
Order Number AA–KX10A–TC

May 1988
This manual describes how to use the MACRO–11 relocatable assembler to develop PDP–11 assembly language programs. Although no prior knowledge of MACRO–11 is required, you should be familiar with the PDP–11 processor addressing modes and instruction set. This manual presents detailed descriptions of MACRO–11's features, including source and command string control of assembly and listing functions, directives for conditional assembly and program sectioning, and user-defined and system macro libraries. The chapters on operating procedures were previously found in two separate manuals (the Macro–11 Language Reference Manual and the IAS/RSX Macro–11 Reference Manual). This manual should be used with a system-specific user's guide as well as a Linker or a Task Builder manual.


This manual contains Update Notice 1, AD–KX10A–T1.

Operating Systems:
IAS
MICRO/RSX
MICRO/RSTS
RSTS/E
RSX–11M
RSX–11M–PLUS
RT–11
P/OS
VAX/VMS

Software:
MACRO–11 Version 5.5

digital equipment corporation
maynard, massachusetts
The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software, if any, described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license. No responsibility is assumed for the use or reliability of software or equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.


All Rights Reserved.
Printed in U.S.A.

The READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

- CTS-300
- MASSBUS
- RSTS
- DEC
- MicroPDP-11
- RSX
- DECmate
- Micro/RSX
- RT-11
- DECnet
- MicroVMS
- RTEM-11
- DECSYSTEM-10
- PDP
- UNIBUS
- DEC
- DECsystem-20
- VAX
- DECUS
- Professional
- VMS
- DECreter
- Q-bus
- VT
- DIBOL
- Rainbow
- Work Processor

ML-S963
## Contents

**Preface**

**Part I**

**Chapter 1** The MACRO–11 Assembler

1.1 Assembly Pass 1 ........................................ 1–1
1.2 Assembly Pass 2 ........................................ 1–2

**Chapter 2** Source Program Format

2.1 Programming Standards and Conventions .................. 2–1
2.2 Statement Format ........................................ 2–1
2.2.1 Label Field .......................................... 2–2
2.2.2 Operator Field ........................................ 2–3
2.2.3 Operand Field ........................................ 2–4
2.2.4 Comment Field ....................................... 2–4
2.3 Format Control .......................................... 2–5

**Part II**

**Chapter 3** Symbols and Expressions

3.1 Character Set ........................................... 3–1
3.1.1 Separating and Delimiting Characters ................ 3–2
3.1.2 Invalid Characters ................................... 3–3
3.1.3 Unary and Binary Operators ......................... 3–3
3.2 MACRO–11 Symbols ..................................... 3–5
3.2.1 Permanent Symbols .................................. 3–5
3.2.2 User-Defined and Macro Symbols .................. 3–5
3.3 Direct Assignment Statements ............................ 3–7
3.4 Register Symbols ....................................... 3–8
3.5 Local Symbols .......................................... 3–10
3.6 Current Location Counter ................................ 3–11
3.7 Numbers ................................................ 3–13
3.8 Terms .................................................... 3–14
Chapter 4  Relocation and Linking

Chapter 5  Addressing Modes

5.1 Register Mode ............................................. 5-3
5.2 Register Deferred Mode ................................. 5-3
5.3 Autoincrement Mode ...................................... 5-4
5.4 Autoincrement Deferred Mode ......................... 5-4
5.5 Autodecrement Mode ..................................... 5-4
5.6 Autodecrement Deferred Mode ......................... 5-4
5.7 Index Mode .................................................. 5-5
5.8 Index Deferred Mode ..................................... 5-5
5.9 Immediate Mode .......................................... 5-6
5.10 Absolute Mode ............................................ 5-6
5.11 Relative Mode ............................................ 5-7
5.12 Relative Deferred Mode ............................... 5-8
5.13 Branch Instruction Addressing ...................... 5-8
5.14 Using TRAP Instructions .............................. 5-9

Part III

Chapter 6  General Assembler Directives

6.1 Listing Control Directives .............................. 6-3
6.1.1 .LIST And .NLIST Directives ...................... 6-6
6.1.2 .TITLE Directive ...................................... 6-10
6.1.3 .SBITL Directive ...................................... 6-11
6.1.4 .IDENT Directive ..................................... 6-12
6.1.5 .PAGE Directive/Page Ejection .................... 6-12
6.1.6 .REM Directive/Begin Remark Lines ............. 6-13
6.2 Function Directives ...................................... 6-13
6.2.1 .ENABL and .DSABL Directives ................... 6-14
6.2.2 Cross-Reference Directives: .CROSS and .NOCROSS 6-16
6.3 Data Storage Directives ............................... 6-17
6.3.1 .BYTE Directive ....................................... 6-17
6.3.2 .WORD Directive ...................................... 6-18
6.3.3 ASCII Conversion Characters ...................... 6-19
6.3.4 .ASCII Directive ..................................... 6-20
6.3.5 .ASCIZ Directive ..................................... 6-21
6.3.6 .RAD50 Directive .................................... 6-22
6.3.7 Temporary Radix-50 Control Operator ............ 6-24
6.3.8 .PACKED Directive .................................................. 6–24
6.4 Radix and Numeric Control Facilities .................................. 6–26
6.4.1 Radix Control and Unary Control Operators ..................... 6–26
6.4.1.1 .RADIX Directive .............................................. 6–26
6.4.1.2 Temporary Radix Control Operators ......................... 6–27
6.4.2 Numeric Directives and Unary Control Operators ............... 6–28
6.4.2.1 One's Complement Operator: 'C ................................ 6–29
6.4.2.2 Floating-Point Storage Directives ............................ 6–30
6.4.2.3 Floating-Point Operator: 'F ................................. 6–30
6.5 Location Counter Control Directives .................................. 6–31
6.5.1 .EVEN Directive .................................................... 6–31
6.5.2 .ODD Directive ..................................................... 6–32
6.5.3 .BLKB and .BLKW Directives .............................. 6–32
6.5.4 .LIMIT Directive .................................................. 6–33
6.7 Program Sectioning Directives ........................................ 6–34
6.7.1 .PSECT Directive .................................................. 6–35
6.7.1.1 Creating Program Sections .................................. 6–38
6.7.1.2 Code or Data Sharing ......................................... 6–40
6.7.1.3 Memory Allocation Considerations ...................... 6–40
6.7.2 .ASECT and .CSECT Directives .............................. 6–40
6.7.3 .SAVE Directive .................................................. 6–41
6.7.4 .RESTORE Directive ............................................ 6–41
6.8 Symbol Control Directives ............................................. 6–43
6.8.1 .GLOBL Directive ............................................... 6–43
6.8.2 .WEAK Directive ................................................. 6–44
6.9 Conditional Assembly Directives ..................................... 6–45
6.9.1 Conditional Assembly Block Directives .................... 6–45
6.9.2 Subconditional Assembly Block Directives ................ 6–48
6.9.3 Immediate Conditional Assembly Directive ............. 6–50
6.10 File Control Directives ................................................. 6–51
6.10.1 .LIBRARY Directive ............................................ 6–51
6.10.2 .INCLUDE Directive ............................................ 6–52

**Chapter 7  Macro Directives**

7.1 Defining Macros .............................................................. 7–2
7.1.1 .MACRO Directive .................................................. 7–2
7.1.2 . ENDM Directive .................................................. 7–3
7.1.3 . MEXIT Directive .................................................. 7–4
7.1.4 MACRO Definition Formatting .................................. 7–4
7.2 Calling Macros ................................................................. 7–5
7.3 Arguments in Macro Definitions and Macro Calls ............... 7–5
7.3.1 Macro Nesting .......................................................... 7–7
7.3.2 Special Characters in Macro Arguments ..................... 7–8
7.3.3 Passing Numeric Arguments as Symbols ....................................................... 7–8
7.3.4 Number of Arguments in Macro Calls ......................................................... 7–9
7.3.5 Creating Local Symbols Automatically ......................................................... 7–9
7.3.6 Keyword Arguments .................................................................................... 7–11
7.3.7 Concatenation of Macro Arguments ............................................................. 7–12
7.4 Macro Attribute Directives: .NARG, .NCHR, and .NTYPE ................................. 7–13
7.4.1 .NARG Directive ......................................................................................... 7–13
7.4.2 .NCHR Directive ....................................................................................... 7–15
7.4.3 .NTYPE Directive ..................................................................................... 7–16
7.5 .ERROR and .PRINT Directives ..................................................................... 7–18
7.6 Indefinite Repeat Block Directives: .IRP and .IRPC .......................................... 7–19
7.6.1 .IRP Directive ......................................................................................... 7–19
7.6.2 .IRPC Directive ....................................................................................... 7–20
7.7 Repeat Block Directive: .REPT, .ENDR ........................................................... 7–21
7.8 Macro Library Directive: .MCALL .................................................................. 7–22
7.9 Macro Deletion Directive: .MDELETE ............................................................. 7–23

Part IV

Chapter 8 IAS/RSX–11M/RSX–11M–PLUS Operating Procedures

8.1 RSX–11M/RSX–11M–PLUS Operating Procedures .............................................. 8–1
8.1.1.1 Direct MACRO–11 Call ........................................................................ 8–2
8.1.1.2 Single Assembly .................................................................................. 8–2
8.1.1.3 Install, Run Immediately, and Remove on Exit ..................................... 8–3
8.1.1.4 Indirect Command Processor ............................................................... 8–3
8.1.2 Default RSX–11 File Specifications ............................................................ 8–4
8.1.3 MCR Command String Format .................................................................... 8–4
8.1.4 DCL Operating Procedures ....................................................................... 8–7
8.1.5 MACRO–11 Command String Examples .................................................... 8–11
8.2 IAS MACRO–11 Operating Procedures .......................................................... 8–11
8.2.1 Running MACRO–11 Under IAS ............................................................... 8–11
8.2.2 IAS Command String ............................................................................. 8–12
8.2.3 IAS Indirect Command Files .................................................................... 8–13
8.2.4 IAS Command String Examples ............................................................... 8–14
8.3 Cross-Reference Processor (CREF) ................................................................. 8–14
Appendix J  Release Notes

J.1 Changes—All Versions of MACRO-11  J-1
J.1.1 V5.5 Update Changes  J-1
J.1.2 V5.4 Update Changes  J-2
J.1.3 V5.3 Update Changes  J-2
J.1.4 V5.2 Update Changes  J-3
J.1.5 V5.1 Update Changes  J-3
J.1.6 V5.0 Update Changes  J-4
J.2.1 V5.5 Update Changes  J-5
J.2.2 V5.4 Update Changes  J-5
J.2.3 V5.3 Update Changes  J-6
J.2.4 V5.2 Update Changes  J-6
J.2.5 V5.1 Update Changes  J-6
J.2.6 V5.0 Update Changes  J-6
J.3 Changes—MACRO-11/RT-11 Version Only  J-6
J.3.1 V5.5 Update Changes  J-6
J.3.2 V5.4 Update Changes  J-7
J.3.3 V5.3 Update Changes  J-7
J.3.4 V5.2 Update Changes  J-7
J.3.5 V5.1 Update Changes  J-7
J.3.6 V5.0 Update Changes  J-7

Index

Figures

3–1 Assembly Listing Showing Local Symbol Block  3–11
6–1 Example of Line Printer Assembly Listing  6–4
6–2 Example of Terminal Assembly Listing  6–5
6–3 Listing Produced with Listing Control Directives  6–9
6–4 Assembly Listing Table of Contents  6–11
6–5 Example of .ENABL and .DSABL Directives  6–16
6–6 Example of the .PACKED Directive  6–25
6–7 Example of .BLKB and .BLKW Directives  6–33
6–8 Example of .SAVE and .RESTORE Directives  6–42
7–1 Example of .NARG Directive  7–14
7–2 Example of .NCHR Directive  7–16
7–3 Example of .NTYPE Directive in Macro Definition  7–17
7–4 Example of .IRP and .IRPC Directives  7–21
8–1 Sample IAS CREF Listing  8–16
A–1 DEC Multinational Character Set  A–10
G–1 Example of Position-Dependent Code  G–3
G–2 Example of Position-Independent Code  G–4
<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Special Characters Used in MACRO–11</td>
<td>3-1</td>
</tr>
<tr>
<td>3-2</td>
<td>Valid Separating Characters</td>
<td>3-3</td>
</tr>
<tr>
<td>3-3</td>
<td>Valid Argument Delimiters</td>
<td>3-3</td>
</tr>
<tr>
<td>3-4</td>
<td>Valid Unary Operators</td>
<td>3-4</td>
</tr>
<tr>
<td>3-5</td>
<td>Valid Binary Operators</td>
<td>3-4</td>
</tr>
<tr>
<td>5-1</td>
<td>Symbols Used in Chapter 5</td>
<td>5-1</td>
</tr>
<tr>
<td>5-2</td>
<td>Addressing Modes</td>
<td>5-2</td>
</tr>
<tr>
<td>5-3</td>
<td>Instruction Differences Among PDP–11 Processors</td>
<td>5-3</td>
</tr>
<tr>
<td>6-1</td>
<td>Directives in Chapter 6</td>
<td>6-1</td>
</tr>
<tr>
<td>6-2</td>
<td>Symbolic Arguments of Listing Control Directives</td>
<td>6-7</td>
</tr>
<tr>
<td>6-3</td>
<td>Symbolic Arguments of Function Control Directives</td>
<td>6-14</td>
</tr>
<tr>
<td>6-4</td>
<td>Symbolic Arguments of .PSECT Directive</td>
<td>6-35</td>
</tr>
<tr>
<td>6-5</td>
<td>Program Section Default Values</td>
<td>6-41</td>
</tr>
<tr>
<td>6-6</td>
<td>Valid Condition Tests for Conditional Assembly Directives</td>
<td>6-46</td>
</tr>
<tr>
<td>6-7</td>
<td>Subconditional Assembly Block Directives</td>
<td>6-48</td>
</tr>
<tr>
<td>7-1</td>
<td>Directives in Chapter 7</td>
<td>7-1</td>
</tr>
<tr>
<td>8-1</td>
<td>RSX–11 File Specification Default Values</td>
<td>8-4</td>
</tr>
<tr>
<td>8-2</td>
<td>RSX–11 File Specification Switches for MACRO–11</td>
<td>8-6</td>
</tr>
<tr>
<td>8-3</td>
<td>RSX–11 DCL Command Qualifiers</td>
<td>8-7</td>
</tr>
<tr>
<td>8-4</td>
<td>RSX–11 DCL Parameter Qualifier</td>
<td>8-10</td>
</tr>
<tr>
<td>9-2</td>
<td>File Specification Options</td>
<td>9-4</td>
</tr>
<tr>
<td>9-3</td>
<td>/C Option Arguments</td>
<td>9-7</td>
</tr>
<tr>
<td>A-1</td>
<td>DEC Multinational Character Set</td>
<td>A-1</td>
</tr>
<tr>
<td>A-2</td>
<td>Radix–50 Character Set</td>
<td>A-8</td>
</tr>
<tr>
<td>A-3</td>
<td>Radix–50 Character Equivalents</td>
<td>A-8</td>
</tr>
<tr>
<td>I-1</td>
<td>Old and New Directives and Syntax</td>
<td>I-1</td>
</tr>
</tbody>
</table>
Manual Objectives and Reader Assumptions

This manual is intended to enable you to write programs in the MACRO-11 assembly language.

No prior knowledge of the MACRO-11 Relocatable Assembler is assumed, but you should be familiar with PDP-11 processors and related terminology, as presented in the PDP-11 Processor Handbook. You are also encouraged to become familiar with the linking process, as presented in the applicable system manual (see the Associated Documents section below), because linking is necessary for the development of executable programs.

If a terminal is available, we suggest that you try some of the examples in the manual or write a few simple programs that illustrate the concepts covered. Even experienced programmers find that working with a simple program helps them to understand a confusing feature of a new language.

The examples in this manual were done on an RT-11 system. You can also use MACRO-11 on IAS, RSX-11M, RSX-11M-PLUS and RSTS systems (see Part IV for information about operating procedures).

All references to RSX-11M also apply to RSX-11M-PLUS with the exception of those in Chapter 8, which deals with each system individually.

Document Structure

This manual has four parts and eight appendixes.

Part I introduces MACRO-11:

- Chapter 1 lists the key features of MACRO-11.
- Chapter 2 discusses the advantages of following programming standards and conventions and describes the format used in coding MACRO-11 source programs.

Part II presents general information essential to programming with the MACRO-11 assembly language:

- Chapter 3 lists the character set and describes the symbols, terms, and expressions that form the elements of MACRO-11 instructions.
- Chapter 4 describes MACRO-11 output and presents concepts essential to the proper relocation and linking of object modules.
- Chapter 5 describes how data stored in memory can be accessed and manipulated by using the addressing modes recognized by the PDP-11 hardware.
Part III describes the MACRO-11 directives that control the processing of source statements during assembly:

- Chapter 6 discusses directives used for generalized MACRO-11 functions.
- Chapter 7 discusses directives used in the definition and expansion of macros.

Part IV presents the operating procedures for assembling MACRO-11 programs:

- Chapter 8 covers the IAS, RSX-11M, and RSX-11M-PLUS systems.
- Chapter 9 covers the RSTS/RT-11 systems.

Appendix A lists the ASCII and Radix-50 character sets used in MACRO-11 programs.

Appendix B lists the special characters recognized by MACRO-11, summarizes the syntax of the various addressing modes used in PDP-11 processors, and briefly describes the MACRO-11 directives in alphabetical order.

Appendix C lists alphabetically the permanent symbols that have been defined for use with MACRO-11.

Appendix D lists alphabetically the error codes produced by MACRO-11 to identify various types of errors detected during the assembly process.

Appendix E contains a coding standard that is recommended practice in preparing MACRO-11 programs.

Appendix F discusses several methods of conserving dynamic memory space for users of small systems who may experience difficulty in assembling MACRO-11 programs.

Appendix G is a discussion of position-independent code (PIC).

Appendix H contains an assembly and cross-reference listing.

Appendix I contains obsolete MACRO-11 directives, syntax, and command line options.

Appendix J describes the differences from the last release of MACRO-11.

**Associated Documents**

For descriptions of documents associated with this manual, refer to the applicable documentation directory listed below:

- IAS Documentation Directory
- RSX-11M-PLUS Information Directory and Master Index
- RSX-11M/RSX-11S Information Directory and Index
- Guide to RT-11 Documentation
- RSTS/E Documentation Directory

**Conventions**

- The color red is used in command string examples to indicate user input.
- The term **printing characters** includes all characters that display or print a symbol.
The term **nonprinting characters** includes all characters other than those defined as printing characters. It includes space, horizontal and vertical tab, carriage return, line feed, and form feed, even though those characters cause cursor or print head movement.

The symbols defined below are used throughout this manual.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Brackets indicate that the enclosed argument is optional.</td>
</tr>
<tr>
<td>...</td>
<td>Ellipsis indicates optional continuation of an argument list in the form of the last specified argument.</td>
</tr>
<tr>
<td>UPPERCASE CHARACTERS</td>
<td>Uppercase characters indicate elements of the language that must be used exactly as shown.</td>
</tr>
<tr>
<td>lowercase characters</td>
<td>Lowercase characters indicate elements of the language that are supplied by the programmer.</td>
</tr>
<tr>
<td>Subscripts</td>
<td>Subscripts indicate the radix of a number. For example, 100₈ indicates 100, base 8.</td>
</tr>
<tr>
<td>(base)</td>
<td>The symbol (base) indicates the radix of numbers in code examples. For example, 100(octal) indicates that 100 is an octal value, while 100(decimal) indicates a decimal value.</td>
</tr>
<tr>
<td>CTRL/x or `x</td>
<td><strong>CTRL/x</strong> signifies a control character, generated by simultaneously pressing the CTRL key and the x key.</td>
</tr>
</tbody>
</table>

CTRL characters are sometimes represented in command line examples by `x; do not confuse this representation of CTRL characters with MACRO-11 unary operators such as `B, `D, `O, and `R (see Section 6.4.1.2).
Part I
Chapter 1

The MACRO–11 Assembler

MACRO–11 provides the following features:

- Source and command string control of assembly functions
- Device and filename specifications for input and output files
- Error listing on command output device
- Alphabetized, formatted symbol table listing; optional cross-reference listing of symbols
- Relocatable object modules
- Global symbols for linking object modules
- Conditional assembly directives
- Program sectioning directives
- User-defined macros and macro libraries
- Comprehensive system macro library
- Extensive source and command string control of listing functions

MACRO–11 assembles one or more source files containing MACRO–11 statements into a single relocatable binary object file. The output of MACRO–11 consists of a binary object file and a listing file containing the table of contents, the assembly listing, and the symbol table. An optional cross-reference listing of symbols and macros is available. A sample assembly listing is provided in Appendix H.

1.1 Assembly Pass 1

During pass 1, MACRO–11 locates and reads all required macros from libraries, builds symbol tables and program section tables for the program, and performs a rudimentary assembly of each source statement.

In the first step of assembly pass 1, MACRO–11 initializes all the impure areas (areas containing data) that will be used internally for the assembly process. These areas include all dynamic storage and buffer areas used as file storage regions. MACRO–11 then calls a system subroutine which transfers a command line into memory. This command line contains the specifications of all files to be used during assembly. After scanning the command line for proper syntax, MACRO–11 opens the specified output files. These files are opened to determine if valid output file specifications have been passed in the command line.
MACRO-11 then initiates a routine which reads source lines from the input file. If no input file is open, as is the case at the beginning of assembly, MACRO-11 opens the next input file specified in the command line and starts assembling the source statements. MACRO-11 first determines the length of the instructions, then assembles them according to length as one word, two words, or three words.

At the end of assembly pass 1, MACRO-11 reopens the output files described above. Such information as the object module name, the program version number, and the global symbol directory (GSD) for each program section are written to the object file to be used later in linking the object modules. After writing out the GSD for a given program section, MACRO-11 scans through the symbol tables to find all the global symbols that are bound to that particular program section. MACRO-11 then writes out GSD records to the object file for these symbols. This process is done for each program section.

