Your attention is invited to the last two pages of this document. The "How to Obtain Software Information" page tells you how to keep up-to-date with DEC's software. Completion and return of the "Reader's Comments" page is beneficial to both you and DEC; all comments received are acknowledged and are considered when documenting subsequent manuals.

Copyright © 1972 by Digital Equipment Corporation

DEC assumes no responsibility for the use or reliability of its software on equipment which is not supplied by DEC. The material in this document is for information purposes and is subject to change without notice.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

- CDP
- COMPUTER LABS
- COMTEX-11
- DDT
- DEC
- DECTAPE
- DIBOL
- DIGITAL
- EDUSYSTEM
- FLIP CHIP
- FOCAL
- GLC-8
- IDACS
- INDACS
- KA10
- LAB-8
- OMNIBUS
- OS/8
- OS/11
- PDP
- PHA
- QUICKPOINT
- RAD-8
- RSTS
- RSX
- SABR
- TYPESET-8
- UNIBUS

Teletype is a registered trademark of the Teletype Corporation.
PREFACE

This document describes the PAL-11S Assembly Language and Assembler (Chapter 1) and the Link-11S Linker (Chapter 2).

PAL-11S and Link-11S are stand-alone programs which are compatible subsets of the Disk Operating System (DOS) PAL-11R and Link-11 system programs. Minimum hardware requirements are an 8K PDP-11 with a teleprinter. A PC11 (high-speed paper tape punch) and/or LP11 (line printer) may be used also.

The inputs and outputs of the stand-alone programs are also compatible with the DOS counterparts. Thus, a program assembled by PAL-11R can be linked by Link-11S, and vice versa. The output of Link-11S is loadable by the DOS Loader or the Absolute Loader from the paper tape system.

PAL-11S has all the capabilities of PAL-11R except for named .CSECT's. PAL-11S has only the unnamed .CSECT and the .ASECT. PAL-11S does not have the redundant mnemonics for some of the additional assembler directives. That is, it has .IFZ, .IFNZ, .IFL, and .IFG, but does not have the equivalent names .IFEO, .IFNE, .IFLT, and .IFGT (PAL-11R has both sets).

Link-11S has most of the capabilities of Link-11 except for library searching. Link-11S has a simple initial dialogue which allows the operator to select the I/O devices, define the addresses where the program is to be linked, and to request a list of undefined globals. Link-11S does handle named and unnamed .CSECT's and therefore can link the output of PAL-11R.
CONTENTS

CHAPTER I  PAL-11S ASSEMBLER
1. Character Set
2. Statements
3. Symbols
4. Expressions
5. Assembly Location Counter
6. Relocation and Linking
7. Addressing
8. Assembler Directives
9. Operating Procedures
10. Error Codes
11. Software Error Halts

CHAPTER II  LINK-11S LIN KER
1. Introduction
2. Input and Output
3. Operating Procedures
4. Preparation

APPENDICES
A  ASCII Character Set
B  PAL-11S Assembly Language and Assembler
C  Assembling and Linking PAL-11S

INDEX
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.9</td>
<td>Immediate Mode and Deferred Immediate (Absolute) Mode</td>
</tr>
<tr>
<td>7.10</td>
<td>Relative and Deferred Relative Modes</td>
</tr>
<tr>
<td>7.11</td>
<td>Table of Mode Formats and Codes</td>
</tr>
<tr>
<td>7.12</td>
<td>Instruction Forms</td>
</tr>
<tr>
<td>8.0</td>
<td>ASSEMBLER DIRECTIVES</td>
</tr>
<tr>
<td>8.1</td>
<td>.TITLE</td>
</tr>
<tr>
<td>8.2</td>
<td>.GLOBL</td>
</tr>
<tr>
<td>8.3</td>
<td>Program Section Directives (.ASECT, .CSECT)</td>
</tr>
<tr>
<td>8.4</td>
<td>.EOT</td>
</tr>
<tr>
<td>8.5</td>
<td>.EVEN</td>
</tr>
<tr>
<td>8.6</td>
<td>.END</td>
</tr>
<tr>
<td>8.7</td>
<td>.WORD</td>
</tr>
<tr>
<td>8.8</td>
<td>.BYTE</td>
</tr>
<tr>
<td>8.9</td>
<td>.ASCII</td>
</tr>
<tr>
<td>8.10</td>
<td>.RAD50</td>
</tr>
<tr>
<td>8.11</td>
<td>.LIMIT</td>
</tr>
<tr>
<td>8.12</td>
<td>Conditioned Assembly Directives</td>
</tr>
<tr>
<td>9.0</td>
<td>OPERATING PROCEDURES</td>
</tr>
<tr>
<td>9.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>9.2</td>
<td>Loading PAL-11S</td>
</tr>
<tr>
<td>9.3</td>
<td>Initial Dialogue</td>
</tr>
<tr>
<td>9.4</td>
<td>Assembly Dialogue</td>
</tr>
<tr>
<td>9.5</td>
<td>Assembly Listing</td>
</tr>
<tr>
<td>9.6</td>
<td>Object Module Output</td>
</tr>
<tr>
<td>9.6.1</td>
<td>Global Symbol Directory</td>
</tr>
<tr>
<td>9.6.2</td>
<td>Text Block</td>
</tr>
<tr>
<td>9.6.3</td>
<td>Relocation Directory</td>
</tr>
<tr>
<td>10.0</td>
<td>ERROR CODES</td>
</tr>
<tr>
<td>11.0</td>
<td>SOFTWARE ERROR HALTS</td>
</tr>
</tbody>
</table>
PAL-11S (Program Assembly Language for the PDP-11, Relocatable, Stand Alone Version) enables you to write source (symbolic) programs using letters, numbers, and symbols which are meaningful to you. The source programs, generated either on-line using the Text Editor (ED-11), or off-line, are then assembled into object modules which are processed by the PDP-11 linker, LINK-11S. LINK-11S produces a load module which is loaded by the Absolute Loader for execution. Object modules may contain absolute and/or relocatable code and separately assembled object modules may be linked with global symbols. The object module is produced after two passes through the Assembler; an optional third pass produces a complete octal/symbolic listing of the assembled program. This listing is especially useful for documentation and debugging purposes.

This chapter not only explains how to write PAL-11S programs but also how to assemble the source programs into object modules. All facets of the assembly language are explained and illustrated with many examples, and the chapter concludes with assembling procedures. In explaining how to write PAL-11S source programs, it is necessary, especially at the outset, to make frequent forward references. Therefore, we recommend that you first read through the entire chapter to get a "feel" for the language, and then reread the chapter, this time referring to appropriate sections as indicated, for a thorough understanding of the language and assembling procedures.

Some notable features of PAL-11S are:

1. Selective assembly pass functions.
2. Device specification for pass functions.
3. Optional error listing on the teleprinter.
4. Double buffered and concurrent I/O (provided by IOXLPT).
5. Alphabetized, formatted symbol table listing.
6. Relocatable object modules.
7. Global symbols for linking between object modules.


The PAL-11S Assembler requires 8K of memory and provides for about 900 user-defined symbols (see Section 3.2). In addition, it allows a line printer to be used for program listing and/or symbol table listing.

The following discussion of the PAL-11S Assembly Language assumes that you have read the PDP-11 Handbook 1971, with emphasis on those sections which deal with the PDP-11 instruction repertoire, formats, and timings -- a thorough knowledge of these is vital to efficient assembly language programming.

1.0 CHARACTER SET

A PAL-11S source program is composed of symbols, numbers, expressions, symbolic instructions, assembler directives, argument separators, and line terminators written using the following ASCII* characters.

1. The letters A through Z. (Upper and lower case letters are acceptable, although upon input, lower case letters will be converted to upper case letters.)

2. The numbers 0 through 9.

3. The characters . and $. (These characters are reserved for systems use.)

4. The separating or terminating symbols:

```
: = % # @ ( ) , ; " ' + - & !
carriage return tab space line feed form feed
```

2.0 STATEMENTS

A source program is composed of a sequence of statements, where each statement is on a single line. The statement is terminated by a carriage return character which must be immediately followed by either a line feed or form feed character. Should a carriage return character be present and not be followed by a line feed or form feed, the Assembler will generate a Q error (Section 10.0), and that portion of the line following the carriage return will be ignored. Since the carriage return is a required statement terminator, a line feed or form feed not immediately preceded by a carriage return will have one inserted by the Assembler.

It should be noted that, if the Editor (ED-11) is being used to create the source program, a typed carriage return (RETURN key) automatically generates a line feed character.

A statement may be composed of up to four fields which are identified by their order of appearance and by specified terminating characters as explained below and summarized

* ASCII stands for American Standard Code for Information Interchange.
in Appendix B. The four fields are:

<table>
<thead>
<tr>
<th>Label</th>
<th>Operator</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
</table>

The label and comment fields are optional. The operator and operand fields are interdependent -- either may be omitted depending upon the contents of the other.

2.1 Label

A label is a user-defined symbol (see Section 3.2) which is assigned the value of the current location counter. This value may be either absolute or relocatable depending on whether the location counter value is absolute or relocatable. In the latter case, the final absolute value is assigned by the Linker, i.e., the value + the relocation constant. A label is a symbolic means of referring to a specific location within a program. If present, a label always occurs first in a statement and must be terminated by a colon. For example, if the current location is absolute 100_8, the statement:

```
ABCD:   MOV A,B
```

will assign the value 100_8 to the label ABCD so that subsequent reference to ABCD will be to location 100_8. In the above case if the location counter were relocatable then the final value of ABCD would be 100_8+K, where K is the location of the beginning of the relocatable section in which the label ABCD appears. More than one label may appear within a single label field; each label within the field will have the same value. For example, if the current location counter is 100_8, multiple labels in the statement:

```
ABC: $DD: A7.7: MOV A,B
```

will equate each of the three labels ABC, $DD, and A7.7 with the value 100_8 ($ and . are reserved for system software).

The error code M (multiple definition of a symbol) will be generated during assembly if two or more labels have the same first six characters.

2.2 Operator

An operator follows the label field in a statement, and may be an instruction mnemonic or an assembler directive (see Section 8 and Appendix B). When it is an instruction mnemonic, it specifies what action is to be performed on any operand(s) which follows it. When it is an assembler directive, it specifies a certain function or action to be performed during assembly.

The operator may be preceded only by one or more labels and may be followed by one or more operands and/or a comment. An operator is legally terminated by a space, tab, or any of the following characters:

```
# + - @ ( " % ! & , ;
```

line feed form feed carriage return
The use of each character above will be explained in this chapter.

Consider the following examples:

\[
\begin{align*}
\text{MOV} & \rightarrow | A, B \quad ; \rightarrow | (\text{TAB}) \text{ terminates operator MOV} \\
\text{MOV} @ A, B & \quad ; @ \text{ terminates operator MOV}
\end{align*}
\]

When the operator stands alone without an operand or comment, it is terminated by a carriage return followed by a line feed or form feed character.

2.3 Operand

An operand is that part of a statement which is operated on by the operator -- an instruction mnemonic or assemble: directive. Operands may be symbols, expressions, or numbers. When multiple operands appear within a statement, each is separated from the next by a comma. An operand may be preceded by an operator and/or label, and followed by a comment.

The operand field is terminated by a semicolon when followed by a comment, or by a carriage return followed by a line feed or form feed character when the operand ends the statement. For example,

\[
\text{LABEL: MOV GEORGE, BOB ; THIS IS A COMMENT}
\]

where the space between MOV and GEORGE terminated the operator field and began the operand field; the comma separated the operands GEORGE and BOB; the semicolon terminated the operand field and began the comment.

2.4 Comments

The comment field is optional and may contain any ASCII character except null, rubout, carriage return, line feed or form feed. All other characters, even those with special significance are ignored by the Assembler when used in the comment field.

The comment field may be preceded by none, any, or all of the other three fields. It must begin with the semicolon and end with a carriage return followed by a line feed or form feed character. For example,

\[
\text{LABEL: CLR HERE ; THIS IS A $1.00 COMMENT}
\]

Comments do not affect assembly processing or program execution, but they are useful in program listings for later analysis, checkout or documentation purposes.

2.5 Format Control

The format is controlled by the space and tab characters. They have no effect on the assembling process of the source program unless they are embedded within a symbol, number, or ASCII text; or are used as the operator field terminator. Thus, they can be used to provide a neat, readable program. A statement can be written:
LABEL: MOV(SP)+, TAG; POP VALUE OFF STACK

or, using formatting characters it can be written:

LABEL: MOV (SP)+, TAG ; POP VALUE OFF STACK

which is much easier to read.

Page size is controlled by the form feed character. A page of n lines is created by inserting a form feed (CTRL/FORM keys on the keyboard) after the nth line. If no form feed is present, a page is automatically terminated after 56 lines.

3.0 SYMBOLS

There are two types of symbols, permanent and user-defined. Both are stored in the Assembler's symbol table. Initially, the symbol table contains the permanent symbols, but as the source program is assembled, user-defined symbols are added to the table.

3.1 Permanent Symbols

Permanent symbols consist of the instruction mnemonics (see Appendix B.3) and assembler directives (see Section 8.0). These symbols are a permanent part of the Assembler's symbol table and need not be defined before being used in the source program.

3.2 User-Defined Symbols

User-defined symbols are those defined as labels (see Section 2.1) or by direct assignment (see Section 3.3). These symbols are added to the symbol table as they are encountered during the first pass of the assembly. They can be composed of alphanumeric characters, dollar signs, and periods only; again $'s and .'s are reserved for system software. Any other character is illegal and, if used, will result in the error message 1 or QU (see Section 10.0). 1 is a low priority error which may be flagged as QU first. The following rules also apply to user-defined symbols:

1. The first character must not be a number.

2. Each symbol must be unique within the first six characters.

3. A symbol may be written with more than six legal characters but the seventh and subsequent characters are only checked for legality, and are not otherwise recognized by the Assembler.

4. Spaces and tabs must not be embedded within a symbol.

A user-defined symbol may duplicate a permanent symbol. The value associated with a permanent symbol that is also user-defined depends upon its use:

1. A permanent symbol encountered in the operator field is associated with its corresponding machine op-code.
2. If a permanent symbol in the operand field is also user-defined, its user-defined value is associated with the symbol. If the symbol is not found to be user-defined, then the corresponding machine op-code value is associated with the symbol.

User-defined symbols are either internal or global. All symbols are internal unless they are explicitly typed as global with the .GLOBL assembler directive (see Section 8.2). Global symbols are used to provide links between object modules. A global symbol which is defined (as a label or by direct assignment) in a program is called an entry symbol or entry point. Such symbols may be referred to from other object modules or assemblies. A global symbol which is not defined in the current assembly is called an external symbol. Some other assembly must define the same symbol as an entry point.

3.3 Direct Assignment

A direct assignment statement associates a symbol with a value. When a direct assignment statement defines a symbol for the first time, that symbol is entered into the Assembler’s symbol table and the specified value is associated with it. A symbol may be redefined by assigning a new value to a previously defined symbol. The newly assigned value will replace the previous value assigned to the symbol.

The symbol takes on the relocatable or absolute attribute of the defining expression. However, if the defining expression is global, the defined symbol will not be global unless previously defined as such (see Section 4.0).

The general format for a direct assignment statement is:

symbol = expression.

The following conventions apply:

1. An equal sign (=) must separate the symbol from the expression defining the symbol.

2. A direct assignment statement may be preceded by a label and may be followed by a comment.

3. Only one symbol can be defined by any one direct assignment statement.

4. Only one level of forward referencing is allowed.

Example of two levels of forward referencing (illegal):

\[
X = Y \\
Y = Z \\
Z = 1
\]

X and Y are both undefined throughout pass 1 and will be listed on the teleprinter as such at the end of that pass. X is undefined throughout pass 2, and will cause a U error message.
Examples:

\[
\begin{align*}
A &= 1 ; \text{THE SYMBOL A IS EQUATED WITH THE VALUE 1} \\
B &= 'A-1\&\text{MASKLOW} ; \text{THE SYMBOL B IS EQUATED WITH THE EXPRESSION'S VALUE} \\
C : D &= 3 ; \text{THE SYMBOL D IS EQUATED WITH 3} \\
E : \text{MOV } \#1, \text{ABLE} ; \text{LABELS C AND E ARE EQUATED WITH THE NUMERICAL MEMORY ADDRESS OF THE MOV COMMAND}
\end{align*}
\]

3.4 Register Symbols

The eight general registers of the PDP-11 are numbered 0 through 7. These registers may be referenced by use of a register symbol; that is, a symbolic name for a register. A register symbol is defined by means of a direct assignment, where the defining expression contains at least one term preceded by a % or at least one term previously defined as a register symbol. In addition, the defining expression of a register symbol must be absolute. For example:

\[
\begin{align*}
R0 &= \%0 ; \text{DEFINE R0 AS REGISTER 0} \\
R3 &= R0 + 3 ; \text{DEFINE R3 AS REGISTER 3} \\
R4 &= 1 + \%3 ; \text{DEFINE R4 AS REGISTER 4} \\
\text{THERE} &= \%2 ; \text{DEFINE "THERE" AS REGISTER 2}
\end{align*}
\]

It is important to note that all register symbols must be defined before they are referenced. A forward reference to a register symbol will generally cause phase errors (see Section 10.0).

The % may be used in any expression thereby indicating a reference to a register. Such an expression is a register expression. Thus, the statement:

\[
\text{CLR} \quad \%6
\]

will clear register 6 while the statement:

\[
\text{CLR} \quad 6
\]

will clear the word at memory address 6. In certain cases a register can be referenced without the use of a register symbol or register expression. These cases are recognized through the context of the statement and are thoroughly explained in Sections 7.11 and 7.12. Two obvious examples of this are:

\[
\begin{align*}
\text{JSR} \quad 5, \text{SUBR} ; \text{THE FIRST OPERAND FIELD MUST ALWAYS BE A REGISTER}
\end{align*}
\]
CLR X(2) ; ANY EXPRESSION ENCLOSED IN ( ) MUST BE
; A REGISTER. IN THIS CASE, INDEX REGISTER
; 2

4.0 EXPRESSIONS

Arithmetic and logical operators (see Section 4.2) may be used to form expressions. A term of an expression may be a permanent or user-defined symbol (which may be absolute, relocatable or global), a number, ASCII data, or the present value of the assembly location counter represented by the period (see Section 5.0). Expressions are evaluated from left to right. Parenthetical grouping is not allowed.

