
<td>ERR4</td>
<td>ASC 14, ILLEGAL OPERAND TERMINATION</td>
</tr>
<tr>
<td>ERR5</td>
<td>ASC 15, ILLEGAL CHARACTER BEGINS LABEL</td>
</tr>
<tr>
<td>ERR6</td>
<td>ASC 8, NO OPERAND FOUND</td>
</tr>
<tr>
<td>ERR7</td>
<td>ASC 10, OPERAND IS UNDEFINED</td>
</tr>
<tr>
<td>ERR8</td>
<td>ASC 13, UNDEFINED LABEL IN OPERAND</td>
</tr>
<tr>
<td>ERR9</td>
<td>ASC 7, NO LABEL FOUND</td>
</tr>
</tbody>
</table>

**PRINT MESSAGE ON TABLE OVERFLOW WITH RESTART INSTRUCTIONS**

```
TBL0V 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
     JMP  DCSS,1

**CONTROL CHANNEL INTERRUPT**

CCSS  NOP
     CLC  CC
     JMP  CCSS,1

**CONFIGURE I/O SUBROUTINES**

**ENTER (5) CHANNEL NUMBER OF I/O DEVICE**

CNFIG  NOP
      LDA  D72
      STA  TEMP2
      LDA  I.0
      STA  TEMP1

CNFG1  LDA  TEMP1;I
       STA  TEMP3
       SSA, RSS
       JMP  CNFG2
       AND  00700
       SZA
       JMP  CNFG2
       LDA  TEMP3

**INSTRUCTION IN (A)**

---

**INSTRUCTION IN (B)**

---

**MEMORY REFERENCE**

---

**INSTRUCTION IN (C)**

---

**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 CNFG4,I
*
* REMOVE CHANNEL NUMBER AND REPLACE WITH NEW ONE
*
* CNFG3 NOP
*
AND D100
*
IOR B  ADD IN NEW VALUE
*
JMP CNFG3,I

246
* GET NEXT CHARACTER FROM 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 SAVEEE SAVE (E) REGISTER
LDB BADDR LOAD BUFFER ADDRESS
ISZ BADDR UPDATE FOR NEXT TIME
CLE,ER8 SET CHARACTER FLAG
LOA B1T 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 END LF LINE
RSS 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 ERROR SHIFT (E) INTO (B)
STB ERROR STORE (B)
JMP SAVEE,I

RSTRE NOP
LDB ERROR CLEAR THEN RESTORE (E)
CLE,ELB
JMP RSTRE,I

249
* MOVE N WORDS FROM (A) TO (B)
* ENTER (A) = FWA OF ORIGIN
* (B) = FWA OF DESTINATION

WMOVE
NOP
STA MORG
LDA SORCE
CMA,INA
STA TEMP4
LDA MORG,I
STA B,I
INB
ISZ MORG
ISZ TEMP4
JMP *-5
ADB M1
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
CMB,INB
ADB A
SSB
JMP DATAD,I
LOA A,I
JMP DATAD,I

DATA 250
* MASK ON INDIRECT BIT IF REQUESTED
* ENTER (A) INSTRUCTION OR ADDRESS

**IDRIT**
NOP
LDB IDRCT INDIRECT FLAG
SZB
IOR MNEG MASK ON BIT 15
JMP IDRIT,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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 SAVER,I

* PREPARE ADDRESS POINTERS FOR EDIT OPERATION

**EDITAD**
NOP
LDA ZUSR P NEXT FREE AREA IN PROGRAM
STA EUSR P SAVE FOR EDIT LINK PURPOSES
ADA 2 ADVANCE FOR EDIT ENTRIES
STA ZUSR
JSB SICK,I CHECK FOR PROGRAM AREA OVERFLOW
JMP EDITAD,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**

<table>
<thead>
<tr>
<th>LOA B, I</th>
<th>ADDR OF LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND B177</td>
<td>NUMBER OF WORDS IN SCB ENTRY</td>
</tr>
<tr>
<td>STA CNFG3</td>
<td></td>
</tr>
<tr>
<td>LOA B, I</td>
<td></td>
</tr>
<tr>
<td>ALF, ALF</td>
<td></td>
</tr>
<tr>
<td>AND B177</td>
<td>NUMBER OF CHARACTERS</td>
</tr>
<tr>
<td>CMA</td>
<td>CONTROL VARIABLE USED IN GETTING</td>
</tr>
<tr>
<td>STA CCNT</td>
<td>NEXT CHARACTER FROM BUFFER</td>
</tr>
</tbody>
</table>

**INS**

<table>
<thead>
<tr>
<th>LOA B, I</th>
<th>ASSEM FLAG, ADDR OF ASSEMBLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP PREPR, I</td>
<td></td>
</tr>
</tbody>
</table>
CLEAR UP LINKAGE IN SOURCE CODE BLOCK ON A DELETE OPERATION

DSCB  NOP
      LDA SCBE0
      SZA
      JMP DSCB2
      LDB DLILN
      SZA,RSS
      JMP DSCB1
      CCA
      STA PREV
      LDA NEXT
      STA FIRST
      JMP DSCB,I

DSCB1 LDA SCBE2
      STA FIRST
      CCB
      LDA SCBE2
      INA
      STA A,I
      JMP DSCB,I

DSCB2 LDA DLILN
      SZA,RSS
      JMP DSCB3
      LDB SCBE0
      STB A,I
      JMP DSCB,I

DSCB3 LDA SCBE0
      LDB SCBE2
      STB SCBE0,I
      INB
      STA B,I
      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 5,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 EUSR
JMP CNTRL,I

* SAVE POSITION IN USER PROGRAM AREA FOR POSITIONING
* LINK INSTRUCTIONS AFTER AN EDIT OPERATION

SVPSN NOP SAVE POSITION
LDB ZUSR NEXT LOCATION IN PROGRAM
STA 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
LDB ZUSR
LDA ASHE2
JNA
JSB JMPS
STB ZUSR
JSB STCK, I
JMP JMPAF, I

** **

PLACE JUMPS TO CONNECT MAIN USER PROGRAM WITH
BEGINNING OF EDIT ENTRY

** **

JMPBF NOP
LDB ASHE1
LDA EDSV
JNA
JSB JMPS
JMP JMPBF, I

** **

STORE JUMPS TO LINK EDITED CODE

* ENTER (A) ADDRESS WHERE JUMP RESULTS
* (B) ADDRESS WHERE JUMP ORIGINATES

** **

JMPS NOP
CLE
AND B1777
ADD A JMP
STA B, I
JNA
SEZ, CME, IN3, RSS
JMP *- 5
JMP JMS, 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 DISPLAYED IN (A)

AND (B) 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
STAB 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 228
JSB RSEEK
JMP DISKI TRY AGAIN 10 MORE TIMES
SUBROUTINE DISK0 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

DISK0
NOP
LDB MADDR
CLC 2
OTB 2
STC 2
LDB LENTH
OTB 2
LDA DCMND
JSB SEEK
JMP DSKD1
LDA DOTA
OTA CC
STC CC,C
CLC CC
STC 5,C
STC CC,C
SFS CC
JMP *-1
JSB STAT
DSKD1
ISZ DISK0
JMP DSKD,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 8377 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

** COMPUTE PHYSICAL HEAD/SECTOR FROM LOGICAL SECTOR
** NUMBER AND HEAD MASK

CLB,RSS
INB
ADA M12
SSA,RSS
JMP *+3
ADA ,12 12 SECTORS PER TRACK
BLF,3LF
** ADMSK 

OR 8SFS DC JMP *-1

** OTA DC** 

** STC DC,C** 

** SFS CC** 

** JMP *-1** 

** JSB STAT** 

** 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 STATUS ERROR**

**P+2 SUCCESSFUL STATUS CHECK**

---

<table>
<thead>
<tr>
<th>STAT</th>
<th>NOP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STF 6</td>
</tr>
<tr>
<td></td>
<td>LIB 2</td>
</tr>
<tr>
<td></td>
<td>STC DC,C</td>
</tr>
<tr>
<td></td>
<td>CLA</td>
</tr>
<tr>
<td></td>
<td>CLC CC</td>
</tr>
<tr>
<td></td>
<td>DTA CC</td>
</tr>
<tr>
<td></td>
<td>STC CC,C</td>
</tr>
<tr>
<td></td>
<td>SFS DC</td>
</tr>
<tr>
<td></td>
<td>SFS DC</td>
</tr>
<tr>
<td></td>
<td>SFS DC</td>
</tr>
<tr>
<td></td>
<td>JMP *-1</td>
</tr>
<tr>
<td></td>
<td>CLC CC</td>
</tr>
<tr>
<td></td>
<td>LIA DC</td>
</tr>
<tr>
<td></td>
<td>STA DSTAT</td>
</tr>
<tr>
<td></td>
<td>CLE,SLA,RSS</td>
</tr>
<tr>
<td></td>
<td>JMP STAT1</td>
</tr>
<tr>
<td></td>
<td>RAL,ARS</td>
</tr>
<tr>
<td></td>
<td>SSA,RSS</td>
</tr>
<tr>
<td></td>
<td>RAR,SLA,RAR</td>
</tr>
</tbody>
</table>

---

*DRIVE NUMBER, STATUS CHECK*

*OUTPUT STATUS COMMAND TO COMMAND CHANNEL*

*WAIT UNTIL DATA CHANNEL CLEAR*

*LOAD STATUS FROM DATA CHANNEL*

*DISC STATUS ERROR*

*NO*

*FIRST SEEK*

*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 248
JMP DISK1  BEGIN READ AGAIN
  *
STAT3 JSB NWLN, I
LDA *14
LDB STATR  DISC NOT READ MESSAGE
JSB WRITE, I
HLT 268
JMP DISK1  BEGIN READ CYCLE AGAIN
  *
STATR DEF *+1
  ASC 7, DISC NOT READ
  *

262
ORG 20000B

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 INTRODUCTORY 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.

RETURN 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 (EDITF)
* 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
LOB ABSSF ABS/BSS FLAG
SZA
JMP ABSSR I RETURN TO LEXICAL ROUTINE
JSB CLER I CLEAR LEXICAL POINTERS
LOB DMPFG DUMP FLAG
SZA
JMP DMP2 RETURN TO DUMP ROUTINE
LOB SEQFG SEQUENCE FLAG
SZA
JMP SEQ RETURN TO SEQUENCE ROUTINE
LOB EDLX SOURCE INPUT DURING EDIT
SZA
JMP EDLEX I RETURN TO EDIT INPUT CONTROL
LOB EDTFG EDIT FLAG, EDIT INSTRUCTION
SZA
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 ASMM I PREPARE FOR STORAGE
LOA 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 LDDEF DEFINE LABEL IF PRESENT
LOB MIIIP MULTIPLE INSERT
SZA RSS
JMP CMAND
JMP MIRTI I RETURN TO MULTIPLE INSERT
* * SUBROUTINE TO REQUEST INPUT AND CALL INPUT ROUTINE

Datin

NOP
JSB CRLFD
LDA M2
LDB RDSYM
JSB I.TTY,P
JSB I.OFF
LDA .72
LDB BUFA

JSB I.TTY.I
STA SRCNT
LDA,SSA,RSS
JMP DAT1
STA CCNT
LDA BUFA
STA BAODR
STA BAODR

STO GETCR
JMP DATIN+1
JMP DATIN,1

RDSYM DEF **1
OCT 40007 INPUT PROMPT

* * MESSAGE ON BUFFER OVERFLOW

Datin

LDA .16
LDB DAT2
JSB ERROR
JMP DATIN+1

Datin

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 SAVE LENGTH
STA COUNT
STB BADDR SET BUFFER ADDRESS
CLB SET CHARACTER COUNTER
LOA IMODE SET TTY TO INPUT MODE

TTY.T1 SET TTY.C REQUEST CHARACTER
TTY.T2 GET NEXT CHARACTER
SFS TTY RECORD COMPLETE RETURN

LIA TTY LOAD CHARACTER
JSB PROCS PROCESS CHARACTER
JMP TTY.T1 RECORD COMPLETE RETURN
OUTPUT ASCII RECORDS THROUGH THE TELETYPewriter

(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
LOAD TTY  SAVE TTY INTERRUPT INSTR
STA TEMPI
LOAD LMODE
STA 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
STA TTY  LOAD TTY BOARD BUFFER
SCC TTY,  C  GIVE PRINT COMMAND
SFS TTY  WAIT FOR FLAG
JMP *-1
LOAD TEMPI
SCC TA,  RSS  IS INTERRUPT ENABLED
JMP TP.3  NO
* THIS SECTION CHECKS IF A CHARACTER HAS BEEN TYPED FROM THE KEYBOARD DURING OUTPUT ON TELETYPewriter.

LIA TTY LOAD FROM BOARD BUFFER
AND B177 FIRST 8 BITS SHOULD BE ONES
SZA,RSS
JMP TP.3 NO KEY STRUCK, CONTINUE
LOA IOR1
STA FINISH+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 TI.II IS INTERRUPT MODE SET
JSB I.ON YES, RE-ENABLE KEYBOARD
LOA IOR1
STA FINISH+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 TELETYPewriter 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 TELETYPewriter INTERRUPT MODE
I.ON NOP
LDB TI.II SET JSB INTO INTERRUPT CELL
LDB I.MODE
OTB TTY SET TTY TO INPUT MODE
STC TTY,C SET TTY TO LOOK FOR INPUT
JMP I.ON,I
IT.JJ 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 CRIN 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 DLETER YES, DELETE PREVIOUS CHARACTER
SLB, IN8 NO, CHECK ODD/EVEN FLAG
JMP PROC2 80 = 0, EVEN CHARACTER
PROC1 ALF, ALF 80 = 1, ODD CHARACTER
STA BADDR,I
JMP PROCS,I RECORD HIGH CHARACTER AND RETURN
PROC2 TOR 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)

DELETE SB, RSS IS BUFFER EMPTY
JMP PROC3, I YES, RETURN

CCA NO

ADB A DECREMENT CHARACTER COUNT
SLB, RSS LOW CHARACTER
JMP PROC3, I YES

ADA BADDR NO DECREMENT ADDRESS POINTER
STA BADDR
LDA BADDR, I GET LAST TWO CHARACTERS
ALF, ALF

AND B177 DELETE 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 PROC3
JMP PROC3, I

* * *

* SUBROUTINE GETCH

* RETURN P+1 BUFFER EMPTY
P+2 CHARACTER IN (A)

GETCH NOP
CPB COUNT
JMP GETCH, I BUFFER 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 NOPOCCE,SSA SET (E) = 1, CHECK FOR (A) < 0
CMACLE,INA SET (E) = 0, (A) = -(A)
STACOUNT SAVE CHARACTER COUNT
STBBADDR SET BUFFER STARTING ADDRESS
GLB INITIALIZE OUTPUT COUNT
JMP INIT, I

* * * LNFD EQU .10 LINE FEED
CRRN EQU .13 CARRIAGE RETURN
LFTAR EQU 8137
CLAI EQU 82400
IORI IOR 8200 ADD IN BIT 7

* * * CRLF A DEF CRLF
CRLF OCT 106612

* * * STOP COMMAND SERVICE *

I.STP NOPOCJSB I.OFF TURN OFF KEYBOARD INTERRUPT
JSB CRLF NEW LINE
LDA *4
LDB STOPA PRINT STOP
JSB TIY.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 ITY,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  
LDB M10  
JSB CNBIN  
JMP CNDEC,I  

CNOCT NOP  
LDB M8  
JSB CNBIN  
JMP CNOCT,I
* 

CNBIN

NOP
STB TEMP5
LDB A00
STB TEMP
STB TEMP1
STB TEMP2
LDB CNMBR
STB TEMP3
JMP CNBN1
JSB DVUKN
ADB TEMP3,I
STB TEMP3,I
SZA,RSS
JMP CNBN2
JSB DVUKN
BLF,5LF
ADB TEMP3,I
STB TEMP3,I
ISZ TEMP3
SZA
JMP CNBN1
LDA TEMP
LDB TEMP2
STB TEMP
STA TEMP2
LDB CNMBR
JMP CNBIN,I

* 

A00 ASC 1,00
CNMBR DEF TEMP
**DVUKN**  NOP
**CLB**  TEMP4
**STB**  TEMP4

**DVUK1**  STA b
**DVUK2**  ADD TEMP5
           ISZ TEMP4
           SSA, RSS
           JMP DVUK1
           CLEAR (b) TO ALLOW EXIT
           SSB
           JMP DVUK2
           EXIT IF POSITIVE
           SSD TEMP5
           JMP DVUKN, I

           DONE IF (A) IS NEG AND (B) IS POS
           ORIG NUMBER TO CONVERT WAS NEG

           REMAINDER TO (B)
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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA EDTFG</td>
<td>EDIT OPERATION</td>
</tr>
<tr>
<td>SZA</td>
<td></td>
</tr>
<tr>
<td>JMP SCB1</td>
<td>YES, ADDRESS OF ENTRY IN SCB</td>
</tr>
<tr>
<td>LDB ADDR1</td>
<td>NO, ADDRESS OF ENTRY IN SCB</td>
</tr>
<tr>
<td>STA B,I</td>
<td></td>
</tr>
<tr>
<td>INB</td>
<td></td>
</tr>
<tr>
<td>LDA PREV,I</td>
<td>SUCCESSOR ADDRESS</td>
</tr>
<tr>
<td>STA B,I</td>
<td>STORE SUCCESSOR ADDRESS</td>
</tr>
<tr>
<td>LDA PREV</td>
<td>ADDRESS OF PREVIOUS INSTRUCTION</td>
</tr>
<tr>
<td>STA B,I</td>
<td></td>
</tr>
<tr>
<td>LDA CUSTN</td>
<td>PREVIOUS STATEMENT NUMBER</td>
</tr>
<tr>
<td>ADA STINC</td>
<td>STATEMENT NUMBER INCREMENT</td>
</tr>
<tr>
<td>STA CUSTN</td>
<td>CURRENT USER STATEMENT NUMBER</td>
</tr>
<tr>
<td>INB</td>
<td></td>
</tr>
<tr>
<td>STA B,I</td>
<td>STORE STATEMENT NUMBER</td>
</tr>
<tr>
<td>LDB ADDR1</td>
<td>ADDR OF PREVIOUS FOR NEXT ENTRY</td>
</tr>
<tr>
<td>LDB ADDR1</td>
<td></td>
</tr>
<tr>
<td>SCB1</td>
<td></td>
</tr>
<tr>
<td>STA B,I</td>
<td></td>
</tr>
<tr>
<td>LDA LNH2</td>
<td>WORD HOLDING LENGTHS</td>
</tr>
<tr>
<td>STA B,I</td>
<td></td>
</tr>
<tr>
<td>INB</td>
<td></td>
</tr>
<tr>
<td>LDA ASMFG</td>
<td>ASSEMBLY FLAG</td>
</tr>
<tr>
<td>SZA,RSS</td>
<td></td>
</tr>
<tr>
<td>JMP SCB2</td>
<td>COMMENT STATEMENT</td>
</tr>
<tr>
<td>CLE,ELA</td>
<td>STORE ASSEMBLY INFORMATION IN (E)</td>
</tr>
<tr>
<td>LDA ZADD</td>
<td>ADDRESS OF ASSEMBLY</td>
</tr>
<tr>
<td>RAL,ERA</td>
<td>ASSEMBLY INFORMATION IN BIT 15</td>
</tr>
<tr>
<td>SCB2</td>
<td></td>
</tr>
<tr>
<td>STA B,I</td>
<td></td>
</tr>
<tr>
<td>INB</td>
<td></td>
</tr>
<tr>
<td>LDA LNH</td>
<td>LENGTH OF ASSEMBLY</td>
</tr>
<tr>
<td>STA B,I</td>
<td></td>
</tr>
<tr>
<td>INB</td>
<td></td>
</tr>
<tr>
<td>LDA SRGNT</td>
<td>NUMBER OF WORDS IN SOURCE INPUT</td>
</tr>
<tr>
<td>STA STORE</td>
<td></td>
</tr>
<tr>
<td>LDA BUF A</td>
<td>INPUT BUFFER ADDRESS</td>
</tr>
<tr>
<td>JSB WMOVE</td>
<td>MOVE INTO SCB</td>
</tr>
<tr>
<td>JMP STSCB,I</td>
<td></td>
</tr>
</tbody>
</table>

277
DEFINE LABEL PRECEDING MNEMONIC

LBOEF NOP
LDB LBLFG LABEL FLAG
SZB, RSS LABEL PRESENT
JMP LBOEF, I NO, RETURN
LDB LBLAD LABEL ADDRESS
LDA B, I PREVIOUS REFERENCE TO LABEL
SZA
JMP LBOEF1 YES
LDA ZADD NO
CLE
JSB SLBL, I STORE LABEL IN SYMBOL TABLE
JMP LBOEF, I
LBDF1 CLA
STA IDRCT DIRECT REFERENCE
ADB .3
STB RSTRE
LDA B,I UNDEFINED DIRECT REFERENCE
ADB D700
JSB FWDRF YES, CLEAR UP ALL DIRECT REFS
ISZ PSTRE
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 CHI 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 introductory 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
- **#EXECUTE**
  - 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(DUMP)**

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

- **CPA D**
  - No, dump directive
- **JMP DUMP**
  - Yes
**E(Edit)**

* 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 3 CR-LF
- LDA .20
- LDB EMSG
- 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 778 YES, STOP
- JMP CMAND PRESS RUN TO CONTINUE
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

<table>
<thead>
<tr>
<th>CPA L</th>
<th>LIST DIRECTIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSS</td>
<td>YES</td>
</tr>
<tr>
<td>JMP SYST4</td>
<td>NO</td>
</tr>
<tr>
<td>JSB RDCOM</td>
<td>READ UP TO COMMA</td>
</tr>
<tr>
<td>JMP SYST3</td>
<td>LIST WHOLE PROGRAM</td>
</tr>
<tr>
<td>JSB TWNT,I</td>
<td>READ STATEMENT NUMBERS</td>
</tr>
<tr>
<td>JMP SYST2</td>
<td>ONE NUMBER FOUND</td>
</tr>
<tr>
<td>RSS</td>
<td>ERROR</td>
</tr>
<tr>
<td>RSS</td>
<td>TWO NUMBERS FOUND</td>
</tr>
<tr>
<td>JMP SYST6</td>
<td>ERROR</td>
</tr>
<tr>
<td>LDB NUM1</td>
<td>ERROR</td>
</tr>
</tbody>
</table>
CMB, IN8
ADB NUM2
SSB
JMP CMAND
LDA FSTMT
CMA,INA
ADA NUM2
SSA
JMP SYST7
JMP +3
SYST2
LDA CUSTN
STA NUM2
LDA NUM1
CMA,INA
ADA CUSTN
SSA
JMP SYST7
JMP +5
SYST3
LDA FSTMT
STA NUM1
LDA CUSTN
STA NUM2
LDA M10
JSB NWLNS
JSB LIST
LDA M10
JSB NWLNS
JMP CMAND

FIRST NUMBER GREATER THAN SECOND
YES, IGNORE LIST INSTRUCTION
CHECK RANGE OF SECOND STATEMENT
NUMBER
TOO SMALL
YES
SET TERMINATOR
CHECK RANGE OF FIRST NUMBER
IN RANGE
NO, TOO BIG
SET PARAMETERS FOR COMPLETE LISTING
OUTPUT 10 CR-LF
CALL LIST ROUTINE
RETURN TO CONTROLLER
* * * * *
*S(QUENCE), 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
JMP SEQ
SEQUENCE DIRECTIVE
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 \textbf{LDA} \texttt{14} \\
\textbf{LDB} \texttt{+2} \\
\textbf{JMP} \textbf{ERCAL}

\textbf{DEF ERR1} \hspace{1cm} \text{BAD DATA INPUT}

SYST7 \textbf{LDA} \texttt{+30} \\
\textbf{LDB} \texttt{+2} \\
\textbf{JMP} \textbf{ERCAL}

\textbf{DEF ERR2} \hspace{1cm} \text{STATEMENT NUMBERS OUT OF RANGE}

\textbf{SEQ} \hspace{1cm} \textbf{JSB SQNC}, \textit{I} \\
\textit{CCA}, \textit{RSS} \hspace{1cm} \text{STATEMENT NUMBER INPUT ERROR} \\
\hspace{1cm} \textbf{JMP} \textbf{SEQ1}

\textbf{SET SEQUENCE FLAG} \\
\textbf{RETURN TO SYSTEM CONTROLLER FOR INPUT}

\textbf{STA} \textbf{SEQFG} \hspace{1cm} \text{SET SEQUENCE FLAG} \\
\hspace{1cm} \textbf{JMP} \textbf{CHAND}

\textbf{SEQ1} \hspace{1cm} \textbf{CLA} \\
\hspace{1cm} \textbf{STA} \textbf{SEQFG} \hspace{1cm} \text{CLEAR SEQUENCE FLAG} \\
\hspace{1cm} \textbf{LDB} \textbf{FIRST} \hspace{1cm} \text{ADDRESS OF FIRST STATEMENT}

\textbf{SEQ2} \hspace{1cm} \textbf{LDA} \texttt{B,I} \hspace{1cm} \text{ADDRESS OF NEXT STATEMENT} \\
\hspace{1cm} \textbf{STA} \textbf{AHEAD} \hspace{1cm} \text{SAVE ADDRESS} \\
\hspace{1cm} \textbf{ADB} \texttt{.2} \hspace{1cm} \text{ADDRESS OF STATEMENT NUMBER} \\
\hspace{1cm} \textbf{LDA} \textbf{CUSTN} \hspace{1cm} \text{CURRENT STATEMENT NUMBER} \\
\hspace{1cm} \textbf{ADA} \textbf{STINC} \hspace{1cm} \text{ADD INCREMENT} \\
\hspace{1cm} \textbf{STA} \textbf{CUSTN} \\
\hspace{1cm} \textbf{STA} \texttt{B,I} \\
\hspace{1cm} \textbf{LDB} \textbf{AHEAD} \\
\hspace{1cm} \textbf{CPB} \textbf{NEXT} \hspace{1cm} \text{TERMINATION} \\
\hspace{1cm} \textbf{JMP} \textbf{SYST3} \hspace{1cm} \text{LIST THE PROGRAM} \\
\hspace{1cm} \textbf{JMP} \textbf{SEQ2} \hspace{1cm} \text{NO, CONTINUE}
DUMP REGISTER CONTENTS AND DATA ADDRESSES

DUMP

LDA SAVA
LDB DUMP1
JSB RGDP1
LDA SAVB
LDB DUMP1+1
JSB RGDP1
LDA SAVEO
LDB DUMP1+2
JSB E00MP
LDA SAVEO
RAR
LDB DUMP1+3
JSB E00MP

DMP1

LDA M2
JSB NWLNS
LDA 16
LDB RGDM4
JSB TTY.*P
LDA 46
LDB RGDM5
JSB TTY.*P

SET FLAG, JUMP TO SYSTEM CONTROLLER

CCB
STB DMPFG
JMP CMAND

RETURN POINT FROM SYSTEM CONTROLLER

DMP2

CPA D
JMP *+4
CLB
STB DMPFG
JMP CMAND
* 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,IN8 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 RGOM3
LDA M5
STA TEMP1
IN8 CHANGE MESSAGE ADDRESS
ISZ TEMP1 TO IGNORE BLANKS
JMP *-2
LDA M9
JSB TTY.P
JSB ASCDC BINARY TO ASCII DECIMAL
LDA M1
LDB RGOM3
JSB TTY.P
LDA M8
LDB RGOM2
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
DEF E
DEF O

**DUMP2**
DEF TEMP7

* PREPARE TO DUMP EITHER (E) OR (O)
* ENTER (A) REGISTER STATUS IN BIT 15
* (B) ADDRESS OF REGISTER NAME

**EODMP**
NOP
CLE,ELA
LDA .48
SEZ
INA
ALF,ALF
JSB R6DPR2
JMP EODMP,I

ADDRESS OF REGISTER NAMES

MOVE BIT FLAG INTO (E)
ASCII ZERO
SHIFT TO FIRST CHARACTER POSITION
DUMP REGISTER
* 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 CNOC7 BINARY TO OCTAL ASCII
LDA .6
JSB TTY.P PRINT OCTAL VALUE
LDA M19
LDB RGDM3
JSB TTY.P
JSB ASCOC 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 #177 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
OPER2  LOA  .11
       LDB  *+2
       JMP  ERCAL
* DEF  ERR9  NO LABEL FOUND
* DEF  ERR8  UNDEFINED LABEL IN OPERAND
* DEF  ERR7  OPERAND IS UNDEFINED
EXECUTE USER PROGRAM

XEQ1 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 XUSRPR FIRST LOCATION IN USER PROGRAM
STA TEMP
LDB BUF1 ADDRESS OF UNDEFINED REFERENCES
STB TEMP1

XEQ2 LDB TEMP1,I ALL UNDEF REF RETURNED TO PROGRAM
CPB ZERO
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 CHAND 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**

<table>
<thead>
<tr>
<th>SSTDF</th>
<th>NOP</th>
<th>LDA XSTBL</th>
<th>ADDRRSS OF SYMBOL TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>JMP *+3</td>
<td></td>
</tr>
<tr>
<td>SST1</td>
<td>LDA RSTRE</td>
<td>RETRIEVE ADDRESS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADA .6</td>
<td>STA RSTRE</td>
<td>SAVE PRESENT POSITION IN SYM TBL</td>
</tr>
<tr>
<td></td>
<td>LDB YSTBL</td>
<td>UPPER BOUND OF SYMBOL TABLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CMN,INB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADB A</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSBNRSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JMP SSTDF,I</td>
<td>YES, RETURN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADA .2</td>
<td>LDB A,I</td>
<td>EXAMINE LABEL INFORMATION</td>
</tr>
<tr>
<td></td>
<td>CLE,ERB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEZ,RSI</td>
<td></td>
<td>LABEL DEFINED</td>
</tr>
<tr>
<td></td>
<td>JMP SST1</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>ADA .2</td>
<td>LDB A,I</td>
<td>ADDRESS IN SOURCE CODE BLOCK</td>
</tr>
<tr>
<td></td>
<td>STB ADDR1</td>
<td>SAVE ADDRESS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADA .1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STA LKPSN</td>
<td></td>
<td>SAVE LINK POSITION</td>
</tr>
<tr>
<td></td>
<td>LDB A,I</td>
<td></td>
<td>LINK TO SST</td>
</tr>
<tr>
<td></td>
<td>SZB,RS</td>
<td></td>
<td>SST ENTRIES</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP SST1</td>
<td>No, examine next area in sym tbl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STB SSTAD</td>
<td>Yes, save address in SST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA B,I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA OPNUM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA B,I</td>
<td>Link back to symbol table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSA RSS</td>
<td>Indirect bit set</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLA</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA IDRCT</td>
<td>Set indication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STB POSN</td>
<td>Save present position in SST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA OPNUM</td>
<td>Operand number value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDB ADDR1</td>
<td>SCB address of label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSB FNDAD</td>
<td>Find address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSA SST4</td>
<td>Address found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP SST4</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA ZADD</td>
<td>Yes, save address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISZ POSN</td>
<td>Next location in SST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA POSN,I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADA D700</td>
<td>Forward references</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSB FWDRF</td>
<td>Yes, clear up fwd refs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISZ POSN</td>
<td>Address of link in SST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDB POSN,I</td>
<td>Value of link</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STB LKPSN,I</td>
<td>Store in new location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA SSTAD,I</td>
<td>Address of entry in SST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA TEMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA M4</td>
<td>Save address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
STA TEMP
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 POSITION OF LINK
ADA .2 NEW LINK ADDRESS POINTER
STA LKPSN
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.
PLC1

LDB XPLC
JMP *+3
LDB STORE
Add \_2
STB STORE
LDB YPLC
SMA, IN
Add B
SSA, RSS
JMP PLCDF, I
LDA B, I
SSA, RSS
JMP PLC1
SSA, RSS
LDA B, I
FCA, CLE, ERA
STA ADDR1
INB
LDA B, I
LDB ADDR1
JSB FNAD
SSA, RSS
JMP PLC1
STA ZADD
LDA ADDR1  SCB ADDRESS
ADDA  4  ADDRESS OF ASSEMBLY
LDB A,I  
STB ADDR1  SAVE ADDRESS
ADDB D340  CORRESPONDING ADDRESS IN
STB ADDR2  ADDRESS BLOCK
LDA ZADD
JSB DATA DETERMINE ADDRESS TYPE
JSB IDIR  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 CPI8  CURRENT PAGE INDIRECT BIT
SWP  STORE ADDRESS IN (3)
AND B1760  SAVE INSTRUCTION SKELETON