1.2 Assembly Pass 2

On pass 2 MACRO-11 writes the object records to the binary output file. MACRO-11 also generates the assembly listing and the symbol table listing for the program, plus a cross-reference table if one was requested.

Basically, assembly pass 2 consists of the same steps performed in assembly pass 1, except that all source statements containing MACRO-11-detected errors are flagged with an error code as the assembly listing file is created. The object file that is created as the final consequence of pass 2 contains all the object records, together with relocation records that hold the information necessary for linking the object file.

The information in the object file, when passed to the Task Builder or Linker, enables the global symbols in the object modules to be associated with absolute or virtual memory addresses, thereby forming an executable body of code.

You may want to become familiar with the macro object file format and description, although you do not need to know the format to use MACRO-11 successfully. This information is presented in the applicable system manual (see the Associated Documents section in the Preface).
Chapter 2
Source Program Format

2.1 Programming Standards and Conventions

Programming standards and conventions allow code written by a person (or group) to be easily understood by another person or group. These standards also make the program easier to:

- Plan
- Comprehend
- Test
- Modify
- Convert

The actual standard used must meet local user requirements. A sample coding standard is provided in Appendix E. Used by DIGITAL and its users, this coding example simplifies both communications and the continuing task of software maintenance and improvement.

2.2 Statement Format

A source program is composed of assembly-language statements. Each statement must be completed on one line. Although a line can contain 132 characters (a longer line causes an error (L) in the assembly listing), a line of 80 characters is recommended because of constraints imposed by listing format and terminal line size. Blank lines, although valid, have no significance in the source program.

A MACRO-11 statement may have as many as four fields. These fields are identified by their order within the statement and/or by the separating characters between the fields. The general format of a MACRO-11 statement is:

```
Label:  Operator Operand ;Comment(s)
```

All the fields are optional, although the operator and operand fields are interdependent; when both operator and operand fields are present in a source statement, each field is evaluated by MACRO-11 in the context of the other.

A statement can contain an operator and no operand, but the reverse is not true. A statement containing an operand with no operator is invalid and is interpreted by MACRO-11 during assembly as an implicit .WORD directive (see Section 6.3.2).

MACRO-11 interprets and processes source program statements one by one. Each statement causes MACRO-11 either to perform a specified assembly process or to generate one or more binary instructions or data words.
2.2.1 Label Field

A label is a user-defined symbol which is assigned the value of the current location counter and entered into the user-defined symbol table. The current location counter is used by MACRO-11 to assign memory addresses to the source program statements as they are encountered during the assembly process. Thus, a label is a means of symbolically referring to a specific statement.

When a program section is absolute, the value of the current location counter is absolute; its value references an absolute virtual memory address, such as location 1100\_8. Similarly, when a program section is relocatable, the value of the current location counter is relocatable; a relocation bias calculated at link time is added to the apparent value of the current location counter to establish its effective absolute virtual address at execution time. (For a discussion of program sections and their attributes, see Section 6.7.)

If present, a label must be the first field in a source statement and must be terminated by a colon (\:). For example, if the value of the current location counter is absolute 1100\_8, the statement:

\texttt{ABCD: MOV A,B}

assigns the value 1100\_8 to the label \texttt{ABCD}. If the location counter value were relocatable, the final value of \texttt{ABCD} would be 1100\_8+K, where K represents the relocation bias of the program section, as calculated by the Task Builder or Linker at link time.

You can assign multiple labels to the same location by putting them on successive lines. For example, the statements:

\texttt{ABC:
$DDD:
A7.7: MOV A,B}

assign the same value to all three labels. This method of assigning multiple labels is preferred, because positioning the fields consistently within the source program makes the program easier to read (see Section 2.3).

More than one label can appear also within a single label field. Each label so specified is assigned the same address value. For example, if the value of the current location counter is 1100\_8, the multiple labels in the following statement are each assigned the value 1100\_8:

\texttt{ABC: $DDD: A7.7: MOV A,B}

However, this method of assigning multiple labels to the same location is more difficult to read and is not recommended.

A double colon (\::) defines the label as a global symbol. For example, the statement:

\texttt{ABCD:: MOV A,B}

establishes the label \texttt{ABCD} as a global symbol. A global symbol can be referenced from an object module other than the module in which the global symbol is defined (see Section 6.8). References from other modules to a global symbol are resolved when the modules are linked as a composite executable image.
The valid characters for defining labels are:

- A through Z
- 0 through 9
- Period (.)
- Dollar Sign ($)

**NOTE**

By convention, the dollar sign ($) and period (.) are reserved for use in defining DIGITAL system software symbols. Therefore these characters should not be used in defining labels in MACRO-11 source programs.

A label can be any length; however, only the first six characters are significant and, therefore, must be unique among all the labels in the source program. An error code (M) is generated in the assembly listing if the first six characters in two or more labels are the same.

A symbol used as a label must not be redefined within the source program. If the symbol is redefined, a label with a multiple definition results, causing MACRO-11 to generate an error code (D) in the assembly listing. Furthermore, any statement in the source program which references a multi-defined label generates an error code (D) in the assembly listing.

### 2.2.2 Operator Field

The operator field specifies the action to be performed. It can consist of an instruction mnemonic (op code), an assembler directive, or a macro call. Chapters 6 and 7 describe these three types of operators.

When the operator is an instruction mnemonic, a machine instruction is generated and MACRO-11 evaluates the addresses of the operands which follow. When the operator is a directive, MACRO-11 performs certain control actions or processing operations during the assembly of the source program. When the operator is a macro call, MACRO-11 inserts the code generated by the macro expansion.

Leading and trailing spaces or tabs in the operator field have no significance; such characters serve only to separate the operator field from the preceding and following fields.

An operator is terminated by a tab, space, or any non-Radix-50 character,¹ as in the following examples:

```
MOV AQ,B ;The tab terminates the operator MOV.
MOV QA,B ;The space terminates the operator MOV.
MOVQA,B ;The @ character terminates the operator MOV.
```

¹ Section A.2 contains a table of Radix-50 characters.
Although the statements above are all equivalent in function, the first statement is the recommended form because it is the most readable and conforms to MACRO-11 coding conventions.

2.2.3 Operand Field

When the operator is an instruction mnemonic (op code), the operand field contains program variables that are to be evaluated/manipulated by the operator. The operand field can also supply arguments to MACRO-11 directives and macro calls, as described in Chapters 6 and 7, respectively.

Operands can be expressions or symbols, depending on the operator. Multiple expressions used in the operand field of a MACRO-11 statement must be separated by a comma; multiple symbols similarly used must be delimited by a valid separator (a comma, tab, and/or space). An operand should be preceded by an operator field; if it is not, the statement is treated by MACRO-11 as an implicit .WORD directive (see Section 6.3.2).

When the operand field contains an op code, associated operands are always expressions, as shown in the following statement:

\[ \text{MOV R0,A+2(R1)} \]

On the other hand, when the operand field contains a MACRO-11 directive or a macro call, associated operands are normally symbols, as shown in the following statement. Assume .COMPR is the name of a user-defined macro:

\[ \text{.COMPR ALPHA SYM1,SYM2} \]

Refer to the description of each MACRO-11 directive (Chapter 7) to determine the type and number of operands required in issuing the directive.

The operand field is terminated by a semicolon when the field is followed by a comment. For example, in the following statement:

\[ \text{LABEL: MOV A,B ;Comment field} \]

the tab between MOV and A terminates the operator field and defines the beginning of the operand field, a comma separates the operands A and B, and a semicolon terminates the operand field and defines the beginning of the comment field. When no comment field follows, the operand field is terminated by the end of the source line.

2.2.4 Comment Field

The comment field normally begins in column 33 and extends through the end of the line, although comments can also be entirely separate lines within the program. This field is optional and can contain any 7-bit ASCII or 8-bit DEC Multinational printing characters plus space and horizontal tab. All other characters appearing in the comment field, even special characters reserved for use in MACRO-11, are checked only for ASCII validity and then included in the assembly listing as they appear in the source text.

Comment fields must begin with a semicolon (;). When a lengthy comment extends beyond the end of the source line (column 80), the comment can be continued on the
following line. The continued comment must be preceded by another semicolon. For readability the continued comment can be indented to begin in the same column as the start of the comment on the previous line.

Comments do not affect assembly processing or program execution. However, comments are necessary in source listings for later analysis, debugging, or documentation purposes.

2.3 Format Control

Horizontal formatting of the source program is controlled by the space and tab characters. These characters have no effect on the assembly process unless they are embedded within a symbol, number, or ASCII text string, or unless they are used as the operator field terminator. Thus, space and tab characters can be used to make the source program orderly and readable.

DIGITAL's standard source line format is shown below:

- Label—begins in column 1
- Operator—begins in column 9
- Operands—begin in column 17
- Comments—begin in column 33

These formatting conventions are not mandatory; free-field coding is permissible. However, note the increased readability after formatting in the example below:

```
REGTST: BIT #MASK, VALUE; COMPARES BITS IN OPERANDS.
1 9 17 33 (columns)
REGTST: BIT #MASK, VALUE ; Compares bits in operands.
```

Page formatting and assembly listing considerations are discussed in Chapter 6 in the context of MACRO-11 directives that can be specified to accomplish desired formatting operations. Appendix E contains a sample coding standard.
Chapter 3
Symbols and Expressions

This chapter describes the components of MACRO-11 instructions: the character set, the conventions for constructing symbols, and the use of numbers, operators, terms, and expressions.

3.1 Character Set

The following characters are valid in MACRO-11 source programs:

- The letters A through Z. Both uppercase and lowercase letters are acceptable, although lowercase can be forced to uppercase if desired (see Section 6.2.1, .DSABL LC).

- Characters in the DEC Multinational character set (MCS). Appendix A contains a table showing the MCS. Specific support for the MCS is included with the description of each directive.

- The digits 0 through 9.

- The characters period (.) and dollar sign ($). These characters are reserved for use in Digital Equipment Corporation system program symbols.

- The special characters listed in Table 3–1.

<table>
<thead>
<tr>
<th>Character</th>
<th>Designation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>Colon</td>
<td>Label terminator</td>
</tr>
<tr>
<td>::</td>
<td>Double colon</td>
<td>Label terminator; defines the label as a global label</td>
</tr>
<tr>
<td>=</td>
<td>Equal sign</td>
<td>Direct assignment operator and macro keyword indicator</td>
</tr>
<tr>
<td>==</td>
<td>Double equal sign</td>
<td>Direct assignment operator; defines the symbol as a global symbol</td>
</tr>
<tr>
<td>=:</td>
<td>Equal sign colon</td>
<td>Direct assignment operator; macro keyword indicator; causes error (M) in listing if an attempt is made to change the value of the symbol</td>
</tr>
<tr>
<td>==:</td>
<td>Double equal sign colon</td>
<td>Direct assignment operator; defines the symbol as a global symbol; causes error (M) in listing if an attempt is made to change the value of the symbol</td>
</tr>
<tr>
<td>%</td>
<td>Percent sign</td>
<td>Register term indicator</td>
</tr>
<tr>
<td>Character</td>
<td>Designation</td>
<td>Function</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td><strong>TAB</strong></td>
<td>Horizontal tab</td>
<td>Item or field terminator</td>
</tr>
<tr>
<td><strong>SP</strong></td>
<td>Space</td>
<td>Item or field terminator</td>
</tr>
<tr>
<td>#</td>
<td>Number sign</td>
<td>Immediate expression indicator</td>
</tr>
<tr>
<td>@</td>
<td>At sign</td>
<td>Deferred addressing indicator</td>
</tr>
<tr>
<td>(</td>
<td>Left parenthesis</td>
<td>Initial register indicator</td>
</tr>
<tr>
<td>)</td>
<td>Right parenthesis</td>
<td>Terminal register indicator</td>
</tr>
<tr>
<td>.</td>
<td>Period</td>
<td>Current location counter</td>
</tr>
<tr>
<td>,</td>
<td>Comma</td>
<td>Operand field separator</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon</td>
<td>Comment field indicator</td>
</tr>
<tr>
<td>&lt;</td>
<td>Left angle bracket</td>
<td>Initial argument or expression indicator</td>
</tr>
<tr>
<td>&gt;</td>
<td>Right angle bracket</td>
<td>Terminal argument or expression indicator</td>
</tr>
<tr>
<td>+</td>
<td>Plus sign</td>
<td>Unary plus, arithmetic addition operator, or autoincrement indicator</td>
</tr>
<tr>
<td>-</td>
<td>Minus sign</td>
<td>Unary minus, arithmetic subtraction operator, or autodecrement indicator</td>
</tr>
<tr>
<td>*</td>
<td>Asterisk</td>
<td>Arithmetic multiplication operator</td>
</tr>
<tr>
<td>/</td>
<td>Slash</td>
<td>Arithmetic division operator</td>
</tr>
<tr>
<td>&amp;</td>
<td>Ampersand</td>
<td>Logical AND operator</td>
</tr>
<tr>
<td>!</td>
<td>Exclamation point</td>
<td>Logical inclusive OR operator</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote</td>
<td>Double ASCII character indicator</td>
</tr>
<tr>
<td>'</td>
<td>Single quote</td>
<td>Single ASCII character indicator or concatenation indicator</td>
</tr>
<tr>
<td>^</td>
<td>Circumflex</td>
<td>Universal unary operator or argument indicator</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
<td>Macro call numeric argument indicator</td>
</tr>
</tbody>
</table>

### 3.1.1 Separating and Delimiting Characters

Valid separating characters and valid argument delimiters are defined in Table 3–2 and Table 3–3, respectively.
Table 3-2: Valid Separating Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Definition</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>One or more spaces and/or tabs</td>
<td>A space is a valid separator between instruction fields and between symbolic arguments within the operand field. Spaces within expressions are ignored (see Section 3.9).</td>
</tr>
<tr>
<td>,</td>
<td>Comma</td>
<td>A comma is a valid separator between symbolic arguments within the operand field. Multiple expressions used in the operand field must be separated by a comma.</td>
</tr>
</tbody>
</table>

3.1.2 Invalid Characters

A character is invalid for one of two reasons:

- If a character is not an element of the recognized MACRO-11 character set, it is replaced in the listing by a question mark, and an error code (1) is printed in the assembly listing. The exception to this is an embedded null which, when detected, is ignored.

- If a valid MACRO-11 character is used in a source statement with invalid or questionable syntax, an error code (Q) is printed in the assembly listing.

Table 3-3: Valid Argument Delimiters

<table>
<thead>
<tr>
<th>Character</th>
<th>Definition</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;...&gt;</td>
<td>Paired angle brackets</td>
<td>Paired angle brackets can be used anywhere in a program to enclose an expression for treatment as a single term. Paired angle brackets are also used to enclose a macro argument, particularly when that argument contains separating characters (see Section 7.3).</td>
</tr>
<tr>
<td>^x...x</td>
<td>Circumflex (unary operator) construction, where the circumflex is followed by an argument that is bracketed by any paired printing characters (x).</td>
<td>This construction is equivalent in function to the paired angle brackets described above and is generally used only where the argument itself contains angle brackets.</td>
</tr>
</tbody>
</table>

3.1.3 Unary and Binary Operators

Table 3-4 describes valid MACRO-11 unary operators. Unary operators are used in connection with single terms (arguments or operands) to indicate an action to be performed on that term during assembly. Because a term preceded by a unary operator is considered to contain that operator, a term so specified can be used alone or as an element of an expression.
Table 3–4: Valid Unary Operators

<table>
<thead>
<tr>
<th>Unary Operator</th>
<th>Name</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Plus sign</td>
<td>+A</td>
<td>Ignored; equivalent to the value of A</td>
</tr>
<tr>
<td>-</td>
<td>Minus sign</td>
<td>−A</td>
<td>Produces the negative (two’s complement) value of A</td>
</tr>
<tr>
<td>′C24</td>
<td>Circumflex, universal unary operator; this usage is described in detail in Section 6.4</td>
<td></td>
<td>Produces the one’s complement value of 24(^{h}). Other unary operators using this syntax include 'B, 'D, 'F, 'O, 'R, and 'X.</td>
</tr>
</tbody>
</table>

Unary operators can be used adjacent to each other or in constructions involving multiple terms, as shown below:

- ′D50          Equivalent to − <′D50>  
- ′C′O12        Equivalent to 'C <′O12>  

Although angle brackets are not required, DIGITAL recommends that you use them for clarity.

Table 3–5 describes valid MACRO–11 binary operators. In contrast to unary operators, binary operators specify actions to be performed on multiple items or terms within an expression.

Table 3–5: Valid Binary Operators

<table>
<thead>
<tr>
<th>Binary Operator</th>
<th>Name</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition operator</td>
<td>A+B</td>
<td>Produces two’s complement sum of A and B</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction operator</td>
<td>A−B</td>
<td>Produces two’s complement difference of A and B</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication operator</td>
<td>A*B</td>
<td>Produces two’s complement signed 16-bit product</td>
</tr>
<tr>
<td>/</td>
<td>Division operator</td>
<td>A/B</td>
<td>Produces two’s complement signed 16-bit quotient</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical AND operator</td>
<td>A&amp;B</td>
<td>Performs bitwise logical AND between A and B</td>
</tr>
<tr>
<td>!</td>
<td>Logical inclusive OR operator</td>
<td>A!B</td>
<td>Performs bitwise logical inclusive OR between A and B</td>
</tr>
</tbody>
</table>

All binary operators have equal priority. Terms enclosed by angle brackets are evaluated first, and remaining operations are performed from left to right, as shown in the examples below:
3.2 MACRO–11 Symbols

MACRO–11 maintains a symbol table for each of the three symbol types that may be defined in a MACRO–11 source program: the Permanent Symbol Table, the User Symbol Table, and the Macro Symbol Table. The Permanent Symbol Table contains all the permanent symbols defined within (and thus automatically recognized by) MACRO–11 and is part of the MACRO–11 image. The User Symbol Table (for user-defined symbols) and Macro Symbol Table (for macro symbols) are constructed as the source program is assembled.

3.2.1 Permanent Symbols

Permanent symbols consist of the instruction mnemonics (see Appendix C) and MACRO–11 directives (see Chapters 6 and 7 and Appendix B). These symbols are a permanent part of the MACRO–11 image and need not be defined before being used in the operator field of a MACRO–11 source statement (see Section 2.2.2).

3.2.2 User-Defined and Macro Symbols

User-defined symbols are those symbols that are equated to a specific value through a direct assignment statement (see Section 3.3), appear as labels (see Section 2.2.1), or act as dummy arguments (see Section 7.1.1). These symbols are added to the User Symbol Table as they are encountered during assembly.

Macro symbols are those symbols used as macro names (see Section 7.1). They are added to the Macro Symbol Table as they are encountered during assembly.

The following rules govern the creation of user-defined and macro symbols:

- Symbols can be composed of alphanumeric characters, dollar signs ($), and periods (.) only (see Note below).
- The first character of a symbol must not be a number (except in the case of local symbols; see Section 3.5).
- The first six characters of a symbol must be unique. A symbol can be written with more than six valid characters, but the seventh and subsequent characters are checked only for ASCII validity and are not otherwise evaluated or recognized by MACRO–11.
- Spaces, tabs, and invalid characters must not be embedded within a symbol. The valid MACRO–11 character set is defined in Section 3.1.

**NOTE**

The dollar sign ($) and period (.) characters are reserved for use in defining Digital Equipment Corporation system software symbols. For example, $READ and .READ are file-processing system macros for RSX–11 and RT–11, respectively. DIGITAL suggests that you not use these
characters in constructing user-defined symbols or macro symbols to avoid possible conflicts with existing or future Digital Equipment Corporation system software symbols.

The value of a symbol depends upon its use in the program. A symbol in the operator field can be any one of the three symbol types described above; permanent, user-defined, or macro. To determine the value of an operator-field symbol, MACRO-11 searches the symbol tables in the following order:

1. Macro Symbol Table
2. Permanent Symbol Table
3. User Symbol Table

This search order allows permanent symbols to be used as macro symbols, but you must keep in mind the sequence in which the search for symbols is performed to avoid incorrect interpretation of the symbol's use.

When a symbol appears in the operand field, the search order is:

1. User Symbol Table
2. Permanent Symbol Table

Depending on their use in the source program, user-defined symbols have either a local (internal) attribute or a global (external) attribute.

Normally, MACRO-11 treats all user-defined symbols as local; that is, their definition is limited to the module in which they appear. However, symbols can be explicitly declared to be global symbols through one of three methods:

- Use of the .GLOBL directive (see Section 6.8.1)
- Use of the double colon (::) in defining a label (see Section 2.2.1)
- Use of the double equal sign (==) or double equal colon sign (==:) in a direct assignment statement (see Section 3.3)

All symbols within a module that remain undefined at the end of assembly are treated as default global references, unless you use the .DSABL GBL directive (see Section 6.2.1). If .ENABL GBL is in effect, the undefined symbols are assigned a value of 0 and placed into the User Symbol Table as undefined default global references. If the .DSABL GBL directive is in effect, however, the statement containing the undefined symbol is flagged with an error code (U) in the assembly listing.

Global symbols provide linkages between independently assembled object modules within the task image. For example, a global symbol defined as a label may serve as an entry point address to another section of code within the image. Such symbols are referenced from other source modules in order to transfer control throughout execution. These global symbols are resolved at link time, ensuring that the resulting image is a logically coherent and complete body of code.
3.3 Direct Assignment Statements

The general format for a direct assignment statement is:

```
symbol=expression
```

or:

```
symbol==expression
```

where:

```
expression  can have only one level of forward reference (see list of rules, below) and
cannot contain an undefined global reference.
```

The colon format for a direct assignment statement is:

```
symbol=:expression
```

or:

```
symbol==:expression
```

where:

```
expression  can have only one level of forward reference (see list of rules, below) and
cannot contain an undefined global reference.
```

All the direct assignment statements above allow you to equate a symbol with a specific value. After the symbol has been defined, it is entered into the User Symbol Table. If the general format is used (= or ==) the value of the symbol can be changed in subsequent direct assignment statements. However, if the colon format is used (=: or ==:), any attempt to change the value of the symbol generates an error (M) in the assembly listing.

