INTEGRATED MICROPROCESSOR

IMP-16
UTILITIES REFERENCE MANUAL

March 1974

© National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051
PREFACE

This publication provides user information about the IMP-16 Debugging Program (DEBUG), the Loader Programs, Teletype Input/Output Routines, Firmware (PROM, ROM) Paper Tape Generation, and the IMP-16 Source File Editor. Supplemental information is included in the appendixes. Familiarity with the IMP-16 Programming and Assembler Manual is a prerequisite for using and understanding the information described in this manual.

This manual was previously titled, "IMP-16L Utilities Reference Manual, Publications Number 4200025A". Henceforth, it shall be titled, "IMP-16 Utilities Reference Manual, Order No. IMP-16S/025YB and Publications Number 4200025B".

Copies of this publication and other National Semiconductor publications may be obtained from the sales offices listed on the back cover. The information in this publication is subject to change without notice.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEBUG</td>
</tr>
<tr>
<td>1.1</td>
<td>INTRODUCTION</td>
</tr>
<tr>
<td>1.2</td>
<td>USAGE</td>
</tr>
<tr>
<td>1.2.1</td>
<td>Memory Requirements</td>
</tr>
<tr>
<td>1.3</td>
<td>DEBUG LANGUAGE</td>
</tr>
<tr>
<td>1.3.1</td>
<td>Conventions Used In This Chapter</td>
</tr>
<tr>
<td>1.3.2</td>
<td>Syntax and Semantics</td>
</tr>
<tr>
<td>1.3.3</td>
<td>Syntax</td>
</tr>
<tr>
<td>1.3.4</td>
<td>Semantics</td>
</tr>
<tr>
<td>1.4</td>
<td>COMMUNICATIONS</td>
</tr>
<tr>
<td>1.5</td>
<td>DEBUG COMMANDS</td>
</tr>
<tr>
<td>1.5.1</td>
<td>Command Descriptions</td>
</tr>
<tr>
<td>1.5.2</td>
<td>TYPE</td>
</tr>
<tr>
<td>1.5.3</td>
<td>CHAR</td>
</tr>
<tr>
<td>1.5.4</td>
<td>ALTER</td>
</tr>
<tr>
<td>1.5.5</td>
<td>HALT</td>
</tr>
<tr>
<td>1.5.6</td>
<td>SNAP</td>
</tr>
<tr>
<td>1.5.7</td>
<td>GO</td>
</tr>
<tr>
<td>1.5.8</td>
<td>MOVE</td>
</tr>
<tr>
<td>1.5.9</td>
<td>NOTE</td>
</tr>
<tr>
<td>1.5.10</td>
<td>FIND</td>
</tr>
<tr>
<td>1.5.11</td>
<td>RESET</td>
</tr>
<tr>
<td>1.5.12</td>
<td>CARRIAGE RETURN</td>
</tr>
<tr>
<td>1.5.13</td>
<td>LINE FEED</td>
</tr>
<tr>
<td>1.5.14</td>
<td>BACK ARROW</td>
</tr>
<tr>
<td>1.5.15</td>
<td>Summary of Commands</td>
</tr>
<tr>
<td>2</td>
<td>IMP-16 LOADERS</td>
</tr>
<tr>
<td>2.1</td>
<td>INTRODUCTION</td>
</tr>
<tr>
<td>2.2</td>
<td>ABSPT (ABSOLUTE PAPER TAPE LOADER)</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Usage</td>
</tr>
<tr>
<td>2.2.2</td>
<td>IMP-16 Loading</td>
</tr>
<tr>
<td>2.3</td>
<td>ABSCR (ABSOLUTE CARD READER LOADER)</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Usage</td>
</tr>
<tr>
<td>2.3.2</td>
<td>RLM Loading</td>
</tr>
<tr>
<td>2.4</td>
<td>CRBOOT (CARD READER BOOTSTRAP LOADER)</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Usage</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Bootstrap Procedure</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Format of CRBOOT</td>
</tr>
<tr>
<td>2.5</td>
<td>GENLDR (GENERAL LOADER)</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Usage</td>
</tr>
<tr>
<td>2.5.2</td>
<td>GENLDR Input</td>
</tr>
<tr>
<td>2.5.3</td>
<td>GENLDR Output</td>
</tr>
<tr>
<td>2.5.4</td>
<td>GENLDR Commands</td>
</tr>
<tr>
<td>2.5.5</td>
<td>!OBS - Origin Base Sector</td>
</tr>
<tr>
<td>2.5.6</td>
<td>!OTS - Origin Top Sector</td>
</tr>
<tr>
<td>2.5.7</td>
<td>!RLM - Relocatable Load Module Identifier</td>
</tr>
<tr>
<td>2.5.8</td>
<td>!CLR - Clear Memory</td>
</tr>
<tr>
<td>2.5.9</td>
<td>!CR - Read Input from the Card Reader</td>
</tr>
<tr>
<td>2.5.10</td>
<td>!TTY - Read Input from the Teletype</td>
</tr>
<tr>
<td>2.5.11</td>
<td>!SY - Print the Symbol Table</td>
</tr>
<tr>
<td>2.5.12</td>
<td>!ER - Print Symbols in Error</td>
</tr>
</tbody>
</table>
## CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.13</td>
<td>!LM - Print Limits</td>
</tr>
<tr>
<td>2.5.14</td>
<td>!NLM - Don't Print Limits</td>
</tr>
<tr>
<td>2.5.15</td>
<td>!SQ - Check Sequence Numbers on Input Deck</td>
</tr>
<tr>
<td>2.5.16</td>
<td>!NSQ - Do Not Make Sequence Number Check</td>
</tr>
<tr>
<td>2.5.17</td>
<td>!GO - Execute the Loaded Program</td>
</tr>
<tr>
<td>2.5.18</td>
<td>Messages</td>
</tr>
<tr>
<td>2.5.19</td>
<td>Sample GENLDR Run</td>
</tr>
</tbody>
</table>

### 3 TELETYPING I/O ROUTINES

<table>
<thead>
<tr>
<th>Subchapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2 USAGE</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2.1 Functions</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2.2 Communications</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2.3 Limitations</td>
<td>3-2</td>
</tr>
<tr>
<td>3.3 LOADING</td>
<td>3-2</td>
</tr>
<tr>
<td>3.4 STORAGE REQUIREMENTS</td>
<td>3-2</td>
</tr>
</tbody>
</table>

### 4 FIRMWARE PAPER TAPE GENERATION

<table>
<thead>
<tr>
<th>Subchapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 INTRODUCTION</td>
<td>4-1</td>
</tr>
<tr>
<td>4.2 PROGRAM ENVIRONMENT</td>
<td>4-1</td>
</tr>
<tr>
<td>4.2.1 Program Loading</td>
<td>4-1</td>
</tr>
<tr>
<td>4.2.2 Memory Requirements</td>
<td>4-1</td>
</tr>
<tr>
<td>4.2.3 Program Messages</td>
<td>4-1</td>
</tr>
<tr>
<td>4.3 INPUT/OUTPUT FORMATS</td>
<td>4-3</td>
</tr>
<tr>
<td>4.3.1 Input Formats</td>
<td>4-3</td>
</tr>
<tr>
<td>4.3.2 Output Formats</td>
<td>4-3</td>
</tr>
<tr>
<td>4.4 USAGE</td>
<td>4-3</td>
</tr>
</tbody>
</table>

### 5 EDIT16

<table>
<thead>
<tr>
<th>Subchapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 INTRODUCTION</td>
<td>5-1</td>
</tr>
<tr>
<td>5.2 COMMAND MODE AND TEXT MODE</td>
<td>5-1</td>
</tr>
<tr>
<td>5.3 COMMAND DELIMITERS</td>
<td>5-1</td>
</tr>
<tr>
<td>5.3.1 Arguments</td>
<td>5-1</td>
</tr>
<tr>
<td>5.3.2 Command Properties</td>
<td>5-2</td>
</tr>
<tr>
<td>5.3.3 The Edit Buffer</td>
<td>5-2</td>
</tr>
<tr>
<td>5.4 COMMAND SET</td>
<td>5-2</td>
</tr>
<tr>
<td>5.4.1 KB - Keyboard Read</td>
<td>5-2</td>
</tr>
<tr>
<td>5.4.2 RT - Read Paper Tape</td>
<td>5-3</td>
</tr>
<tr>
<td>5.4.3 RC - Read Card</td>
<td>5-4</td>
</tr>
<tr>
<td>5.4.4 LS, LF, LL - Teletype List</td>
<td>5-4</td>
</tr>
<tr>
<td>5.4.5 PT - Punch Paper Tape</td>
<td>5-4</td>
</tr>
<tr>
<td>5.4.6 TL - Punch Leader/Trailer</td>
<td>5-4</td>
</tr>
<tr>
<td>5.4.7 HP - High-Speed Printer List</td>
<td>5-4</td>
</tr>
<tr>
<td>5.4.8 MD - Modify Line</td>
<td>5-4</td>
</tr>
<tr>
<td>5.4.9 MS - Modify String</td>
<td>5-4</td>
</tr>
<tr>
<td>5.4.10 DL - Delete Line</td>
<td>5-6</td>
</tr>
<tr>
<td>5.4.11 CL - Copy Line</td>
<td>5-6</td>
</tr>
<tr>
<td>5.4.12 MV - Move Line</td>
<td>5-6</td>
</tr>
<tr>
<td>5.4.13 CB - Clear Buffer</td>
<td>5-6</td>
</tr>
<tr>
<td>5.4.14 FS - Find String</td>
<td>5-7</td>
</tr>
<tr>
<td>5.4.15 ST - Set Tab</td>
<td>5-7</td>
</tr>
<tr>
<td>5.5 OPERATING PROCEDURES</td>
<td>5-7</td>
</tr>
<tr>
<td>5.5.1 Starting</td>
<td>5-7</td>
</tr>
<tr>
<td>Chapter</td>
<td>Content</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Error Corrections</td>
</tr>
<tr>
<td>5.6</td>
<td>SAMPLE OF EDIT16 USE</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>IMP-16 CHARACTER SET</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>INSERTION OF RLM CORRECTIONS</td>
</tr>
<tr>
<td>APPENDIX C</td>
<td>FORMAT OF INSTRUCTIONS</td>
</tr>
<tr>
<td>APPENDIX D</td>
<td>CONVERSION TABLES</td>
</tr>
<tr>
<td>APPENDIX E</td>
<td>EDIT16 SYMBOL MEANINGS AND USAGE</td>
</tr>
</tbody>
</table>
ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Example of Card Load by ABSCR</td>
<td>2-2</td>
</tr>
<tr>
<td>2-2</td>
<td>Program Cards for CR Boots</td>
<td>2-5</td>
</tr>
<tr>
<td>2-3</td>
<td>Memory Map</td>
<td>2-7</td>
</tr>
<tr>
<td>2-4</td>
<td>Sequence of Sample IMP-16L GENLDR Run</td>
<td>2-13</td>
</tr>
<tr>
<td>4-1</td>
<td>Input/Output Options</td>
<td>4-4</td>
</tr>
<tr>
<td>5-1</td>
<td>Sample Program Needing Correction</td>
<td>5-9</td>
</tr>
<tr>
<td>5-2</td>
<td>EDIT16 - Implementation of Correction Commands</td>
<td>5-11</td>
</tr>
<tr>
<td>5-3</td>
<td>Corrected Program Listing</td>
<td>5-15</td>
</tr>
<tr>
<td>5-4</td>
<td>Program Listing on PT Command</td>
<td>5-17</td>
</tr>
</tbody>
</table>

TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Summary of DEBUG Commands</td>
<td>1-10</td>
</tr>
<tr>
<td>4-1</td>
<td>Input/Output Options</td>
<td>4-4</td>
</tr>
</tbody>
</table>
Chapter 1

DEBUG

1.1 INTRODUCTION

DEBUG is a relocatable object program that supervises the operation of a user's program during checkout. DEBUG provides the following facilities for testing computer programs.

- Printing selected areas of memory in hexadecimal or ASCII format,
- Modifying the contents of selected areas in memory,
- Modifying computer registers including the stack,
- Inserting instruction breakpoint halts,
- Taking memory snapshots during execution of user's program,
- Initiating execution at any point in program,
- Searching memory,

1.2 USAGE

DEBUG is a relocatable load module that is loaded into the user's environment as any other relocatable program. DEBUG is initially entered through the global location DEBUG. This may be performed by the loader at the completion of the loading of all programs, by a direct jump from a user's program or by an alteration to the program counter at the control panel. DEBUG may also be entered through the global location, DEBUG1. This entry point loads locations 0 and 3 with an initialization routine that enables the user to recover to DEBUG by pressing the INITIALIZE button and then the RUN button.

DEBUG is self-contained; it includes all of its own input and output. The output and formatting ability of DEBUG is available to the user by subroutine calls to the global UCALL program in DEBUG. The calling sequence is as follows:

```
JSR  UCALL
.WORD  x x x x x x x
.WORD  x x x x x x x ;BASE LOCATION TO PRINT
```

NOTE

The definitions of the above assembler language instructions are given in the IMP-16 Programming and Assembler Manual. The locations specified are formatted with eight words per output line and are displayed on the Teletype. The use of UCALL is an independent subfunction of DEBUG and does not affect normal DEBUG control.

1.2.1 Memory Requirements

The following memory is needed to run DEBUG:

- Top Sector: X'42D or 1069₁₀ words,
- Base Page: 6 words.
1.3 DEBUG LANGUAGE

The control statements which are used to command the operation of DEBUG are confined within a set of rules which define the syntax (the format of control statements) and semantics (the meanings of the various symbols and characters comprising the control statement) of the language.

1.3.1 Conventions Used In This Chapter

The following notation conventions are used to describe the commands, both in the general case (in the command descriptions) and in the specific case (in the examples).

- Mixed upper- and lower-case characters are used for comments and notes.

- Nonunderlined characters, numbers, and symbols, used in the examples, indicate computer-generated output from the Teletype printer.

  For example: TURN ON PUNCH

- Underlined characters, numbers, and symbols, used in the examples, indicate user-generated input at the Teletype keyboard. Two types of underlined statements are used:

  1) Lower-case statements or statement parts represent the general case (to be further defined by the rules of syntax).

  2) Upper-case statements or statement parts represent the exact (specific) form of the input required to be typed in.

  For example: -T <address argument> (general case)
               -T 2345:7F (specific case)
               -NOTE ADDRESS (specific case)

- Circled, upper-case characters represent operation of Teletype keyboard keys that do not generate a printed character.

  For example: \text{CR} represents the carriage return key.
               \text{ALT MODE} represents the alternate mode key.

- \text{CTRL/x} represents the operation of the CONTROL key in conjunction with an alphabetic key. The CONTROL key must be depressed first, prior to pressing the alphabetic key, and held in while the alphabetic key is pressed. The combined symbol is circled because a control operation is initiated; no printed symbol is produced.

1.3.2 Syntax and Semantics

The basic elements of DEBUG commands are defined below. In the formal (symbolic) descriptions of DEBUG commands, the following symbols are used:

\(<a>\) Specifies an element "a" either of a command or of another element.

\(::=\) May be read as, is defined as, and appears in a statement which defines the element to its left.
Indicates that at least one of the elements (a, b, c, etc.) should be specified in the command (a or b or c or ... etc.).

\[ a \] Indicates that one of the elements specified inside the braces must be included in the statement.

\[ a \] Indicates that the element(s) specified within the brackets are optional and need not be included in the command, unless desired.

1.3.3 Syntax

The following meanings are assigned to the terms used in the general-case form of the statements.

\[ \langle \text{hexadecimal number} \rangle ::= \]

From one to four digits from the hexadecimal set 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Leading zeros may be omitted.

\[ \langle \text{address argument} \rangle ::= \]

\[ \{ \text{Memory address} \]
\[ \text{Memory address range} \]
\[ \text{Register address} \]
\[ \text{Register address range} \]

A four-digit hexadecimal number specifying a memory location.

\[ \langle \text{memory address} \rangle ::= \]

A memory address, followed with a colon (:), followed with a second memory address.

For example: 3528:354A

The memory address to the left of the colon represents the lower limit of the range; the memory address to the right of the colon represents the high limit of the range.

\[ \langle \text{register address} \rangle ::= \]

The letter 'R', followed with a two-digit hexadecimal number from the range of 0 through X'13'. Because leading zeros may be omitted, the number may appear as a one-, or two-digit number.

For example: R0, R13, R5

\[ \langle \text{register address range} \rangle ::= \]

The letter 'R' alone, specifying all registers; or a register address, followed with a colon (:), followed with a two-digit hexadecimal number. The register address to the left of the colon represents the lower limit of the range; the hexadecimal number to the right of the colon represents the upper limit of the range. The upper limit must be numerically greater than the lower limit.

For example: R6:13

\[ \langle \text{comment} \rangle ::= \]

English language text, including letters and numbers, exactly as typed in.

1-3
\[\text{value} \; ::= \; \text{A four-digit hexadecimal number used as the contents of a memory location or the contents of a register. Consists of a 16-bit number. May also be specified as two ASCII characters.}\
\]

\[\text{ASCII characters} \; ::= \; \text{A two-character set of ASCII characters, preceded with and terminated by a single quote.}\
\]

1.3.4 Semantics

All numbers input to DEBUG are interpreted as hexadecimal digits. Characters may be either decoded as ASCII or hexadecimal, depending on the particular context. The following description explains the use of certain characters:

\[
, \quad (\text{comma})
\]

Delimiter of address and range arguments.

\[
: \quad (\text{colon})
\]

Delimiter for a range argument; signifies that all the locations from the first entry through the last are included in the range. For example, ab signifies all the locations from a through b, including a and b.

\[
\text{R}
\]

Signifies that the address following the character R is a register address or register address range. A two-digit hexadecimal number is associated with the range symbol; this number is limited to the range 0 through X'13.

The values 0 through 3 are for registers AC0, AC1, AC2, and AC3. The values 4 through X'13 are for the stack contents.

R without any value represents all registers, 0 through X'13.

\[
. \quad (\text{period})
\]

The current location (CL) is the last location addressed by the previous DEBUG command.

1.4 COMMUNICATIONS

The user can communicate with DEBUG through a Teletype keyboard, printer, paper tape punch, and paper tape reader. Whenever DEBUG takes control, it types the minus sign character (-) to indicate that it is ready to accept a command. The user then may type control statements to direct the operation of DEBUG. All commands must be terminated by a carriage return (CR) or a line feed (LF). To ignore a command, the (ALT MODE) key may be pressed at any time before the LF or the CR; in this case, the # symbol is printed and no action occurs. If DEBUG detects an error in the command, it types a question mark (?) and prompts for a new command.

As information is transmitted to the Teletype, the Teletype is interrogated for input. If any character is detected, the current output is terminated and the user is prompted for another DEBUG command. This feature is particularly useful for terminating excessive or undesirable output.
More specific information pertaining to particular commands and situations is given in the appropriate command descriptions, below.

Control is returned to DEBUG from a user's program by use of the HALT command. Upon halting, DEBUG types the halt address. Control is transferred back to the user's program from DEBUG by the GO directive. Details pertaining to the HALT and GO directives are described under the descriptions of the commands themselves.

1.5 DEBUG COMMANDS

DEBUG commands consist of a single alphabetic character representing the command to be performed, followed by a list of the arguments for the commanded operation. The arguments are separated by commas. The numeric fields in an argument list must be in hexadecimal notation; leading zeros may be omitted. Blank characters are ignored, except when enclosed by quotation marks. In this manner, blanks may be used to enhance readability. A statement must be terminated by a carriage return or a line feed.