ADDA B
PLC2  STA ADDR1,I  RETURN INSTRUCTION
STA STORE,I  CLEAR AREA IN PLC TABLE STORE
JMP PLC1

*  TWO WORD ASSEMBLY

Plc3  ISZ ADDR1  POSITION IN ADDRESS BLOCK
LDA ADDR2  ADDRESS BLOCK
IOR MNEG  INDIRECT BIT
JMP PLC2
**FIND ADDRESS FOR COMPOUND OPERAND**

**ENTER (A) OPERAND NUMBER VALUE**

(8) SOURCE CODE BLOCK ADDRESS OF LABEL

**RETURN (A) = -1 ADDRESS NOT FOUND**

ADDRESS IN ASSEMBLED CODE

---

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA VALUE</td>
<td>DETERMINE DIRECTION OF SEARCH</td>
</tr>
<tr>
<td>SSA</td>
<td></td>
</tr>
<tr>
<td>JMP FNDD5</td>
<td></td>
</tr>
<tr>
<td>LDA B,I</td>
<td>SEARCH AHEAD</td>
</tr>
<tr>
<td>STA AHEAD</td>
<td>ADDRESS OF NEXT ENTRY</td>
</tr>
<tr>
<td>ADB .5</td>
<td></td>
</tr>
<tr>
<td>LDA B,I</td>
<td>LENGTH OF ASSEMBLY</td>
</tr>
<tr>
<td>STA LENGTH3</td>
<td>SAVE LENGTH OF ASSEMBLY</td>
</tr>
<tr>
<td>CMA,INA</td>
<td></td>
</tr>
<tr>
<td>ADA VALUE</td>
<td></td>
</tr>
<tr>
<td>SSA</td>
<td></td>
</tr>
<tr>
<td>JMP FNDD3</td>
<td></td>
</tr>
<tr>
<td>CLE, SZA, RSS</td>
<td></td>
</tr>
<tr>
<td>STA VALUE</td>
<td>RETAIN NEW VALUE</td>
</tr>
<tr>
<td>LDB AHEAD</td>
<td>ADDRESS OF NEXT ENTRY</td>
</tr>
<tr>
<td>CPB NEXT</td>
<td>TERMINATION</td>
</tr>
<tr>
<td>FNDD2</td>
<td></td>
</tr>
<tr>
<td>CCA, RSS</td>
<td>YES</td>
</tr>
<tr>
<td>RSS</td>
<td></td>
</tr>
<tr>
<td>JMP FNDAI</td>
<td>RETURN ADDRESS NOT FOUND</td>
</tr>
<tr>
<td>SEZ, RSS</td>
<td></td>
</tr>
<tr>
<td>JMP FNDD1</td>
<td></td>
</tr>
<tr>
<td>INB</td>
<td>BACK UP IN SCAN OF SCB</td>
</tr>
<tr>
<td>LDB B,I</td>
<td>TO RETRIEVE ADDRESS OF ASSEMBLY</td>
</tr>
<tr>
<td>ADB .5</td>
<td>FOR PREVIOUS INSTRUCTION</td>
</tr>
</tbody>
</table>

---
* ADDRESS FOUND

**FNDD3** ADA LNH3 LENGTH OF ASSEMBLY
**FNDD4** ADB M1 ADDRESS IN ASSEMBLED CODE

LOB B,I
ELL,CLE,ERB CLEAR BIT 15 IF NECESSARY
ADA B RETAIN ADDRESS IN (A)
JMP FNDDA,I

* SEARCH BACKWARD IN SOURCE CODE BLOCK

**FNDD5** CMA,INA

**FNDD6** IN3

LOB B,I ADDRESS OF PREVIOUS INSTRUCTION
CPB M1 TERMINATION
JMP FNDD2 YES
IN3
LDA B,I RETAIN ADDRESS OF PREVIOUS INSTR
STA BACK
ADB 
LDA B,I LENGTH OF ASSEMBLY
CMA,INA SUBTRACT LENGTH OF ASSEMBLY
ADA VALUE FROM CONSTANT
S3A
JMP FNDD7 POSITION FOUND
S3A,RSS
JMP FNDD7
STA VALUE SAVE NEW VALUE
LDB BACK ADDR OF PREV IN SGB ENTRIES
CPB M1 TERMINATOR
JMP FNDD2 YES, ADDRESS NOT FOUND
JMP FNDD6+4 NO

**FNDD7** CMA,INA
JMP FNDD4
**DEFINE FORWARD REFERENCES**

<table>
<thead>
<tr>
<th>FWDRF</th>
<th>NOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWDR1</td>
<td>ADA B700 ADDRESS OF FIRST REFERENCE</td>
</tr>
<tr>
<td></td>
<td>ADA JMP ACTUAL ADDRESS</td>
</tr>
<tr>
<td></td>
<td>STA WMOVE</td>
</tr>
<tr>
<td></td>
<td>LDA WMOVE,I RETRIEVE INSTRUCTION</td>
</tr>
<tr>
<td>AND B1777 RELATIVE ADDR OF NEXT INSTR</td>
<td></td>
</tr>
<tr>
<td>STA SAVEE RETAIN ADDRESS</td>
<td></td>
</tr>
<tr>
<td>LDA WMOVE,I RETRIEVE INSTRUCTION</td>
<td></td>
</tr>
<tr>
<td>AND B1760 REMOVE POINTER TO NEXT REFERENCE</td>
<td></td>
</tr>
<tr>
<td>STA WMOVE,I</td>
<td></td>
</tr>
<tr>
<td>AND B0700 LENGTH OF ASSEMBLY</td>
<td></td>
</tr>
<tr>
<td>SZA</td>
<td></td>
</tr>
<tr>
<td>GLB,INB,RSS ONE WORD ASSEMBLY</td>
<td></td>
</tr>
<tr>
<td>GCB TWO WORD ASSEMBLY</td>
<td></td>
</tr>
<tr>
<td>SIB LENTH ADDRESS OF ASSEMBLY</td>
<td></td>
</tr>
<tr>
<td>LDA WMOVE</td>
<td></td>
</tr>
<tr>
<td>ADA D340 ADDR IN ADDR BLOCK</td>
<td></td>
</tr>
<tr>
<td>STA ADDR3</td>
<td></td>
</tr>
<tr>
<td>LDA ZADD ASSEMBLY ADDRESS</td>
<td></td>
</tr>
<tr>
<td>JSB DATAD UPDATE DATA ADDRESS</td>
<td></td>
</tr>
<tr>
<td>JSB IDIRT</td>
<td></td>
</tr>
<tr>
<td>STA ADDR3,I STORE ADDRESS</td>
<td></td>
</tr>
<tr>
<td>LDA ADDR3</td>
<td></td>
</tr>
<tr>
<td>LDB LENTH GET ADDRESS IN ADDRESS BLOCK</td>
<td></td>
</tr>
<tr>
<td>SSB,RSS FOR INSTRUCTION</td>
<td></td>
</tr>
<tr>
<td>AND B1777 REL ADDR FOR 1 WORD ASSEMBLY</td>
<td></td>
</tr>
<tr>
<td>STA TEMP</td>
<td></td>
</tr>
<tr>
<td>LDA WMOVE,I RETRIEVE UNDEF INSTRUCTION</td>
<td></td>
</tr>
<tr>
<td>ADA TEMP ADDRESS</td>
<td></td>
</tr>
<tr>
<td>FOR GPIB CURRENT PAGE INDIRECT BIT</td>
<td></td>
</tr>
<tr>
<td>STA WMOVE,I STORE INSTRUCTION</td>
<td></td>
</tr>
<tr>
<td>LDA SAVEE POINTER TO NEXT INSTRUCTION</td>
<td></td>
</tr>
<tr>
<td>ADA D700 RETURN TO SYMBOL TABLE</td>
<td></td>
</tr>
<tr>
<td>SSA</td>
<td></td>
</tr>
<tr>
<td>JMP FWDR1 NO, FORWARD REFERENCE</td>
<td></td>
</tr>
<tr>
<td>JMP FWDRF,I</td>
<td></td>
</tr>
</tbody>
</table>
** SYSTEM LIST ROUTINE

** ENTER (A) > 0 SYSTEM DIRECTIVE

** (A) < 0 CALL FROM EDITOR

**

LIST

NOP
STA ENDFG CLEAR PRINT FLAG
CALL FROM EDIT
JMP LST1 FIRST ENTRY IN SCB

LST1

LOAD SUCAD GET PREVIOUS SUCCESSOR ADDRESS
CPA NEXT TERMINATION
JMP LST3 YES
LDB A,I ADDRESS OF NEXT STATEMENT
RSB RSS END OF USER PROGRAM
JMP LST3
SIB SUCAD NO, SAVE NEXT ADDRESS

ADD A,2
STA ADDR
LOAD A,I
GET STATEMENT NUMBER
STA SNUM
SEZ
JMP LST2 PRINT FLAG
SET
LDB NUM1
CMN INB
CHECK RANGE
ADD SNUM
CLE,SSB
JMP LST1 GET NEXT STATEMENT
CMA,INA -STATEMENT NUMBER
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA STNUM</td>
<td>STATEMENT NUMBER</td>
</tr>
<tr>
<td>JSB CNDEC</td>
<td>BINARY TO ASCII DECIMAL</td>
</tr>
<tr>
<td>LDA M6</td>
<td></td>
</tr>
<tr>
<td>JSB TTY.P</td>
<td>PRINT STATEMENT NUMBER</td>
</tr>
<tr>
<td>LDA M1</td>
<td></td>
</tr>
<tr>
<td>LDB RGCM3</td>
<td>ADDRESS OF 1 BLANK</td>
</tr>
<tr>
<td>LDB ADDR</td>
<td></td>
</tr>
<tr>
<td>LDA B1</td>
<td>LENGTH</td>
</tr>
<tr>
<td>ALF,ALF</td>
<td></td>
</tr>
<tr>
<td>AND B177</td>
<td>NUMBER OF CHARMS IN SOURCE INPUT</td>
</tr>
<tr>
<td>ADB .3</td>
<td>ADDRESS OF SOURCE LINE</td>
</tr>
<tr>
<td>JSB TTY.P</td>
<td>PRINT LINE, AND CR-LF</td>
</tr>
<tr>
<td>JMP LST1</td>
<td>GET NEXT LINE</td>
</tr>
<tr>
<td>LDB ENDFG</td>
<td>END MESSAGE</td>
</tr>
<tr>
<td>JMP LIST,I</td>
<td>NO, RETURN</td>
</tr>
<tr>
<td>LDA M2</td>
<td></td>
</tr>
<tr>
<td>JSB NHLNS</td>
<td></td>
</tr>
<tr>
<td>LDA .12</td>
<td></td>
</tr>
<tr>
<td>LDB LSTMG</td>
<td>PRINT -LIST ENDS-</td>
</tr>
<tr>
<td>JSB TTY.P</td>
<td></td>
</tr>
<tr>
<td>JMP LIST,I</td>
<td></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>GROUP NUMBER</th>
<th>INSTRUCTION TYPE</th>
<th>OPERAND REQUIRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO OPERAND REQUIRED</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INPUT / OUTPUT</td>
<td>CLEAR FLAG MAY BE PRESENT</td>
</tr>
<tr>
<td>3</td>
<td>INPUT / OUTPUT</td>
<td>CHANNEL NUMBER EXPECTED</td>
</tr>
<tr>
<td>4</td>
<td>INPUT / OUTPUT</td>
<td>CHANNEL NUMBER EXPECTED CLEAR FLAG MAY BE SPECIFIED</td>
</tr>
<tr>
<td>5</td>
<td>EXTENDED ARITH</td>
<td>NUMBER OF SHIFTS</td>
</tr>
<tr>
<td></td>
<td>REGISTER REFERENCE</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MEMORY REFERENCE</td>
<td>LABEL NUMBER</td>
</tr>
<tr>
<td>7</td>
<td>EXTENDED ARITH</td>
<td>ASTERISK INDIRECT FLAG</td>
</tr>
<tr>
<td>PSEUDO OP</td>
<td>OPERAND</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>8</td>
<td>END</td>
<td>NO OPERAND REQUIRED</td>
</tr>
<tr>
<td>9</td>
<td>ASC</td>
<td>LENGTH AND LIST OF ASCII DATA</td>
</tr>
<tr>
<td>10</td>
<td>DEC</td>
<td>REAL OR DECIMAL INTEGER</td>
</tr>
<tr>
<td>11</td>
<td>OCT</td>
<td>OCTAL INTEGER VALUES</td>
</tr>
<tr>
<td>12</td>
<td>EQU</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>13</td>
<td>ABS</td>
<td>ADDRESS VALUE</td>
</tr>
<tr>
<td>14</td>
<td>BSS</td>
<td>VALUE</td>
</tr>
<tr>
<td>15</td>
<td>DEF</td>
<td>ADDRESS DEFINITION</td>
</tr>
</tbody>
</table>

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 LEX1 YES, RETURN
CPA BLANK BLANK, NO LABEL
JMP LEX2 YES
JSB LTPR,I NO, LETTER OR PERIOD
JMP LXR2 NO, ILLEGAL FIRST CHARACTER
LDB LAB1 YES, ADDRESS FOR FIRST LABEL
JSB LBRD,I READ LABEL
JMP LXR9 ILLEGAL CHARACTER BEGINS LABEL
JSB GETCR ILLEGAL TERMINATION AFTER LABEL
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
S: 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

<table>
<thead>
<tr>
<th>LEX1 STA LBLFG</th>
<th>ADDRESS IN ASSEMBLED CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEX2 JSB NTBLK</td>
<td>NEXT NON BLANK CHARACTER</td>
</tr>
<tr>
<td></td>
<td>NO OPCODE FOUND</td>
</tr>
<tr>
<td>JSB BCKSP</td>
<td>RETURN LAST CHAR TO BUFFER</td>
</tr>
<tr>
<td>LDA M3</td>
<td></td>
</tr>
<tr>
<td>STA TEMP3</td>
<td>READ THREE CHARACTERS</td>
</tr>
<tr>
<td>CLE</td>
<td></td>
</tr>
</tbody>
</table>

| LEX3 JSB GETCR  | READ CHARACTER            |
|                 | MNEMONIC NOT FOUND        |
| SEZ,RSS        |                            |
| ALF,ALF        | SHIFT ALTERNATE CHAR      |
| IOR OPADD,I    | STORE CHAR IN OPCODE BUFFER|
| STA OPADD,I    |                            |
| SEZ,CME        | ADVANCE BUFFER ADDRESS    |
| ISZ OPADD      | CHARACTER COUNT           |
| ISZ TEMP3      |                            |

| JMP LEX3       | READ NEXT CHAR            |
| JSB MNEM       | LOOK UP OPCODE 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                       |
| CGB            |                            |
| ADA M8         | MACHINE CODE OR DATA EDIT |
| SSA,RSS        |                            |
| STB ASMFG      | DATA                      |
| JMP LEX,I      |                            |
**LEX4**  
JSB GETCR  
GET TERMINATOR CHAR  
RSS  
JMP LEX5  

**LEX5**  
* END OF LINE CHECK INSTRUCTION NUMBER  
*  
LOA 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  
CPA BLANK VALID TERMINATOR AFTER OPCODE  
RSS YES  
JMP LXR8 NO  
LOA 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  
LOA 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
   JMP LEX35
   JSB LRECK
   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 IDROT FLAG TO SIGNAL BSS
   JMP LEX,I

LEX33 JSB VAL
   JMP LEX32+1
JMP LEX10 NO, ADVANCE TO REGISTER REFERENCE
JSB NM9R,I READ CHANNEL NUMBER
JMP LEX9 FIRST CHAR NOT A NUMBER
LDB D100
JSB RANGE CHECK RANGE OF OPERAND
JMP LEX8
LDB INSNM INSTRUCTION
CPB .3 TYPE 3 OPERAND
JMP LXR8 YES, ILLEGAL CHAR IN OPERAND
CPA COMMA COMMA BEFORE 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 LABD,I READ LABEL
JMP LXR9 ILLEGAL CHAR BEGINS LABEL
LDB LAB2 MEMORY ADDRESS OF LABEL
JSB LLOOK SYMBOL TABLE LOOK UP
SZA,RSS RETURN POSITIVE ADDRESS IN
JMP LXR10 ASSEMBLED CODE
SSA POSITIVE ADDRESS
JMP LXR10 NO, UNDEFINED LABEL
LDA A,1 RETRIEVE VALUE
JMP LEX8
LEX10
CPA .5 TYPE 5 OPERAND
RSS
JMP LEX11
JSB NM9R,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,1
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
   JMP LEX13 RSS
   JMP LEX13 NO
   ISZ LENTH
   LEX12 JSB OPRC.I
   JMP LEX.I

* END PSEUDO OP BRANCHES TO EXECUTE ROUTINE

* LDB EDTFG EDIT OPERATION
   STB LENTH TWO WORD ASSEMBLY
   JMP LXR12 ILLEGAL OP CODE DURING EDIT
   JMP XEQ.I YES BEGIN EXECUTION

* THE REMAINDER OF THE INSTRUCTIONS ARE FOR DATA DEFINITION PURPOSES

* LEX13 LDB M29
   STB LNTH2
   INB
   STB TEMP3
   LDB DATABF ADDRESS OF DATA BUFFER
   STB DATBF 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, 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

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

JMP LXR8, MULTIPY BY 2 FOR CHARACTER COUNT
CMB, INB
STB TEMP

LEX14
CLO
CLE
SOC

JMP LEX15
JSB GECR, NEXT CHARACTER
RSS, NONE FOUND

JMP LEX15+1

LEX15
LOA BLANK
SEZ, RSS
ALF, ALF
IDR TEMP3
STA TEMP3
SEZ, CME
JSB STDAT, STORE DATA IN BUFFER
STA TEMP3
ISZ TEMP
JMP LEX14
JMP LEX,1
* LEX16 CPA .10 DEC PSEUDO OP
  RSS
  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 LABEL_FLAG
LDB LBLFG
SZB RSS
JMP LXR14 NO LABEL PRECEDES EQU
JSB L33CK READ IN AND EXAMIN OPERAND
JMP LEX26 LABEL NOT FOUND
JMP LXR10 LABEL IS UNDEFINED
ADD OPNUM ADD IN CONSTANT
JSB OTRG,I CHECK RANGE OF ADDRESS
STA ZADD ADDRESS IN ASSEMBLED CODE
JS3 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
ADD 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
ADD M1
STB YDATA UPPER BOUND OF DATA BLOCK
JMP LEX,I
* LEX28 CPA .13 ABS PSEUDO OP
  JMP LEX31 READ IN OPERAND
  JSB LABCK NO LABEL
  JMP LEX29 NO LABEL
  JMP LEX30 UNDEFINED LABEL
  JSB DTRG,I DEFINED LABEL
  ADA OPNUM Include constant
  JSB DTRG,I CHECK ADDRESS RANGE
  LOA A,I RETRIEVE DATA ADDRESS
  SZA
  JMP LXR17 UNDEFINED OPERAND
  JSB SIDAT STORE DATA
  JMP LEX,I

* LEX29 CLA ADA OPNUM
  SSA NEGATIVE
  JMP LXR13 YES, ERROR
  LDB D100
  ADA A
  SSB RSS CHECK RANGE OF NUMERIC
  JMP LXR13
  JSB SIDAT STORE DATA VALUE IN BUFFER
  JMP LEX,I

* LEX30 CCB REQUEST USER ENTRY
  JSB VAL
  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 OPERATIONS
REFERENCE SUBSEQUENT DATA ENTRIES.
* LEX35 CPA .15 DEF PSEUDO OP
RSS
JMP LXR16

QCA
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 INDIRECT BIT
RSS YES
JMP LXR11 INDIRECT FLAG
STA IDRCT ADDRESS OF LABEL
LEX36 LDB LAB2 SYMBOL TABLE LOOK UP
JSB LOKUP LABEL EXIST
SZA,RSS LABEL DEFINED
JMP LEX37 NO
SSA
JMP LEX37 .3
ADB YES
LEX37  LDB  EDTFG  UNDEFINED LABEL NOT PERMITTED 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
       JSZ  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 CUT 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
(3) UPPER BOUND OF OPERAND VALUE

RANGE N OP
STA OPNUM CHANNEL NUMBER/NUMBER OF SHIFTS
SSA POSITIVE
JMP LXR13 NO, VALUE OUT OF RANGE
ADA 8
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

STORE DATA IN SPECIAL STORE BUFFER DURING LEXICAL SCAN
ENTER (A) DATA ITEM TO STORED IN BUFFER

STDOUT N OP
STA DATPT, I STORE DATA IN BUFFER
CLA
ISZ DATPT ADVANCE POINTER
ISZ LEN TH COUNT LENGTH
ISZ L NTH 2 DATA BUFFER OVERFLOW
JMP STDOUT, 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
- JSB LXNTY
- LDB SRCNT
- STB RDCOM
- LDA BUFA
- LDB BUFB
- STA HMOVE
- STB BUFA

**JUMP TO SYSTEM CONTROLLER TO READ INPUT**

- JMP CNTRL,I

**RETURN POINT FROM CONTROLLER**

- LXRTN
- JSB BCKSP
- LDB RDCOM
- STB SRCNT
- LDA HMOVE
- STA BUFA
- CLR
- STB ABSSF
- JSB NMBR,I
- JMP LXR8
- 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
SZB, RSS
JMP LABCK,I
SSB
JMP LXR8
LDB LAB2
JSB LOKUP
SSA
JMP LABC1
SSZ, RSS
JMP LABC1
JSB DIRG,I
ISZ LABCK
LABC1 ISZ LABCK
JMP LABCK,I
CHECK FOR OVERFLOW IN DATA TABLE

DATFL

NOP
LDA ZDATA
CMA,INA
ADA YDATA
SSA,RSS
JMP DATFL1
LDA .30
LDB *+2

JMP TOLOV
 TABLE OVERFLOW

DEF **1
ASC 15, OVERFLOW IN PROGRAM DATA TABLE

DATFL1

ADA M10
SSA,RSS
JMP DATFL I
JSB NWLN I
LDA .40
LDB *+3
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 ASM8Y 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 ADDRESS IN ASSEMBLED CODE
STB LENTH LENGTH OF ASSEMBLY
STB MNMNC
STB OPADD OPERAND ADDRESS STORE
JMP CLEAR,I
* * SYMBOL TABLE LOOKUP
* * ENTER (B) = ADDRESS OF LABEL

* RETURN (A) > 0 ADDRESS OF LABEL IN PROGRAM
* (A) = 0 Label DOES NOT EXIST
* (A) < 0 Undefined Label

* (B) Symbol Table Address of Label

Lokup NOP
JSB FIND FIND LABEL IN SYMBOL TABLE
SZF,RSS LABEL EXISTS
JMP LOKUP,I NO, LABEL NOT IN TABLE
ADBC2 YES, GET INFO ON LABEL
LCR,ERA
INB

LDA B,I ADDRESS IN ASSEMBLED CODE
SEZ,RSS UNDEFINED REFERENCE
CMA,INA YES
ADBM3 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 (3) ADDRESS OF LABEL BUFFER

RETURN (9) SYMBOL TABLE ADDRESS OF LABEL
   (A) = 0 Label not in table
FIND1
NOP
STB ADDR3
STB ADDR2
CLB
STB LMTEG
STB TEMP4
LOA M3
STA TEMP3
CLB
ISZ ADDR3
LOA ADDR3,I
ADD TEMP4
STA TEMP4
ISZ TEMP3
JMP FIND1-1
DIV *125
BLS A
STB A
BLS
ADB A
ADB XSTBL
STB TEMP3
FIND2
LOA B,I
SZA
JMP FIND6
LABEL CELL EMPTY
NO, SOMETHING IN SYMBOL TABLE
* EITHER LABEL NOT IN TABLE OR LOCATION FREE TO STORE LABEL *

JMP FIND1

FIND2 LDA LMFG
SZA, RSS
JMP FIND5
LDA TEMP3
CMA, INA
ADA B
SSA, RSS
JMP FNDER YES

FIND5 LDA YSTBL
OMA, INA
ADA B
SSA
JMP *+3 TABLE BOUND EXCEEDED
ADB M750 YES, SEARCH BEGINNING OF TABLE
STB LMFG SEARCH OTHER SIDE OF TABLE
JSB FIND1
JMP FIND2

FIND6 JSB FIND1
STB TEMP4 RETAIN SYM TBL ADDR
CLE
JMP *+4

FIND7 INB
ISZ ADDR3
LDA B,I
CPA ADDR3,I MATCH

--- BEGINNING OF TABLE --
--- TABLE OVERFLOW ---
--- TABLE UNDERFLOW ---

--- TABLE OVERFLOW ---
--- TABLE UNDERFLOW ---
FIND8
JMP *+2
YES
FIND9
JMP FIND9
NO
SEZ,RSS
JMP FIND7
INB
ADVANCED ADDRESS
ISZ ADDR3
LOA B,I
AND CH1
MATCH
CPA ADDR3,I
MATCH
CCA,RSS
SET (A) NE 0
JMP FIND9
LDB TEMP4
RETRIEVE LABEL ADDR
JMP FIND4
LDB TEMP4
GET PREVIOUS ADDRESS
JMP FIND4
CHECK NEXT ENTRY

FINDER 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**

<table>
<thead>
<tr>
<th>MNEM</th>
<th>NOP</th>
<th>CLE</th>
<th>LDA XOPCD</th>
<th>ADDR OF OP CLOE TABLE</th>
<th>STA LWRBD</th>
<th>LOWER BOUND IN SEARCH</th>
<th>LDB .86</th>
<th>ADB</th>
<th>RECB</th>
<th>STB UPRBD</th>
<th>UPERBOUND</th>
<th>ADB LWRBD</th>
<th>MJP MNEM1</th>
<th>LDA LWRBD</th>
<th>CMA</th>
<th>COMPLEMENT LOWER BOUND</th>
<th>ADB UPRBD</th>
<th>CLE, SZA, RSS</th>
<th>CHECK FOR CONVERGENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CLE</td>
<td>LDA XOPCD</td>
<td>ADDR OF OP CLOE TABLE</td>
<td>STA LWRBD</td>
<td>LOWER BOUND IN SEARCH</td>
<td>LDB .86</td>
<td>ADB</td>
<td>RECB</td>
<td>STB UPRBD</td>
<td>UPERBOUND</td>
<td>ADB LWRBD</td>
<td>MJP MNEM1</td>
<td>LDA LWRBD</td>
<td>CMA</td>
<td>COMPLEMENT LOWER BOUND</td>
<td>ADB UPRBD</td>
<td>CLE, SZA, RSS</td>
<td>CHECK FOR CONVERGENCE</td>
</tr>
<tr>
<td>MNEM1</td>
<td>BRS</td>
<td></td>
<td>LDA B,1</td>
<td>FIRST TWO CHARACTERS MATCH</td>
<td>JMP MNEM3</td>
<td>YES</td>
<td>SEZ</td>
<td>NO</td>
<td>JMP MNER</td>
<td>MNEMONIC NOT IN TABLE</td>
<td>CMA, INA</td>
<td>HALVE INTERVAL</td>
<td>ADA MNC</td>
<td>JMP MNEM2</td>
<td>STB LWRBD</td>
<td>SET NEW LOWER BOUND</td>
<td>ADB UPRBD</td>
<td>JMP MNEM2</td>
<td>SET NEW LOWER BOUND</td>
</tr>
<tr>
<td>MNEM2</td>
<td></td>
<td></td>
<td>JMP MNEM2</td>
<td>STB LWRBD</td>
<td>ADB UPRBD</td>
<td>JMP MNEM2</td>
<td>SET NEW LOWER BOUND</td>
<td>STB UPRBD</td>
<td>ADB LWRBD</td>
<td>JMP MNEM2</td>
<td>SET NEW UPPER BOUND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MNEM3</td>
<td>ADD H1</td>
<td></td>
<td>ADD H1</td>
<td>BACK UP FOR SEVERAL MNEMONICS</td>
<td>LDA B,1</td>
<td>BEGIN WITH THE SAME TWO LETTERS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

336
CPA MNC
JMP *-3
ADD .86
LOA M6
STA TEMP3

MNE4
INB
LDA B,1
AND CH1
CPA MNC+1
JMP MNEM5
ISZ TEMP3
JMP MNEM4
LOA AND 9,ICH1 TEST FOR
MASS
OUT THIP.O FIRST
CHARACTER POSITION
CHARACTER MATCH
YES
NO
LOOK AT NEXT OPCODE
OPCODE NOT FOUND

MNE5
ADD M86
LDA B,1
CPA MNC
RSS
JMP MNER
ERR
ADD .86
LOA B,1
AND B177
GET INSTRUCTION NUMBER
STA INSNM
RETN
ADD M8
SSA
CLA,INA,RSS
MACHINE INSTRUCTION
CCA
DATA
STA ASMFG
ADD .86
LOA B,1
STA ASM6Y
SKELETON OF ASSEMBLED INSTRUCTION
JMP MNEM,I

* INSTRUCTION NOT FOUND

MNER
LDA .30
LDB *+2
JMP EPICAL
DEF *+1
ASC 15,ILLEGAL ASSEMBLER INSTRUCTION
ORG 60009

* 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
CGB, 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,1
FETCH NUMBER AND CONVERT TO BINARY
RETURN P+1 VALID DATA RETURNED IN (A) AND (B)

NUMCK NOP
CL6
STB EXP
STB MANT1
STB MANT2
STB EXPON
STB TEMP3
STB DPF LG
STB DPF LG
STB EFLG
STB EFLG
CPA PRIOD
ISZ DPF LG
JMP NUMC2
CLA
STA EXPON
JMP NUMC3+1
JMP NUMC7
ISZ EXPON
ALF, ALF
ALF, RAR
STA TEMP4
JSB MBY10
LDB EXP
SZB
JMP NUMC4
LDA .4
STA EXP
LDA TEMP4
LOAD NUMBER
<table>
<thead>
<tr>
<th>NUMC3</th>
<th>JSB</th>
<th>NORMALIZE THE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSB</td>
<td>ISZ</td>
<td>TEMP3</td>
</tr>
<tr>
<td>JSB</td>
<td>GETCR</td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>NUM12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NUMC4</th>
<th>JMP</th>
<th>NUMC1</th>
<th>NEW CHARACTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADB</td>
<td>CM3</td>
<td>LDA</td>
<td>TEMP4</td>
</tr>
<tr>
<td>STB</td>
<td>TEMP4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NUMC5</th>
<th>ISZ</th>
<th>TEMP4</th>
<th>DIGIT POSITIONED</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>NUMC6</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NUMC6</th>
<th>CLE</th>
<th>ERA</th>
<th>YES ADD IN LOW PART OF NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADB</td>
<td>MANT2</td>
<td>CLO</td>
<td></td>
</tr>
<tr>
<td>SEZ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INA</td>
<td>ADDA</td>
<td>MANT1</td>
<td>YES ADVANCE A</td>
</tr>
<tr>
<td>SOS</td>
<td>OVERFLOW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>NUCM3</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NUMC6</th>
<th>CLE</th>
<th>ERA</th>
<th>YES ROTATE DOWN AND BUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERB</td>
<td>EXP</td>
<td>ISZ</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NUMC7</th>
<th>JMP</th>
<th>NUMC5</th>
<th>SHIFT DIGIT RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB</td>
<td>TEMP4</td>
<td>STB</td>
<td></td>
</tr>
<tr>
<td>CPB</td>
<td>TEMP3</td>
<td>OR DIGIT FOUND</td>
<td></td>
</tr>
</tbody>
</table>
JMP NUMR3  
CPA E  
ISZ EFLG  
JMP NUM12  
JSB GETCR  
JMP NUMR4  
CPA PLUS  
JMP NUMC8  
CPA NUNS  
CCA, RSS  
JMP NUMC9  
STA TEMP4  
STA TEMP3  

NUMC8 JS3 GETCR  
RSS  
NUMC9 JS3 DECHK  
DIGIT  
RSS  
JMP NUMCA  
CLA  
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  
LDH TEMP3  
YES  
BLS, BLSAOB  
ADD TEMP3  
MULTIPLY PRIOR DIGIT BY 10  
RLS  
AODA B  
ADD NEW DIGIT  
STA TEMP3  
SAVE EXPONENT  
JSB GETCR
JMP NUM10  THIRD DIGIT
JSB DECK
RSS
JMP NUM4  EXPONENT TOO LONG
NUM10 LDA TEMP3  RETRIEVE EXPONENT
ISZ TEMP4  POSITIVE
CMA, INA  YES, COMPLEMENT IT
RSS
NUM12 CLA
ISZ DPFLG  DECIMAL POINT
ADA EXPON  YES, CORRECT EXPONENT
SZA, RSS  ZERO EXPONENT
JMP NUM14  YES
SSA
JMP NUM13  NO, NEGATIVE EXPONENT
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
LOB MANT2  NUMBER
ISZ SIGN  POSITIVE
JMP NUM15  YES
CMA
CMA, INB, SZB, RSS
INA
NUM15 JSB PACK  PACK NUMBER INTO (A) AND (B)
STA TEMP1
STB TEMP2
JSB BCKSP  RETURN TERMINATOR TO BUFFER
LOA TEMP1  RESTORE (A)
JMP NUM10
NORMALIZE AND PACK FLOATING POINT NUMBER

PACK NOP
JSB NORML
CLE, SZA, RSS
JMP .PACK1
ADD B177
SSA, RSS
INB
CLO
SEZ
CLE, INA
SOS
RAL
SSA, SLA, RSS
JMP .PACK1
CCE
ARS, SLA, ALS
PACK1
RAR
STA MBY10
LOA 1

MANTISSA IN (A) AND (B)
EXPONENT IN EXP, (E) CLEARED
ZERO RESULT
YES
NO, ROUND
POSITIVE NUMBER
YES, FINISH ROUND
OVERFLOW FROM (B)
YES, ADVANCE (A)
OVERFLOW (A) = 100000, (B) = 0
TWO HIGH BITS 1, (A) = 140000
NO
YES
SET (A) = 100000 AND SKIP
COUNTERPART TO *-5
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**

<table>
<thead>
<tr>
<th>Instruction(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>STA MBY10</td>
<td>SET LEFT-SHIFT</td>
</tr>
<tr>
<td>STA MPY</td>
<td>COUNTER TO ZERO</td>
</tr>
<tr>
<td>LDA MBY10</td>
<td></td>
</tr>
<tr>
<td>SZA, RSS</td>
<td>ON ZERO CLEAR EVERYTHING</td>
</tr>
<tr>
<td>SZZ</td>
<td></td>
</tr>
<tr>
<td>JMP NORM3</td>
<td></td>
</tr>
<tr>
<td>STA EXP</td>
<td></td>
</tr>
<tr>
<td>STA MANT1</td>
<td>STORE MANTISSA AND RETURN</td>
</tr>
<tr>
<td>STA MANT2</td>
<td></td>
</tr>
<tr>
<td>JMP NORML, I</td>
<td></td>
</tr>
<tr>
<td>NORM2 ISZ MPY</td>
<td>COUNT LEFT SHIFTS</td>
</tr>
<tr>
<td>ELA</td>
<td></td>
</tr>
<tr>
<td>NORM3 CLE, ELB</td>
<td>ROTATE (A) AND (B) LEFT INTO (E)</td>
</tr>
<tr>
<td>SEZ, SSA, RSS</td>
<td>TWO HIGHEST BITS 0</td>
</tr>
<tr>
<td>JMP NORM2</td>
<td>YES, + UNNORMALIZED</td>
</tr>
<tr>
<td>SEZ, SSA</td>
<td>NO, TWO HIGHEST BITS 1</td>
</tr>
<tr>
<td>JMP NORM2</td>
<td>YES, - UNNORMALIZED</td>
</tr>
<tr>
<td>ERA</td>
<td></td>
</tr>
<tr>
<td>ERB, CLE</td>
<td>SHIFT TO NORMALIZE MANTISSA</td>
</tr>
<tr>
<td>STA MANT1</td>
<td>NO, COMPUTE CORRECTED EXPONENT</td>
</tr>
<tr>
<td>LDA MPY</td>
<td></td>
</tr>
<tr>
<td>LDA, ENA</td>
<td></td>
</tr>
<tr>
<td>ADA EXP</td>
<td></td>
</tr>
<tr>
<td>STA EXP</td>
<td></td>
</tr>
<tr>
<td>LDA MANT1</td>
<td></td>
</tr>
<tr>
<td>JMP NORM1</td>
<td></td>
</tr>
</tbody>
</table>
**MULTIPLY UNPACKED NUMBER BY 10**

M8Y10 NOP
LDA, MANT1
S2A, RSS
JMP MBY10, I
LD9 EXP
AD9 *3
SIB EXP
LDB MANT2
CLE, ERA
ER9
CLE, ERA
ERB, CLE
AD9 MANT2
SEZ
INA
ADA MANT1
SSA, RSS
JMP *+5
CLE, ERA
ER9
ISZ EXP
NOP
STA MANT1
SIB MANT2
JMP MBY10, I
return on zero mantissa
multiply by 8
load mantissa
divide by 4
double add to produce 1.25 $ mantissa
correct on overflow
**DIVIDE UNPACKED NUMBER BY 10**