Expressions are evaluated as word quantities. The operands of a .BYTE directive (Section 8.8) are evaluated as word expressions before truncation to the low-order eight bits. The evaluation of an expression includes the evaluation of the mode of the resultant expression; that is, absolute, relocatable or external. The definition of the modes of expression are given below in Section 4.4.

A missing term, expression or external symbol will be interpreted as 0. A missing operator will be interpreted as +. The error code Q (Questionable syntax) will be generated for a missing operator. For example,

\[ A + 100 \]; OPERAND MISSING

will be evaluated as \[ A + 0 - 100 \], and

\[ TAG ! LA 177777 \]; OPERATOR MISSING

will be evaluated as \[ TAG ! LA + 177777 \].

The value of an external expression will be the value of the absolute part of the expression; e.g., EXT+A will have a value of A. This will be modified by the linker to become EXT+A.

4.1 Numbers

The Assembler accepts both octal and decimal numbers. Octal numbers consist of the digits 0 through 7 only. Decimal numbers consist of the digits 0 through 9 followed by a decimal point. If a number contains an 8 or 9 and is not followed by a decimal point, the N error code (see Section 10.0) will be printed and the number will be interpreted as decimal. Negative numbers may be expressed as a number preceded by a minus sign rather than in a two's complement form. Positive numbers may be preceded by a plus sign although this is not required.

If a number is too large to fit into 16 bits, the number is truncated from the left. In the assembly listing the statement will be flagged with a Truncation (T) error. Numbers are always considered to be absolute quantities (that is, not relocatable).
4.2 Arithmetic and Logical Operators

The arithmetic operators are:

+ indicates addition or a positive number

- indicates subtraction or a negative number

The logical operators are:

& indicates the logical AND operation

! indicates the logical inclusive OR operation

4.3 ASCII Conversion

When preceded by an apostrophe, any ASCII character (except null, rubout, carriage return, line feed, or form feed) is assigned the 7-bit ASCII value of the character (see Appendix A). For example,

'A'

is assigned the value 1018.

When preceded by a quotation mark, two ASCII characters (not including null, rubout, carriage return, line feed, or form feed) are assigned the 7-bit ASCII values of each of the characters to be used. Each 7-bit value is stored in an 8-bit byte and the bytes are combined to form a word. For example "AB" will store the ASCII value of A in the low-order (even) byte and the value of B in the high-order (odd) byte:

<table>
<thead>
<tr>
<th>high-order byte</th>
<th>low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>B's value = 100 001 001</td>
<td>000 001</td>
</tr>
</tbody>
</table>

"AB=041101"

ASCII text is always absolute.
4.4 Mode of Expressions

The mode of an expression may be absolute, relocatable or external as defined below:

A term of an expression is absolute, relocatable or external depending on whether its definer (i.e., number, symbol, etc.) is absolute, relocatable or external. Numbers, permanent symbols and generated data are always treated as absolute.

An absolute expression is defined as:

1. Absolute term preceded optionally by a single plus or minus sign, or
2. Relocatable expression minus a relocatable term, or
3. Absolute expression followed by an operator followed by an absolute expression.

A relocatable expression is defined as:

1. Relocatable term, or
2. Relocatable expression followed by an arithmetic operator followed by an absolute expression, or
3. Absolute expression followed by a plus operator followed by a relocatable expression.

An external expression is defined as:

1. External term, or
2. External expression followed by an arithmetic operator followed by an absolute term, or
3. Absolute expression followed by a plus operator followed by an external expression.

In the following examples:

ABS is an absolute symbol,
REL is a relocatable symbol,
EXT is an external symbol.

Examples:

The following are valid expressions:
EXT + ABS ;External expression

REL+REL-REL- ;Relocatable expression

ABS+REL-REL & ABS ; Absolute expression

The following are illegal expressions:

EXT+REL
REL+REL
ABS-EXT

5.0 ASSEMBLY LOCATION COUNTER

The period (.) is the symbol for the assembly location counter. (Note difference of Program Counter. . ≠ PC. See Section 7.0.) When used in the operand field of an instruction, it represents the address of the first word of the instruction. When used in the operand field of an assembler directive, it represents the address of the current byte or word. For example,

A: MOV #.,R0 ; . refers to location A,
        ; i.e., the address of the
        ; MOV instruction

(# is explained in Section 7.9.)

At the beginning of each assembly pass, the Assembler clears the location counter. Normally, consecutive memory locations are assigned to each byte of object data generated. However, the location where the object data is stored may be changed by a direct assignment altering the location counter:

.=expression

Similar to other symbols, the location counter symbol "." has a mode associated with it. However, the mode cannot be external. Neither can one change the existing mode of the location counter by using a defining expression of a different mode.

The mode of the location counter symbol can be changed by the use of the .ASECT or .CSECT directive as explained in section 8.3.

The expression defining the location counter must not contain forward references or symbols that vary from one pass to another.

Examples:

 .ASECT
  .=500 ;SET LOCATION COUNTER TO ABSOLUTE 500
FIRST: MOV .+10, COUNT

; THE LABEL FIRST HAS THE VALUE 5008.
; .+10 EQUALS 5108. THE CONTENTS OF
; THE LOCATION 5108 WILL BE DEPOSITED
; IN LOCATION COUNT.

. = 520

SECOND: MOV . , INDEX

; THE LABEL SECOND HAS THE VALUE 5208.
; THE CONTENTS OF LOCATION 5208, THAT
; IS, THE BINARY CODE FOR THE INSTRUCTION ITSELF, WILL BE DEPOSITED IN
; LOCATION INDEX.

.CSECT

; SET LOCATION COUNTER TO RELOCATABLE
; 20.

THIRD: .WORD 0

; THE LABEL THIRD HAS THE VALUE OF
; RELOCATABLE 20.

Storage area may be reserved by advancing the location counter. For example, if the
current value of the location counter is 1000, the direct assignment statement

. = . + 100

will reserve 1008 bytes of storage space in the program. The next instruction will be
stored at 1100.

6.0 RELOCATION AND LINKING

The output of the relocatable assembler is an object module which must be processed by
the PDP-11 Linker, LINK-11S, before loading and execution. The Linker essentially
fixes (i.e., makes absolute) the values of external or relocatable symbols and creates
another module (load module) which contains the binary data to be loaded and executed.

To enable the Linker to fix the value of an expression the assembler issues certain direc-
tives to the Linker together with the required parameters. In the case of relocatable
expressions the Linker adds the base of the relocatable section (the location in memory
of relocatable 0) to the value of the relocatable expression provided by the Assembler.
In the case of an external expression the value of the external term in the expression is
determined by the Linker (since the external symbol must be defined in one of the other
object modules being linked and adds it to the value of the external expression provided
by the Assembler.

All instructions that are to be modified as described above will be marked by a single
apostrophe in the assembly listing. Thus the binary text output will look as follows
for the given examples:
7.0 ADDRESSING

The Program Counter (register 7 of the eight general registers) always contains the address of the next word to be fetched; i.e., the address of the next instruction to be executed, or the second or third word of the current instruction.

In order to understand how the address modes operate and how they assemble, the action of the Program Counter must be understood. The key rule is:

Whenever the processor implicitly uses the Program Counter to fetch a word from memory, the Program Counter is automatically incremented by two after the fetch.

That is, when an instruction is fetched, the PC is incremented by two, so that it is pointing to the next word in memory; and, if an instruction uses indexing (see Sections 7.7, 7.8 and 7.10), the processor uses the Program Counter to fetch the base from memory. Hence, using the rule above, the PC increments by two, and now points to the next word.

The following conventions are used in this section:

1. Let E be any expression as defined in Section 4.0.

2. Let R be a register expression. This is any expression containing a term preceded by a % character of a symbol previously equated to such a term.

Examples:

\[
\begin{align*}
R0 &= \%0 \quad ; \text{GENERAL REGISTER 0} \\
R1 &= R0+1 \quad ; \text{GENERAL REGISTER 1} \\
R2 &= 1+\%1 \quad ; \text{GENERAL REGISTER 2}
\end{align*}
\]

3. Let ER be a register expression or an expression in the range 0 to 7 inclusive.

4. Let A be a general address specification which produces a 6-bit mode address field as described in the PDP-11 Handbook 1971.

The addressing specifications, A, may now be explained in terms of E, R, and ER as defined above. Each will be illustrated with the single operand instruction CLR or double operand instruction MOV.

1-13
7.1 Register Mode
The register contains the operand.

Format: \( R \)

Example:

\[
\text{RO}=\%0 \quad ;\text{DEFINE R0 AS REGISTER 0}
\]
\[
\text{CLR} \quad \text{R0} \quad ;\text{CLEAR REGISTER 0}
\]

7.2 Deferred Register Mode
The register contains the address of the operand.

Format: \( \sim R \) or \( (ER) \)

Example:

\[
\text{CLR} \quad \sim \text{R1} \quad ;\text{CLEAR THE WORD AT THE}
\]
\[
\text{or} \quad ;\text{ADDRESS CONTAINED IN}
\]
\[
\text{CLR} \quad (1) \quad ;\text{REGISTER 1}
\]

7.3 Autoincrement Mode
The contents of the register are incremented immediately after being used as the address of the operand.

Format: \( (ER)+ \)

Examples:

\[
\text{CLR} \quad (\text{R0})+ \quad ;\text{CLEAR WORDS AT ADDRESSES}
\]
\[
\text{CLR} \quad (\text{R0+3})+ \quad ;\text{CONTAINED IN REGISTERS 0, 3, AND 2}
\]
\[
\text{CLR} \quad (2)+ \quad ;\text{AND INCREMENT REGISTER CONTENTS BY TWO.}
\]

NOTE

Both JMP and JSR instructions using mode 2 (non-deferred autoincrement mode), autoincrement the register before its use.

In double operand instructions of the addressing form \( %R \), \( (R)+ \) or \( %R \), \(- (R)\) where the source and destination registers are the same, the source operand is evaluated as the autoincremented or autodecremented value; but the destination register, at the time it is used, still contains the originally intended effective address. For example, if Register 0 contains 100, the following occurs:

\[
\text{MOV R0, (0)+} \quad ;\text{THE QUANTITY 102 IS MOVED TO LOCATION 100}
\]
\[
\text{MOV R0, -(0)} \quad ;\text{THE QUANTITY 76 IS MOVED TO LOCATION 76}
\]

The use of these forms should be avoided, as they are not guaranteed to remain in future PDP-11's.

7.4 Deferred Autoincrement Mode
The register contains the pointer to the address of the operand. The contents of the register are incremented after being used.
7.5 Autodecrement Mode

The contents of the register are decremented before being used as the address of the operand (see note in Section 7.3).

Format: -(ER)

Examples:

CLR -(RO) ;DECREMENT CONTENTS OF REGISTERS
CLR -(RO+3) ;0, 3 and 2 BEFORE USING
CLR -(2) ;AS ADDRESSES OF WORDS TO BE CLEARED

7.6 Deferred Autodecrement Mode

The contents of the register are decremented before being used as the pointer to the address of the operand.

Format: @-(ER)

Example:

CLR @-(2) ;DECREMENT CONTENTS OF REG. 2
;BEFORE USING AS POINTER TO ADDRESS
;OF WORD TO BE CLEARED.

7.7 Index Mode

Format: E(ER)

The value of an expression E is stored as the second or third word of the instruction. The effective address is calculated as the value of E plus the contents of register ER. The value E is called the base.

Examples:

CLR X+2(R1) ;EFFECTIVE ADDRESS IS X+2 PLUS
;THE CONTENTS OF REGISTER 1
CLR -2(3) ;EFFECTIVE ADDRESS IS -2 PLUS
;THE CONTENTS OF REGISTER 3
7.8 Deferred Index Mode

An expression plus the contents of a register gives the pointer to the address of the operand.

Format:  \( \# E(ER) \)

Example:

\[
\text{CLR } \# 14(4) ; \text{IF REGISTER 4 HOLDS 100, AND LOCATION 114 HOLDS 2000, LOC.2000 IS CLEARED.}
\]

7.9 Immediate Mode and Deferred Immediate (Absolute) Mode

The immediate mode allows the operand itself to be stored as the second or third word of the instruction. It is assembled as an autoincrement of register 7, the PC.

Format:  \( \# E \)

Examples:

\[
\text{MOV } \#100, R0 \quad \text{;MOVE AN OCTAL 100 TO REGISTER 0}
\]
\[
\text{MOV } \#X, R0 \quad \text{;MOVE THE VALUE OF SYMBOL X TO REGISTER 0.}
\]

The operation of this mode is explained as follows:

The statement \( \text{MOV } \#100, R3 \) assembles as two words. These are:

\[
012703 \\
000100
\]

Just before this instruction is fetched and executed, the PC points to the first word of the instruction. The processor fetches the first word and increments the PC by two. The source operand mode is 27 (autoincrement the PC). Thus the PC is used as a pointer to fetch the operand (the second word of the instruction) before being incremented by two, to point to the next instruction.

If the \( \# E \) is preceded by \( \# \), \( E \) specifies an absolute address.

7.10 Relative and Deferred Relative Modes

Relative mode is the normal mode for memory references.

Format:  \( E \)

Examples:

\[
\text{CLR } 100 \quad ; \text{CLEAR LOCATION 100}
\]
\[
\text{MOV } X, Y \quad ; \text{MOVE CONTENTS OF LOCATION X TO LOCATION Y.}
\]
This mode is assembled as Index mode, using 7, the PC, as the register. The base of the address calculation, which is stored in the second or third word of the instruction, is not the address of the operand. Rather, it is the number which, when added to the PC, becomes the address of the operand. Thus, the base is \( \text{X-PC} \). The operation is explained as follows:

If the statement \( \text{MOV 100, R3} \) is assembled at absolute location 20 then the assembled code is:

\[
\begin{align*}
\text{Location 20:} & \quad 016703 \\
\text{Location 22} & \quad 000054
\end{align*}
\]

The processor fetches the MOV instruction and adds two to the PC so that it points to location 22. The source operand mode is 67; that is, indexed by the PC. To pick up the base, the processor fetches the word pointed to by the PC and adds two to the PC. The PC now points to location 24. To calculate the address of the source operand, the base is added to the designated register. That is, \( \text{BASE+PC=54+24=100} \), the operand address.

Since the Assembler considers "." as the address of the first word of the instruction, an equivalent statement would be

\[
\text{MOV 100 -. - 4(PC), R3}
\]

This mode is called relative because the operand address is calculated relative to the current PC. The base is the distance (in bytes) between the operand and the current PC. If the operator and its operand are moved in memory so that the distance between the operator and data remains constant, the instruction will operate correctly.

If E is preceded by \(@\) the expression's value is the pointer to the address of the operand.

7.11 Table of Mode Forms and Codes (6-bit(A) format only – see Section 7.12)

Each instruction takes at least one word. Operands of the first six forms listed below, do not increase the length of an instruction. Each operand in one of the other modes, however, increases the instruction length by one word.
Any of these forms adds a word to the instruction length.

Notes:
1. An alternate form for @R is (ER). However, the form @ (ER) is equivalent to @0(ER).
2. The form @ #E differs from the form E in that the second or third word of the instruction contains the absolute address of the operand rather than the relative distance between the operand and the PC. Thus, the statement CLR @ #100 will clear location 100 even if the instruction is moved from the point at which it was assembled.

The Assembler is not particular about left and right and dangling + and - signs in address fields. The following are some examples of incorrect syntax that assemble as indicated, without an error indication.

<table>
<thead>
<tr>
<th>Form</th>
<th>Assembles As:</th>
<th>Form</th>
<th>Assembles As:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R2)A</td>
<td>A(R2)</td>
<td>(R2)-</td>
<td>-(R2)</td>
</tr>
<tr>
<td>A-(R2)</td>
<td>A(R2) or A-Ø(R2)</td>
<td>@ (R2)A</td>
<td>@ A(R2)</td>
</tr>
<tr>
<td>A(Rw)+</td>
<td>A(R2)</td>
<td>A(R2)+B</td>
<td>A+B(R2)</td>
</tr>
<tr>
<td>+R2)</td>
<td>(R2)+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7.12 Instruction Forms

The instruction mnemonics are given in Appendix B. This section defines the number and nature of the operand fields for these instructions.

In the table that follows, let R, E, and ER represent expressions as defined in Sections 4.0 and 7.0 and let A be a 6-bit address specification of the forms:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double Operand</td>
<td>Op A, A</td>
<td>MOV (R6)+, @ Y</td>
</tr>
<tr>
<td>Single Operand</td>
<td>Op A</td>
<td>CLR -(R2)</td>
</tr>
<tr>
<td>Operate</td>
<td>OP</td>
<td>HALT</td>
</tr>
<tr>
<td>Branch</td>
<td>Op E</td>
<td>BR X+2</td>
</tr>
<tr>
<td></td>
<td>where (-128 &lt; E - 2 &lt; 127)</td>
<td>BLO .-4</td>
</tr>
<tr>
<td>Subroutine Call</td>
<td>JSR ER, A</td>
<td>JSR PC, SUBR</td>
</tr>
<tr>
<td>Subroutine Return</td>
<td>RTS ER</td>
<td>RTS PC</td>
</tr>
<tr>
<td>EMT/TRAP</td>
<td>Op or OP E</td>
<td>EMT</td>
</tr>
<tr>
<td></td>
<td>where (0 &lt; E &lt; 377)</td>
<td>EMT 31</td>
</tr>
</tbody>
</table>
The branch instructions are one word instructions. The high byte contains the op code and
the low byte contains an 8-bit signed offset (7 bits plus sign) which specifies the branch
address relative to the PC. The hardware calculates the branch address as follows:

1. Extend the sign of the offset through bits 8-15.
2. Multiply the result by 2. This creates a word offset rather than a byte offset.
3. Add the result to the PC to form the final branch address.