1.5.1 Command Descriptions

The commands that are used in DEBUG are described in the following paragraphs. In the examples contained within the command descriptions, all information input by the user is underlined to distinguish it from the information generated by DEBUG. This is fully discussed in paragraph 1.3.1.

1.5.2 TYPE

-T <address argument> [, <address argument> ......]

The contents of the specified locations are printed on the terminal in hexadecimal notation. The starting address is printed, followed by one to eight locations per line. If the contents of consecutive locations (starting at any location on a line and extending to the end of the line) are alike, only the first of the like locations is printed. The remaining ones are omitted. The next line also is omitted if the contents of all locations on the line are identical to the contents of the last printed location on the previous line. However, if the contents of any location within such a line are different, then the contents of the locations for that line are printed according to the rules given above. The address for a new line always is a multiple of 8 for consistency and ease of reading. The final location referenced becomes the value of the current location. The format for the TYPE printout is illustrated below.

A printout of the contents of locations 104 through 120 is requested as follows:

-T104:120 (CR)

DEBUG responds with the following output: (typical data)

```
0104 88DF 08DF 08DF 08FF 00FF 80FF 08FF 88FF
010C 00FF 88FF 08FF 88DF 88FF 80FF 88DF
0114 08FF 08FF 88FF 08FB 88FB 80EB 80FB 80FB
011C 00FB 00FB A8FB 88F3 4000
```

(Would have been 80FB)

1.5.3 CHAR

-C <address argument> [, <address argument> ......]
This command performs in a manner similar to that of the TYPE command with the exception that the information is printed in ASCII character format. The output line consists of the contents of one to eight words preceded by the address of the first word. The format for the CHAR printout is illustrated below.

The following example shows the memory contents in locations 220 through 223, interpreted as hexadecimal (TYPE statement) and ASCII (CHAR statement).

Using the TYPE statement,

```
-T 220:223 CR
0220 4131 4242 4147 4243
```

Using the CHAR statement,

```
-C 220:223 CR
0220 A1 BB AG BC
```

An ASCII to hexadecimal conversion table is included in appendix A.

1.5.4 ALTER

```
A \{ \langle memory address \rangle \} \{ \langle register address \rangle \} , \langle hexadecimal number \rangle [ , \langle \ldots \rangle , \ldots ]
```

The ALTER command alters the contents of the specified memory or register location(s). Arithmetic and stack registers have register addresses 0 through 3 and 4 through X'13, respectively. Address 4 is the top of the stack; that is, the last value placed in the stack. The user specifies the address of the location or register that is to be altered, followed by a comma and the value to be placed in the addressed location. If the value is terminated by \( \text{CR} \), the value is stored and the ALTER is terminated. If an \( \text{LF} \) terminates the expression, the value is stored and the user is prompted with the address of the next location or register. A comma may be used to indicate successive locations or registers to be altered without repeated prompts. Any error detected by DEBUG during processing prevents the alteration of the location being processed. Pressing the \( \text{ALT MODE} \) key terminates the ALTER without storing the last value. The value of the last prompt is the new current location.

The following example describes altering locations 2212 through 2215:

```
-A 2212, 0000, 0CDD, 00F7, 80EB CR
```

The following example describes the use of the \( \text{LF} \) key and the \( \text{CR} \) key to control and terminate an ALTER operation:

```
-A12A, 5133 LF
012B S LF
012C 'AA' CR
```

In the above example, after altering location 012A, DEBUG responds with the address of the next location to be altered (012B). Without further initialization, the value 8 is entered. The second \( \text{LF} \) causes DEBUG to again prompt with the next available location to be altered (012C). The ASCII characters AA are entered and the ALTER operation is then terminated with a \( \text{CR} \). The final prompt (-) signals the user that DEBUG is ready for another command input.
1.5.5  HALT

-H \langle memory address \rangle [ , \langle \text{hexadecimal number} \rangle ]

HALT terminates control by the user's program at the location specified by the memory address and returns control to DEBUG. The hexadecimal number (optional), if specified, is the number of times to pass through this location before halting execution. The instruction at the given location is not executed when execution of the user's program is terminated; the instruction is normally executed immediately after control is returned to the user's program by use of the GO command.

The HALT command can be used successfully only when the instruction at the HALT location and the instruction at the following location always are executed consecutively. Thus, the instruction at a HALT location cannot be either a skip or a transfer such that the instruction at the following location would not be executed consecutively. Execution of the HALT does not remove it from the user's environment. It is in effect each time that the instruction at the specified location is executed.

NOTE

DEBUG allows the user to set up to seven HALTS and/or SNAPS. If the user attempts to specify an eighth HALT or SNAP, DEBUG responds with an OV and refuses to accept the command.

All registers are saved by DEBUG when it is entered and restored upon a GO command. The register stack must have one location for the DEBUG HALT execution. Upon halting, DEBUG prints CLxxxx, where xxxx is the location of the halt.

For example, a breakpoint halt is required, in the user's program, at location 200. However, if it is desired to halt only after the fifth pass past that point in the routine, the command may appear as follows:

-H 200, 5

CR

1.5.6  SNAP

-S \langle memory address \rangle , \langle address argument \rangle [ , \ldots ]

Each time the memory location specified by the memory address is encountered, the contents of the ranges specified by the address argument are typed at the terminal in hexadecimal form.

The SNAP can be used successfully only when the instructions at the SNAP location and at the next location always are executed consecutively. Thus, the instruction at a SNAP location cannot be either a skip or a transfer such that the instruction at the following location would not be executed consecutively. Execution of the SNAP command does not remove it from the user's environment. It is in effect each time that instruction at the specified location is executed. The user may type any character during the SNAP output to terminate the output and the DEBUG prompt is issued (as if a HALT occurred at the SNAP location).

NOTE

DEBUG allows the user to set up to seven HALTS and/or SNAPS. If the user attempts to specify an eighth HALT or SNAP, DEBUG responds with an OV and refuses to accept the command.

For example, if the user wishes to see the contents of arithmetic register AC0 and the contents of memory addresses 145 through 148, each time the user routine reaches address 224A, the following SNAP command may be used:

-S224A, R0, 145:148

CR
The following is an example of the output that is printed by DEBUG each time the specified address is encountered:

```
CL  224A
R0000  1234
  0145  FFF  1234  5678  9ABC
   typical data
```

1,5,7  GO
-G  <memory address>

The GO command initiates transfer of control to the user's program at the location specified by the optional memory address. If no memory address is specified with the GO statement, the program continues at the last HALT location. All registers are restored in either case.

To start a user's program at location 2600, the following GO statement can be used:
-G2600  CR

To continue execution after a HALT, the following GO statement can be used:
-G  CR

1,5,8  MOVE
-M  <value>,  <address argument>

A selected hexadecimal number or ASCII character pair is placed in the specified range of memory addresses or registers.

For example:
-M'XX', 250:25F  CR  loads all locations with XX

and  -M 0, 250:25F  CR  puts zeros into the range of locations

1,5,9  NOTE
-N  <comment>

The NOTE command permits the user to comment his debugging. All text prior to the carriage return or the line feed is printed on the terminal. No other action is performed.

For example, a NOTE comment may appear as follows:

-N  A COMMENT MAY BE PLACED AFTER THE N  CR
-N  AND SUBSEQUENT LINES MUST ALSO BE  CR
-N  PRECEDED BY AN N.  CR
FIND

- \text{-F} \text{ } \langle \text{value} \rangle, \langle \text{address argument} \rangle

The locations in the range of address argument are searched and the first location that is equal to the value is typed. If there is no match, DEBUG simply reprompts. The current location is either the location found or the last location searched.

To find the pair of ASCII characters 'LK' in the first 4K of memory, the following command may be used:

\text{-F'LK', 0:FFF CR}

DEBUG responds with the following output:

\text{CL 0AAA} \quad \text{if the characters are located at memory location AAA.}

To find a hexadecimal number, the following command may be used:

\text{-F 48, 0:FFF CR}

The first 4K of memory is searched. If the number is not found, DEBUG reprompts.

RESET

- \text{-R}

The RESET command causes all of the SNAPs or HALTs to be removed and the original code replaced.

For example:

\text{-R CR}

CARRIAGE RETURN \text{(CR)}

Typing only the carriage return causes typing of the current location (CL) in the format (ASCII/hexadecimal) of the last command.

LINE FEED \text{(LF)}

Typing only the line feed causes typing of the current location plus one (CL+1) in the current format. The current location is also increased by one.

BACK ARROW \text{(arrow)}

Typing only the back arrow causes typing of the current location minus one (CL-1) in the current format. The current location is decreased by one.

Summary of Commands

Table 1-1 lists the commands in alphabetical order.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Command</th>
<th>Structure of Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ALTER</td>
<td>A ({\text{memory address}}), (\text{hexadecimal number}) [, \langle\ldots\rangle\langle\ldots\rangle]</td>
</tr>
<tr>
<td>C</td>
<td>CHAR</td>
<td>C (\text{address argument}) [, \langle\text{address argument}\rangle\ldots]</td>
</tr>
<tr>
<td>F</td>
<td>FIND</td>
<td>F (\langle\text{value}\rangle), \langle\text{address argument}\rangle)</td>
</tr>
<tr>
<td>G</td>
<td>GO</td>
<td>G (\text{memory address})</td>
</tr>
<tr>
<td>H</td>
<td>HALT</td>
<td>H (\text{memory address}), (\text{hexadecimal number})</td>
</tr>
<tr>
<td>M</td>
<td>MOVE</td>
<td>M (\langle\text{value}\rangle), \langle\text{address argument}\rangle)</td>
</tr>
<tr>
<td>N</td>
<td>NOTE</td>
<td>N (\langle\text{comment}\rangle)</td>
</tr>
<tr>
<td>R</td>
<td>RESET</td>
<td>R \langle\text{}\rangle)</td>
</tr>
<tr>
<td>S</td>
<td>SNAP</td>
<td>S (\text{memory address}), \langle\text{address argument}\rangle [, \ldots]</td>
</tr>
<tr>
<td>T</td>
<td>TYPE</td>
<td>T \langle\text{address argument}\rangle [, \langle\text{address argument}\rangle\ldots]</td>
</tr>
</tbody>
</table>

| CR     | Type Current Location (CL) |
| LF     | Type Next Location (CL+1) |
| ←      | Type Previous Location (CL-1) |
Chapter 2

IMP-16 LOADERS

2.1 INTRODUCTION

The IMP-16 loaders are a compilation of programs that read and load one or more Relocatable Load Modules (RLMs), produced by the IMP-16 Assembler, into the main memory for execution. Each of these programs is introduced briefly below and then is described in subsequent sections of this manual.

Absolute Paper Tape Loader (ABSPT) is a stand-alone program that reads GENLDR (or any other single RLM not requiring relocation) from the paper tape reader and loads it into main memory for execution. ABSPT is permanently resident in the IMP-16 ROM. Paper tapes read by the IMP-16 are assumed to contain eight channels of binary data. An RLM appears as a series of records of binary data in standard format (IMP-16 Programming and Assembler Manual); each record is preceded by a Start of Text (STX) character and separated by Null characters.

ABSCR is a stand-alone program that reads any RLM not requiring relocation from the card reader and loads the program into the main memory for execution. The primary use of ABSCR is to load GENLDR, ABSCR is loaded into the main memory of the IMP-16L for execution by the Card Reader Bootstrap Loader (CRBOOT) but is permanently resident in IMP-16P ROM.

CRBOOT is a stand-alone program for the IMP-16L and is not required for the IMP-16P. It is bootstrapped directly into main memory for execution under control of IMP-16L equipment.

General Loader (GENLDR) is a self-contained IMP-16 program and performs the following functions:

- Reads and loads one or more RLMs from the card reader and/or the Teletype.
- Relocates the modules.
- Resolves external linkages between the modules.
- Provides descriptive information describing memory, globals.

GENLDR is command-driven and provides comprehensive control over the loading process. GENLDR has the capability to change the loading input device. Otherwise loading proceeds in sequence from the device upon which it was initiated.

2.2 ABSPT (ABSOLUTE PAPER TAPE LOADER)

ABSPT loads GENLDR or any other RLM (not requiring relocation) into the IMP-16 main memory from the paper tape reader.

2.2.1 Usage

The RLM tape loaded by ABSPT is an 8-channel tape, composed of successive RLM records, each preceded by a Start-of-text (STX) character. Since each record contains its own length, no extra characters may appear within records, but any character may appear between records.
2.2.2  IMP-16 Loading

ABSPT is resident in Read–only Memory (ROM) on the IMP-16. The procedure for loading paper tape is as follows:

1. Press the INITIALIZE button on the IMP-16 panel.
2. Place the RLM paper tape into the paper tape reader.
3. Press the LOAD PROG button on the IMP-16 panel.
4. Turn on the paper tape reader.

The RLM is loaded, the entry point address transferred to AC2, and the processing halted. At this point, the user can perform one of the following actions:

1. Press RUN to cause execution of the program loaded.
2. Alter the entry point address contained in AC2 and press RUN to cause execution to start at the modified entry point.
3. Load another RLM in the same manner as before. No resolution of inter–RLM linkages is performed; the user is cautioned to ensure that an RLM does not overlay a previously loaded module.

ABSPT checks only for a checksum error and halts if one is discovered. To retry the load, position the paper tape at the beginning of the record in error; press RUN; and turn on the reader. In order to ignore the error, press RUN.

2.3  ABSCR (ABSOLUTE CARD READER LOADER)

ABS CR loads one or more RLMs (which do not require relocation) into the main memory from the card reader. In the IMP–16L, CRBOOT is used to load ABS CR into memory; because of this close relationship, CRBOOT is described under the current heading. Refer to figure 2-1 for a description of a sample deck using CRBOOT and ABS CR.

Figure 2-1. Example of Card Load by ABS CR
2.3.1 Usage

In the IMP-16P, ABSCR is resident in ROM. In the IMP-16L, it is resident in memory, starting at the location specified by the user, and using E9₁₆ words. ABSCR cannot load RLMs into memory location which it occupies.

The RLM card deck loaded by ABSCR is punched one card per RLM record; columns 73 through 80 of the card are ignored. Only the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and blank are allowed in columns 1 through 72; a blank is treated as a zero. Each record is the hexadecimal character equivalent of an RLM record as output by the IMP-16 Assembler (see IMP-16 Programming and Assembler Manual).

2.3.2 RLM Loading

The sequence of events for loading an RLM, which contains nonrelocatable coding, from the IMP-16L card reader is as follows:

1. Place CRBOOT in the card reader, followed by ABSCR, followed by the RLMs to be loaded, and a 1GO card.
2. Press INITIALIZE.
3. If ABSCR is to be loaded at other than location 120₁₀, perform the following steps:
   a. Set Mode Switch to ACO.
   b. Set ABSCR load address into switches.
   c. Press LOAD DATA.
4. Press AUXL.
5. Press RUN.

The sequence of events for loading RLMs from the IMP-16P card reader is as follows:

1. Place the RLMs into the card reader followed by a 1GO card and ready the card reader.
2. Press INITIALIZE.
3. Set MODE Switch to PC.
4. Set X'7F00 into switches.
5. Press LOAD DATA.

NOTE

The following step must be performed; otherwise the system halts.

6. Set MODE Switch to PROG DATA.
7. Press RUN.

As shown in figure 2-1, ABSCR continues to load RLMs until the 1GO card is encountered. When this occurs, if a nonzero entry point is specified in the last RLM, ABSCR loads AC3 with a 1 to indicate that the load device is the card reader and transfers control to the specified entry point.
If the last specified entry point is a '0' (supplied by the assembler as a default value), ABSCR halts. (See Error Code 5, below.) At this point, the user may enter the correct entry point into AC1 and press RUN.

When the user loads more than one RLM, no resolution of inter-RLM linkages is performed; the user is cautioned to ensure that an RLM does not overlay a previously loaded module.

When ABSCR detects an error, it places an error code in AC0 and halts execution. The following codes comprise the error codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I/O Error - A transmission error or data overrun condition occurred on the card reader. In the IMP-16P, this code is the result of a motion error. The status word returned from the reader is placed in AC1 before halting. To reread the card, replace the card in the reader and press RUN.</td>
</tr>
<tr>
<td>2</td>
<td>Invalid Character - Only punches for 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and blank are allowed in card columns 1 through 72. Correct the card; replace it in the card reader; and press RUN.</td>
</tr>
<tr>
<td>3</td>
<td>Checksum - The checksum calculated by ABSCR did not match that found on the last card read, indicating either a read error or a bad card. Correct the card; replace it in the reader; and press RUN.</td>
</tr>
<tr>
<td>5</td>
<td>Invalid Entry - The last END record read contained an entry-point address of '0' and a 1GO card was read. Place the correct entry-point address in AC1 and press RUN.</td>
</tr>
</tbody>
</table>

2.4 CRBOOT (CARD READER BOOTSTRAP LOADER)

CRBOOT is the card reader bootstrap program for the IMP-16L. Its sole purpose is to load a single program that is punched onto cards in a specific format. Typically, it is used to load ABSCR.

2.4.1 Usage

CRBOOT reads successive cards containing only hexadecimal characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, or blank in columns 1 through 72; blank is treated as a zero. Characters are converted to their 4-bit hexadecimal equivalents, packed 4 characters to a word, and stored in successive memory locations. CRBOOT passes control to the loaded program (at the first location loaded) when it reads a card with an asterisk (*) in column 1. (The rest of that card is ignored.) When it transfers control, interrupts are disabled.

2.4.2 Bootstrap Procedure

The following procedure should be performed to bootstrap from the card reader:

1. Press INITIALIZE.
2. Place CRBOOT in the card reader, followed by deck to be loaded (typically ABSCR) and a card containing an asterisk (*) in column 1.
3. If deck is loaded at other than location 12016, perform the following steps:
   a. Set Mode Switch to AC0.

2-4
b. Set load address into switches.

c. Press LOAD DATA.


5. Press RUN.

Note that because CRBOOT resides in memory words 0 through $4F_{16}$ and uses words $50_{16}$ through $9F_{16}$ as an input buffer, CRBOOT cannot load a program into locations 0 through $9F_{16}$.

CRBOOT halts at location $0010_{16}$ if it detects a transmission error or a data overrun condition on the card reader; the entire load process then has to be repeated.

CRBOOT halts at location $0020_{16}$ if it detects an invalid punch. (Only punches 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and blank are allowed.) Correct the card and repeat the entire load process.

2.4.3 Format of CRBOOT

Since CRBOOT is read by the IMP-16L equipment card reader bootstrap, it must be punched in a special format.

Each (odd, even) pair of card columns is packed into one memory word. Rows 2 through 9 of the odd-numbered column are moved into bits 15 through 8; rows 2 through 9 of the even-numbered column are moved into bits 7 through 0. The hardware bootstrap reads one card, packs it into words 0 through $39_{10}$ ($27_{16}$), and passes control to word 0.

CRBOOT is a 2-card bootstrap (see figure 2-2). The first card contains the instructions necessary to read the second card, which has the same format as the first card.

![Program Cards for CR Bootstrap](image)

Figure 2-2. Program Cards for CR Bootstrap
2.5 GENLDR (GENERAL LOADER)

GENLDR is a stand-alone IMP-16 program that loads one or more Relocatable Load Modules (RLMs) produced by the IMP-16 Assembler, performs relocation and resolves external linkages, and loads the RLMs into main memory for execution.

Each RLM must be transcribed to punched cards or paper tape (ordinarily, on the same machine used for assembling IMP-16 Assembly Source programs). These RLMs, as well as the commands that control the loading process, are then input to GENLDR. RLM record formats are described in appendix A of the IMP-16 Assembler Manual.

The paragraphs that follow describe the commands to control GENLDR and the Input sequences required to load an executable program into the IMP-16 main memory. Error messages and diagnostic output of GENLDR are also described.