```
DBY10 NOP
LDA MANT1 MULTPLY BY DOUBLE-LENGTH TENTH
SZA, RSS RETURN ON ZERO MANTISSA
JMP DBY10,I
LDB M2
ADD EXP
STB EXP
LDA MANT2
CLE, ERA ADD EXPONENT OF TENTH TO MANTISSA EXPONENT
JSB MPY
DEF TENTH
CLE, ELA
CLE, CLE
ADA 8
SEZ
INB
STB MANT2
LDA MANT1 DO SAME FOR HIGH MANTISSA
JSB MPY
DEF TENTH
CLE
ADA 3
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                  SET -2 IN SIGN TEMP
     STB MBY10               LOAD MULTIPLIER
     LDB MPY,I               (A) NEGATIVE
     LDB B,I                 YES, COMPLEMENT (A) AND (E)
     CLE,SSA
     CMA,CME,INA             (B) NEGATIVE
     SSB
     CMZ,CME,INB             YES, COMPLEMENT (B) AND (E)
     SEZ
     ISZ MBY10               (E) = 0
     STB NORML               NO, SET SIGN OF RESULT NEGATIVE
     STB M16                 SAVE MULTIPLIER
     STB TEMP1               SET COUNTER
     CLB                     ZERO PRODUCT
     ELA                     BIAS A TO LEFT
     MPY1
     ERA,CLE,SLA             SHIFT, TEST, AND ADD UPON NON-ZERO BIT
     ADB NORML
     ERB
     ISZ TEMP1               DONE
     JMP MPY1                NO
     ERA,CLE                 YES, ADJUST FINAL RESULT
     ISZ MBY10               NEGATIVE RESULT
     JMP MPY2                NO
     CMZ
     CMA,INA,SZA,RSS         YES, COMPLEMENT RESULT
     INB
     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 (9)  
JSB SAVEE SAVE (E) REGISTER  
LDB 072  
ADB A  
SSB,RSS ASCII 72B OR GREATER  
JMP ++6 YES RETURN WITH CHARACTER  
ADB +10 NO, ASCII 60B OR GREATER  
SSB  
JMP +3  
LDA 8 COPY DIGIT INTO (A)  
ISZ DECHK  
JSB RSTRE RESTORE (E)  
LDB STORE RESTORE (B)  
JMP DECHK,1  