A direct assignment statement using either the double equal (==) sign or the double equal colon (==:) sign, as shown above, defines the symbol as global (see Section 6.8.1). The following examples illustrate the coding of direct assignment statements.

; Example 1:
```
A=10  ;Direct assignment
B==30 ;Global assignment
A=15  ;Valid reassignment
L=:5  ;Equal colon assignment
M==:A+2 ;Double equal colon assignment
;M becomes equal to 17
L=4   ;Invalid reassignment
;M error is generated
```
Example 2:

C:  D=             ;The symbol D is equated to ., and

E:  MOV #1,ABLE   ;the labels C and E are assigned a
       ;value that is equal to the location
       ;of the MOV instruction. C, D, and E
       ;all have the same value.

The code in Example 2 above would not usually be used and is shown only to illustrate
the performance of MACRO-11 in such situations. See Section 3.6 for a description
of the period (.) as the current location counter symbol.

The following rules apply to the coding of direct assignment statements:

- An equal sign (=), double equal sign (==), equal colon sign (=:), or double
  equal colon sign (==:) must separate the symbol from the expression defining
  the symbol's value. Spaces preceding and/or following the direct assignment operators,
  although permissible, have no significance in the resulting value.

- The symbol being assigned in a direct assignment statement is placed in the label
  field.

- Only one symbol can be defined in a single direct assignment statement.

- A direct assignment statement can be followed only by a comment field.

- Only one level of forward referencing is allowed. The following example would
  cause an error code (U) in the assembly listing on the line containing the invalid
  forward reference:

      X=Y       ;Invalid forward reference
      Y=Z       ;Valid forward reference
      Z=1

Although one level of forward referencing is allowed for local symbols, no forward
referencing is allowed for global symbols. In other words, the expression being
assigned to a global symbol can contain only previously defined symbols. A forward
reference in a direct assignment statement defining a global symbol causes an error
code (A) in the assembly listing.

3.4 Register Symbols

The eight general registers of the PDP-11 processor are numbered 0 through 7 and
can be expressed in the source program in the following manner:

%0
%1
...
%7

where % indicates a reference to a register rather than a location. The digit specifying
the register can be replaced by any valid, absolute term that can be evaluated during
the first assembly pass.
The register definitions listed below are the normal default values predefined by MACRO–11. They remain valid for all register references within a source program.

- R0=%0 ;Register 0 definition.
- R1=%1 ;Register 1 definition.
- R2=%2 ;Register 2 definition.
- R3=%3 ;Register 3 definition.
- R4=%4 ;Register 4 definition.
- R5=%5 ;Register 5 definition.
- SP=%6 ;Stack pointer definition.
- PC=%7 ;Program counter definition.

Registers 6 and 7 are given special names because of their unique system functions. The symbolic default names assigned to the registers, as listed above, are the conventional names used in all DIGITAL-supplied PDP–11 system programs. For this reason, you are advised to follow these conventions.

A register symbol can be defined in a direct assignment statement appearing in the program. The defining expression of a register symbol must be a valid, absolute value between 0 and 7, inclusive, or an error code (R) will appear in the assembly listing. Although you can reassign the standard register symbols through the use of the .DSABL REG directive (see Section 6.2.1), this practice is not recommended. An attempt to redefine a default register symbol without first specifying the .DSABL REG directive to override the normal register definitions causes that assignment statement to be flagged with an error code (R) in the assembly listing. All nonstandard register symbols must be defined before they are referenced in the source program.

The % character can be used with any valid term or expression to specify a register. For example, the statement:

```
CLR %3+1
```

is equivalent in function to the statement:

```
CLR %4
```

and clears the contents of register 4.

In contrast, the statement:

```
CLR 4
```

clears the contents of virtual memory location 4.

The accumulator registers used in floating-point instructions can be defined in a similar manner. For example, with the definition:

```
ACO=%0
```

the statement:

```
MULF @R0,ACO
```

multiplies the contents of floating-point accumulator register AC0 by the floating-point number addressed by R0.
3.5 Local Symbols

Local symbols are specially formatted symbols used as labels within a block of coding that has been delimited as a local symbol block. Local symbols are of the form n$, where n is a decimal integer from 1 to 65535, inclusive. Examples of local symbols are:

1$
27$
59$
104$

A local symbol block is delimited in one of three ways:

- The range of a local symbol block usually consists of those statements between two normally constructed symbolic labels (see Figure 3–1). Note that a statement of the form:

  \[
  \text{ALPHA}\rightarrow\text{EXPRESSION}
  \]

  is a direct assignment statement (see Section 3.3) but does not create a label and thus does not delimit the range of a local symbol block.

- The range of a local symbol block is normally terminated upon encountering a .PSECT, .CSECT, .ASECT, or .RESTORE directive in the source program (see Figure 3–1).

- The range of a local symbol block is delimited through MACRO–11 directives, as follows:

  Starting delimiter: .ENABL LSB (see Section 6.2.1)
  Ending delimiter: .DSABL LSB

  or one of the following:

  Symbolic label (see Section 2.2.1)
  .PSECT (see Section 6.7.1)
  .CSECT (see Section 6.7.2)
  .ASECT (see Section 6.7.2)
  .RESTORE (see Section 6.7.4)

  encountered after a .DSABL LSB (see Section 6.2.1).

Local symbols provide a convenient means of generating labels for branch instructions and other such references within local symbol blocks. Using local symbols reduces the possibility of symbols with multiple definitions appearing within a user program. In addition, the use of local symbols differentiates entry-point labels from local labels, since local symbols cannot be referenced from outside their respective local symbol blocks. Thus, local symbols of the same name can appear in other local symbol blocks without conflict. Local symbols do not appear in cross-reference listings and require less symbol table space than other types of symbols. Their use is recommended.

When defining local symbols, use the range from 1$ to 29999$ first. Local symbols within the range 30000$ through 65535$, inclusive, can be generated automatically as a feature of MACRO–11. Such local symbols are useful in the expansion of macros during assembly (see Section 7.3.5).
Be sure to avoid multiple definitions of local symbols within the same local symbol block. For example, if the local symbol 10$ is defined more than once within the same local symbol block, each symbol represents a different address value. Such a multidefined symbol causes an error code (P) in the assembly listing.

For examples of local symbols and local symbol blocks as they appear in a source program, see Figure 3–1.

Figure 3–1: Assembly Listing Showing Local Symbol Block

```
; Simple illustration of local symbols; the second block is delimited by the label XCTPAS.

000000 012700 XCTPAS: MOV #IMPURE,RO ;Point to impure area
000000
7 000004 008020 1$: CLR (RO)+ ;Clear a word
000006 000027 CMP RO,#IMPUTR ;Test if at top of area

9 000012 001574 BNE 1$ ;Iterate if not
10
11 000014 012700 XCTPAS: MOV #IMPPAS,RO ;Point to pass storage area
000000
12 000020 008020 1$: CLR (RO)+ ;Clear the area
13 000022 000027 CMP RO,#IMMAT ;Test if at top of area

14 000028 001574 BNE 1$ ;Iterate if not
15 000030 000027 RETURN ;Return if so
```

3.6 Current Location Counter

The period (.) is the symbol for the current location counter. When used in the operand field of an instruction, the period represents the address of the first word of the instruction, as shown in this example:

```
A: MOV .,RO ;The period (.) refers to the address of the MOV instruction.
```

The function of the number sign (#) is explained in Section 5.9.

When used in the operand field of a MACRO–11 directive, the period represents the address of the current byte or word, as shown here:

```
SAL=0
.WORD 177535, +4, SAL ;The operand .+4 in the .WORD directive represents a value that is stored as the second of three words during assembly.
```

Assume that the current value of the location counter is 1500$8. During assembly, MACRO–11 reserves storage in response to the .WORD directive (see Section 6.3.2), beginning with location 1500$. The operands accompanying the .WORD directive determine the values so stored. The value 177535$8 is thus stored in location 1500. The value represented by .+4 is stored in location 1502; this value is derived as the current value of the location counter (which is now 1502), plus the absolute value 4, thereby depositing the value 1506 in location 1502. Finally, the value of SAL, previously equated to 0, is deposited in location 1504.
At the beginning of each assembly pass, MACRO–11 resets the location counter. Normally, consecutive memory locations are assigned to each byte of object data generated. However, the value of the location counter can be changed through a direct assignment statement of the following form:

```
=expression
```

The current location counter symbol (.) is either absolute or relocatable, depending on the attribute of the current program section.

The attribute of the current location counter can be changed only through the program sectioning directives (.PSECT, .ASECT, .CSECT, and .RESTORE), as described in Section 6.7. Therefore, assigning to the current location counter an expression having an attribute other than that of the current program section will generate an error code (A) in the assembly listing.

Furthermore, an expression assigned to the current location counter cannot contain a forward reference (a reference to a symbol that is not previously defined). You must also be sure that the expression assigned does not force the current location counter into another program section, even if both sections involved have the same relocatability. Either of these conditions causes MACRO–11 to generate incorrect object file code, and may cause statements following the error to be flagged with an error code (P) in the assembly listing.

The following coding illustrates the use of the current location counter:

```assembly
.ASECT
=1500

FIRST: MOV .+10,COUNT
;The label "FIRST" has the value
;1500(octal).
;.+10 equals 1510(octal). The
;contents of the location
;1510(octal) will be deposited
;in the location "COUNT".

=1520

SECOND: MOV _,INDEX
;The label "SECOND" has the
;value 1520(octal).
;The contents of location
;1520(octal), that is, the binary
;code for the instruction
;itself, will be deposited in the
;location "INDEX".

.PSECT
=.+20

THIRD: .WORD 0
;The label "THIRD" has the
;value of relocatable 20.
```
Storage areas can be reserved in the program by advancing the location counter. For example, if the current value of the location counter is 1000, each of the following statements:

```
.=.+40
```
or:
```
.BLKB 40
```
or:
```
.BLKW 20
```
reserves 40\textsubscript{8} bytes of storage space in the source program starting at location 1000.
The \texttt{.BLKB} and \texttt{.BLKW} directives, however, are the preferred ways to reserve storage space (see Section 6.5.3).

### 3.7 Numbers

MACRO–11 assumes that all numbers in the source program are to be interpreted in octal radix, unless otherwise specified. An exception to this assumption is that operands associated with Floating Point Processor instructions and Floating Point Data directives are treated as decimal (see Section 6.4.2). The default radix (octal) can be changed with the \texttt{.RADIX} directive (see Section 6.4.1.1). Also, individual numbers can be designated as binary, octal, decimal, or hexadecimal numbers through temporary radix control operators (see Section 6.4.1.2).

If a statement in the source program contains a digit that is not in the current radix, MACRO–11 generates an error code (N) in the assembly listing. However, MACRO–11 continues with the scan of the statement and evaluates each such number encountered as a decimal value.

Negative numbers must be preceded by a minus sign; MACRO–11 translates such numbers into two’s complement form. Positive numbers may (but need not) be preceded by a plus sign.

A number containing more than 16 significant bits (greater than $177777_{8}$) is truncated from the left and flagged with an error code (T) in the assembly listing.

Numbers are always considered to be absolute values; therefore, they are never relocatable.

Single-word floating-point numbers can be generated with the \texttt{\^F} operator (see Section 6.4.2.3) and are stored in the following format:

```
\textbf{15} | \textbf{14} | 7 | 6 | 0
```

| S | EEEEEEE | NNNNNNN

Sign (1 bit)

Exponent (8 bits)

Mantissa (7 bits)

Refer to the \textit{PDP–11 Processor Handbook} for details of the floating-point number format.
3.8 Terms

A term is a component of an expression and can be one of the following:

- A number (see Section 3.7) whose 16-bit value is used.
- A symbol (see Section 3.2). Symbols are evaluated as follows:
  - A period (.) specified in an expression causes the value of the current location counter to be used.
  - A defined symbol is located in the User Symbol Table and its value is used.
  - A permanent symbol's basic value is used, with zero substituted for the addressing modes. (Appendix C lists all op codes and their values.)
  - An undefined symbol is assigned a value of zero and inserted in the User Symbol Table as an undefined default global reference. If the .DSABL GBL directive (see Section 6.2.1) is in effect, the automatic global reference default function of MACRO-11 is inhibited, and the statement containing the undefined symbol is flagged with an error code (U) in the assembly listing.
- A single quote followed by a single ASCII character, or a double quote followed by two ASCII characters. This type of expression construction is explained in detail in Section 6.3.3.
- An expression enclosed in angle brackets (<>). Any expression so enclosed is evaluated and reduced to a single term before the remainder of the expression in which it appears is evaluated. For example, angle brackets can be used to alter the left-to-right evaluation of expressions, as in A*B+C versus A*<B+C>, or to apply a unary operator to an entire expression, as in -<A*B>.
- A unary operator followed by a symbol or number.

3.9 Expressions

Expressions are combinations of terms joined together by binary operators (see Table 3-5). Expressions reduce to a 16-bit value. The evaluation of an expression includes the determination of its attributes. A resultant expression value can be any one of four types: relocatable, absolute, external, or complex relocatable.

Expressions are evaluated from left to right with no operator hierarchy rules, except that unary operators take precedence over binary operators. A term preceded by a unary operator is considered to contain that operator. (Terms are evaluated, where necessary, before their use in expressions.) Multiple unary operators are valid and are treated as follows:

\[-++A\]

is equivalent to:

\[-<<<<-A>>\]

---

1 The maximum depth of an expression is governed by the MACRO-11 assembler's expression stack space. If an expression exceeds the assembler's maximum expression depth, the statement is marked with an (E) error, and processing continues.
A missing term, expression, or external symbol is interpreted as a zero. A missing or invalid operator terminates the expression analysis, causing error codes (A) and/or (Q), to be generated in the assembly listing, depending on the context of the expression itself. For example, the expression:

\[ A + B \ 177777 \]

is evaluated as:

\[ A + B \]

because the first nonblank character following the symbol B is not a valid binary operator, an expression separator (a comma), or an operand field terminator (a semicolon or the end of the source line).

Spaces within expressions can serve as delimiters only between symbols. In other words, the expressions:

\[ A + B \]

and:

\[ A+B \]

are the same, but the symbols:

\[ B17 \]

and:

\[ B \ 17 \]

are not (B 17 is not a single symbol).

At assembly time the value of an external (global) expression is equal to the value of the absolute part of that expression. For example, the expression \texttt{EXTERN+A}, where \texttt{EXTERN} is an external symbol, has a value at assembly time that is equal to the value of the internal (local) symbol \texttt{A}. However, when evaluated at link time, this expression takes on the resolved value of the symbol \texttt{EXTERN} plus the value of symbol \texttt{A}.

When evaluated by MACRO–11, expressions are one of four types: relocatable, absolute, external, or complex relocatable. The following distinctions are important:

- An expression is \textbf{relocatable} if its value is fixed relative to the base address of the program section in which it appears; it will have an offset value added at link time. Terms that contain labels defined in relocatable program sections will have a relocatable value; similarly, a period (.) in a relocatable program section, representing the value of the current location counter, will also have a relocatable value.

- An expression is \textbf{absolute} if its value is fixed. An expression whose terms are numbers and ASCII conversion characters will reduce to an absolute value. A relocatable expression or term minus a relocatable term, where both elements being evaluated belong to the same program section, is an absolute expression. This is because every term in a program section has the same relocation bias. When one term is subtracted from another, the resulting bias is zero. MACRO–11 can then treat the expression as absolute and reduce it to a single term upon completion.
of the expression scan. Terms that contain labels defined in an absolute program section also have an absolute value.

- An expression is **external** (or global) if it contains a single global reference (plus or minus an absolute expression value) that is not defined within the current program. Thus, an external expression is only partially defined following assembly and must be resolved at link time.

- An expression is **complex relocatable** if any one of the following conditions applies:
  - It contains a global reference and a relocatable symbol.
  - It contains more than one global reference.
  - It contains relocatable terms belonging to different program sections.
  - The value resulting from the expression has more than one level of relocation. For example, if the relocatable symbols TAG1 and TAG2, associated with the same program section, are specified in the expression TAG1+TAG2, two levels of relocation will be introduced, since each symbol is evaluated in terms of the relocation bias in effect for the program section.
  - An operation other than addition is specified on an undefined global symbol.
  - An operation other than addition, subtraction, negation, or complementation is specified for a relocatable value.

The evaluation of relocatable, external, and complex relocatable expressions is completed at link time. The maximum number of terms that can be specified in a complex expression is $20_{10}$ and is limited by the maximum size of the object record.
The output of MACRO-11 is an object module that must be processed or linked before it can be loaded and executed. Linking fixes (makes absolute) the values of relocatable or external symbols in the object module, thus transforming the object module, or several object modules, into an executable image.

To allow the value of an expression to be fixed at link time, MACRO-11 writes certain instructions in the object file, together with other required parameters. For relocatable expressions in the object module, the base of the associated relocatable program section is added to the value of the relocatable expression provided by MACRO-11. For external expression values (those containing a reference to a global symbol defined in another module), the value of the external term in the expression (since the external symbol must be defined in one of the other object modules being linked together) is determined and then added to the absolute portion of the external expression, as provided by MACRO-11.

All instructions that require modification at link time are flagged in the assembly listing, as illustrated in the example below. The single quote (') following the octal expansion of the instruction indicates that simple relocation is required; the letter G indicates that the value of an external (global) symbol must be added to the absolute portion of an expression; and the letter C indicates that complex relocation analysis at link time is required in order to fix the value of the expression.

Example:

```
005065 CLR RELOC(R5) ;Assuming that the value of the symbol "RELOC", 40, is relocatable
                      ;the relocation bias
                      ;will be added to this value.
005065 CLR EXTERN(R5) ;The value of the symbol "EXTERN" is
                      ;assembled as zero and is
                      ;resolved at link time.
005065 CLR EXTERN+6(R5) ;The value of the symbol "EXTERN"
                     ;is resolved at link time
                     ;and added to
                     ;the absolute portion (+6) of
                     ;the expression.
005065 CLR -<EXTERN+RELOC>(R5) ;This expression is complex
                          ;relocatable because it requires
                          ;the negation of an expression
                          ;that contains a global "EXTERN"
                          ;reference and a relocatable term.
```

For a complete description of object records written by MACRO-11, refer to the applicable system manual (see the Associated Documents section in the Preface).
Chapter 5

Addressing Modes

To understand how the address modes operate and how they assemble, you must understand the action of the program counter. The key rule to remember is:

"Whenever the processor implicitly uses the program counter (PC) to fetch a word from memory, the program counter is automatically incremented by 2 after the fetch operation is completed."

The PC always contains the address of the next word to be fetched. This word will be either the address of the next instruction to be executed or the second or third word of the current instruction.

Table 5–1 lists the symbols used in this chapter to describe the address modes, and Table 5–2 lists the address modes. This chapter illustrates each mode of address using the single operand instruction CLR or the double operand instruction MOV. Section B.2 gives a summary of address mode syntax.

Certain special instruction/address mode combinations, which are rarely or never used, do not operate the same on all PDP–11 processors. There are three major classes of instructions you are most likely to encounter. These are listed in Table 5–3. Avoid using these addressing modes if there is the slightest chance a program will ever have to run on more than one type of processor. MACRO–11 prints an error code (Z) in the assembly listing with each instruction containing an addressing mode incompatible among all members of the PDP–11 family.

Table 5–1: Symbols Used in Chapter 5

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Any expression, as defined in Chapter 3.</td>
</tr>
<tr>
<td>R</td>
<td>A register expression; that is, any expression containing a term preceded by a percent sign (%) or a symbol previously equated to such a term, as shown below:</td>
</tr>
<tr>
<td></td>
<td>R0=0%0 ; General register 0.</td>
</tr>
<tr>
<td></td>
<td>R1=R0+1 ; General register 1.</td>
</tr>
<tr>
<td></td>
<td>R2=1+1%1 ; General register 2.</td>
</tr>
<tr>
<td></td>
<td>This symbol may also represent any of the normal default register definitions (see Section 3.4).</td>
</tr>
<tr>
<td>ER</td>
<td>A register expression or an absolute expression in the range 0 to 7, inclusive.</td>
</tr>
</tbody>
</table>

1 The PDP–11 Architecture Handbook lists all the differences among all PDP–11 processors.
<table>
<thead>
<tr>
<th>Mode</th>
<th>Form</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register mode(^1)</td>
<td>R</td>
<td>5.1</td>
</tr>
<tr>
<td>Register deferred mode(^1)</td>
<td>@R or (ER)</td>
<td>5.2</td>
</tr>
<tr>
<td>Autoincrement mode(^1)</td>
<td>(ER)(^+)</td>
<td>5.3</td>
</tr>
<tr>
<td>Autoincrement deferred mode(^1)</td>
<td>@ (ER)(^+)</td>
<td>5.4</td>
</tr>
<tr>
<td>Autodecrement mode(^1)</td>
<td>-(ER)</td>
<td>5.5</td>
</tr>
<tr>
<td>Autodecrement deferred mode(^1)</td>
<td>@-(ER)</td>
<td>5.6</td>
</tr>
<tr>
<td>Index mode(^2)</td>
<td>E(ER)</td>
<td>5.7</td>
</tr>
<tr>
<td>Index deferred mode(^2)</td>
<td>@E(ER)</td>
<td>5.8</td>
</tr>
<tr>
<td>Immediate mode(^2)</td>
<td>#E</td>
<td>5.9</td>
</tr>
<tr>
<td>Absolute mode(^2)</td>
<td>@#E</td>
<td>5.10</td>
</tr>
<tr>
<td>Relative mode(^2)</td>
<td>E</td>
<td>5.11</td>
</tr>
<tr>
<td>Relative deferred mode(^2)</td>
<td>@E</td>
<td>5.12</td>
</tr>
<tr>
<td>Branch</td>
<td>Address</td>
<td>5.13</td>
</tr>
</tbody>
</table>

\(^1\)Does not increase the length of an instruction.

\(^2\)Adds one word to the instruction length for each occurrence of an operand of this form.