2.5.1 Usage

GENLDR is a nonrelocatable stand-alone IMP-16 program that must be loaded into memory by one of two absolute loaders: (1) ABSCR allows GENLDR to be input from cards and (2) ABSPT, from paper tape. Once loaded, GENLDR can accept input from either cards or paper tape; although, initially, it accepts input from the device from which it was loaded.

GENLDR occupies approximately $1700_{10}$ words of memory and is typically loaded into upper memory. Programs cannot be loaded by GENLDR into memory that it occupies or uses for the symbol table it generates. However, GENLDR allows the user full use of base page. The memory layout is described in figure 2-3.

The IMP-16 Assembler allows the user to allocate portions of his program in three ways:

- At an absolute memory location
- Relative to the origin of the base sector
- Relative to the origin of the top sector

Typically, absolute allocation is employed to assign locations dependent upon equipment (for example, interrupt entrance address) or to communicate with special-purpose routines. The base sector must be located such that it is contained within the first $256_{10} \ (100_{10})$ locations of memory and typically contains data and pointers necessary for inter-RLM communication. The top sector may reside anywhere in memory (subject to the limitations mentioned above) and normally contains the main portion of the RLM. Care must be exercised to ensure that an absolute sector does not overlay a previously loaded base sector or top sector. (See IOBS and IOTS commands in the following paragraphs.)

Two other limitations are imposed upon the base sector by the IMP-16 computer architecture and the method for resolving certain external linkages. First, any base sector variable that is referenced by an indexed instruction must be allocated to one of the first $7F_{16}$ locations of memory. Second, in resolving certain external linkages, GENLDR may force an indirect reference to a global variable through a pointer in the memory area $FF_{16}$ and downward.

The area of IMP-16L memory between locations $100_{16}$ and $11F_{16}$ is used by the control panel service routine and may not be used by the user. Above address $FF_{16}$, loading is limited, only within the area occupied by GENLDR and the symbol table it generates. (This area may be used by the loaded program, after it receives control from GENLDR.)

As an entry point, GENLDR selects the last nonzero value specified for the set of RLMs loaded. The entry point for any particular RLM, if specified, appears in the END record of that RLM. If the user desires, he may override the entry point selected by GENLDR by specifying the desired entry point in the IGO command (paragraph 2.5.17). If neither of these methods is chosen, GENLDR prints an "ENT" error message and prompts for a new command.
Figure 2-3. Memory Map
2.5.2 GENLDR Input

GENLDR is command-driven. It reads commands and RLMs from either cards or paper tape, and commands are available to switch between input devices. (See ICR and ITTY, paragraphs 2.5.9 and 2.5.10). GENLDR does not recognize any distinction between the Teletype paper tape reader and the Teletype keyboard; therefore, the user may type in his commands at the keyboard and input the RLM from paper tape. Commands entered either on paper tape or the keyboard are echoed back to the Teletype printer; the RLM itself is not echoed.

Commands entered on punched cards must contain an exclamation point (!) in column 1. When the command is entered from the Teletype, GENLDR types the exclamation point to prompt for a command. The user should not type the exclamation point.

Input lines on the Teletype can be terminated by either a carriage return or a line feed. GENLDR supplies a line feed if a carriage return is issued, or a carriage return if a line feed is issued. GENLDR recognizes the following special characters when reading from the Teletype:

- Backspace character
- ALT MODE Delete entire line
- RUB Rubout character (ignored)
- Null character (ignored)

A maximum of 72 characters is allowed in one Teletype input record; excess characters are ignored.

2.5.3 GENLDR Output

GENLDR may be directed to print information descriptive of the loading process. The title information, base sector limits, top sector limits, absolute sector limits, indirect pointer limits, and entry point address of each RLM may be printed on the Teletype (see PIM command). The user may also request the printing of the symbol table, or only those entries of the symbol table that refer to undefined (U) or multiply-defined (M) symbols (see !SY and !ER commands in the following paragraphs).

If PIM is issued, GENLDR types the following information at the end of each RLM:

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>QUALIFYING STRING</th>
<th>AAAA</th>
<th>BBBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS=XXXX;XXXX</td>
<td>TS=XXXX;XXXX</td>
<td>AS=XXXX;XXXX</td>
<td>PTR=XXXX;XXXX</td>
</tr>
</tbody>
</table>

where:

- MNEMONIC is the name of the RLM from the TITLE record.
- QUALIFYING STRING is the qualifying string from the TITLE record.
- AS specifies the low and high addresses of the absolute sector.
- BS specifies the low and high addresses of the base sector.
- TS specifies the low and high addresses of the top sector.
- AAAA and BBBB are the RLM source and object checksums, respectively.
- PTR is the number of indirect pointers generated.
- ENT is the entry address from the END record.
All numbers are printed in hexadecimal notation. If !NLM is the last command issued (of !LM or !NLM), when 1GO is executed, one line of the above format is output containing composite limits of all RLMs over the scope of the !NLM command.

If !SY or !ER is specified (or defaulted), GENLDR prints symbols as follows:

SYMBOL XXXX F

where:

- SYMBOL is the symbol name,
- XXXX is the hexadecimal address of the symbol,
- F is one of the following:
  - M - multiple-defined symbol
  - U - undefined symbol
  - blank - defined symbol

2.5.4 GENLDR Commands

All commands must begin in column 1 of the input record. One or more blanks must separate the command from an operand. Unless otherwise specified, where the term ⟨hex-value⟩ is used below, it represents a hexadecimal number in the range 0000 to FFFF. Leading zeros need not be specified, but no more than four hexadecimal characters can be given for ⟨hex-value⟩.

2.5.5 !OBS - Origin Base Sector

!OBS ⟨hex-value⟩

NOTE

\[0_{16} \leq \text{hex-value} \leq FF_{16}\]. If this command is not given, the first base sector is loaded at location 10_{16}.

The origin for the next base sector is set to ⟨hex-value⟩. If this command is not specified, the next base sector is loaded immediately following the previous base sector. This command should be used to prevent loading a base sector on top of an absolute sector.

2.5.6 !OTS - Origin Top Sector

!OTS ⟨hex-value⟩

NOTE

The highest value of ⟨hex-value⟩ is a function of the memory available, and must not cause overlaying of the locations occupied by the symbol table or GENLDR. If this command is not given, the first top sector is loaded at location 120_{16}.

The origin for the next top sector is set to ⟨hex-value⟩. If this command is not specified, the next top sector is loaded immediately following the previous top sector. This command should be used to prevent loading a top sector on top of an absolute sector.
2.5.7  IRLM - Relocatable Load Module Identifier

IRLM

This command must precede each RLM to be loaded. The RLM is loaded from the same device from which the RLM command is entered.

2.5.8  ICLR - Clear Memory

ICLR

Memory below GENLDR is cleared to zeros if this command is issued before the first RLM is loaded. After loading is completed (that is, a !GO command is read), memory containing the symbol table and GENLDR is zeroed; this latter function is performed even if the ICLR command is issued after an RLM is loaded.

2.5.9  ICR - Read Input from the Card Reader

ICR

Subsequent input is accepted from the card reader.

2.5.10 ITTY - Read Input from the Teletype

ITTY

Subsequent input is accepted from the Teletype. Teletype input is accepted from either the paper tape or the keyboard, but only commands are echoed to the Teletype printer.

2.5.11 ISY - Print the Symbol Table

ISY

The symbol table is printed upon execution of this command.

2.5.12 IER - Print Symbols in Error

IER

Multiply-defined and undefined symbols are printed when this command is read.

2.5.13 ILM - Print Limits

ILM

As each RLM following the ILM command is loaded, GENLDR prints the name, checksums, base sector limits, top sector limits, and absolute sector limits of the RLM. Each time RLM limits are printed, they are reset so that a record of areas occupied by only single RLMs is maintained by the program. The default command is INLM.
2.5.14 !NLM - Don't Print Limits

!NLM

The name, base sector limits, and top sector limits are not printed. Instead, when a !GO command (2,3,4,13) is executed, GENLDR prints the combined base sector limits, top sector limits, and absolute sector limits of all programs loaded. The default command is !NLM.

2.5.15 !SQ - Check Sequence Numbers on Input Deck

!SQ

After this command is executed, the sequence number field of each input card (columns 73 through 80) is tested to ensure that the sequence numbers contained therein appear in ascending order from one card to the next. If they do not appear this way, an error message is printed. The default command is !NSQ.

2.5.16 !NSQ - Do Not Make Sequence Number Check

!NSQ

Execution of this command nullifies the execution of an !SQ command. The default command is !NSQ.

2.5.17 !GO - Execute the Loaded Program

!GO 〈hex-value〉

The entry point specified in the last RLM loaded can be overridden by specifying the entry point address ("〈hex-value〉"). If 〈hex-value〉 is omitted, the last nonzero entry point specified is executed. If no nonzero entry point is specified and no value appears in the command, GENLDR prints an error message and returns to the command mode. If a ICLR command is previously read, the symbol table and memory containing GENLDR are zeroed before execution of the loaded program. Before transfer to the entry point, the combined limits of all programs loaded are printed on the Teletype if the individual program limits are not printed.

2.5.18 Messages

The following messages may be output by GENLDR:

GENERAL LOADER (REV, X) READY.

GENLDR is ready to accept commands, X is the current revision level of GENLDR.

CMND The command is invalid or unrecognized. Reenter the command.

CHAR An RLM record contains characters other than 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Correct the record; reload; and press RUN.

SEQ Record sequence error. The correct sequence is (1) Title Record, (2) Zero or more Symbol Records, (3) Zero or more Data Records, and (4) End Record. Correct record sequence; reload; and press RUN.

CKSM Checksum error on the next-to-last record read. If the checksum field is 0000, no checksum test is made. The read may be retried. Reload the record and press RUN.
NMBR  A sequence number error is detected. Place the input cards in the correct order and restart, or continue.

BSOV  Base sector overflow. The run must be restarted, but the error may be corrected by proper use of OBS and OTS commands.

TSOV  Top sector overflow. The run must be restarted, but the error may be corrected by proper use of OBS and OTS commands.

SYMB  Symbol table overflow. Too many external symbols defined. The run must be restarted, but the error may be corrected by proper use of the OTS command.

ADDR  Addressing error. This error occurs under the following conditions and the run must be restarted:

1. Attempting to reference an indirect pointer generated by the assembler which, because of relocation, is forced to an address greater than $255_{10}$ ($\text{FF}_{16}$).

2. Using an index register in an instruction referencing a base sector variable allocated to a memory address between $128_{10}$ ($\text{80}_{16}$) and $255_{10}$ ($\text{FF}_{16}$).

3. Attempting to use an index register in an instruction referencing an undefined external variable.

4. Referencing an undefined external variable in an instruction which either is flagged indirect already or cannot be so flagged.

EXTN  Unable to locate external symbol in Symbol Table. This error may be caused by attempting to load an RLM with some missing symbol records or by an erroneous patch which looks as if it is referencing an illegal external reference number. The run must be restarted.

AREA  Loading in illegal area (possibly on top of the loader). Restart with valid OBS or OTS commands.

MEM  Memory size exceeded. Loading into nonexistent memory. Recovery not possible, but error may be corrected by proper use of OBS and OTS commands.

SYST  System error caused by a malfunction in system software or hardware. Recovery not possible.

PNCH  Invalid punch in input record. Only the characters: blank, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, ..., X, Y, and Z accepted. Correct, reload record, and press RUN.

CRDR  1. Card reader is offline. Place reader online and press RUN.

2. Transmission error or data overrun on card reader. The status word returned by the card reader is in AC0. Replace card in reader and press RUN.

ENT  No entry point specified for program. GENLDR transfers control to the Teletype for new command.

DROP  Card dropped out of deck (that is, sequence number incremented by more than 10 and IQ in effect). Check last two cards read. If there are no cards missing between them, press RUN; otherwise, correct card deck and reload.

PTCH  nnnnnnn

A patch card (card with sequence number that is not a multiple of 10) with sequence number 'nnnnnnnn' was processed.
2.5.19 Sample GENLDR Run

The sequence of a sample IMP-16L GENLDR run is shown in figure 2-4. The sequence starts with CRBOOT and proceeds through the steps shown to !GO, upon which execution of GENLDR starts at location 288_{10} (120_{16}). When GENLDR is run on an IMP-16P, CRBOOT and ABSCR are not required in the sample deck.

Figure 2-4. Sequence of Sample IMP-16L GENLDR Run
Chapter 3

TELETYPE I/O ROUTINES

3.1 INTRODUCTION

STTYIO contains five Teletype input/output routines assembled in one relocatable object program. Each routine performs one commonly used Teletype function.

3.2 USAGE

A transfer vector is reserved in base page to establish a fixed interface between user and STTYIO

3.2.1 Functions

The five Teletype functions in STTYIO are:

- SETPL  Resets the Teletype and initializes the other four routines for IMP-16L/16P operation.
- INTEST  Tests for Teletype input.
- PUTC  Transmits bits 0 - 7 of Accumulator 0 (AC0) to the Teletype.
- GETC  Transfer from Teletype to bits 0 - 7 of Accumulator 0 (AC0).
- GECO  Same as GETC plus an echo of the character on the Teletype printer.

3.2.2 Communications

The user may call these routines by using the JSR@ instruction and the transfer vector reserved in base page. Addresses in base page are reserved as follows:

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>000B</td>
<td>SETPL</td>
</tr>
<tr>
<td>000C</td>
<td>INTEST</td>
</tr>
<tr>
<td>000D</td>
<td>PUTC</td>
</tr>
<tr>
<td>000E</td>
<td>GETC</td>
</tr>
<tr>
<td>000F</td>
<td>GECO</td>
</tr>
</tbody>
</table>

For example, to transmit bits 0 - 7 of AC0 to the Teletype, the following instruction may be included in the instruction stream:

```
JSR  @X'1D
```

When using INTEST, return from the routine is as follows:

```
CALL+1   INPUT AVAILABLE FROM TELETYPE
CALL+2   NO INPUT FROM TELETYPE
```
A typical user's program using INTES may appear as follows:

```
JSR @X'C
JMP ATTINPUT ;ATTEMPT TO INPUT
JMP NOINPUT ;NO INPUT FROM TELETYPE
```

Return from all the other routines is RTS 0.

### 3.2.3 Limitations

When STTY10 is loaded in memory, locations X'B through X'F in base page are reserved locations. SETPL must be called prior to use of any of the other four routines. Registers and flags are not altered by any of the five routines except AC0 in GECO and GETC. The stack is pushed three levels deep during execution of these routines.

These routines are dependent on system speed. They are good only at normal system speed (1.4 μsecond per microcycle).

### 3.3 LOADING

STTY10 may be loaded by the absolute card reader loader, absolute paper tape loader, or the general loader. Refer to chapter 2 of this manual for additional information concerning IMP-16 loaders.

### 3.4 STORAGE REQUIREMENTS

- **Base Page**: X'B through X'F
- **Top Sector**: STTY10 is currently assembled top sector relocatable and occupies 12310 (7B16) words.
Chapter 4

FIRMWARE PAPER TAPE GENERATION

4.1 INTRODUCTION

PROMP is a program that generates paper tapes for PROM programming. It has the following capabilities:

- Punch PN format PROM tape.
- Punch BC (binary complemented) format PROM tape.
- Punch paper tape RLM from card RLM.

PROMP output tapes for ROM programming are used only in ROMs programmed 8- by-256 bits (used in multiples of two).

4.2 PROGRAM ENVIRONMENT

4.2.1 Program Loading

PROMP may be loaded by GENLDR or by the absolute loader (ABSCR), when PROMP is in card deck object format, and by the LOAD PROG function in the IMP-16 when PROMP is in paper tape object format. PROMP uses the Teletype I/O functions provided in SITY10 and thus requires that SITY10 also be loaded into memory.

4.2.2 Memory Requirements

PROMP requires $655_{16}$ words of top sector memory.

4.2.3 Program Messages

To guide the user in the use of PROMP, query messages are typed. There are four general messages corresponding to the four possible program states. Initially, PROMP types:

NSC IMP-16 FIRMWARE PAPER TAPE GENERATOR
OUTPUT TYPE:

The user must then respond with one of the following codes:

- PN - For PN format PROM tape.
- BC - For binary (complemented) format PROM tape.
- OM - For card RLM to tape RLM conversion.

All responses must be terminated by a carriage return. For the OM option, PROMP types the following message and goes into an output state:

TURN PUNCH ON
MAKE CARD READER READY

At this point, RLM records are converted from card to paper tape. After processing the END record, PROMP goes into a wait state. The user may now turn off the Teletype punch. Typing any key returns the PROMP program to the initial state.

4-1
In either PN or BC options, PROMP goes into an input state. The following message is typed:

**INPUT DEVICE:**

The user may respond with one of the following codes:

- **CR** - Card Reader
- **PT** - Teletype
- **ME** - Memory

**ME Option.** If this device is selected, PROMP then types:

**SPECIFY MEMORY --**

The user must then type the memory range, where the user program is located. A range is designated by the start address and the last address, delimited by a colon (:); the range must be in blocks of 256₁₀ or 512₁₀ words.

Example:

```
SPECIFY MEMORY --FF00:FFFF
```

**CR Option.** PROMP types the following message:

```
MAKE CARD READER READY
TO LOAD RLM
```

The card reader must be turned on at this point.

**PT Option.** If this device is selected, PROMP types the message:

```
MAKE TAPE READER READY
TO LOAD 0M
```

After the loading process, when the END record is recognized, PROMP types:

```
TURN READER OFF
```

The user must turn off the reader and press any key to initiate the output process.

After the loading process, PROMP goes into an output state and types:

**OUTPUT OPTION:**

The user must respond with one of the following codes:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CR</strong></td>
<td>Default, options 1 through 4, below.</td>
</tr>
<tr>
<td>1</td>
<td>First 256 words, left byte.</td>
</tr>
<tr>
<td>2</td>
<td>First 256 words, right byte.</td>
</tr>
<tr>
<td>3</td>
<td>Second 256 words, left byte.</td>
</tr>
<tr>
<td>4</td>
<td>Second 256 words, right byte.</td>
</tr>
</tbody>
</table>
If more than one option is desired, the user may type options consecutively, delimiting with either a comma (,) or space. After the option is typed, PROMP comes back with the message:

TURN PUNCH ON
HIT ANY KEY TO START

After the output process, PROMP goes into its last state, the wait state. This wait state is provided so that the user can turn off the punch before any message is typed by PROMP. To return to the initial state, the user must press any key on the keyboard.

Typing CTRL/D when PROMP is waiting for a response transfers the user to the initial state. Leaders/trailers are punched automatically. If PN is chosen as an output type code, pressing any key during the output process causes an interrupt of the current output option and initiates processing of the next output option.

4.3 INPUT/OUTPUT FORMATS

4.3.1 Input Formats

PROMP accepts object programs in RLM format and memory contents for input. Input device may be a card reader, Teletype, or memory.

4.3.2 Output Formats

There are three possible output formats from PROMP: PN and BC for PROM tapes, and RLM tapes.

**PN Format.** In this format, 1 (one) bit is represented by one frame P (for 1) and N (for 0) in ASCII format. Each 8-bit block is preceded by the character B and followed by F. CR or LF precedes each B character.

**NOTE**

This format should be used when ordering ROMs or PROMs from National Semiconductor Corporation.

**BC Format.** In this format, 8 bits (may be left or right) is represented by one frame. The frame contents is the binary complement.

**RLM Format.** Refer to the IMP-16 Programming and Assembler Manual and chapter 2 of this manual for a complete description of RLM object programs.