The Assembler performs the reverse operation to form the byte offset from the specified
address. Remember that when the offset is added to the PC, the PC is pointing to the word
following the branch instruction; hence the factor -2 in the calculation.

\[ \text{Byte offset} = \frac{(E-\text{PC})}{2} \text{ truncated to eight bits.} \]

Since \( \text{PC} = +2 \), we have

\[ \text{Byte offset} = \frac{(E-2)}{2} \text{ truncated to eight bits.} \]

**NOTE**

It is illegal to branch to a location specified as an external symbol,
or to a relocatable symbol when within an absolute section, or to an absolute symbol when within a relocatable section.

The EMT and TRAP instructions do not use the low-order byte of the word. This allows in­
formation to be transferred to the trap handlers in the low-order byte. If EMT or TRAP is
followed by an expression, the value is put into the low-order byte of the word. However, if
the expression is too bit (>377) it is truncated to eight bits and a Truncation (T) error occurs.

Do not try to micro-program the condition code operators (see Appendix B, B.4). This makes
sense in the PDP-11 hardware; however, the current PAL-11S Assembler does not support this
capability. Thus:

\[ \text{CLC} \sim \text{CLV} \]
results in a \( \text{Q} \) error (see Appendix B, B.5) and the statement is assembled as \text{CLC}.

Expressions in the Assembler do, however, allow logical operators and the use of instruction
mnemonics. Thus, the proper ways to write the above statement:

```
.WORD CLC~
+CLC~CLV
!CLC~CLV
```

8.0 ASSEMBLER DIRECTIVES

Assembler directives (sometimes called pseudo-ops) direct the assembly process and may
generate data.

Assembler directives may be preceded by a label and followed by a comment. The assembler
directive occupies the operator field. Only one directive may be placed in any one state­
ment. One or more operands may occupy the operand field or it may be void -- allowable
operands vary from directive to directive.

8.1 .TITLE

The .TITLE directive is used to name the object module. The name is assigned by the first
symbol following the directive. If there is no .TITLE statement the default name assigned
is ".MAIN.".
8.2 .GLOBL

The .GLOBL directive is used to declare a symbol as being global. It may be an entry symbol, in which case it is defined in the program, or it may be an external symbol, in which case it should be defined in another program which will be linked with this program by the linker.

The form of the .GLOBL directive is

```
.GLOBL NAME, NAME2, ..., NAME N
```

**NOTE**

A symbol cannot be declared global by defining it as a global expression in a direct assignment statement.

If an illegal character is detected in the operand field of a .GLOBL statement, an error message is not generated; and the Assembler may ignore the remainder of the statement.

Thus:

```
GLOBL A,B, @ C,D
```

assembles without error as:

```
.GLOBL A, B
```

8.3 Program Section Directives (.ASECT and .CSECT)

The relocatable assembler provides for two program sections, an absolute section declared by an .ASECT directive and a relocatable section declared by a .CSECT directive. These directives therefore enable the programmer to specify that parts of his program be assembled in the absolute section and others in a relocatable section. The scope of each directive extends until a directive to the contrary is given. The Assembler initially starts in the relocatable section. Thus, if the first statement of a program were

```
A: .ASECT
```

the label "A" would be a relocatable symbol which is assigned the value of relocatable zero. The absolute value of A will be calculated by the Linker by adding the value of the base of the relocatable section.

Example:

```
.ASECT ;ASSEMBLER IN ABSOLUTE SECTION
=1000 ;PC = 1000 ABSOLUTE
A: CLR X ;A = 1000 ABSOLUTE
.CSECT ;ASSEMBLE IN RELOCATABLE SECTION
X: JMP A ;X=0 RELOCATABLE
.END
```

The absolute and/or relocatable section may be discontinued (by switching to the alternate section) and then continued where they left off by using another .ASECT or .CSECT statement.

Example:
.CSECT
.WORD 0,1,2 ;ASSEMBLED AT RELOCATABLE 0, 2 and 4
.ASECT
.WORD 0,1,2 ;ASSEMBLED AT ABSOLUTE 0, 2 and 4
.CSECT
.WORD 0 ;ASSEMBLED AT RELOCATABLE 6.
.END

If a label is defined twice, first in an absolute section and then in a relocatable section, the symbol will be relocatable but its value will be as defined in the absolute section.

8.4 .EOT

The .EOT directive indicates the physical End Of Tape though not the logical end of the program. If the .EOT is followed by a single line feed or form feed, the Assembler will still read to the end of the tape, but will not process anything past the .EOT directive. If .EOT is followed by at least two line feeds or form feeds, the Assembler will stop before the end of the tape. Either case is proper, but it should be understood that even though it appears as if the Assembler has read too far, it actually hasn't.

If a .EOT is embedded in a tape, and more information to be assembled follows it, .EOT must be immediately followed by at least two line feeds or form feeds. Otherwise, the first line following the .EOT will be lost.

Any operands following a .EOT directive will be ignored. The .EOT directive allows several physically separate tapes to be assembled as one program. The last tape should be terminated by a .END directive (see Section 8.6) but may be terminated with .EOT (see .END emulation in Section 9.4).

8.5 .EVEN

The .EVEN directive ensures that the assembly location counter is even by adding one if it is odd. Any operands following a .EVEN directive will be ignored.

8.6 .END

The .END directive indicates the logical and physical end of the source program. The .END directive may be followed by only one operand, an expression indicating the program’s transfer address.

At load time, the load module will be loaded and program execution will begin at the transfer address indicated by the .END directive. If the address is not specified, the loader will halt after reading in the load module.

8.7 .WORD

The .WORD assembler directive may have one or more operands, separated by commas. Each operand is stored in a word of the object program. If there is more than one operand, they are stored in successive words. The operands may be any legally formed expression. For example,
Values exceeding 16 bits will be truncated from the left, to word length.

A .WORD directive followed by one or more void operands separated by commas will store zeros for the void operands. For example,

`.=1430
.WORD ,5, ;ZERO, FIVE, AND ZERO ARE STORED IN WORDS 1430, 1432, AND 1434.

An operator field left blank will be interpreted as the .WORD directive if the operand field contains one or more expressions. The first term of the first expression in the operand field must not be an instruction mnemonic or assembler directive unless preceded by a +, -, or one of the logical operators, !, and &. For example,

`.=440
.LABEL: +MOV, LABEL ;IS STORED IN LOCATION 440. 440 IS STORED IN LOCATION 442.

Note that the default .WORD will occur whenever there is a leading arithmetic or logical operator, or whenever a leading symbol is encountered which is not recognized as an instruction mnemonic or assembler directive. Therefore, if an instruction mnemonic or assembler directive is misspelled, the .WORD directive is assumed and errors will result. Assume that MOV is spelled incorrectly as MOR:

MOR A,B

Two error codes can result: A Q will occur because an expression operator is missing between MOR and A, and a U will occur if MOR is undefined. Two words will be generated; one for MOR A and one for B.

8.8 .BYTE

The .BYTE assembler directive may have one or more operands separated by commas. Each operand is stored in a byte of the object program. If multiple operands are specified, they are stored in successive bytes. The operands may be any legally formed expression with a result of 8 bits or less. For example,

SAM=5
`.=410
.BYTE 48.,SAM ;STORED IN LOCATION 410 WILL BE 060 (THE OCTAL EQUIVALENT OF 48).

If the expression has a result of more than 8 bits, it will be truncated to its low-order 8 bits and will be flagged as a T error. If an operand after the .BYTE directive is left void, it will be interpreted as zero. For example,
If the expression is relocatable, a warning flag, A, will be given.

8.9 .ASCII

The .ASCII directive translates strings of ASCII characters into their 7-bit ASCII codes with the exception of null, rubout, carriage return, line feed and form feed. The text to be translated is delimited by a character at the beginning and the end of the text. The delimiting character may be any printing ASCII character except colon and equal sign and those used in the text string. The 7-bit ASCII code generated for each character will be stored in successive bytes of the object program. For example,

```
.ASCII /YES/
.ASCII /5+3/2/
```

The .ASCII directive may be terminated by any legal terminator except for = and :. Note that if the text delimiter is also a terminator, the leading text delimiter can also serve as the .ASCII directive terminator. For example,

```
.ASCII /ABCD/  ;THE SPACE IS REQUIRED
.ASCII+ABCD+  ;NO SPACE IS REQUIRED.
```

8.10 .RAD50

PDP-11 system programs often handle symbols in a specially coded form called "RADIX 50" (this form is sometimes referred to as "MOD40"). This form allows 3 characters to be packed into 16 bits; therefore, any symbol can be held in two words, the form of the directive is:

```
.RAD50 /CCC/
```

The single operand is of the form /CCC/ where the slash (the delimiter) can be any printable character except for = and :. The delimiters enclose the characters to be converted which may be A through Z, 0 through 9, dollar ($), dot (.) and space ( ). If there are fewer than 3 characters they are considered to be left-justified and trailing spaces are assumed. Any characters following the trailing delimiter are ignored and no error results.
Examples:

```
.RAD50 /ABC/ ;PACK ABC INTO ONE WORD
.RAD50 /AB/ ;PACK AB (SPACE) INTO ONE WORD;
.RAD50 // ;PACK 3 SPACES INTO ONE WORD
```

The packing algorithm is as follows:

A. Each character is translated into its RADIX 50 equivalent as indicated in the following table:

<table>
<thead>
<tr>
<th>Character</th>
<th>RADIX 50 Equivalent (octal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SPACE)</td>
<td>0</td>
</tr>
<tr>
<td>A-Z</td>
<td>1-32</td>
</tr>
<tr>
<td>$</td>
<td>33</td>
</tr>
<tr>
<td>.</td>
<td>34</td>
</tr>
<tr>
<td>0-9</td>
<td>36-47</td>
</tr>
</tbody>
</table>

Note that another character can be defined for code 35.

B. The RADIX 50 equivalents for characters 1 through 3 (C1, C2, C3) are combined as follows:

\[
\text{RESULT} = ( (C1 \times 50) + C2) \times 50 + C3
\]

8.11 .LIMIT

A program often wishes to know the boundaries of the relocatable code. The .LIMIT directive generates two words into which the linker puts the low and high addresses of the relocated code. The low address (inserted into the first word) is the address of the first byte of code. The high address is the address of the first free byte following the relocated code. These addresses will always be even since all relocatable sections are loaded at even addresses and if a relocatable section consists of an odd number of bytes the linker adds one to the size to make it even.

8.12 CONDITIONAL ASSEMBLY DIRECTIVES

Conditional assembly directives provide the programmer with the capability to conditionally include or not include portions of his source code in the assembly process. In what follows, E denotes an expression and S(i) denotes a symbol. The conditional directives are:

```
.IFZ E ;IF E=0
.IFNZ E ;IF E≠0
.IFL E ;IF E<0
.IFLE E ;IF E≤0
.IFG E ;IF E>0
.IFGE E ;IF E≥0
.IFDF S (1) [!, &] S (2) [!, &] ... [!, &] S(N) (!=OR, &=AND)
.IFNDF S (1) [!, &] S (2) [!, &] ... [!, &] S(N)
```

1-24
If the condition is met, all statements up to the matching .ENDC are assembled. Otherwise, the statements are ignored until the matching .ENDC is detected.

In the above, .IFDF and .IFNDF mean "if defined" and "if undefined" respectively. The scan is left to right, no parentheses permitted.

Example:

```
.IFDF S!T&U
 Means assemble if either S or T is defined
and U is defined

.IFNDF T&U!S
 Means assemble if both T and U are undefined
or if S is undefined
```

General Remarks:

An errored or null expression takes the default value 0 for purposes of the conditional test. An error in syntax, e.g., a terminator other than ;, !, &, or CR results in the undefined situation for .IFDF and .IFNDF, as does an errored or null symbol.

All conditionals must end with the .ENDC directive. Anything in the operand field of .ENDC is ignored. Nesting is permitted up to a depth of 127. Labels are permitted on conditional directives, but the scan is purely left to right. For example:

```
.A:
.IFZ 1
 .ENDC
```

A is ignored.

```
.A:
 .IFZ 1
 .ENDC
```

A is entered in the symbol table.

If a .END is encountered while inside a satisfied conditional, a Q flag will appear, but the .END directive will still be processed normally. If more .ENDC's appear than are required, Q flags appear on the extras.

9.0 OPERATING PROCEDURES

9.1 Introduction

The Assembler enables you to assemble an ASCII tape containing PAL-11 statements into a relocatable binary tape (object module). To do this, two or three passes are necessary. On the first pass, the Assembler creates a table of user-defined symbols and their associated values, and a list of undefined symbols is printed on the teleprinter. On the second pass the Assembler assembles the program and punches out an absolute binary tape and/or outputs an assembly listing. During the third pass (this pass is optional), the Assembler punches an absolute binary tape or outputs an assembly listing. The symbol
table (and/or a list of errors) may be output on any of these passes. The input and output
devices as well as various options are specified during the initial dialogue (see Section
9.3). The Assembler initiates the dialogue immediately after being loaded and after the
last pass of an assembly.

9.2 Loading PAL-11S

PAL-11S is loaded by the Paper Tape Software Absolute Loader. Note that the start
address of the Absolute Loader must be in the Switch Register when loading the Assembler.
This is because the Assembler tape has an initial program which clears all of core up to
the address specified in the Switch Register, and jumps to that address to start loading the
Assembler.

9.3 Initial Dialogue

After being loaded, the Assembler prints its name and version and then initiates dialogue
by printing on the teleprinter:

*S

meaning "What is the Source symbolic input device?" The response may be

) use Low-speed reader ( ) denotes typing the RETURN key)
H meaning High-speed reader
L meaning Low-speed reader
T meaning Teleprinter keyboard

The device specification is terminated, as is all user response, by typing the RETURN key.

If an error is made in typing at any time, typing the RUBOUT key will erase the im-
mmediately preceding character if it is on the current line. Typing CTRL/U will erase the
whole line on which it occurs.

After the *S question and response, the Assembler prints:

*B

meaning "What is the Binary output device?" The responses to *B are similar to those
for *S:

H meaning High-speed punch
L meaning Low-speed punch
) meaning do not output binary tape ( ) denotes typing the RETURN key)

In addition to I/O device specification, various options may be chosen. The binary output
will occur on the second pass unless /3 (indicating the third pass) is typed following the H
or L. Errors will be listed on the same pass if /E is typed. If /E is typed in response to more
than one inquiry, only the last occurrence will be honored. It is strongly suggested that
the errors be listed on the same pass as the binary output, since errors may vary from pass to pass.

If both /3 and /E are typed, /3 must precede /E. The response is terminated by typing the RETURN key. Examples:

*B L/E Binary output on the low-speed punch and the errors on the teleprinter, both during the second pass.
*B H/3/E Binary output on the high-speed punch and the errors on the teleprinter during the third pass.
*B ) The RETURN key alone will cause the Assembler to omit binary output.

After the *B question and response, the Assembler prints:

*L

meaning "What is the assembly Listing output device?" The response to *L may be:

L meaning Low-speed punch
H meaning High-speed punch
T meaning Teleprinter
P meaning Line Printer
) meaning do not output listing ( ) denotes typing RETURN.

After the I/O device specification, pass and error list options similar to those for *B may be chosen. The assembly listing will be output on the third pass unless /2 (indicating the second pass) is typed following H, L, T, or P. Errors will be listed on the teleprinter during the same pass if /E is typed. If both /2 and /E are typed, /2 must precede /E. The response is terminated by typing the RETURN key. Examples:

*L L/2/E Listing on low-speed punch and errors on teleprinter during second pass.
*L H Listing on high-speed punch during third pass
*L ) The RETURN key alone will cause the Assembler to omit listing output.

After the *L question and response, the final question is printed on the teleprinter:

*T

meaning "What is the symbol Table output device?" The device specification is the same
as for *L question. The symbol table will be output at the end of the first pass unless /2 or /3 is typed in response to *T. The first tape to be assembled should be placed in the reader before typing the RETURN key because assembly will begin upon typing RETURN to the *T question. The /E option is not a meaningful response to *T. Example:

* T  T/3  Symbol table output on teleprinter at end of third pass.

* T  Typing the RETURN key alone will cause the Assembler to omit symbol table output.

The symbol table is printed alphabetically, three symbols per line. Each symbol printed is followed by its identifying characters and by its value. If the symbol is undefined, six asterisks replace its value. The identifying characters indicate the class of the symbol; that is, whether it is a label, direct-assignment, register symbol, etc. The following examples show the various forms:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDEF</td>
<td>001244</td>
<td>(Defined label)</td>
</tr>
<tr>
<td>R3</td>
<td>%000003</td>
<td>(Register symbol)</td>
</tr>
<tr>
<td>DIRASM</td>
<td>177777</td>
<td>(Direct assignment)</td>
</tr>
<tr>
<td>XYZ</td>
<td>*****</td>
<td>(Undefined direct assignment)</td>
</tr>
<tr>
<td>R6</td>
<td>%*****</td>
<td>(Undefined register symbol)</td>
</tr>
<tr>
<td>LABEL</td>
<td>*****</td>
<td>(Undefined label)</td>
</tr>
</tbody>
</table>

Generally, undefined symbols and external symbols will be listed as undefined direct assignments.

If the symbol is relocatable or global or both, the symbol's value will be followed by an R, a G or both.

It is possible to output both the binary tape and the assembly listing on the same pass, thereby reducing the assembly process to two passes (see Example 1 below). This will happen automatically unless the binary device and the listing device are conflicting devices or the same device (see Example 2 below). The only conflicting devices are the teleprinter and the low-speed punch. Even though the Assembler deduces that three passes are necessary, the binary and listing can be forced on pass 2 by including /2 in the responses to *B and *L (see Example 3 below).

Example 1. Runs 2 passes:

* S  H  High-speed reader
* B  H  High-speed punch
* L  P  Line Printer
* T  T  Teleprinter
Example 2. Runs 3 passes:

*S    H High-speed reader
*B    H High-speed punch
*L    H High-speed punch
*T    T Teleprinter

Example 3. Runs 2 passes:

*S    H High-speed reader
*B    H/2 High-speed punch on pass 2
*L    H/2 High-speed punch on pass 2
*T    T Teleprinter

Note that there are several cases where the binary output can be intermixed with ASCII output:

a.  *B    H/2 Binary and listing to punch on pass 2.
    *L    H/2

b.  *B    L/E Binary to low-speed punch and error listing to teleprinter (and low-speed punch).

b.  *B    L/2/E Binary, error listing, and
    *L    T/2 listing to low speed punch.

