DEC-11-ASDB-D

P A L - 1 1 R A S S E M B L E R

P R O G R A M M E R ' S M A N U A L

Program Assembly Language

and

Relocatable Assembler

for the

Disk Operating System

For additional copies, order No. DEC-11-ASDB-D, from Digital Equipment Corporation, Direct Mail, Bldg. 1-1, Maynard, Mass. 01754

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

First Printing, February 1971
Revised, May 1971

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; any comments received are acknowledged and are considered when documenting subsequent manuals.

Associated Documents:

PDP-11 Disk Operating System Monitor, Programmer's Handbook, DEC-11-MWDA-D

PDP-11 Edit-11 Text Editor, Programmer's Manual, DEC-11-EEDA-D

PDP-11 ODT-11R Debugging Program, Programmer's Manual, DEC-11-OODA-D

PDP-11 PIP, File Utility Package, Programmer's Manual, DEC-11-PIDA-D

PDP-11 Link-11 Linker and Libr-11 Librarian, Programmer's Manual, DEC-11-ZLDA-D

PDP-11 FORTRAN IV, Programmer's Manual, DEC-11-KFDA-D

P R E F A C E

This manual describes PAL-11R, the Assembly Language and Assembler for the PDP-11 Disk Operating System. Thus, familiarity with the DOS Monitor (see PDP-11 Disk Operating System Monitor, Programmer's Handbook, DEC-11-MWDA-D) is assumed.

The manual explains how to write PAL-11R source programs and how to assemble them into object modules. All facets of the assembly language are explained and illustrated with many examples, and the manual concludes with assembling procedures.

In explaining how to write PAL-11R source programs, it is necessary, especially at the outset, to make frequent forward references. Therefore, we recommend that you first read through the entire manual to get a "feel" for the language, and then reread the manual for a complete understanding of the language and assembling procedures.

In addition to the PAL-11R Assembler, the Disk Operating System software includes:

DOS Monitor
Edit-11 Text Editor
ODT-11R Debugging Program
PIP, File Utility Package
Link-11 Linker
Libr-11 Librarian
FORTRAN IV

C O N T E N T S

CHAPTER 1

INTRODUCTION

PAL-11R (Program Assembly Language for the PDP-11, Relocatable Version) operates under the Disk Operating System (DOS). PAL-11R 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 Edit-11 (see PDP-11 Edit-11 Text Editor, Programmer's Manual, DEC-11-EEDA-D for details), or off-line are then assembled into object modules which are processed by the PDP-11 Linker, Link-11. Link-11 produces a load module which is loaded by the monitor command RUN 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. A complete octal/symbolic listing of the assembled program may also be obtained. This listing is especially useful for documentation and debugging purposes.

Some notable features of PAL-11R are:

1. Selective assembly pass functions
2. Device and file name specification for pass functions
3. Error listing on command output device
4. Double buffered and concurrent I/O
5. Alphabetized, formatted symbol table listing
6. Relocatable object modules
7. Global symbols for linking between object modules
8. Conditional assembly directives
9. Program sectioning directives

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

CHAPTER 2

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 and 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 (Chapter 10), 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 Edit-11 Text Editor 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 in Appendix B. The four fields are:

Label Operator Operand Comment

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

2.1 CHARACTER SET

A PAL-11R source program is composed of symbols, numbers, expressions, symbolic instructions, assembler directives, argument separators, and line terminators written using the following ASCII[1] 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.)

[1]ASCII stands for American Standard Code for Information Interchange.

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

2.2 LABEL

A label is a user-defined symbol (Chapter 3) 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 plus 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 example if the location counter were relocatable, 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, 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.

2.3 OPERATOR

An operator follows the label field in a statement, and may be an instruction mnemonic or an assembler directive (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.

Consider the following examples:

```
MOV→|A,B        ; →| (TAB) terminates operator MOV
MOV@A,B         ; @ terminates operator MOV
```

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.

2.4 OPERAND

An operand is that part of a statement which is operated on by the operator -- an instruction mnemonic or assembler 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:

```
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.

2.5 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:

```
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.

2.6 FORMAT CONTROL

Formatting of the source program 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 terminated after 56 lines.

CHAPTER 3

SYMBOLS

There are two types of symbols: permanent and user-defined. There are two symbol tables: the Permanent Symbol Table (PST) and the User Symbol Table (UST). The PST contains all the permanent symbols and is part of the Assembler's load module. The UST is constructed as the source program is assembled; user-defined symbols are added to the table as they are encountered.

3.1 PERMANENT SYMBOLS

Permanent symbols consist of the instruction mnemonics (Appendix B.3) and assembler directives (Chapter 8). These symbols are a permanent part of the Assembler 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 (Section 2.2) or by direct assignment (Section 3.3). These symbols are added to the User 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; any other character is illegal and, if used, will result in the error message I (Chapter 10). (Again, $ and . are reserved for system software.) 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 absolute value is associated with the symbol.

User-defined symbols are either internal or global. All symbols are internal unless they are explicitly defined as global with the .GLOBL assembler directive (Section 8.2).

Global symbols are used to provide links between object modules. A global symbol which is defined (as a label or 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.

Since PAL-11R provides program sectioning capabilities (Section 8.3), two types of internal symbols must be considered:

1. a symbol that belongs to the current program section;
2. a symbol that belongs to any of the other program sections.

In both cases, the symbol must be defined within the current assembly; the significance of the distinction is critical in evaluating expressions involving type 2. above (Section 4.4).

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 (Chapter 4).

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 command output device as such at the end of that pass. X is undefined throughout pass 2, and will cause a U error message.

Examples:

```
        A = 1               ;THE SYMBOL A IS EQUATED WITH THE VALUE 1.
        B = 'A-1&MASKLOW    ;THE SYMBOL B IS EQUATED WITH THE EXPRES-
                            ;SION'S VALUE
C:      D = 3               ;THE SYMBOL D IS EQUATED WITH 3.  THE
E:      MOV #1,ABLE         ;LABELS C AND E ARE EQUATED WITH THE
                            ;NUMERICAL MEMORY ADDRESS OF THE MOV
                            ;COMMAND.
```

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:

```
RØ=%Ø          ;DEFINE RØ AS REGISTER Ø
R3=RØ+3        ;DEFINE R3 AS REGISTER 3
R4=1+%3        ;DEFINE R4 AS REGISTER 4
THERE=%2       ;DEFINE THERE AS REGISTER 2
```

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 (Chapter 10).

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

```
CLR  %6
```

will clear register 6 while the statement:

```
CLR  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:

```
JSR    5,SUBR    ;THE FIRST OPERAND FIELD MUST ALWAYS BE
                 ;A REGISTER.

CLR    X(2)      ;ANY EXPRESSION ENCLOSED IN ( ) MUST BE
                 ;A REGISTER.  IN THIS CASE, INDEX REGIS-
                 ;TER 2.
```

CHAPTER 4

EXPRESSIONS

Arithmetic and logical operators (see Section 4.2) may be used to form expressions. A term of an expression may be a permanent symbol, a 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. 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 (Chapter 10) 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

AND	OR
Ø & Ø = Ø	Ø ! Ø = Ø
Ø & 1 = Ø	Ø ! 1 = 1
1 & Ø = Ø	1 ! Ø = 1
1 & 1 = 1	1 ! 1 = 1

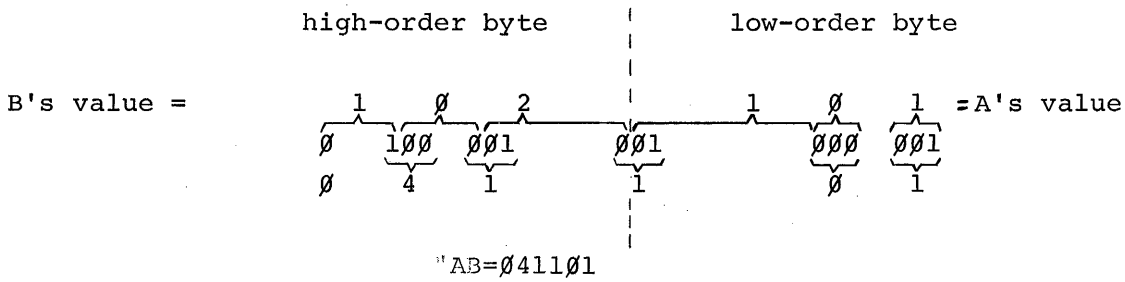
4.3 ASCII CONVERSION

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 (Appendix A). For example:

```
'A
```

is assigned the value 101_8.

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:



ASCII text is always absolute.

4.4 MODE OF EXPRESSIONS

The mode of an expression may be absolute, relocatable in the current program section, relocatable in another program section, or external as defined below.

A term of an expression is absolute, relocatable (current section or other section), or external, correspondingly as its definer (i.e., number, symbol, etc.) is absolute, relocatable (current or other), 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 where both items belong to the same program section or
3. Absolute expression followed by an operator followed by an absolute expression.

A relocatable (current or other program section) 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
RELC is a relocatable symbol in current program section
RELO is a relocatable symbol in non-current program section
EXT is an external symbol

Examples:

Legal Expressions	Illegal Expressions
;EXTERNAL EXPRESSIONS	EXT+RELC
EXT	EXT+RELO
EXT -ABS	RELC+RELC
ABS +EXT	RELO+RELO
	ABS-RELC
;RELOCATABLE EXPRESSION	RELC-RELO
	RELO-EXT
RELC	RELO!RELO
RELC+ABS	RELC&RELC
ABS+RELC	
RELC+RELC-RELC	
RELO	
RELO-ABS	
ABS+RELO	
RELO+RELO-RELO	
RELO-RELO+RELO	
;ABSOLUTE EXPRESSION	
RELC-RELC	
RELO-RELO	
ABS+RELC-RELC&ABS	
ABS+RELO-RELO!ABS	

CHAPTER 5

ASSEMBLY LOCATION COUNTER

The period (.) is the symbol for the assembly location counter. 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 #.,RØ      ;. 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
    .=5ØØ                  ;SET LOCATION COUNTER TO ABSOLUTE
                           ;500
```

```
FIRST:    MOV   .+10,COUNT   ;THE LABEL FIRST HAS THE VALUE 500(8)
                             ;.+10 EQUALS 510(8). THE CONTENTS OF
                             ;THE LOCATION 510(8) WILL BE DEPOSITED
                             ;IN LOCATION COUNT.

    .=520                    ;THE ASSEMBLY LOCATION COUNTER NOW
                             ;HAS A VALUE OF ABSOLUTE 520(8).

SECOND:   MOV   .,INDEX      ;THE LABEL SECOND HAS THE VALUE 520(8).
                             ;THE CONTENTS OF LOCATION 520(8), THAT
                             ;IS, THE BINARY CODE FOR THE INSTRUC-
                             ;TION ITSELF, WILL BE DEPOSITED IN
                             ;LOCATION INDEX.

.CSECT
    .=.+20                   ;SET LOCATION COUNTER TO RELOCATABLE
                             ;20 OF THE UNNAMED PROGRAM SECTION.

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 100_8 bytes of storage space in the program. The next instruction will be stored at 1100.

CHAPTER 6

RELOCATION AND LINKING

The output of the relocatable assembler is an object module which must be processed by the Link-11 Linker before loading and execution. (See PDP-11 Link-11 Linker, Programmer's Manual, DEC-11-ZLDA-D for details.) 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 directives to the Linker together with the required parameters. In the case of relocatable expressions the Linker adds the base of the associated 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 which are being linked together) 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 an apostrophe in the assembly listing. Thus the binary text output will look as follows for the given examples:

```
005065'  CLR  EXTERNAL(5)
000000                          ;VALUE OF EXTERNAL SYMBOL
                                ;ASSEMBLED ZERO; WILL BE
                                ;MODIFIED BY THE LINKER.

005065'  CLR  EXTERNAL+6(5)     ;THE ABSOLUTE PORTION OF THE
000006                          ;EXPRESSION (000006) IS ADDED
                                ;BY THE LINKER TO THE VALUE
                                ;OF THE EXTERNAL SYMBOL

005065'  CLR RELOCATABLE(5)     ;ASSUMING WE ARE IN THE ABSOLUTE
000040                          ;SECTION AND THE VALUE OF RELOCAT-
                                ;ABLE IS RELOCATABLE 40.
```

CHAPTER 7

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 (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.

a. Let E be any expression as defined in Chapter 4.

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

   Examples:

```
R0 = %0          ;GENERAL REGISTER 0
R1 = R0+1        ;GENERAL REGISTER 1
R2 = 1+%1        ;GENERAL REGISTER 2
```

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

d. Let A be a general address specification which produces a 6-bit mode address field as described in Section I, Chapter 2 of 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.

7.1 REGISTER MODE

The register contains the operand.

```
Format:     R
Example:    R0=%0           ;DEFINE R0 AS REGISTER 0
            CLR R0          ;CLEAR REGISTER 0
```

7.2 DEFERRED REGISTER MODE

The register contains the address of the operand.

```
Format:     @R or (ER)
Example:    CLR @R1         ;CLEAR THE WORD AT THE
              or            ;ADDRESS CONTAINED IN
            CLR (1)         ;REGISTER 1
```

7.3 AUTOINCREMENT MODE

The contents of the register are incremented immediately after being used as the address of the operand. (See note in Section 7.5)

```
Format:     (ER)+
Examples:   CLR  (R0)+      ;CLEAR WORDS AT ADDRESSES CON-
            CLR  (R0+3)+    ;TAINED IN REGISTERS 0, 3 AND 2
            CLR  (2)+       ;AND INCREMENT REGISTER CONTENTS
                            ;BY TWO.
```

NOTE

Both JMP and JSR instructions using 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:

```
MOV R0,(0)+     ;The quantity 102 is moved
                ;to location 100
MOV R0,-(0)     ;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.

```
Format:     @(ER)+
Example:    CLR @(3)+       ;CONTENTS OF REGISTER 3 POINT TO
                            ;ADDRESS OF WORD TO BE CLEARED
                            ;BEFORE BEING INCREMENTED BY TWO.
```

7.5 AUTODECREMENT MODE

The contents of the register are decremented before being used as the address of the operand (see note at top of page).

```
Format:     -(ER)
Examples:   CLR -(R0)       ;DECREMENT CONTENTS OF REGISTERS
            CLR -(R0+3)     ;0, 3, AND 2 BY TWO 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 BY
                            ;TWO 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:  CLR @14(4)     ;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:
          MOV #100, R0   ;MOVE AN OCTAL 100 TO REGISTER 0
          MOV  #X, R0    ;MOVE THE VALUE OF SYMBOL X TO
                         ;REGISTER 0.
```

The operation of this mode is explained as follows:

The statement MOV #100,R3 assembles as two words. These are:

```
0 1 2 7 0 3
0 0 0 1 0 0
```

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:
          CLR  100        ;CLEAR LOCATION 100
          MOV  X,Y        ;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 X-PC. The operation is explained as follows:

If the statement MOV 100,R3 is assembled at absolute location 20 then the assembled code is:

```
Location 20:          0 1 6 7 0 3
Location 22           0 0 0 0 5 4
```

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, 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

```
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

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.

Form	Mode	Meaning
R	0n	Register
@R or (ER)	1n	Register deferred
(ER)+	2n	Autoincrement
@(ER)+	3n	Autoincrement deferred
-(ER)	4n	Autodecrement
@-(ER)	5n	Autodecrement deferred

None of the above forms increases the instruction length.

E (ER)	6n	Index
@E(ER)	7n	Index deferred
#E	27	Immediate
@#E	37	Absolute memory reference
E	67	Relative
@E	77	Relative deferred reference

Any of the above 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 instruction CLR @#100 will clear location 100 even if the instruction is moved from the point at which it was assembled.

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 on page 7-1, and let A be a 6-bit address specification of any one of the forms:

E	@E
R	@R or (R)
(ER)+	@(ER)+
-(ER)	@-(ER)
E(ER)	@E(ER)
#E	@#E

Table 2.1 Instruction Operand Fields

Instruction	Form	Example
Double Operand	Op A,A	MOV (R6)+,@Y
Single Operand	Op A	CLR -(R2)
Operate	Op	HALT
Branch	Op E	BR X+2
		BLO .-4
	where $-128 \leq (E-.-2)/2 \leq 127_{10}$	
Subroutine Call	JSR ER,A	JSR PC,SUBR
Subroutine Return	RTS ER	RTS PC
EMT/TRAP	Op or OP E	EMT
	where $0 \leq E \leq 377_8$	EMT 31

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:

a) Extend the sign of the offset through bits 8-15.
b) Multiply the result by 2. This creates a word offset rather than a byte offset.
c) 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.

Byte offset = (E-PC)/2 truncated to eight bits.

Since PC = .+2, we have

Byte offset = (E-.-2)/2 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 or a relocatable symbol of another program section when within a relocatable section.

The EMT and TRAP instructions do not use the low-order byte of the word. This allows information 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 big (>377_8) it is truncated to eight bits and a Truncation (T) error occurs.

CHAPTER 8

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 statement. 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 assigned is the first symbol following the directive. If there is no .TITLE statement the default name assigned is ".MAIN.". If there is more than one .TITLE directive, only the last one encountered is operative.

8.2 .GLOBL

The .GLOBL directive is used to declare a symbol as being global. This is the mechanism by which separately assembled object modules communicate with one another. 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 Link-11.

The form of the .GLOBL directive is

```
.GLOBL  NAMA,NAMB,...,NAMN
```

NOTE

A symbol cannot be declared global by assigning it to a global expression in a direct assignment statement.

8.3 PROGRAM SECTION DIRECTIVES