4.4 USAGE

By typing the right codes in response to PROMP queries, the user can use PROMP in a variety of ways. The following table shows all possible input/output options.
<table>
<thead>
<tr>
<th>Function</th>
<th>Input</th>
<th>Output</th>
<th>Output Option Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLM to PROM</td>
<td>CR Card Reader</td>
<td>PN BC Binary</td>
<td>(CR), 1, 2, 3, 4</td>
</tr>
<tr>
<td></td>
<td>PT Teletype</td>
<td>PN BC Binary</td>
<td>(CR), 1, 2, 3, 4</td>
</tr>
<tr>
<td></td>
<td>ME Memory</td>
<td>PN BC Binary</td>
<td>(CR), 1, 2, 3, 4</td>
</tr>
<tr>
<td>Card RLM</td>
<td>Card Reader</td>
<td>OM RLM</td>
<td>Not applicable</td>
</tr>
<tr>
<td>to Tape RLM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

EDIT16

5.1 INTRODUCTION

EDIT16 is a paper tape source editor program, used with the IMP-16L/16P processors. EDIT16 enables editing of a previously prepared source program (or any text) or generating and editing new text. Once loaded, the program is self-starting and provides approximately 4000 characters of working storage in a 4K processor.

The normal editing procedure is to input text, edit the text, and output the edited text. Refer to paragraph 5.6 for a sample edit run.

Prepared text, in punched paper tape format, is read in through the Teletype paper tape reader. New text is generated by typing lines of text on the Teletype keyboard. A line of text is a string of characters followed by a (CR) character. Output text is punched on the Teletype paper tape punch.

EDIT16 commands are line oriented. Character editing capability is provided through the use of the "Modify Line/String" command. Automatic line renumbering is performed when lines are inserted, deleted, and moved.

Appendix E contains a table of the symbol meanings that are used in the discussions that follow.

5.2 COMMAND MODE AND TEXT MODE

The EDIT16 is in the command mode when it types a '?' prompt character and waits for a command input. Most EDIT16 commands operate in this mode, but there are a few commands that require additional information for the command to be carried out properly. In getting this additional information, EDIT16 goes into a text mode. Text mode is terminated by either a (CTRL/A) or a (CR) depending on the command being processed. Command mode always follows successful command processing.

If an invalid EDIT16 command is typed, the message ERROR is typed, followed by CR/LF, and then the prompt character (?). Note that only one command per line is accepted.

All EDIT16 commands are terminated by a (CR). If the command is not a text-type command, processing starts after the (CR) character is typed and EDIT16 remains in command mode. If the command is a text type command, typing (CR) causes EDIT16 to go into text mode.

Typing (CR) causes EDIT16 to respond with CR/LF; that is, a carriage return followed by a line feed. Note that when the keyboard is used as an input, (CR) signals the end of a line and also appears in the edit buffer as the last character of a line.

5.3 COMMAND DELIMITERS

5.3.1 Arguments

Arguments are used in EDIT16 commands to specify portions of the edit buffer upon which the command operates. When a command requires an argument but none is typed, default values are called upon and these vary according to the command. EDIT16 uses the following arguments:

\[ n \]

Is an unsigned decimal number from 1 to 9999. It denotes existing line numbers in the edit buffer.
/ Specifies the high and low values of a range argument.

TO Must be followed by n, and operates on n such that n signifies the destination in a transfer type command.

m Number of lines count; usually used in addition or insertion of new lines.

' Single quote; used to enclose a string of characters for commands that require string arguments.

The first four symbols may be combined in different formats to help the user achieve specific tasks. Valid combinations are as follows:

m TO n Usually used to insert new lines into existing text. m gives the number of lines to insert before line number n.

TO n Same as above except that the number of lines to insert before line number n is not a fixed amount. (CTRL/A) is used to terminate insertion.

n₁ TO n₂ Used in copy and move line commands. n₁ is the line to be moved or copied and n₂ is the destination line number. Note that both n₁ and n₂ must be existing line numbers.

n₁/n₂ Used to specify a line number range.

5.3.2 Command Properties

All commands are line oriented. Line numbers specified in a command argument must be lines existing in the edit buffer. Specifying non-existent lines causes abnormal EDIT16 execution, which may result in program crash.

Arguments must be separated from the command by at least one space; although, some commands may execute properly even without command and argument separation.

All valid commands are two characters long.

5.3.3 The Edit Buffer

The edit buffer contains the source text which is being edited. Each line is composed of a line number, line source, and a (CR) character. Source is packed, two ASCII characters per word, and repeated spaces are packed using a repeat count. The edit buffer is located on top of the EDIT16 code such that its size is dependent on the machine memory space.

5.4 COMMAND SET

Commands are grouped into three sets: Input/Output Commands, Text Modify Commands, and Search Commands.

5.4.1 KB - Keyboard Read

The KB command is used to enter text into the edit buffer from the Teletype keyboard. EDIT16 types -> whenever it is ready to accept a new line. Typing CTRL/Q as the first character in a new line terminates the keyboard entry and EDIT16 goes to command mode.
Example:

<table>
<thead>
<tr>
<th>KB</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>FIRST LINE</td>
</tr>
<tr>
<td>→</td>
<td>CNTL/Q</td>
</tr>
<tr>
<td>? KB</td>
<td>1</td>
</tr>
<tr>
<td>→</td>
<td>APPEND ONE LINE</td>
</tr>
<tr>
<td>? KB</td>
<td>2 TO 2</td>
</tr>
<tr>
<td>→</td>
<td>FIRST INSERT</td>
</tr>
<tr>
<td>→</td>
<td>SECOND INSERT</td>
</tr>
<tr>
<td>→</td>
<td>THIRD INSERT</td>
</tr>
<tr>
<td>→</td>
<td>FOURTH INSERT</td>
</tr>
<tr>
<td>→</td>
<td>FIFTH INSERT</td>
</tr>
<tr>
<td>→</td>
<td>CNTL/Q</td>
</tr>
<tr>
<td>?</td>
<td>CR</td>
</tr>
</tbody>
</table>

KB without an operand appends all text entered from keyboard.

Append one line to edit buffer.

Insert two lines before current line 2.

Insert all text entered before current line 3.

Next prompt.

A description of the symbols used in these examples is included in appendix E. Typing CNTL/Q when EDIT16 is waiting for a command initiates the keyboard input mode.

Example:

? CNTL/Q

The current line being entered is ignored if CNTL/Q is entered before the CR. A back arrow deletes the last character typed.

5.4.2 RT - Read Paper Tape

The RT command enters text into the edit buffer from the Teletype reader, but does not print the lines. This command processes input lines, similar to the KB command. To terminate text entry when no fixed number of lines is specified in the argument, the user must turn off the reader and enter a CNTL/Q on the keyboard. The same action may be used to abort the entry. If the reader is turned off in the middle of a line, only the last complete line is transferred to the buffer.

If the edit buffer becomes full when entering lines using the KB, RT, and RC commands, the message:

BUFFER FULL

is typed and EDIT16 goes into command mode. Again, an incomplete line is not entered to the buffer.

Example:

? RT 2 | CR |

TURN READER OFF NOW

Append two lines to the edit buffer. This message is typed after the second line is read.
5.4.3  RC - Read Card

The RC command must be issued only when applicable. That is, on systems equipped with a card reader. It reads a card and treats it as one line. Arguments applicable to the KB command (refer to paragraph 5.4.1) apply here also.

5.4.4  LS, LF, LL - Teletype List

The LS command lists text on the Teletype. Listed lines are numbered. LF lists line number 1, the first line of text. LL lists the last line of text.

Example:

? LS 2/4,6 \[CR\]  Lists lines 2 through 4, and line 6 on the Teletype.

? LS \[CR\]  Lists entire text.

To interrupt the list, press any key on the Teletype keyboard.

5.4.5  PT - Punch Paper Tape

The PT command is used to punch text on paper tape. Immediately following the PT command, the following message is typed:

TURN PUNCH ON

EDIT16 then goes into the wait state. To continue operation, turn on the punch and press any key on the keyboard. To interrupt the punch, press any key. The punch operation is interrupted immediately. At the end of the punch operation, EDIT16 goes to a wait state. Turn off the punch and press any key. Line numbers are suppressed.

Example:

? PT 1/3 \[CR\]  Punch lines 1 through 3.

? PT \[CR\]  Punch entire text.

5.4.6  TL - Punch Leader/Trailer

The TL command punches approximately 5 inches of null characters for use as trailer or leader. Note that the PT command does not provide for a leader or trailer.

5.4.7  HP - High Speed Printer List

The HP command must be issued only where applicable. That is, for systems equipped with a high-speed printer. HP provides the same output list format as the LS command (refer to paragraph 5.4.4).

5.4.8  MD - Modify Line

The MD command enables character editing in a line. Characters may be inserted or deleted and lines may be truncated or extended.
Examples:

? MD 2 CR
Modify line 2.
2 SECOND LINE
Line 2 is typed back.
ALTERS? SECOND TEST LINE CR
Line 2 is extended.
SECOND TEST LINE
ALTERS? CR
(CR) first character typed, means end of modify
? line command.

CTRL/Z followed by any character 'C' advances the carriage to the first occurrence of 'C' in that line.

Example:

? MD 2 CR
Modify line 2.
2 SECOND TEST LINE
Line 2 is typed back.
ALTERS? CTRL/Z D
(CR) , then D advances the carriage to the first occurrence of the character D. (CTRL/Z)
and D are not echoed but the line is.

ALTERS? SECON
Carriage stops here, where the D would have been.

CTRL/Z 'C' may be issued at any point within the line as long as 'C' is present in the columns between
the carriage and the last character of the line.

Typing CTRL/X on any column deletes the character in that column. A '↑' is echoed on the printer
for each CTRL/X.

Example:

2 SECOND TEST LINE
'TEST' is to be deleted.
ALTERS? SECOND ↑↑↑↑↑ CR
2 SECOND LINE

Typing CTRL/A inserts all characters typed after it, up to, but not including a CR. Characters
inserted are enclosed by '<' and '>'.

Example:

? MD 2 CR
Modify line 2.
2 SECOND LINE
ALTERS? SECOND <TEST>
(CR) typed - '<' echoed.
2 SECOND TEST LINE
(CR) typed - '>' echoed.
ALTERS?

Typing a CTRL/Q aborts the current line modification.

Typing a CTRL/D truncates the line.

Example:

? MD 2 CR
Modify line 2
2 SECOND TEST LINE
ALTERS? SECOND TEST CTRL/D
CTRL/D truncates rest of line.
2 SECOND TEST
5.4.9 MS - Modify String

The MS command modifies lines like the MD command. The argument must be a character string enclosed in single quotes ('). If the string argument is not found in the text, EDIT16 merely prompts for the next command.

Example:

? MS 'SEC' CR
  2
  ALTERS?

'SEC' string is in line 2; thus, proceed to modify line 2.

5.4.10 DL - Delete Line

The DL command deletes a line or a range of lines. After the specified lines are deleted, EDIT16 automatically renumbers the text.

Example:

? DL 1,3/5 CR

Delete lines 1, 3, 4, and 5.

EDIT16 types 'VOID RANGE' when the specified line is out of range.

5.4.11 CL - Copy Line

The CL command copies existing lines to anywhere in the buffer. Lines are renumbered automatically.

Example:

? CL 1/2 CR

? CL 3/4 TO 6 CR

Append copies of lines 1 and 2 to the buffer.

Insert copies of lines 3 and 4 before line 6. Line 6 becomes line 8.

5.4.12 MV - Move Line

The MV command moves lines to anywhere in the buffer. Note that MV is like the CL command except that the lines being moved are deleted from their original locations. Lines are renumbered automatically.

Example:

? MV 1 CR

This command results in line 1 becoming the last line and line 2 then becomes the first line - and on, through the buffer.

5.4.13 CB - Clear Buffer

The CB command clears the entire buffer. If an output command is issued and the buffer is clear, EDIT16 types:

NO ACTIVE FILE
5.4.14 FS - Find String

The FS command searches the entire edit buffer for all occurrences of the specified string. Lines containing the string are typed. If no string is found EDIT16 types:

```plaintext
VOID RANGE
```

It then prompts for the next command. Quote mark (') is treated just like any other character.

Example:

```plaintext
? FS 'AC1' CR
VOID RANGE
Find string 'AC1' in edit buffer.

? FS 'X'FF' CR
VOID RANGE
Find string X'FF in edit buffer.

? 3 .WORD X'FF
FF found in line 3.
```

5.4.15 ST - Set Tab

The ST command enables fixed horizontal spacing when collecting text in keyboard mode.

Example:

```plaintext
? ST CR
START? VERIFY?

1 2 3 4 5 6 7 8 9 % 1 2 3 4....% 1 2 3 4 5

Column 1

Column 65
```

EDIT16 prints the row of numbers, one character for each column. After 65 columns are marked in this manner, EDIT16 prints START?. The user then can mark up to three columns, where a tab is desired, with any printable character (in the above example, the number 1 is used). EDIT16 replies with a verification by typing a corresponding 1 in each position where a tab is specified.

5.5 OPERATING PROCEDURES

5.5.1 Starting

The EDIT16 program may be loaded by the absolute loader or general loader and starts automatically. Once loaded, the following sequence occurs:

```plaintext
NSC EDIT16 REV X
MEMORY:
```

Type in the memory range in the format 0:xx, where xx lies in the range 4 ≤ xx ≤ 64 and must be a multiple of 4. "xx" must not be greater than the actual system memory. To get the default of 4, type CR . EDIT16 repeats the message if input format is erroneous. If accepted, EDIT16 goes into command mode and types the prompt character (?).
5.5.2 Error Corrections

EDIT16 processing may be interrupted by the use of the following:

1. **CTRL/Q** aborts the current command input if in command mode, or the current line input if in text mode.

2. Pressing any key on the keyboard during an input or output operation aborts processing and EDIT16 goes into command mode.

3. Back arrow (←), typed while in text mode, deletes the previous character input. Successive back arrows delete preceding characters in the line. Back arrows may not be used in command mode or when inserting characters (characters typed after **CNTL/A** ) in the modify line/ string commands.

5.6 SAMPLE OF EDIT16 USE

The program, listed in figure 5–1 needs to be corrected; the corrections are shown pencilled-in. Figure 5–2 shows the use of EDIT16 commands to implement changes to the sample program. Figure 5–3 shows the corrected program listing.
TITLE 'MULTIPLY AND DIVIDE ROUTINES'

LS
PSIGN=2
NEGATE=1
SELF=2
BIT=3
AC0=0
AC1=1
AC2=2
AC3=3

; AC0=POS JUMP CONDITION
; AC0<=0 JUMP CONDITION
; SELX FLAG
; BIT=-1 JUMP CONDITION
; DEFINE ACCUMULATORS

; MAIN PROGRAM
LD AC0,EA
JSH MULT
HALT
JMP -.3
ST AC3,SAV3
LD AC3,EA
JSR DIVD
HALT
JMP -.4

EA: .WORD 0

; SUBROUTINE MULTIPLY
MULT: ST AC2,SAVE2
ST AC3,SAVE3
LI AC2,0
LI AC3,#16
CAI AC0,0
SFLG SELFF
SHL AC2,1
LOOP: BOC BIT0,+.2
RADD AC1,AC0
ROR AC2,1
SHR AC0,1
AISZ AC3,-1
JSR LOOP
RCPY AC0,AC1
RCPY AC1,AC0
LD AC3,SAVE3
LD AC2,SAVE2
PFLG SELFF
RTS

SAVE2: .WORD 0
SAVE3: .WORD 0

Figure 5-1. Sample Program Needing Correction (Sheet 1 of 2)
; SUBROUTINE DIVIDE

; COUNT:  WORD 0

DIVIDE:  ST AC2,SAV2
        HCPY AC0,AC2
        CAI AC0,1
        RADD AC3,AC0
        BOC NRT0,OVFLW
        LI AC0,-16
        NO
        ST AC0,COUNT
        SET COUNT = 16
        SPLG SELFX
        LI AC0,0
        SHL AC0,1
        SET SELX
        POOL:  ROL AC2,1
        HCPY AC2,AC0
        CAI AC0,1
        RADD AC3,AC0
        BOC NRT0,GOES
        LI AC0,0
        NO
        SHL AC0,1
        CLEAR LINK
        E:
        JMP SHFTLO
        G00:  CAI AC0,1
        HCPY AC0,AC2
        LI AC0,-1
        SHL AC0,1
        SET LINK
        SHFTLO:  ROL AC1,1
        ISZ COUNT
        ARE WE DONE
        JMP POOL
        NO
        HCPY AC1,AC0
        BOC PSIGN,N2
        JMP OVFLW
        YES, OVERFLOW
        HCPY AC2,AC0
        NO MOVE REMAINDER TO AC0,QUOTE
        IN AC1
        OVFLW:  LD AC3,H7000
        HADD AC3,AC3
        JMP DONE
        SAV2:  .WORD 0
        SAV3:  .WORD 0
        H7000:  .WORD X'7000

; DONE:  PFLE SELF Y
        LD A0,SAY
        RETRUE AC2
        LD AC3,SAV3
        RETRUE AC3

Figure 5-1. Sample Program Needing Correction (Sheet 2 of 2)
? KB TO 1

-> \textbf{TITLE 'MULTIPLY AND DIVIDE ROUTINES'}

-> ; CR

-> ; CR

-> (CTRL/Q) ***

7 MD 4/11 CR
4 PSIGN=2
ALTERS?PSIGN = 2 CR

4 PSIGN = 2
ALTERS? CR
5 Ngitol = 11
ALTERS?Ngitol = 11 CR

5 Ngitol = 11
ALTERS? CR
6 SEL=2
ALTERS?SEL = 2 CR

6 SEL = 2
ALTERS? CR
7 Bito = 3
ALTERS?Bito = 3 CR

7 Bito = 3
ALTERS? CR
8 Ac0 = 0
ALTERS?Ac0 = 0 CR

8 Ac0 = 0
ALTERS? CR
9 Ac1 = 1
ALTERS?Ac1 = 1 CR

9 Ac1 = 1
ALTERS? CR
10 Ac2 = 2
ALTERS?Ac2 = 2 CR

10 Ac2 = 2
ALTERS? CR
11 Ac3 = 3
ALTERS?Ac3 = 3 CR

11 Ac3 = 3
ALTERS? CR

; AC0=POS JUMP CONDITION
; AC0=POS JUMP CONDITION
; AC0<=0 JUMP CONDITION
; AC0<=0 JUMP CONDITION
; SELX FLAG
; SELX FLAG
; BIT0=1 JUMP CONDITION
; BIT0=1 JUMP CONDITION
; DEFINE ACCUMULATORS
; DEFINE ACCUMULATORS

Figure 5-2. EDIT16 - Implementation of Correction Commands (Sheet 1 of 4)
? MS 'MAIN' CR
13 ; MAIN PROGRAM
ALTERS?: MAIN <CALLING > CR

ALTERS? CR
13 ; MAIN CALLING PROGRAM
ALTERS? CR
83 RCPY AC2,AC0 ; NO MOVE REMAINDER TO AC0,QUOTE
ALTERS? CR
83 RCPY AC2,AC0 ; NO MOVE REMAINDER TO AC0,QUOTE CR
ALTERS? CR
16 MD 16,21,26,31,34,37,41 CR
16 JSR MULT ; CALL MULTIPLY ROUTINE CR
ALTERS? CR
16 JSR MULT ; CALL MULTIPLY ROUTINE
ALTERS? CR
21 JSR DIVD ; CALL DIVIDE ROUTINE CR
ALTERS? CR
21 JSR DIVD ; CALL DIVIDE ROUTINE
ALTERS? CR
26 ; SUBROUTINE MULT
ALTERS?; SUBROUTINE MULTIPLY CR