The object module so generated is acceptable to the Linker as long as there are no CTRL/A characters in the source program. The start of every block on the binary tape is indicated by a 001 and the Linker ignores all information until a 001 is detected. Thus, all source and/or error messages will be ignored if they do not contain any CTRL/A characters (octal 001).

If a character other than those mentioned is typed in response to a question, the Assembler will ignore it and print the question again. Example:

*S    H High-speed reader
*B    Q Q is not a valid response
*B    The question is repeated

If at any time you wish to restart the Assembler, type CTRL/P. If the low-speed reader is the source input device, turn it off before typing CTRL/P.

When no passes are omitted or error options specified, the Assembler performs as follows:

PASS 1:

Assembler creates a table of user-defined symbols and their associated values to be used in assembling the source to object program. Undefined symbols (not including external
globals) are listed on the teleprinter at the end of the pass. The symbol table is also listed at this time. If an illegal location statement of the form \( = \) expression is encountered, the line and error code will be printed out on the teleprinter before the assembly proceeds. An error in a location statement is usually a fatal error in the program and should be corrected.

PASS 2:

Assembler punches the object module, and prints the pass error count and undefined location statements on the teleprinter.

PASS 3:

Assembler prints or punches the assembly program listing, undefined location statements, and the pass error count on the teleprinter.

The functions of passes 2 and 3 will occur simultaneously on pass 2 if the binary and listing devices are different, and do not conflict with each other (the low-speed punch and teleprinter conflict). Furthermore, if the binary object module is not requested, the listing will be produced on pass 2.

The following table summarizes the initial dialogue questions:

<table>
<thead>
<tr>
<th>PRINTOUT</th>
<th>INQUIRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>*S</td>
<td>What is the input device of the Source symbolic tape?</td>
</tr>
<tr>
<td>*B</td>
<td>What is the output device of the Binary object tape?</td>
</tr>
<tr>
<td>*L</td>
<td>What is the output device of the assembly Listing?</td>
</tr>
<tr>
<td>*T</td>
<td>What is the output device of the symbol Table?</td>
</tr>
</tbody>
</table>

The following table summarizes the legal responses:

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>RESPONSE INDICATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>Teleprinter keyboard</td>
</tr>
<tr>
<td>L</td>
<td>Low-speed reader or punch</td>
</tr>
<tr>
<td>H</td>
<td>High-speed reader or punch</td>
</tr>
<tr>
<td>P</td>
<td>Line Printer</td>
</tr>
<tr>
<td>/1</td>
<td>Pass 1</td>
</tr>
<tr>
<td>/2</td>
<td>Pass 2</td>
</tr>
<tr>
<td>/3</td>
<td>Pass 3</td>
</tr>
<tr>
<td>/E</td>
<td>Errors listed on same pass (not meaningful response to *S or *T)</td>
</tr>
<tr>
<td>↓</td>
<td>Omit function (except in response to *S).</td>
</tr>
</tbody>
</table>

Typical examples of complete initial dialogues:

For minimal PDP-11 configuration:
9.4 Assembly Dialogue

During assembly, the Assembler will pause to print on the teleprinter various messages to indicate that you must respond in some way before the assembly process can continue. You may also type CTRL/P, at any time, if you wish to stop the assembly process and restart the initial dialogue, as mentioned in the previous section.

When a .EOT assembler directive is read on the tape, the Assembler prints

**EOF ?**

and pauses. During this pause, the next tape is placed in the reader, and RETURN is typed to continue the assembly.

If the specified assembly listing output device is the high-speed punch and if it is out of tape, or if the device is the Line Printer and is out of paper, the Assembler prints on the teleprinter

**EOM ?**

and waits for tape or paper to be placed in the device. Type the RETURN key when the tape or paper has been replenished; assembly will continue.

Conditions causing the EOM ? messages for an assembly listing device are:

<table>
<thead>
<tr>
<th>HSP</th>
<th>LPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>No power</td>
<td>No power</td>
</tr>
<tr>
<td>No tape</td>
<td>Printer drum gate open</td>
</tr>
<tr>
<td></td>
<td>Too hot</td>
</tr>
<tr>
<td></td>
<td>No paper</td>
</tr>
</tbody>
</table>

There is no EOM if the line printer is switched off-line, although characters may be lost for this condition as well as for an EOM.
If the binary output device is the high-speed punch and if it is out of tape, the Assembler prints:

\[ \text{EOM ?} \]

The assembly process is aborted and the initial dialogue is begun again.

When a .END assembler directive is read on the tape, the Assembler prints:

\[ \text{END ?} \]

and pauses. During the pause the first tape is placed in the reader, and the RETURN key is typed to begin the next pass. On the last pass, the .END directive causes the Assembler to begin the initial dialogue for the next assembly.

If you are starting the binary pass and the binary is to be punched on the low-speed punch, turn the punch on before typing the RETURN key for starting the pass. The carriage return and line feed characters will be punched onto the binary tape, but the Linker will ignore them.

If the last tape ends with a .EOT, the Assembler may be told to emulate a .END assembler directive by responding with E followed by the RETURN key. The Assembler will then print

\[ \text{END ?} \]

and wait for another RETURN before starting the next pass. Example:

\[ \text{EOF ? E} \]
\[ \text{END ?} \]

Note that forcing a .END in this manner causes the error counter to be incremented by one.

9.5 Assembly Listing

PAL-11S produces a side-by-side assembly listing of symbolic source statements, their octal equivalents, assigned addresses, and error codes, as follows:

\[ \text{EELLLLLLL OOOOOOASSS.......S} \]
\[ \text{OOOOOO} \]
\[ \text{OOOOOO} \]

The E's represent the error field. The L's represent the address. The O's represent the object data in octal. The S's represent the source statement. "A" represents a single apostrophe which indicates that either the second, third or both words of the instruction will be modified by the Linker. While the Assembler accepts 72 characters per line on input, the listing is reduced by the 16 characters to the left of the source statement.
The above represents a three-word statement. The second and third words of the statement are listed under the command word. No addresses precede the second and third words since the address order is sequential.

The third line is omitted for a two-word statement; both second and third lines are omitted for a one-word statement.

For a .BYTE directive, the object data field is three octal digits.

For a direct assignment statement, the value of the defining expression is given in the object code field although it is not actually part of the code of the object program.

The .ASECT and .CSECT directives cause the current value of the appropriate location counter (absolute or relocatable) to be printed.

Each page of the listing is headed by a page number (octal).

9.6 Object Module Output

The output of the assembler during the binary object pass is an object module which is meaningful only to the linker. What follows gives an overview of what the object module contains and at what stage each part of it is produced.

The binary object module consists of three main types of data block:

a) Global symbol directory (GSD)
b) Text blocks (TXT)
c) Relocation Directory (RLD)

9.6.1 Global Symbol Directory

As the name suggests, the GSD contains a list of all the global symbols together with the name of the object module. Each symbol is in Radix-50 form and contains information regarding its mode and value whenever known.

The GSD is created at the start of the binary object pass.

9.6.2 Text Block

The text blocks consist entirely of the binary object data as shown in the listing. The operands are in the unmodified form.

9.6.3 Relocation Directory

The RLD blocks consist of directives to the Linker which may reference the text block preceding the RLD. These directives control the relocation and linking process.

Text and RLD blocks are constructed during the binary object pass. Outputting of each
block is done whenever either the TXT or RLD buffer is full and whenever the location counter needs to be modified.

## 10.0 ERROR CODES

The error codes printed beside the octal and symbolic code in the assembly listing have the following meanings:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Addressing error. An address within the instruction is incorrect. Also may indicate a relocation error.</td>
</tr>
<tr>
<td>B</td>
<td>Bounding error. Instructions or word data are being assembled at an odd address in memory. The location counter is updated by +1.</td>
</tr>
<tr>
<td>D</td>
<td>Doubly-defined symbol referenced. Reference was made to a symbol which is defined more than once.</td>
</tr>
<tr>
<td>I</td>
<td>Illegal character detected. Illegal characters which are also non-printing are replaced by a ? on the listing.</td>
</tr>
<tr>
<td>L</td>
<td>Line buffer overflow. Extra characters on a line (more than 72\textsubscript{10}) are ignored.</td>
</tr>
<tr>
<td>M</td>
<td>Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.</td>
</tr>
<tr>
<td>N</td>
<td>Number containing 8 or 9 has decimal point missing.</td>
</tr>
<tr>
<td>P</td>
<td>Phase error. A label's definition or value varies from one pass to another.</td>
</tr>
<tr>
<td>Q</td>
<td>Questionable syntax. There are missing arguments or the instruction scan was not completed or a carriage return was not immediately followed by a line feed or form feed.</td>
</tr>
<tr>
<td>R</td>
<td>Register-type error. An invalid use of or reference to a register has been made.</td>
</tr>
<tr>
<td>S</td>
<td>Symbol table overflow. When the quantity of user-defined symbols exceeds the allocated space available in the user's symbol table, the assembler outputs the current source line with the S error code, then returns to the initial dialogue.</td>
</tr>
</tbody>
</table>
Error Code | Meaning
---|---
T | Truncation error. A number generated more than 16 bits of significance or an expression generated more than 8 bits of significance during the use of the .BYTE directive.
U | Undefined symbol. An undefined symbol was encountered during the evaluation of an expression. Relative to the expression, the undefined symbol is assigned a value of zero.

II.0 SOFTWARE ERROR HALTS

PAL-11S loads all of its unused trap vectors with the code

```
.WORD .+2, HALT
```

so that if the trap does occur, the processor will halt in the second word of the vector. The address of the halt, displayed in the console address register, therefore indicates the cause of the halt.

<table>
<thead>
<tr>
<th>Address of Halt (octal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Reserved instruction executed</td>
</tr>
<tr>
<td>16</td>
<td>Trace trap occurred</td>
</tr>
<tr>
<td>26</td>
<td>Power fail trap</td>
</tr>
<tr>
<td>32</td>
<td>EMT executed</td>
</tr>
</tbody>
</table>

A halt at address 40 indicates an IOXLPT detected error. R0 (displayed in the console lights) contains an identifying code:

<table>
<thead>
<tr>
<th>Code in R0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Illegal memory reference, SP overflow or illegal instruction.</td>
</tr>
<tr>
<td>1</td>
<td>Illegal IOX command.</td>
</tr>
<tr>
<td>2</td>
<td>Slot number out of range.</td>
</tr>
<tr>
<td>3</td>
<td>Device number illegal</td>
</tr>
<tr>
<td>4</td>
<td>Referenced slot not INITed.</td>
</tr>
<tr>
<td>5</td>
<td>Illegal Data Mode.</td>
</tr>
</tbody>
</table>

IOXLPT also sets R1 as follows:

If the error code is 0, R1 contains the PC at the time of the error.

If the error code is 1-5, R1 points to some element in the IOT argument list or to the instruction following the argument list, depending on whether IOXLPT has finished decoding all the arguments when it detects the error.
CHAPTER 2
LINK - 11S LINKER

Contents

1.0 INTRODUCTION 1
1.1 General Description 1
1.2 Absolute and Relocatable Program Sections 2
1.3 Global Symbols 2

2.0 INPUT AND OUTPUT 3
2.1 Object Module 3
2.2 Load Module 3
2.3 Load Map 4

3.0 OPERATING PROCEDURES 5
3.1 Loading and Command String 5
3.1.1 Operational Cautions 7
3.2 Error Procedures and Messages 7
3.2.1 Restarting 7
3.2.2 Non-Fatal Errors 7
3.2.3 Fatal Errors 8
3.2.4 Error Halts 9

4.0 PREPARATION 11
1.0 INTRODUCTION

1.1 General Description

LINK-11S (stand alone) is a PDP-11 system program designed to link and relocate programs previously assembled by PAL-11S. This capability allows the user to separately assemble his main program and each of his various subroutines without assigning an absolute load address at assembly time. The binary output of each assembly (called object modules) is processed by LINK-11S (hereafter called the Linker or LINK-11) to:

a. Relocate each object module and assign absolute addresses.

b. Link the modules by correlating global symbols defined in one module and referenced in another module.

c. Print a load map which displays the assigned absolute addresses.

d. Punch a load module which can subsequently be loaded (by the Absolute loader) and executed.

Some of the advantages of using PAL-11S and LINK-11 are:

a. The program is divided into segments (usually subroutines) which are assembled separately. If an error is discovered in one segment, only that segment needs to be reassembled. The new object module is then linked with the other object modules.

b. Absolute addresses need not be assigned at assembly time. The Linker automatically assigns absolute addresses. This keeps programs from overlaying each other. This also allows subroutines to change size without influencing the placement of other routines.

c. Separate assemblies allow the total number of symbols to exceed the number allowed in a single assembly.

d. Internal symbols (symbols which are not global) need not be unique among object modules. Thus, naming rules are required only for global symbols when separate programmers prepare separate subroutines of a single program.
e. Subroutines may be provided for general use in object module form to be linked into the user's program.

LINK-11 is designed to run on an 8K PDP-11 with an ASR-33. A PC11 (high speed paper tape reader and punch) and an LP11 (line printer) may be used if available. The PC11 significantly speeds up the linking process. An LP11 provides a fast device for the load map listing.

1.2 Absolute and Relocatable Program Sections

A program assembled by PAL-11S may consist of an absolute program section, declared by the .ASECT assembler directive, and a relocatable program section, declared by the .CSECT assembler directive. (If a program has neither an .ASECT or .CSECT directive, the assembler implicitly assumes a .CSECT directive.) The program and data in the absolute section are assigned absolute addresses as specified by the location counter setting statements (.A=x). The program and data in the relocatable section are assigned absolute addresses by the linker. Addresses are normally assigned such that the relocatable section is at the high end of memory. The assignment of addresses may be influenced by command string options (see Section 3.2).

The Linker appropriately modifies all instructions and/or data as necessary to account for the relocation of the control section.

LINK-11 has the capability to handle object modules containing named control (relocatable) sections as generated by PAL-11R. However, PAL-11S can only create the unnamed control section (which has the special default name of 6 blanks) and the absolute section (with the special name .ABS.). The unnamed control section is internal to each object module. That is, every object module may have an unnamed control section (each with the name 6 blanks) but the Linker treats them independently. Each is assigned an absolute address such that they occupy mutually exclusive areas of memory. Named control sections, on the other hand, are treated globally. That is, if different object modules each have control sections with the same name, they are all assigned the same absolute load address and the size of the area reserved for loading of the section is the maximum of the sizes of each section. Thus, named control sections allow the sharing of data and/or programs among object modules. This is very similar to the handling and function of labelled COMMON in FORTRAN IV.

A restriction of LINK-11S is that the name of a control section must not be the same as the name of a global entry symbol. This will result in multiple definition errors.

1.3 Global Symbols

Global symbols provide the links or communication between object modules (or assemblies). Global symbols are created with the .GLOBAL assembler directive. Symbols which are not global are called internal symbols. If the global symbol is defined (as a label or direct assignment) in an object module it is called an entry symbol, and other object modules may reference it. If the global symbol is not defined in the object module it is an external symbol. It is assumed to be defined (as an entry symbol) in some other object module.

As the Linker reads the object modules it keeps track of all the global symbol definitions and references. It then modifies the instructions and/or data which reference the global symbols.
2.0 INPUT AND OUTPUT

2.1 Object Module

LINK-11's input is the object module. This is the output of PAL-11S (or any other program which can create an object module). The Linker reads each object module twice; that is, it is a two pass processor.

On pass 1, the Linker reads each object module to gather enough information so that absolute addresses can be assigned to all relocatable sections and all globals can be assigned absolute values. This information appears in the global symbol directory (GSD) of the object module.

On pass 2, the Linker reads all of each object module and produces the load module (see Section 2.2). The data gathered on pass 1 guides the relocation and linking process on pass 2.

2.2 Load Modules

The normal output of the Linker is a load module which may be loaded and run.

A load module consists of formatted binary blocks holding absolute load addresses and object data as specified for the Paper Tape System Absolute Loader and the PDP-11 Disk Monitor. The first few words of data will be the communications directory (COMD) and will have an absolute load address equal to the lowest relocated address of the program. The absolute loader will load the COMD at the specified address but then the program will overlay the COMD*. The disk monitor loader will expect the COMD and will load it where the monitor wants it. The end of the load module will be indicated by a TRA block; that is, a block containing only a load address. The byte count in the formatted binary block will be 6 on this block; on all other blocks the byte count will be larger than 6. The TRA (transfer address) is selected by the Linker to be the first even transfer address seen. Thus, if four object modules are linked together and if the first and second had a .END statement, the third had a .END A and the fourth had a .END B, the transfer address would be A of module three.

*Note:
The overlaying of the COMD by the relocated program is a trick to allow the Absolute Loader to handle load modules with a COMD. However, a problem arises if a load module is to be loaded by the absolute loader and either of the following conditions is true:

a. The object modules used to construct the load module contained no relocatable code; or

b. The total sizes of the relocatable code is less than 20(10) bytes (the size of the COMD).

In either case, there is not enough relocatable code to overlay the COMD which means the COMD will load into parts of memory not intended to be altered by the user. The COMD's load address, selected by the linker in the above cases, is such that it will be up against the current top of memory (see *T option in section 3.1). If the top happens to be very low, the linker will not allow the COMD to be loaded below address 0; it will load it at 0.
2.3 Load Map

The load map provides several types of information concerning the load module's make-up. The map begins with an indication of the low and high limits of the relocatable code and the transfer address. Then there is a section of the map for each object module included in the linking process. Each of these sections begins with the module's name followed by a list of the control sections and the entry points for each control section. For each control section, the base of the section (its low address) and its size (in bytes) is printed to the right of the section name (enclosed in angle brackets). Following each section name printout is a list of entry points and their addresses. After all information has been printed for each object module, any undefined symbols are listed. Note that modules are loaded such that if modules A, B and C are linked together, A is lowest and C is highest in memory.