```
.ASECT
.CSECT
.CSECT  symbol
```

The Assembler provides for 255_{10} program sections; an absolute section declared by .ASECT, an unnamed relocatable program section declared by .CSECT, and 253_{10} named relocatable program sections declared by .CSECT symbol, where symbol is any legal symbolic name. These directives allow the user to:

1. Create his program (object module) in sections:

The Assembler maintains separate location counters for each section. This allows the user to write statements which are not physically contiguous but will be loaded contiguously. The following examples will clarify this:

```
    .CSECT              ;START THE UNNAMED RELOCATABLE SECTION
A:  0                   ;ASSEMBLED AT RELOCATABLE 0,
B:  0                   ;     RELOCATABLE 2 AND
C:  0                   ;     RELOCATABLE 4,
ST: CLR A               ;ASSEMBLE CODE AT
    CLR B               ;     RELOCATABLE ADDRESS
    CLR C               ;     6 THROUGH 21
    .ASECT              ;START THE ABSOLUTE SECTION
    .=4                 ;ASSEMBLE CODE AT
    .WORD .+2,HALT      ;ABSOLUTE 4 THROUGH 7,
    .CSECT              ;RESUME THE UNNAMED RELOCATABLE
                        ;   SECTION
    INC A               ;ASSEMBLE CODE AT
    BR ST               ;   RELOCATABLE 22 THROUGH 27,
    .END
```

The first appearance of .CSECT or .ASECT assumes the location counter is at relocatable or absolute zero, respectively. The scope of each directive extends until a directive to the contrary is given. Further occurrences of the same .CSECT or .ASECT resume assemblying where the section was left off.

```
        .CSECT  COM1        ;DECLARE SECTION COM1
A:      0                   ;ASSEMBLED AT RELOCATABLE 0,
B:      0                   ;ASSEMBLED AT RELOCATABLE 2,
C:      0                   ;ASSEMBLED AT RELOCATABLE 4,
        .CSECT  COM2        ;DECLARE SECTION COM2
X:      0                   ;ASSEMBLED AT RELOCATABLE 0,
Y:      0                   ;ASSEMBLED AT RELOCATABLE 2,
        .CSECT  COM1        ;RETURN TO COM1
D:      0                   ;ASSEMBLED AT RELOCATABLE 6,
        .END
```

The Assembler automatically begins assemblying at relocatable zero of the unnamed .CSECT if not instructed otherwise; that is, the first statement of an assembly is an implied .CSECT.

All labels in an absolute section are absolute; all labels in a relocatable section are relocatable. The location counter symbol, ".", is relocatable or absolute when referenced in a relocatable or absolute section, respectively. Undefined internal symbols are assigned the value of relocatable or absolute zero in a relocatable or absolute section, respectively. Any labels appearing on a .ASECT or .CSECT statement are assigned the value of the location counter before the .ASECT or .CSECT takes effect. Thus, if the first statement of a program is:

```
A:      .ASECT
```

Then A is assigned to relocatable zero and is associated with the unnamed relocatable section (because the Assembler implicitly begins assembly in the unnamed relocatable section).

Since it is not known at assembly time where the program sections are to be loaded, all references between sections in a single assembly are translated by the Assembler to references relative to the base of that section. The Assembler provides the Linker with the necessary information to resolve the linkage.

Note that this is not necessary when making a reference to an absolute section (the Assembler knows all load addresses of an absolute section).

Examples:

```
        .ASECT
        .=1000
A:      CLR X              ;ASSEMBLED AS CLR BASE OF UNNAMED
                           ;   RELOCATABLE SECTION + 10
        JMP Y              ;ASSEMBLED AS JMP BASE OF UNNAMED
                           ;   RELOCATABLE SECTION + 6
        .CSECT
        MOV R0,R1
        JMP A              ;ASSEMBLED AS JMP 1000
Y:      HALT
X:      0
        .END
```

In the above example the references to X and Y were translated into references relative to the base of the unnamed relocatable section.

2. Share code and/or data between object modules (separate assemblies):

Named relocatable program sections operate as Fortran labeled COMMON; that is, sections of the same name from different assemblies are all loaded at the same location by Link-11. The unnamed relocatable section is the exception to this as all unnamed relocatable sections are loaded in unique areas by Link-11.

Note that there is no conflict between internal symbolic names and program section names; that is, it is legal to use the same symbolic name for both purposes. In fact, considering Fortran again, this is a necessity to accommodate the Fortran statement

```
COMMON /X/A,B,C,X
```

Where the symbol X represents the base of this program section and also the 4th element of this program section.

Also, there is no conflict between program section names and .GLOBL names. In FORTRAN language, COMMON block names and SUBROUTINE names may be the same.

8.4 .EOT

Under the Disk Operating System, the .EOT directive is effectively ignored. The physical End Of Tape itself allows several physically separate tapes to be assembled as one program. The last tape should be terminated by a .END directive; however, in its absence, the TAPES command string switch (Section 9.3) may be used.

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 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:

```
.=1420
SAL=0
.WORD 177535,.+4,SAL          ;STORED IN WORDS 1420, 1422, AND
                              ;1424 WILL BE 177535, 1426, AND 0.
```

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          ;ZERO, FIVE, AND ZERO ARE STORED
.WORD ,5,       ;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, ! or &. For example,

```
.=440                         ;THE OP-CODE FOR MOV, WHICH
                              ;    IS 010000,
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, <u>if an instruction mnemonic or assembler directive is misspelled, the .WORD directive is assumed and errors will result.</u> Assume that MOV is spelled incorrectly as MOR:

```
MOR  A,B
```

Two error codes can result: 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                   ;STORED IN LOCATION 410 WILL BE
.=410                   ;060 (THE OCTAL EQUIVALENT OF 48).
.BYTE 48.,SAM           ;IN 411 WILL BE 005.
```

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:

```
.=420           ;ZERO WILL BE STORED IN
.BYTE , ,       ;BYTES 420, 421 and 422.
```

If the expression is relocatable, a warning flag, A, will be given, since at link time it is very likely that the relocation will result in an expression of more than 8 bits.

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:

```
.=500                           ;THE ASCII CODE FOR "Y" WILL BE
.ASCII      /YES/               ;STORED IN 500, THE CODE FOR "E"
                                ;IN 501, THE CODE FOR "S" IN 502.

.ASCII      /5+3/2/             ;THE DELIMITING CHARACTER OCCURS
                                ;AMONG THE OPERANDS. THE ASCII
                                ;CODES FOR "5", "+", AND "3" ARE
                                ;STORED IN BYTES 503,504, AND
                                ;505. 2/ IS NOT ASSEMBLED.
```

The .ASCII directive may be terminated by any legal terminator.

8.10 .RAD50

PDP-11 systems 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 6-character 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.

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:

Character	RADIX 50 Equivalent (octal)
(space)	0
A-Z	1-32
$	33
.	34
0-9	36-47

Note that another character could be defined for code 35.

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

```
RESULT=((C1*50)+C2)*50+C3
```

8.11 .LIMIT

A program often wishes to know the boundaries of the load module's 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 conditonal 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)
```

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_{10}. Labels are permitted on conditional directives, but the scan is purely left to right. For example:

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

A is ignored.

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

A is entered in the symbol table.

If an .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.

CHAPTER 9

OPERATING PROCEDURES

The Assembler enables you to assemble into a relocatable binary file (object module) an ASCII source containing PAL-11R statements. 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 command output device. On subsequent passes the Assembler assembles the program and produces the outputs specified during the initial dialogue (Section 9.2). The Assembler initiates the dialogue immediately after being loaded and after the last pass of an assembly.

9.1 LOADING PAL-11R

PAL-11R is loaded via the Disk Monitor command RUN. For example:

```
RUN PAL11R
```

9.2 INITIAL DIALOGUE

When the Assembler is ready to accept the user's command string, it outputs the following lines to the command device:

```
PAL11R  V002A
#
```

The user must now type his command string on the same line as the #.

If an error is made in typing at any time, typing the RUBOUT key will erase the immediately preceding character if it is on the current line. Typing CTRL/U will erase the entire line.

The format of the command string adheres to the requirements of the Disk Operating System's (DOS) Command String Interpreter (CSI). The Assembler's file specifications must appear in the following order:

Binary, Listing, Symbol Table < Source 1, Source 2

A null specification field signifies that the associated input or output type is not desired. Each specification contains the following information:

dev:filnam.ext[uic]/PASS:value/TAPES:value

If a syntactical error is detected in the command string, the Assembler will output the command string up to and including the point where the error was detected followed by the character "?". The Assembler will then output # again, and the user must retype his entire command.

The following command string semantic errors will be detected:

		Error Code
1.	Illegal switch Too many switches Illegal switch value Too many switch values	S203
2.	Too many output file specifications	S204
3.	Too many input file specifications	S205
4.	Input file missing	S206

After outputting the appropriate error code, the Assembler will output # again and the user must retype his entire command.