\(^3\)Section B.2 contains a summary of addressing mode syntax.
### Table 5–3: Instruction Differences Among PDP–11 Processors

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation A</th>
<th>Operation B</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPR R, (R) +2</td>
<td>Contents of R are incremented (or decremented) by 2 before being used as the source operand⁵⁶</td>
<td>Initial contents of R are used as the source operand⁵⁶</td>
</tr>
<tr>
<td>OPR R, -(R)</td>
<td>Location A will contain the PC of OPR+4⁵</td>
<td>Location A will contain the PC of OPR+2⁶</td>
</tr>
<tr>
<td>OPR R, @R</td>
<td>Contents of R are incremented by 2, then used as the new PC⁷</td>
<td>Initial contents of R are used as the new PC⁸</td>
</tr>
</tbody>
</table>

1️⃣ OPR represents any two-operand instruction
2️⃣ R is the same for both source and destination
3️⃣ A represents any address expression
4️⃣ Rn is not necessarily the same as R
5️⃣ 23/24, 15/20, 35/40, 60, J11, and T11 processors
6️⃣ 04, 05/10, 34, 44, 45, and 70 processors
7️⃣ 05/10 and 15/20 processors
8️⃣ All except 05/10 and 15/20 processors

### 5.1 Register Mode

Format:

```
R
```

The register R contains the operand for the instruction.

Example:

```
CLR R3 ;Clears register 3.
```

### 5.2 Register Deferred Mode

Format:

```
@R
(ER)
```

The register R contains the address of the operand for the instruction.

Example:

```
CLR @R1 ;All these instructions clear
CLR (R1) ;the word at the address
CLR (@1) ;contained in register 1.
```
5.3 Autoincrement Mode

Format:

\((\text{ER})^+\)

The contents of the register ER are incremented immediately after being used as the address of the operand (see Table 5–3 for possible processor incompatibilities).

Example:

\[
\begin{align*}
\text{CLR} & \quad (\text{R0})^+ \quad \text{;Each instruction clears} \\
\text{CLR} & \quad (\text{R4})^+ \quad \text{;the word at the address} \\
\text{CLR} & \quad (\text{R2})^+ \quad \text{;contained in the specified} \\
& \quad \quad \text{;register and increments} \\
& \quad \quad \text{;that register's contents} \\
& \quad \quad \text{;by 2.}
\end{align*}
\]

5.4 Autoincrement Deferred Mode

Format:

\((\text{~ER})^+\)

The register ER contains a pointer to the address of the operand. The contents of the register are incremented after being used as pointer.

Example:

\[
\begin{align*}
\text{CLR} & \quad (\text{R3})^+ \quad \text{;The contents of register 3 point} \\
& \quad \quad \text{;to the address of a word to be} \\
& \quad \quad \text{;cleared before the contents of the} \\
& \quad \quad \text{;register are incremented by 2.}
\end{align*}
\]

5.5 Autodecrement Mode

Format:

\(-(\text{ER})\)

The contents of the register ER are decremented before being used as the address of the operand (see Table 5–3 for possible processor incompatibilities).

Example:

\[
\begin{align*}
\text{CLR} & \quad -(\text{R0}) \quad \text{;Decrement the contents of the} \\
& \quad \quad \text{;specified register (0, 3, or 2)} \\
\text{CLR} & \quad -(\text{R3}) \quad \text{;by 2 before using its contents} \\
\text{CLR} & \quad -(\text{R2}) \quad \text{;as the address of the word to be} \\
& \quad \quad \text{;cleared.}
\end{align*}
\]

5.6 Autodecrement Deferred Mode

Format:

\((\text{~ER})^-\)

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

Example:

CLR @-(R3) ; Decrement the contents of
; register 3 by 2 before
; using its contents as a pointer
; to the address of the word to be
; cleared.

5.7 Index Mode

Format:

E(ER)

An expression E, plus the contents of a register ER, yields the effective address of the
operand. In other words, the value E is the offset of the instruction, and the contents
of register ER form the base. The value of the expression E is stored as the second or
third word of the instruction.

Example:

CLR X+2(R1) ; The effective address of the word
; to be cleared is X+2, plus the
; contents of register 1.
MOV R0,-2(R3) ; The effective address of the
; destination location is -2, plus
; the contents of register 3.

5.8 Index Deferred Mode

Format:

@E(ER)

An expression E, plus the contents of register ER, yields a pointer to the address of the
operand. The value E is the offset of the instruction, and the contents of register ER
form the base. The value of the expression E is stored as the second or third word of
the instruction.

Example:

CLR @114(R4) ; If register 4 contains 100, this
; value, plus the offset 114, yields
; the pointer 214. If location 214
; contains the address 2000, location
; 2000 would be cleared.

NOTE

The expression @E(ER) can be used, but it will be assembled as if it were written @0(ER), and a word will be used to store the 0.
5.9 Immediate Mode

Format:

#E

Immediate mode stores the operand itself (E) as the second or third word of the instruction. The number sign (#) is an addressing mode indicator. This character appearing in the operand field specifies the immediate addressing mode, indicating to MACRO-11 that the operand itself immediately follows the instruction word. This mode is assembled as an autoincrement of the PC.

Example:

    MOV   #100,RO
    MOV   #X,RO

;Move the value 100 into register 0.
;Move the value of symbol X into
;register 0.

The operation of this mode can be shown through the first example, MOV #100,RO, which assembles as two words:

Location n:  012700
Location n+2: 000100
Location n+4: Next instruction

The source operand (the value 100) is assembled immediately following the instruction word. Upon execution of the instruction, the processor fetches the first word (MOV) and increments the PC by 2, so that it points to the second word, location n+2, which contains the source operand.

After the next fetch and increment cycle, the source operand (100) is moved into register 0, leaving the PC pointing to location n+4 (the next instruction).

5.10 Absolute Mode

Format:

@#E

Absolute mode is the equivalent of immediate mode deferred. The address expression @#E specifies an absolute address that is stored as the second or third word of the instruction. In other words, the value immediately following the instruction word is taken as the absolute address of the operand. Absolute mode is assembled as an autoincrement deferred of the PC. You can use this mode to reference specific memory addresses from within position-independent code.

Example:

    MOV   @#100,RO
    CLR   @#X

;Move the contents of absolute
;location 100 into register RO.
;Clear the contents of the location
;whose address is specified by
;the symbol X.

The operation of this mode can be shown through the first example:

    MOV   @#100,RO
which assembles as two words:

Location n: 013700
Location n+2: 000100
Location n+4: Next instruction

The absolute address 100 is assembled immediately following the instruction word. Upon execution of the instruction, the processor fetches the first word (MOV) and increments the PC by 2, so that it points to the second word, location n+2, which contains the absolute address of the source operand. After the next fetch and increment cycle, the contents of absolute address 100 (the source operand) are moved into register 0, leaving the PC pointing to location n+4 (the next instruction).

5.11 Relative Mode

Format:

E

Relative mode is the normal mode for memory references within your program. It is assembled as index mode, using the PC as the index register. The offset for the address calculation is assembled as the second or third word of the instruction. This value is added to the contents of the PC to yield the address of the source operand.

Example:

CLR 100 ;Clear absolute location 100
MOV RO,Y ;Move the contents of register 0
to location Y

Assume the current value of the PC is 1020. The operation of relative mode can be shown with the statement:

MOV 100,R3

which assembles as two words:

Location 1020: 016703
Location 1022: 177054
Location 1024: Next instruction

The offset, the constant 177054, is assembled immediately following the instruction word. Upon execution of the instruction, the processor fetches the first word (MOV) and increments the PC by 2, so that it points to the second word, location 1022, containing the value 177054. After the next fetch and increment cycle, the processor calculates the effective address of the source operand by taking the contents of location 1022 (the offset) and adding it using two's complement arithmetic to the current value of the PC, which now points to location 1024 (the next instruction). Thus, the source operand address is the result of the calculation:

OFFSET+PC = 177054+1024 = 1008

so the contents of location 100 are moved into register 3.

The index mode statement:

MOV 100-4(PC),R3
is equivalent to the relative mode statement:

```
MOV 100,R3
```

The term 100.—4 is the offset for the index mode statement. The current location counter (.) holds the address of the first word of the instruction (1020, in this case), and the PC has to move down four bytes to reach location 1024 (the next instruction). So, the offset could be written as 100—1020—4, or 1770548.

Therefore, for the index mode, the offset (1770548) added to the PC (10248) yields the effective address (177054 + 1024 = 1008) of the operand.

Thus, both statements move the contents of location 100 into register 3.

**NOTE**
The addressing form @#E differs from 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 (see Section 5.10). Thus, the instruction CLR @#100 clears absolute location 100, even if the instruction is moved from the point at which it was assembled. See Section 6.2.1 for a description of the .ENABLAMA function, which causes all relative mode addresses to be assembled as absolute mode addresses.

### 5.12 Relative Deferred Mode

**Format:**

```
@E
```

Relative deferred mode is similar in operation to relative mode, except that the expression E is used as a pointer to the address of the operand. In other words, the operand following the instruction word is added to the contents of the PC to yield a pointer to the address of the operand.

**Example:**

```
MOV @@X,RO ;Relative to the current value of 
             ;the PC, move the contents of the 
             ;location whose address is pointed 
             ;to by location X into register 0.
```

### 5.13 Branch Instruction Addressing

Branch instructions are 1-word instructions. The high-order byte contains the operator, and the low-order byte contains an 8-bit signed offset (seven bits, plus sign), which specifies the branch address relative to the current value of the PC. The hardware calculates the branch address as follows:

1. Extends the sign of the offset through bits 8 to 15.
2. Multiplies the result by 2, creating a byte offset rather than a word offset.
3. Adds the result to the current value of the PC to form the effective branch address. 

MACRO-11 performs the reverse operation to form the word offset from the specified address:

Word offset = \((E-PC)/2\), truncated to eight bits.

When the offset is added to the PC, the PC is moved to the next word \((PC=PC+2)\). Hence the \(-2\) in the following calculation:

Word offset = \((E-\text{-}2)/2\), truncated to eight bits.

The following conditions generate an error code \((A)\) in the assembly listing:

- Branching from one program section to another
- Branching to a location that is defined as an external (global) symbol
- Specifying a branch address that is out of range, meaning that the branch offset is a value that exceeds the range \(-128\) to \(+127\)

5.14 Using TRAP Instructions

Since the EMT and TRAP instructions do not use the low-order byte of the instruction word, information is transferred to the trap handlers in the low-order byte. If the EMT or TRAP instruction is followed by an expression, the value of the expression is stored in the low-order byte of the word. Expressions greater than \(377\) are truncated to eight bits, and an error code \((A)\) is generated in the assembly listing.

For more information on traps, see the PDP-11 Processor Handbook and the applicable system manual (see the Associated Documents section in the Preface).
Chapter 6
General Assembler Directives

A MACRO–11 directive is placed in the operator field of a source line. Only one directive is allowed per source line. A directive may have a blank operand field or one or more operands. Valid operands differ with each directive.

General assembler directives are divided into the following categories:

- Listing control
- Function control
- Data storage
- Radix and numeric control
- Location counter control
- Terminator
- Program sectioning and boundaries
- Symbol control
- Conditional assembly
- File control

Each is described in its own section of this chapter. See Table 6–1 for an alphabetical listing of the directives and the associated section reference. Also refer to Section B.3 for a complete list of all MACRO–11 assembler directives.

Table 6–1: Directives in Chapter 6

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
<th>Section Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ASCII</td>
<td>Stores delimited string as a sequence of the 8-bit ASCII code of their characters.</td>
<td>6.3.4</td>
</tr>
<tr>
<td>.ASCIZ</td>
<td>Same as .ASCII except the string is followed by a zero byte.</td>
<td>6.3.5</td>
</tr>
<tr>
<td>.AECT</td>
<td>Similar to .PSECT.</td>
<td>6.7.2</td>
</tr>
<tr>
<td>.BLKB</td>
<td>Allocates bytes of data storage.</td>
<td>6.5.3</td>
</tr>
<tr>
<td>.BLKW</td>
<td>Allocates words of data storage.</td>
<td>6.5.3</td>
</tr>
<tr>
<td>.BYTE</td>
<td>Stores successive bytes of data.</td>
<td>6.3.1</td>
</tr>
<tr>
<td>.CROSS</td>
<td>Enables cross referencing.</td>
<td>6.2.2</td>
</tr>
</tbody>
</table>
### Table 6–1 (Cont.): Directives in Chapter 6

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
<th>Section Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>.CSECT</td>
<td>Similar to .PSECT.</td>
<td>6.7.2</td>
</tr>
<tr>
<td>.DSABL</td>
<td>Disables specified assembler functions.</td>
<td>6.2.1</td>
</tr>
<tr>
<td>.ENABL</td>
<td>Enables specified assembler functions.</td>
<td>6.2.1</td>
</tr>
<tr>
<td>.END</td>
<td>Indicates end of source input.</td>
<td>6.6</td>
</tr>
<tr>
<td>.ENDC</td>
<td>Indicates end of conditional assembly block.</td>
<td>6.9.1</td>
</tr>
<tr>
<td>.EVEN</td>
<td>Ensures that current value of the location counter is even.</td>
<td>6.5.1</td>
</tr>
<tr>
<td>.FLT2</td>
<td>Generates 2 words of storage for each floating-point number argument.</td>
<td>6.4.2.2</td>
</tr>
<tr>
<td>.FLT4</td>
<td>Generates 4 words of storage for each floating-point number argument.</td>
<td>6.4.2.2</td>
</tr>
<tr>
<td>.GLOBL</td>
<td>Defines listed symbols as global.</td>
<td>6.8.1</td>
</tr>
<tr>
<td>.IDENT</td>
<td>Provides additional means of labeling an object module.</td>
<td>6.1.4</td>
</tr>
<tr>
<td>.IF</td>
<td>Assembles block if specified conditions are met.</td>
<td>6.9.1</td>
</tr>
<tr>
<td>.IFF</td>
<td>Assembles block if condition tests false.</td>
<td>6.9.2</td>
</tr>
<tr>
<td>.IFT</td>
<td>Assembles block if condition tests true.</td>
<td>6.9.2</td>
</tr>
<tr>
<td>.IFFF</td>
<td>Assembles block regardless of whether condition tests true or false.</td>
<td>6.9.2</td>
</tr>
<tr>
<td>.IIF</td>
<td>Permits writing a 1-line conditional assembly block.</td>
<td>6.9.3</td>
</tr>
<tr>
<td>.INCLUDE</td>
<td>Includes another MACRO–11 source file.</td>
<td>6.10.2</td>
</tr>
<tr>
<td>.LIBRARY</td>
<td>Adds file to MACRO–11 library search list.</td>
<td>6.10.1</td>
</tr>
<tr>
<td>.LIMIT</td>
<td>Allocates 2 words for storage. At link time the Linker or Task Builder puts the lowest address of the load image in the first of the saved words and the address of the first free word following the image in the second.</td>
<td>6.5.4</td>
</tr>
<tr>
<td>.LIST</td>
<td>Increments listing count or lists certain types of code.</td>
<td>6.1.1</td>
</tr>
<tr>
<td>.NLIST</td>
<td>Decrements listing count or suppresses certain types of code.</td>
<td>6.1.1</td>
</tr>
<tr>
<td>.NOCROSS</td>
<td>Disables cross referencing.</td>
<td>6.2.2</td>
</tr>
<tr>
<td>.ODD</td>
<td>Ensures that the current value of the location counter is odd.</td>
<td>6.5.2</td>
</tr>
<tr>
<td>.PACKED</td>
<td>Generates packed decimal data, two digits per byte.</td>
<td>6.3.8</td>
</tr>
<tr>
<td>.PAGE</td>
<td>Starts a new listing page.</td>
<td>6.1.5</td>
</tr>
<tr>
<td>.PSECT</td>
<td>Declares names for program sections and establishes their attributes.</td>
<td>6.7.1</td>
</tr>
<tr>
<td>.RAD50</td>
<td>Generates data in Radix–50 packed format.</td>
<td>6.3.6</td>
</tr>
</tbody>
</table>
### Table 6–1 (Cont.): Directives in Chapter 6

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
<th>Section Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>.RADIX</td>
<td>Changes the default radix throughout or in portions of the source program.</td>
<td>6.4.1.1</td>
</tr>
<tr>
<td>.REM</td>
<td>Delimits a section of comments.</td>
<td>6.1.6</td>
</tr>
<tr>
<td>.RESTORE</td>
<td>Retrieves a previously .SAVED program section.</td>
<td>6.7.4</td>
</tr>
<tr>
<td>.SAVE</td>
<td>Places the current program section on top of the program section context stack.</td>
<td>6.7.3</td>
</tr>
<tr>
<td>.SBTTL</td>
<td>Produces a table of contents immediately preceding the assembly listing and puts subheadings on each page in the listing.</td>
<td>6.1.3</td>
</tr>
<tr>
<td>.TITLE</td>
<td>Assigns a name to the object module and puts headings on each page of the assembly listing.</td>
<td>6.1.2</td>
</tr>
<tr>
<td>.WEAK</td>
<td>Defines listed symbols as WEAK.</td>
<td>6.8.2</td>
</tr>
<tr>
<td>.WORD</td>
<td>Generates successive words of data in the object module.</td>
<td>6.3.2</td>
</tr>
</tbody>
</table>

### 6.1 Listing Control Directives

Listing control directives control the content, format, and pagination of all line printer (see Figure 6–1) and terminal (see Figure 6–2) assembly listing output. On the first line of each page, MACRO–11 prints the following (from left to right):

1. Title of the object module, as established through the .TITLE directive (see Section 6.1.2)
2. Assembler version identification
3. Day of the week
4. Date
5. Time of day
6. Page number

The second line of each assembly listing page contains the subtitle text specified in the last-encountered .SBTTL directive (see Section 6.1.3).

In line printer format (Figure 6–1), binary extensions for statements generating more than one word are listed horizontally.

In terminal format (Figure 6–2), binary extensions for statements generating more than one word are listed vertically. There is no explicit truncation of output to 80 characters by the assembler.
Figure 6–1: Example of Line Printer Assembly Listing

1  ; GETSYN
2  ; Scan off a RAD50 symbol. Leave with scan pointer set at next non-blank
3  ; char past end of symbol. Symbol buffer clear and Z set if no symbol
4  ; seen: in this case scan pointer is unaltered.
5  ;
6  ;
7  ;
8  000126 010146  GETSYN: MOV R1,-(SP) ;Save work register
9  000130 016767 0000000 0000000 MOV CHREPST,SYMREG ;Save scan pointer in case of rescan
10  000136 012701 0000000 MOV #STORSHL+4,R1 ;Point at end of symbol buffer
11  000142 000641  CLR -(R1) ;Now clear it
12  000144 000641  CLR -(R1)
13  000146 186627 0000000 0000000 BITB CTBL2(R5),#CT.ALPH ;Test first char for alphabetic
14  000148 001436  BEQ 48 ;Exit if not, with Z set
15  000150 116600 000262' MOVW CTBL2(R5),R0 ;Map to RAD50
16  000152 003431  BLE 38 ;Exit if not valid RAD50
17  000154 000630  ASL R0 ;Make word index
18  000156 016011 0000000 MOV $50TH(R0),R1 ;Load the high char
19  000158 000172  GETCHR ;Get another char
20  00015A 116600 000262' MOVW CTBL2(R5),R0 ;Handle it as above
21  000162 003421  BLE 38
22  000164 000630  ASL R0
23  000166 000011 0000000 ADD $50TH(R0),R1
24  000168 000211  GETCHR ;Now get low order char
25  00016A 116600 000262' MOVW CTBL2(R5),R0 ;Map and test it
26  00016C 003411  BLE 38
27  00016E 000021  ADD R0,(R1)+ ;Just add in the low char, advance pointer
28  000170 000226  GETCHR ;Get following char
29  000172 020127 0000000 CMP R1,#STORSHL+4 ;Test if at end of symbol buffer
30  000174 001347  BNE 18 ;Go again if no
31  000176 105766 000262' TESTB CTBL2(R5) ;Flush to end of symbol if it yes
32  000178 000370  BGT 28
33  00017A 003446  SETNB ;Now scan to a non-blank char
34  00017C 012801 48 MOV (SP)+,R1 ;Restore work register
35  00017E 001700 0000000 MOV SYMREG,R0 ;Set Z if no symbol found
36  000180 000207  RETURN
37  ;
38  ; Table CTBL2
39  ; Index with 7-bit ASCII value to get corresponding RAD50 value
40  ; If EQ 0 then space, if LT 0 then not RAD50; Other bits reserved.
41  ;
42  ;
43  ;
44  000262 200 200 200 CTBL2: BYTE 200,200,200,200,200,200,200,200 ;
45  000272 200 200 200 .BYTE 200,200,200,200,200,200,200,200 ;
46  000302 200 200 200 .BYTE 200,200,200,200,200,200,200,200 ;
47  000312 200 200 200 .BYTE 200,200,200,200,200,200,200,200 ;
48  000322 200 200 200 .BYTE 200,200,200,200,200,200,200,200 ;
49  000332 200 200 200 .BYTE 200,200,200,200,200,200,200,200 ;
50  000342 038 037 040 .BYTE 038,037,040,041,042,043,044,046 ;01234567
51  000352 048 047 .BYTE 048,047,200,200,200,200,200,200 ;0
52  000362 200 001 002 .BYTE 200,001,002,003,004,005,006,007 ;abcdefg
53  000372 010 011 012 .BYTE 010,011,012,013,014,015,016,017 ;HIJKLMNO
54  000402 020 021 022 .BYTE 020,021,022,023,024,025,026,027 ;PQRSTUFW
55  000412 030 031 032 .BYTE 030,031,032,200,200,200,200,200 ;XYZ
56  000422 200 001 002 .BYTE 200,001,002,003,004,005,006,007 ;abcdefg
Figure 6–2: Example of Terminal Assembly Listing