The format is quite self-explanatory as can be seen from the following example:

LOAD MAP

TRANSFER ADDRESS: $37434
LOW LIMIT: $37406
HIGH LIMIT: $37460
**********

MODULE MOD1
SECTION ENTRY ADDRESS SIZE
<.ABS.> 000000 000000
< > $37406 000044
X3 $37452
X4 $37440
X5 $37450
X7 $37430
**********

MODULE MOD2
SECTION ENTRY ADDRESS SIZE
< > $37452 000006
X1 $37452
X2 $37452
**********

UNDEFINED REFERENCES
X6
3.0 OPERATING PROCEDURES

3.1 Loading and Command String

The Linker is loaded by the Absolute Loader and is self-starting. It will use a simple command dialogue which allows the object module, load module and load map devices to be specified. During pass 1 and pass 2, the Linker asks for each object module individually.

Note: The non-printing characters carriage return, line feed and space are represented in this chapter as <CR>, <LF> and <SPACE>.

Operation begins by the linker typing its name and version. This is followed by the input option printed as *I<SPACE>. The responses are:

- <CR>  Read object module from HSR.
- H<CR>  Read object module from HSR.
- L<CR>  Read object module from LSR

The input option is followed by the output option *O<SPACE>. The responses are:

- <CR>  Punch load module on HSP.
- H<CR>  Punch load module on HSP.
- L<CR>  Punch load module on LSP.

LINK-1I asks if a load map is desired by typing *M<SPACE>. The legal responses are
- <CR>  for no map, T<CR> or H<CR> or P<CR> for a map on the teleprinter, high-speed punch, or line printer, respectively.

The next two options concern the placement of the relocated object program in memory. The standard version of the Linker assumes it is linking for an 8K machine. It relocates the program such that it is as high as possible in 8K but leaves room for the Absolute and Boot Loaders.

These assumed values may be changed by altering parameters HGHMEM (highest legal memory address +1) and ALODSZ (number of bytes allocated for Absolute Loader and Boot Loader) and reassembling the Linker. The user may control where a program is relocated to with the *T and *B options. After the option *T<SPACE> has been typed, the user may respond as follows:

- <CR>  Relocate so that program is up against the current top of memory. If the top has not been changed, then the top is the assembled-in top (GHGHMEM-ALODSZ). The standard assumption is 16384.-112.=16272 (37460(8)).

- N<CR>  N is an octal number (unsigned) which defines a new top address.

If a new top is specified, the *B option is suppressed.

After the option *B<SPACE> has been printed the user may respond as follows:
<CR> Use current top of memory.

N<CR> N is an unsigned octal number which defines the bottom address of the program. That is, a new top of memory is calculated so that the bottom of the program corresponds with N.

Once a top of memory has been calculated (by *T or *B), that value is used until it is changed.

LINK-11 indicates the start of pass one by typing PASS 1. The input is requested by the Linker, one tape at a time, by typing *<SPACE>. The legal responses are:

<CR> Read a tape and request more input.

U<CR> List all undefined globals on the teleprinter and request more input.

E<CR> End of input. If there are undefined globals, list them on the teleprinter and request more input. Otherwise print the load map, if requested, and enter pass 2.

C<CR> End of input. Assign Ø to any undefined globals, print the load map (if requested), and enter pass 2.

The Linker indicates the start of pass 2 by typing PASS 2. It then requests each input tape as in pass 1.

A <CR> is the only useful response to * on pass 2. The modules must be read on pass 2 in the same order as pass 1. When the last module has been read the Linker will automatically finish the load module and restart itself.

Leader and trailer will be punched on the load module.

If the LSP is being used for the load module output, it should be turned on before pass 2 begins. Thus, turn it on before typing E<CR> or C<CR>. The echo of these characters (and the load map, if printed on the TTY) will be punched on the load module but may be easily removed since leader is punched on the load module. In any case, ASCII information in a load module will be ignored by the Absolute and Disk Monitor loaders. However, the LSP can be turned on while leader is being punched (after the linker has typed PASS 2) to keep the load map, etc., from being punched onto the tape.

Note:

On all command string options, except for *T and *B, the linker only examines the last character typed preceding the carriage return. Thus,

ABCDE<CR>

is equivalent to H<CR>.
3.1.1 Operational Cautions

The Linker does not give a warning if a program is linked so low in memory that it goes below address 0. However, this case is easily seen by examining the low and high limits which are always printed (on the load map or on the teleprinter).

The Linker reads object modules until an end of medium is detected. Object modules from the DEC Program Library contain a special checksum at the end of the tape which must be removed before they are linked. Failure to remove this checksum can result in fatal Linker errors.

3.2 Error Procedure and Messages

3.2.1 Restarting

Control/P (symbolized as ↑ P) is used for two purposes by LINK-11. If a ↑ P is typed while a load map is being printed, the load map will be aborted and the Linker will continue. A ↑ P typed at any other time will cause the Linker to restart itself.

3.2.2 Non-Fatal Errors

a. Non-unique object module name - this error is detected during pass 1 and results in an error message and the module is rejected. The message is:

?MODULE NAME XXXXXX NOT UNIQUE

The Linker will then ask for more input.

b. Load map device EOM - this error allows the user an option to fix the device and continue or abort the map listing. The Linker prints:

?MAP DEVICE EOM.
TYPE <CR> TO CONTINUE

Any response, terminated by <CR> or <LF> will cause the Linker to continue. A ↑ P will cause the map be to aborted.

c. A byte relocation error - the Linker will try to relocate and link byte quantities. However, relocation will usually fail and linking may fail. Failure is defined as the high byte of the relocated value (or the linked value) not being all zero. In such a case, the value is truncated to 8 bits and the following message is printed:

?BYTE RELOC ERROR AT ABS ADDRESS XXXXXX.

The linker automatically continues.
d. If the object modules are not read in the same order on pass 2 as pass 1, the Linker will indicate which module should be loaded next by typing:

?LOAD XXXXXX NEXT!

The linker will then ask for more input.

e. Multiply-Defined Globals - this results, during pass 1, in the following error message:

?XXXXXX MULTIPLY DEFINED BY MODULE XXXXXX.

The second definition is ignored and the Linker continues.

3.2.3 Fatal Errors

All of the following errors cause the indicated error message to be printed and the Linker is restarted.

a. Symbol Table overflow - the message is:

?SYMBOL TABLE OVERFLOW - MODULE XXXXXX, SYMBOL XXXXXX

b. System Errors - this class of errors prints:

?SYSTEM ERROR XX

where XX is an identifying number as follows:

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø1</td>
<td>Unrecognized symbol table entry found.</td>
</tr>
<tr>
<td>Ø2</td>
<td>A relocation directory references a global name which cannot be found in the symbol table.</td>
</tr>
<tr>
<td>Ø3</td>
<td>A relocation directory contains a location counter modification command which is not last.</td>
</tr>
<tr>
<td>Ø4</td>
<td>Object module does not start with a GSD.</td>
</tr>
<tr>
<td>Ø5</td>
<td>The first entry in the GSD is not the module name.</td>
</tr>
</tbody>
</table>
6. An RLD references a section name which cannot be found.
7. The TRA specification references a non-existent module name.
8. The TRA specification references a non-existent section name.
9. An internal jump table index is out of range.
10. A checksum error occurred on the object module.
11. An object module binary block is too big (more than 64(10) words of data).
12. A device error occurred on the load module output device.

All system errors except for numbers 10 and 12 indicate a program failure either in the Linker or the program which generated the object module. Error 6 can occur if a tape is read which is not an object module.

3.2.4 Error HALTs

LINK-11 loads all of its unused trap vectors with the code:

```
.WORD .+2, HALT
```

so that if the trap occurs, the processor will halt in the second word of the vector. The address of the halt, displayed in the console lights, therefore indicates the cause of the halt.

<table>
<thead>
<tr>
<th>Address of HALT (octal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Reserved instruction executed.</td>
</tr>
<tr>
<td>16</td>
<td>Trace trap occurred.</td>
</tr>
<tr>
<td>26</td>
<td>Power fail trap.</td>
</tr>
<tr>
<td>32</td>
<td>EMT executed.</td>
</tr>
</tbody>
</table>
A halt at address 40 indicates an IOXLPT detected error. R0 (displayed in the console lights) contains an identifying code:

<table>
<thead>
<tr>
<th>Code in R0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Illegal memory reference, SP overflow or illegal instruction.</td>
</tr>
<tr>
<td>1</td>
<td>Illegal IOX command.</td>
</tr>
<tr>
<td>2</td>
<td>Slot number out of range.</td>
</tr>
<tr>
<td>3</td>
<td>Device number illegal.</td>
</tr>
<tr>
<td>4</td>
<td>Referenced slot not INIT ed.</td>
</tr>
<tr>
<td>5</td>
<td>Illegal data mode.</td>
</tr>
</tbody>
</table>

IOXLPT also sets R1 as follows:

If the error code is 0, R1 contains the PC at the time of the error.

If the error code is 1-5, R1 points to some element in the IOT argument list or to the instruction following the argument list, depending on whether IOXLPT has finished decoding all the arguments when it detects the error.
4.0 PREPARATION

LINK-11S is available as an absolute load module (for an 8K machine), as two object modules (for relinking) and as several ASCII source tapes. There is one object module for the Linker and one for IOXLPT. The supplied object modules may be relinked (using the supplied load module) to load into any size machine larger than 8K. However, the resulting Linker will still assume a top of memory corresponding to an 8K machine (this can be overridden in the command string options). The assumed top of memory and reserved Absolute Loader space may be changed by editing the first linker ASCII tape with ED-11. The parameters to be changed are HGHMEM (high memory address +1 (always even)) and ALODSZ (Absolute Loader size (always even)). The source tapes for the Linker may then be assembled with PAL-11S and the new object module can then replace the supplied Linker object module.

The tapes are identified as follows:

<table>
<thead>
<tr>
<th>Library Code</th>
<th>Tape 1 of 6</th>
<th>Tape 2 of 6</th>
<th>Tape 3 of 6</th>
<th>Tape 4 of 6</th>
<th>Tape 5 of 6</th>
<th>Tape 6 of 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC-11-ZLQA-PA</td>
<td>LINK-11S (Main Program)</td>
<td>One</td>
<td>Assembly</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC-11-ZLQA-PR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC-11-ZLQA-PL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC-11-ZLQA-PA</td>
<td>IOXLPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC-11-ZLQA-PR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC-11-ZLQA-PL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tape 1 of 2
Tape 2 of 2
LINK-11S Object Module
IOXLPT Object Module
LINK-11S Load Module
## APPENDIX A

### ASCII CHARACTER SET

<table>
<thead>
<tr>
<th>EVEN PARITY BIT</th>
<th>7-BIT OCTAL CODE</th>
<th>CHARACTER</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>NUL</td>
<td>NULL, TAPE FEED, CONTROL SHIFT P.</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>START OF HEADING; ALSO SOM, START OF MESSAGE, CONTROL A,</td>
</tr>
<tr>
<td>1</td>
<td>002</td>
<td>STX</td>
<td>START OF TEXT; ALSO EOA, END OF ADDRESS, CONTROL B,</td>
</tr>
<tr>
<td>0</td>
<td>003</td>
<td>ETX</td>
<td>END OF TEXT: ALSO EOM, END OF MESSAGE, CONTROL C,</td>
</tr>
<tr>
<td>1</td>
<td>004</td>
<td>EOT</td>
<td>END OF TRANSMISSION(END): SHUTS OF TWX MACHINES, CONTROL D,</td>
</tr>
<tr>
<td>0</td>
<td>005</td>
<td>ENQ</td>
<td>ENQUIRY (ENQRY); ALSO WRU, CONTROL E,</td>
</tr>
<tr>
<td>0</td>
<td>006</td>
<td>ACK</td>
<td>ACKNOWLEDGE. ALSO RU, CONTROL F.</td>
</tr>
<tr>
<td>1</td>
<td>007</td>
<td>BEL</td>
<td>RINGS THE BELL. CONTROL G.</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
<td>BS</td>
<td>BACKSPACE: ALSO FEO, FORMAT EFFECTOR. BACKSPACE SOME MACHINES, CONTROL H.</td>
</tr>
<tr>
<td>0</td>
<td>011</td>
<td>HT</td>
<td>HORIZONTAL TAB. CONTROL I.</td>
</tr>
<tr>
<td>0</td>
<td>012</td>
<td>LF</td>
<td>LINE FEED OR LINE SPACE (NEW LINE): ADVANCES PAPER TO NEXT LINE, DUPLICATED BY CONTROL J.</td>
</tr>
<tr>
<td>1</td>
<td>013</td>
<td>VT</td>
<td>VERTICAL TAB (VTAB). CONTROL K.</td>
</tr>
<tr>
<td>0</td>
<td>014</td>
<td>FF</td>
<td>FORM FEED TO TOP OF NEXT PAGE (PAGE). CONTROL L.</td>
</tr>
<tr>
<td>1</td>
<td>015</td>
<td>CR</td>
<td>CARRIAGE RETURN TO BEGINNING OF LINE. DUPLICATED BY CONTROL M.</td>
</tr>
<tr>
<td>1</td>
<td>016</td>
<td>SO</td>
<td>SHIFT OUT: CHANGES RIBBON COLOR TO RED. CONTROL N.</td>
</tr>
<tr>
<td>0</td>
<td>017</td>
<td>SI</td>
<td>SHIFT IN: CHANGES RIBBON COLOR TO BLACK. CONTROL O.</td>
</tr>
<tr>
<td>1</td>
<td>020</td>
<td>DLE</td>
<td>DATA LINK ESCAPE. CONTROL P (DC0).</td>
</tr>
<tr>
<td>0</td>
<td>021</td>
<td>DC1</td>
<td>DEVICE CONTROL 1, TURNS TRANSMITTER (READER) ON, CONTROL Q (XON).</td>
</tr>
<tr>
<td>0</td>
<td>022</td>
<td>DC2</td>
<td>DEVICE CONTROL 2, TURNS PUNCH OR AUXILIARY ON. CONTROL R (TAPE, AUX ON).</td>
</tr>
<tr>
<td>1</td>
<td>023</td>
<td>DC3</td>
<td>DEVICE CONTROL E, TURNS TRANSMITTER (READER) OFF, CONTROL S (XOFF).</td>
</tr>
<tr>
<td>0</td>
<td>024</td>
<td>DC4</td>
<td>DEVICE CONTROL 4. TURNS PUNCH OR AUXILIARY OFF. CONTROL T (TAPE, AUX OFF).</td>
</tr>
<tr>
<td>1</td>
<td>025</td>
<td>NAK</td>
<td>NEGATIVE ACKNOWLEDGE: ALSO ERR. ERROR. CONTROL U.</td>
</tr>
<tr>
<td>1</td>
<td>026</td>
<td>SYN</td>
<td>SYNCHRONOUS IDLE (SYNC). CONTROL V.</td>
</tr>
<tr>
<td>0</td>
<td>027</td>
<td>ETB</td>
<td>END OF TRANSMISSION BLOCK: ALSO LEM. LOGICAL END OF MEDIUM. CONTROL W.</td>
</tr>
<tr>
<td>0</td>
<td>030</td>
<td>CAN</td>
<td>CANCEL (CANCL). CONTROL X.</td>
</tr>
<tr>
<td>1</td>
<td>031</td>
<td>EM</td>
<td>END OF MEDIUM. CONTROL Y.</td>
</tr>
<tr>
<td>1</td>
<td>032</td>
<td>SUB</td>
<td>SUBSTITUTE. CONTROL Z.</td>
</tr>
<tr>
<td>0</td>
<td>033</td>
<td>ESC</td>
<td>ESCAPE. PREFIX.</td>
</tr>
<tr>
<td>1</td>
<td>034</td>
<td>FS</td>
<td>FILE SEPARATOR. CONTROL SHIFT L.</td>
</tr>
<tr>
<td>0</td>
<td>035</td>
<td>GS</td>
<td>GROUP SEPARATOR. CONTROL SHIFT M.</td>
</tr>
<tr>
<td>0</td>
<td>036</td>
<td>RS</td>
<td>RECORD SEPARATOR. CONTROL SHIFT N.</td>
</tr>
<tr>
<td>1</td>
<td>037</td>
<td>US</td>
<td>UNIT SEPARATOR. CONTROL SHIFT O.</td>
</tr>
<tr>
<td>1</td>
<td>040</td>
<td>SP</td>
<td>SPACE.</td>
</tr>
<tr>
<td>EVEN PARITY BIT</td>
<td>7-BIT OCTAL CODE</td>
<td>CHARACTER</td>
<td>REMARKS</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>0</td>
<td>041</td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>042</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>043</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>044</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>045</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>046</td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>047</td>
<td>'</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>050</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>051</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>052</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>053</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>054</td>
<td>,</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>055</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>056</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>057</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>060</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>061</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>062</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>063</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>064</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>065</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>066</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>067</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>070</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>071</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>072</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>073</td>
<td>;</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>074</td>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>075</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>076</td>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>077</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>@</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>101</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>102</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>103</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>104</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>105</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>106</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>107</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>110</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>112</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>113</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>114</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>115</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>116</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>117</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>120</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>121</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>122</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>123</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>124</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>125</td>
<td>U</td>
<td></td>
</tr>
</tbody>
</table>

ACCUTE ACCENT OR APOSTROPHE.
<table>
<thead>
<tr>
<th>EVEN PARITY</th>
<th>7-BIT OCTAL</th>
<th>CHARACTER</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>126</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>127</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>131</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>132</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>133</td>
<td>[</td>
<td>SHIFT K</td>
</tr>
<tr>
<td>0</td>
<td>134</td>
<td>]</td>
<td>SHIFT L</td>
</tr>
<tr>
<td>1</td>
<td>135</td>
<td>)</td>
<td>SHIFT M</td>
</tr>
<tr>
<td>1</td>
<td>136</td>
<td>↑</td>
<td>SHIFT N</td>
</tr>
<tr>
<td>0</td>
<td>137</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>140</td>
<td></td>
<td>ACCENT GRAVE.</td>
</tr>
<tr>
<td>0</td>
<td>175</td>
<td></td>
<td>THIS CODE GENERATED BY ALT MODE.</td>
</tr>
<tr>
<td>0</td>
<td>176</td>
<td></td>
<td>THIS CODE GENERATED BY ESC KEY (IF PRESENT)</td>
</tr>
<tr>
<td>1</td>
<td>177</td>
<td>DEL</td>
<td>DELETE, RUB OUT.</td>
</tr>
<tr>
<td>1</td>
<td>141</td>
<td>a</td>
<td>LOWER CASE ALPHABET FOLLOWS (TELETETYPE MODEL 37 ONLY).</td>
</tr>
<tr>
<td>1</td>
<td>142</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>143</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>144</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>145</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>146</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>147</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>150</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>151</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>152</td>
<td>j</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>153</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>154</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>155</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>156</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>157</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>160</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>161</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>162</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>163</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>164</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>165</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>166</td>
<td>v</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>167</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>170</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>171</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>172</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>174</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B.1 TERMINATORS

The list below defines all characters which are considered to be terminators. The order of the list implies the descending hierarchy of significance.