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 TYPCK1    NO DECIMAL POINT
LDA TEMP
JMP TYPCK1,I  RESTORE (A)
JMP TYPCK1    RETURN WITH REAL NUMBER

TYPCK1 LDA TEMP
ISZ EFLG
JMP TYPCK1,I  REAL NUMBER
JSB IFIX
ISZ TYPCK
JMP TYPCK1,I  CONVERT TO INTEGER
INTEGERIZE FLOATING POINT NUMBER

ENTER NUMBER IN (A) AND (B)

RETURN P+1 INTEGER IN (A)

IFIX
NOP
STA MPY
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, SZ3 SET (E) = 0 IF (B) = 0
CME
STA B SAVE SHIFT COUNT IN (B)

IFX1
LDA MPY
ISZ B ANY MORE SHIFTS
JMP IFX2 YES
SEZ, SSA IF NUM3 LT 0 AND FRACT NOT 0
IN3 BUMP RESULT
JMP IFIX, I
IFX2
SLA, AR3 SHIFT RIGHT AND TEST BIT LOST
CCE
JMP IFX1

UNPACK LOW WORD OF NUMBER

FLUN
NOP
LDA B WORD IN (B)
AND B377 EXTRACT EXPONENT IN (A)
CMB SUBTRACT OFF EXPONENT FROM
ADA 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 GNUM GET FIRST INTEGER
STA NUM1 STORE VALUE
JSB TRMCK CHECK TERMINATOR
JMP TWINT, I FIRST RETURN 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 GNUM READ INTEGER
STA NUM2
STA NUM2
JSB TRMCK
JMP TWIN2 THIRD RETURN CONDITION
JSB BCKSP RETURN CHARACTER TO BUFFER
ISZ TWINT
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)

STNUM
NOP
JSB CONST INPUT A CONSTANT
SSA NEGATIVE NUMBER
JMP NUR6 YES
JSB TYPCK NO, REAL OR INTEGER
JMP NUR6 REAL
JMP GTNUM,1

READ IN OCTAL INTEGER
RETURN (A) OCTAL INTEGER

OCTIN
NOP
JSB NTBLK NEXT NON BLANK CHARACTER
JMP NUR1 NO DATA FOUND
CLE
STB NUM1 INITIALIZE
INB
STB SIGN SET SIGN POSITIVE
* * 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
LDB 070

ADB A CHARACTE IN (A)
SSB, RSS CHARACTER 703 OR GREATER
JMP *+6 YES, RETURN WITH CHARACTER

ADB *+8 NO, ASCII 603 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)

<table>
<thead>
<tr>
<th>NUMBR</th>
<th>NOP</th>
<th>JSB</th>
<th>NEXT NON BLANK CHAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JSB</td>
<td>NIBLK</td>
<td>NEXT NON BLANK CHAR</td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>NUM?1</td>
<td>NO DATA FOUND</td>
</tr>
<tr>
<td></td>
<td>CLB</td>
<td>INB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STB</td>
<td>SIGN</td>
<td>SET SIGN POSITIVE</td>
</tr>
<tr>
<td></td>
<td>CPA</td>
<td>PLUS</td>
<td>POSITIVE SIGN</td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>NUM?1</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>CPA</td>
<td>MINUS</td>
<td>NO, NEGATIVE SIGN</td>
</tr>
<tr>
<td></td>
<td>CCB</td>
<td>RSS</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>NUM?2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STB</td>
<td>SIGN</td>
<td>RECORD SIGN</td>
</tr>
<tr>
<td>NUMB1</td>
<td>JSB</td>
<td>GETCR</td>
<td>SOLITARY SIGN</td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>NUMR?2</td>
<td></td>
</tr>
<tr>
<td>NUMB2</td>
<td>JSB</td>
<td>DECK</td>
<td>DECIMAL DIGIT</td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>NUMR?1</td>
<td>FIRST RETURN</td>
</tr>
<tr>
<td></td>
<td>CLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STB</td>
<td>NUM?1</td>
<td>DECIMAL</td>
</tr>
<tr>
<td></td>
<td>STB</td>
<td>NUM?2</td>
<td>OCTAL</td>
</tr>
<tr>
<td></td>
<td>STB</td>
<td>TEMP4</td>
<td>OCTAL ERROR FLAG</td>
</tr>
<tr>
<td>NUMB3</td>
<td>AOB</td>
<td>A</td>
<td>DECIMAL OVERFLOW</td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>NUMR?5</td>
<td>YES</td>
</tr>
<tr>
<td>AOB</td>
<td>NUM?1</td>
<td>NO, ADD IN DIGIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STB</td>
<td>NUM?1</td>
<td>CHECK FOR OCTAL DIGIT</td>
</tr>
<tr>
<td></td>
<td>ADB</td>
<td>M?8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADB</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSB</td>
<td>RSS</td>
<td>OCTAL DIGIT</td>
</tr>
<tr>
<td></td>
<td>ISZ</td>
<td>TEMP4</td>
<td>NO, RECORD ERROR</td>
</tr>
</tbody>
</table>
ADA NUM2
STA NUM2
SQC
JMP NUMR5
JS3 GETCR
JMP NUMB6 END OF LINE TERMINATION
JS3 DECK 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 MULTIPY DECIMAL BY 10 USING
STB NUM2
LDB NUM1
BLS BLS
AND NUM1
STB NUM1
JMP NUMB3
NUMB4 CPA 9E OCTAL FLAG
RSS YES
JMP NUMB6 NO
LOA TEMP4
SZA
JMP NUMR5 OCTAL ERROR
LOA NUM2 YES
JMP NUM2 RETURN OCTAL
NUMB6 JSB BCKSP RETURN TERMINATOR TO BUFFER
LOA NUM1 RETURN DECIMAL
NUMB7 LDB SIGN NEGATIVE SIGN
SSB CM4,INA YES, COMPLEMENT
ISZ NUMBR
JMP NUMBR1
**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 OVERFLOW
*

NUMR6 LDA *+2
  LDB *+2
  JMP NUMER

* DEF *+1
  ASC 13, POSITIVE INTEGER EXPECTED
*

NUMR7 LDA *+2
  LDB *+2
  JMP NUMER

* DEF *+1
  ASC 12, BAD DATA FOLLOWS INTEGER
*

NUMR8 LDA *+2
  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 GRIER,I
  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 (*)