1  
2  :*
3  : GETSYM
4  ; Scan off a RADO symbol. Leave with scan pointer set at next non-blank
5  ; char past end of symbol. Symbol buffer clear and Z set if no symbol
6  ; seen; in this case scan pointer is unaltered.
7  :*
8  000126 010146 GETSYM: MOV R1, -(SP) ;Save work register
9  000130 016767 MOV CHPRT, SYMRED ;Save scan pointer in case of rescan
10  000000
11  000000
12  000136 012701 MOV #SYMRED+4, R1 ;Point at end of symbol buffer
13  000000
14  000142 005041 CLR -R1 ;Now clear it
15  000144 005041 CLR -R1
16  000146 136677 BITB CTBLR(5), #GT, ALP ;Test first char for alphabetic
17  000000
18  000000
19  000154 001498 BEQ 4$ ;Exit if not, with Z set
20  000166 116600 1$: MOVB CTBLB(5), R0 ;Map to RADO
21  000262
22  000162 003431 BLE 3$ ;Exit if not valid RADO
23  000164 006500 ASL R0 ;Make word index
24  000166 016011 MOV R5GETB(R0), R1 ;Load the high char
25  000000
26  000172 GETCHR ;Get another char
27  000176 116600 MOVDB CTBLB(5), R0 ;Handle it as above
28  000262
29  000202 003431 BLE 3$ ;Get lower char
30  000204 006500 ASL R0
31  000206 066011 ADD R5GETB(R0), R1
32  000000
33  000212 GETCHR ;Now get low order char
34  000216 116600 MOVDB CTBLB(5), R0 ;Map and test it
35  000262
36  000222 003431 BLE 3$ ;Just add in the low char, advance pointer
37  000224 006021 ADD R0, R1
38  000226 2$: GETCHR ;Get following char
39  000232 020127 CMP R1, #SYMRED+4 ;Test if at end of symbol buffer
40  000000
41  000236 001347 BNE 1$ ;Go again if no
42  000240 106765 TSTB CTBLB(5) ;Flush to end of symbol if it yes
43  000262
44  000244 003370 BGT 2$ ;Set if no symbol found
45  000000
46  000260 000207 RET
47  37
48  38 : Table CTBL2
49  ; Index with 7-bit ASCII value to get corresponding RADO value
50  ; If EQ 0 then space, if LT 0 then not RADO; Other bits reserved.
51  :*
52  000262 00 CTBL2: BYTE 200, 200, 200, 200, 200, 200, 200, 200, 200 ;
53  000272 00 .BYTE 200, 200, 200, 200, 200, 200, 200, 200, 200, 200 ;
54  000302 00 .BYTE 200, 200, 200, 200, 200, 200, 200, 200, 200, 200 ;
55  000312 00 .BYTE 200, 200, 200, 200, 200, 200, 200, 200, 200, 200 ;
56  000322 00 .BYTE 200, 200, 200, 200, 200, 200, 200, 200, 200, 200 ;
57  000332 00 .BYTE 200, 200, 200, 200, 200, 200, 200, 200, 200, 200 ;
58  000342 00 .BYTE 036, 037, 040, 041, 042, 043, 044, 045, 01234567 ;
59  000352 00 .BYTE 046, 047, 200, 200, 200, 200, 200, 200, 200, 200 ;
60  000362 00 .BYTE 200, 001, 002, 003, 004, 005, 006, 007, abcd
61  000372 00 .BYTE 010, 011, 012, 013, 014, 015, 016, 017, hijkl
62  000402 00 .BYTE 020, 021, 022, 023, 024, 025, 026, 027, pqrs
63  000412 00 .BYTE 030, 031, 032, 200, 200, 200, 200, 200, xyz
64  000422 00 .BYTE 200, 001, 002, 003, 004, 005, 006, 007, abcd
65  000000

General Assembler Directives 6–5
6.1.1 .LIST And .NLIST Directives

Format:

```
.LIST
.LIST arg
.NLIST
.NLIST arg
```

where:

- `arg` represents one or more of the optional symbolic arguments defined in Table 6-2.

As indicated above, the listing control directives can be used without arguments, in which case the listing directives alter the listing level count. The listing level count is initialized to zero. At each occurrence of a .LIST directive, the listing level count is incremented; at each occurrence of a .NLIST directive, the listing level count is decremented. When the level count is negative, the listing is suppressed (unless the line contains an error). Conversely, when the level count is greater than zero, the listing is generated regardless of the context of the line. Finally, when the count is zero, the line is either listed or suppressed, depending on the listing controls currently in effect for the program. The following macro definition employs the .LIST and .NLIST directives to list selected portions of the macro body when the macro is expanded:

```
.MACRO LTEST ;List test
  ; A-this line should list   ;Listing level count is 0.
  .NLIST                     ;Listing level count is -1.
  ; B-this line should not list
  .NLIST                     ;Listing level count is -2.
  ; C-this line should not list
  .LIST                      ;Listing level count is -1.
  ; D-this line should not list
  .LIST                      ;Listing level count is 0.
  ; E-this line should list   ;Listing level count is 0.
  ; F-this line should list   ;Listing level count is 0.
  ; G-this line should list   ;Listing level count is 0.
  .ENDM

.LIST ME ;List macro expansion.
LTEST ;Call the macro
  ; A-this line should list   ;Listing level count is 0.
  ; E-this line should list   ;Listing level count is 0.
  ; F-this line should list   ;Listing level count is 0.
  ; G-this line should list   ;Listing level count is 0.
```

Note that the lines following line E will list because the listing level count remains 0. If a .LIST ME directive is placed at the beginning of a program, all macro expansions will be listed unless a .NLIST directive is encountered.
An important purpose of the level count is to allow macro expansions to be listed selectively and yet exit with the listing level count restored to the value existing prior to the macro call.

When used with arguments, the listing directives do not alter the listing level count. However, the .LIST and .NLIST directives can be used to override current listing control, as shown in the example below:

```
.MACRO XX
.
.
.LIST ;List next line.
X=
.NLIST ;Do not list remainder of macro
    ;expansion.
.
.ENDM
.NLIST ME ;Do not list macro expansions.
XX
X=
```

Table 6–2 describes the symbolic arguments you can use with .LIST and .NLIST. These arguments can be used singly or in combination with each other. If multiple arguments are specified in a listing directive, each argument must be separated by a comma, tab, or space. For any argument not specifically included in the control statement, the associated default assumption (List or No List) is applicable throughout the source program. The default assumptions for the listing control directives also appear in Table 6–2.

Table 6–2: Symbolic Arguments of Listing Control Directives

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEX</td>
<td>List</td>
<td>Controls the listing of binary extensions (the locations and binary contents beyond those that will fit on the source statement line). This is a subset of the BIN argument.</td>
</tr>
<tr>
<td>BIN¹</td>
<td>List</td>
<td>Controls the listing of generated binary code. If this field is suppressed through a .NLIST BIN directive, left-justification of the source code field occurs in the same manner described above for the LOC field.</td>
</tr>
<tr>
<td>CND</td>
<td>List</td>
<td>Controls the listing of unsatisfied conditional coding and associated .IF and .ENDC directives in the source program. A .NLIST CND directive lists only satisfied conditional coding.</td>
</tr>
<tr>
<td>COM</td>
<td>List</td>
<td>Controls the listing of comments. This is a subset of the SRC argument. The .NLIST COM directive reduces listing time and space when comments are not desired.</td>
</tr>
</tbody>
</table>

¹If the .NLIST arguments SEQ, LOC, BIN, and SRC are in effect at the same time, that is, if all four significant fields in the listing are to be suppressed, the printing of the resulting blank line is inhibited.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td>No list</td>
<td>Controls radix used for assembly listing. If you specify .LIST HEX, addresses and contents are given in hexadecimal, rather than octal.</td>
</tr>
<tr>
<td>LOC¹</td>
<td>List</td>
<td>Controls the listing of the current location counter field. Normally, this field is not suppressed. However, if it is suppressed through the .NLIST LOC directive, MACRO-11 does not generate a tab, nor does it allocate space for the field, as is the case with the SEQ field described above. Thus, the suppression of the current location counter (LOC) field effectively left-justifies all subsequent fields (while preserving positional relationships) to the position normally occupied by the counter's field.</td>
</tr>
<tr>
<td>MC</td>
<td>List</td>
<td>Controls the listing of macro calls and repeat range expansions.</td>
</tr>
<tr>
<td>MD</td>
<td>List</td>
<td>Controls the listing of macro definitions and repeat range expansions.</td>
</tr>
<tr>
<td>ME</td>
<td>No list</td>
<td>Controls the listing of macro expansions.</td>
</tr>
<tr>
<td>MEB</td>
<td>No list</td>
<td>Controls the listing of macro expansion binary code. A .LIST MEB directive lists only those macro expansion statements that generate binary code. This is a subset of the ME argument.</td>
</tr>
<tr>
<td>SEQ¹</td>
<td>List</td>
<td>Controls the listing of the sequential numbers assigned to the source lines. If this number field is suppressed through a .NLIST SEQ directive, MACRO-11 generates a tab, effectively allocating blank space for the field. Thus, the positional relationships of the other fields in the listing remain undisturbed. During the assembly process, MACRO-11 examines each source line for possible error conditions. For any line in error, the error code is printed preceding the number field. (MACRO-11 does not assign line numbers to files that have had line numbers assigned by an editor such as SOS.)</td>
</tr>
<tr>
<td>SRC¹</td>
<td>List</td>
<td>Controls the listing of source lines.</td>
</tr>
<tr>
<td>SYM</td>
<td>List</td>
<td>Controls the listing of the symbol table resulting from the assembly of the source program.</td>
</tr>
<tr>
<td>TOC</td>
<td>List</td>
<td>Controls the listing of the table of contents during assembly pass 1 (see Section 6.1.3 describing the .SBTTL directive). This argument does not affect the printing of the full assembly listing during assembly pass 2.</td>
</tr>
<tr>
<td>TTM</td>
<td>No list</td>
<td>Controls the listing output format. The default is set to line printer format. Figure 6-1 illustrates line printer output format; Figure 6-2 illustrates terminal output format.</td>
</tr>
</tbody>
</table>

¹If the .NLIST arguments SEQ, LOC, BIN, and SRC are in effect at the same time, that is, if all four significant fields in the listing are to be suppressed, the printing of the resulting blank line is inhibited.
If you use an argument in a .LIST/.NLIST directive other than those listed in Table 6–2, the directive is flagged with an error code (A) in the assembly listing.

You can also specify the listing control options at assembly time through qualifiers included in the command string to MACRO-11 (see Table 8–3 and/or the appropriate system manual). The use of these qualifiers overrides all corresponding listing control (.LIST or .NLIST) directives specified in the source program.

Figure 6–3 shows a listing produced in line printer format that shows the use of .LIST and .NLIST directives in the source program and the effects the directives have on the assembly listing output.

Figure 6–3: Listing Produced with Listing Control Directives

```
1 .TITLE LISTING CONTROL EXAMPLE

2 .LIST ME ;List macro expansions

3

4 ; Listing control test macro

5 ;

6 .MACRO LSTMAC ARG

7 .NLIST ARG

8 .WORD 1,2,3,4 ;This is a test comment

9 .LIST ARG

10 .ENDM

11

12 000000 .LSTMAC LOC ;Location counter test

13 000001 000002 000003 .WORD 1,2,3,4 ;This is a test comment

14 000004 .LIST LOC

15

16 000010 .NLIST BIN ;Generated binary test

17 000010 .WORD 1,2,3,4 ;This is a test comment

18 .LIST BIN

19 000020 .LSTMAC BEX ;Binary extensions test

20 000020 000001 000002 000003 .NLIST BEX ;This is a test comment

21 .WORD 1,2,3,4 .LIST BEX

22 .LSTMAC SRC ;Source lines test

23 000030 .LIST SRC

24 000030 000001 000002 000003

25 000036 000004

26 000040 .LSTMAC <COM,BEX> ;Comment lines test

27 000040 000001 000002 000003 .NLIST COM .WORD 1,2,3,4

28 000048 000004 .LIST COM

29 000050 .LSTMAC <COM,BEX> ;Comment lines and extended binary test

30 000050 000001 000002 000003 .NLIST COM .WORD 1,2,3,4

31 .LIST COM

32

33 Figure 6–3 Cont’d. on next page
```
6.1.2 .TITLE Directive

Format:

.TITLE string

where:

| string | represents an identifier of from one to six Radix-50 characters. The identifier can be followed by a string of one or more 7-bit ASCII or 8-bit DEC Multinational printing characters plus space and horizontal tab. Any MCS character must be preceded by six Radix-50 characters. Characters after the first $31_{10}$ do not appear in the title line of the listing. |

Section A.1 contains a table that includes all MCS characters. Section A.2 contains a table of Radix-50 characters.

The .TITLE directive assigns a name to the object module. The name assigned is the first six nonblank Radix-50 characters following the .TITLE directive. MACRO-11 ignores all spaces and/or tabs up to the first nonspace/nontab character following the .TITLE directive. Any characters beyond the first six Radix-50 characters are optional, and are checked only for MCS validity.

The name of an object module (specified in the .TITLE directive) appears in the load map produced at link time. This is also the module name which the Librarian will recognize.

If the .TITLE directive is not specified, MACRO-11 assigns the default name .MAIN. to the object module. If more than one .TITLE directive is specified in the source program, the last .TITLE directive encountered during assembly pass 1 establishes the name for the entire object module.

If the .TITLE directive is specified without an object module name, or if the first nonspace/nontab character in the object module name is not Radix-50 character, the directive is flagged with an error code (A) in the assembly listing; some combinations of invalid characters may also give a (Q) error.
6.1.3 .SBTTL Directive

Format:

.SBTTL string

where:

string represents an identifier of one or more 7-bit ASCII or 8-bit DEC Multinational printing characters plus space and horizontal tab. Only the first 80_10 characters appear in the subtitle line of the listing, although .SBTTL strings up to the full width of the line appear on the contents page.

The text strings following .SBTTL directives produce a table of contents listing immediately preceding the assembly listing. The text following each .SBTTL directive also prints as the second line of the header of each page in the listing following the .SBTTL directive. The subheading is listed until altered by a subsequent .SBTTL directive in the program. For example, the directive:

.SBTTL Conditional assemblies

prints the text:

Conditional assemblies

as the second line in the header of the assembly listing.

During assembly pass 1, a table of contents containing the line sequence number, the page number, and the text accompanying each .SBTTL directive is printed for the assembly listing. The listing of the table of contents is suppressed whenever a .NLIST TOC directive is encountered in the source program (see Table 6–2). An example of a table of contents listing is shown in Figure 6–4.

Figure 6–4: Assembly Listing Table of Contents

<table>
<thead>
<tr>
<th>MTTEMT - RT--11 MULTI-TTY EMT SE</th>
<th>MACRO V05.04 Tuesday 02-Jun-87 15:47</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of contents</td>
<td></td>
</tr>
<tr>
<td>50- 1</td>
<td>.MTOUT - Single character output EMT</td>
</tr>
<tr>
<td>51- 1</td>
<td>.MRCETO - Reset CTRL/O EMT</td>
</tr>
<tr>
<td>52- 1</td>
<td>.MTATCH - Attach to terminal EMT</td>
</tr>
<tr>
<td>54- 1</td>
<td>.MTDTON - Detach from a terminal EMT</td>
</tr>
<tr>
<td>55- 1</td>
<td>.MTPRINT - Print message EMT</td>
</tr>
<tr>
<td>56- 1</td>
<td>.MTSTAT - Return multi-terminal system status EMT</td>
</tr>
<tr>
<td>57- 1</td>
<td>MTIN - Single character input</td>
</tr>
<tr>
<td>58- 1</td>
<td>MTGET - Get a character from the ring buffer</td>
</tr>
<tr>
<td>59- 1</td>
<td>TTSET - Reset terminal status bits</td>
</tr>
<tr>
<td>60- 1</td>
<td>MTPUT - Single character output</td>
</tr>
<tr>
<td>62- 1</td>
<td>MTRSET - Stop and detach all terminals attached to a job</td>
</tr>
<tr>
<td>63- 1</td>
<td>ESCAPE SEQUENCE TEST SUBROUTINE</td>
</tr>
</tbody>
</table>
6.1.4 .IDENT Directive

Format:

    .IDENT /string/

where:

string represents a string of six or fewer Radix–50 characters which establish the program identification or version number. This string is included in the global symbol directory of the object module and is printed in the link map and Librarian listing.

/ ... / represent delimiting characters. These delimiters can be any paired printing characters other than the colon (:) and left angle bracket ( < ), as long as the delimiting character is not contained within the text string itself. The equal sign (=) and the semicolon (;) can be used with caution, as explained in Section 6.3.4. If the delimiting characters do not match, or if an invalid delimiting character is used, the directive is flagged with an error code (A) in the assembly listing.

In addition to the name assigned to the object module with the .TITLE directive (see Section 6.1.3), the .IDENT directive allows you to label the object module with the program version number.

An example of the .IDENT directive is shown below:

    .IDENT /V01.00/

The character string is converted to Radix–50 representation and is included in the global symbol directory of the object module. This character string also appears in the link map produced at link time and the Librarian directory listings.

When more than one .IDENT directive is encountered in a given program, the last such directive encountered establishes the character string which forms part of the object module identification1.

The RSX–11M Task Builder allows a .IDENT string for each module in the program. The Task Builder uses the first .IDENT directive in each module to establish the character string that will be identified with that module. Like the RT–11 Linker, the RSX–11M Task Builder uses the .IDENT directives encountered on the first pass.

6.1.5 .PAGE Directive/Page Ejection

Format:

    .PAGE

The .PAGE directive is used within the source program to start the listing on a new page at desired points in the listing. This directive takes no arguments and causes a skip to the top of the next page when encountered. It also increments the page number and (under RT–11) clears the line sequence counter. The .PAGE directive does not appear in the listing.

---

1 The RT–11 Linker allows only one .IDENT string in a program. The Linker uses the first .IDENT directive encountered during the first pass to establish the character string that will be identified with all of the object modules.
When used within a macro definition, the .PAGE directive is ignored during the assembly of the macro definition. Rather, the page eject operation is performed as the macro itself is expanded. In this case, the page number is also incremented. .PAGE directives in unexpanded macros are ignored.

Page ejection is accomplished in three other ways:

- After reaching a count of 58 lines in the listing, MACRO-11 automatically performs a page eject to skip over page perforations on line printer paper and to formulate terminal output into pages. The page number is not changed.

- A page eject is performed when a form feed character is encountered. If the form feed character appears within a macro definition, a page eject occurs during the assembly of the macro definition, but not during the expansion of the macro itself. A page eject resulting from the use of the form feed character increments the page number and (under RT-11) clears the line sequence counter.

- A page eject is performed when a new source file is encountered. In this case, the page number is incremented and the line sequence count reset.

If the listing is already at top-of-page, no action is taken.

### 6.1.6 .REM Directive/Begin Remark Lines

**Format:**

```
.REM comment-character
```

where:

```
  comment-character represents a 7-bit ASCII or 8-bit DEC Multinational character that marks the end of the comment block when the character recurs.
```

The .REM directive lets you insert a block of comments into a MACRO-11 source program without having to precede the comment lines with the comment character (;). The text between the specified delimiting characters is treated as comments. The comments can span any number of lines. The following example uses ampersand (&) as the delimiting character:

```
.TITLE Remark example
.REM &
   All the text that resides here is interpreted by MACRO-11 to be comment lines until another ampersand character is found. Any character can be used in place of the ampersand.&
CLR PC
.END
```

### 6.2 Function Directives

The following function directives are included in a source program to invoke or inhibit certain MACRO-11 functions and operations incidental to the assembly process itself.
6.2.1 .ENABL and .DSABL Directives

Format:

```
.ENABL    arg
.DSABL    arg
```

where:

- `arg` represents one or more of the optional symbolic arguments defined in Table 6–3.

If you specify any argument in a `.ENABL`/.`DSABL` directive other than those listed in Table 6–3, the line will be flagged with an error code (A) in the assembly listing.

### Table 6–3: Symbolic Arguments of Function Control Directives

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Disable</td>
<td>Enabling this function produces output in absolute binary (.LDA) format.</td>
</tr>
<tr>
<td>AMA</td>
<td>Disable</td>
<td>Enabling this function causes all relative addresses (address mode 67) to be assembled as absolute addresses (address mode 37). This function is useful during the debugging phase of program development.</td>
</tr>
<tr>
<td>CDR</td>
<td>Disable</td>
<td>Enabling this function causes source columns from 73 to the end of the line to be treated as a comment. The most common use of this feature is to permit sequence numbers in card columns 73 to 80.</td>
</tr>
<tr>
<td>CRF</td>
<td>Enable</td>
<td>Disabling this function inhibits the generation of cross-reference output. This function has meaning only if cross-reference output generation is specified in the command string.</td>
</tr>
<tr>
<td>FPT</td>
<td>Disable</td>
<td>Enabling this function causes floating-point truncation; disabling this function causes floating-point rounding.</td>
</tr>
<tr>
<td>GBL</td>
<td>Enable</td>
<td>Disabling this function causes MACRO–11 to mark all undefined references in assembly pass 2 with a (U) error in the assembly listing. The default for this option is Enable, so MACRO–11 normally treats all undefined symbol references as global, allowing the Linker to resolve them.</td>
</tr>
<tr>
<td>LC</td>
<td>Enable</td>
<td>Disabling this function causes MACRO–11 to convert all ASCII input to uppercase before processing it. An example of the <code>.ENABL LC</code> and <code>.DSABL LC</code> directives, as typically used in a source program, is shown in Figure 6–5.</td>
</tr>
<tr>
<td>LCM</td>
<td>Disable</td>
<td>Enabling this function causes the MACRO–11 conditional assembly directives <code>.IF IDN</code> and <code>.IF DIF</code> to be alphabetically case sensitive. By default, these directives are not case sensitive.</td>
</tr>
</tbody>
</table>
Table 6-3 (Cont.): Symbolic Arguments of Function Control Directives

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSB</td>
<td>Disable</td>
<td>This argument permits the enabling or disabling of a local symbol block. Although a local symbol block is normally established by encountering a new symbolic label, a .PSECT directive, or a .RESTORE directive in the source program, a .ENABL LSB directive establishes a new local symbol block which is not terminated until another .ENABL LSB is encountered, or another symbolic label, .PSECT directive, or .RESTORE directive is encountered following a paired .DSABL LSB directive. The basic function of this directive with regard to .PSECT is limited to those instances where it is desirable to leave a program section temporarily to store data, followed by a return to the original program section. This temporary dismissal of the current program section can also be accomplished through the .SAVE and .RESTORE directives (see Sections 6.7.3 and 6.7.4). Attempts to define local symbols in an alternate program section are flagged with an error code (P) in the assembly listing.</td>
</tr>
<tr>
<td>MCL</td>
<td>Disable</td>
<td>Enabling this function causes MACRO-11 to search all known macro libraries for a macro definition that matches any undefined symbols appearing in the op code field of a MACRO-11 statement. By default, this option is disabled. If MACRO-11 finds an unknown symbol in the op code field, it either declares a (U) undefined symbol error or declares the symbol an external symbol, depending on the .ENABL/.DSABL setting of GBL.</td>
</tr>
<tr>
<td>PNC</td>
<td>Enable</td>
<td>Disabling this function inhibits binary output until a .ENABL PNC statement is encountered within the same module.</td>
</tr>
</tbody>
</table>
| REG      | Enable  | Disabling this function inhibits the normal MACRO-11 default register definitions. The default register definitions are listed below:  

\[
\begin{align*}
R0 &= %0 \\
R1 &= %1 \\
R2 &= %2 \\
R3 &= %3 \\
R4 &= %4 \\
R5 &= %5 \\
SP &= %6 \\
PC &= %7
\end{align*}
\]

The .ENABL REG directive can be used as the logical complement of the .DSABL REG directive. The use of these directives, however, is not recommended. For logical consistency, use the normal default register definitions listed above. |
6.2.2 Cross-Reference Directives: .CROSS and .NOCROSS

Format:

.CROSS
.CROSS sym1,sym2,...symn
.NOCROSS
.NOCROSS sym1,sym2,...symn

where:

sym1, sym2,...symn represent valid symbolic names. When multiple symbols are specified, they are separated by any valid separator (comma, space, and/or tab).