<table>
<thead>
<tr>
<th>Character</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL/FORM</td>
<td>Source line terminator.</td>
</tr>
<tr>
<td>LINE FEED</td>
<td>Source line terminator.</td>
</tr>
<tr>
<td>RETURN</td>
<td>Source line terminator.</td>
</tr>
<tr>
<td>:</td>
<td>Label terminator</td>
</tr>
<tr>
<td>=</td>
<td>Direct assignment delineator</td>
</tr>
<tr>
<td>%</td>
<td>Register term delineator</td>
</tr>
<tr>
<td>TAB</td>
<td>Item terminator</td>
</tr>
<tr>
<td></td>
<td>Field terminator</td>
</tr>
<tr>
<td>BLANK or SPACE</td>
<td>Item terminator</td>
</tr>
<tr>
<td></td>
<td>Field terminator</td>
</tr>
<tr>
<td>#</td>
<td>Immediate expression field indicator</td>
</tr>
<tr>
<td>@</td>
<td>Deferred addressing indicator</td>
</tr>
<tr>
<td>(</td>
<td>Initial register field indicator</td>
</tr>
<tr>
<td>)</td>
<td>Terminal register field indicator</td>
</tr>
<tr>
<td>'</td>
<td>Operand field separator</td>
</tr>
<tr>
<td>;</td>
<td>Comments field delimiter</td>
</tr>
<tr>
<td>+</td>
<td>Arithmetic addition operator</td>
</tr>
<tr>
<td>-</td>
<td>Arithmetic subtraction operator</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical AND operator</td>
</tr>
<tr>
<td>!</td>
<td>Logical OR operator</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double ASCII text indicator</td>
</tr>
<tr>
<td>'</td>
<td>Single ASCII text indicator</td>
</tr>
</tbody>
</table>


B.2 ADDRESS MODE SYNTAX

`r` is an integer between 0 and 7.

`R` is a register expression, `E` is an expression, and `ER` is either a register expression or an absolute expression in the range of 0 to 7.

<table>
<thead>
<tr>
<th>Address Mode Number</th>
<th>Address Mode Name Field</th>
<th>Symbol in Operand</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0r</td>
<td>Register</td>
<td>R</td>
<td>Register R contains the operand. R is a register expression.</td>
</tr>
<tr>
<td>1r</td>
<td>Deferred Register @R or (R)</td>
<td></td>
<td>Register R contains the operand address.</td>
</tr>
<tr>
<td>2r</td>
<td>Autoincrement</td>
<td>(ER) +</td>
<td>The contents of the register specified by ER is incremented after being used as the address of the operand.</td>
</tr>
<tr>
<td>3r</td>
<td>Autoincrement</td>
<td>@(ER) +</td>
<td>ER contains the pointer to the address of the operand. ER is incremented after use.</td>
</tr>
<tr>
<td>4r</td>
<td>Autodecrement</td>
<td>-(ER)</td>
<td>The contents of register ER is decremented before it is used as the address of the operand.</td>
</tr>
<tr>
<td>5r</td>
<td>Autodecrement</td>
<td>@-(ER)</td>
<td>The contents of register ER is decremented before it is used as the pointer to the address of the operand.</td>
</tr>
<tr>
<td>6r</td>
<td>Specified</td>
<td>E(ER)</td>
<td>E plus the contents of the register specified, ER, is the address of the operand.</td>
</tr>
<tr>
<td>7r</td>
<td>Specified</td>
<td>@E(ER)</td>
<td>E added to ER gives the pointer to the address of the operand.</td>
</tr>
<tr>
<td>27</td>
<td>Immediate Operand #E</td>
<td>E</td>
<td>E is the operand.</td>
</tr>
<tr>
<td>37</td>
<td>Absolute address @@E</td>
<td>E</td>
<td>E is the operand address.</td>
</tr>
<tr>
<td>67</td>
<td>Relative address E</td>
<td>E</td>
<td>E is the address of the operand.</td>
</tr>
<tr>
<td>77</td>
<td>Deferred relative address. @E</td>
<td>E</td>
<td>E is the pointer to the address of the operand.</td>
</tr>
</tbody>
</table>
The tables of instructions which follow are grouped according to
the operands they take and according to the bit patterns of their
op-codes.

In the representation of op-codes, the following symbols are used:

- **SS** Source operand specified by a 6-bit
  address mode
- **DD** Destination operand specified by a 6-bit
  address mode
- **XX** 8-bit offset to a
  location (branch instructions)
- **R** Integer between 0 and 7 representing a general
  register

Symbols used in the description of instruction operations are:

- **SE** Source effective address
- **DE** Destination effective address
- **()** contents of
  becomes

The condition codes in the processor status word (PS) are affected
by the instructions; these condition codes are represented as follows:

- **N** Negative bit: set if the result is negative
- **Z** Zero bit: set if the result is zero
- **V** Overflow bit: set if the result had an overflow
- **C** Carry bit: set if the result had a carry

In the representation of the instruction's effect on the condition
codes, the following symbols are used:

- * Conditionally set
- - Not affected
- 0 Cleared
- 1 Set

To set conditionally means to use the instruction's result to deter-
mine the state of the code.
Logical operators are represented by the following symbols:

- **!** Inclusive OR
- **!** Exclusive OR
- **&** AND
- **¬** (used over a symbol) NOT (i.e., l's complement)

### B.3.1 Double Operand Instructions

<table>
<thead>
<tr>
<th>Op-code</th>
<th>MNEMONIC</th>
<th>Stands for</th>
<th>Operation</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>01SSDD</td>
<td>MOV</td>
<td>MOVe</td>
<td>(SE) → DE</td>
<td>* * 0 -</td>
</tr>
<tr>
<td>02SSDD</td>
<td>CMP</td>
<td>CoMPare</td>
<td>(SE)¬(DE)</td>
<td>* * * *</td>
</tr>
<tr>
<td>03SSDD</td>
<td>BIT</td>
<td>BIT Test</td>
<td>(SE)&amp; (DE)</td>
<td>* * 0 -</td>
</tr>
<tr>
<td>04SSDD</td>
<td>BIC</td>
<td>Bit Clear</td>
<td>(SE) &amp; (DE) → DE</td>
<td>* * 0 -</td>
</tr>
<tr>
<td>05SSDD</td>
<td>BIS</td>
<td>Bit Set</td>
<td>(SE) ! (DE) → DE</td>
<td>* * 0 -</td>
</tr>
<tr>
<td>06SSDD</td>
<td>ADD</td>
<td>ADD</td>
<td>(SE) + (DE) → DE</td>
<td>* * * *</td>
</tr>
<tr>
<td>07SSDD</td>
<td>SUB</td>
<td>SUBtract</td>
<td>(DE) - (SE) → DE</td>
<td>* * * *</td>
</tr>
</tbody>
</table>

### B.3.2 Single Operand Instructions

<table>
<thead>
<tr>
<th>Op-code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation:</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0050DD</td>
<td>CLR</td>
<td>CLeaR</td>
<td>0 → DE</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>0051DD</td>
<td>COM</td>
<td>COMplement</td>
<td>(DE) → DE</td>
<td>* * 0 1</td>
</tr>
<tr>
<td>0052DD</td>
<td>INC</td>
<td>INCrement</td>
<td>(DE) + 1 → DE</td>
<td>* * * 1</td>
</tr>
<tr>
<td>0053DD</td>
<td>DEC</td>
<td>DECrement</td>
<td>(DE) - 1 → DE</td>
<td>* * * -</td>
</tr>
<tr>
<td>0054DD</td>
<td>NEG</td>
<td>NEGate</td>
<td>(DE) + 1 → DE</td>
<td>* * * *</td>
</tr>
<tr>
<td>0055DD</td>
<td>ADC</td>
<td>Add Carry</td>
<td>(DE) + (C) → DE</td>
<td>* * * *</td>
</tr>
</tbody>
</table>

B-4
### B.3.3 Rotate/Shift

<table>
<thead>
<tr>
<th>Op-code</th>
<th>MNEMONIC</th>
<th>Stands for</th>
<th>Operation</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0056DD</td>
<td>SBC</td>
<td>Subtract Carry</td>
<td>(DE) - (C) → DE</td>
<td>* * * *</td>
</tr>
<tr>
<td>1056DD</td>
<td>SBCB</td>
<td>Subtract Carry</td>
<td>Byte</td>
<td></td>
</tr>
<tr>
<td>0057DD</td>
<td>TST</td>
<td>Test</td>
<td>(DE) - 0 → DE</td>
<td>* * 0 0</td>
</tr>
<tr>
<td>10577DD</td>
<td>TSTB</td>
<td>Test Byte</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### B.3.4 Operation Instructions

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>MNEMONIC</th>
<th>Stands for</th>
<th>Operation</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>HALT</td>
<td>HALT</td>
<td>The computer stops - - - - all functions.</td>
<td></td>
</tr>
<tr>
<td>0000001</td>
<td>WAIT</td>
<td>WAIT</td>
<td>The computer stops - - - - and waits for an interrupt.</td>
<td></td>
</tr>
<tr>
<td>0000002</td>
<td>RTI</td>
<td>Return from</td>
<td>The PC and ST are - - - - popped off the SP stack:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interrupt</td>
<td>((SP)) → PC (SP)+2 → SP ((SP)) → ST</td>
<td></td>
</tr>
</tbody>
</table>
### Condition Codes

<table>
<thead>
<tr>
<th>Op-code</th>
<th>MNEMONIC</th>
<th>Stands for</th>
<th>Operation</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>000003</td>
<td>000003</td>
<td>breakpoint trap</td>
<td>Trap to location 14. This is used to call ODT-11.</td>
<td>* * * *</td>
</tr>
<tr>
<td>000004</td>
<td>IOT</td>
<td>Input/Output Trap</td>
<td>Trap to location 20. This is used to call IOX.</td>
<td>* * * *</td>
</tr>
<tr>
<td>000005</td>
<td>RESET</td>
<td>RESET</td>
<td>Returns all I/O device handlers to power-on state.</td>
<td>- - - -</td>
</tr>
</tbody>
</table>

| Trapping Op or Op E where 0 ≤ E ≤ 377, 8 |

<table>
<thead>
<tr>
<th>Op-code</th>
<th>MNEMONIC</th>
<th>Stands for</th>
<th>Operation</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>104000-104377</td>
<td>EMT</td>
<td>EMulator Trap</td>
<td>Trap to location 30. This is used to call system programs.</td>
<td>* * * *</td>
</tr>
<tr>
<td>104400-104777</td>
<td>TRAP</td>
<td>TRAP</td>
<td>Trap to location 34. This is used to call any routine desired by the programmer.</td>
<td>* * * *</td>
</tr>
</tbody>
</table>

### Condition Code Operates

<table>
<thead>
<tr>
<th>Op-code</th>
<th>MNEMONIC</th>
<th>Stands for</th>
</tr>
</thead>
<tbody>
<tr>
<td>000241</td>
<td>CLC</td>
<td>Clear Carry bit in PS</td>
</tr>
<tr>
<td>000261</td>
<td>SEC</td>
<td>Set Carry bit.</td>
</tr>
<tr>
<td>000252</td>
<td>CLV</td>
<td>Clear Overflow bit.</td>
</tr>
<tr>
<td>000262</td>
<td>SEV</td>
<td>Set Overflow bit</td>
</tr>
<tr>
<td>000244</td>
<td>CLZ</td>
<td>Clear Zero bit.</td>
</tr>
<tr>
<td>000264</td>
<td>SEZ</td>
<td>Set Zero bit.</td>
</tr>
<tr>
<td>000250</td>
<td>CLN</td>
<td>Clear Negative bit.</td>
</tr>
<tr>
<td>000270</td>
<td>SEN</td>
<td>Set Negative bit.</td>
</tr>
<tr>
<td>000254</td>
<td>CNZ</td>
<td>Clear Negative and Zero bits.</td>
</tr>
<tr>
<td>000257</td>
<td>CCC</td>
<td>Clear all Condition Codes</td>
</tr>
<tr>
<td>000277</td>
<td>SCC</td>
<td>Set all Condition Codes.</td>
</tr>
<tr>
<td>000240</td>
<td>NOP</td>
<td>No-operation</td>
</tr>
</tbody>
</table>
### B.3.5 Branch Instructions

Op E where \[-128 \leq (E-\cdot-2)/2 < 127\]

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>MNEMONIC</th>
<th>Stands for</th>
<th>Condition to be met if branch is to occur</th>
</tr>
</thead>
<tbody>
<tr>
<td>0004XX</td>
<td>BR</td>
<td>BRanch always</td>
<td></td>
</tr>
<tr>
<td>0010XX</td>
<td>BNE</td>
<td>Branch if Not Equal to Zero</td>
<td>Z=0</td>
</tr>
<tr>
<td>0014XX</td>
<td>BEQ</td>
<td>Branch if Equal (to zero)</td>
<td>Z=1</td>
</tr>
<tr>
<td>0020XX</td>
<td>BGE</td>
<td>Branch if Greater than or equal(to zero)</td>
<td>N(V)</td>
</tr>
<tr>
<td>0024XX</td>
<td>BLT</td>
<td>Branch if Less Than (zero)</td>
<td>N(V)</td>
</tr>
<tr>
<td>0030XX</td>
<td>BGT</td>
<td>Branch if Greater Than (zero )</td>
<td>Z!(N(V))=0</td>
</tr>
<tr>
<td>0034XX</td>
<td>BLE</td>
<td>Branch if Less than or Equal (to zero)</td>
<td>Z!(N(V))=1</td>
</tr>
<tr>
<td>1000XX</td>
<td>BPL</td>
<td>Branch if PLUS</td>
<td>N=0</td>
</tr>
<tr>
<td>1004XX</td>
<td>BMI</td>
<td>Branch if MINus</td>
<td>N=1</td>
</tr>
<tr>
<td>1010XX</td>
<td>BHI</td>
<td>Branch if HIgher</td>
<td>C(Z)</td>
</tr>
<tr>
<td>1014XX</td>
<td>BLOS</td>
<td>Branch if LOwer or Same</td>
<td>C!Z=0</td>
</tr>
<tr>
<td>1020XX</td>
<td>BVC</td>
<td>Branch if oVerflow Clear</td>
<td>V=0</td>
</tr>
<tr>
<td>1024XX</td>
<td>BVS</td>
<td>Branch if oVerflow Set</td>
<td>V=1</td>
</tr>
<tr>
<td>1030XX</td>
<td>BCC (or BHIS)</td>
<td>Branch if Carry Clear</td>
<td>C=0</td>
</tr>
<tr>
<td>1034XX</td>
<td>BCS (or BLO)</td>
<td>Branch if Carry Set (or Branch if LOw)</td>
<td>C=1</td>
</tr>
</tbody>
</table>

### B.3.6 Subroutine Call JSR ER,A

<table>
<thead>
<tr>
<th>Op-code</th>
<th>MNEMONIC</th>
<th>Stands for</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>004RDD</td>
<td>JSR</td>
<td>Jump to Sub-Routine</td>
<td>Push register on the SP stack, put the PC in the register:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DE\rightarrow TEMP -a temporary storage register internal to processor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(SP)-2 \rightarrow SP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(REG)\rightarrow (SP)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(PC)+m \rightarrow REG -m depends upon the address mode,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(TEMP)\rightarrow PC</td>
</tr>
</tbody>
</table>

B-7
### B.3.7 Subroutine Return

<table>
<thead>
<tr>
<th>Op-code</th>
<th>MNEMONIC</th>
<th>Stands for</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00020R</td>
<td>RTS</td>
<td>ReTurn from Subroutine</td>
<td>Put register contents in PC and pop old contents from SP stack into register.</td>
</tr>
</tbody>
</table>

### B.4 ASSEMBLER DIRECTIVES

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>Operand</th>
<th>Stands for</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.EOT</td>
<td>none</td>
<td>End Of Tape</td>
<td>Indicates the physical end of the source input medium</td>
</tr>
<tr>
<td>.EVEN</td>
<td>none</td>
<td>EVEN</td>
<td>Insures that the assembly location counter is even by adding 1 if it is odd.</td>
</tr>
<tr>
<td>.END</td>
<td>E</td>
<td>END</td>
<td>Indicates the physical and logical end of the program and optionally specifies the entry point (E)</td>
</tr>
</tbody>
</table>

<p>| WORD     | E, E,... | WORD                         | Generates words of data                                                  |
| .BYTE    | E, E,... | BYTE                         | Generates bytes of data                                                  |
| .ASCII   | /xxx...x/ | ASCII                        | Generates 7-bit ASCII characters for text enclosed by delimiters.         |
| .TITLE   | NAME    | TITLE                        | Generates a name for the object module.                                  |
| .ASECTION| none    | ASECTION                     | Initiates the Absolute section.                                          |
| .CSECTION| none    | CSECTION                     | Initiates the Relocatable Control section.                               |
| .LIMIT   | none    | LIMIT                        | Generates two words containing the low and high limits of the relocatable section. |
| .GLOBAL  | NAME, NAME,... | GLOBAL                    | Specifies each name to be a global symbol                               |
| .RAD50   | /XXX/   | RADIX 50                     | Generates the RADIX 50 representation of the ASCII character in delimiters. |</p>
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand</th>
<th>Stands For</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.IFZ</td>
<td>E</td>
<td>IF E=0</td>
<td>Assemble what follows up to the terminating .ENDC if the expression E is 0.</td>
</tr>
<tr>
<td>.IFNZ</td>
<td>E</td>
<td>IF E≠0</td>
<td>Assemble what follows up to the terminating .ENDC, if the expression E is not 0.</td>
</tr>
<tr>
<td>.IFL</td>
<td>E</td>
<td>IF E&lt;0</td>
<td>Assemble what follows up to the terminating .ENDC, if the expression E is less than 0.</td>
</tr>
<tr>
<td>.IFLE</td>
<td>E</td>
<td>IF E≤0</td>
<td>Assemble what follows up to the terminating .ENDC, if the expression E is less than or equal to 0.</td>
</tr>
<tr>
<td>.IFG</td>
<td>E</td>
<td>IF E&gt;0</td>
<td>Assemble what follows up to the terminating .ENDC, if the expression E is greater than 0.</td>
</tr>
<tr>
<td>.IFGE</td>
<td>E</td>
<td>IF E≥0</td>
<td>Assemble what follows up to the terminating .ENDC, if the expression E is greater than or equal to 0.</td>
</tr>
<tr>
<td>.IFDF</td>
<td>NAME</td>
<td>IF NAME defined</td>
<td>Assemble what follows up to the terminating .ENDC if the symbol NAME is defined.</td>
</tr>
<tr>
<td>.IFNDF</td>
<td>NAME</td>
<td>IF NAME undefined</td>
<td>Assemble what follows up to the terminating .ENDC if the symbol NAME is undefined.</td>
</tr>
<tr>
<td>.ENDC</td>
<td>none</td>
<td>End of Conditional</td>
<td>Terminates the range of a conditional directive.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Meaning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Addressing error. An address within the instruction is incorrect. Also includes relocation errors.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Bounding error. Instructions or word data are being assembled at an odd address in memory.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Doubly-defined symbol referenced. Reference was made to a symbol which is defined more than once.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>Illegal character detected. Illegal characters which are also non-printing are replaced by a ? on the listing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>Line buffer overflow. All extra characters beyond 72 are ignored.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Number containing an 8 or 9 was not terminated by a decimal point.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>Phase error. A label's definition or value varies from one pass to another.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>Questionable syntax. There are missing arguments or the instruction scan was not completed, or a carriage return was not followed by a linefeed or form feed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Register-type error. An invalid use of or reference to a register has been made.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Symbol table overflow. When the quantity of user-defined symbols exceeds the allocated space available in the user's symbol table, the assembler outputs the current source line with the S error code, then returns to the command string interpreter to await the next command string to be typed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Truncation error. More than the allotted number of bits were input so the leftmost bits were truncated. T error does not occur for the result of an expression.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>Undefined symbol. An undefined symbol was encountered during the evaluation of an expression. Relative to the expression, the undefined symbol is assigned a value of zero.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B.6 INITIAL OPERATING PROCEDURES

Loading: Use Absolute Loader. The start address of the Loader must be in the console switches.