The default value for each specification is noted below:

	DEV	FILNAM	EXT	UIC	PASS	TAPES
Source 1	DF0	None	PAL	This User	All Passes***	1
Source 2	DF0*	None	PAL	This User	All Passes	1
Binary	DF0	Same as Source** FILNAM	OBJ	This User	2	
Listing	DF0*	Same as Source** FILNAM	LST	This User	2	
Symbols	DF0*	Same as Source** FILNAM	SYM	This User	End of Pass 1	

* Or last device specified on this side of the <.

** The default FILNAM is Source 2's if this field is present; otherwise, Source 1's is used.

*** If /PASS:n (where $0 < n < 4$) is in the Source 1 field, this file will only be accessed during pass 1. This is useful for parameter assignments associated with conditional assemblies.

Source 1 and Source 2 cannot both be associated with the same unit file device such as the paper tape reader. This will result in a Monitor error.

The TAPES switch is recognized in the two input fields of PAL-11R's command string; it is ignored in the output fields. The value associated with the switch specifies the maximum decimal number of tapes* that will be processed from the designated device. The .END assembly directive always terminates an assembly pass; however, if it is missing, the maximum tape count will effect termination. In the latter case, the error line count output by the Assembler is incremented by one to warn the user of the omission. The default number of tapes is 1. The following examples should clarify the usages of the TAPES switch.

1. #PP:,LP:,LP:<PR:/TAPES:5
   All passes of the assembly will terminate either when a) the .END directive is encountered or b) five tapes have been input from the reader.

2. #PP:,LP:<PR:
   All passes of the assembly will terminate when one tape has been input from the reader.

3. #PP:<PR:/PASS:1/TAPES:2,DT1:FILE
   Two parameter tapes will be input from the reader during pass 1 only. All passes of the assembly will terminate either when a) the .END directive is encountered or b) the source file FILE.PAL has been input from DT1.

4. #PP:<DT1:PARAM/PASS:1,PR:/TA:4
   One parameter file, PARAM.PAL, will be input from DT1 during pass 1 only. All passes of the assembly will terminate either when a) the .END directive is encountered or b) four tapes have been input from the reader.

---

* This switch is most applicable to non-file oriented devices such as the paper tape reader. However, it would cause the same file on DECtape to be processed the specified number of times during each pass.

The 37 switch is recognized in all the fields of PAL-11R's command string. Its presence directs the Assembler to assemble all relative addresses (address mode 67) as absolute addresses (address mode 37).

e.g., CLR ADDR normally assembles as

ØØ5Ø67
  N

where N is the offset between ADDR and the current PC.

When the 37 switch has been encountered in the command string

CLR ADDR assembles as

ØØ5Ø37
  ADDR

just as if the source code read

CLR @#ADDR

This switch is useful during the debugging phase of program development as it facilitates tracing through assembly listings. Note that when the 37 switch is used, position independent coding is not possible.

The following example command string shows the 37 switch specification

```
#PP:,LP:/37<DT1:FILE
```

Note that DOS only allows one output file to be open on a DECtape at a time. The Assembler loosens this restriction by:

1. Providing the PASS switch, for example:

   ```
   #DTA1:,DTA1:/PASS:3<TEST
   ```

   This command causes the input file TEST.PAL on DF0 to be assembled with binary output, TEST.OBJ, to DTA1 during pass 2 and listing output, TEST.LST, to DTA1 during pass 3.

2. Closing files and releasing datasets[1] as soon as possible, for example:

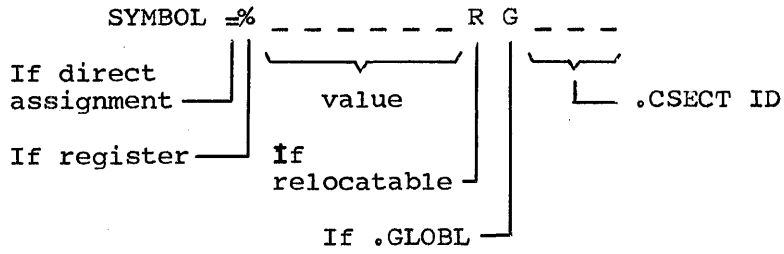
   ```
   #,DTA1:,DTA1:/PASS:2<TEST
   ```

---

1 The releasing of datasets has the additional advantage that all drivers and buffers do not have to be core resident during the entire assembly.

This command causes the input file TEST.PAL on DF0 to be assembled with no binary outout, listing output, TEST.LST to DTA1 during pass 2 and symbol table output, TEST.SYM to DTA1 at the end of pass 2.

The symbol table is output alphabetically, three symbols per line. Each symbol is output in the following format:



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, relocatable, absolute, or global.

The .CSECT ID field is left blank for .ASECT symbols and unnamed .CSECT symbols as they can be identified by the absence or presence of the R , respectively.

Symbols belonging to named .CSECT's have the ID's 002 through 376_8 associated with them, where the nth named .CSECT encountered will have the ID n+1.

Immediately following the symbol table listing will be a listing of the .CSECT names, their sizes in bytes, and their corresponding ID's. There will be one entry per line in the following format:

```
FIRST       ØØ4Ø2Ø      ØØ2
SECOND      ØØØ376      ØØ3
```

.
.
.

When no outputs are omitted and the default passes are in effect, 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 .GLOBL's) are listed on the command output device at the end of the pass. The symbol table is also output at this time.

PASS 2: Assembler outputs the object module, and the listing file which includes the pass error line count (in octal), and prints the pass error line count on the command output device.

9.3 ASSEMBLY DIALOGUE

During assembly, the Assembler may output various messages to the command device. The message may be:

1. informative,
2. request for operator action, or
3. terminal.

The operator may also initiate a message. For example, CTRL/C and RESTART at any time, to stop the assembly process and restart the initial dialogue as mentioned in the previous section.

At the end of each pass (except the last) when the .END assembly directive is encountered, END is output to the command device. For file structured input devices, the next pass is automatically begun. For non-file structured input devices, e.g., paper tape reader, the Disk Operating System will announce that the reader must be reloaded prior to continuing:

```
A002    063320
$
```

A002 denotes device not ready, and
063320 is the radix 50 notation for PR.

After the user has reloaded the reader, he should type CTRL/C (↑C) on the command input device. When the monitor acknowledges with the character . , the user should type the monitor command CONTINUE.

This continuing from device-not-ready is also applicable when .EOT is encountered during multiple tape assemblies and when any I/O device requires servicing for such purposes as:

a. line printer paper
b. write enable DECtape
c. tape for punch.

The following terminal errors are printed on the command device. An attempt is made to close all output files and delete the binary file, if open. The Assembler then outputs # and waits for further commands:

S200	more than 376_8 .CSECT's
S201	more than 177_8 nested conditionals
S202	binary or listing file structured device full or input command string too long
S207	input .TRAN error on Assembler's overlay file

RESTART capability:

The assembly may be aborted at any time by typing CTRL/C followed by either the Monitor command RESTART or the Monitor command BEGIN on the command device. All open output files will be closed and released. If the binary file was open, it is deleted. The Assembler will output # and wait for the next command string. It does this automatically at the normal completion of an assembly.