The .CROSS and the .NOCROSS directives control which symbols are included in the cross-reference listing produced by the MACRO-11 assembler. These directives have an effect only if the /C[R] or the /CROSS qualifier was used in the command line to select the cross-reference capability.

By default, the cross-reference listing includes the definition and all the references to every user symbol in the module. The cross-reference listing can be disabled for all symbols or for a specified list of symbols.

When the .NOCROSS directive is used without a symbol list, the cross-reference listing of all the symbols in the module is disabled. The cross-reference listing of all the symbols in the module is reenabled when the .CROSS directive is used without a symbol list. Any symbol definition or reference that appears after a .NOCROSS directive that is used without a symbol list and before the next .CROSS directive that is used without a symbol list is excluded from the cross-reference listing.
The .NOCROSS directive used with a symbol list disables the cross-reference listing for the listed symbols. When the .CROSS directive is used with a symbol list, the cross-reference listing of the listed symbols is reenabled.

In the following example, the definition of LABEL1 and the reference to LOC1 and LOC2 are not included in the cross-reference listing.

```
.NOCROSS
LABEL1: MOV LOC1,LOC2 ;Stop cross reference
       .CROSS ;Reenable cross reference
```

In the next example, the definition of LABEL2 and the reference to LOC2 are included in the cross reference, but the reference to LOC1 is not included.

```
.NOCROSS LOC1 ;Do not cross reference LOC1
LABEL2: MOV LOC1,LOC2 ;Copy data
       .CROSS LOC1 ;Reenable cross reference
```

The .CROSS directive used without a symbol list cannot be used to reenable the cross-reference listing of a symbol specified in the symbol list of a .NOCROSS directive. In addition, if the cross-reference listing of all the symbols in a module is disabled, the .CROSS directive used with a symbol list will have no effect until the cross-reference listing is reenabled by the .CROSS directive used without a symbol list.

The .CROSS directive with no symbol list is equivalent to the .ENABL CRF directive, and the .NOCROSS directive with no symbol list is equivalent to the .DSABL CRF directive.

### 6.3 Data Storage Directives

A wide range of data and data types can be generated with the directives, ASCII conversion characters, and radix-control operators described in the following sections.

#### 6.3.1 .BYTE Directive

Format:

```
.BYTE exp ;Stores the binary value of the
           ;expression in the next byte.
.BYTE exp1,exp2,expn ;Stores the binary values of the list
                       ;of expressions in successive bytes.
```

where:

- `exp1, exp2,...,expn` represent expressions that must be reduced to eight bits of data or less. Each expression will be read as a 16-bit word expression, the high-order byte to be truncated. The high-order byte must be all zeros, or a (A) error results. Multiple expressions must be separated by commas.

The .BYTE directive stores successive bytes of binary data in the object module.
Example:

```
SAM = 5
.=1410
.BYTE D48, SAM ; The value 060 (octal equivalent of 48
; decimal) is stored in location 1410.
; The value 005 is stored in location
; 1411.
```

The construction `D` in the first operand of the .BYTE directive above illustrates the use of a temporary radix-control operator. The function of these special unary operators is described in Section 6.4.1.2. At link time, it is likely that a relocatable expression will result in a value having more than eight bits, in which case the Linker or Task Builder issues a truncation (T) error for the object module in question. For example, the following statements create such a possibility:

```
A:
.BYTE 23 ; Stores octal 23 in next byte.
=1420
.BYTE ... ; Relocatable value A will probably
; cause truncation error.
```

If an expression following the .BYTE directive is null, it is interpreted as a zero:

```
=1420
.BYTE ... ; Zeros are stored in bytes 1420, 1421,
; 1422, and 1423.
```

In the above example, four bytes of storage result from the .BYTE directive. The three commas in the operand field represent an implicit declaration of four null values, each separated from the other by a comma. Hence, four bytes, each containing a value of zero (0), are reserved in the object module.

### 6.3.2 .WORD Directive

**Formats:**

```
.WORD exp ; Stores the binary equivalent of the
; expression in the next word.
.WORD exp1, exp2, expn ; Stores the binary equivalents of the
; list of expressions in successive words.
```

**where:**

```
exp1, exp2, ..., expn
```

represent expressions that must reduce to 16 bits of data or less. Multiple expressions must be separated by commas.

The .WORD directive stores successive words of data in the object module.

**Example:**

```
SAL = 0
.=1500
.WORD 177535, +4, SAL ; Stores the values 177535, 1506, and
; 0 in words 1500, 1502, and 1504,
; respectively.
```

If an expression following the .WORD directive contains a null value, it is interpreted as a zero, as shown in the following example:
.=1500
.WORD .5,
;Stores the values 0, 5, and 0 in
;location 1500, 1502, and 1504,
;respectively.

A statement with a blank operator field (one that contains a symbol other than a macro call, an instruction mnemonic, a MACRO-11 directive, or a semicolon) is interpreted during assembly as an implicit .WORD directive, as shown in the example below:

.=1440
LABEL: 100, LABEL
;Stores the value 100 in location 1440
;and the value 1440 in location 1442.

NOTE
You should not use this technique to generate .WORD directives because it may not be included in future PDP-11 assemblers.

6.3.3 ASCII Conversion Characters

The single quote ('') and the double quote ('"') characters are unary operators that can appear in any MACRO-11 expression. Used in MACRO-11 expressions, these characters generate a 16-bit expression value.

When the single quote is used, MACRO-11 takes the next character in the expression and converts it from its 7-bit ASCII or 8-bit DEC Multinational character set value to a 16-bit expression value. The high-order byte of the resulting expression value is always zero (0). The 16-bit value is then used as an absolute term within the expression. For example, the statement:

    MOV #A, R0

moves the 16-bit binary expression value:

00000000 01000001

into register 0. (010000012 is the binary value of ASCII A.)

Thus, the expression 'A results in a value of 1018.

The single quote ('') character must not be followed by a carriage return, null, RUBOUT, line feed, or form feed character; if it is, an error code (A) is generated in the assembly listing. When the double quote is used, MACRO-11 converts the next two characters in the expression to a 16-bit binary expression value from their 7-bit ASCII or 8-bit DEC Multinational values. This 16-bit value is then used as an absolute term within the expression. For example, the statement:

    MOV #"AB, R0

moves the 16-bit expression value:

01000010 01000001

into register 0. (01000010010000012 is the concatenated binary byte values of the ASCII characters A and B.)
Thus, the expression "AB results in a value of 04110108.

The double quote ("), character, like the single quote (') character, must not be followed by a carriage-return, null, RUBOUT, line-feed, or form-feed character; if it is, an error code (A) is generated in the assembly listing.

The DEC Multinational character set is listed in Section A.1.

6.3.4 .ASCII Directive

Format:

```
.ASCII /string 1/.../string n/
```

where:

```
string
```

is a string of 7-bit ASCII or 8-bit DEC Multinational printing characters, plus space and horizontal tab. All nonprinting characters except carriage return and form feed cause an error code (1) if used in a .ASCII string. Carriage return and form feed characters are flagged with an error code (A) because they end the scan of the line, preventing MACRO-11 from detecting the matching delimiter at the end of the character string.

```
/.../
```

represent delimiting characters. These delimiters can be any paired printing characters other than the colon (:) and left angle bracket (<), as long as the delimiting character is not contained within the text string itself. The equal sign (=) and the semicolon (;) can be used with caution, as explained below. If the delimiting characters do not match, or if an invalid delimiting character is used, the directive is flagged with an error code (A) in the assembly listing.

The .ASCII directive translates character strings into their 7-bit ASCII or 8-bit DEC Multinational equivalents and stores them in the object module. A nonprinting character can be expressed only by enclosing its equivalent octal value within angle brackets. Each set of angle brackets so used represents a single character. For example, in the following statement:

```
.ASCII <15>/ABC/<A+2>/DEF/<5><4>
```

the expressions <15>, <A+2>, <5>, and <4> represent the values of nonprinting characters. Each bracketed expression must reduce to eight bits of absolute data or less. The expression cannot contain any global symbols.

Angle brackets can be embedded between delimiting characters in the character string, but angle brackets so used do not take on their usual significance as delimiters for nonprinting characters. For example, the statement:

```
.ASCII /ABC<expression>DEF/
```

contains a single ASCII character string, and performs no evaluation of the embedded, bracketed expression. This use of the angle brackets is shown in the third example of the .ASCII directive below:

```
.ASCII /HELLO/ ;Stores the binary representation of the letters HELLO in five consecutive bytes.
```
.ASCII /ABC/<15><12>/DEF/ ; Stores the binary representation of the characters A, B, C, carriage return, line feed, D, E, F in eight consecutive bytes.

.ASCIIZ /A<15>B/

; Stores the binary representation of the characters A, <, 1, 5, >, and B in six consecutive bytes.

The colon (:) character can never be used as a delimiting character. The semicolon (;) and equal sign (=) can be used as delimiting characters in the string, but care must be exercised in so doing because of their significance as a comment indicator and assignment operator, respectively, as illustrated in the examples below:

.ASCIIZ ;ABC;/DEF/

; Stores the binary representation of the characters A, B, C, D, E, and F in six consecutive bytes; not recommended practice.

.ASCIIZ /ABC;/DEF;

; Stores the binary representations of the characters A, B, and C in three consecutive bytes; the characters D, E, F, and ; are treated as a comment.

.ASCIIZ /ABC/=DEF=

; Stores the binary representation of the characters A, B, C, D, E, and F in six consecutive bytes; not recommended practice.

An equal sign is treated as an assignment operator when it appears as the first character in the ASCII string, as illustrated by the following example:

.ASCIIZ =DEF=

; The direct assignment operation .ASCIIZ=DEF is performed, and a syntax error (Q) is generated upon encountering the second = sign.

6.3.5 .ASCIIZ Directive

Format:

.ASCIIZ /string 1/.../string n/

where:

string is a string of 7-bit ASCII or 8-bit DEC Multinational printing characters, plus space and horizontal tab. All nonprinting characters except carriage return and form feed cause an error code (I) if used in a .ASCII string. Carriage return and form feed characters are flagged with an error code (A) because they end the scan of the line, preventing MACRO-11 from detecting the matching delimiter at the end of the character string.
represent delimiting characters. These delimiters can be any paired printing characters other than the colon (:) and left angle bracket (<), as long as the delimiting character is not contained within the text string itself. The equal sign (=) and the semicolon (;) can be used with caution, as explained in Section 6.3.4. If the delimiting characters do not match, or if an invalid delimiting character is used, the directive is flagged with an error code (A) in the assembly listing.

The .ASCIZ directive is similar to the .ASCII directive described above, except that a zero byte is automatically inserted as the final character of the string. Thus, when a list or text string has been created with a .ASCIZ directive, a search for the null character in the last byte can effectively determine the end of the string, as reflected in the example below:

```
CR=15
LF=12
HELLO: .ASCIZ <CR><LF>/MACRO--11 V05.00/<CR><LF>; Introductory message
.EVEN

MOV  #HELLO,R1 ; Get address of message.
MOV  #LINBUF,R2 ; Get address of output buffer.
10$: MOVB (R1)+,(R2)+ ; Move a byte to output buffer.
    BNE  10$ ; If not null, move another byte.
```

### 6.3.6 .RAD50 Directive

**Format:**

```
.RAD50 /string 1/.../string n/
```

**where:**

- string represents a series of characters to be packed. The string must consist of the characters A through Z, 0 through 9, dollar sign ($), period (.), and space ( ). An invalid printing character causes an error flag (Q) to be printed in the assembly listing.

If fewer than three characters are to be packed, the string is packed left-justified within the word, and trailing spaces are assumed.

All nonprinting characters except carriage return and form feed cause an error code (I) if used in a .ASCII string. Carriage return and form feed characters are flagged with an error code (A) because they end the scan of the line, preventing MACRO-11 from detecting the matching delimiter at the end of the character string.
represent delimiting characters. These delimiters can be any paired printing characters other than the colon (:) and left angle bracket (<), as long as the delimiting character is not contained within the text string itself. The equal sign (=) and the semicolon (;) can be used with caution, as explained in Section 6.3.4. If the delimiting characters do not match, or if an invalid delimiting character is used, the directive is flagged with an error code (A) in the assembly listing.

The .RAD50 directive generates data in Radix-50 packed format. Radix-50 form allows three characters to be packed into 16 bits (one word); therefore, any 6-character symbol can be stored in two consecutive words. Examples of .RAD50 directives are shown below:

```
.RAD50 /ABC/ ;Packs ABC into one word.
.RAD50 /AB/ ;Packs AB (SPACE) into one word.
.RAD50 /ABCD/ ;Packs ABC into first word and
;D (SPACE) (SPACE) into second word.
.RAD50 /ABCDEF/ ;Packs ABC into first word, DEF into
;second word.
```

Each character is translated into its Radix-50 equivalent, as indicated in the following table:

<table>
<thead>
<tr>
<th>Character</th>
<th>Radix-50 Octal Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>(space)</td>
<td>0</td>
</tr>
<tr>
<td>A–Z</td>
<td>01–32</td>
</tr>
<tr>
<td>$</td>
<td>33</td>
</tr>
<tr>
<td>.</td>
<td>34</td>
</tr>
<tr>
<td>(undefined)</td>
<td>35</td>
</tr>
<tr>
<td>0–9</td>
<td>36–47</td>
</tr>
</tbody>
</table>

The Radix-50 equivalents for characters 1 through 3 (C1,C2,C3) are combined as follows:

Radix-50 value = ((C1*50₈)+C2)*50₈+C3

For example:

Radix-50 value of ABC = ((1*50₈)+2)*50₈+3 = 3223₈

Refer to Section A.2 for a table of Radix-50 equivalents.

Angle brackets (<> ) must be used in the .RAD50 directive whenever special codes are to be inserted in the text string, as shown in the example below:
.RAD50 /AB/<35> ;Stores 3255 in one word.
CHR1=1
CHR2=2
CHR3=3
.
.
.RAD50 <CHR1><CHR2><CHR3> ;Equivalent to .RAD50 /ABC/.

6.3.7 Temporary Radix-50 Control Operator

Format:
```
^Rccc
```

where:

```
ccc
```

represents a maximum of three characters to be converted to a 16-bit Radix-50 value. If more than three characters are specified, any following the third character are ignored. If fewer than three are specified, the trailing characters are assumed to be blanks.

The ^R operator converts its argument to Radix-50 format. This allows up to three characters to be stored in one word. If you use ^R with no argument, MACRO-11 generates a word of 0.

The following example shows how the ^R operator might be used to pack a 3-character file type specifier (MAC) into a single 16-bit word.

```
MOV #^RMAC,FILEXT ;Store RAD50 MAC as file extension
```

The number sign (#) indicates immediate data (data to be assembled directly into object code). ^R specifies that the characters MAC are to be converted to Radix-50. This value is then stored in location FILEXT.

6.3.8 .PACKED Directive

Format:
```
.PACKED decimal-string[,symbol]
```

where:

```
decimal-string
```

represents a decimal number from 0 to 3110 digits long. Each digit must be in the range 0 to 9. The number can have a sign, but it is not required and is not counted as a digit in the total of 3110.

```
symbol
```

is assigned a value equivalent to the number of decimal digits in the string.

The .PACKED directive generates packed decimal data, four bits per digit (two digits per byte) plus a 4-bit sign designator. The sign designator can have one of three values:
1100_2    Positive
1101_2    Negative
1111_2    Unsigned

Arithmetic and operational properties of packed decimals are similar to those of numeric strings. Figure 6–6 is an example of the .PACKED directive.

**Figure 6–6:** Example of the .PACKED Directive

```
   .LIST BEX
1   000000  017   .PACKED  0,  UOLEN
2   000001  014   .PACKED +0,  POLEN
3   000002  015   .PACKED -0,  NOLEN
4   000003  037   .PACKED  1,  U1LEN
5   000004  034   .PACKED +1,  P1LEN
6   000005  035   .PACKED -1,  N1LEN
7   000006  001   .PACKED 12,  U12LEN
8   000007  057
9   000010  001   .PACKED +12,  P12LEN
600011  054
10  000012  001   .PACKED -12,  N12LEN
600013  055
11  000014  001   .PACKED 1234567890,  UXLEN
600015  043
600016  105
600017  147
600020  211
600021  017
12  000022  001   .PACKED +1234567890,  PXLEN
600023  043
600024  105
600025  147
600026  211
600027  014
13  000030  001   .PACKED -1234567890,  NXLEN
600031  043
600032  105
600033  147
600034  211
600035  015
14   .EVEN
15
16  000001   .END
```

Symbol table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXLEN</td>
<td>000012</td>
</tr>
<tr>
<td>NOLEN</td>
<td>000001</td>
</tr>
<tr>
<td>N1LEN</td>
<td>000001</td>
</tr>
<tr>
<td>N12LEN</td>
<td>000002</td>
</tr>
<tr>
<td>PXLEN</td>
<td>000012</td>
</tr>
<tr>
<td>POLEN</td>
<td>000001</td>
</tr>
<tr>
<td>UOLEN</td>
<td>000001</td>
</tr>
<tr>
<td>U1LEN</td>
<td>000001</td>
</tr>
<tr>
<td>U12LEN</td>
<td>000002</td>
</tr>
</tbody>
</table>
6.4 Radix and Numeric Control Facilities

6.4.1 Radix Control and Unary Control Operators

Any numeric or expression value in a MACRO–11 source program is read as an octal value by default. Occasionally, however, an alternate radix is useful. By using the MACRO–11 facilities described below, you can declare a radix to affect a term or an entire program depending on your needs.

**NOTE**

When two or more unary operators appear together, modifying the same term, the operators are applied to the term from right to left.

6.4.1.1 .RADIX Directive

Format:

```
.RADIX  n
```

where:

```
  n
```

represents one of the radices 2, 8, 10, or 16. Any value other than null or one of the acceptable radices is flagged with an error code (A) in the assembly listing. If no argument is specified, the octal default radix is assumed. The argument (n) is always read as a decimal value.

Numbers used in a MACRO–11 source program are initially assumed to be octal values; however, with the .RADIX directive you can declare alternate radices applicable throughout the source program or within specific portions of the program.

Any alternate radix declared in the source program through the .RADIX directive remains in effect until altered by the occurrence of another such directive, for example:

```
.RADIX 10 ;begins a section of code having a
decimal radix.
.RADIX ;reverts to octal radix.
```

In general, macro definitions should not contain or rely on radix settings established with the .RADIX directive. Rather, temporary radix control operators should be used within a macro definition. Where a possible radix conflict exists within a macro definition or source program, specify numeric or expression values using the temporary radix control operators described below.

**NOTE**

All hexadecimal values used with .RADIX 16 must begin with a digit, which can be 0. For example, the hexadecimal value F3 must be written as 0F3. Otherwise, MACRO–11 assumes the item is a symbolic name, not a hexadecimal number.
6.4.1.2 Temporary Radix Control Operators

Formats:

~Bn ; n is evaluated as a binary number
~Dn ; n is evaluated as a decimal number
~On ; n is evaluated as an octal number
~Xn ; n is evaluated as a hexadecimal number

These unary operators establish an alternate radix for a single term. A temporary alternate is useful because, after you have specified a radix for a section of code or have decided to use the default octal radix, you may discover a number of cases where an alternate radix is more convenient or desirable (particularly within macro definitions). Creating a mask word (used to check bit status), for example, might be accomplished best through the use of a binary radix.

An alternate radix can be declared temporarily to meet a localized requirement in the source program. The temporary radix control operator can be used any time regardless of the radix in effect or other radix declarations within the program. Because the operator affects only the term immediately following it, it can be used anywhere a numeric value is valid. The term (or expression) associated with the temporary radix control operator is evaluated during assembly as a 16-bit entity.

The expressions below are representative of the methods of specifying temporary radix control operators:

~D123 Decimal Radix
~0 47 Octal Radix
~B 0001101 Binary Radix
~0<A+13> Octal Radix
~X0F3 Hexadecimal Radix

The circumflex and the radix control operator cannot be separated, but the radix control operator and the following term or expression can be separated by spaces or tabs for legibility or formatting. A multielement term or expression that is to be interpreted in an alternate radix should be enclosed within angle brackets, as shown in the last of the four temporary radix control expressions above.