ALTERS? CR
26 ; SUBROUTINE MULTIPLY
ALTERS? CR
31 LI AC3,016 ; BIT COUNT=16
ALTERS? CR
31 LI AC3 <> CR
ALTERS? CR
31 LI AC3,16 ; BIT COUNT=16
ALTERS? CR
31 LI AC3,16 <> CR
ALTERS? CR
34 SFLG SEL ; INCLUDE LINK IN SHIFTS
ALTERS? CR
34 SFLG SELFF ; INCLUDE LINK IN SHIFTS
ALTERS? CR
34 SFLG SELFF ; INCLUDE LINK IN SHIFTS
ALTERS? CR
37 RADD AC1,AC1 ; AC1+AC2 --> AC2
ALTERS? CR
37 RADD AC1,AC2 CR
ALTERS? CR
37 RADD AC1,AC2 ; AC1+AC2 --> AC2
ALTERS? CR
41 JSR LOOP
ALTERS? CR
41 JMP LOOP CR

? Figure 5-2. EDIT16 - Implementation of Correction Commands (Sheet 2 of 4)
Figure 5-2. EDIT16 - Implementation of Correction Commands (Sheet 3 of 4)
MD 80  BOC  PSIGN,+2  ; IS RESULT NEG
ALTERS?  BOC  PSIGN,+2  ; IS RESULT NEG
ALTERS?  CR

FS 'OVFLW'  CR
57  BOC  NEQTO,OVFLW  ; IS HI ORDER 7 = DIVISOR
81  JMP  OVFLW  ; YES, OVERFLOW
84  OVFLW:  LD  AC3,H7000

KB TO 84  CR
  ->  DONE:  PFLG  SELFF  ; CLEAN SELX  CR
  ->  RA  CTRL/Q  ***
  ->  LD  AC2,SAV2  ; RESTORE AC2  CR
  ->  LD  AC3,SAV3  ; RESTORE AC3  CR
  ->  RTS  CR

CTRL/Q  ***

LL  CR
95  *END

MD 95  CR
95  *END
ALTERS?  >
ALTERS?  CR  *END
?
TITLE 'MULTIPLY AND DIVIDE ROUTINES'

PSIGN = 2 ; AC0=POS JUMP CONDITION
NRGT0 = 11 ; AC0<>0 JUMP CONDITION
SEFF = 2 ; SELX FLAG
BIT0 = 3 ; BIT0=1 JUMP CONDITION
AC0 = 0 ; DEFINE ACCUMULATORS
AC1 = 1
AC2 = 2
AC3 = 3

; MAIN CALLING PROGRAM
LD AC0,EA ; LOAD MULTIPLIER
JSR MULT ; CALL MULTIPLY ROUTINE
HALT
JMP -3 ; RETURN
ST AC3,SAV3 ; SAVE AC3
LD AC3,EA ; LOAD DIVISOR
JSR DIVD ; CALL DIVIDE ROUTINE
HALT
JMP -4 ; RETURN
EA: WORD 0

; SUBROUTINE MULTIPLY
MULT: ST AC2,SAVE2 ; SAVE AC2
ST AC3,SAVE3 ; SAVE AC3
LI AC2,0 ; CLEAR AC2
LI AC3,16 ; BIT COUNT=16
CAI AC0,0 ; COMPLEMENT AC0 TO SIMPLIFY
BRANCH ON MULTIPLIER BIT0
SFLG SELF ; INCLUDE LINK IN SHIFTS
SHL AC2,1 ; CLEAR LINK
LOOP: BOC BIT0,+,+ ; BRANCH IF AC0 COMPLIMENTED=0
RADD AC1,AC2 ; AC1+AC2 --> AC2
ROR AC2,1 ; ROTATE RESULT OF ADD INTO LINK
SHR AC0,1 ; SHIFT LINK INTO AC0
AISZ AC3,-1 ; DECR COUNT, SKIP IF ZERO
JMP LOOP
KCPFY AC0,AC1 ; MOVE LO ORDER RESULT TO AC1
RCPY AC2,AC0 ; MOVE HI ORDER RESULT TO AC0
LD AC3,SAVE3 ; RESTORE AC3
LD AC2,SAVE2 ; RESTORE AC2
PFLG SELFF ; CLEAR SELF
RTS
SAVE2: WORD 0
SAVE3: WORD 0

Figure 5-3. Corrected Program Listing (Sheet 1 of 2)
SUBROUTINE DIVIDE

51 ;
52 ;
53 DIVD: ST AC2,SAV2
54 RCPY AC0,AC2 ; SAVE AC2
55 CAI AC0,1
56 HADD AC3,AC0 ; SUBTRACT HI ORDER FROM DIVISOR
57 BOC NRG0,OVFLW ; IS HI ORDER == DIVISOR
58 LI AC0,16 ; NO
59 ST AC0,COUNT ; SET COUNT = 16
60 SFLG SELFF ; SET SELX
61 LI AC0,0 ; CLEAR LINK
62 SHL AC0,1
63 SHL AC1,1 ; ROTATE HI ORDER LEFT WITH LINK
64 POOL: KOL AC2,1
65 RCPY AC2,AC0
66 CAI AC0,1
67 HADD AC3,AC0 ; SUBTRACT HI ORDER FROM DIVISOR
68 BOC NRG0,GOES ; IS HI ORDER >= DIVISOR
69 LI AC0,0 ; NO
70 SHL AC0,1 ; CLEAR LINK
71 JMP SHFTLO
72 GOES: CAI AC0,1 ; YES
73 RCPY AC0,AC2 ; HI ORDER = HI ORDER - DIVISOR
74 LI AC0,-1
75 SHL AC0,1 ; SET LINK
76 SHFTLO: KOL AC1,1 ; ROTATE LO ORDER WITH LINK LEFT
77 ISZ COUNT ; ARE WE DONE
78 JMP POOL ; NO
79 RCPY AC1,AC0 ; YES
80 BOC PSGN,+,+2
81 JMP OVFLW ; YES, OVERFLOW
82 RCPY AC0,AC0 ; NO MOVE REMAINDER TO AC0, QUOT
83 IN AC1
84 DONE: PFLG SELFF ; CLEAR SELX
85 LD AC0,SAV2 ; RESTORE AC2
86 LD AC0,SAV3 ; RESTORE AC3
87 RTS
88 OVFLW: LD AC3,H7000 ; SET OVERFLOW
89 HADD AC3,AC3
90 JMP DONE
91 SAV2: .WORD 0
92 SAV3: .WORD 0
93 H7000: .WORD X'7000
94 COUNT: .WORD 0
95 .END

Figure 5-3. Corrected Program Listing (Sheet 2 of 2)
Punch the edited text on paper tape. Observe that punched text is echoed on the printer.

? PT
TURN PUNCH ON

*TITLE 'MULTIPLY AND DIVIDE ROUTINES'

; ; PSIGN = 2 ; AC0=POS JUMP CONDITION
HNGT0 = 11 ; AC0<=0 JUMP CONDITION
SELF = 2 ; SELX FLAG
BIT0 = 3 ; BIT0=1 JUMP CONDITION
AC0 = 0 ; DEFINE ACCUMULATORS
AC1 = 1
AC2 = 2
AC3 = 3

; ; MAIN CALLING PROGRAM
; ; LD AC0,EA ; LOAD MULTIPLIER

Figure 5-4. Program Listing on PT Command
### Appendix A

**IMP-16 CHARACTER SET**

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII</th>
<th>7-Bit Hexadecimal Number</th>
<th>Punched Card Code</th>
<th>Character</th>
<th>ASCII</th>
<th>7-Bit Hexadecimal Number</th>
<th>Punched Card Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>0O</td>
<td>12-0-1-8-9</td>
<td></td>
<td>!</td>
<td>21</td>
<td>11-2-8</td>
<td></td>
</tr>
<tr>
<td>SOH</td>
<td>01</td>
<td>12-1-9</td>
<td></td>
<td>&quot;</td>
<td>22</td>
<td>7-8</td>
<td></td>
</tr>
<tr>
<td>STX</td>
<td>02</td>
<td>12-2-9</td>
<td></td>
<td>#</td>
<td>23</td>
<td>3-8</td>
<td></td>
</tr>
<tr>
<td>EOT</td>
<td>03</td>
<td>12-3-9</td>
<td></td>
<td>$</td>
<td>24</td>
<td>11-3-8</td>
<td></td>
</tr>
<tr>
<td>ENQ</td>
<td>04</td>
<td>7-9</td>
<td></td>
<td>%</td>
<td>25</td>
<td>0-4-8</td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td>05</td>
<td>0-5-8-9</td>
<td>&amp;</td>
<td>26</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEL</td>
<td>06</td>
<td>0-6-8-9</td>
<td>'</td>
<td>27</td>
<td>5-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td>07</td>
<td>0-7-8-9</td>
<td>(</td>
<td>28</td>
<td>12-5-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HT</td>
<td>08</td>
<td>11-6-9</td>
<td>)</td>
<td>29</td>
<td>11-5-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LF</td>
<td>09</td>
<td>12-5-9</td>
<td>*</td>
<td>30</td>
<td>11-4-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VT</td>
<td>0A</td>
<td>0-5-9</td>
<td>+</td>
<td>31</td>
<td>12-6-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>0B</td>
<td>12-3-8-9</td>
<td>,</td>
<td>2B</td>
<td>12-5-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>0C</td>
<td>12-4-8-9</td>
<td>-</td>
<td>2C</td>
<td>0-3-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SO</td>
<td>0D</td>
<td>12-5-8-9</td>
<td>.</td>
<td>2D</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td>0E</td>
<td>12-6-8-9</td>
<td>/</td>
<td>2E</td>
<td>12-3-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLE</td>
<td>0F</td>
<td>12-7-8-9</td>
<td>0</td>
<td>2F</td>
<td>0-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC1</td>
<td>10</td>
<td>12-11-1-8-9</td>
<td>1</td>
<td>30</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC2</td>
<td>11</td>
<td>12-1-9</td>
<td>3</td>
<td>31</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC3</td>
<td>12</td>
<td>11-2-9</td>
<td>4</td>
<td>32</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC4</td>
<td>13</td>
<td>11-3-9</td>
<td>5</td>
<td>33</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAK</td>
<td>14</td>
<td>11-4-9</td>
<td>6</td>
<td>34</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYN</td>
<td>15</td>
<td>11-5-9</td>
<td>7</td>
<td>35</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETB</td>
<td>16</td>
<td>5-8-9</td>
<td>8</td>
<td>36</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td>17</td>
<td>2-9</td>
<td>9</td>
<td>37</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM</td>
<td>18</td>
<td>11-8-9</td>
<td>3</td>
<td>38</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>19</td>
<td>11-1-8-9</td>
<td>9</td>
<td>39</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESC</td>
<td>20</td>
<td>7-8-9</td>
<td>:</td>
<td>3A</td>
<td>2-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FS</td>
<td>21</td>
<td>0-9-9</td>
<td>;</td>
<td>3B</td>
<td>11-6-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GS</td>
<td>22</td>
<td>11-4-8-9</td>
<td>&lt;</td>
<td>3C</td>
<td>12-4-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td>23</td>
<td>11-5-8-9</td>
<td>&gt;</td>
<td>3D</td>
<td>6-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US</td>
<td>24</td>
<td>11-6-8-9</td>
<td>?</td>
<td>3E</td>
<td>0-6-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>25</td>
<td>11-7-8-9</td>
<td>@</td>
<td>3F</td>
<td>0-7-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>26</td>
<td>12-1</td>
<td>a</td>
<td>40</td>
<td>4-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>27</td>
<td>12-2</td>
<td>b</td>
<td>41</td>
<td>12-0-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>28</td>
<td>12-3</td>
<td>c</td>
<td>42</td>
<td>12-0-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>29</td>
<td>12-4</td>
<td>d</td>
<td>43</td>
<td>12-0-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>30</td>
<td>12-5</td>
<td>e</td>
<td>44</td>
<td>12-0-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>31</td>
<td>12-6</td>
<td>f</td>
<td>45</td>
<td>12-0-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>32</td>
<td>12-7</td>
<td>g</td>
<td>46</td>
<td>12-0-6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>33</td>
<td>12-8</td>
<td>h</td>
<td>47</td>
<td>12-0-7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>34</td>
<td>12-9</td>
<td>i</td>
<td>48</td>
<td>12-0-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>35</td>
<td>11-1</td>
<td>j</td>
<td>49</td>
<td>12-0-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>36</td>
<td>11-2</td>
<td>k</td>
<td>50</td>
<td>12-11-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A-1
<table>
<thead>
<tr>
<th>Character</th>
<th>7-Bit Hexadecimal Number</th>
<th>Punched Card Code</th>
<th>Character</th>
<th>7-Bit Hexadecimal Number</th>
<th>Punched Card Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>4C</td>
<td>11-3</td>
<td>l</td>
<td>6C</td>
<td>12-11-3</td>
</tr>
<tr>
<td>M</td>
<td>4D</td>
<td>11-4</td>
<td>m</td>
<td>6D</td>
<td>12-11-4</td>
</tr>
<tr>
<td>N</td>
<td>4E</td>
<td>11-5</td>
<td>n</td>
<td>6E</td>
<td>12-11-5</td>
</tr>
<tr>
<td>O</td>
<td>4F</td>
<td>11-6</td>
<td>o</td>
<td>6F</td>
<td>12-11-6</td>
</tr>
<tr>
<td>P</td>
<td>50</td>
<td>11-7</td>
<td>p</td>
<td>70</td>
<td>12-11-7</td>
</tr>
<tr>
<td>Q</td>
<td>51</td>
<td>11-8</td>
<td>q</td>
<td>71</td>
<td>12-11-8</td>
</tr>
<tr>
<td>R</td>
<td>52</td>
<td>11-9</td>
<td>r</td>
<td>72</td>
<td>12-11-9</td>
</tr>
<tr>
<td>S</td>
<td>53</td>
<td>0-2</td>
<td>s</td>
<td>73</td>
<td>11-0-2</td>
</tr>
<tr>
<td>T</td>
<td>54</td>
<td>0-3</td>
<td>t</td>
<td>74</td>
<td>11-0-3</td>
</tr>
<tr>
<td>U</td>
<td>55</td>
<td>0-4</td>
<td>u</td>
<td>75</td>
<td>11-0-4</td>
</tr>
<tr>
<td>V</td>
<td>56</td>
<td>0-5</td>
<td>v</td>
<td>76</td>
<td>11-0-5</td>
</tr>
<tr>
<td>W</td>
<td>57</td>
<td>0-6</td>
<td>w</td>
<td>77</td>
<td>11-0-6</td>
</tr>
<tr>
<td>X</td>
<td>58</td>
<td>0-7</td>
<td>x</td>
<td>78</td>
<td>11-0-7</td>
</tr>
<tr>
<td>Y</td>
<td>59</td>
<td>0-8</td>
<td>y</td>
<td>79</td>
<td>11-0-8</td>
</tr>
<tr>
<td>Z</td>
<td>5A</td>
<td>0-9</td>
<td>z</td>
<td>7A</td>
<td>11-0-9</td>
</tr>
<tr>
<td>[</td>
<td>5B</td>
<td>12-2-8</td>
<td>ALT</td>
<td>7B</td>
<td>12-0</td>
</tr>
<tr>
<td>\</td>
<td>5C</td>
<td>0-8-2</td>
<td>ESC</td>
<td>7C</td>
<td>12-11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5D</td>
<td>12-7-8</td>
<td></td>
<td>7D</td>
</tr>
<tr>
<td>-</td>
<td>5E</td>
<td>11-7-8</td>
<td>DEL, RUB</td>
<td>7E</td>
<td>11-0-1</td>
</tr>
<tr>
<td>\</td>
<td>5F</td>
<td>0-5-8</td>
<td></td>
<td>7F</td>
<td>12-7-9</td>
</tr>
<tr>
<td>0</td>
<td>60</td>
<td>8-1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table A-2. Legend for Nonprintable Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Definition</th>
<th>Character</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>Null</td>
<td>SO</td>
<td>Shift out</td>
</tr>
<tr>
<td>SOH</td>
<td>Start of heading (also start of message)</td>
<td>SI</td>
<td>Shift in</td>
</tr>
<tr>
<td>STX</td>
<td>Start of text (also EOA, end of address)</td>
<td>DLE</td>
<td>Data link escape</td>
</tr>
<tr>
<td>ETX</td>
<td>End of text (also EOM, end of message)</td>
<td>DC1</td>
<td>Device control 1</td>
</tr>
<tr>
<td>EOT</td>
<td>End of transmission (also EN)</td>
<td>DC2</td>
<td>Device control 2</td>
</tr>
<tr>
<td>ENQ</td>
<td>Enquiry (also ENQRY, WRU)</td>
<td>DC3</td>
<td>Device control 3</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledge (also RU)</td>
<td>DC4</td>
<td>Device control 4</td>
</tr>
<tr>
<td>BEL</td>
<td>Rings the bell</td>
<td>NAK</td>
<td>Negative acknowledge</td>
</tr>
<tr>
<td>BS</td>
<td>Backspace</td>
<td>SYN</td>
<td>Synchronous idle (SYNC)</td>
</tr>
<tr>
<td>HT</td>
<td>Horizontal tab</td>
<td>ETB</td>
<td>End of transmission block</td>
</tr>
<tr>
<td>LF</td>
<td>Line feed or line space (also new line, advances paper to next line, beginning of line)</td>
<td>CAN</td>
<td>Cancel (CANCL)</td>
</tr>
<tr>
<td>VT</td>
<td>Vertical tab (VTAB)</td>
<td>EM</td>
<td>End of medium</td>
</tr>
<tr>
<td>FF</td>
<td>Form feed to top of next page (PAGE)</td>
<td>SUB</td>
<td>Substitute</td>
</tr>
<tr>
<td>CR</td>
<td>Carriage return</td>
<td>ESC</td>
<td>Escape, Prefix</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FS</td>
<td>File Separator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GS</td>
<td>Group separator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RS</td>
<td>Record separator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US</td>
<td>Unit separator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SP</td>
<td>Space</td>
</tr>
</tbody>
</table>
Appendix B

INSERTION OF RLM CORRECTIONS

Corrections may be inserted in RLMs (decks or tapes) by correcting DATA records or adding DATA records just before the RLM END record. These records should agree in format with a standard DATA record (see IMP-16 Programming and Assembler Manual, appendix A). For simplicity, this section explains how a single record may be used to load a single memory location.

The first word of the corrector record should contain X'8005'. The second word of the corrector record should contain a checksum word X'0000' so GENLDR does not attempt to checksum the record. The third word of the record should contain one of the following values indicating the relocation, if any is to be performed:

- X'0000' Absolute record (no relocation)
- X'0001' Base sector relocatable record
- X'0002' Top sector relocatable record

Word 4 of the record should contain either the absolute initial load address of the record or the proper displacement relative to the base-sector or the top-sector origin of the RLM (as indicated by word 3).

Word 5 of the record should contain one of the following values to indicate the relocation, if any, to be performed upon the contents of the data word:

- X'0000' The data is absolute.
- X'4000' The data references a base-sector relocatable address.
- X'8000' The data references a top-sector relocatable address.

Word 6 should contain X'0000'.

The data word to be inserted should appear in word 7.