9.4 ASSEMBLY LISTING

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

```
EELLLLLL·OOOOOOASSS.......S
         OOOOOO
         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. The A represents an apostrophe which indicates that either the second, third, or both words of the instruction are to be modified by the Linker.

The above represent 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).

9.5 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)

9.5.1 Global Symbol Directory

As the name suggests, the Global Symbol Directory (GSD) contains a list of all global symbols (entry points and externals) with all entry points associated with the program section symbol in which they reside. Each symbol is in RADIX 50 form and contains

mode information and relative addresses for entry points, and section sizes for program section names. The name of the object module is also in the GSD.

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

9.5.2. Text Block

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

9.5.3 Relocation Directory

The Relocation Directory (RLD) blocks consist of directives to the Linker which may reference the text block preceding it. 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 Text or RLD buffer is full and whenever the location counter needs to be modified.

CHAPTER 10

ERROR CODES

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

Error Code	Meaning
A	Addressing error. An address within the instruction is incorrect. Also may indicate a relocation error.
B	Bounding error. Instructions or word data are being assembled at an odd address in memory. The location counter is updated by +1.
D	Doubly-defined symbol referenced. Reference was made to a symbol which is defined more than once.
I	Illegal character detected. Illegal characters which are also non-printing are replaced by a ? on the listing.
L	Line buffer overflow. Extra characters on a line (more than 72_{10}) are ignored.
M	Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.
N	Number containing 8 or 9 has decimal point missing.
P	Phase error. A label's definition or value varies from one pass to another.
Q	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.
R	Register-type error. An invalid use of or reference to a register has been made.
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.

APPENDIX A

ASCII CHARACTER SET

EVEN PARITY BIT	7-BIT OCTAL CODE	CHARACTER	REMARKS
Ø	ØØØ	NUL	NULL, TAPE FEED, CONTROL SHIFT P.
1	ØØ1	SOH	START OF HEADING; ALSO SOM, START OF MESSAGE, CONTROL A.
1	ØØ2	STX	START OF TEXT; ALSO EOA, END OF ADDRESS, CONTROL B.
Ø	ØØ3	ETX	END OF TEXT; ALSO EOM, END OF MESSAGE, CONTROL C.
1	ØØ4	EOT	END OF TRANSMISSION (END); SHUTS OFF TWX MACHINES, CONTROL D.
Ø	ØØ5	ENQ	ENQUIRY (ENQRY); ALSO WRU, CONTROL E.
Ø	ØØ6	ACK	ACKNOWLEDGE; ALSO RU, CONTROL F.
1	ØØ7	BEL	RINGS THE BELL. CONTROL G.
1	Ø1Ø	BS	BACKSPACE; ALSO FEO, FORMAT EFFECTOR. BACKSPACES SOME MACHINES, CONTROL H.
Ø	Ø11	HT	HORIZONTAL TAB. CONTROL I.
Ø	Ø12	LF	LINE FEED OR LINE SPACE (NEW LINE); ADVANCES PAPER TO NEXT LINE, DUPLICATED BY CONTROL J.
1	Ø13	VT	VERTICAL TAB (VTAB). CONTROL K.
Ø	Ø14	FF	FORM FEED TO TOP OF NEXT PAGE (PAGE). CONTROL L.
1	Ø15	CR	CARRIAGE RETURN TO BEGINNING OF LINE. DUPLICATED BY CONTROL M.
1	Ø16	SO	SHIFT OUT; CHANGES RIBBON COLOR TO RED. CONTROL N.
Ø	Ø17	SI	SHIFT IN; CHANGES RIBBON COLOR TO BLACK. CONTROL O.
1	Ø2Ø	DLE	DATA LINK ESCAPE. CONTROL P (DCØ).
Ø	Ø21	DC1	DEVICE CONTROL 1, TURNS TRANSMITTER (READER) ON, CONTROL Q (X ON).
Ø	Ø22	DC2	DEVICE CONTROL 2, TURNS PUNCH OR AUXILIARY ON. CONTROL R (TAPE, AUX ON).
1	Ø23	DC3	DEVICE CONTROL 3, TURNS TRANSMITTER (READER) OFF, CONTROL S (X OFF).
Ø	Ø24	DC4	DEVICE CONTROL 4, TURNS PUNCH OR AUXILIARY OFF. CONTROL T (~~TAPE~~, AUX OFF).
1	Ø25	NAK	NEGATIVE ACKNOWLEDGE; ALSO ERR, ERROR. CONTROL U.
1	Ø26	SYN	SYNCHRONOUS IDLE (SYNC). CONTROL V.
Ø	Ø27	ETB	END OF TRANSMISSION BLOCK; ALSO LEM, LOGICAL END OF MEDIUM. CONTROL W.
Ø	Ø3Ø	CAN	CANCEL (CANCL). CONTROL X.
1	Ø31	EM	END OF MEDIUM. CONTROL Y.
1	Ø32	SUB	SUBSTITUTE. CONTROL Z.
Ø	Ø33	ESC	ESCAPE. PREFIX. CONTROL SHIFT K.
1	Ø34	FS	FILE SEPARATOR. CONTROL SHIFT L.

EVEN PARITY BIT	7-BIT OCTAL CODE	CHARACTER
0	167	w
0	170	x
1	171	y
1	172	z
0	173	{
1	174	|

APPENDIX B

PAL-11A ASSEMBLY LANGUAGE AND ASSEMBLER

B.1 SPECIAL CHARACTERS

Character	Function
form feed	Source line terminator
line feed	Source line terminator
carriage return	Source statement terminator
:	Label terminator
=	Direct assignment indicator
%	Register term indicator
tab	Item terminator Field terminator
space	Item terminator Field terminator
#	Immediate expression indicator
@	Deferred addressing indicator
(	Initial register indicator
)	Terminal register indicator
,	Operand field separator
;	Comment field indicator
+	Arithmetic addition operator
-	Arithmetic subtraction operator
&	Logical AND operator
!	Logical OR operator
"	Double ASCII character indicator
'	Single ASCII character indicator
.	Assembly location counter

B.2 ADDRESS MODE SYNTAX

n is an integer between 0 and 7 representing a register. R is a register expression, E is an expression, ER is either a register expression or an expression in the range 0 to 7.

Format	Address Mode Name	Address Mode Number	Meaning
R	Register	0n	Register R contains the operand. R is a register expression.
@R or (ER)	Deferred Register	1n	Register R contains the operand address.
(ER)+	Autoincrement	2n	The contents of the register specified by ER are incremented after being used as the address of the operand.
@(ER)+	Deferred Autoincrement	3n	ER contains the pointer to the address of the operand. ER is incremented after use.
-(ER)	Autodecrement	4n	The contents of register ER are decremented before being used as the address of the operand.
@-(ER)	Deferred Autodecrement	5n	The contents of register ER are decremented before being used as the pointer to the address of the operand.
E(ER)	Index	6n	E plus the contents of the register specified, ER, is the address of the operand.
@E(ER)	Deferred Index	7n	E added to ER gives the pointer to the address of the operand.
#E	Immediate	27	E is the operand.
@#E	Absolute	37	E is the address of the operand.
E	Relative	67	E is the address of the operand.
@E	Deferred Relative	77	E is the pointer to the address of the operand.

B.3 INSTRUCTIONS

The instructions which follow are grouped according to the operands they take and 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 operation caused an overflow
C	Carry bit:	set if the operation caused 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 determine the state of the code (see the PDP-11 Handbook).

Logical operations are represented by the following symbols:

!	Inclusive OR
⊕	Exclusive OR
&	AND
¯	(used over a symbol) NOT (i.e., 1's complement)

B.3.1 Double-Operand Instructions Op A,A

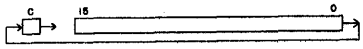
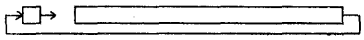
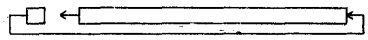
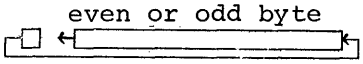
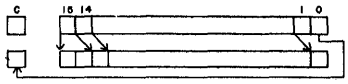
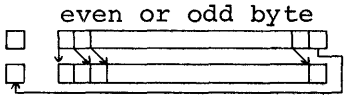
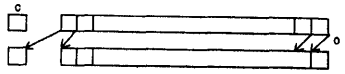
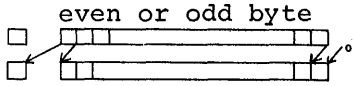
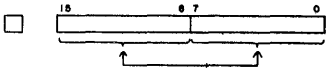
Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
01SSDD	MOV	MOVe	(SE)→(DE)	*	*	0	-
11SSDD	MOVB	MOVe Byte					
02SSDD	CMP	CoMPare	(SE)-(DE)	*	*	*	*
12SSDD	CMPB	CoMPare Byte					
03SSDD	BIT	BIt Test	(SE)&(DE)	*	*	0	-
13SSDD	BITB	BIt Test Byte					
04SSDD	BIC	BIt Clear	$(\overline{SE})$&(DE)→(DE)	*	*	0	-
14SSDD	BICB	BIt Clear Byte					
05SSDD	BIS	BIt Set	(SE)!(DE)→(DE)	*	*	0	-
15SSDD	BISB	BIt Set Byte					
06SSDD	ADD	ADD	(SE)+(DE)→(DE)	*	*	*	*
16SSDD	SUB	SUBtract	(DE)-(SE)→(DE)	*	*	*	*

B.3.2 Single-Operand Instructions Op A

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
0050DD	CLR	CLeaR	Ø→(DE)	0	1	0	0
1050DD	CLRB	CLeaR Byte					
0051DD	COM	COMplement	$(\overline{DE})$→(DE)	*	*	0	1
1051DD	COMB	COMplement Byte					
0052DD	INC	INCrement	(DE)+1→(DE)	*	*	*	-
1052DD	INCB	INCrement Byte					
0053DD	DEC	DECrement	(DE)-1→(DE)	*	*	*	-
1053DD	DECB	DECrement Byte					
0054DD	NEG	NEGate	$(\overline{DE})$+1→(DE)	*	*	*	*
1054DD	NEGB	NEGate Byte					

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
0055DD 1055DD	ADC ADCB	ADd Carry ADd Carry Byte	(DE)+(C)→(DE)	*	*	*	*
0056DD 1056DD	SBC SBCB	SuBtract Carry SuBtract Carry Byte	(DE)-(C)→(DE)	*	*	*	*
0057DD 1057DD	TST TSTB	TeST TeST Byte	(DE)-Ø→(DE)	*	*	0	0

B.3.3 Rotate/Shift Instructions Op A

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
0060DD	ROR	ROtate Right		*	*	*	*
1060DD	RORB	ROtate Right Byte	even or odd byte 	*	*	*	*
0061DD	ROL	ROtate Left		*	*	*	*
1061DD	ROLB	ROtate Left Byte	even or odd byte 	*	*	*	*
0062DD	ASR	Arithmetic Shift Right		*	*	*	*
1062DD	ASRB	Arithmetic Shift Right Byte	even or odd byte 	*	*	*	*
0063DD	ASL	Arithmetic Shift Left		*	*	*	*
1063DD	ASLB	Arithmetic Shift Left Byte	even or odd byte 	*	*	*	*
0001DD	JMP	JuMP	DE→(PC)	-	-	-	-
0003DD	SWAB	SWAp Bytes		*	*	0	0

B.3.4 Operate Instructions

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
000000	HALT	HALT	The computer stops all functions.	-	-	-	-
000001	WAIT	WAIT	The computer stops and and waits for an interrupt.	-	-	-	-
000002	RTI	ReTurn from Interrupt	The PC and PS are popped off the SP stack: ((SP))→(PC) (SP)+2→(SP) ((SP))→(PS) (SP)+2→(SP) RTI is also used to return from a trap.	*	*	*	*
000005	RESET	RESET	Returns all I/O devices to power-on state.	-	-	-	-

Condition Code Operates

Op-Code	Mnemonic	Stands for
000241	CLC	CLear Carry bit in PS.
000261	SEC	SEt Carry bit.
000242	CLV	CLear oVerflow bit.
000262	SEV	SEt oVerflow bit.
000244	CLZ	CLear Zero bit.
000264	SEZ	SEt Zero bit.
000250	CLN	CLear Negative bit.
000270	SEN	SEt Negative bit.
000254	CNZ	CLear Negative and Zero bits.
000257	CCC	Clear all Condition Codes
000277	SCC	Set all Condition Codes.
000240	NOP	No OPeration.

B.3.5 Trap Instructions Op or Op E where $0 \leq E \leq 377_8$
*OP (only)

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
*000003	(none)	(breakpoint trap)	Trap to location 14. This is used to call ODT.	*	*	*	*
*000004	IOT	Input/Output Trap	Trap to location 20. This is used to call IOX.	*	*	*	*
104000-104377	EMT	EMulator Trap	Trap to location 30. This is used to call system programs.	*	*	*	*
104400 104777	TRAP	TRAP	Trap to location 34. This is used to call any routine desired by the programmer.	*	*	*	*

B.3.6 Branch Instructions Op E where $-128_{10} \leq (E--2)/2 \leq 127_{10}$

Op-Code	Mnemonic	Stands for	Condition to be met if branch is to occur
0004XX	BR	BRanch always	
0010XX	BNE	Branch if Not Equal (to zero)	Z=0
0014XX	BEQ	Branch if EQual (to zero)	Z=1
0020XX	BGE	Branch if Greater than or Equal (to zero)	N⊕V=0
0024XX	BLT	Branch if Less Than (zero)	N ⊕ V=1
0030XX	BGT	Branch if Greater Than (zero)	Z!(N⊕V)=0
0034XX	BLE	Branch if Less than or Equal (to zero)	Z!(N⊕V)=1
1000XX	BPL	Branch if PLus	N=0
1004XX	BMI	Branch if MInus	N=1
1010XX	BHI	Branch if HIgher	C ! Z = 0
1014XX	BLOS	Branch if LOwer or Same	C ! Z = 1
1020XX	BVC	Branch if oVerflow Clear	V=0
1024XX	BVS	Branch if oVerflow Set	V=1
1030XX	BCC (or BHIS)	Branch if Carry Clear (or Branch if HIgher or Same)	C=0
1034XX	BCS (or BLO)	Branch if Carry Set (or Branch if LOwer)	C=1

B.3.7 Subroutine Call Op ER, A

Op-Code	Mnemonic	Stands for	Operation
004RDD	JSR	Jump to SubRoutine	Push register on the SP stack, put the PC in the register: DE→(TEMP) - a temporary storage register internal to processor. (SP)-2→(SP) (REG)→((SP)) (PC)→(REG) (TEMP)→(PC)

B.3.8 Subroutine Return Op ER

00020R	RTS	ReTurn from Subroutine	Put register in PC and pop old contents from SP stack into register.

B.4 ASSEMBLER DIRECTIVES

Mnemonic	Operand	Stands for	Operation
.EOT	none	End Of Tape	Indicates the physical end of the source input medium
.EVEN	none	EVEN	Insures that the assembly location counter is even by adding 1 if it is odd.
.END	E	END	Indicates the physical end of the program and optionally specifies the transfer address (E).
.WORD	E, E,... E, E,...	WORD (the void operator)	Generates words of data 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.
.ASECT	none	Absolute SECTion	Initiates the Absolute section.
.CSECT	name	Control SECTion	Initiates and identifies Relocatable Program section (section is unnamed if no operand).
.LIMIT	none	LIMIT	Generates two words containing the low and high limits of the relocatable section.
.GLOBL	name,name,...	GLOBaL	Specifies each name to be a global symbol.
.RAD50	/XXX/	RADix 50	Generates the RADIX 50 representation of the ASCII characters in delimiters.

Mnemonic	Operand	Stands For	Operation
.IFZ	E	IF E=0	Assemble what follows up to the terminating .ENDC if the expression E is 0.
.IFNZ	E	IF E≠0	Assemble what follows up to the terminating .ENDC, if the expression E is not 0.
.IFL	E	IF E<0	Assemble what follows up to the terminating .ENDC, if the expression E is less than 0.
.IFLE	E	IF E≤0	Assemble what follows up to the terminating .ENDC, if the expression E is less than or equal to 0.
.IFG	E	IF E>0	Assemble what follows up to the terminating .ENDC, if the expression E is greater than 0.
.IFGE	E	IF E≥0	Assemble what follows up to the terminating .ENDC, if the expression E is greater than or equal to 0.
.IFDF	name	IF name DeFined	Assemble what follows up to the terminating .ENDC if the symbol name is defined.
.IFNDF	name	IF name uNDeFined	Assemble what follows up to the terminating .ENDC if the symbol name is undefined.
.ENDC	none	END of Conditional	Terminates the range of a conditional directive.

B.5 ERROR CODES

Error Code	Meaning
A	Addressing error. An address within the instruction is incorrect. Also includes relocation errors.
B	Bounding error. Instructions or word data are being assembled at an odd address in memory.
D	Doubly-defined symbol referenced. Reference was made to a symbol which is defined more than once.
I	Illegal character detected. Illegal characters which are also non-printing are replaced by a ? on the listing.

Error Code	Meaning
L	Line buffer overflow. All extra characters beyond 72 are ignored.
M	Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.
N	Number containing an 8 or 9 was not terminated by a decimal point.
P	Phase error. A label's definition or value varies from one pass to another.
Q	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.
R	Register-type error. An invalid use of or reference to a register has been made.
T	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.
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.

APPENDIX C

LINKING PAL-11R AND ITS OVERLAY BUILDER AND CONSTRUCTING THE RUN TIME SYSTEM

At assembly time, the Assembler exists as:

1. a load module of its core resident portion, and
2. N non-resident overlays in a contiguous file.

The Assembler package is delivered to the customer as three (3) object modules:

1. the Assembler (PAL11R)
2. the Permanent Symbol Table (PALSYM)
3. the Overlay Builder (OVRBLD)

Two links must be built from these object modules:

1. the Assembler and the Permanent Symbol Table
2. the Overlay Builder

It is imperative that both links have exactly the same TOP specified at link time.

The load module of the Overlay Builder must now be executed via the Monitor command RUN. It will produce a contiguous file, PAL11R.OVR, on DF$\emptyset$[1] with User Identification Code [1,1]. Monitor commands CTRL/C BEGIN or CTRL/C RESTART will abort the build process and automatically begin it again. When the building is done, control will return to the monitor.

The load module of the Assembler/Permanent Symbol Table may now be executed via the monitor command RUN. This run requires the presence of the overlay file, PAL11R.OVR, on the overlay device DF$\emptyset$[1] with User Identification Code [1,1].

Overlay Size and Makeup

The size of each overlay on the overlay device and the size of the

[1]The file specifier and device may be changed with the Monitor command ASSIGN on logical dataset OVR prior to issuing the RUN command.

overlay area in core is currently 512 decimal words. This size was chosen after examination of the segmentable sections of the Assembler and their relationships to each other.

There are currently five overlays:

```
OVERLAY #1
   Assembler Initialization
   CSI Interface
   CTRL/C RESTART handler
OVERLAY #2
   .CSECT   assembler directive
   .ASECT      "          "
   .GLOBL      "          "
   .TITLE      "          "
   .LIMIT      "          "
   GSD (Global Symbol Directory) Output
   Linear Search Routine (used by .CSECT and
            GSD Output routine)
OVERLAY #3
   .END  assembler directive
   Conditional assembler directives
   Symbol Table Lister
OVERLAY #4
   .ASCII  assembler directive
   .BYTE       "         "
   .RAD5Ø      "         "
   .WORD       "         "
   Direct Assignment processor
   PASS Initialization
OVERLAY #5
   GET INPUT LINE ROUTINE
   STATEMENT EVALUATOR
   LABEL FIELD PROCESSOR
   INSTRUCTION PROCESSORS
   PHYSICAL END OF INPUT (EOD) PROCESSOR
```

Programming Hints for the User

The user can organize his source code to cut down on the number of overlay transfers needed to accomplish his assembly.

He should gather all his executable code into one area because once OVERLAY #5 is loaded into core to process source lines restricted to labels and instructions it will remain in core until one of the following occurs:

1. an assembly directive is encountered
   e.g., .WORD
   .CSECT
2. a non-instruction line defaults to .WORD
   e.g., ADDR

3. a direct assignment statement is encountered

   e.g., A=1ØØ
   .=.+6Ø

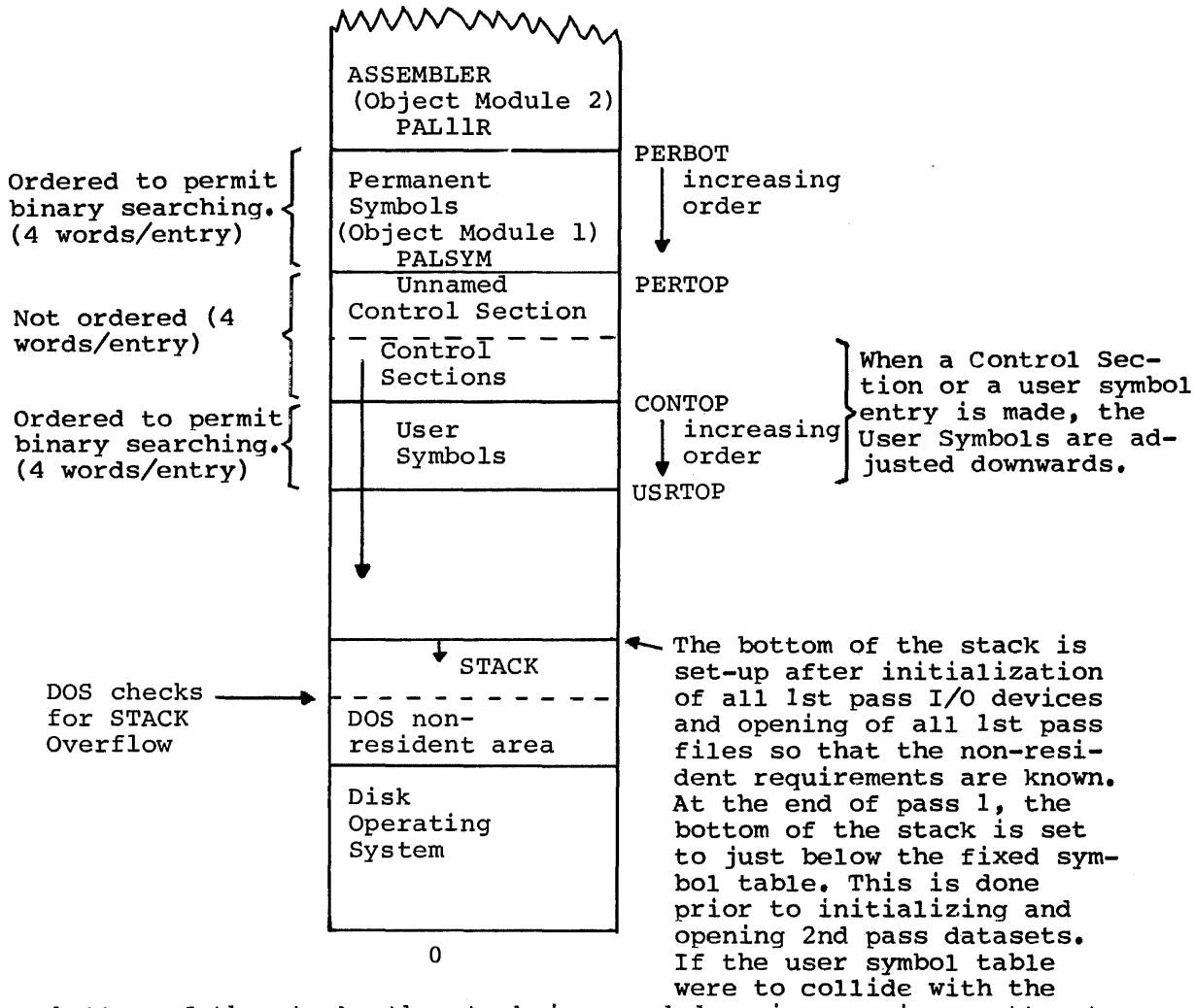
4. the end of a pass of the assembly process

Linking the Assembler and the Permanent Symbol Table

The load module of the Assembler and the Permanent Symbol Table should be given the file name:

PAL11R

PALSYM must be the first object module input to the Linker and PAL11R the second. This order allows for assembly time core utilization as illustrated by the memory map on the following page.



The format of the Permanent Symbol Table (PST) is described below to assist users who wish to add and/or delete symbols.

The PST is a separate object module which is loaded to reside immediately below the assembler in core memory. Once assembled, it is fixed in size; however, it may be edited, assembled, and linked to suit the specific needs of customers.

The PST is bounded by the internal global symbols SYMTBB and SYMTBT, where the former is the highest address of the PST and the latter is the address of the 1st word below the PST.

The PST is ordered (to permit binary searching) with the smallest symbol (in Radix50 packed notation) high in core and the largest symbol (in Radix50 packed notation) low in core.

Radix50 character representation:

```
A thru Z  :  1 thru 32 respectively
       $  :  33
       .  :  34
0 thru 9  :  36 thru 47 respectively
```

Radix50 packed notation:

```
1st packed triad = CHAR1*50*50+CHAR2*50+CHAR3
2nd packed triad = CHAR4*50*50+CHAR5*50+CHAR6
```

Each entry is 4 words with the lowest word containing the 1st packed triad. The next lowest containing the 2nd packed triad. The next lowest containing the value (which for assembler directives is an external global to be linked to the appropriate processor in PAL-11R) and the highest word containing the flags in the low byte and the control section ID (which is always 0 for PST entries) in the high byte.

Internal global symbols:

```
       .GLOBL SYMTBB, SYMTBT
```

External global symbols:

```
       .GLOBL  ASCII, ASECT, BYTE, CSECT, END
       .GLOBT  ENDC, EOT, EVEN, GLOBL, IFDF
       .GLOBL  IFG, IFGE, IFL, IFLE, IFNDF
       .GLOBL  IFNZ, IFZ, LIMIT, RAD50
       .GLOBL  TITLE, WORD
```

Flags:

ASMDIR=10	Bit 3 being on in the flag Byte indicates that this PST entry is an assembler directive.
BYTFLG=1	Bit 0 being on in the flag Byte indicates that this PST entry is byte enabled. This allows one entry to satisfy searches for word and byte instructions. E.G. the entry 'MOV', because bit 0 is on, will satisfy searches for 'MOV' or 'MOVB'

Instruction Class:
Bits 4-7 of the flag byte designate the type of instruction to provide dispatch information to PAL-11R.

SCLAS0=0	Operate group
SCLAS1=20	Unary group
SCLAS2=40	Binary group
SCLAS3=60	RTS
SCLAS4=100	Branch group
SCLAS5=120	JSR
SCLAS6=140	TRAP group

A listing of the Permanent Symbol Table follows:

```
        .EVEN
SYMTBT: .WORD   0               ;1ST REGISTER BELOW PST,
;
        .WORD   131247          ; .WORD
        .WORD   070440
        .WORD   WORD
        .WORD   ASMDIR
;
        .WORD   131051          ; .TITLE
        .WORD   077345
        .WORD   TITLE
        .WORD   ASMDIR
;
        .WORD   130721          ; .RAD50
        .WORD   017226
        .WORD   RAD50
        .WORD   ASMDIR
;
        .WORD   130351          ; .LIMIT
        .WORD   051274
        .WORD   LIMIT
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFZ
        .WORD   121200
        .WORD   IFZ
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFNZ
        .WORD   055620
        .WORD   IFNZ
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFNDF
        .WORD   054046
        .WORD   IFNDF
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFLE
        .WORD   045710
        .WORD   IFLE
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFL
        .WORD   045400
        .WORD   IFL
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFGE
        .WORD   026210
        .WORD   IFGE
        .WORD   ASMDIR
;
        .WORD   130156          ; .IFG
        .WORD   025700
        .WORD   IFG
        .WORD   ASMDIR
```