The following example also illustrates the use of angle brackets to delimit an expression that is to be interpreted in an alternate radix. When the temporary radix control operator is used, only numeric values are affected. Any symbols used with the operator are evaluated with respect to the radix in effect at their declaration:

```
.RADIX 10
A=10
.WORD ~0<A+10>+10
```

When the temporary radix expression in the .WORD directive above is evaluated, it yields the following equivalent statement:

```
.WORD 180
```
MACRO-11 also allows a temporary radix change to decimal by specifying a number immediately followed by a decimal point (.), as shown below:

100. ;Equivalent to 144(octal)
1376. ;Equivalent to 2540(octal)
128. ;Equivalent to 200(octal)

The above expression forms are equivalent in function to:

```
"D100
"D1376
"D128
```

**NOTE**

All hexadecimal values used with `"x` must begin with a digit, which can be 0. For example, the hexadecimal value F3 must be written as 0F3. Otherwise, MACRO-11 assumes the item is a symbolic name, not a hexadecimal number.

### 6.4.2 Numeric Directives and Unary Control Operators

Two storage directives and two numeric control operators are available to simplify the use of the floating-point hardware on the PDP-11. These facilities allow floating-point data to be created in the program, and numeric values to be complemented or treated as floating-point numbers.

A floating-point number is represented by a string of one or more decimal digits. The string can contain an optional decimal point and can be followed by an optional exponent indicator in the form of the letter E and a signed decimal integer exponent. The number cannot contain embedded blanks, tabs, or angle brackets and cannot be an expression; such a string will result in one or more errors (A and/or Q) in the assembly listing.

The list of numeric representations below contains seven distinct, valid representations of the same floating-point number:

```
3
3.
3.0
3.0E0
3E0
.3e1
300E-2
```

As can be inferred, the list could be extended indefinitely (3000E-3, .03E2, and so on). A leading plus sign is optional (3.0 is considered to be +3.0). A leading minus sign complements the sign bit. No other operators are allowed; for example, 3.0+N is invalid.
All floating-point numbers are evaluated as 64 bits in the following format:

```
63 62  55 54   0
S  EEEEEEEE MMM MMM
```

Sign (1 bit)
Exponent (8 bits)
Mantissa (55 bits)

MACRO–11 returns a value of the appropriate size and precision by means of floating-point directives. The value returned can be truncated or rounded (see Section 6.2.1).

Floating-point numbers are normally rounded. That is, when a floating-point number exceeds the limits of the field in which it is to be stored, the high-order bit of the unretained word is added to the low-order bit of the retained word, as shown below. For example, if the number is to be stored in a 2-word field, but more than 32 bits are needed to express its exact value, the highest bit (32) of the unretained field is added to the least significant bit (0) of the retained field (see illustration below). The .ENABLE FPT directive is used to enable floating-point truncation; .DISABLE FPT is used to return to floating-point rounding (see Table 6–3).

```
32 032 0
Retained field Unretained field
```

All numeric operands associated with Floating Point Processor instructions are automatically evaluated as single-word, decimal, floating-point values unless a temporary radix control operator is specified. For example, to add (floating) the constant \(41040_{8}\) to the contents of floating accumulator zero, the following instruction must be used:

```
ADDF #^041040,F0
```

where:

\(F0\)

is assumed to represent floating accumulator zero.

Floating-point numbers are described in greater detail in the PDP–11 Processor Handbook.

### 6.4.2.1 One's Complement Operator: `C`

The `C` unary operator complements an argument as it is evaluated during assembly.

As with the radix control operators such as `D` and `O`, the numeric control operator `C` can be used anywhere in the source program that an expression value is valid. Such a construction is evaluated by MACRO–11 as a 16-bit binary value before being complemented. For example, the following statement:

```
TAG4: .WORD ^C151
```

stores the one's complement of the value \(151_{8}\) as a 16-bit value in the program. The resulting value expressed in octal form is \(17762\)\(_{8}\).
Because the ~c construction is a unary operator, the operator and its argument are regarded as a term. Thus, more than one unary operator can be applied to a single term. For example, the following construction:

`~c'0D25`

complements the value 25₁₀ during assembly. The resulting binary value, when expressed in octal form, reduces to 177746₈.

The term created through the use of the temporary numeric control operator can be used alone or in combination with other expression elements. For example, the following construction:

`~c2+6`

is equivalent in function to:

`~c2>6`

This expression is evaluated during assembly as the one's complement of 2, plus the absolute value of 6. When these terms are combined, the resulting expression value generates a carry beyond the most significant bit, leaving 000000₈ as the reduced value.

6.4.2.2 Floating-Point Storage Directives

Format:

```
.FR2    arg1, arg2, ... argn
.FR4    arg1, arg2, ... argn
```

where:

```
arg1, arg2, ... argn    represent one or more floating-point numbers as described in
                       Section 6.4.2. Multiple arguments must be separated by commas.
```

.FR2 generates two words of storage for each argument, while .FR4 generates four words of storage for each argument. As in the .WORD directive, the arguments are evaluated and the results stored in the object module.

6.4.2.3 Floating-Point Operator: ~F

The ~F unary operator for numeric control lets you specify an argument that is a 1-word floating-point number. For example, the following statement:

```
A:      MOV    #~F3.7,R0
```

creates a 1-word floating-point number at location A+2 containing the value 3.7 formatted as shown below:

```
[1][1][4]  7 6 0
S=EEEEEEEEMMMMM
```

Sign (1 bit)

Exponent (8 bits)

Mantissa (7 bits)
The importance of ordering with respect to unary operators is shown below:

```
^F1.0  = 040200
^F-1.0 = 140200
^-F1.0 = 137600
^-F-1.0 = 037600
```

The value created by the `^-F` unary operator and its argument is, like `^-C` and its argument, a term that can be used by itself or in an expression. For example:

```
^-C^F6.2
```

is equivalent to:

```
^-C<^-F6.2>
```

Expressions used as terms or arguments of a unary operator must be explicitly grouped. As illustrated above and in Section 6.4.2.1, when a temporary numeric control operator and its argument are coded as a term within an expression, angle brackets should be used as delimiters to ensure precise evaluation and readability.

### 6.5 Location Counter Control Directives

The directives used in controlling the value of the current location counter and in reserving storage space in the object program are described in the following sections.

Several MACRO–11 statements (listed below) may allocate an odd number of bytes:

- `.BYTE` directive
- `.BLKB` directive
- `.ASCII` or `.ASCIZ` directive
- `.ODD` directive
- `.PACKED` directive
- A direct assignment statement of the form `.=`*expression*, which results in the assignment of an odd address value.

In cases that yield an odd address value, the next instruction on a word boundary automatically forces the location counter to an even value, but that instruction is flagged with an error code (B) in the assembly listing.

#### 6.5.1 .EVEN Directive

Format:

```
.EVEN
```

The `.EVEN` directive ensures that the current location counter contains an even value by adding 1 if the current value is odd. If the current location counter is already even, no action is taken. Any operands following a `.EVEN` directive are flagged with an error code (Q) in the assembly listing.
The .EVEN directive is used as follows:

```
.ASCIIZ /This is a test/
.EVEN ;Ensures that the next statement will
        ;begin on a word boundary.
.WORD XYZ
```

### 6.5.2 .ODD Directive

**Format:**

```
.ODD
```

The .ODD directive ensures that the current location counter contains an odd value by adding 1 if the current value is even. If the current location counter is already odd, no action is taken. Any operands following a .ODD directive are flagged with an error code (Q) in the assembly listing.

### 6.5.3 .BLKB and .BLKW Directives

**Format:**

```
.BLKB exp
.BLKW exp
```

where:

- **exp** represents the specified number of bytes or words to be reserved in the object program. Any expression that is defined at assembly time and that reduces to an absolute value is valid. If the expression specified in either of these directives is not an absolute value, the statement is flagged with an error code (A) in the assembly listing. Furthermore, if the expression contains a forward reference (a reference to a symbol that is not previously defined), MACRO-11 generates incorrect object file code and may cause statements following the .BLKB/.BLKW directive to be flagged with phase (P) errors. These directives should not be used without arguments. However, if no argument is present, a default value of 1 is assumed.

The .BLKB directive reserves byte blocks in the object module; the .BLKW directive reserves word blocks. Figure 6–7 illustrates the use of the .BLKB and .BLKW directives.
Figure 6–7: Example of .BLKB and .BLKW Directives

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>; Illustrate use of .BLKB and .BLKW directives</td>
</tr>
<tr>
<td>2</td>
<td>;+</td>
</tr>
<tr>
<td>3</td>
<td>:=</td>
</tr>
<tr>
<td>4</td>
<td>000000 .PSECT IMPURE,D,GBL,RW</td>
</tr>
<tr>
<td>5</td>
<td>COUNT: .BLKW 1 ;Character counter</td>
</tr>
<tr>
<td>6</td>
<td>000000 MESSAG: .BLKB 80. ;Message text buffer</td>
</tr>
<tr>
<td>7</td>
<td>000002 CHRSAV: .BLKB ;Saved character</td>
</tr>
<tr>
<td>8</td>
<td>000122 FLAG: .BLKB ;Flag byte</td>
</tr>
<tr>
<td>9</td>
<td>000123 MAGPTR: .BLKW ;Message buffer pointer</td>
</tr>
</tbody>
</table>

The .BLKB directive in a source program has the same effect as the following statement:

```
.=.+expression
```

which adds the value of the expression to the current value of the location counter. The .BLKB directive, however, is easier to interpret in the context of the source code in which it appears and is therefore recommended.

### 6.5.4 .LIMIT Directive

**Format:**

```
.LIMIT
```

To know the upper and lower address boundaries of the image is often desirable. When the .LIMIT directive is specified in the source program, MACRO-11 generates the following instruction:

```
.BLKW 2
```

and reserves two storage words in the object module. Later, at link time, the lowest address in the load image (the initial value of SP) is inserted into the first reserved word, and the address of the first free word following the image is inserted into the second reserved word.

During linking, the size of the image is rounded upward to the nearest 2-word boundary.
6.6 Terminating Directive: .END Directive

Format:

.END [exp]

where:

exp represents an optional expression value which, if present, indicates the program-entry point, which is the transfer address where the program begins.

When MACRO-11 encounters a valid occurrence of the .END directive, it terminates the current assembly pass. Any text beyond this point in the current source file, or in additional source files identified in the command line, is ignored.

When an image consisting of several object modules is created, only one object module can be terminated with a .END exp statement (where exp is the starting address). All other object modules must be terminated with a .END statement (where .END has no argument); otherwise, an error message will be issued at link time. If no starting address is specified in any of the object modules, image execution begins at location 1 of the image and immediately faults because of an odd addressing error.

The .END statement must not be used within a macro expansion or a conditional assembly block; if it is so used, it is flagged with an error code (O) in the assembly listing. The .END statement can be used, however, in an immediate conditional statement (see Section 6.9.3).

If the source program input is not terminated with a .END directive, an error code (E) results in the assembly listing.

6.7 Program Sectioning Directives

The MACRO-11 program sectioning directives declare names for program sections (p-sections) and establish certain program section attributes essential to linking.
6.7.1 .PSECT Directive

Format:

.PSECT name, arg1, arg2,..., argn

where:

name represents the symbolic name of the program section, as described in Table 6–4.

., represents any valid separator (comma, tab and/or space).

arg1, arg2,..., argn represent one or more of the valid symbolic arguments defined for use with the .PSECT directive, as described in Table 6–4. The slash separating each pair of symbolic arguments listed in the table indicates that one or the other, but not both, can be specified. Multiple arguments must be separated by a valid separating character. Any symbolic argument specified in the .PSECT directive other than those listed in Table 6–4 will be flagged with an error code (A) in the assembly listing.

Table 6–4: Symbolic Arguments of .PSECT Directive

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Blank</td>
<td>Establishes the program section name, which is specified as one to six Radix–50 characters. If this argument is omitted, a comma must appear in place of the name parameter. The Radix–50 character set is listed in Section A.2.</td>
</tr>
<tr>
<td>RO/RW</td>
<td>RW</td>
<td>Defines which type of access is permitted to the program section:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RO = Read-Only Access</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RW = Read/Write Access</td>
</tr>
<tr>
<td>I/D</td>
<td>I</td>
<td>RT–11 and RSX–11M use only Read/Write access.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defines the contents of the program section:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I = Instructions. If a p-section has the I attribute and the program is overlaid, all calls to the p-section are referenced through a body of overlay code stored in the root.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If a concatenated p-section has the I attribute, code is concatenated on even bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D = Data. If a p-section has the D attribute, all calls to the p-section are referenced directly.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If a concatenated p-section has the D attribute, code is concatenated on the next byte regardless of whether the byte is odd or even.</td>
</tr>
</tbody>
</table>

1Where two possible arguments are separated by a slash (/), you can choose one or the other.
Table 6-4 (Cont.): Symbolic Arguments of .PSECT Directive

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBL/LCL(^1)</td>
<td>LCL</td>
<td>Defines the scope of the program section, as it will be interpreted at link time:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>LCL = Local.</strong> If an object module contains a local program section, then the storage allocation for that module will remain in the segment containing the module. Many modules can contribute (allocate memory) to this same program section; the memory allocation for each contributing module is either concatenated or overlaid within the segment, depending on the allocation argument of the program section (see CON/OVR below).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>GBL = Global.</strong> If a global program section is used in more than one segment of a program, all references to the p-section are collected across segment boundaries. The program sections are then stored in the segment (of those originally containing the p-sections) that is nearest the root.</td>
</tr>
<tr>
<td>RT-11 store</td>
<td></td>
<td>RT-11 stores the collected p-sections in the root.</td>
</tr>
<tr>
<td>ABS/REL(^1)</td>
<td>REL</td>
<td>Defines the relocatability attribute of the program section:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>ABS = Absolute (non-relocatable).</strong> The ABS argument causes the Linker or Task Builder to treat the p-section as an absolute module; therefore, no relocation is required. The program section is assembled and loaded, starting at absolute virtual address 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The location of data in absolute program sections must fall within the virtual memory limits of the segment containing the program section; otherwise, an error results at link time. For example, the following code, although valid during assembly, may generate an error message (A) if virtual location 100000 is outside the segment’s virtual address space:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.PSECT  ALPHA,ABS *   = .+100000 *   .WORD   X *</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>REL = Relocatable.</strong> The REL argument causes the Linker or Task Builder to treat the p-section as a relocatable module and a relocation bias is added to all location references within the program section making the references absolute.</td>
</tr>
</tbody>
</table>

\(^1\) Where two possible arguments are separated by a slash (/), you can choose one or the other.
Table 6–4 (Cont.): Symbolic Arguments of .PSECT Directive

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| CON/OVR\(^1\) | CON     | Defines the allocation requirements of the program section:  
CON = Concatenated. All references to one program section are concatenated to determine the total memory space needed for the p-section.  
OVR = Overlaid. All references to one program section are overlaid; the total memory space needed equaling the largest, individual p-section. |
| SAV/NOSAV\(^1\) | NOSAV   | Determines where the Linker allocates storage for the program section:  
SAV = Save. The Linker always forces allocation for the program section to the root of the image.  
NOSAV = No Save. The Linker allocates the program section normally. |

\(^1\)Where two possible arguments are separated by a slash (/), you can choose one or the other.

NAME is the only position-dependent argument for the .PSECT directive. If NAME is omitted, a comma must be used in its place. For example, the directive:

```
PSECT ,GBL
```

shows a .PSECT directive with a blank name argument and the GBL argument. Default values (see Table 6–4) are assumed for all other unspecified arguments.

The .PSECT directive can be used without a name or arguments (see Section 6.7.1.1).

The .PSECT directive lets you create program sections (see Section 6.7.1.1) and to share code and data among the sections you have created (see Section 6.7.1.2). In declaring the program sections (also called p-sections), you can declare the attributes of the p-sections. This lets you control memory allocation and at the same time increases program modularity. (For a discussion of memory allocation, refer to the applicable system manual—see the Associated Documents section in the Preface.)

MACRO–11 provides for 256\(^{10}\) program sections, as listed below:

- One default absolute program section (. ABS.)
- One default relocatable program section (. BLK.)\(^1\)
- 254\(^{10}\) named program sections. (You can have more, but only the first 254 appear in the symbol table.)

\(^1\) In RT–11, this program section is unnamed.
For each program section specified or implied, MACRO-11 maintains the following information:

- Program section name
- Contents of the current location counter
- Maximum location counter value encountered
- Program section attributes (described in Table 6–4)

### 6.7.1.1 Creating Program Sections

The first statement of a source program is always an implied .PSECT directive; this causes MACRO-11 to begin assembling source statements at relocatable zero of the unnamed program section.

The first occurrence of a .PSECT directive with a given name assumes that the current location counter is set at relocatable zero. The scope of this directive then extends until a directive declaring a different program section is specified. Subsequent .PSECT directives cause assembly to resume where the named section previously ended; for example:

```
.PSECT ;Declares unnamed relocatable program
A: .WORD 0 ;section assembled at relocatable
B: .WORD 0 ;addresses 0 through 5.
C: .WORD 0
    .PSECT ALPHA ;Declares relocatable program section
X: .WORD 0 ;named ALPHA assembled at relocatable
Y: .WORD 0 ;addresses 0 through 3.
    .PSECT ;Returns to unnamed relocatable
D: .WORD 0 ;program section and continues assem-
    ;bly at relocatable address 6.
```

A given program section can be defined completely upon encountering its first .PSECT directive. Thereafter, the section can be referenced by specifying its name only or by completely respesifying its attributes. For example, a program section can be declared through the directive:

```
.PSECT ALPHA,ABS,UVR
```

and later referenced through the equivalent directive:

```
.PSECT ALPHA
```

which requires no arguments. If arguments are specified, they must be identical to the ones previously declared for the p-section. If the arguments differ, the arguments of the first .PSECT will remain in effect, and an error code (A) will be generated as a warning.

By maintaining separate location counters for each program section, MACRO-11 lets you write statements that are not physically sequential but that can be loaded sequentially following assembly, as shown in the following example.
.PSECT SEC1,REL,RO ;Start a relocatable program section
A:  .WORD 0 ;named SEC1 assembled at relocatable
B:  .WORD 0 ;addresses 0 through 5.
C:  .WORD 0
ST:  CLR A ;Assemble code at relocatable
     CLR B ;addresses 6 through 21(octal).
     CLR C
     .PSECT SECA,ABS ;Start an absolute program section
     .WORD .+2,A ;absolute addresses 0 through 3.
     .PSECT SEC1 ;named SECA. Assemble code at
     INC A ;Resume relocatable program section
     BR ST ;addresses 22 through 27(octal).

All labels in an absolute program section are absolute; likewise, all labels in a relocatable section are relocatable. The current location counter symbol (.) is relocatable or absolute when referenced in a relocatable or absolute program section, respectively.

Any labels appearing on a line containing a .PSECT (or .ASECT or .CSECT) directive are assigned the value of the current location counter before the .PSECT (or other) directive takes effect. Thus, if the first statement of a program is:

A:  .PSECT ALT,REL

the label A is assigned to relocatable address zero of the unnamed program section.

Since it is not known during assembly where relocatable program sections will be loaded, all references to relocatable program sections are assembled as references relative to the base of the referenced section.

In the following example, references to the symbols X and Y are translated into references relative to the base of the relocatable program section named SEN.

.PSECT ENT,ABS
   .=+.+1000
A:  CLR X ;Assembled as CLR base of
     JMP Y ;relocatable section + 10(octal).
     .PSECT SEN,REL
     MOV RO,R1
     JMP A ;Assembled as JMP 1000.

Y:  HALT
X:  .WORD 0

NOTE
In the preceding example, using a constant in conjunction with the current location counter symbol (.) in the form .=1000 would result in an error, because constants are always absolute and are always associated with the program’s .ASECT (..ABS.). If the form .=1000 were used, a program section incompatibility would be detected. See Section 3.6 for a discussion of the current location counter.
Thus, MACRO-11 provides the Linker or Task Builder with the necessary information to resolve the linkages between various program sections. Such information is not necessary, however, when an absolute program section is referenced, because all instructions in an absolute program section are associated with an absolute virtual address.

6.7.1.2 Code or Data Sharing

Named relocatable program sections with the arguments GBL and OVR operate in the same manner as FORTRAN COMMON; that is, program sections of the same name with the arguments GBL and OVR from different assemblies are all loaded at the same location at link time. All other program sections (those with the argument CN) are concatenated.

A single symbol could name both an internal symbol and a program section. Considering FORTRAN again, using the same symbolic name is necessary to accommodate the following statement:

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

where:

```
X
```

represents the base of the program section and also the fourth element of that section.

6.7.1.3 Memory Allocation Considerations

MACRO-11 does not generate an error when a module ends at an odd location. You can, therefore, place odd length data at the end of a module. However, when several modules contain object code contributions to the same program section having the concatenate attribute (see Table 6-4; CN/OVR), odd length modules (except the last) may cause succeeding modules to be linked starting at odd locations, thereby making the linked program unexecutable. To avoid this problem, separate code and data from each other and place them in separately named program sections (see Table 6-4; I/D). The Linker or Task Builder can then begin each program section on an even address. Refer to the applicable system manual for further information on memory allocation of tasks (see the Associated Documents section in the Preface).

6.7.2 .ASECT and .CSECT Directives

Format:

```
ASECT
CSECT
CSECT symbol
```

where:

```
symbol
```

represents one or more of the arguments in Table 6-4.