Example:

In the top sector of an RLM, the following correction is to be inserted:

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
<th>Relocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAA</td>
<td>8109</td>
<td>B</td>
</tr>
<tr>
<td>DAB</td>
<td>5802</td>
<td>A</td>
</tr>
<tr>
<td>DAC</td>
<td>A107</td>
<td>T</td>
</tr>
<tr>
<td>DAD</td>
<td>6107</td>
<td>B</td>
</tr>
<tr>
<td>DAE</td>
<td>A200</td>
<td>A</td>
</tr>
<tr>
<td>DAF</td>
<td>4A01</td>
<td>A</td>
</tr>
<tr>
<td>DB0</td>
<td>2435</td>
<td>B</td>
</tr>
</tbody>
</table>
The origin of the current top sector is D00. The correctors should contain:

<table>
<thead>
<tr>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>8 0 0 5</td>
</tr>
<tr>
<td>8 0 0 5 0</td>
</tr>
<tr>
<td>8 0 0 5 0</td>
</tr>
<tr>
<td>8 0 0 5 0</td>
</tr>
<tr>
<td>8 0 0 5 0</td>
</tr>
<tr>
<td>8 0 0 5 0</td>
</tr>
<tr>
<td>8 0 0 5 0</td>
</tr>
</tbody>
</table>

B-2
Appendix C

FORMAT OF INSTRUCTIONS

A summary of the instruction types and their assembler language formats is given below for reference. A more-detailed breakdown of the instruction codes is shown in the next table; it is suitable for hand-coding small programs.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>MACHINE FORMAT</th>
<th>Assembler Language Format</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register to Register</td>
<td>op sr dr op</td>
<td>Op sr, dr</td>
<td></td>
</tr>
<tr>
<td>Register to Memory</td>
<td>op r disp</td>
<td>Op r, disp</td>
<td></td>
</tr>
<tr>
<td>Memory Reference (Class 1)</td>
<td>op r xr disp</td>
<td>Op r, disp(xr)</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Op r, @disp(xr)</td>
<td>Indirect</td>
</tr>
<tr>
<td>Memory Reference (Class 2)</td>
<td>op xr disp</td>
<td>Op disp(xr)</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Op @disp(xr)</td>
<td>Indirect</td>
</tr>
<tr>
<td>I/O and Miscellaneous</td>
<td>op ctl</td>
<td>Op ctl</td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td>op cc disp</td>
<td>Op cc, disp</td>
<td></td>
</tr>
</tbody>
</table>

Explanation of Symbols

- OP - Instruction Mnemonic
- Op - Operation Code
- sr - Source Register Value
- dr - Destination Register Value
- xr - Index Register Value
- disp - Displacement Value
- cc - Condition Code Value
- r - Register Value
- ctl - Control Bits Value

C-1
### Table C-1. Instruction Set with Bit Patterns

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Base</th>
<th>Word Format</th>
<th>Addressing Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>8000</td>
<td>$1 = \text{BASE} \lor r \lor \text{xr} \lor \text{disp}$</td>
<td>BASE SECTOR</td>
</tr>
<tr>
<td>LD Indirect</td>
<td>9000</td>
<td></td>
<td>PC RELATIVE</td>
</tr>
<tr>
<td>ST</td>
<td>A000</td>
<td></td>
<td>INDEXED - AC2</td>
</tr>
<tr>
<td>ST Indirect</td>
<td>B000</td>
<td></td>
<td>INDEXED - AC3</td>
</tr>
<tr>
<td>ADD</td>
<td>C000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>SUB</td>
<td>D000</td>
<td>0400</td>
<td>0411</td>
</tr>
<tr>
<td>SKG</td>
<td>E000</td>
<td>0800</td>
<td>0200</td>
</tr>
<tr>
<td>SKNE</td>
<td>F000</td>
<td>0C00</td>
<td>0300</td>
</tr>
<tr>
<td>AND</td>
<td>6000</td>
<td>$r$</td>
<td>REGISTER</td>
</tr>
<tr>
<td>OR</td>
<td>6800</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>SKAZ</td>
<td>7000</td>
<td>0400</td>
<td>1</td>
</tr>
<tr>
<td>ISZ</td>
<td>7800</td>
<td>JMP Indirect</td>
<td>2400</td>
</tr>
<tr>
<td>DSZ</td>
<td>7C00</td>
<td>JSR</td>
<td>2800</td>
</tr>
<tr>
<td>JMP</td>
<td>2000</td>
<td>JSR Indirect</td>
<td>2C00</td>
</tr>
<tr>
<td>BOC</td>
<td>1000</td>
<td>$1 = \text{BASE} \lor \text{cc} \lor \text{disp}$</td>
<td></td>
</tr>
</tbody>
</table>

#### Branch on

<table>
<thead>
<tr>
<th>Branch on</th>
<th>INT</th>
<th>AC0=0</th>
<th>AC0 ≥ 0</th>
<th>AC0 ODD</th>
<th>AC0 Bit 1=1</th>
<th>AC≠0</th>
<th>CPINT</th>
<th>START</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>0000</td>
<td>0100</td>
<td>0200</td>
<td>0300</td>
<td>0400</td>
<td>0500</td>
<td>0600</td>
<td>0700</td>
</tr>
</tbody>
</table>

#### Branch on

<table>
<thead>
<tr>
<th>Branch on</th>
<th>STFL</th>
<th>INEN</th>
<th>CYOV</th>
<th>AC0 ≤ 0</th>
<th>USER POA</th>
<th>USER SEL</th>
<th>USER</th>
<th>USER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>0800</td>
<td>0900</td>
<td>0A00</td>
<td>0B00</td>
<td>0C00</td>
<td>0D00</td>
<td>0E00</td>
<td>0F00</td>
</tr>
</tbody>
</table>

#### Word Format

$1 = \text{BASE} \lor r \lor \text{disp}$

#### AISZ

<table>
<thead>
<tr>
<th>AISZ</th>
<th>4800</th>
</tr>
</thead>
</table>

#### LI

<table>
<thead>
<tr>
<th>LI</th>
<th>4C00</th>
</tr>
</thead>
</table>

#### CAI

<table>
<thead>
<tr>
<th>CAI</th>
<th>5000</th>
</tr>
</thead>
</table>

#### PUSH

<table>
<thead>
<tr>
<th>PUSH</th>
<th>4000</th>
</tr>
</thead>
</table>

#### PULL

<table>
<thead>
<tr>
<th>PULL</th>
<th>4400</th>
</tr>
</thead>
</table>

#### XCHRS

<table>
<thead>
<tr>
<th>XCHRS</th>
<th>5400</th>
</tr>
</thead>
</table>

#### ROR/ROL

<table>
<thead>
<tr>
<th>ROR/ROL</th>
<th>5800</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LEFT DISP POSITIVE</td>
</tr>
</tbody>
</table>

#### SHR/SHL

<table>
<thead>
<tr>
<th>SHR/SHL</th>
<th>5C00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RIGHT DISP NEGATIVE</td>
</tr>
</tbody>
</table>

#### Word Format

$1 = \text{BASE} \lor r \lor \text{disp}$

<table>
<thead>
<tr>
<th>RADD</th>
<th>3000</th>
</tr>
</thead>
</table>

#### RXCH

<table>
<thead>
<tr>
<th>RXCH</th>
<th>3080</th>
</tr>
</thead>
</table>

#### RCPY

<table>
<thead>
<tr>
<th>RCPY</th>
<th>3081</th>
</tr>
</thead>
</table>

#### RXOR

<table>
<thead>
<tr>
<th>RXOR</th>
<th>3082</th>
</tr>
</thead>
</table>

#### RAND

<table>
<thead>
<tr>
<th>RAND</th>
<th>3083</th>
</tr>
</thead>
</table>

#### Word Format

$1 = \text{BASE} \lor s_r \lor d_r$
Table C-1. Instruction Set with Bit Patterns (Continued)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Base</th>
<th>fc</th>
<th>FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFLG</td>
<td>0800</td>
<td>0000</td>
<td>8</td>
</tr>
<tr>
<td>PFLG</td>
<td>0880</td>
<td>0100</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0200</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0300</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0400</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0500</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0600</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0700</td>
<td>15</td>
</tr>
</tbody>
</table>

**Word Format**

\[ 1 = \text{BASE} \lor \text{fc} \lor \text{ctl} \]

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Base</th>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HALT</td>
<td>0000</td>
<td>RTI</td>
<td>0100</td>
</tr>
<tr>
<td>PUSHF</td>
<td>0080</td>
<td>RTS</td>
<td>0200</td>
</tr>
<tr>
<td>PULLF</td>
<td>0280</td>
<td>JSRI</td>
<td>0380</td>
</tr>
</tbody>
</table>

**Word Format**

\[ 1 = \text{BASE} \lor \text{ctl} \]

The instruction is formed by the inclusive Or of each field. For example, the instruction RADD 2, 3 is coded as X'3C00'.

For instructions that use the CTL field, only the first 7 bits (bits 0 through 6) are considered.

Examples of coding follow:

**Example 1**

RADD 2, 3

\[
\begin{align*}
\text{BASE} & = 3000 \\
\text{sr} & = 0800 \\
\text{dr} & = 0300 \\
\text{INSTRUCTION} & = 3C00 \\
\end{align*}
\]

Comments

Add AC2 to AC3.

**Example 2**

JMP-1 (3)

\[
\begin{align*}
\text{BASE} & = 2000 \\
\text{xr} & = 0300 \\
\text{disp} & = 00FF \\
\text{INSTRUCTION} & = 23FF \\
\end{align*}
\]

Comments

Jump to the location specified by the index register AC3 modified by the displacement-1.

**Example 3**

SHR 0, 1

\[
\begin{align*}
\text{BASE} & = 5C00 \\
\text{r} & = 0000 \\
\text{disp} & = 00FF \\
\text{INSTRUCTION} & = 5CFF \\
\end{align*}
\]

Comments

Shift the contents of AC0 one place to the right.
### Appendix D

CONVERSION TABLES

#### Table D-1. Positive Powers of Two

<table>
<thead>
<tr>
<th>n</th>
<th>2^n</th>
<th>n</th>
<th>2^n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>51</td>
<td>22517 99813 68524 8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>52</td>
<td>45035 99627 37049 6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>53</td>
<td>90071 99254 74099 2</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>54</td>
<td>18014 39850 94819 84</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>55</td>
<td>36028 79701 86559 68</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>56</td>
<td>72057 59403 79279 36</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>57</td>
<td>14411 51880 75855 872</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>58</td>
<td>28823 05761 51711 744</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>59</td>
<td>57646 07253 03423 488</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
<td>60</td>
<td>11529 21504 60884 6976</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
<td>61</td>
<td>23058 40009 21389 3952</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
<td>62</td>
<td>46116 80018 42738 7904</td>
</tr>
<tr>
<td>13</td>
<td>8192</td>
<td>63</td>
<td>92232 60036 85457 5808</td>
</tr>
<tr>
<td>14</td>
<td>16384</td>
<td>64</td>
<td>18446 74407 37055 51816</td>
</tr>
<tr>
<td>15</td>
<td>32768</td>
<td>65</td>
<td>36893 48814 74191 02522</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>66</td>
<td>73786 97629 48382 06464</td>
</tr>
<tr>
<td>17</td>
<td>131072</td>
<td>67</td>
<td>14757 39525 89676 41292 8</td>
</tr>
<tr>
<td>18</td>
<td>262144</td>
<td>68</td>
<td>29514 79051 79352 82558 6</td>
</tr>
<tr>
<td>19</td>
<td>524288</td>
<td>69</td>
<td>59029 58103 58705 61371 2</td>
</tr>
<tr>
<td>20</td>
<td>1048576</td>
<td>70</td>
<td>11805 91620 71741 13034 24</td>
</tr>
<tr>
<td>21</td>
<td>2097152</td>
<td>71</td>
<td>23611 83241 43482 26068 48</td>
</tr>
<tr>
<td>22</td>
<td>4194304</td>
<td>72</td>
<td>47223 66482 86964 52136 96</td>
</tr>
<tr>
<td>23</td>
<td>8388608</td>
<td>73</td>
<td>94487 32865 73929 84271 92</td>
</tr>
<tr>
<td>24</td>
<td>16777216</td>
<td>74</td>
<td>18889 46203 14785 80584 784</td>
</tr>
<tr>
<td>25</td>
<td>33554432</td>
<td>75</td>
<td>37778 93186 29571 61709 568</td>
</tr>
<tr>
<td>26</td>
<td>67108864</td>
<td>76</td>
<td>75557 86372 59143 23419 136</td>
</tr>
<tr>
<td>27</td>
<td>134217728</td>
<td>77</td>
<td>15111 57274 51828 64683 8272</td>
</tr>
<tr>
<td>28</td>
<td>268435456</td>
<td>78</td>
<td>30223 25840 50507 25367 5244</td>
</tr>
<tr>
<td>29</td>
<td>536870912</td>
<td>79</td>
<td>60446 29098 07314 58755 3088</td>
</tr>
<tr>
<td>30</td>
<td>1073741824</td>
<td>80</td>
<td>12089 25219 71402 91747 60176</td>
</tr>
<tr>
<td>31</td>
<td>2147483648</td>
<td>81</td>
<td>24178 51639 22925 83494 12352</td>
</tr>
<tr>
<td>32</td>
<td>4294967296</td>
<td>82</td>
<td>48357 03278 45851 69988 24704</td>
</tr>
<tr>
<td>33</td>
<td>8589934592</td>
<td>83</td>
<td>96714 06556 91703 33976 49408</td>
</tr>
<tr>
<td>34</td>
<td>17179869184</td>
<td>84</td>
<td>19342 18311 38340 67695 29881 6</td>
</tr>
<tr>
<td>35</td>
<td>34359738368</td>
<td>85</td>
<td>38685 62222 76681 35950 19963</td>
</tr>
<tr>
<td>36</td>
<td>68719477376</td>
<td>86</td>
<td>77371 25245 53362 67811 18526 4</td>
</tr>
<tr>
<td>37</td>
<td>13743995544</td>
<td>87</td>
<td>15474 25049 10672 33436 25610 28</td>
</tr>
<tr>
<td>38</td>
<td>27487990088</td>
<td>88</td>
<td>30948 50098 21845 65872 47810 56</td>
</tr>
<tr>
<td>39</td>
<td>55975981776</td>
<td>89</td>
<td>61897 00196 42900 13744 95621 12</td>
</tr>
<tr>
<td>40</td>
<td>111951963552</td>
<td>90</td>
<td>12379 40039 28358 02748 99124 224</td>
</tr>
<tr>
<td>41</td>
<td>2199039270208</td>
<td>91</td>
<td>24758 80078 57076 50497 98248 448</td>
</tr>
<tr>
<td>42</td>
<td>4398078410416</td>
<td>92</td>
<td>49517 60157 14112 10995 96496 806</td>
</tr>
<tr>
<td>43</td>
<td>8796156820832</td>
<td>93</td>
<td>99035 20314 28304 21991 92993 792</td>
</tr>
<tr>
<td>44</td>
<td>1759231364064</td>
<td>94</td>
<td>19807 40628 85600 84398 38598 7584</td>
</tr>
<tr>
<td>45</td>
<td>3518462728128</td>
<td>95</td>
<td>39614 81255 71321 68768 77197 5168</td>
</tr>
<tr>
<td>46</td>
<td>7036812545256</td>
<td>96</td>
<td>79228 16251 42643 37593 54395 0336</td>
</tr>
<tr>
<td>47</td>
<td>14073250900512</td>
<td>97</td>
<td>15845 63250 28528 67518 70879 00672</td>
</tr>
<tr>
<td>48</td>
<td>28147497800524</td>
<td>98</td>
<td>31691 26300 57057 53057 41758 01344</td>
</tr>
<tr>
<td>49</td>
<td>56294995600548</td>
<td>99</td>
<td>63382 53000 14114 70074 85316 02688</td>
</tr>
<tr>
<td>50</td>
<td>112589912001096</td>
<td>100</td>
<td>12676 106000 22822 94014 96703 20537 6</td>
</tr>
<tr>
<td>101</td>
<td>25353 01200 45645 88029 93406 41075 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table D-2. Negative Powers of Two

<table>
<thead>
<tr>
<th>n</th>
<th>$2^{-n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
</tr>
<tr>
<td>4</td>
<td>0.0625</td>
</tr>
<tr>
<td>5</td>
<td>0.03125</td>
</tr>
<tr>
<td>6</td>
<td>0.015625</td>
</tr>
<tr>
<td>7</td>
<td>0.0078125</td>
</tr>
<tr>
<td>8</td>
<td>0.00390625</td>
</tr>
<tr>
<td>9</td>
<td>0.001953125</td>
</tr>
<tr>
<td>10</td>
<td>0.0009765625</td>
</tr>
<tr>
<td>11</td>
<td>0.00048828125</td>
</tr>
<tr>
<td>12</td>
<td>0.000244140625</td>
</tr>
<tr>
<td>13</td>
<td>0.0001220703125</td>
</tr>
<tr>
<td>14</td>
<td>0.00006103515625</td>
</tr>
<tr>
<td>15</td>
<td>0.000030515625</td>
</tr>
<tr>
<td>16</td>
<td>0.0000152578125</td>
</tr>
<tr>
<td>17</td>
<td>0.00000762933484375</td>
</tr>
<tr>
<td>18</td>
<td>0.000003814697265625</td>
</tr>
<tr>
<td>19</td>
<td>0.0000019073486328125</td>
</tr>
<tr>
<td>20</td>
<td>0.00000095367431640625</td>
</tr>
<tr>
<td>21</td>
<td>0.000000476837158203125</td>
</tr>
<tr>
<td>22</td>
<td>0.0000002384185791015625</td>
</tr>
<tr>
<td>23</td>
<td>0.00000011920928955078125</td>
</tr>
<tr>
<td>24</td>
<td>0.000000059604644775390625</td>
</tr>
<tr>
<td>25</td>
<td>0.0000000298023223876953125</td>
</tr>
<tr>
<td>26</td>
<td>0.00000001490116119384765625</td>
</tr>
<tr>
<td>27</td>
<td>0.000000007450380596923828125</td>
</tr>
<tr>
<td>28</td>
<td>0.0000000037252902984619140625</td>
</tr>
<tr>
<td>29</td>
<td>0.00000000186264514923095703125</td>
</tr>
<tr>
<td>30</td>
<td>0.000000000931322574615478515625</td>
</tr>
<tr>
<td>31</td>
<td>0.0000000004656612873077392578125</td>
</tr>
<tr>
<td>32</td>
<td>0.00000000023283064365386962890625</td>
</tr>
<tr>
<td>33</td>
<td>0.00000000016415321826034814453125</td>
</tr>
<tr>
<td>34</td>
<td>0.000000000082076609134674072265625</td>
</tr>
<tr>
<td>35</td>
<td>0.0000000000410383045673370361328125</td>
</tr>
<tr>
<td>36</td>
<td>0.000000000021551915228366851806560625</td>
</tr>
<tr>
<td>37</td>
<td>0.00000000001072759576141834259033203125</td>
</tr>
<tr>
<td>38</td>
<td>0.000000000005363797880709171295166015625</td>
</tr>
<tr>
<td>39</td>
<td>0.0000000000026818989403545850475830078125</td>
</tr>
<tr>
<td>40</td>
<td>0.000000000001349447017729282379150390625</td>
</tr>
<tr>
<td>41</td>
<td>0.00000000000067473505864641189575953125</td>
</tr>
<tr>
<td>42</td>
<td>0.00000000000033736724433232059478759765625</td>
</tr>
<tr>
<td>43</td>
<td>0.00000000000016868377216160297393798828125</td>
</tr>
<tr>
<td>44</td>
<td>0.00000000000008434188608081486968994140625</td>
</tr>
<tr>
<td>45</td>
<td>0.000000000000042170494040407434844970703125</td>
</tr>
<tr>
<td>46</td>
<td>0.00000000000002108547152020037174224853515625</td>
</tr>
<tr>
<td>47</td>
<td>0.00000000000001054273576010018587124267578125</td>
</tr>
<tr>
<td>48</td>
<td>0.00000000000000522713675880500929355621337890625</td>
</tr>
<tr>
<td>49</td>
<td>0.00000000000000257358689480250484677810669453125</td>
</tr>
<tr>
<td>50</td>
<td>0.00000000000000128817841970012523233890533447265625</td>
</tr>
</tbody>
</table>
Table D-3. Hexadecimal and Decimal Integer Conversion Table