```
;
        .WORD   130156          ; .IFDF
        .WORD   014760
        .WORD   IFDF
        .WORD   ASMDIR
;
        .WORD   130044          ; .GLOBL
        .WORD   057034
        .WORD   GLOBL
        .WORD   ASMDIR
;
        .WORD   127736          ; .EVEN
        .WORD   020560
        .WORD   EVEN
        .WORD   ASMDIR
;
        .WORD   127727          ; .EOT
        .WORD   076400
        .WORD   EOT
        .WORD   ASMDIR
;
        .WORD   127726          ; .ENDC
        .WORD   014570
        .WORD   ENDC
        .WORD   ASMDIR
;
        .WORD   127726          ; .END
        .WORD   014400
        .WORD   END
        .WORD   ASMDIR
;
        .WORD   127613          ; .CSECT
        .WORD   017714
        .WORD   CSECT
        .WORD   ASMDIR
;
        .WORD   127551          ; .BYTE
        .WORD   076710
        .WORD   BYTE
        .WORD   ASMDIR
;
        .WORD   127473          ; .ASECT
        .WORD   017714
        .WORD   ASECT
        .WORD   ASMDIR
;
        .WORD   127473          ; .ASCII
        .WORD   012061
        .WORD   ASCII
        .WORD   ASMDIR
;
;
;
        .WORD   107761          ;WAIT
        .WORD   076400
        .WORD   000001
        .WORD   SCLAS0
;
        .WORD   100014          ;TST
        .WORD   0
        .WORD   005700
        .WORD   SCLAS1+BYTFLG
```

```
;
        .WORD   077721          ;TRAP
        .WORD   062000
        .WORD   104400
        .WORD   SCLAS6
;
        .WORD   075131          ;SWAB
        .WORD   006200
        .WORD   000300
        .WORD   SCLAS1
;
        .WORD   075012          ;SUB
        .WORD   0
        .WORD   160000
        .WORD   SCLAS2
;
        .WORD   073642          ;SEZ
        .WORD   0
        .WORD   000264
        .WORD   SCLAS0
;
        .WORD   073636          ;SEV
        .WORD   0
        .WORD   000262
        .WORD   SCLAS0
;
        .WORD   073626          ;SEN
        .WORD   0
        .WORD   000270
        .WORD   SCLAS0
;
        .WORD   073613          ;SEC
        .WORD   0
        .WORD   000261
        .WORD   SCLAS0
;
        .WORD   073473          ;SCC
        .WORD   0
        .WORD   000277
        .WORD   SCLAS0
;
        .WORD   073423          ;SBC
        .WORD   0
        .WORD   005600
        .WORD   SCLAS1+BYTFLG
;
        .WORD   071663          ;RTS
        .WORD   0
        .WORD   000200
        .WORD   SCLAS3
;
        .WORD   071651          ;RTI
        .WORD   0
        .WORD   000002
        .WORD   SCLAS0
;
        .WORD   071352          ;ROR
        .WORD   0
        .WORD   006000
        .WORD   SCLAS1+BYTFLG
```

```
;
        .WORD   071344          ;ROL
        .WORD   0
        .WORD   006100
        .WORD   SCLAS1+BYTFLG
;
        .WORD   070533          ;RESET
        .WORD   021140
        .WORD   000005
        .WORD   SCLAS0
;
        .WORD   054750          ;NOP
        .WORD   0
        .WORD   000240
        .WORD   SCLAS0
;
        .WORD   054117          ;NEG
        .WORD   0
        .WORD   005400
        .WORD   SCLAS1+BYTFLG
;
        .WORD   051656          ;MOV
        .WORD   0
        .WORD   010000
        .WORD   SCLAS2+BYTFLG
;
        .WORD   040612          ;JSR
        .WORD   0
        .WORD   004000
        .WORD   SCLAS5
;
        .WORD   040230          ;JMP
        .WORD   0
        .WORD   000100
        .WORD   SCLAS1
;
        .WORD   035254          ;IOT
        .WORD   0
        .WORD   000004
        .WORD   SCLAS0
;
        .WORD   035163          ;INC
        .WORD   0
        .WORD   005200
        .WORD   SCLAS1+BYTFLG
;
        .WORD   031064          ;HALT
        .WORD   076400
        .WORD   0
        .WORD   SCLAS0
;
        .WORD   020534          ;EMT
        .WORD   0
        .WORD   104000
        .WORD   SCLAS6
;
        .WORD   014713          ;DEC
        .WORD   0
        .WORD   005300
        .WORD   SCLAS1+BYTFLG
```

```
;
        .WORD   012445          ;COM
        .WORD   0
        .WORD   005100
        .WORD   SCLAS1+BYTFLG
;
        .WORD   012412          ;CNZ
        .WORD   0
        .WORD   000254
        .WORD   SCLAS0
;
        .WORD   012330          ;CMP
        .WORD   0
        .WORD   020000
        .WORD   SCLAS2+BYTFLG
;
        .WORD   012272          ;CLZ
        .WORD   0
        .WORD   000244
        .WORD   SCLAS0
;
        .WORD   012266          ;CLV
        .WORD   0
        .WORD   000242
        .WORD   SCLAS0
;
        .WORD   012262          ;CLR
        .WORD   0
        .WORD   005000
        .WORD   SCLAS1+BYTFLG
;
        .WORD   012256          ;CLN
        .WORD   0
        .WORD   000250
        .WORD   SCLAS0
;
        .WORD   012243          ;CLC
        .WORD   0
        .WORD   000241
        .WORD   SCLAS0
;
        .WORD   011473          ;CCC
        .WORD   0
        .WORD   000257
        .WORD   SCLAS0
;
        .WORD   010003          ;BVS
        .WORD   0
        .WORD   102400
        .WORD   SCLAS4
;
        .WORD   007763          ;BVC
        .WORD   0
        .WORD   102000
        .WORD   SCLAS4
;
        .WORD   007520          ;BR
        .WORD   0
        .WORD   000400
        .WORD   SCLAS4
```

```
;
        .WORD   007414          ;BPL
        .WORD   0
        .WORD   100000
        .WORD   SCLAS4
;
        .WORD   007265          ;BNE
        .WORD   0
        .WORD   001000
        .WORD   SCLAS4
;
        .WORD   007221          ;BMI
        .WORD   0
        .WORD   100400
        .WORD   SCLAS4
;
        .WORD   007164          ;BLT
        .WORD   0
        .WORD   002400
        .WORD   SCLAS4
;
        .WORD   007157          ;BLOS
        .WORD   073300
        .WORD   101400
        .WORD   SCLAS4
;
        .WORD   007157          ;BLO
        .WORD   0
        .WORD   103400
        .WORD   SCLAS4
;
        .WORD   007145          ;BLE
        .WORD   0
        .WORD   003400
        .WORD   SCLAS4
;
        .WORD   006774          ;BIT
        .WORD   0
        .WORD   030000
        .WORD   SCLAS2+BYTFLG
;
        .WORD   006773          ;BIS
        .WORD   0
        .WORD   050000
        .WORD   SCLAS2+BYTFLG
;
        .WORD   006753          ;BIC
        .WORD   0
        .WORD   040000
        .WORD   SCLAS2+BYTFLG
;
        .WORD   006711          ;BHIS
        .WORD   073300
        .WORD   103000
        .WORD   SCLAS4
;
        .WORD   006711          ;BHI
        .WORD   0
        .WORD   101000
        .WORD   SCLAS4
```

```
;
        .WORD   006654          ;BGT
        .WORD   0
        .WORD   003000
        .WORD   SCLAS4
;
        .WORD   006635          ;BGE
        .WORD   0
        .WORD   002000
        .WORD   SCLAS4
;
        .WORD   006531          ;BEQ
        .WORD   0
        .WORD   001400
        .WORD   SCLAS4
;
        .WORD   006413          ;BCS
        .WORD   0
        .WORD   103400
        .WORD   SCLAS4
;
        .WORD   006373          ;BCC
        .WORD   0
        .WORD   103000
        .WORD   SCLAS4
;
        .WORD   004512          ;ASR
        .WORD   0
        .WORD   006200
        .WORD   SCLAS1+BYTFLG
;
        .WORD   004504          ;ASL
        .WORD   0
        .WORD   006300
        .WORD   SCLAS1+BYTFLG
;
        .WORD   003344          ;ADD
        .WORD   0
        .WORD   060000
        .WORD   SCLAS2
;
        .WORD   003343          ;ADC
        .WORD   0
        .WORD   005500
        .WORD   SCLAS1+BYTFLG
;
;
;
SYMTBB=.-2                      ;HIGHEST ADDRESS OF PST.

        .END
```

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

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

PDP-11 PAL-11R Assembler
Programmer's Manual
DEC-11-ASDB-D
May 1971

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback -- your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability and readability.

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

How can this manual be improved?

Other comments?

Please state your position. _____ Date: _____

Name: _____ Organization: _____

Street: _____ Department: _____

City: _____ State: _____ Zip or Country _____

— — — — — — — — — — — — — — — — Fold Here — — — — — — — — — — — — — — — — — —

— — — — — — — — — — — — Do Not Tear - Fold Here and Staple — — — — — — — — — — —

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

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