<table>
<thead>
<tr>
<th>OPREC</th>
<th>NOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPRC1</td>
<td>JSB NUMBR READ IN INTEGER</td>
</tr>
<tr>
<td></td>
<td>JMP OPRC4 FIRST CHAR NOT A NUMBER</td>
</tr>
<tr>
<td></td>
<td>CCB</td>
</tr>
<tr>
<td></td>
<td>STB NUMFG SET OPERAND NUMBER FLAG</td>
</tr>
<tr>
<td></td>
<td>STA OPRNUM STORE VALUE</td>
</tr>
<tr>
<td>OPRC2</td>
<td>JSB TRMCK TERMINATION</td>
</tr>
<tr>
<td></td>
<td>JMP OPRC8 YES, CHECK FOR LABEL</td>
</tr>
<tr>
<td>CPA</td>
<td>PLUS NO, POSITIVE SIGN</td>
</tr>
<tr>
<td></td>
<td>JMP OPRC3 YES, SET SIGN</td>
</tr>
<tr>
<td>CPA</td>
<td>MINUS NO, MINUS SIGN</td>
</tr>
<tr>
<td>RSS</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>JMP OPRC7</td>
</tr>
<tr>
<td></td>
<td>CCB, RSS</td>
</tr>
<tr>
<td>OPRC3</td>
<td>CLB, INB</td>
</tr>
<tr>
<td></td>
<td>STB SIGN SET SIGN</td>
</tr>
<tr>
<td>LDB OPRHL LABEL FLAG</td>
<td></td>
</tr>
<tr>
<td>SZB, RSS SET</td>
<td></td>
</tr>
<tr>
<td>JMP OPRC5 NO, READ LABEL</td>
<td></td>
</tr>
<tr>
<td>LDB NUMFG</td>
<td></td>
</tr>
<tr>
<td>SZB</td>
<td></td>
</tr>
<tr>
<td>JMP OPR1 ERROR IN OPERAND</td>
<td></td>
</tr>
<tr>
<td>JSB BCKSP RETURN SIGN TO BUFFER</td>
<td></td>
</tr>
<tr>
<td>JMP OPRC1 READ INTEGER</td>
<td></td>
</tr>
<tr>
<td>OPRC4 JSB LETPR</td>
<td>LETTER OR PERIOD</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>JMP OPRC6</td>
<td>NO, SPECIAL CHARACTER</td>
</tr>
<tr>
<td>RSS</td>
<td>YES, VALID CHARACTER</td>
</tr>
<tr>
<td>OPRC5 LDA M1</td>
<td>NO CHARACTER PREVIOUSLY READ</td>
</tr>
<tr>
<td>LDB SIGN</td>
<td>SIGN BEFORE LABEL</td>
</tr>
<tr>
<td>SSB</td>
<td>NEGATIVE SIGN</td>
</tr>
<tr>
<td>JMP OPER2</td>
<td>YES, ERROR</td>
</tr>
<tr>
<td>LDB LAB2</td>
<td>STORE ADDRESS FOR LABEL</td>
</tr>
<tr>
<td>JSB LABRD</td>
<td></td>
</tr>
<tr>
<td>JMP OPRC6</td>
<td>ILLEGAL CHARACTER BEGINS LABEL</td>
</tr>
<tr>
<td>GLB, INB</td>
<td></td>
</tr>
<tr>
<td>STB OPLBL</td>
<td>SET OPERAND LABEL FLAG</td>
</tr>
<tr>
<td>JMP OPRC2</td>
<td></td>
</tr>
<tr>
<td>OPRC6 LDB DMPFG</td>
<td>DUMP FLAG</td>
</tr>
<tr>
<td>SZB</td>
<td>DUMP OPERATION</td>
</tr>
<tr>
<td>JMP OPER1</td>
<td>YES, ERROR</td>
</tr>
<tr>
<td>CPA</td>
<td></td>
</tr>
<tr>
<td>STAR</td>
<td>NO, ASTERISK</td>
</tr>
<tr>
<td>CCA, RSS</td>
<td>YES</td>
</tr>
<tr>
<td>JMP, OPER1</td>
<td></td>
</tr>
<tr>
<td>LDB SIGN</td>
<td></td>
</tr>
<tr>
<td>SSB</td>
<td></td>
</tr>
<tr>
<td>JMP OPER3</td>
<td>MINUS SIGN PRECEDES ASTERISK</td>
</tr>
<tr>
<td>STA OPLBL</td>
<td>SET P.L.G. INDICATOR</td>
</tr>
<tr>
<td>OPRC7 LDB DMPFG</td>
<td></td>
</tr>
<tr>
<td>SZB</td>
<td></td>
</tr>
<tr>
<td>JMP OPER1</td>
<td></td>
</tr>
<tr>
<td>CPA COMMA</td>
<td>COMMA</td>
</tr>
<tr>
<td>RSS</td>
<td>YES</td>
</tr>
<tr>
<td>JMP OPER1</td>
<td>NO, ERROR</td>
</tr>
<tr>
<td>JSS GETCR</td>
<td></td>
</tr>
<tr>
<td>JMP OPER1</td>
<td>ILLEGAL CHAR IN OPERAND</td>
</tr>
<tr>
<td>CPA I</td>
<td>INDIRECT FLAG</td>
</tr>
<tr>
<td>CCA, RSS</td>
<td>YES</td>
</tr>
<tr>
<td>JMP OPER1</td>
<td>ILLEGAL CHAR IN OPERAND</td>
</tr>
<tr>
<td>STA IDCRT</td>
<td></td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSB TRMCK</td>
<td>END OF OPERAND</td>
</tr>
<tr>
<td>JMP OPER1</td>
<td>ILLEGAL TERMINATION</td>
</tr>
<tr>
<td>LDA IDRCT</td>
<td>INDIRECT FLAG SET</td>
</tr>
<tr>
<td>SZA OPRC8</td>
<td></td>
</tr>
<tr>
<td>JMP OPR10</td>
<td>YES</td>
</tr>
<tr>
<td>LDA OPLBL</td>
<td>CHECK FOR LABEL</td>
</tr>
<tr>
<td>SZA</td>
<td>LABEL FOUND</td>
</tr>
<tr>
<td>JMP OPRC9</td>
<td>YES, RETURN</td>
</tr>
<tr>
<td>LDA INSNM</td>
<td>NO LABEL FOUND</td>
</tr>
<tr>
<td>CPA ZERO</td>
<td></td>
</tr>
<tr>
<td>JMP OPRC9</td>
<td></td>
</tr>
<tr>
<td>LDA M8</td>
<td>MEMORY REFERENCE TYPE INSTRUCTION</td>
</tr>
<tr>
<td>SSA RSS</td>
<td></td>
</tr>
<tr>
<td>JMP OPRC9</td>
<td>NO</td>
</tr>
<tr>
<td>LDA OPNUM</td>
<td>YES, CHECK RANGE</td>
</tr>
<tr>
<td>SZA</td>
<td></td>
</tr>
<tr>
<td>JMP OPRC9</td>
<td>NEGATIVE</td>
</tr>
<tr>
<td>LDA 0100</td>
<td></td>
</tr>
<tr>
<td>SZA</td>
<td></td>
</tr>
<tr>
<td>JMP OPRC9</td>
<td>RETURN VALUE IN RANGE</td>
</tr>
<tr>
<td>OPRC9</td>
<td></td>
</tr>
<tr>
<td>LDA 26</td>
<td></td>
</tr>
<tr>
<td>LDB *+2</td>
<td></td>
</tr>
<tr>
<td>JMP ERCAL</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
</tr>
<tr>
<td>DEF ERR3</td>
<td>OPERAND VALUE OUT OF RANGE</td>
</tr>
<tr>
<td>OPR10</td>
<td></td>
</tr>
<tr>
<td>LDB INSNM</td>
<td>INSTRUCTION NUMBER</td>
</tr>
<tr>
<td>ADD M8</td>
<td>MEMORY REFERENCE</td>
</tr>
<tr>
<td>SSB</td>
<td></td>
</tr>
<tr>
<td>JMP OPRC8+3</td>
<td>YES</td>
</tr>
<tr>
<td>LDA .38</td>
<td>NO, INDIRECT REFERENCE NOT ALLOWED</td>
</tr>
<tr>
<td>LDB OPRM1</td>
<td></td>
</tr>
<tr>
<td>JSB BPLN</td>
<td></td>
</tr>
</tbody>
</table>
```
LOA .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*

<table>
<thead>
<tr>
<th>LABRO</th>
<th>NOP</th>
<th>STB ADDR</th>
<th>SAVE ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDB M5</td>
<td>STB TEMP3</td>
<td>CLE,SSA,RSS</td>
<td>CHARACTER COUNT</td>
</tr>
<tr>
<td>JMP LABR1</td>
<td>JSB NTBLK</td>
<td>FIRST NON BLANK CHARACTER</td>
<td></td>
</tr>
<tr>
<td>JMP LBER1</td>
<td>JMP LBER1</td>
<td>RETURN NO LABEL FOUND</td>
<td></td>
</tr>
<tr>
<td>JSB LETPR</td>
<td>JMP LABR2+1</td>
<td>LETTER-PERIOD CHECK</td>
<td></td>
</tr>
<tr>
<td>JMP LABR2+1</td>
<td>SEZ,RS</td>
<td>BAD CHARACTER BEGINNING LABEL</td>
<td></td>
</tr>
<tr>
<td>SEP</td>
<td>LABR1</td>
<td>ALF,ALF</td>
<td>SHIFT CHARACTER</td>
</tr>
<tr>
<td>IOR ADDR,1</td>
<td>STA ADDR,1</td>
<td>STORE CHARACTER</td>
<td></td>
</tr>
<tr>
<td>SEZ,CM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ISZ ADDR  ADVANCE BUFFER POINTER
ISZ TEMP3
RSS
JMP LABR2  FIVE CHARACTERS READ
JSB GETCR  NEXT CHARACTER
JMP LABR3
JSB LETPR  LETTER-PERIOD
RSS
JMP LABR1-1  YES, STORE CHARACTER
JSB DFCHK  DECIMAL NUMBER
JMP LABR3  NO
ADA .48  YES, BUT RETAIN AS CHARACTER
JMP LABR1-1
LABR2 ISZ LABRD
JMP LABRD,1
LABR3 JSB BCKSP  RETURN TERMINATOR TO BUFFER
JMP LABR2
* *
LBER1 LDA .14
LDB .42
JMP ERGAL
* DEF ERR9  NO LABEL FOUND
**CHECK FOR LETTER OR PERIOD**

*RETURN P+1 CHARACTER IN (A)*
*P+2 LETTER OR PERIOD IN (A)*

<table>
<thead>
<tr>
<th>LETPR</th>
<th>NOP</th>
<th>JSB SAVEE</th>
<th>SAVE (E) REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPA</td>
<td>PERIOD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>*+7</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>A</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>ADB</td>
<td>D133</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSB,RSS</td>
<td>ASCII 133B OR GREATER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>*+4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADB</td>
<td>.26</td>
<td>NO ASCII 101B</td>
</tr>
<tr>
<td></td>
<td>SSB,RSS</td>
<td>OR GREATER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JSB</td>
<td>LETPR</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>RESTORE</td>
<td>RESTORE (E) REGISTER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>LETPR,1</td>
<td></td>
</tr>
</tbody>
</table>
* * * * * 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 DTRG1 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 0100
ADB A
SSB
JMP DATRG,I

DTER1 LDA -30
LDB *,+2
JMP ERGL

* 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

CDSN1

LDA TEMP1
SSA, RSS  BIT 15 SET
JMP CDSN2  NO
SZA
JMP CDSN4  YES
LDA TEMP1
AND 82000  YES, I/O INSTRUCTION
SZA, RSS
JMP CDSN4  REGISTER REFERENCE
ISZ TEMP, I Extended arithmetic memory reference
LDA TEMP, I Examine address
SSA SSA Indirect bit set
JMP CDSN4 Yes, defined reference
ADA D100
CLE
JMP CDSN3

*CDSN2 AND B0700 Memory reference
SZA, RSS No
JMP CDSN4 Yes, retrieve instruction
LDA TEMP, I Current page bit set
AND B2000

SZA JMP CDSN4
LDA TEMP, I
AND B1777
ADA D100
CCE

GDSN3 SSA Forward reference
JMP CDSN4 No
CL A, SEZ, RSS Yes

GDSN3
ADA TEMP
STA TEMP1 Address of forward ref
LDA TEMP1, I Get instruction
STA TEMP2, I Save instruction
LDA MPPFX Force printing of warning message
STA TEMP1, I During execution

ISZ TEMP2
ISZ TEMP3
RSS
JMP CDSN4, I First 99 fwd ref saved
GDSN4 ISZ TEMP
LDA TEMP
CPA ZUSR P End of user program
JMP CDSN4, 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                    NOP
JSB RDCOM              RDCOM
RSS                    RSS
JSB TWINT              TWINT
NOP                    NOP
RSS                    RSS
RSS                    BAD DATA INPUT
JMP SQER1              BAD DATA INPUT
LDB M1001              TWO POSITIVE INTEGERS READ IN
AD3 NUM1               CHECK RANGE OF FIRST
SSB, RSS               NUM1 IN RANGE
JMP SQER2              TOO LARGE
LDB NUM2               TOO LARGE
SZB, RSS               NUM2 IN RANGE
JMP SQER2             -too LARGE
AD3 M26                NOT IN RANGE
SSB, RSS               NOT IN RANGE
JMP SQER2              TOO LARGE
LDA NUM1               BOTH NUMBERS IN RANGE
STA FSIMT              FIRST STATEMENT NUMBER
LDB NUM2               FIRST STATEMENT NUMBER
STB STING              STATEMENT NUMBER INCREMENT
GMB, IN3               STATEMENT NUMBER INCREMENT
4DA 3                  STATEMENT NUMBER INCREMENT
STA CUSTN              STATEMENT NUMBER INCREMENT
ISZ SQNCE              STATEMENT NUMBER INCREMENT
JMP SQNCE, I
**SQER1**
LDA .14
LDB *+2
JMP SQER3

**DEF ERR1**
BAD DATA INPUT

**SQER2**
LDA .30
LDB *+3

**SQER3**
JSB ERROR
JMP SNGE,1
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 SETCD1,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 SETCD1,I

STORE ASSEMBLED MACHINE INSTRUCTIONS

STCD1 LDB ZUSRN 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
STC02 LDA ASM3Y  NO, GET ASSEMBLED CODE
JSB STC09  STORE CODE
JMP SETCD,1

* CLEAR UP OPERAND FOR MEMORY REFERENCE INSTRUCTIONS

STC03 LDA OPLBL  OPERAND LABEL PRESENT
SZA  YES
JMP STC04
JSB DETLN  NO, GET INSTR SKELETON
ADA OPNUM  ADD OPERAND IF PRESENT
JSB IDIRI  CHECK FOR INDIRECT FLAG
JMP STCD2+1  STORE ASSEMBLED INSTRUCTION

* LABEL OR ASTERISK IS PRESENT

STC04 SSA,RSS  LABEL
JMP STC05
LDA ADDR1  ASTERISK, PLC REFERENCE
JSB IDIRI  ADDRESS IN SGB
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 STC08  NO
SSA  LABEL DEFINED
JMP STC09  NO
LDB OPNUM  OPERAND NUMBER
SZA
JMP TCD11  STORE OPERAND IN SSI
JSB DATAD  UPDATE FOR DATA ADDRESS
JSB IDIR
LDH ZUSR
ADB D340
STA B,I
STB ADDR3
JSP DTLN
JMP STCD6
SHP ONE WORD ASSEMBLY
AND B1777
ADA 3
ADA CPIB
JMP STCD2+1
STCD6
ADA ADDR3
IOR MNEG
JMP STCD2+1
* LABEL DOES NOT EXIST

STCD8
JSP STBL
STCD9
LDA OPMU
SZA
JMP TC011
LDH BCKSP
ADB 3
LDA IDRCT
SZA
INR
LDA B,I
STA ROCOM
STB BCKSP
JSR DTLN
STA ASMBY
LDA ZUSR
AND B1777
STA BCKSP,I
LDA ASMBY
ADA ROCOM
JMP STCD2+1
### LABEL WITH OPERAND NUMBER

<table>
<thead>
<tr>
<th>TCD11</th>
<th>CLA</th>
<th>VARIABLE TO CONTROL PRINTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA GECR</td>
<td>GETCR</td>
<td>OF WARNING MESSAGE</td>
</tr>
<tr>
<td>STA LINK</td>
<td>LINK FOR SST</td>
<td></td>
</tr>
<tr>
<td>LDA 9CKSP</td>
<td>SYM TBL ADDR OF LABEL</td>
<td></td>
</tr>
<tr>
<td>ADR .5</td>
<td>LINK TO SST</td>
<td></td>
</tr>
<tr>
<td>STB RDCOM</td>
<td>SAVE LINK CHARACTER</td>
<td></td>
</tr>
<tr>
<td>LOA B,I</td>
<td>PREVIOUS SST ENTRIES</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCD12</th>
<th>LDA XSST</th>
<th>ADDRESS OF SST</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADB .4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA YSST</td>
<td>UPPER BOUND OF SST</td>
<td></td>
</tr>
<tr>
<td>CMA, INA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADA 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCD13</th>
<th>JMP TCD13</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADA .32</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>LDB TCDR1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP TBLOV</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCD14</th>
<th>ADA .40</th>
<th>TABLE NEAR OVERFLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP TCD14</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>LOA GECR</td>
<td>YES, WARNING PREVIOUSLY PRINTED</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCD14</th>
<th>ADA .48</th>
<th>PRINT WARNING TO USER REGARDING</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA .48</td>
<td></td>
<td>TABLE OVERFLOW</td>
</tr>
<tr>
<td>LDB TCDR2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSB BPLN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA GECR</td>
<td>SET FLAG TO SUPPRESS MESSAGE</td>
<td></td>
</tr>
<tr>
<td>TCD14</td>
<td>LOA B,I</td>
<td>AREA OCCUPIED</td>
</tr>
</tbody>
</table>
SZA
JMP TCD12 YES
LOA 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
LOA OPNUM SET OPERAND VALUE IN SST
STA B,T
STB ADDR3 SAVE ADDRESS
LOA BCKSP ADDR OF SYM TBL ENTRY
JSB IDIR1 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
LOA ADDR3, I
STA GETCR VALUE OF LAST FORWARD REF
TCD15 JSB DETLN
STA ASM8Y
LOA ZUSR3
AND #1777 RELATIVE ADDRESS
STA ADDR3, I SET ADDRESS IN SST
LOA GETCR VALUE OF LAST FORWARD REFERENCE
ADA ASM8Y
JMP TCD2+1

* * PREVIOUS ENTRIES FOR THIS LABEL

* (A) CONTAINS LINK FROM SYMBOL TABLE

TCD16 LOB A,I VALUE IN SST
CPB OPNUM MATCH
RSS
JMP TCD18 NO
STA REENT SAVE ADDRESS
INA
LD3 A, I  GET WORD HOLDING INDIRECT FLAG
CLE, ELB  INDIRECT BIT FLAG IN (E)
LD3 IDRCT  INDIRECT FLAG ON OPERAND
BLS  CLEAR BIT 0

SEZ
SZ3, RSS
JMP TCD17
SSR, SLS, RSS  MATCH
JMP TCD18-1  NO MATCH

TCD17
ADA +1
LDB A, I  ADDR OF PREV REF
STA ADDR3  SAVE ADDRESS IN SST
STD GETCR  SAVE VALUE OF PREV REFERENCE
JMP TCD15

TCD18
LDA REENT
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
*
DFTLN NOP
  LDA ASMBY RETRIEVE INSTR SKELETON
  CLE,SSA,RSS TWO WORD ASSEMBLY
  JMP *+3 NO
  JSB STRCD YES, STORE WORD
  CLA,COM SET INDICATOR
  JMP DFTLN,I
*
* ALLOCATE STORAGE SPACE FOR STORING PROGRAM STATEMENT IN SOURCE CODE BLOCK
*
ASMBL NOP
  LDA SRCNT CHAR LENGTH OF INPUT STRING
  STA 3
  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
  ADA A
  STB LNH2 INPUT LENGTH FOR SCB
* SCAN FREE SPACE AREA FIRST BEFORE ALLOCATING NEXT AREA IN SCB

ASM31
LDA 3,1
SZA
JMP ASM4
YES
A03 .2
NO
LDA YFRSP
UPPER BOUND OF FREE SPACE AREA
CMA, INA
A03 3
SSA
OVERFLOW IN FREE SPACE AREA
JMP ASM31
NO

ASM32
LDA NEXT
YES, NEXT LOCATION IN SCB
LDB TEMP3
NUMBER OF WORDS IN SCB ENTRY
A03 A
STB NEXT
PREPARE FOR NEXT ENTRY
A03 M1
CHECK FOR TABLE OVERFLOW
CMB, INB
A03 YSCB
UPPER BOUND OF SOURCE CODE BLOCK
SSB, RSS
OVERFLOW
JMP ASM33
NO
LDA .30
YES
LDB ASMR1
JMP TBLOV

* ASM33 A03 M125
TABLE NEAR OVERFLOW
SSB, RSS
JMP ASM35
LDA .52
LDB ASMR2
JSB DPLN
JMP ASM35
```
* 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
    IN$, YES
    LDA B, I GET ADDRESS IN SCB
    RSS SKIP NEXT INSTR (E) MAY BE SET
    ASM35 CLE INHIBITS OPERATION OF FREE SPACE

    STA ADDR1 SAVE ADDR IN SC3
    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

* 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 (3) = 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 EXISTENT MEMORY
STA TEMP3 SAVE BUFFER ADDRESS
LDA LENGTH 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 RETRIEVE VALUE
STB A,I STORE VALUE AT APPROPRIATE ADDR
ISZ TEMP3 ADVANCE BUFFER POINTERS
ISZ RDCOM
ISZ TEMP4
JMP DIST1
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  SYMBOL TABLE NEARLY FULL
SSA
JMP STBL1  NO
LDA .42
LDB STBLR
JSB BPLN
STBL1  LDA .3  NUMBER OF WRODS TO BE MOVED
STA  SOURCE
LDA LAB1
SEZ
LDA LAB2  GET PROPER LABEL ADDRESS
JSB WMOVE  MOVE THE LABEL
LDA TEMP3  LABEL DEFINED
SSA,RSS
JMP STBL1,I  NO, RETURN SET NO FLAGS
STA TEMP3  YES
LOA 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 STBL1,I