Storage Requirements: PAL-11$ uses 8K of memory.

Starting: Immediately upon loading, PAL-11$ will be in control and initiate dialogue.

Initial Dialogue:

<table>
<thead>
<tr>
<th>Printout</th>
<th>Inquiry</th>
</tr>
</thead>
<tbody>
<tr>
<td>*S</td>
<td>What is the input device of the <em>Source</em> symbolic tape?</td>
</tr>
<tr>
<td>*B</td>
<td>What is the output device of the <em>Binary</em> object tape?</td>
</tr>
<tr>
<td>*L</td>
<td>What is the output device of the assembly Listing?</td>
</tr>
<tr>
<td>*T</td>
<td>What is the output device of the symbol Table?</td>
</tr>
</tbody>
</table>

Each of these questions may be answered by any one of the following characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Answer Indicated</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>Teleprinter keyboard</td>
</tr>
<tr>
<td>L</td>
<td>Low-speed reader or punch</td>
</tr>
<tr>
<td>H</td>
<td>High-speed reader or punch</td>
</tr>
<tr>
<td>P</td>
<td>Line Printer</td>
</tr>
</tbody>
</table>

Each of these answers may be followed by the other characters indicating options:

<table>
<thead>
<tr>
<th>Option Typed</th>
<th>Function to be performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>/1</td>
<td>on pass 1</td>
</tr>
<tr>
<td>/2</td>
<td>on pass 2</td>
</tr>
<tr>
<td>/3</td>
<td>on pass 3</td>
</tr>
<tr>
<td>/E</td>
<td>errors to be listed on the Teletype on the same pass (meaningful only for *B or *L).</td>
</tr>
</tbody>
</table>

Each answer is terminated by typing the RETURN key. A RETURN alone as answer will delete the function.
## Dialogue During Assembly:

<table>
<thead>
<tr>
<th>Printout</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOF ?</td>
<td>Place next tape in reader and type RETURN. A .END statement may be forced by typing E followed by RETURN.</td>
</tr>
<tr>
<td>END ?</td>
<td>Start next pass by placing first tape in reader and typing RETURN.</td>
</tr>
<tr>
<td>EOM ?</td>
<td>If the end-of-medium is on the listing device, the device may be readied and the assembly may be continued by typing RETURN. If the end-of-medium is on the binary device, the assembler will discontinue the assembly and restart itself.</td>
</tr>
</tbody>
</table>

### Restarting:

Type CTRL/P. The initial dialogue will be started again.
APPENDIX C

ASSEMBLING AND LINKING PAL-11S

PAL-11S consists of two independent programs. The first program is a memory clear program. The second is the assembler. All programs are available as ASCII source tapes, object modules and as a load module.

The memory clear program, MEMCLR, consists of one ASCII tape. This program should never need to be assembled. The object module may be used when constructing a new load module of PAL-11S.

The assembler consists of three program modules which are assembled separately and then linked together. The first is the main program called PAL-11S. It consists of 13 ASCII tapes. The second module is the symbol table, PALSYM, which consists of 2 ASCII tapes. The third is IOXLPT consisting of 2 ASCII tapes.

If changes are made in any of these modules, that module must be assembled by PAL-11S and the new object module can be linked with the other object modules. It should be noted that assembly of these programs will result in:

<table>
<thead>
<tr>
<th>Program</th>
<th>Pages of Listing (Decimal)</th>
<th>Number of Symbols (Decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL-11S</td>
<td>160</td>
<td>756</td>
</tr>
<tr>
<td>PALSYM</td>
<td>11</td>
<td>32</td>
</tr>
<tr>
<td>IOXLPT</td>
<td>29</td>
<td>191</td>
</tr>
</tbody>
</table>

Also note that there will be two undefined symbols listed at the end of pass 1. These are forward references on direct assignments which get defined properly in pass 2.

The final load module is constructed by LINK-11S. First the memory clear program object module is processed by the linker and the resulting load module is left in the punch while the PAL-11S, PALSYM and IOXLPT object modules are linked to create a second load module. The resulting tape contains two load modules. The first clears memory and then jumps to the absolute loader to load the second.

Do not re-link PAL-11S to run above 16K. The size of the symbol table is fixed, and there is no need to re-link at a higher address even on large systems.
The supplied tapes are identified as follows:

<table>
<thead>
<tr>
<th>Library Code</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC-11-ASQA-PA</td>
<td>RELMEM (Memory Clear Program)</td>
</tr>
<tr>
<td></td>
<td>One Assembly</td>
</tr>
<tr>
<td>Tape 1 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 2 of 18</td>
<td>PAL-11S (Main Program)</td>
</tr>
<tr>
<td>Tape 3 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 4 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 5 of 19</td>
<td></td>
</tr>
<tr>
<td>Tape 6 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 7 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 8 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 9 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 10 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 11 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 12 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 13 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 14 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 15 of 18</td>
<td>PALSYM (Symbol Table)</td>
</tr>
<tr>
<td>Tape 16 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 17 of 18</td>
<td>IOXLPT</td>
</tr>
<tr>
<td>Tape 18 of 18</td>
<td></td>
</tr>
<tr>
<td>Tape 1 of 4</td>
<td>RELMEM Object Module</td>
</tr>
<tr>
<td>Tape 2 of 4</td>
<td>PAL-11S Object Module</td>
</tr>
<tr>
<td>Tape 3 of 4</td>
<td>PALSYM Object Module</td>
</tr>
<tr>
<td>Tape 4 of 4</td>
<td>IOXLPT Object Module</td>
</tr>
<tr>
<td>DEC-11-ASQA-PL</td>
<td>PAL-11S Load Module*</td>
</tr>
</tbody>
</table>

*This tape is the concatenation of a link of the RELMEM object module followed by a link of the PAL-11S, PALSYM, and IOXLPT object modules.
INDEX

Absolute
expression, 1-10
loader, 1-26
mode, 1-16
program, 2-2
Addition, 1-9
Addresses, 2-2
Addressing, 1-13
Address mode syntax, 1-40
Apostrophe ( ' ) usage, 1-9, 1-12
Argument separators, 1-2
Arithmetic operators, 1-8, 1-9
ASCII
character set, Appendix A
conversion, 1-22
.ASCII directive, 1-22
.ASECT directive, 1-11, 1-20,
1-33, 2-2
Assembler directives, 1-3, 1-19,
B-8
Assembling and linking PAL-11A,
C-1
Assembly dialogue, 1-31
Assembly listing, 1-25, 1-32
apostrophe usage, 1-12
Asterisk ( * ) usage, 1-28
Autodecrement mode, 1-15
Autoincrement mode, 1-14
Binary output, 1-29
Branch instructions, 1-19, B-7
.BYTE directive, 1-8, 1-22, 1-33
Carriage return, 1-2, 1-3
Character set, 1-2
Checksum, 2-7
CLR instruction, 1-13
Colon ( : ) usage, 1-3, 1-23
Comma ( , ) usage, 1-4, 1-21
Comments, 1-4
field, 1-3
Communications directory (COMD),
2-3
Condition codes, B-3, B-6
Conditional directives, 1-24
Conversion, ASCII, 1-9
.CSECT directive, 1-11, 1-20,
1-33, 2-2
CTRL/A, 1-29
CTRL/P, 1-29, 1-31
CTRL/U, 1-26
CTRL/FORM key, 1-5
Current location counter, 1-3
Data sharing, 2-2
Decimal numbers, 1-8
Deferred autodecrement mode, 1-15
Deferred autoincrement mode, 1-14
Deferred immediate mode, 1-16
Deferred register mode, 1-14
Deferred relative mode, 1-16
Device specification, 1-26
Dialogue, initial, 1-30, 1-31
Direct assignment statement, 1-6
Directives, Assembler, 1-19, 1-46
.ASCII, 1-23
.ASECT, 1-11, 1-20, 1-33, 2-2
.BYTE, 1-8, 1-22, 1-33
conditional, 1-24
.CSECT, 1-11, 1-20, 1-33, 2-2
.END, 1-21
.ENDC, 1-25
.EOT, 1-21
.EVEN, 1-21
.GLOBL, 1-20
.LIMIT, 1-24
.RAD50, 1-23
.TITLE, 1-19
.WORD, 1-21
Double operand instructions, B-4
Editor, 1-2
EMT instructions, 1-19
.END?, 1-32
.END directive, 1-21
.ENDC directive, 1-25
End of tape, 1-21
Entry point (entry symbol), 1-6
EOF?, 1-31
EOM?, 1-31
.EOT directive, 1-21
Equal sign ( = ) usage, 1-23
Error codes, 1-34, B-10
Error
expression, 1-25
phase, 1-7
software, 1-35
syntax, 1-25
truncation, 1-8
typing, 1-26
Errors, Linker, 2-7, 2-8, 2-9
.EVEN directive, 1-21
Expressions, 1-8
missing, 1-8
mode, 1-10
null, 1-25
External expression, 1-8, 1-10
External symbol missing, 1-8
Features, PAL-11S, 1-1
Fields, 1-2, 1-3
Format, 1-4, 1-5
Form feed, 1-2, 1-3, 1-5
Forward references, 1-6, 1-11

General registers, 1-7
.GLOBL directive, 1-20
Global symbol directory (GSD), 1-33, 2-3
Global symbols, 1-6, 2-2

Hardware requirements, 1-2

Immediate mode, 1-16
Inclusive OR operation, 1-9
Indexing, 1-13
Index mode, 1-15
Initial dialogues, 1-26, 1-30, 1-31
Instruction mnemonic, 1-3
Internal symbols, 1-6

JMP instruction, 1-14
JRS instruction, 1-14

Label fields, 1-3
.LIMIT directive, 1-24
Line feed, 1-2, 1-3
Line printer, 1-31
Line terminators, 1-2
Linker operation instructions, 2-5
Linker operational cautions, 2-7
Linking, 1-12
Loading PAL-11S, 1-26
Load map, 2-4
Load module, 1-1, 1-21
Location counter, 1-11
Logical AND operation, 1-9
Logical inclusive OR operation, 1-9
Logical operators, 1-8, 1-9, B-4
Low-speed punch, 1-32

Memory references, 1-16
Missing term, expression, or external symbol, 1-8
MOD40, 1-23

Mode, 1-11
address, 1-13 through 1-16
expression, 1-10
forms and codes, 1-17
of operand, 1-16
MOV instruction, 1-13
Multiple definition of symbol (M), 1-3
Multiple operands, 1-4
Multiple statement labels, 1-3

Null expression, 1-25
Numbers, 1-8

Object modules, 1-1, 1-25, 2-3
output, 1-33
Octal numbers, 1-8
Offset, 1-19
Op-code, 1-5
Operands, 1-4, 1-41
fields, 1-3, 1-18
mode, 1-16
Operating procedures
Assembler, 1-25, 1-49
Linker, 2-5
Operational cautions, Linker, 2-7
Operation instructions,
Assembler, B-5
Linker, 2-5
Operators, 1-8, 1-9
fields, 1-3
Output, object module, 1-33

Page size, 1-5
PAL-11A, assembling and linking, C
PAL-11R object modules, 2-2
PAL-11S features, 1-1
PASS 1, 1-29, 2-6
PASS 2, 1-30, 2-6
PASS 3, 1-30
PC, 1-11
Percent sign ( % ) usage, 1-7
Period ( . ) usage, 1-11
Phase errors, 1-7
Positive numbers, 1-8
Program counter, 1-11, 1-13
Program sharing, 2-2
Pseudo-ops, 1-19
Quotation mark usage, 1-9

.RAD50 directive, 1-23
RADIX 50 packing algorithm, 1-24
Register mode, 1-14
Register symbols, 1-7
Relative mode, 1-16
Relocatable expression, 1-10
Relocatable program, 2-2
Relocation, 1-12
directory, 1-33
Restarting Linker, 2-7
RETURN key, 1-2, 1-26
RUBOUT key, 1-26

Semicolon (;) usage, 1-4
Single operand instructions, 1-42
Slash (/) usage, 1-23
Software, Linker, 2-11
Source program, 1-2
Source tapes, Linker, 2-11
Space character, 1-4
Statement
direct assignment, 1-6
labels, 1-3
terminator, 1-2
Storage area, 1-12
Symbols, 1-5, 1-6, 1-8
user defined, 1-3
Symbol table, 1-5, 1-28
Subroutine calls, B-7
Subtraction, 1-9

Tab character, 1-4
Table of mode forms and codes, 1-17
Terminators
assembly, B-1
directive, 1-23
of operator, 1-3
Term missing, 1-8
Text block, 1-33
Text Editor, 1-1
.TITLE directive, 1-19
Trap instructions, 1-19
Trap vectors, 2-9
Truncation (T) error, 1-8
Typing error, 1-26

User-defined symbol, 1-3, 1-25

.WORD directive, 1-21, 1-22

X-3
Announcements for new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters:

- Digital Software News for the PDP-8 & PDP-12
- Digital Software News for the PDP-11
- Digital Software News for the PDP-9/15 Family

These newsletters contain information applicable to software available from Digital's Program Library. Articles in Digital Software News update the cumulative Software Performance Summary which is contained in each basic kit of system software for new computers. To assure that the monthly Digital Software News is sent to the appropriate software contact at your installation, please check with the Software Specialist or Sales Engineer at your nearest Digital office.

Questions or problems concerning Digital's Software should be reported to the Software Specialist. In cases where no Software Specialist is available, please send a Software Performance Report form with details of the problem to:

Software Information Service
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

These forms which are provided in the software kit should be fully filled out and accompanied by teletype output as well as listings or tapes of the user program to facilitate a complete investigation. An answer will be sent to the individual and appropriate topics of general interest will be printed in the newsletter.

Orders for new and revised software and manuals, additional Software Performance Report forms, and software price lists should be directed to the nearest Digital Field office or representative. U.S.A. customers may order directly from the Program Library in Maynard. When ordering, include the code number and a brief description of the software requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information please write to:

DECUS
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754
NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Please indicate the type of user/reader that you most nearly represent.

☐ Assembly language programmer
☐ Higher-level language programmer
☐ Occasional programmer (experienced)
☐ User with little programming experience
☐Student programmer
☐ Non-programmer interested in computer concepts and capabilities

Name ___________________________ Date ____________________
Organization ___________________________
Street ___________________________
City ___________________________ State _________ Zip Code ____________
or Country

If you do not require a written reply, please check here. ☐
PRODUCT CODE: DEC-11-YRWB-DN


DATE CREATED: November, 1972

MAINTAINER: Development
Your attention is invited to the last two pages of this document. The "How to Obtain Software Information" page tells you how to keep up-to-date with DEC's software. The "Reader's Comments" page, when filled in and mailed, is beneficial to both you and DEC; all comments received are acknowledged and are considered when documenting subsequent documents.

Copyright © 1971, 1972 by Digital Equipment Corporation

Technical changes to this manual are indicated by a bar in the margin.

This document is for information purposes and is subject to change without notice.