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Hex</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>268,435,456</td>
<td>1</td>
<td>16,777,216</td>
<td>1</td>
<td>1,048,576</td>
<td>1</td>
<td>65,536</td>
<td>1</td>
<td>4,096</td>
<td>1</td>
<td>256</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>336,703,122</td>
<td>2</td>
<td>33,554,432</td>
<td>2</td>
<td>2,048,352</td>
<td>2</td>
<td>131,072</td>
<td>2</td>
<td>8,192</td>
<td>2</td>
<td>512</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>405,306,368</td>
<td>3</td>
<td>50,331,648</td>
<td>3</td>
<td>3,145,728</td>
<td>3</td>
<td>196,608</td>
<td>3</td>
<td>12,288</td>
<td>3</td>
<td>768</td>
<td>3</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>474,611,642</td>
<td>4</td>
<td>67,108,864</td>
<td>4</td>
<td>4,194,304</td>
<td>4</td>
<td>267,144</td>
<td>4</td>
<td>16,384</td>
<td>4</td>
<td>1,024</td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>543,916,388</td>
<td>5</td>
<td>80,486,016</td>
<td>5</td>
<td>5,242,880</td>
<td>5</td>
<td>327,680</td>
<td>5</td>
<td>20,480</td>
<td>5</td>
<td>1,280</td>
<td>5</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>613,221,176</td>
<td>6</td>
<td>100,663,296</td>
<td>6</td>
<td>6,291,456</td>
<td>6</td>
<td>393,216</td>
<td>6</td>
<td>24,576</td>
<td>6</td>
<td>1,536</td>
<td>6</td>
<td>96</td>
</tr>
<tr>
<td>7</td>
<td>682,526,928</td>
<td>7</td>
<td>117,440,312</td>
<td>7</td>
<td>7,340,032</td>
<td>7</td>
<td>458,752</td>
<td>7</td>
<td>28,672</td>
<td>7</td>
<td>1,792</td>
<td>7</td>
<td>112</td>
</tr>
<tr>
<td>8</td>
<td>741,832,704</td>
<td>8</td>
<td>134,217,728</td>
<td>8</td>
<td>8,388,608</td>
<td>8</td>
<td>524,288</td>
<td>8</td>
<td>32,768</td>
<td>8</td>
<td>2,048</td>
<td>8</td>
<td>128</td>
</tr>
<tr>
<td>9</td>
<td>801,138,480</td>
<td>9</td>
<td>150,994,944</td>
<td>9</td>
<td>9,437,184</td>
<td>9</td>
<td>589,824</td>
<td>9</td>
<td>36,864</td>
<td>9</td>
<td>2,304</td>
<td>9</td>
<td>144</td>
</tr>
<tr>
<td>B</td>
<td>918,647,600</td>
<td>B</td>
<td>184,549,376</td>
<td>B</td>
<td>11,534,336</td>
<td>B</td>
<td>720,896</td>
<td>B</td>
<td>45,056</td>
<td>B</td>
<td>2,816</td>
<td>B</td>
<td>176</td>
</tr>
<tr>
<td>C</td>
<td>977,893,224</td>
<td>C</td>
<td>201,316,592</td>
<td>C</td>
<td>12,582,912</td>
<td>C</td>
<td>786,432</td>
<td>C</td>
<td>49,152</td>
<td>C</td>
<td>3,072</td>
<td>C</td>
<td>192</td>
</tr>
<tr>
<td>D</td>
<td>1,036,138,880</td>
<td>D</td>
<td>218,103,056</td>
<td>D</td>
<td>13,631,488</td>
<td>D</td>
<td>851,968</td>
<td>D</td>
<td>53,248</td>
<td>D</td>
<td>3,328</td>
<td>D</td>
<td>208</td>
</tr>
<tr>
<td>E</td>
<td>1,095,385,536</td>
<td>E</td>
<td>234,881,024</td>
<td>E</td>
<td>14,680,064</td>
<td>E</td>
<td>917,504</td>
<td>E</td>
<td>57,344</td>
<td>E</td>
<td>3,584</td>
<td>E</td>
<td>224</td>
</tr>
<tr>
<td>F</td>
<td>1,154,632,192</td>
<td>F</td>
<td>251,658,240</td>
<td>F</td>
<td>15,728,640</td>
<td>F</td>
<td>983,040</td>
<td>F</td>
<td>61,440</td>
<td>F</td>
<td>3,840</td>
<td>F</td>
<td>240</td>
</tr>
</tbody>
</table>

TO CONVERT HEXADECIMAL TO DECIMAL

1. Locate the column of decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number that corresponds to the position of the hexadecimal digit or letter.

2. Repeat step 1 for the next (second from the left) position.

3. Repeat step 1 for the units (third from the left) position.

4. Add the numbers selected from the table to form the decimal number.

To convert integer numbers greater than the capacity of table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position.

Example: $D_{16} = 3380_{10}$

\[
\begin{array}{c}
D = 13 \\
\times 16 \\
208
\end{array}
\]

\[
\begin{array}{c}
3 = +3 \\
211 \\
\times 16 \\
3376
\end{array}
\]

\[
\begin{array}{c}
4 = +4 \\
3380
\end{array}
\]

EXAMPLE

Conversion of Hexadecimal Value $D_{16}$

1. $D = 3328$

2. $3 = 48$

3. $4 = 4$

4. Decimal 3380

TO CONVERT DECIMAL TO HEXADECIMAL

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.

(b) Record the hexadecimal of the column containing the selected number.

(c) Subtract the selected decimal from the number to be converted.

2. Using the remainder from step 1(c) repeat all of step 1 to develop the second position of the hexadecimal (and a remainder).

3. Using the remainder from step 2 repeat all of step 1 to develop the units position of the hexadecimal.

4. Combine terms to form the hexadecimal number.

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example: $3380_{10} = X_{16}$

\[
\begin{array}{c}
16 \quad 3380 \\
\underline{16 \quad 211} \quad \text{remainder} \\
\underline{16 \quad 13} \quad 3 \\
\underline{3380_{10} - D_{16}}
\end{array}
\]

<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>Conversion of Decimal Value</th>
<th>$3380_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. D</td>
<td>3328</td>
<td>3380_{10}</td>
</tr>
<tr>
<td>2. 3</td>
<td>$-3328$</td>
<td>52</td>
</tr>
<tr>
<td>3. 4</td>
<td>$-48$</td>
<td>4</td>
</tr>
<tr>
<td>4. Hexa-decimal D16</td>
<td></td>
<td>3380_{10}</td>
</tr>
</tbody>
</table>
### Table D-4. Hexadecimal and Decimal Fraction Conversion Table

<table>
<thead>
<tr>
<th></th>
<th>Hex</th>
<th>Decimal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Hex</th>
<th>Decimal Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.0000</td>
<td>.0000</td>
<td>2</td>
<td>.0000</td>
<td>0</td>
<td>.0000</td>
<td>3</td>
<td>.0000</td>
</tr>
<tr>
<td>.1</td>
<td>.0625</td>
<td>.01</td>
<td>.2</td>
<td>.1250</td>
<td>.02</td>
<td>.0078</td>
<td>.3</td>
<td>.1875</td>
</tr>
<tr>
<td>.4</td>
<td>.2500</td>
<td>.04</td>
<td>.5</td>
<td>.3125</td>
<td>.05</td>
<td>.0195</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.7</td>
<td>.4375</td>
<td>.07</td>
<td>.8</td>
<td>.5000</td>
<td>.08</td>
<td>.0312</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>.6250</td>
<td>.0A</td>
<td>B</td>
<td>.6875</td>
<td>.0B</td>
<td>.0429</td>
<td>C</td>
<td>.7500</td>
</tr>
<tr>
<td>D</td>
<td>.8125</td>
<td>.0D</td>
<td>E</td>
<td>.8750</td>
<td>.0E</td>
<td>.0546</td>
<td>F</td>
<td>.9375</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6250</td>
<td></td>
<td>1250</td>
<td></td>
<td>1875</td>
<td></td>
<td>2500</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3125</td>
<td></td>
<td>3125</td>
<td></td>
<td>3750</td>
<td></td>
<td>4375</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5000</td>
<td></td>
<td>6250</td>
<td></td>
<td>6875</td>
<td></td>
<td>7500</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6875</td>
<td></td>
<td>8125</td>
<td></td>
<td>8750</td>
<td></td>
<td>9375</td>
<td></td>
</tr>
</tbody>
</table>

**TO CONVERT ABC HEXADECIMAL TO DECIMAL**

Find .A in position 1 .6250  
Find .0B in position 2 .0429 6875  
Find .00C in position 3 .0029 2968 7500  
ABC Hex is equal to .6708 9843 7500

### Table D-5. Integer Conversion Table

**POWERS OF 16 TABLE**

*Example: 268,435,456₁₀ = (2.68435456 × 10⁸)₁₀ = 1,000 0000₁₆ = (10⁸)₁₆*

<table>
<thead>
<tr>
<th>16ⁿ</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>65</td>
<td>4</td>
</tr>
<tr>
<td>1024</td>
<td>5</td>
</tr>
<tr>
<td>16,384</td>
<td>6</td>
</tr>
<tr>
<td>268,435,456</td>
<td>7</td>
</tr>
<tr>
<td>4,294,967,296</td>
<td>8</td>
</tr>
<tr>
<td>68,719,476,736</td>
<td>9</td>
</tr>
<tr>
<td>1,099,511,627,776</td>
<td>10 = A</td>
</tr>
<tr>
<td>17,592,186,044,416</td>
<td>11 = B</td>
</tr>
<tr>
<td>281,474,976,710,656</td>
<td>12 = C</td>
</tr>
<tr>
<td>4,503,599,627,370,496</td>
<td>13 = D</td>
</tr>
<tr>
<td>72,057,594,037,927,936</td>
<td>14 = E</td>
</tr>
<tr>
<td>1,152,921,504,606,846,976</td>
<td>15 = F</td>
</tr>
</tbody>
</table>

Decimal Values
NEGATIVE HEXADECIMAL NUMBERS

The IMP-16 maintains negative numbers in twos-complement form. To convert a number in hexadecimal notation to its twos-complement equivalent, subtract the number from $2^n$ expressed in hexadecimal form. The number "n" is the number of binary bits in the computer word. For example, if the computer uses a 16-bit word, the number "n" is equal to 16. Thus, the negative of $1245_{16}$ is derived as follows:

\[
\begin{array}{cc}
10000 & 1000000000000000 \\
-1245 & -0001001001001011 \\
EDBB & 1110110110111011 \\
\end{array}
\]

Note that a hexadecimal number will be negative in the IMP-16 computer if the left most digit is 8, 9, A, B, C, D, E, or F. Thus, $FACE_{16}$ is equal to $1111101011001110$; the twos complement is:

\[
\begin{array}{cc}
10000 & 1000000000000000 \\
FACE & 1111101011001110 \\
532 & 0000010100110010 \\
\end{array}
\]
Appendix E

EDIT16 SYMBOL MEANINGS AND USAGE

The following symbols and their meanings are used in the examples associated with chapter 5, EDIT16.

- All underlined, upper-case characters represent Teletype keyboard, user entries.
- Encircled characters or combinations of characters represent nonprinting, or control characters. If underlined, they represent user initiated input; if not underlined, they represent computer generated characters.
- CTRL/X represents a user input where the Teletype CONTROL key is pressed and held, while the X key is pressed. Similarly, any key (for a printing character) may be pressed in combination with the CONTROL key. The symbol then is CTRL/(key).

Table E-1 lists the symbols used in this text for the EDIT16 program.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>←</td>
<td>Back arrow. Indicates Teletype keyboard error correction of previous character, or multiples of characters (depending on number of arrows used).</td>
</tr>
<tr>
<td>↑</td>
<td>Echo for CNTL/X input, indicates position of character to be deleted.</td>
</tr>
<tr>
<td>←</td>
<td>Prompt, from EDIT16, shows readiness to accept a new line.</td>
</tr>
<tr>
<td>?</td>
<td>Prompt, from EDIT16, shows readiness to accept command.</td>
</tr>
<tr>
<td>CR</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>LF</td>
<td>Line Feed</td>
</tr>
<tr>
<td>CTRL/A</td>
<td>Start of character insert operation.</td>
</tr>
<tr>
<td>CTRL/D</td>
<td>Truncates the current line.</td>
</tr>
<tr>
<td>CTRL/Q</td>
<td>Aborts the current line modification operation.</td>
</tr>
<tr>
<td>CTRL/X</td>
<td>Deletes the character in the corresponding position in the previous line.</td>
</tr>
<tr>
<td>CTRL/Z</td>
<td>Carriage Tab feature</td>
</tr>
</tbody>
</table>

Table E-1. EDIT16 Symbols
This change is effective immediately for the following programs:

- PROMP 4300308B
- DEBUG 4300112C
- EDIT16 4300332B
- STTYIO 4300158C

Page ii. PREFACE. Add to paragraph 1:

To facilitate use of these programs and to minimize loading time, DEBUG, PROMP, and EDIT16 are assembled in absolute format so that they may be loaded into the user's environment using either GENLDR or one of the available absolute loaders.

Page 1-1. Section 1.2. Insert before paragraph 1:

DEBUG is assembled relative to location X'10 in Base Page and location X'210 in Top Sector. This means that DEBUG may be loaded into the user's environment either by an absolute loader (see sections 2.2 and 2.3 of this manual) or by GENLDR (see section 2.5 of this manual). If loaded by an absolute loader, DEBUG will occupy Base Page locations X'10 through X'16, and Top Sector locations X'210 through X'654.

If DEBUG is loaded by GENLDR, DEBUG will appear to occupy Base Page locations 0 through X'F and Top Sector locations 0 through X'20F as well as its normal memory areas. If the GENLDR commands

```
!OBS 0
!OTS 0
```

are executed prior to loading DEBUG, the program will occupy the same memory locations as if it were loaded absolutely. If it is desired to use the X'10 locations preceding the DEBUG Base Page or the X'210 locations preceding the DEBUG Top Sector, appropriate !OTS and !OBS commands should be executed. As an example, DEBUG Top Sector might be located at X'1000 by using a

```
!OTS DF0 (X'1000 - X'210)
```

command. In essence, the specified Top Sector origin must precede the desired Top Sector origin by X'210 locations.

Page 1-1. Section 1.2. Existing paragraph 1, delete the first sentence, reading:

DEBUG is a relocatable ... other relocatable program.

Page 1-1. Section 1.2.1, change to read:

The following memory is needed to execute DEBUG:

- Top Sector: X'445 or 1093 words.
- Base Page: 7 words.
Page 3-2.  Section 3.2.3, sentence 1. Change "STTYIO (zero)" to read "STTYIO (letter)".

Page 3-2.  Section 3.3. Change to read:

STTYIO is assembled as a relocatable load module and must be loaded by GENLDR. (See section 2.5 of this manual.)

Page 4-1.  Section 4.1, replace last sentence by:

PROMP output tapes may be used to program either MM5203 2K PROMs in a 256-by-8 structure or MM5204 4K PROMs in a 512-by-8 structure. If MM5203 PROMs are used, two PROMs are required to make up a 256-word by 16-bit memory page.

Page 4-1.  Section 4.1, add paragraph 2:

If the BC tape option is selected, PROMP will calculate a 16-bit checksum of the tape and punch it at the end of the tape as four hexadecimal characters in ANSI code.

Page 4-1.  Section 4.2.1. Change to read:

PROMP is assembled relative to location X'250 in Top Sector. This means that PROMP may be loaded into the user's environment either by an absolute loader (see sections 2.2 and 2.3 of this manual) or by GENLDR (see section 2.5 of this manual). If loaded by an absolute loader, PROMP will occupy memory locations X'250 through X'9CC.

If PROMP is loaded by GENLDR, it will appear to occupy Top Sector locations 0 through X'24F as well as its normal memory area. If the GENLDR command

!OTS 0

is executed prior to loading PROMP, the program will occupy the same memory locations as if it were loaded absolutely. If the user desires to use the X'250 locations preceding the PROMP Top Sector, an appropriate !OTS command should be executed. For example, PROMP might be loaded at X'1000 by using a

!OTS D80 (X'1000 - X'250)

command. In essence, the specified Top Sector origin must precede the desired Top Sector origin by X'250 locations.

Page 4-1.  Section 4.2.2. Change to read:

PROMP requires 780 16 words of top sector memory.

Page 4-1 through 4-3, section 4.2.3, replace with the following:

4.2.3 Program Messages

To guide the user in the use of PROMP, query messages are typed. There are four general messages corresponding to the four possible program states. Initially, PROMP types:

NSC IMP-16 FIRMWARE PAPER TAPE GENERATOR
OUTPUT TYPE:
The user must then respond with one of the following codes:

- PN - For PN format PROM tape.
- BC - For binary (complemented) format PROM tape.
- OM - For card RLM to tape RLM conversion.

All responses must be terminated by a carriage return. For the OM option, PROMP types the following message and goes into an output state:

```
MAKE CARD READER READY
TURN PUNCH ON
HIT ANY KEY TO START
```

At this point, RLM records are converted from card to paper tape. After processing the END record, PROMP goes into a wait state. The user may now turn off the Teletype punch. Typing any key returns the PROMP program to the initial state.

In either PN or BC options, PROMP goes into an input state. The following message is typed:

```
INPUT DEVICE:
```

The user may respond with one of the following codes:

- CR - Card Reader
- PT - Teletype
- ME - Memory

**ME Option.** If this device is selected, PROMP then types:

```
SPECIFY MEMORY --
```

The user must then type the message range, where the user program is located. A range is designated by the start address and the last address, delimited by a colon (:); the range must be in blocks of $256_{10}$ or $512_{10}$ words.

Example:

```
SPECIFY MEMORY -- FF00:FFFF
```

**CR Option.** PROMP types the following message:

```
MAKE CARD READER READY
TO LOAD LM
```

The card reader must be turned on at this point. The LM will be immediately read into memory.

**PT Option.** If this device is selected, PROMP types the message:

```
MAKE TAPE READER READY
TO LOAD LM
```

The LM will be loaded immediately into memory. After the loading process, when the END record is recognized, PROMP types:

```
TURN READER OFF
```

The user must turn off the reader and press any key to initiate the output process.
Mode and Byte Requests. PROMP may be used for punching programming tapes for either the MM5203 2K PROM or the MM5204 4K PROM. After loading the LM, PROMP will query the user as to which PROM is being used.

SET MODE:
TAPE FOR MM5203 2K PROM - TYPE 2
TAPE FOR MM5204 4K PROM - TYPE 4

TYPE:

The MM5203 PROM is structured in a 256-word by 8-bit format, while the MM5204 PROM is structured in a 512-word by 8-bit format. PROMP allows the user to program PROMs for either 256- or 512-word memory regions using the MM5203, and for a 512-word region using the MM5204. In order to do this, the user must respond to the query

BYTE:

with one or more of the options, L, R, LL, LR, HL, and HR according to the following tables:

### MM5203 2K PROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Range</th>
<th>BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N+255</td>
<td>LL</td>
</tr>
<tr>
<td>N+256</td>
<td>N+511</td>
<td>LR</td>
</tr>
</tbody>
</table>

### MM5204 4K PROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Range</th>
<th>BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N+511</td>
<td>L</td>
</tr>
</tbody>
</table>

If more than one option is desired, each entry may be separated either by commas or by spaces. For a particular PROM, if all options are desired, the user may respond with a carriage return. After the option is typed, PROMP comes back with the message:

**TURN PUNCH ON**

**HIT ANY KEY TO START**

After the output process, PROMP goes into its last state, the wait state. This wait state is provided so that the user can turn off the punch before any message is typed by PROMP. To return to the initial state, the user must press any key on the keyboard.