* STBLR DEF *+1
ASC 21;SYMBOL TABLE NEARLY FULL, BEGIN EXECUTION
STORE INSTRUCTION IN PROGRAM AREA

ENTER (A) ASSEMBLED INSTRUCTION

STRCD NOP
STA ZUSRPI
ISZ ZUSRPI
JSB STRCK
JMP STRCD, I

CHECK USER PROGRAM AREA FOR OVERFLOW

STRCK NOP
LDB YUSRPI
GMB, INB
ADB ZUSRPI
SSB OVERFLOW
JMP STRCK, I
LDA .24 YES
LDB STRER
JMP TBOV

PROMPT USER IF PROGRAM AREA IS ABOUT TO OVERFLOW

STRCK1 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 ADDRESS 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
LD3 A, I AREA OCCUPIED:
SZB
JMP STPL1 YES
<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1777</td>
<td>AND</td>
<td>No save address</td>
</tr>
<tr>
<td>WMOVE</td>
<td>STA</td>
<td></td>
</tr>
<tr>
<td>HOLDA</td>
<td>LDA</td>
<td></td>
</tr>
<tr>
<td>ZPLC,I</td>
<td>STA</td>
<td></td>
</tr>
<tr>
<td>ZPLC</td>
<td>ISZ</td>
<td></td>
</tr>
<tr>
<td>OPNUM</td>
<td>LDA</td>
<td>Operand number</td>
</tr>
<tr>
<td>ZPLC,I</td>
<td>STA</td>
<td></td>
</tr>
<tr>
<td>STPL,C,I</td>
<td>JMP</td>
<td>Return number</td>
</tr>
<tr>
<td>YPLC</td>
<td>LDB, IN8</td>
<td>Upper bound of PLC are</td>
</tr>
<tr>
<td>ZPLC</td>
<td>ADB</td>
<td>Overflow</td>
</tr>
<tr>
<td></td>
<td>SSB</td>
<td>No</td>
</tr>
<tr>
<td>STP2</td>
<td>JMP</td>
<td>STP3</td>
</tr>
<tr>
<td></td>
<td>LOA, A$24</td>
<td>STP2+4</td>
</tr>
<tr>
<td></td>
<td>LDB, STPL+1+4</td>
<td>Table nearly full</td>
</tr>
<tr>
<td></td>
<td>ADB</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>SSB</td>
<td>Message already printed</td>
</tr>
<tr>
<td>STP3</td>
<td>JMP</td>
<td>STP1+4</td>
</tr>
<tr>
<td></td>
<td>LOA, A$42</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>LDB, STPL+1+4</td>
<td>SFT search flag</td>
</tr>
<tr>
<td></td>
<td>JMP</td>
<td>STP1+4</td>
</tr>
</tbody>
</table>

**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
LDA ZUSR
STA 8,1
STB ADDR2
JSB CMVE3
TSZ ADDR2
LDB ADDR2,1
CP 2.2
JMP CMVE2
LDA HOLD3
AND 80700
SZA,SS
JMP CMVE1
LDA HOLD3
AND 82000
SZA
JMP CMVE1
LDA HOLD3
AND 81777
SZA
JMP CMVE1
LDA HOLD3
SZA
JMP CMVE1
LDA HOLD3
SZA
```

- **ADB .4**: ASSEMB ADDR IN SCB
- **LDA ZUSR**: ADDR WHERE CODE WILL RESIDE
- **STA 8,1**: REDIFINE ASSEM ADDR IN SCB
- **STB ADDR2**: SAVE ADDRESS
- **JSB CMVE3**: ASSEMB LENGTH ADDR
- **TSZ ADDR2**: ASSEMB LENGTH
- **LDB ADDR2,1**: TWO WORD ASSEMBLY

CMVE1

- **LDA HOLD3**: CURRENT PAGE BIT
- **SZA**: CORRENT PAGE BIT SET, RETURN
- **JMP CMVE1**: RESTORE INSTRUCTION
- **SZA**: DEFINED, INDIRECT BIT SET
- **JMP CMVE1**: YES, RETURN
- **LDA HOLD3**: VALID REFERENCE TO BASE PAGE
- **AND 81777**: GET ADDRESS
- **SZA**:
JMP CMOVE, I YES REFERENCE TO (A) OR (B)
AOD 64 NO, INVALID REFERENCE TO BASE PAGE
STA HOLDB LDB 31500 CHECK FOR PLC REFERENCE
CMB, INB
AOD A
SSB, RSS UNDEFINED PLC REFERENCE
JMP CMOVE, I YES, RETURN
LDA HOLDA AND 31777 ADDRESS BEING SOUGHT
STA GECR NO, SAVE ADDRESS
LDA ZUSR P
AOD M1
AND 31777 ADDRESS TO BE INCLUDED DUE TO
STA RDCOM REPLACEMENT
JSB CASCO
JMP CMOVE, I

* TWO WORD ASSEMBLY
* CMVE2 ISZ HOLDA
JSB CMVE3
JMP CMVE1

* RETRIEVE ASSEMBLED INSTRUCTION
* CMVE3 NOP
LDA HOLDA, I RETRIEVE ASSEMBLED CODE
STA HOLDB
STA ZUSR P, I MOVE CODE INTO NEW LOCATION
CLA
STA HOLDA, I PLACE NOP IN VACATED AREA
ISZ ZUSR P
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  HOLD3
LDB  D701

ADD 'A'  POINTER TO SYMBOL TABLE
SS3, RSS
JMP  CSCD3  YES
JMP  JMP  CALCULATE ADDR OF NEXT REFERENCE
STA  ADDR2
LDA  ADDR2, I
AND  B1777  ADDRESS OF NEXT REFERENCE
CSCD2 STA  HOLD3
LDA  ADDR2, I
AND  B1777
JMP  CSCD1
NO
LDA  ADDR2, I
RETRIEVE INSTRUCTION
AND  B1760
SAVE INSTRUCTION SKELETON
ADA  RECOM
ADD IN NEW ADDRESS
STA  ADDR2, I
JMP  CSCD2
I

* EXAMINE SYMBOL TABLE FOR FORWARD REFERENCES

CSCD3 LDA  CSDFG
SZA
JMP  CSDDER  ERROR, CANNOT FIND FWD REF IN TBL
LDA  *125
CMA, INA
ADA  8
SSA
JMP  CSCD4  DIRECT REFERENCE IN SYM TBL
ADB  M125
LDA *125
CMA, INA
ADO R
SSA, RSS
JMP CSCD5

CSCD4
STA 1DRTC
INDIRECT REFERENCE IN SYM TBL
BLA B
STB A
MULTIPLY BY 5 FOR SYMBOL TABLE
BLA A
ADO XSTBL
BASE ADDRESS OF SYMBOL TABLE
ADO 3
LDA 1DRTC
INDIRECT REFERENCE

SZA
IND
JMP CSCD6
ADO *125
LDA *75
CMA, INA
ADO B
SSA, RSS

CSCD5
JMP CSDER
ADDRESS NOT IN SYMBOL TABLES
BLB, BLS
ADO XSTT
MULTIPLY BY 4
BASE ADDRESS OF SST
ADO 2

CSCD6
STB ADDR2
LDA R, I
STA CSDFG
JMP CSCD2

CSDER
LDA *34
LDB *+3
JSB BPN
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

DELETE NOP
CLE,ELB
STB 9ADDR
CLE,ELB
STB 9ADDR
ADDRTOKEN

FLAG TO DENOTE LEXICAL SCAN

OF CODE TO BE DELETED

JSB CLER,1
JSB LEXI,1

CLEAR LEX-EDIT FLAG

LOA LBLFG
SZA,RSS
JMP DLT1
LO3 LBLAD

LABEL FLAG FROM SOURCE CODE
LABEL PRESENT
NO
LABEL ADDR IN SYMBOL TABLE

ADDRESS OF LABEL INFORMATION

SAVE LAST CHARACTER IN LABEL

LOA 9,I
AND CH1
STA 9,I

SAVE SYMBOL TABLE ADDRESS
BASE ADDRESS OF SYMBOL TABLE

ADDRESS OF DELETED LABEL

RELATIVE POSITION OF LABEL
SYMBOL TBL POSITION POINTER
STA SAVR, I  INDICATING UNDEFINED LABELS
ADA ,125  STORE IN SYMBOL TABLE TO BE USED
ISZ SAVR
STA SAVR, I

**DELT1**

LDA ASMF0  ASSEMBLY FLAG
SSA, RSS
JMP DELT2  MACHINE INSTRUCTION
LDA SCBE1  SCR ADDR OF DATA TO BE DELETED
ADA ,5  ADDR OF LENGTH OF ASSEMBLY
LDB A,I  LENGTH OF ASSEMBLY
STB LENTH
JSB DETEED  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
S28, RSS
JMP DELTE, I  NO, RETURN DEFINED REFERENCE
SSB
JMP DELT3  PROGRAM LOCATION COUNTER REF
LDB LAB2  NO, DO A SYMBOL TABLE LOOK UP
JSB LOKP, I
SSA, RSS  DEFINED REFERENCE
JMP DELTE, I  YES, RETURN
LDA ASMEI  ADDR OF ASSEMBLED INSTRUCTION
LDB ASMBY  TWO WORD ASSEMBLY
SSB
INA YES
INA STA TEMP3
AND B1777 GET RELATIVE ADDRESS OF
STA GETCR UNDEFINED REFERENCE
LDA TEMP3,I GET ASSEMBLED INSTRUCTION
AND B1777 ADDR OF NEXT FORWARD REFERENCE
ADA 0100
SSA
JMP DELTE,I YES
ADA .64 NO
STA RDCOM SAVE ADDRESS TO BE MOVED
STA HOLDR
JSB CASCO 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
IN3
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

DTEOD NOP

LDA LENTH  LENGTH OF DATA TO BE DELETED
SZA,RSS   LENGTH ZERO
JMP DTEOD,I EQU PSEUDO OP, NO OPERATION NEEDED
CMA,INA  
STA TEMP7
LDA ASME1  AADDR OF FIRST WORD TO BE DELETED
ELA,CLERA 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 ITEMS TO BE MOVED
SZA,RSS  

394
JMP DTD02-3 NO DATA ITEMS TO BE MOVED

LD3 B,I GET ADDRESS
LDA A,I GET VALUE
ADA TEMP7 ADD DISPLACEMENT TO ADDR
STB A,I STORE VALUE IN NEW ADDR

STA HOLA,I STORE ADDR IN NEW POSITION
ISZ HOLA ADVANCE ADDR POINTERS
ISZ HOLD3
ISZ TEMP5
JMP DTD01

LDA ASHE2 PARAMETERS TO RESET SYMBOL TABLE
LDA TEMP7 PROGRAM ADDR AREA AND SC3 ADDR
JSB SCSYM
LDA TEMP7
STA TEMP5 CLA

DTDD2
LD3 HOLA,I CLEAR VACATED DATA AREA TO ZERO
STA B,I
STA HOLA,I
ISZ HOLDA
ISZ TEMP6
JMP DTD02
LDA ZDATA RESET NEXT FREE AREA IN DATA
ADA TEMP7 AREA AFTER DATA DELETE
STA ZDATA
JMP DTD00,I
**INSERT DATA**

**SHIFT DATA AND DATA ADDRESSES WHICH LOGICALLY FOLLOW**

**INSERT THEN STORE INSERTED DATA**

**NO INSERT INVOLVED WITH EQU PSEUDO OP FOR ENTRY WILL**

**BE SET IN SYMBOL TABLE**

**DTED1**

```
NOP
LDB LENTH
EQU PSEUDO OP LENGTH IS ZERO
```

**DTED2**

```
MPP M4
LDA 4,I
```

**DTED3**

```
LDB ZDATA
CPA ENEXT
CTERM
JMP DTED1
ADA *.4
LBA A,I
SSB,RSS
JMP DTED1
ELB,CLE,ERB
YES, CLEAR BIT 15
```

**DTED4**

```
LDA LENTH
ADA ZDATA
SHA ZDATA
STA ZDATA
JSB DTFL,I
LDB ZADD
JSB DTFL,I
```

**DTED5**

```
LDA ZDATA
STA HOLDA
CMAX,INA
ADA ASME2
```

**NEXT FREE AREA IN DATA TABLE**

**LOCATION OF FIRST INSERT**
STA TEMP3  -NUMBER OF WORDS TO BE MOVED
LDA HOLDA
STA HOLDA
LDB A,I
STB HOLD9
LDA B,I
A DB LENTH
STA B,I
LDA HOLDA
AAD LENTH
STA LENTH
STB A,I
ISZ TEMP3
JMP DTEI6
LDB SDBE2
ADD .1
LDB B,I
STB SDBE1
LDA LENTH
SCAN THROUGH SYMBOL TABLE
LDB ASME2
JSB SCSYM
TO CLEAR UP ADDRESS CHANGES
LDB ASME2
ADD ADDR WHERE DATA WILL BE INSERTED
STB ZADD
ADD 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

<table>
<thead>
<tr>
<th>SCSYM</th>
<th>NOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA</td>
<td>NUM1 VALUE</td>
</tr>
<tr>
<td>CM1, INB</td>
<td></td>
</tr>
<tr>
<td>STA</td>
<td>NUM2 ADDRESS</td>
</tr>
<tr>
<td>XSTBL</td>
<td></td>
</tr>
<tr>
<td>LOD</td>
<td>ADDR TABLE OF SYMBOL TABLE</td>
</tr>
<tr>
<td>RSS</td>
<td></td>
</tr>
<tr>
<td>SCSM1</td>
<td>ADA, 6 NEXT ENTRY IN SYMBOL TABLE</td>
</tr>
<tr>
<td>LSTBL</td>
<td></td>
</tr>
<tr>
<td>ADA</td>
<td>INB UPPER BOUND OF SYM TBL</td>
</tr>
<tr>
<td>CM1, INB</td>
<td></td>
</tr>
<tr>
<td>ADA</td>
<td>A</td>
</tr>
<tr>
<td>SSB, RSS</td>
<td></td>
</tr>
<tr>
<td>JMP SCSM4 OVERFLOW</td>
<td></td>
</tr>
<tr>
<td>LOD</td>
<td>A, I NO CONTENTS OF ADDRSS</td>
</tr>
<tr>
<td>RSS</td>
<td></td>
</tr>
<tr>
<td>SSB, RSS</td>
<td></td>
</tr>
<tr>
<td>JMP SCSM1 NO ENTRIES</td>
<td></td>
</tr>
<tr>
<td>SCKSP</td>
<td></td>
</tr>
<tr>
<td>STA</td>
<td>ENTRY, SAVE ADDRESS</td>
</tr>
<tr>
<td>ADA</td>
<td>2</td>
</tr>
<tr>
<td>LOD</td>
<td>A, I GET LABEL INFORMATION</td>
</tr>
<tr>
<td>CM1, INB</td>
<td></td>
</tr>
<tr>
<td>ADA</td>
<td>1</td>
</tr>
<tr>
<td>SCSM2</td>
<td></td>
</tr>
<tr>
<td>LOD</td>
<td>SCKSP</td>
</tr>
<tr>
<td>SCSM1</td>
<td></td>
</tr>
<tr>
<td>JMP SCSM3 LABEL DEFINED</td>
<td></td>
</tr>
<tr>
<td>SCSM3</td>
<td></td>
</tr>
<tr>
<td>SCSM1</td>
<td></td>
</tr>
<tr>
<td>SCSM3</td>
<td></td>
</tr>
<tr>
<td>SCSM2</td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>SCSM3</td>
</tr>
<tr>
<td>ADDR2</td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>A, I ADDR IN ASSEMBLED CODE</td>
</tr>
</tbody>
</table>

398
LDA NUM2 TEST ADDRESS
ADD B ADD IN ADDRESS
SSA TOO SMALL
JMP SCBM2 YES
LDA B,I RETRIEVE ADDR POINTER
ADD XDATA EQU ADDR DEFINITION
SSA
JMP SCBM2 YES
ADD NUM1 N0, REDEFINE LABEL
STB ADDR2,I SET VALUE IN SYMBOL TABLE
JMP SCBM2 CONTINUE

* CHECK FOR DATA OR MACHINE CODE LABEL

SCSM4 LDA NUM2 TEST LABEL ADDRESS
STA,INA CONVERT TO POSITIVE
JSB DADAD CORRECTION IF DATA ADDRESS
STA,INA CONVERT TO NEGATIVE VALUE
STA NUM2

* EXAMINE LABEL AREA IN PROGRAM

LDA DJ37
STA TEMP3
LDA JMP
SCSM5 STA TEMP4
ISZ TEMP4
GLB
LDA TEMP4,I LOAD ADDRESS
SBS
CCB
STB IDRCT SET INDIRECT FLAG
EALC-LER CLEAR BIT 15
LDB NUM2 TEST ADDRESS
AQB A
SSB CORRECTION REQUIRED
JMP *+6
LDB IDRCT NO
ADA NUM1 YES ADD IN CORRECTION
SSB
IOR MNEG MASK ON INDIRECT BIT
STA TEMP4,I RETURN ADDRESS
ISZ TEMP3
JMP SCSM5

* Satisfy SC3 References with Data

LDB SCBE1,I GET ADDRESS

SCSM5 LDB 9,I ADDRESS OF NEXT STATEMENT
CPB FNEXT FINISHED
JMP SCSYM,I RETURN
ADB 4 ADDR OF ASSEM FLAG, ASSEM ADDR
LDA 8,I
SSA,RSS DATA
JMP SCSM7 NO
STA STFSP SAVE (A)

INB ADDR OF LENGTH OF ASSEMBLY
LDA 3,I ASSEMBLY LENGTH IN (A)
ADB M1 DECREMENT (B)
SSA,RSS EQU PSEUDO OP
JMP SCSM7 YES
LDA STFSP NO, RESTORE (A)
ELA,CLE,ERA REMOVE BIT 15
ADB NUM1 ADD CORRECTION TERM TO DATA ADDR

SCSM6 ADB 4 ADD CORRECTION TERM TO DATA ADDR
STA 3,I
JMP SCSM5 RESTORE BIT 15

SCSM7 ADB 44 RESET (B)
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
ST5 TEMP5
LDB CSSG3 LENGTH OF DELETION
CM8,IN3
STB TEMP6
ST8 TEMP5
ISZ TEMP5 CLEAR DELETED AREA
ISZ TEMP6 ADVANCE POINTERS
JMP FSP1 ADDRESS OF TABLE
LOAD XFRSP
LOAD SSS
ADO 2 ADVANCE TO NEXT POSITION IN TABLE
LOAD ZFRSP SAVE PRESENT POSTION
CM8,FINA
ADO YFRSP UPPER BOUND OF TABLE
LOAD FSP3 IF TABLE FULL FIND SMALLER ENTRY
LOAD ZFRSP RETRIEVE PRESENT POSTION:
LOAD ZFRSP,I AREA OCCUPIED
SUB SMB
JMP FSP2 YES, LOOK AT NEXT POSITION
LOAD CSSG3 NO, GET LENGTH OF DELETION
STA ZFRSP, I  STORE LENGTH
ISZ ZFRSP
LOA SCBE1  ADDRESS OF DELETION
STA ZFRSP, I  STORE ADDRESS
JMP STFSP, I

FSP3
CLA
STA TEMP2
LOA XFRSP
JMP *+3
LOA TEMP1
ADA *2
STA TEMP1
LOA YFRSP
ADD ADDRESS OF NEXT BLOCK IN FSP
FSP4
CMU, INB
ADD A
SSB, RSS
JMP FSP5
LOA TEMP1, I
CMU, INB
ADD CNFG3
SSB
JMP FSP4
SBB, RSS
JMP FSP4
NO, GET LENGTH OF DELETION
CMU, INB
ADD CNFG3
SSB
JMP FSP4
SBB, RSS
JMP FSP4
NO
LOA TEMP2
SBB
JMP FSP5
STA TEMP2
JMP FSP4
ADDRESS IN FREE SPACE
FSP5
LOA TEMP2, I
CMU, INB
ADD A, I
SSB
STA TEMP2
JMP FSP4
LOA TEMP2
SBB, RSS
JMP STFSP, I
LOA CNFG3
STA A, I
INA
LOA SCBE1  ADDRESS OF DELETION
STA A, I
JMP STFSP, I
* RETRIEVE ASSEMBLED CODE ADDRESSES OF INSTRUCTIONS
* INVOLVED IN THE EDIT OPERATION