Associated documents:

  DEC-11-GGPC-D

  DEC-11-ASDC-D

The following are trademarks of Digital Equipment Corporation:

- DEC
- Digital (logo)
- DECTape
- Unibus
- PDP-11
- Comtex-11
- RSTS-11
- RSX-11
The attached pages should be inserted in the PAL-11S Assembler and LINK-11S LINKER Programmer's Manual. Technical changes have been marked with a bar in the page margin.
CONTENTS

CHAPTER I  PAL-11S ASSEMBLER

1. Character Set
2. Statements
3. Symbols
4. Expressions
5. Assembly Location Counter
6. Relocation and Linking
7. Addressing
8. Assembler Directives
9. Operating Procedures
10. Error Codes
11. Software Error Halts

CHAPTER II LINK-11S LINKER

1. Introduction
2. Input and Output
3. Operating Procedures
4. Preparation

APPENDICES

A  ASCII Character Set
B  PAL-11S Assembly Language and Assembler
C  Assembling and Linking PAL-11S
D  Note to Users of Serial LA30 and 600, 1200 and 2400 Baud VT05's

INDEX
as for *L question. The symbol table will be output at the end of the first pass unless /2 or /3 is typed in response to *T. The first tape to be assembled should be placed in the reader before typing the RETURN key because assembly will begin upon typing RETURN to the *T question. The /E option is not a meaningful response to *T. Example

<table>
<thead>
<tr>
<th>*T</th>
<th>T/3</th>
<th>Symbol table output on teleprinter at end of third pass.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*T</td>
<td></td>
<td>Typing the RETURN key alone will cause the Assembler to omit symbol table output.</td>
</tr>
</tbody>
</table>

The symbol table is printed alphabetically, three symbols per line. Each symbol printed is followed by its identifying characters and by its value. If the symbol is undefined, six asterisks replace its value. The identifying characters indicate the class of the symbol; that is, whether it is a label, direct assignment, register symbol, etc. The following examples show the various forms.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDEF</td>
<td>001244</td>
<td>(Defined Label)</td>
</tr>
<tr>
<td>R3</td>
<td>%000003</td>
<td>(Register Symbol)</td>
</tr>
<tr>
<td>DIRASM</td>
<td>177777</td>
<td>(Direct Assignment)</td>
</tr>
<tr>
<td>XYZ</td>
<td>*****</td>
<td>(Undefined direct assignment)</td>
</tr>
<tr>
<td>R6</td>
<td>%**</td>
<td>(Undefined register symbol)</td>
</tr>
<tr>
<td>LABEL</td>
<td>*****</td>
<td>(Undefined label)</td>
</tr>
</tbody>
</table>

Generally, undefined symbols and external symbols will be listed as undefined direct assignments. Multiply-defined symbols are not flagged in the symbol table printout but are flagged wherever they are used in the program.

If the symbol is relocatable or global or both, the symbol's value will be followed by an R, a G or both.

It is possible to output both the binary tape and the assembly listing on the same pass, thereby reducing the assembly process to two passes (see Example 1 below). This will happen automatically unless the binary device and the listing device are conflicting devices or the same device (see Example 2 below). The only conflicting devices are the teleprinter and the low-speed punch. Even though the Assembler deduces that three passes are necessary, the binary and listing can be forced on pass 2 by including /2 in the responses to *B and *L (see Example 3 below).

Example 1. Runs 2 passes:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*S</td>
<td>H</td>
<td>High-speed reader</td>
</tr>
<tr>
<td>*B</td>
<td>H</td>
<td>High-speed punch</td>
</tr>
<tr>
<td>*L</td>
<td>P</td>
<td>Line Printer</td>
</tr>
<tr>
<td>*T</td>
<td>T</td>
<td>Teleprinter</td>
</tr>
</tbody>
</table>
4.0 PREPARATION

LINK-11S is available as an absolute load module (for an 8K machine), as two object modules (for relinking) and as several ASCII source tapes. There is one object module for the Linker and one for IOXLPT. The supplied object modules may be relinked (using the supplied load module) to load into any size machine larger than 8K. However, the resulting Linker will still assume a top of memory corresponding to an 8K machine (this can be overridden in the command string options). The assumed top of memory and reserved Absolute Loader space may be changed by editing the first linker ASCII tape with ED-11. The parameters to be changed are HGHMEM (high memory address +1 (always even)) and ALODSZ (Absolute Loader size (always even)). The source tapes for the Linker may then be assembled with PAL-11S and the new object module can then replace the supplied Linker object module.

The tapes are identified as follows:

<table>
<thead>
<tr>
<th>Library Code</th>
<th>Tape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC-11-ULKSA-A-PA1</td>
<td>Tape 1 of 6</td>
<td>LINK-11S (Main Program)</td>
</tr>
<tr>
<td>DEC-11-ULKSA-A-PA2</td>
<td>Tape 2 of 6</td>
<td>One Assembly</td>
</tr>
<tr>
<td>DEC-11-ULKSA-A-PA3</td>
<td>Tape 3 of 6</td>
<td></td>
</tr>
<tr>
<td>DEC-11-ULKSA-A-PA4</td>
<td>Tape 4 of 6</td>
<td></td>
</tr>
<tr>
<td>DEC-11-ULKSA-A-PA5</td>
<td>Tape 5 of 6</td>
<td>One IOXLPT</td>
</tr>
<tr>
<td>DEC-11-ULKSA-A-PA6</td>
<td>Tape 6 of 6</td>
<td>Assembly</td>
</tr>
<tr>
<td>DEC-11-ULKSA-A-PR1</td>
<td>Tape 1 of 2</td>
<td>LINK-11S Object Module</td>
</tr>
<tr>
<td>DEC-11-ULKSA-A-PR2</td>
<td>Tape 2 of 2</td>
<td>IOXLPT Object Module</td>
</tr>
<tr>
<td>DEC-11-ULKSA-A-PL</td>
<td>Tape 2 of 2</td>
<td>LINK-11S Load Module</td>
</tr>
</tbody>
</table>
APPENDIX C

ASSEMBLING AND LINKING PAL-11S

PAL-11S consists of two independent programs. The first program is a memory clear program. The second is the assembler. All programs are available as ASCII source tapes, object modules and as a load module.

The memory clear program, MEMCLR, (DEC-11-UPLSA-A-PA1) consists of one ASCII tape. This program should never need to be assembled. The object module may be used when constructing a new load module of PAL-11S.

The assembler consists of three program modules which are assembled separately and then linked together. The first is the main program called PAL-11S. It consists of 13 ASCII tapes (DEC-UPLSA-A-PA2-PA14). The second module is the symbol table, PALSYM, which consists of 2 ASCII tapes (DEC-11-UPLSA-A-PA15-PA16). The third is IOXLPT consisting of 2 ASCII tapes (DEC-11-UPLSA-A-PA17-PA18). Also included is PALSYM, specially created for 12K and 16K, consisting of one tape each (DEC-11-UPLSA-A-PA19-PA20).

If changes are made in any of these modules, that module must be assembled by PAL-11S (V003A) and the new object module can be linked with the other object modules. It should be noted that assembly of these programs will result in:

<table>
<thead>
<tr>
<th>Program</th>
<th>Pages of Listing (Decimal)</th>
<th>Number of Symbols (Decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL-11S</td>
<td>160</td>
<td>756</td>
</tr>
<tr>
<td>PALSYM</td>
<td>11</td>
<td>32</td>
</tr>
<tr>
<td>IOXLPT</td>
<td>29</td>
<td>191</td>
</tr>
</tbody>
</table>

Also note that there will be two undefined symbols listed at the end of pass 1. These are forward references on direct assignments which get defined properly in pass 2.

An example of the PAL-11S assembly follows:

```
PAL-11S V003A
*S H
*B H
*L P
*T P/2
END ?
00000000' ERRORS

PAL-11S V003A
*S H
*B H
*L P
*T P/2               (first pass on PA1)
EOF ?               (2nd pass on PA1)
EOF ?               (End of Tape #1 assembly)
EOF ?               (Remove tape from punch)
EOF ?               (Insert PA2 for 1st pass)
EOF ?               (End of PA2, insert PA3)
EOF ?               (End of PA3, insert PA4)
EOF ?               (End of PA4, insert PA5)
EOF ?               (End of PA5, insert PA6)
EOF ?               (End of PA6, insert PA7)
EOF ?               (End of PA7, insert PA8)
EOF ?               (End of PA8, insert PA9)
```
EOF ? (End of PA9, insert PA10)  
EOF ? (End of PA10, insert PA11)  
EOF ? (End of PA11, insert PA12)  
EOF ? (End of PA12, insert PA13)  
EOF ? (End of PA13, insert PA14)  

BINCNT = ***** SIMBC = *****  

END ? (End of PA14 and 1st pass)  
EOF ? (Insert PA2 for 2nd pass)  
EOF ? (End of PA2, insert PA3)  
EOF ? (End of PA3, insert PA4)  
EOF ? (End of PA4, insert PA5)  
EOF ? (End of PA5, insert PA6)  
EOF ? (End of PA6, insert PA7)  
EOF ? (End of PA7, insert PA8)  
EOF ? (End of PA8, insert PA9)  
EOF ? (End of PA9, insert PA10)  
EOF ? (End of PA10, insert PA11)  
EOF ? (End of PA11, insert PA12)  
EOF ? (End of PA12, insert PA13)  
EOF ? (End of PA13, insert PA14)  
EOF ? (End of PA14 and 2nd pass)  

Remove tape from punch

PAL-11S VØ3A
*S H  
*B H  
*L P  
*T P/2  
EOF ? (1st pass on PA15)  
END ? (End of PA15, insert PA16)  
EOF ? (End of PA16, insert PA15 for 2nd pass)  
EOF ? (End of PA15, insert PA16)  
EOF ? (End of 2nd pass)  

Remove tape from punch

PAL-11S VØ3A
*S H  
*B H  
*L P  
*T P/2  
EOF ? (1st pass on PA17)  
END ? (End of PA17, insert PA18)  
EOF ? (End of PA18, insert PA17 for 2nd pass)  
EOF ? (End of PA17, insert PA18)  
EOF ? (End of 2nd pass)  

Remove tape from punch

PAL-11S VØ3A
*S H  
*B H  
*L P  
*T P/2  
END ? (Pass 1 on PA20)  
EOF ? (Pass 2 on PA20)  
EOF ? (End of pass 2)  

Remove tape from punch

C-2
The final load module is constructed by LINK-11S. First the memory clear program object module is processed by the linker and the resulting load module is left in the punch while the PAL-11S, PALSYM and IOXLPT object modules are linked to create a second load module. The resulting tape contains two load modules. The first clears memory and then jumps to the absolute loader to load the second.

In order to take advantage of core sizes larger than 8K, PALSYM, the symbol table, specially created for 12K core and 16K core, and the object modules are included with the assembler. To link for 12K (or 16K), simply substitute the appropriate object tape for PALSYM (use DEC-11-UPLSA-A-PR5 for 12K or DEC-11-UPLSA-A-PR6 for 16K) specify a top address to LINK-11S of 57460 for 12K (77460 for 16K) and link as described in the preceding paragraph.

Do not re-link PAL-11S to run above 16K. The size of the symbol table is fixed, and there is no need to re-link at a higher address even on large systems.

The supplied tapes are identified as follows:

<table>
<thead>
<tr>
<th>Library Code</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC-11-UPLSA-A-PA1</td>
<td>Tape 1 of 20 One Assembly RELMEM (Memory Clear Program)</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA2</td>
<td>Tape 2 of 20 One Assembly PAL-11S (Main Program)</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA3</td>
<td>Tape 3 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA4</td>
<td>Tape 4 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA5</td>
<td>Tape 5 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA6</td>
<td>Tape 6 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA7</td>
<td>Tape 7 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA8</td>
<td>Tape 8 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA9</td>
<td>Tape 9 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA10</td>
<td>Tape 10 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA11</td>
<td>Tape 11 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA12</td>
<td>Tape 12 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA13</td>
<td>Tape 13 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA14</td>
<td>Tape 14 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA15</td>
<td>Tape 15 of 20 One Assembly PALSYM (Symbol Table) for 8K</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA16</td>
<td>Tape 16 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA17</td>
<td>Tape 17 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA18</td>
<td>Tape 18 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA19</td>
<td>Tape 19 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PA20</td>
<td>Tape 20 of 20 One Assembly</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PR1</td>
<td>Tape 1 of 6 RELMEM Object Module</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PR2</td>
<td>Tape 2 of 6 PAL-11S Object Module</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PR3</td>
<td>Tape 3 of 6 PALSYM Object Module for 8K</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PR4</td>
<td>Tape 4 of 6 IOXLPT Object Module</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PR5</td>
<td>Tape 5 of 6 PALSYM Object Module for 12K assembler</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PR6</td>
<td>Tape 6 of 6 PALSYM Object Module for 16K Assembler</td>
</tr>
<tr>
<td>DEC-11-UPLSA-A-PL</td>
<td>Tape 6 of 6 PAL-11S Load Module*</td>
</tr>
</tbody>
</table>

*This tape is the concatenation of a link of the RELMEM object module followed by a link of the PAL-11S, PALSYM for 8K, and IOXLPT object modules.
NOTE TO USERS OF SERIAL LA3Ø AND 600, 1200 AND 2400 BAUD VT05'S

The serial LA3Ø requires that filler characters follow each carriage return; the 600, 1200 and 2400 baud VT05's require that filler characters follow each line feed. The following table lists the filler characters needed. The byte at location 44₈ has been established as the filler count and the byte at location 45₈ contains the character to be filled. These locations are initially set to zero by LINK-11S and PAL-11S to allow normal operation of the program.

Depending on the terminal, change the locations as follows:

<table>
<thead>
<tr>
<th>Terminals</th>
<th>LOC 44</th>
<th>LOC 45</th>
<th>Resulting Word (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA3Ø</td>
<td>01₁₈</td>
<td>01₅₈</td>
<td>0000110100001001</td>
</tr>
<tr>
<td>VT05 600 Baud</td>
<td>00₁₀₈</td>
<td>01₂₈</td>
<td>0000101000000001</td>
</tr>
<tr>
<td>VT05 1200 Baud</td>
<td>00₂₀₈</td>
<td>01₂₈</td>
<td>000010100000000100</td>
</tr>
<tr>
<td>VT05 2400 Baud</td>
<td>00₄₀₈</td>
<td>01₂₈</td>
<td>0000101000000001000</td>
</tr>
</tbody>
</table>

The proper binary word can be stored at location 44₈ by using the console switches as described in section 2.1.2 of the Papertape Software Programming Handbook (DEC-11-XPTSA-A-D).

Furthermore, users with a 2400 baud VT05 should avoid the use of vertical tab characters in their programs. Vertical tabs will not be properly filled and may cause characters to be lost.

Once the changes have been made, the program may be dumped to paper tape by using the bootstrap version of DUMPAB (see instructions for use in section 6.3 of DEC-11-XPTSA-A-D).

The above changes only affect output to the console teleprinter.
Features, PAL-11S, 1-1
Fields, 1-2, 1-3
Format, 1-4, 1-5
Form feed, 1-2, 1-3, 1-5
Forward references, 1-6, 1-11
General registers, 1-7
.GLOBL directive, 1-20
Global symbol directory (GSD), 1-33, 2-3
Global symbols, 1-6, 2-2
Hardware requirements, 1-2
Immediate mode, 1-16
Inclusive OR operation, 1-9
Indexing, 1-13
Index mode, 1-15
Initial dialogues, 1-26, 1-30, 1-31
Instruction mnemonic, 1-3
Internal symbols, 1-6
JMP instruction, 1-14
JRS instruction, 1-14
LA30 Users, D-1
Label fields, 1-3
.LIMIT directive, 1-24
Line feed, 1-2, 1-3
Line printer, 1-31
Line terminators, 1-2
Linker operation instructions, 2-5
Linker operational cautions, 2-7
Linking, 1-12
Loading PAL-11S, 1-26
Load map, 2-4
Load module, 1-1, 1-21
Location counter, 1-11
Logical AND operation, 1-9
Logical inclusive OR operation, 1-9
Logical operators, 1-8, 1-9, 8-4
Low-speed punch, 1-32
Memory references, 1-16
Missing term, expression, or external symbol, 1-8
MOD40, 1-23
Mode, 1-11
address, 1-13 through 1-16
expression, 1-10
forms and codes, 1-17
of operand, 1-16
MOV instruction, 1-13
Multiple definition of symbol (M), 1-3
Multiple operands, 1-4
Multiple statement labels, 1-3
Null expression, 1-25
Numbers, 1-8
Object Modules, 1-1, 1-25, 2-3
output, 1-33
Octal numbers, 1-8
Offset, 1-19
Op-code, 1-5
Operands, 1-4, 1-41
fields, 1-3, 1-18
mode, 1-16
Operating procedures
Assembler, 1-25, 1-49
Linker, 2-5
Operational cautions, Linker, 2-7
Operational instructions,
Assembler, B-5
Linker, 2-5
Operators, 1-8, 1-9
fields, 1-3
Output, object module, 1-33
Page size, 1-5
PAL-11A, assembling and linking, C-1
PAL-11R object modules, 2-2
PAL-11S features, 1-1
PASS 1, 1-29, 2-6
PASS 2, 1-30, 2-6
PASS 3, 1-30
PC, 1-11
Percent sign (%) usage, 1-7
Period (.) usage, 1-11
Phase errors, 1-7
Positive numbers, 1-8
Program counter, 1-11, 1-13
Program sharing, 2-2
Pseudo-ops, 1-19
Quotation mark usage, 1-9
.RAD50 directive, 1-23
RADIX 50 packing algorithm, 1-24
Register mode, 1-14
Register symbols, 1-7
Relative mode, 1-16
Relocatable expression, 1-10
Relocatable program, 2-2
Relocation, 1-12
directory, 1-33
Restarting Linker, 2-7
RETURN key, 1-2, 1-26
RUBOUT key, 1-26
Semicolon (;) usage, 1-4
Single operand instructions, 1-42
Slash (/) usage, 1-23
Software, Linker, 2-11
Source program, 1-2
Source tapes, Linker, 2-11
Space character, 1-4
Statement
direct assignment, 1-6
labels, 1-3
terminator, 1-2
Storage area, 1-12
Symbols, 1-5, 1-6, 1-8
user defined, 1-3
Symbol table, 1-5, 1-28
Subroutine calls, B-7
Subtraction, 1-9
Tab character, 1-4
Table of Mode forms and codes, 1-17
Terminators
assembly, B-1
directive, 1-23
of operator, 1-3
Term missing, 1-8
Text block, 1-33
Text editor, 1-1
.TITLE directive, 1-19
Trap instructions, 1-19
Trap vectors, 2-9
Truncation (T) error, 1-8
Typing error, 1-26
User-defined symbol, 1-3, 1-25
VT05 users, D-1
.WORD directive, 1-21, 1-22