Typing CTRL/D when PROMP is waiting for a response transfers the user to the initial state. Leaders/trailers are punched automatically. If PN is chosen as an output type code, pressing any key during the output process causes an interrupt of the current output option and initiates processing of the next output option. Under certain conditions, the key must be struck several times in order to cause an interrupt.
Page 4-4. Section 4.4, following table 4-1, add:

Figure 4-1 illustrates the PROMP operational sequence in flowchart form.

Add figure 4-1 (which is on next page, 4-4A).

Page 5-7. Section 5.5.1. Insert before paragraph 1:

EDIT16 is assembled relative to location 0 in Base Page and location X'120 in Top Sector. This means that EDIT16 may be loaded either by an absolute loader (see sections 2.2 and 2.3 of this manual) or by GENLDR (see section 2.5 of this manual). If loaded by an absolute loader, EDIT16 will occupy the Base Page locations 0 through X'78, and Top Sector locations X'120 through X'7FB.

If EDIT16 is loaded by GENLDR, it will appear to occupy Top Sector locations 0 through X'11F as well as its normal memory areas. If the GENLDR commands,

\begin{verbatim}
!OBS 0
!OTS 0
\end{verbatim}

are executed prior to loading EDIT16, the program will occupy the same memory locations as if it were loaded absolutely. If it is desired to use the X'120 locations preceding the EDIT16 Top Sector, the appropriate !OTS command should be executed. As an example, EDIT16 Top Sector might be located at X'1000 by using a

\begin{verbatim}
!OTS EE0 (X'1000 - X'120)
\end{verbatim}

command. In essence, the specified top sector origin must precede the desired top sector origin by X'120 locations.

Page 5-7. Section 5.5.1. Existing paragraph 1: delete the first sentence, reading:

"The EDIT16 program ... starts automatically".

Change the second sentence to read:

Once loaded, the following messages are typed:

\begin{verbatim}
NSC EDIT 16 REV X
MEMORY:
\end{verbatim}

Effective immediately, all references to "Relocatable Load Module" (RLM) shall be changed to "Load Module" (LM). Load Modules may be either relocatable or absolute, depending on context or usage.
Figure 4-1. Operation Of The PROMP Program
This change is effective immediately.

Page 5-1. Section 4.2. Add to first paragraph:

"Those commands that require text mode processing are KB, RT, RC, MD, and MS."

Page 5-2. In first line, replace "Specifies" with "Separates".

Page 5-2. Section 5.3.3. Change last sentence of paragraph to read:

"The edit buffer is located in the remainder of memory above the EDIT16 code so its size is dependent on the machine memory space."

Page 5-3. Section 5.4.2. After the first paragraph, change to read as follows:

Example:

? RT 2 
TURN READER OFF NOW

Apend two lines to the edit buffer. This message is typed after the second line is read.

"If the edit buffer becomes full when entering lines using KB, RT, and RC commands, the message

BUFFER FULL

is typed and EDIT16 goes into command mode. Again, an incomplete line is not entered to the buffer."

"If a BUFFER FULL message is typed, the user may recover by punching a few lines from the beginning of the buffer onto paper tape and, then, deleting those lines from the buffer."

Page 5-4. Section 5.4.8. Add to first paragraph:

"EDIT16 will type the line to be modified and, then, will prompt for modifications by typing ALTERS?."  

Page 5-4. Section 5.4.8. Add after first paragraph:

"Typing \; as the first character after ALTERS? causes EDIT16 to terminate the modification and to prompt for a command."

Page 5-5. Add after sentence following second example of Section 5.4.8:

"If CTRL/Z is followed by another control character, EDIT16 will reprompt for ALTERS?."  

Page 5-5. After the third example of Section 5.4.8 between ALTERS? and "Typing a CRTL/Q aborts the current line modification.", add the following:

"Attempting to add a line feed (LF) character will cause loss of a character from the original text."

"EDIT16 limits the maximum line length to 65 characters. If the user attempts to exceed this limit by inserting characters, the insertion will be aborted. The user, then, may delete characters from the end of the line, re-perform the insertion, and then add the deleted characters to the next line or insert them following the current line."
Page 5–6. Section 5.4.9. Immediately preceding 5.4.10 (that is, as the last of Section 5.4.9), add the following:

"The user will be prompted with ALTERS? for each line containing the specified string. If alteration of a particular line is not necessary, he may respond with a \[R\] immediately following the prompt."

"If it is desired to terminate modifications, the user may type \[CTRL/Q\]."

Page 5–7. Section 5.4.15. Add new paragraph:

"When entering data via EDIT16, \[CTRL/I\] should be used to skip to the next tab setting."

Page 5–8. Add the following as Section 5.5.3:

"5.5.3 Restarting EDIT16

EDIT16 may be restarted by pressing INITIALIZE, setting the PC to the value contained in the data item RINIT1 (see listing), and pressing RUN."

Page C-1. FORMAT OF INSTRUCTIONS delete the words "to Memory" for second format, now "Register to Memory."
CHANGE NOTICE NUMBER 3
(28 February 1975)

Publication No. 4200025B
Order No. IMP-16S/025YB

IMP-16 Utilities
Reference Manual

This change is effective immediately for the following program:

PROMP 4300308C

Only changes incurred by Change Notice Number 1 are affected by this change notice. Thus, references to page numbers and sections apply to Change Notice Number 1. Changes or additions are underscored.

NOTE
Information in Change Notice Number 1 pertaining to other programs listed therein is still in effect.

Page 2 — affecting manual Section 4.1, addition of paragraph 2.3 should state:

If the BC tape option is selected, PROMP will calculate a 32-bit checksum of the tape and punch it at the end of the tape as eight hexadecimal characters in ANSI code.

Page 3 — affecting replacement of Section 4.2.3, Program Messages, of the manual, add second sentence so fourth paragraph states:

At this point, RLM records are converted from card to paper tape. GENLDR commands such as !RLM are ignored. After processing the END record, PROMP goes into a wait state. The user may now turn off the Teletype punch. Typing any key returns the PROMP program to the initial state.
This change notice affects the IMP-16 General Loader Program, GENLDR.

Replace the existing section 2.5 (through 2.5.19) of the subject manual with the attached section 2.5.

2.5 GENLDR (IMP-16 GENERAL LOADER)

GENLDR is a stand-alone IMP-16 program that reads one or more load modules (LMs) produced by the IMP-16 assembler, performs relocation, resolves external linkages, and loads the LMs into main memory for execution. The load modules produced by the assembler may be read from either cards or paper tape.

The paragraphs that follow describe the commands to control GENLDR and the input sequences required to load an executable program into the IMP-16 main memory. Error messages and diagnostic output of GENLDR are also described.

NOTE

The prefix X' designates that the number that follows is hexadecimal.

2.5.1 Usage

GENLDR is a relocatable stand-alone IMP-16 program that may be loaded into memory by one of two absolute loaders: (1) ABSCH allows GENLDR to be input from cards and (2) ABSPT, from paper tape. Once loaded, GENLDR can accept input from either cards or paper tape; although, initially, it accepts input from the device from which it was loaded.

GENLDR occupies approximately 1568 words of memory and is typically loaded into upper memory. Programs cannot be loaded by GENLDR into memory that it occupies or uses for the symbol table it generates. However, GENLDR allows the user full use of base page. The memory layout is described in figure 2-3.

GENLDR is assembled relocatable and occupies the (relative) addresses X'09E0 through X'0FFF. It is recommended that it be loaded into the highest locations of memory available in the system, for ease of use and to enable GENLDR to remain resident while other programs are running. To load it into the top of an 8K memory, for example, first load it with the absolute loader, and then use GENLDR to relocate itself with the !OTS 1000 command. GENLDR would then reside in the memory locations X'19E0 through X'1FFF.

GENLDR is self-initializing; it may be entered at its entry point at any time. The entry point is X'0A5F in a 4K system. The value of AC3 is used to determine the initial load device: if AC3 = 0, the device is the Teletype; otherwise, it is the card reader. If GENLDR is entered after pressing initialize, the device will be the Teletype.

The IMP-16 assembler allows the user to allocate portions of his program in three ways:

- At an absolute memory location
- Relative to the origin of the base sector
- Relative to the origin of the top sector
Figure 2-3. Memory Map
Typically, absolute allocation is employed to assign locations dependent upon equipment (for example, interrupt entrance address) or to communicate with special-purpose routines. The base sector must be located such that it is contained within the first 256 ($X'100$) locations of memory and typically contains data and pointers necessary for inter-LM communication. The top sector may reside anywhere in memory (subject to the limitations mentioned above) and normally contains the main portion of the LM. Care must be exercised to ensure that an absolute sector does not overlay a previously loaded base sector or top sector. (See !OBS and !OTS commands in the following paragraphs.)

Two other limitations are imposed upon the base sector by the IMP-16 computer architecture and the method for resolving external linkages. First, any base sector variable that is referenced by an indexed instruction must be allocated to one of the first $X'7F$ locations of memory. Second, in resolving certain external linkages, GENLDR may force an indirect reference to a global variable through a pointer in the memory area $X'FF$ and downward.

The area of IMP-16L memory between locations $X'100$ and $X'11F$ is used by the control panel service routine and may not be used by the user. Above address $X'FF$, loading is limited only within the area occupied by GENLDR and the symbol table it generates. In the IMP-16P, GENLDR allows the locations used for its IMP-16L Teletype input/output to be overwritten, so the effective starting location of GENLDR is $X'0A31$. The GENLDR area may be used by the loaded program, after it receives control from GENLDR; but it is recommended that GENLDR be left in memory so that it may be reused without necessitating reloading.

As an entry point, GENLDR selects the last nonzero value specified for the set of LMs loaded. The entry point for any particular LM, if specified, appears in the end record of that LM. If the user desires, he may override the entry point selected by GENLDR by specifying the desired entry point in the !GO command (paragraph 2.5.4.13). If neither of these methods is chosen, GENLDR prints an "ENT?" error message and prompts for a new command.

### 2.5.2 GENLDR Input

GENLDR is controlled by commands and by the load module data. GENLDR reads commands and LMs from either cards or paper tape, and commands are available to switch between input devices. (See !CR and !TTY, paragraphs 2.5.4.6 and 2.5.4.7.) GENLDR does not recognize any distinction between the Teletype paper tape reader and the Teletype keyboard; therefore, the user may type in his commands at the keyboard and input the LM from paper tape. Commands entered on paper tape, the keyboard, or from the card reader are echoed back to the Teletype printer: the LM itself is not echoed.

Commands entered on punched cards must contain an exclamation point (!) in column 1. When the command is entered from the Teletype, GENLDR types the exclamation point to prompt for a command. Any additional exclamation point typed from the keyboard is ignored. If it is necessary to return to command mode while in the middle of an LM, an exclamation point may be typed on the keyboard (between LM records only), and GENLDR then prompts for a command. Any command may be given at this point, but only commands not affecting loading may be used if the LM is to be continued.

Input lines on the Teletype are terminated by a carriage return. A maximum of 72 characters is allowed in one Teletype input record; excess characters are not allowed. Only the characters from $X'20$ through $X'5F$ are allowed in Teletype commands; any other character except carriage return causes the command to be aborted and a new prompt issued. The Null ($X'00$) and Rubout ($X'7F$) characters are, however, ignored.
2.5.3 GENLDR Output

GENLDR prints information descriptive of the loading process. The title information, absolute sector limits, base sector limits, top sector limits, and entry point address of each LM are printed on the Teletype. Unless !NLM is in effect, GENLDR types the following information for each LM:

```
MNEMONIC STRING
AAAA BBBB
AS = XXXX:XXXX BS = XXXX:XXXX TS = XXXX:XXXX ENT = XXXX
```

Where:

- MNEMONIC is the name of the LM from the title record.
- STRING is the qualifying string from the title record.
- AAAA BBBB are the LM source and object checksums, respectively.
- AS = XXXX:XXXX specifies the low and high addresses of the absolute sector (if any).
- BS = XXXX:XXXX specifies the base sector origin and last base sector address (if any).
- TS = XXXX:XXXX specifies the top sector origin and last top sector address (if any).
- ENT = XXXX is the entry address from the end record (if any).

All numbers (XXXX) are printed in hexadecimal notation.

If !SY or !ER is executed, GENLDR prints symbols as follows:

```
SYMBOL XXXX F
```

Where:

- SYMBOL is the symbol name.
- XXXX is the hexadecimal address of the symbol.
- F is one of the following:
  - M — multiply-defined symbol
  - U — undefined symbol
  - blank — defined symbol

The address printed for an undefined symbol is the last address where the symbol is referenced (in a .WORD) or the base page pointer location.

If the !RA command is given to list the range of loaded addresses, or upon execution of a !GO command, the following information is typed:

```
AS = XXXX:XXXX BS = XXXX:XXXX TS = XXXX:XXXX PTR = XXXX:XXXX ENT = XXXX
```

Where the first hexadecimal value is the lowest address actually loaded in the specified sector, and the second hexadecimal value is the highest address actually loaded. PTR = XXXX:XXXX gives the limits of the loader-generated indirect pointers. These ranges are global; that is, they cover all programs loaded and are not affected by the use of the !LM or !NLM commands.

2.5.4 GENLDR Commands

All commands must begin with the exclamation point (!) in column 1 of the input record and the command string beginning in column 2. All commands have the format:

```
!CCC XXXX Comments......
```

Where CCC is the command name, of which only the first two characters are significant. XXXX is a hexadecimal value that must be separated from the command by at least one blank. Unless otherwise specified, where the term <hex-value> is used below, it represents a hexadecimal number in the range 0000 to FFFF. Leading zeros need not be specified, and only the last four significant digits are retained. If no value is specified, zero is used. Scanning of the operand is terminated by either the end of the line or encountering a character not in the hexadecimal set. Therefore, all commands may be commented if desired.
2.5.4.1 OBS — Origin Base Sector

The origin for the next base sector is set to <hex-value>. If this command is not specified, the next base sector is loaded immediately following the previous base sector (the initial base sector starts at X'10 if no OBS command is given). This command should be used to prevent loading a base sector on top of an absolute sector.

<hex-value> must be in the range 0 < <hex-value> < X'FF or in the range X'FF01 < <hex-value> < X'FFFF. If the value is outside of these ranges, the base sector overflow message is given, the command is ignored, and GENLDR prompts for a new command from the Teletype.

2.5.4.2 OTS — Origin Top Sector

The origin for the next top sector is set to <hex-value>. If this command is not specified, the next top sector is loaded immediately following the previous top sector. This command should be used to prevent loading a top sector on top of an absolute sector.

The highest value of <hex-value> is a function of the memory available, and must not cause overlaying of the locations occupied by GENLDR. If this command is not given, the first top sector is loaded at location X'120.

2.5.4.3 RLM — Read Load Module

This command may precede each LM to be loaded. The LM is loaded from the same device from which the !RLM command is entered. The !RLM command is not required if input is from the card reader, but it is necessary in the Teletype mode to start the reading of the paper tape.

2.5.4.4 CLR — Clear Memory and Restart GENLDR

All computer memory outside of GENLDR is cleared to zeroes; then, GENLDR is reinitialized and started from its entry point. Since this command clears any previously loaded information, it should only be used as the first command to GENLDR or as a means of reinitialization following an error.

2.5.4.5 RE — Restart GENLDR

GENLDR is reinitialized and restarted at its entry point. This command does not clear memory, but is otherwise the same as the !CLR command, above.

2.5.4.6 CR — Read Input from the Card Reader

Subsequent input is accepted from the card reader.

2.5.4.7 TTY — Read Input from the Teletype

Subsequent input is accepted from the Teletype. The card reader should be turned off before using the Teletype, since the IMP-16P card reader interface is also used by the high speed paper tape reader. Teletype input is accepted from either the paper tape or the keyboard, but only commands are echoed to the Teletype printer.

2.5.4.8 SY — Print the Symbol Table

The symbol table is printed upon execution of this command.
2.5.4.9 !ER — Print Symbols in Error

Multiply-defined and undefined symbols are printed when this command is read.

2.5.4.10 !RA — Type Global Loaded Range

The range of loaded addresses for each sector, plus the range of pointers generated (if any) and the current value of the entry point, is typed. This information is the same as printed on execution of a !GO command. This command may be used to determine the status of the loaded program if an error occurs.

2.5.4.11 !NLM — Inhibit Load Module Limit Printing

This command inhibits the printing of the title record information, checksums, and address ranges that are printed for each load module. Printing may be restored by the !LM command.

2.5.4.12 !LM — Enable Load Module Limit Printing

This command enables the printing of the information for each load module that was inhibited by !NLM. Printing of the information is the default state that is set upon initialization of GENLDR.

2.5.4.13 !GO — Execute the Loaded Program

The entry Point specified in the last LM loaded can be overridden by specifying the entry point address (⟨hex-value⟩). If ⟨hex-value⟩ is omitted, the last nonzero entry point specified is executed. If no nonzero entry point is specified and no value appears on the command, GENLDR prints an error message and prompts for a new command from the Teletype. Before transfer to the entry point, the global loaded program limits are printed on the Teletype.

2.5.5 Messages

The following messages may be output by GENLDR:

GENERAL LOADER (REV. X) READY — GENLDR is initialized and ready to read commands. X is the current revision level of GENLDR.

CMND? — The command is invalid or unrecognized. GENLDR prompts for a new command from the Teletype.

CHAR — An LM card contains characters other than 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Since invalid punch codes are translated as '?', this error can also occur on an invalid punch. Correct the card, reload it, and press RUN.

CKSM — Checksum error on the last record read. If the checksum field is 0000, no checksum test is made. Processing of the record is completed after 'RUN' is pressed. The read may be retried: reload the record and then press RUN.

BSOV — Base sector overflow. The run must be restarted, but the error may be corrected by proper use of the !OBS and !OTS commands.

TSOV — Top sector overflow. The run must be restarted, but the error may be corrected by proper use of the !OBS and !OTS commands. This error is caused by exceeding the upper limit of memory, X'FFFF.

AREA — Illegal attempt to load on top of the loader. Restart with valid !OBS or !OTS commands.

⟨address⟩ MEM — Memory error at ⟨address⟩: probably an attempt to load into non-existent memory.
ENT? — No entry point specified for program. GENLDR transfers control to the Teletype for a new command.

LENGTH — Record length error: the record is too long to fit in the buffer in GENLDR. The run may be restarted after the record is corrected.

EXTN — Unable to locate external symbol in Symbol Table. This error may be caused by attempting to load an LM with some missing symbol records or an erroneous patch which looks as if it is referencing an illegal external reference number. The run must be restarted.

ADDR — Addressing error. This error occurs under the following conditions and the run must be restarted:
1. Attempting to reference an indirect pointer generated by the assembler which, because of relocation, is forced to an address greater than X'7F.'
2. Using an index register in an instruction referencing a base sector variable allocated in a memory address in the range X'80 to X'FF.'
3. Attempting to use an index register in an instruction referencing an undefined external variable.
4. Referencing an undefined external variable in an instruction which either is flagged indirect already or cannot be so flagged.

SEQ — Record sequence error. The correct sequence is: (1) Title record, (2) Zero or more symbol records, (3) Zero or more data records, and (4) End record. Correct the record sequence, reload the LM, and restart the load.

SYST — System error caused by a malfunction in system software or hardware. Recovery is not possible; GENLDR should be reloaded.

SYMB — Symbol table overflow. Too many external symbols defined. The symbol table is allocated downward from the start of GENLDR; overflow will occur if an attempt is made to expand the symbol table into a region already loaded.