```
ASMA D NOP
  LDA SCRE  ADDRESS OF SCB ADDRESS
  STA TEMP1
  LDB ASME  ADDR OF ASSEMBLED CODE ADDR STORE
  STB TEMP2
  LDA M3
  STA TEMP3
  LDA TEMP1, I
  ADDR IN SCB
  SIZ, RRS
  JMP ASMD2
  AOA, 4
  ADDR OF ASSEMBLY
  LOA A, I
  ADDR IN ASSEMBLED CODE
ASMD1   STB TEMP2, I
  ISZ TEMP1
  ISZ TEMP2
  ADVANCE ADDRESS POINTERS
  ISZ TEMP3
  JMP ASMD1
  JMP ASMA D, I
```
ORG 120009

**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(DELETE), 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 EDITED.
* A RESPONSE OF YES(S) WILL CONTINUE THE EDIT INSTRUCTION WITH ANY OTHER RESPONSE CAUSING THE EDIT INSTRUCTION TO BE IGNORED.

* TO INSERT BETWEEN SUCCESSIVE STATEMENTS
*  /I(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.
*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 STATEMENT TO BE INSERTED WOULD EXCEED THE NEXT STATEMENT NUMBER IN THE PROGRAM.

*TO REPLACE A SINGLE STATEMENT*

\[R(\text{PLACE}), M(, V)\]

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.

\[E(\text{ND})\]

EDIT CPA SLASH SLASH PRECEDING EDIT OPERATION

RSS JMP EDR1 NO, ERROR

EDIT1 LDB MIIP MULTIPLE INSERT NOW COMPLETE

SZB JSB ENMI 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 EDIT40 YES
CPA I NO, INSERT REQUEST
JMP EDIT2 YES
ADB .2 NO ADVANCE INSTR NUMBER
CPA R REPLACE REQUEST
RSS YES
JMP EDR2 NO, UNDEFINED EDIT OPERATION
EDIT2 STB EDNUM SAVE INSTRUCTION NUMBER
JSB ROCOM READ UPTO COMMA
JMP EDR3
JSB TWNTI READ IN STATEMENT NUMBERS
JMP EDIT4+1
JMP EDIT3
JMP EDIT4
LDB EDNUM EDIT INSTRUCTION NUMBER
GPB .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
GPB .5
JMP EDR3
EDIT4 ISZ EDNUM ADVANCE INSTRUCTION NUMBER
LDA NUM1
*
* CHECK RANGE OF FIRST NUMBER
*
LDB FSTMT FIRST STATEMENT NUMBER IN PROGRAM
CM3, IN3
AD3 A ADD FIRST EDIT STATEMENT NUMBER
SSB
JMP EDR5
LDB CUSTN LAST STATEMENT NUMBER IN PROGRAM
CM3
AD3 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
RS
EDIT5 LDA A,1 ADDRESS OF NEXT ENTRY
CPA ENEXT END OF PROGRAM
JMP EDIT7 YES
ADA *2 ADDRESS OF STATEMENT NUMBER
LDB A,1 STATEMENT NUMBER
ADA M2
STD STORE
CM3, IN5
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 SCBF2 SAVE ADDRESS OF INSTRUCTION
JMP EDIT10 WHICH FOLLOWS EDIT OPERATION
EDIT8 LDB STORE RETRIEVE STATEMENT NUMBER
CPB NUM2 LAST STATEMENT TO BE DELETED
JMP EDIT9 YES
CPA ENEXT TERMINATION
JMP EDIT9+2 YES
CM3, IN3
ADB NUM2 FIRST STATEMENT NUMBER AFTER
SSB, RSS MULTIPLE DELETE
JMP EDIT5 NO
RSS YES
EDIT9 LDA A,1 ADDR OF NEXT STATEMENT
CPA ENEXT TERMINATOR IN SCB
```
STA DLTLN  DELETE LAST LINE
STA SC9E2
JMP EDIT12

* CHECK FOR MULTIPLE INSERT

EDIT10 CPB *4  MULTIPLE INSERT
RSS  YES
JMP EDIT12  NO
LOA NUM2
SZA, RSS  ZERO INCREMENT
JMP EDR5  YES, ERROR
LOD SCBE2  INSTRUCTION AFTER INSERT
AD8 *2
LOD 8, I  STATEMENT NUMBER

* UPPER LIMIT OF STATEMENT NUMBER ON A MULTIPLE INSERT

STB EDLMT
CM8, INB
AD8 NUM1
AD8 NUM2  STATEMENT NUMBER INCREMENT
SSB, RSS  TOO LARGE
JMP EDIT11  YES, CONVERT TO SINGLE INSERT
LOD NUM2  PREPARE STATEMENT NUMBERS
CM8, INB  FOR FIRST ENTRY OF MULTIPLE
AD8 NUM1  INSERT
STB NUM1
JMP EDIT13

EDIT11 LDA *3  CONVERT TO AS SINGLE INSERT
STA EDNUM
LDA *40  WARNING TO USERS
LDR EDM1
JSB BP LN
JMP EDIT13

* EDM1 DEF *+1
ASC 20, MULTIPLE INSERT CHANGED TO SINGLE INSERT
* EXAMINE VETO FLAG

**EDIT12**

LDA VETO
SZA, RSS
JMP EDIT13
LDA NUM1
LD3 NUM2
CP8 ZERO
STA NUM2
LDA SC3E0
LD3 FIRST
S7A
LDB AN1
STB SUCAD
JSB NULN, I
CC8 LISTI, I
LDA .30
Lo8 VETRQ
JSB BPLN
JSB DATN, I
CPA Y
JMP EDIT13
JMP CNTRL, I

* VETRQ DEF *+1
ASC 15, DO YOU WISH TO EDIT THIS CODE

**EDIT13**

LDA NUM1
LD3 NUM2
STA ENM1
STM ENM2

* GET ASSEMBLED CODE ADDRESSES OF INSTRUCTIONS
* INVOLVED IN THE EDIT

JSB ASMD, I
LOA EONUM CPA .1 SINGLE DELETE
RSS YES
JMP EDT16 NO
JSB DSCB SET SCB REFERENCES FOR A DELETE
LD3 SCBE1 ADDR OF STATEMENT TO BE DELETED
SZB, RSS
JMP EDR9
JS3 PREPR PREPARE FOR SCAN OF STORED CODE
SZ4, 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
LOA VETO ASSEM FLAG, ADDR OF ASSEMBLY
SSA
JMP EDT14 DATA
LD3 LENTH LENGTH OF ASSEMBLY
CPB .2 TWO WORD ASSEMBLY
RSS
JMP EDT15 NO, ONE WORD ASSEMBLY
LD3 ASME1 STORE JUMPS IN DELETED INSTRUCTION
LOA .2
ADA 3 ADDR WHERE JUMP POINTS
JSB JMSPS REPLACE TWO WORD ASSEMBLY BY JUMPS
EDT14
JSB SFPSP, I
JMP CNTRL, I RETURN TO CONTROLLER

EDT15 JSB SNGDL SINGLE DELETE
JMP EDT14

* MULTIPLE DELETE INSTRUCTION

EDT16 CPA .2 MULTIPLE DELETE
GL3, RSS YES
JMP EDT21 NO
STA VETCK
LD3 NUM1
CMB, INB
ADB NUM2 CHECK THAT FIRST STATEMENT NUMBER
SSB IS LESS THAN SECOND
JMP EDR5 NO, ERROR
LD3 FIRST
LDA SCBE0
SZA
LDB A,I
STB SCBE1
JSB DSC6
LDB SCBE1
JMP *+3

EDT17

LDB SAVR
STB SCBE1
LDA B,I
STA SAVR
CP3 SCBE2
JMP EDT19
JSB PREPR
STA VETO
SZA,RSS
JMP EDT18+2
CLE,ELA
STA ASM1
REAL
CLE
DATA
JMP EDT18
STA DADR2

* CLEAR LOCATION INVOLVED IN EDIT
* (FIRST WORD IN A TWO WORD ASSEMBLY)

CLA
STA DADR2,I
LDA DADR1
SZA
JMP EDT18
LDA DADR2
STA DADR1

EDT18 ADDR
JSB DLTE,I
JSB SFSP,I
LOAD VETO
SZA, RSS
JMP EDIT17
SZA
JMP EDIT17
LOAD LENGTH
CPA *2
RSS
JMP *+4

* CLEAR SECOND WORD IN A TWO WORD DELETE

ISZ DAOR2

CLA
STA DAOR2, I CLEAR DELETION
LOAD VETCK MACHINE INSTRUCTION
ADDB LENGTH SAVE LENGTH OF DELETED
STB VETCK MACHINE CODE
JMP EDIT17

EDIT19
LOAD VETCK NUMBER OF MC WORDS DELETED
SZA, RSS
JMP CNTRL, I NO MC INSTRUCTION DELETED
CPB *1 ONE MC WORD TO BE DELETED
RSS YES
JMP EDIT20 NO
LOAD DAOR1 ADDR OF WORD TO BE DELETED
STA ASMFL
JSB SGDL DELETE ONE MACHINE INSTR
JMP CNTRL, I RETURN TO CONTROLLER

* EDIT20 LOAD DAOR1 ADDR WHERE JUMP ORIGINATES
LOAD DAOR2 ADDR WHERE JUMP RESULTS
INM ADDR OF NEXT INSTR IN ASSEM CODE
JSB JUMPS INSERT JUMPS TO FINISH MULT DELETE
JMP CNTRL, I RETURN TO CONTROLLER
* SINGLE INSERT

EDT21 CPA .3 SINGLE INSERT
RSS YES
JMP EDT24 NO
LSB SCBE1
JMP EDR7 INSTR EXISTS AT POSITION OF INSRT
JSB EDIPT EDITOR SOURCE INPUT
LDA ASMFLG ASSEMBLY FLAG
SZA, RSS COMMENT
JMP EDT22 YES
SSA, RSS
JMP EDT23 MACHINE INSTRUCTION
JMP ODII, I DATA
EDT22 JSB ISC3
JMP SCBI, I
EDT23 JSB ISC3 INSERT INTO SC3
JSB XINS FIND ASSEM CODE BEFORE INSERT
JSB SVPSN HOLD NEXT FREE POSN IN PROGRAM
LDA ASME0 ADDR IN ASSEM CODE
LDD PRENT
JSB CMVE, I MOVE CODE BEFORE INSERT
CLA, INA FLAS FOR M C TO BE STORED
JSB STCO, 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

LDA ASMEO
LDA EDTSV
JSB JMP5
JSB JMPAF
JMP SCBI, I
* MULTIPLE INSERT
* EDT24 CPA .4 MULTIPLE INSERT
   RSS YES
   JMP EDT25 NO
   LDB SC9E1 STATEMENT NUMBER ALREADY
   SZB DEFINED
   JMP EDR7
   CCB
   STB MIIP MULTIPLE INSERT IN PROGRESS
* RETURN FROM SYSTEM CONTROLLER
* MIRT JSB EDIP3 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
   CCB
   STB EDLX EDIT INPUT REQUEST FLAG
   JMP EDR6
EDT25 SSA,RSS
   JMP EDT27 MACHINE INSTRUCTION
   JSB DTD1,I DATA INSERT
EDT26 JSB ISCO SATISFY SC3 REFERENCES
   JMP SC31,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 ISCO
   JSB STCO,I STORE CODE
* 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

EDT30 JSB DLTE, I
JSB SFSP, I
JSB EDIRP 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
RSB YES
JMP EDT35 NO DATA OR COMMENT
CPA .8 MACHINE CODE INSERT
RSB YES
JMP EDT34 NO COMMENT INSERT
LDA ASMFG ADDR IN ASSEM CODE OF DELETION
LDB ELNTH LENGTH OF DELETED CODE
CPB .2 TWO WORD ASSEMBLY
RSB YES
JMP EDT33 NO
LDB LENTH LENGTH OF ASSEM REPLACEMENT CODE
CPB .2 TWO WORD ASSEMBLY
JMP EDT32 YES
JSB STCO, I NO ONE WORD
JSB JNPPF JMP TO EDIT ENTRY
LDB ZUSR P JUMPS AFTER EDIT ENTRY
LDA ASMFE2
SZA, RSS
LDA EUSR P
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  LENGTH  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  AND EDIT CHANGES
       JMP  SCBI,I

*   MACHINE CODE REPLACED BY A COMMENT

*   EDT34  LDA  ASME1  SINGLE DELETE
       JSB  SNGDL  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  DTDL,I  YES
       JMP  SCBI,I
* END REQUEST

EDT40 LDA NEXT
STA PREV,I CLEAR UP REFERENCES IN SC9
LDB EUSRMP SET JUMP TO LINK EXISTING PROGRAM
LDA ZUSRMP WITH REMAINING PROGRAM AREA
JSB JMPS
CLB
STB EDTOG CLEAR EDIT FLAG
JMP CTRL,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 DLT LN DELETE LAST LINE
STB EDT INT EDIT VARIABLE FOR LEXICAL SCAN
STB FOLMT STAT NUM LIMIT ON MULT INSERT
STB EDTS V ADDR FOR MOVING CODE
STB EXPEC INPUT EXPECTATION FLAG
STB MIIP MULTIPLE INSERT IN PROGRESS
STB MCMIP MACHINE CODE MULT INSERT
STB SCBE0 SOURCE CODE BLOCK ADDRESSES
STB SCBE1
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 FOLLORS EDIT STATEMENT
  JSB NIBLK NEXT NON BLANK CHAR
  JMP EDR3
  GTA V VEPO FLAG
  RSS YES
  JMP EDR3 NO
  LDB ECNUM INSTRUCTION NUMBER
  GPO *3 INSERT
  JMP EDR4 YES, ERROR
  STA VETO NO, SET VETO FLAG
  JMP VETCK,I

* * *

EDR1 LDA .38
  LDB *+2
  JMP ERCAL

* *

DEF *+1 ILLEGAL DATA PRECEDES EDIT INSTRUCTION
* *

EDR2 LDA .26
  LDB *+2
  JMP ERCAL

* *

DEF *+1 UNDEFINED EDIT INSTRUCTION
EDR3  LOA *34
      LDB *+2
      JMP ERCAL

DEF *+1
ASC 17, BAD DATA FOLLOWS EDIT INSTRUCTION

EDR4  LOA *32
      LDB *+2
      JMP ERCAL

DEF *+1
ASC 16, VETO NOT PERMITTED ON AN INSERT

EDR5  LOA *30
      LDB *+2
      JMP ERCAL

DEF ERR2  STATEMENT NUMBER OUT OF RANGE

EDR6  LOA *38
      LDB *+2
      JMP ERCAL

DEF *+1
ASC 19, ILLEGAL SOURCE TYPE ENTRY DURING EDIT
EDR7: LDA .32  
       LDB .12  
       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

EDIP TNOP

* 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\4 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
ADD A
SZB, RSS
JMP EDR6 ERROR, TYPE CLASH ON ENTRY
EDPT1  LOB  MIIP  MULTIPLE INSERT IN PROGRESS
SZB, RSS
JMP EDPT2
LDB  ENM1
ADD  ENM2  ADD INCREMENT
STB  ENM1  NEW STATEMENT NUMBER

* CHECK FOR STATEMENT NUMBER RANGE

CHB
ADD  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  *65
LDB  EDPTM
J33  BPLN
JMP CNTRL,I  RETURN TO CONTROLLER

EDPTM  DEF  *+1
ASC 23, STATEMENT IGNORED, MULTIPLE INSERT TERMINATED

* CLEAR CONTROL FLAG
EDPT2  CLR

STB  EDLX
J3B  ASSM,I  GET SC3 ADDRESS
JMP EDIPT,I
**LINK INSERT WITH EXISTING SOURCE CODE BLOCK ENTRIES**

<table>
<thead>
<tr>
<th>SCB</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>ADDR OF INSERT IN SCB</td>
</tr>
<tr>
<td>STA</td>
<td>ADDR OF NEXT IN PREV INSTR</td>
</tr>
<tr>
<td>LDB</td>
<td>ADDR OF NEXT IN NEW INSTR</td>
</tr>
<tr>
<td>STA</td>
<td>ADDR OF PREV IN NEXT INSTR</td>
</tr>
<tr>
<td>LDB</td>
<td>SET ADDR OF PREV IN NEW INSTR</td>
</tr>
<tr>
<td>STA</td>
<td></td>
</tr>
<tr>
<td>LDB</td>
<td>STATEMENT NUMBER</td>
</tr>
<tr>
<td>STA</td>
<td>STORE STATEMENT NUMBER</td>
</tr>
<tr>
<td>LDA</td>
<td>SAVE ADDRESS OF INSERT ON A</td>
</tr>
<tr>
<td>STA</td>
<td>MULTIPLE INSERT OPERATION</td>
</tr>
<tr>
<td>JMP</td>
<td>ISCB, I</td>
</tr>
</tbody>
</table>

---
FIND INSTRUCTION IN PROGRAM WHICH LOGICALLY PRECEDES

INSERTED MACHINE CODE

<table>
<thead>
<tr>
<th>XINS</th>
<th>NOP</th>
<th>ADDR IN ASSEM CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA ASMEO</td>
<td>ADDR IN SOURCE CODE</td>
<td></td>
</tr>
<tr>
<td>LDB SCBEO</td>
<td>PRECEDING STATEMENT NOT FOUND</td>
<td></td>
</tr>
<tr>
<td>SZB, RSS</td>
<td>SAVE ADDRESS</td>
<td></td>
</tr>
<tr>
<td>JMP XINS1</td>
<td>COMMENT</td>
<td></td>
</tr>
<tr>
<td>STB REENT</td>
<td>MACHINE INSTRUCTION</td>
<td></td>
</tr>
<tr>
<td>JMP XINS2</td>
<td>DATA</td>
<td></td>
</tr>
<tr>
<td>JMP XINS3</td>
<td>TERMINATOR</td>
<td></td>
</tr>
<tr>
<td>IN6</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>LDB B, I</td>
<td>ADDR OF PREVIOUS INSTR</td>
<td></td>
</tr>
<tr>
<td>CPB M1</td>
<td>RESTORE ADDR</td>
<td></td>
</tr>
<tr>
<td>JMP XINS4</td>
<td>ASSEM FLAG, ASSEM ADDR</td>
<td></td>
</tr>
<tr>
<td>ADB M4</td>
<td>STC0D, I</td>
<td></td>
</tr>
<tr>
<td>JMP XINS5</td>
<td>STORE CODE</td>
<td></td>
</tr>
</tbody>
</table>

NO MACHINE INSTRUCTION PRECEDES INSERT

<table>
<thead>
<tr>
<th>XINS3</th>
<th>GCB</th>
<th>FIND NEXT INSTR IN ASSEMBLED CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>STB CNFIG</td>
<td>AFTER INSERT</td>
<td></td>
</tr>
<tr>
<td>JSB YINS</td>
<td>SAVE SCR ADDRESS</td>
<td></td>
</tr>
<tr>
<td>JMP XINS5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STB CNFIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSB STCD, I</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LDA ZADD
STA SAVR
LDA ASMF2
CPA XUSRIP
JMP XINS5
YES
NO
SUBTRACT 1 FROM ADDR OF
NEXT STATEMENT IN PROGRAM
INSTR JUMPS
ADDRESS TO LINK EDIT WITH
BEGINNING OF PROGRAM
MULTIPLE INSERT
RETURN TO APPROPRIATE PROGRAM
NO MACHINE CODE PREcedes OR FOLLOWS INSERT
RETRIEVE EDIT POINTER
STORE CODE
RESET EDIT POINTERS
STORE IN SCB
MOVE CODE IF FIRST AREA IN PROGRAM MUST BE RETAINED
SCB ADDRESS
MOVE THE CODE
INSERT JUMPS TO LINK CHANGES
**FIND NEXT INSTRUCTION IN ASSEMBLED PROGRAM AFTER AN INSERT**

**RETURN** P+1 EDIT TEXT LINKED WITH PROGRAM  
P+2 EDIT TEXT NOT LINKED WITH PROGRAM

**YINS**
- NOP
- LDA ASME2
- LDB SCBE2
- CPB ENEXT
- JMP YINS3

**YINS1**
- SZA, RSS
- JMP YINS2
- SSA
- JMP YINS2
- ISZ YINS
- JMP YINS,1

**YINS2**
- LDB B,1
- CPB ENEXT
- JMP YINS3
- ADD,4
- LDA B,1
- ADD M4
- STA ASME2
- JMP YINS1

END OF USER PROGRAM

DATA

MACHINE INSTRUCTION

END OF PROGRAM

YES

SAVE FOR INSERTING JUMPS
* INSERT FOLLOWS LAST MACHINE CODE STATEMENT IN THE PROGRAM

<table>
<thead>
<tr>
<th>YINS3</th>
<th>LDB CNFIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPB M1</td>
<td>CALL FROM SUBROUTINE XINS</td>
</tr>
</tbody>
</table>

| JMP YINS,1 | YES |
| LDB EDNUM | REPLACE OPERATION |
| CPB .5 |
| RSS | YES |
| JMP YINS4 | NO |
| LDA ZADD |
| LDA ASME1 | LINK PROGRAM WITH REPLACEMENT |
| JSB JMPF1 |
| LDA ZUSR |
| JSB JMPF1 |
| JSZ ZUSR | ADVANCE PROGRAM POINTER |
| JSB STCK,I |
| JMP SCBI,I |

<table>
<thead>
<tr>
<th>YINS4</th>
<th>LDB ASME0</th>
<th>ADDR WHERE JUMP ORIGINATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA EDTSV</td>
<td>ADDR WHERE JUMP RESULTS</td>
<td></td>
</tr>
<tr>
<td>JSB JMPS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| JSB EDIAD |
| JMP YINS,I | UPDATE EDIT LINK POINTERS |
**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
CPR 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
```
CP9 ENEXT TERMINATION
JMP MLN4 YES

A8B 4
LDA B,I ADDRESS OF ASSEMBLY
A8B M4

SZA, R85
JMP MLN3
SSA
JMP MLN3
STA ASME2
R85

STB VENTK NO ASSEMBLED CODE FOLLOWS INSERT
JSB ISCR CLEAR UP SCB REFERENCES

LD3 EDCLR ASSEMBLED CODE PRECEDE INSERT
SZB
JMP MLN5
JSB SVPSN

LDA ASMEO
LD3 SAVR MOVE CODE BEFORE INSERT
JSB CMVE,I
JSB STCD,I STORE INSERTED CODE
JMP SC81,I

* NO ASSEMBLED CODE PRECEDES INSERT

MLN5 JSB STCD,I STORE CODE
LDA ZADD SAVE POSITION
STA SAVR SAVE POSITION
JMP SC81,I
** END A MULTIPLE INSERT OPERATION **

** ENDM1 **
NOP

LD B MCMIP
SZ A, RSS
JMP ENDM3
LO A EDCLR
SZ A, RSS
JMP ENDM1
LO A VETCK
SZ A, RSS
JMP XINS4-6
JS B EDTAD
JMP ENDM3
LO A VETCK
SZ A
JMP ENDM2
LO A ASM62
L D B CNFIG
JS B CMVE, I
LO A EDTSV
LO B ASMED
JS B JMP CF
JS B JMPAF
JMP ENDM3

** ENDM2 **
LO A EDTSV
LO B ASMED
JS B JMP CF
JS B EDTAD

** ENDM3 **
CLA
STA MIIP
STA MCMIP
JMP ENDM1, I

M C MULTIPLE INSERT
ASSEMBLED CODE PRECEDE INSERT
YES, ASSEMBLE CODE FOLLOW INSERT
ASSEMBLED CODE FOLLOW INSERT
NO, ASSEMBLE CODE FOLLOW INSERT
MOVE CODE FOLLOWING INSERT
LINK INSERT BACK INTO PROGRAM
STORE JUMP TO LINK INSERTED CODE
CLEAR MULTIPLE INSERT FLAGS
**SET SOURCE CODE BLOCK POINTERS FOR A REPLACE OPERATION**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RSCB</strong></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>LDB ENM1</td>
<td>Replacement first statement</td>
</tr>
<tr>
<td>CPB FSTMT</td>
<td>YES</td>
</tr>
<tr>
<td>JMP RSCB1</td>
<td>NO</td>
</tr>
<tr>
<td>LDA ADDR1</td>
<td></td>
</tr>
<tr>
<td>STA FIRST</td>
<td>Pointer to first statement</td>
</tr>
<tr>
<td>LDB SCBE2</td>
<td>Successor statement</td>
</tr>
<tr>
<td>ST3 A,I</td>
<td>Addr of next in new statement</td>
</tr>
<tr>
<td>INB</td>
<td></td>
</tr>
<tr>
<td>STA B,I</td>
<td>Addr of prev in next statement</td>
</tr>
<tr>
<td>CCB -1</td>
<td>-1 terminator for beginning</td>
</tr>
<tr>
<td>INA</td>
<td>Addr of SCB</td>
</tr>
<tr>
<td>STB A,I</td>
<td>Store terminator</td>
</tr>
<tr>
<td>JMP RSCB3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RSCB1</strong></td>
<td>Replacement last statement</td>
</tr>
<tr>
<td>LDA ENM1</td>
<td></td>
</tr>
<tr>
<td>CPA CUSTN</td>
<td></td>
</tr>
<tr>
<td>JMP RSCB2</td>
<td>YES</td>
</tr>
<tr>
<td>JSB ISGB</td>
<td>NO</td>
</tr>
<tr>
<td>JMP RSCB3,I</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RSCB2</strong></td>
<td>Replacement last statement</td>
</tr>
<tr>
<td>LDA ADDR1</td>
<td>Last statement after edit</td>
</tr>
<tr>
<td>STA PREV</td>
<td></td>
</tr>
<tr>
<td>STA SCBE0,I</td>
<td>Addr of next in prev instr</td>
</tr>
<tr>
<td>LDB SCBE0</td>
<td>Addr of prev statement</td>
</tr>
<tr>
<td>INA</td>
<td></td>
</tr>
<tr>
<td>STB A,I</td>
<td>Store addr of prev</td>
</tr>
<tr>
<td>JMP RSCB3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RSCB3</strong></td>
<td>Store statement number</td>
</tr>
<tr>
<td>LDB ENM1</td>
<td>Statement number</td>
</tr>
<tr>
<td>INA</td>
<td></td>
</tr>
<tr>
<td>STB A,I</td>
<td>Store statement number</td>
</tr>
<tr>
<td>JMP RSCB3,I</td>
<td></td>
</tr>
</tbody>
</table>
初始化存储器以备新程序之需

以下几个表将会被初始化:
- 源代码块 (SCB) 表,用于储存用户源程序
- 主符号表
- 特殊符号表 (SST),用于复合操作数
- 程序位置计数器 (PLC) 表,用于处理未定义的 PLC 引用
- 自由空间表,用于保存地址和长度的删除内容
- 用户程序区,用于容纳机器指令和数据定义

GREET CLC 0, C
TURN OFF ALL I/O
STF 0
TURN ON INTERRUPT SYSTEM

配置I/O子程序

LOAD .15
JSB CNFIG
I/O THROUGH TTY
JSB IOFF, 1
TURN OFF INTERRUPT
* SET MAIN FRAME INTERRUPT LOCATIONS FOR EACH NEW PROGRAM

* LOB .2  FIRST ADDRESS TO BE SET
* LDA MPPEX  JUMP TO FORWARD REFERENCE WARNING
* STA B,1
* INB
* STA B,1
* INB
* LOA HLT4  POWER FAIL HALT
* STA B,1
* INB
* LDA HLT5  MEMORY PROTECT / PARITY ERROR HALT
* STA B,1
* INB
* LDA DMA1  JUMP TO DMA SERVICE ROUTINE
* STA B,1
* LOB .9
* LDA OCI  JUMP TO DATA CHANNEL SERVICE ROUTINE
* STA B,1
* INB
* LDA CCI  CONTROL SERVICE ROUTINE
* STA B,1

* INITIALIZE LENGTH AND ADDRESS POINTERS FOR INPUT FROM DISC

* LDA TRACK  DISC ADDRESS OF DATA
* STA TEMP6
* LDA BUF1  BUFFER LENGTHS FOR OUTPUT
* STA TEMP7
* LDA XSTR1  MEMORY ADDR TO STORE INPUT FROM DISC
* STA ADDR1

* PREPARE TO PRINT FIRST PAGE OF INTRODUCTORY TEXT

* LOA TEMP6,1  DISC ADDRESS
* LOB TEMP7,1  LENGTH OF INPUT
READ DATA FROM DISC AND OUTPUT TO USER
USER MAY SPECIFY OPTIONAL I/O DEVICE

JSB GRT10  READ FROM DISC, THEN PRINT
JSB DATN,1 READ RESPONSE, RETURN FIRST CHAR
GPA S     OUTPUT TO CRT SCREEN
RSS       YES
JMP GRT6  NO
LDB .11
JSB CFNFG CONFIGURE I/O SUBROUTINES
JSB IOFF,1

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 GRT10 READ THEN PRINT DISC INPUT

CLEAR USER PROGRAM TABLES BEFORE READING USER RESPONSE

LDA XSTBL
STA TEMP
LDA XSTBL STARTING ADDR OF SYMBOL TABLE
CLB,RSS
INAA
STB A,I ADVANCE TO NEXT LOCATION
ISZ TEMP
JMP *-3
LDA M125
STA TEMP
LDA XSTBL
ADA .3
STA ADDR1
LDA 3700   UNDEF FORWARD REF INDICATOR
LDB 3700   UNDEF INDIRECT FORWARD REFERENCE
AD3 *125   INDICATOR
JMP *+6    

GRT1 STA SAVA   SAVE (A)
LOA ADDR1    ADVANCE ADDRESS IN SYMBOL TABLE
ADD *5
STA ADDR1
LOA 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
ISZ TEMP     
JMP GRT1     

LDA M75      
STA TEMP     
LDA X5ST     BASE ADDR OF SPECIAL SYM TBL
ADD *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 YUSRIP   UPPER BOUND OF USER PROGRAM
INA
LDB XRTRN    RETURN FROM EXECUTION
STB A,I      STORE RETURN FROM EXECUTION
**INITIALIZE SYSTEM VARIABLES**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JF. CCA STA GRITFG</td>
<td>SET GREET FLAG</td>
</tr>
<tr>
<td>LDA XSCB STA FIRST</td>
<td>PREVIOUS ENTRY SET AS -1</td>
</tr>
<tr>
<td>LDA XUSR STA NEXT</td>
<td>FIRST ENTRY IN SOURCE CODE BLOCK</td>
</tr>
<tr>
<td>LDA XDATA STA ZDATA</td>
<td>NEXT ENTRY IN SOURCE CODE BLOCK</td>
</tr>
<tr>
<td>LDA XSCB STA FIRST</td>
<td>NEXT LOCATION USER PROG AREA</td>
</tr>
<tr>
<td>LDA XDATA STA ZDATA</td>
<td>NEXT LOCATION IN PROG DATA AREA</td>
</tr>
<tr>
<td>LDB YDAT STB YDATA</td>
<td>INITIALIZE VARIABLES</td>
</tr>
<tr>
<td>STB ABSSF</td>
<td>ABS/BSF PSEUDO OP FLAG</td>
</tr>
<tr>
<td>STB DMPFG</td>
<td>DUMP FLAG</td>
</tr>
<tr>
<td>STB EDINT</td>
<td>EDIT INPUT REQUEST</td>
</tr>
<tr>
<td>STB EDLX</td>
<td>SOURCE DURING EDIT</td>
</tr>
<tr>
<td>STB EOTFG</td>
<td>EDIT FLAG</td>
</tr>
<tr>
<td>STB LGCN</td>
<td>COUNT SYMBOL TABLE ENTRIES</td>
</tr>
<tr>
<td>STB MCMIP</td>
<td>CLEAR MULTIPLE INSERT FLAGS</td>
</tr>
<tr>
<td>STB MIIP</td>
<td></td>
</tr>
<tr>
<td>STB SAVA</td>
<td>DUMP VARIABLES</td>
</tr>
<tr>
<td>STB SAVB</td>
<td></td>
</tr>
<tr>
<td>STB SAVEO</td>
<td></td>
</tr>
<tr>
<td>STB SEQFG</td>
<td>SEQUENCE DIRECTIVE FLAG</td>
</tr>
</tbody>
</table>

**RESPONSE TO SEQUENCE REQUEST**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRT8 JSB DATN,I</td>
<td>READ RESPONSE</td>
</tr>
<tr>
<td>CPA S</td>
<td>STATEMENT NUMBER REQUEST</td>
</tr>
<tr>
<td>JMP GRT10</td>
<td>YES</td>
</tr>
<tr>
<td>CLA</td>
<td>NO</td>
</tr>
<tr>
<td>STA CUSTN</td>
<td>CURRENT USER STATEMENT NUMBER</td>
</tr>
<tr>
<td>LDB *10</td>
<td></td>
</tr>
<tr>
<td>STB FSMT</td>
<td>FIRST STATEMENT NUMBER</td>
</tr>
<tr>
<td>STB STINC</td>
<td>STATEMENT NUMBER INCREMENT</td>
</tr>
<tr>
<td>JMP GRT12</td>
<td></td>
</tr>
</tbody>
</table>
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 OSIPT MEMORY ADDR FOR FURTHER DISC INPUT
STA ADDR1
ISZ TEMP6
ISZ TEMP7
LDA TEMP6,I DISC ADDRESS
LDB TEMP7,I INPUT LENGTH
JSB GRT10
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
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 INTRODUCTORY TEXT FROM DISC
* * ENTER (A) DISC ADDRESS
  (B) LENGTH OF INPUT (WORDS)
* *
GRTIO NOP
STB TEMP1
CMN,INB
STB LENTH  NEGATIVE WORD COUNT FOR DMA
LDB DMACH  OUTPUT FIRST DMA CONTROL WORD
OTB 6
CLB
STB HDMSK  DISC HEAD MASK
LDB ADDR1  MEMORY ADDRESS FOR INPUT
JSB DISKI
LDA M12
JSB NWLS1
LDA TEMP1  LENGTH OF INPUT (WORDS)
ALS
LDB ADDP1  LENGTH OF INPUT (CHARACTERS)
LDB TEMP1
JSB WRITE,1
JMP GRTIO,1
* 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 INTRODUCTORY TEXT
* DATA BEGINS ON FIRST SECTOR OF FIRST TRACK ON CARTRIDGE
* DISC

Track def *+1
Oct 400 page 1
Oct 405 page 2
Oct 411 page 3
Oct 415 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 152008**

**MNEMONIC TABLE**

**FIRST TWO LETTERS OF MNEMONIC**

<table>
<thead>
<tr>
<th>ASC 2</th>
<th>ABAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC 7</td>
<td>ADALALARARAS</td>
</tr>
<tr>
<td>ASC 7</td>
<td>ASASASRLBLBLSBS</td>
</tr>
<tr>
<td>ASC 7</td>
<td>CCCCCCCLCLCLCL</td>
</tr>
<tr>
<td>ASC 7</td>
<td>CLCLCMCMCMCP</td>
</tr>
<tr>
<td>ASC 7</td>
<td>DEDEDEDEDEE</td>
</tr>
<tr>
<td>ASC 7</td>
<td>FNEFREPHLININ</td>
</tr>
<tr>
<td>ASC 7</td>
<td>IOISMJSLDLDDL</td>
</tr>
<tr>
<td>ASC 7</td>
<td>LILILSMIMIMN</td>
</tr>
<tr>
<td>ASC 7</td>
<td>OCOTOTRAARBR</td>
</tr>
<tr>
<td>ASC 7</td>
<td>RRRRRRSESFESFSL</td>
</tr>
<tr>
<td>ASC 7</td>
<td>SLSRSSSSSSIST</td>
</tr>
<tr>
<td>ASC 7</td>
<td>STSTSTSSSZSXXO</td>
</tr>
</tbody>
</table>

**THIRD LETTER OF MNEMONIC AND INSTRUCTION NUMBER**

<table>
<thead>
<tr>
<th>OCT</th>
<th>051415, 040405, 041006, 043001</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCT</td>
<td>051001, 051401, 042006, 051401</td>
</tr>
<tr>
<td>OCT</td>
<td>041411, 046005</td>
</tr>
<tr>
<td>OCT</td>
<td>051005, 043001, 051001, 051401</td>
</tr>
<tr>
<td>OCT</td>
<td>051401, 051415, 040401, 041001</td>
</tr>
<tr>
<td>OCT</td>
<td>042401, 040401, 041001, 041404</td>
</tr>
<tr>
<td>OCT</td>
<td>042401, 053003, 047401, 040401</td>
</tr>
<tr>
<td>OCT</td>
<td>041001, 042401, 040406, 041006</td>
</tr>
<tr>
<td>OCT</td>
<td>041412, 043017, 051007, 042007</td>
</tr>
<tr>
<td>OCT</td>
<td>052007, 040401, 041001, 042010</td>
</tr>
<tr>
<td>OCT</td>
<td>052414, 040401, 041001, 052004</td>
</tr>
<tr>
<td>OCT</td>
<td>040401, 041001, 051006, 055006</td>
</tr>
<tr>
<td>OCT 050001, 041001, 041006, 040406</td>
<td>OCT 040406, 041004, 040406, 041005</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>OCT 050001, 041001, 041006, 040406</td>
<td>OCT 040406, 041004, 040406, 041005</td>
</tr>
<tr>
<td>OCT 052013, 040404, 041004, 045001</td>
<td>OCT 051001, 045001, 051001, 046005</td>
</tr>
<tr>
<td>OCT 051005, 051401, 055001, 041403</td>
<td>OCT 051403, 040401, 041001, 041402</td>
</tr>
<tr>
<td>OCT 051402, 040401, 041001, 040406</td>
<td>OCT 041006, 041044, 043003, 047401</td>
</tr>
<tr>
<td>OCT 050001, 040401, 041001, 051006</td>
<td></td>
</tr>
</tbody>
</table>

**Skeleton of Assembled Code**

<table>
<thead>
<tr>
<th>OCT 177777, 040000, 044000, 001700</th>
<th>OCT 001400, 001000, 010000, 001100</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCT 177777, 040000, 044000, 001700</td>
<td>OCT 001400, 001000, 010000, 001100</td>
</tr>
<tr>
<td>OCT 101020, 005700, 005400, 005000</td>
<td>OCT 005100, 007700, 003400, 007400</td>
</tr>
<tr>
<td>OCT 002300, 002400, 006400, 106700</td>
<td>OCT 002100, 103100, 103100, 033000</td>
</tr>
<tr>
<td>OCT 007000, 002200, 050000, 054000</td>
<td>OCT 177777, 100400, 104200</td>
</tr>
<tr>
<td>OCT 177777, 100400, 104200</td>
<td>OCT 177777, 001500, 005500, 177777</td>
</tr>
<tr>
<td>OCT 002004, 005004, 030000, 034000</td>
<td>OCT 002004, 005004, 030000, 034000</td>
</tr>
<tr>
<td>OCT 002004, 005004, 060000, 064000</td>
<td>OCT 002004, 005004, 060000, 064000</td>
</tr>
<tr>
<td>OCT 102500, 006900, 000000, 010000</td>
<td>OCT 102500, 006900, 000000, 010000</td>
</tr>
<tr>
<td>OCT 102400, 006400, 002000, 000000</td>
<td>OCT 102400, 006400, 002000, 000000</td>
</tr>
<tr>
<td>OCT 177777, 102600, 106600, 001200</td>
<td>OCT 177777, 102600, 106600, 001200</td>
</tr>
<tr>
<td>OCT 001300, 005200, 005300, 100100</td>
<td>OCT 001300, 005200, 005300, 100100</td>
</tr>
<tr>
<td>OCT 101100, 002001, 002001, 102200</td>
<td>OCT 101100, 002001, 002001, 102200</td>
</tr>
<tr>
<td>OCT 102300, 000000, 004010, 102200</td>
<td>OCT 102301, 002020, 006020, 070000</td>
</tr>
<tr>
<td>OCT 074000, 102700, 102100, 102100</td>
<td>OCT 074000, 102700, 102100, 102100</td>
</tr>
<tr>
<td>OCT 101100, 002002, 006002, 020000</td>
<td>OCT 101100, 002002, 006002, 020000</td>
</tr>
</tbody>
</table>

---

442
APPENDIX H

BIBLIOGRAPHY
BIBLIOGRAPHY


(9) HEWLETT PACKARD COMPANY, HP Assembler, HP 02116-9014, June 1971, Hewlett Packard Company, Cupertino, California.