IAS and RSX-11M assembly language programs use the .PSECT and .ASECT directives exclusively, because the .PSECT directive provides all the capabilities of the .CSECT directive defined for other PDP-11 assemblers. MACRO-11 accepts both .ASECT and .CSECT directives, but assembles them as though they were .PSECT directives with the default attributes listed in Table 6-5. Compatibility exists between other MACRO-11
programs and the IAS and RSX-11M Task Builders, because the Task Builders also
treat the .ASECT and .CSECT directives like .PSECT directives with the default values
listed in Table 6–5.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>.ASECT</th>
<th>.CSECT (named)</th>
<th>.CSECT (unnamed)</th>
<th>.PSECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>.ABS.</td>
<td>name</td>
<td>.BLK. 1</td>
<td>name</td>
</tr>
<tr>
<td>Access</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>Type</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>Scope</td>
<td>GBL</td>
<td>GBL</td>
<td>LCL</td>
<td>LCL</td>
</tr>
<tr>
<td>Relocation</td>
<td>ABS</td>
<td>REL</td>
<td>REL</td>
<td>REL</td>
</tr>
<tr>
<td>Allocation</td>
<td>OVR</td>
<td>OVR</td>
<td>CON</td>
<td>CON</td>
</tr>
</tbody>
</table>

1In RT–11, this program section has no default name.

Note that the statement:

```
.CSECT JIM
```

is identical to the statement:

```
.PSECT JIM, GBL, OVR
```

because the .CSECT default values GBL and OVR are assumed for the named program
section.

### 6.7.3 .SAVE Directive

**Format:**

```
.SAVE
```

The .SAVE directive stores the current program section context on the top of the
program section context stack, while leaving the current program section context in
effect. If the program section context stack is full when .SAVE is issued, the directive is
flagged with an error code (A) in the assembly listing. The program section context
stack can handle 16 .SAVES. The program section context includes the values of the
current location counter and the maximum value assigned to the location counter in
the current program section.

See Figure 6–8 for an example of .SAVE.

### 6.7.4 .RESTORE Directive

**Format:**

```
.RESTORE
```
The .RESTORE directive retrieves the program section context from the top of the program section context stack. If the program section context stack is empty when .RESTORE is issued, the directive is flagged with an error code (A) in the assembly listing. When .RESTORE retrieves a program section, it restores the current location counter to the value it had when the program section was saved.

When saving and restoring program sections, be careful not to store data in a program section for which you have saved but not yet restored the context; when you restore the context, the data will be lost. For example, the .WORD 1000 in the following series of instructions has no effect:

```
.PSECT A
.SAVE ;Save context of psect A
.PSECT A ;Re-establish psect A
.WORD 1000 ;The effect of this instruction...
.RESTORE ;gets wiped out by this .RESTORE directive
          ;that restores the old context of psect A
```

See Figure 6–8 for an example of .RESTORE.

**Figure 6–8:** Example of .SAVE and .RESTORE Directives

```
.MAIN. MACRO V08.04 Wednesday 05-Jun-87 10:06 Page 1
Example of .SAVE/.RESTORE usage

.ENTRY Example of .SAVE/.RESTORE usage
1       2
3       4
  ; Define local impure storage
 5       6
  ;
7       8
     .MACRO DS NAME, SIZE
9       10
     .SAVE ;Save the current .PSECT
11      12
     .PSECT IMPURE,D,GBL ;Store the data in the IMPURE .PSECT
13      14
     NAME: BLKW SIZE ;Set aside the space
14      15
     .RESTORK ;Reenter the current .PSECT
15      16
    .ENDM
16      17
     ; SCANSY
17      18
    ; Scan the hash table for valid entries
18      19
     ;
19      20
    000000 016701 000000' SCANSY: MOV SYMBAS,R1 ;Get base of table
20      21
    000004 010167 000002' MOV R1,CURSYM ;Initialize pointer to table
21      22
    000010 006701 000004' ADD SYMIZ,R1 ;Point past the table
22      23
    000014 010167 000006' MOV R1,SYMTOP ;Save end address
23      24
    ; Rest of SCANSY routine....
24      25
    ;
25      26
    000020 000207 RETURN ;Table is scanned, exit.
26      27
27      28
28      29
29      30
30      31
31      32
32      33
33      34
34      35
35      36
36      37
37      38
```

**Figure 6–8 Cont’d. on next page**
6.8 Symbol Control Directives

The symbol control directives are used to set the type of a given symbol.

6.8.1 .GLOBL Directive

Format:

.GLOBL sym1,sym2,...,symn

where:

sym1,sym2,...,symn represent valid symbolic names. When multiple symbols are specified, they are separated by any valid separator (comma, space, and/or tab).

A statement line containing a .GLOBL directive can also include a label field and/or a comment field.

The .GLOBL directive defines (and thus provides linkage to) symbols not otherwise defined as global symbols within a module. In defining global symbols, the directive .GLOBL A,B,C is similar to:

A==:expression A==:expression A::
B==:expression or B==:expression or B::
C==:expression C==:expression C::

Because object modules are linked by global symbols, these symbols are vital to a program. The following paragraph, describing the processing of a program from assembly to linking, explains the global's role.

In assembling a source program, MACRO-11 produces a relocatable object module and a listing file containing the assembly listing and symbol table. The Linker or Task Builder joins separately assembled object modules into a single executable image. During linking, object modules are relocated relative to the base of the module and linked by global symbols. Because these symbols will be referenced by other program modules, they must be singled out as global symbols in the defining modules. As shown above, the .GLOBL directive, global assignment operator, or global label operator will define a symbol as global.

All internal symbols appearing within a given program must be defined at the end of assembly pass 1, or they will be assumed to be default global references. Refer to Section 6.2.1 for a description of enabling/disabling of global references.
In the following example, A and B are entry-point symbols. The symbol A has been explicitly defined as a global symbol by means of the .GLOBAL directive, and the symbol B has been explicitly defined as a global label by means of the double colon (:). Since the symbol C is not defined as a label within the current assembly, it is an external (global) reference if .ENABLE GBL is in effect.

```
; Define a subroutine with 2 entry points which calls an
; external subroutine

PSECT       ;Declare the unnamed program section.
GLOBAL A    ;Define A as a global symbol.
MOV @(RS)+,R0 ;Define entry point A.
MOV #X,R1
JSR PC,C    ;Call external subroutine C.
RTS R5      ;Exit.
MOV (RS)+,R1 ;Define entry point B.
CLR R2
BR X
```

External symbols can appear in the operand field of an instruction or MACRO-11 directive as a direct reference, as shown in the examples below:

```
CLR EXT
.WORD EXT
CLR @EXT
```

External symbols can also appear as a term within an expression, as shown below:

```
CLR EXT+A
.WORD EXT-2
CLR @EXT+A(R1)
```

An undefined external symbol cannot be used in the evaluation of a direct assignment statement or as an argument in a conditional assembly directive (see Sections 3.3, 6.9.1, and 6.9.3).

### 6.8.2 .WEAK Directive

Format:

```
.WEAK sym1,sym2,...symn
```

where:

```
sym1,sym2,...symn represent valid symbolic names. When multiple symbols are specified, they are separated by any valid separator (comma, space, and/or tab).
```

Example:

```
.WEAK SUB1,SUB2
```

A statement line containing a .WEAK directive can also include a label field and/or a comment field.
The `.WEAK` directive is used to specify symbols that are either defined externally in another module or defined globally in the current module. This directive suppresses object library searches for specified external symbols.

When the `.WEAK` directive specifies a symbol that is externally defined, it is considered a global symbol. If the Linker finds the symbol's definition in another module, it uses that definition. If the Linker does not find an external definition, the symbol is given a value of 0. The Linker does not search a library for the global symbol, but if a module brought in from a library for another reason contains the symbol's definition, the Linker uses that definition.

If a symbol that is defined in the current module is specified by the `.WEAK` directive, the symbol is considered globally defined. However, if the current module is inserted in an object library, the symbol is not inserted in the library's symbol table. Consequently, the module is not found when the library is searched at link time to resolve the symbol.

**NOTE**

The `.WEAK` directive is supported only by the RT-11 Librarian (LIBR) and Linker (LINK). Support is not yet implemented in the RSX-11 Task Builder (TKB) or Librarian (LBR).

### 6.9 Conditional Assembly Directives

Conditional assembly directives allow you to include or exclude blocks of source code during the assembly process, based on the evaluation of stated condition tests within the body of the program.

#### 6.9.1 Conditional Assembly Block Directives

Format:

```
.IFDEF cond,argument(s) ;Start conditional assembly block.
.
.
range ;Range of conditional assembly block.
.
.
.ENDC ;End of conditional assembly block.
```

where:

- `cond` represents a specified condition that must be met if the block is to be included in the assembly. The conditions that can be tested by the conditional assembly directives are defined in Table 6–6.

- `,` represents any valid separator (comma, space, and/or horizontal tab).
argument(s) represent(s) the symbolic argument(s) or expression(s) of the specified conditional test. These arguments are thus a function of the condition to be tested (see Table 6-6).

range represents the body of code that is either included in the assembly, or excluded, depending upon whether the condition is met.

.ENDC terminates the conditional assembly block. This directive must be present to end the conditional assembly block.

A condition test other than those listed in Table 6-6, an invalid argument, or a null argument specified in a .IF directive causes that line to be flagged with an error code (A) in the assembly listing.

**Table 6-6: Valid Condition Tests for Conditional Assembly Directives**

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Arguments</th>
<th>Assemble Block If:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>Complement</td>
<td></td>
</tr>
<tr>
<td>EQ</td>
<td>NE</td>
<td>Expression</td>
</tr>
<tr>
<td>GT</td>
<td>LE</td>
<td>Expression</td>
</tr>
<tr>
<td>LT</td>
<td>GE</td>
<td>Expression</td>
</tr>
<tr>
<td>DF</td>
<td>NDF</td>
<td>Symbolic argument</td>
</tr>
<tr>
<td>B</td>
<td>NB</td>
<td>Macro(^1)argument</td>
</tr>
<tr>
<td>IDN</td>
<td>DIF</td>
<td>Two 7-bit ASCII or 8-bit DEC Multinational macro(^1) arguments</td>
</tr>
</tbody>
</table>

\(^1\)A macro argument (a form of symbolic argument) is enclosed within angle brackets or delimited by the circumflex construction, as described in Section 7.3. For example, <A,B,C> \(/124/\)

\(^2\)Use P1 and P2 with great care. Most programs do not need them. If used incorrectly, they can cause P (phase) errors during assembly. Before you use P1 or P2, examine your program and make sure you are not trying to use them to disguise some sort of logic error in the way the program is written.

An example of a conditional assembly directive follows:

```
.IF EQ ALPHA+1 ;Assemble block if ALPHA+1=0
   ...
. ENDC
```

The two operators \& and ! have special meaning within DF and NDF conditions, in that they are allowed in grouping symbolic arguments.

\& Logical AND operator
!
Logical inclusive OR operator

For example, the conditional assembly statement:

```
.IF DF SYM1 & SYM2
   ...
. ENDC
```

results in the assembly of the conditional block if the symbols SYM1 and SYM2 are both defined. Nested conditional directives take the form:

```
.IF condition1
   .IF condition2
   ...
   . ENDC ;condition2
. ENDC ;condition1
```

For example, the following conditional directives:

```
.IF DF SYM1
   .IF DF SYM2
   ...
   . ENDC ;DF SYM2
. ENDC ;DF SYM1
```

can govern whether assembly is to occur. In the example above, if the outermost condition is unsatisfied, no deeper level of evaluation of nested conditional statements within the program occurs.

Although indentation is not required, you can indent nested conditionals to improve readability, and you can include comments on .ENDC statements to help you match them to their corresponding conditional assembly directives, as shown above.

Each conditional assembly block must terminate with a .ENDC directive. A .ENDC directive encountered outside a conditional assembly block is flagged with an error code (O) in the assembly listing.

MACRO-11 permits a nesting depth of 16₁₀ conditional assembly levels. Any statement that attempts to exceed this nesting level depth is flagged with an error code (O) in the assembly listing.
6.9.2 Subconditional Assembly Block Directives

Formats:

.IFF
.IFT
.IFTF

Subconditional directives can be placed within conditional assembly blocks to indicate:

- The assembly of an alternate body of code when the condition of the block tests false
- The assembly of a noncontiguous body of code within the conditional assembly block, depending upon the result of the conditional test in entering the block
- The unconditional assembly of a body of code within a conditional assembly block

Subconditional directives are described in detail in Table 6–7. If a subconditional directive appears outside a conditional assembly block, an error code (O) is generated in the assembly listing.

<table>
<thead>
<tr>
<th>Subconditional Directive</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>.IFF</td>
<td>If the condition tested upon entering the conditional assembly block is false, the code following this directive, and continuing up to the next occurrence of a subconditional directive or to the end of the conditional assembly block, is to be included in the program.</td>
</tr>
<tr>
<td>.IFT</td>
<td>If the condition tested upon entering the conditional assembly block is true, the code following this directive, and continuing up to the next occurrence of a subconditional directive or to the end of the conditional assembly block, is to be included in the program.</td>
</tr>
<tr>
<td>.IFTF</td>
<td>The code following this directive, and continuing up to the next occurrence of a subconditional directive or to the end of the conditional assembly block, is to be included in the program, regardless of the result of the condition tested upon entering the conditional assembly block.</td>
</tr>
</tbody>
</table>

The implied argument of a subconditional directive is the condition test specified upon entering the conditional assembly block, as reflected by the initial directive in the conditional coding examples below. Conditional or subconditional directives in nested conditional assembly blocks are not evaluated if the previous (or outer) condition in the block is not satisfied. Examples 3 and 4 below illustrate nested directives that are not evaluated because of previously unsatisfied conditional coding.
Example 1: Assume that symbol SYM is defined.

```
.IFDEF SYM
  ;Tests TRUE, SYM is defined. Assemble
  ;the following code.
.
.IFF
  ;Tests FALSE. SYM is defined. Do not
  ;assemble the following code.
.
.IFT
  ;Tests TRUE. SYM is defined. Assemble
  ;the following code.
.
.IFTF
  ;Assemble following code unconditionally.
.
.IFT
  ;Tests TRUE. SYM is defined. Assemble
  ;remainder of conditional assembly block.
.
.ENDIF ;DF SYM
```

Example 2: Assume that symbol X is defined and that symbol Y is not defined.

```
.IFDEF X
  ;Tests TRUE, symbol X is defined.
.
.IFDEF Y
  ;Tests FALSE, symbol Y is not defined.
.
.IFF ;DF Y
  ;Tests TRUE, symbol Y is not defined,
  ;assemble the following code.
.
.IFT ;DF Y
  ;Tests FALSE, symbol Y is not defined.
  ;Do not assemble the following code.
.
.ENDIF ;DF Y
.ENDIF ;DF X
```

Example 3: Assume that symbol A is defined and that symbol B is not defined.

```
.IFDEF A
  ;Tests TRUE. A is defined.
  ;Assemble the following code.
.
MOV A, @R1
.
.
.IFF ;DF A
  ;Tests FALSE. A is defined. Do not
  ;assemble the following code.
.
MOV R1, R0
.
.
IFDEF NDF B
  ;Nested conditional directive is not
  ;evaluated.
.
.ENDIF ;NDF B
.ENDIF ;DF A
```
Example 4: Assume that symbol X is not defined and that symbol Y is defined.

```assembly
.IF DF X ;Tests FALSE. Symbol X is not defined.
    ;Do not assemble the following code.
.IF DF Y ;Nested conditional directive is not
    ;evaluated.
    .
    .
.IFF ;DF Y ;Nested subconditional directive is
    ;not evaluated.
    .
    .
.IFT ;DF Y ;Nested subconditional directive is
    ;not evaluated.
    .
    .
.ENDC ;DF Y
.ENDC ;DF X
```

6.9.3 Immediate Conditional Assembly Directive

Format:

```assembly
.IIF cond, arg, statement
```

where:

cond represents a valid condition test defined for conditional assembly blocks in Table 6–6.

represents any valid separator (comma, space, and/or tab), unless cond is B or NB; in that case, a comma must be used unless the argument is enclosed in angle brackets or delimited by the circumflex construction (see Table 3–3).

arg represents the argument associated with the immediate conditional directive; an expression, symbolic argument, or macro argument (see Table 6–6).

represents the separator between the conditional argument and the statement field. If the preceding argument is an expression, then a comma must be used; otherwise, a comma, space, and/or tab can be used.

statement represents the specified statement to be assembled if the condition is satisfied.

An immediate conditional assembly directive lets you write a 1-line conditional assembly block. The use of this directive requires no terminating .ENDC statement and the condition to be tested is completely expressed within the line containing the directive.

For example, the immediate conditional statement:

```assembly
.IIF DF FOO, BEQ ALPHA
```

generates the code:

```assembly
BEQ ALPHA
```

if the symbol FOO is defined within the source program.
As with the .IF directive, a condition test other than those listed in Table 6–6, an invalid argument, or a null argument specified in a .IF directive results in an error code (A) in the assembly listing.

6.10 File Control Directives

The MACRO–11 file control directives are used to add file names to macro library lists and to insert a source file into the source file being currently used.

6.10.1 .LIBRARY Directive

Format:

.LIBRARY string

where:

string represents a delimited string that is the file specification of a macro library.

The .LIBRARY directive adds a file name to a macro library list that is searched. A library list is searched whenever a .MCALL or an undefined op code is encountered within a MACRO–11 program. The libraries that make up the list are searched in the reverse order in which they were specified to the MACRO–11 assembler.

If any information was omitted from the macro library argument, default values are assumed. The default library device and file type for MACRO–11/RT–11 are DX: and .MLB, and for other systems they are SY: and .MLB.

The .LIBRARY directive is used as follows:

.LIBRARY /DB1: [SMITH]USERLIB/
.LIBRARY ?DX:SYSDEF. MLB?
.LIBRARY \CURRENT. MLB\

MACRO–11 searches all macro libraries if it finds an unknown symbol in the op code field and the auto-mcall option has been previously enabled by .EXABL MCL.

NOTE

If you are using MACRO–11 with the RT–11 operating system, the device handler for the device the .LIBRARY file resides on must already be loaded, either explicitly with the KMON LOAD command, or implicitly by reference to the device on the original MACRO–11 command line. The maximum number of .LIBRARY files that you can specify is limited to twelve minus the number of files specified in the MACRO–11 command line. Up to eight files can be specified on a MACRO–11/RT–11 command line, so at least four slots are available for .LIBRARY files.
6.10.2 .INCLUDE Directive

Format:

```
.INCLUDE string
```

where:

- `string` represents a delimited string that is the file specification of a macro source file.

The .INCLUDE directive inserts a source file within the source file currently being used. When this directive is encountered, an implicit .PAGE directive is issued, status of the current source file is stacked, and the source file specified by the directive is read into memory. When the end of the specified source file is reached, an implicit .PAGE directive is issued, the original source file status is popped from the stack, and assembly resumes at the line following the directive. A source file can also be inserted within a source file that has already been specified by the .INCLUDE directive. In this case, the status of the original source file and the first source file specified by the .INCLUDE directive are stacked, and the second specified source file is read into memory. When the end of the second source file is reached, the status of the first specified source file is popped from the stack, and assembly resumes at the line following the directive. When the end of the first specified source file is reached, the status of the original source file is popped from the stack, and assembly of that file is started again at the line following the .INCLUDE directive. An implicit .PAGE directive precedes and follows each included source file. The maximum nesting level of source files specified by the .INCLUDE directive is five.

If any information is omitted from the source file argument, default values are assumed. The default source file device and file type for MACRO-11/RT-11 are DK: and .MAC, and for other systems they are SY: and .MAC.

The .INCLUDE directive is used as follows:

```
.INCLUDE /DR3:[1,2]MACROS/ ;File MACROS.MAC
.INCLUDE ?DK:SYSDEF?    
.INCLUDE \CURRENT.MAC\    
```

**NOTE**

If you are using MACRO-11 with an RT-11 operating system, the device handler for the device that the .INCLUDE file resides on must already be loaded, either explicitly with the KMON LOAD command, or implicitly by reference to the device on the original MACRO-11 command line.
Chapter 7
Macro Directives

This chapter tells you how to use MACRO-11’s macro directives to define and write macros. Macro directives let you:

- Define macros
- Call macros
- Test and substitute macro arguments
- Test macro attributes
- Report error conditions
- Perform counted or indefinite repeat loops
- Call macros from libraries
- Delete macro definitions

Each function is described in its own section of this chapter. Table 7–1 gives an alphabetical list of all directives described in this chapter and the associated section reference. Also refer to Section B.3 for a complete list of all MACRO-11 directives.

Table 7–1: Directives in Chapter 7

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
<th>Section Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ENDM</td>
<td>Terminates a macro definition.</td>
<td>7.1.2</td>
</tr>
<tr>
<td>.ENDR</td>
<td>Terminates a counted or indefinite repeat block.</td>
<td>7.7</td>
</tr>
<tr>
<td>.ERROR</td>
<td>Writes a message to the listing file to flag invalid macro arguments or conditions.</td>
<td>7.5</td>
</tr>
<tr>
<td>.IRP</td>
<td>Creates an indefinite repeat block.</td>
<td>7.6.1</td>
</tr>
<tr>
<td>.IRPC</td>
<td>Creates an indefinite repeat block for character string arguments.</td>
<td>7.6.2</td>
</tr>
<tr>
<td>.MACRO</td>
<td>Begins a macro definition.</td>
<td>7.1.1</td>
</tr>
<tr>
<td>.MCALL</td>
<td>Calls a previously-defined macro from a library.</td>
<td>7.8</td>
</tr>
<tr>
<td>.MDELETE</td>
<td>Deletes a macro definition from MACRO-11’s macro symbol table.</td>
<td>7.9</td>
</tr>
<tr>
<td>.MEXIT</td>
<td>Prematurely terminates execution of a macro.</td>
<td>7.1.3</td>
</tr>
<tr>
<td>.NARG</td>
<td>Returns number of macro arguments.</td>
<td>7.4.1</td>
</tr>
<tr>
<td>.NCHR</td>
<td>Returns number of characters in an argument.</td>
<td>7.4.2</td>
</tr>
</tbody>
</table>
Table 7–1 (Cont.): Directives in Chapter 7

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
<th>Section Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NTYPE</td>
<td>Returns addressing mode of an argument.</td>
<td>7.4.3</td>
</tr>
<tr>
<td>.PRINT(^1)</td>
<td>Writes a message to the listing file to flag invalid macro arguments or conditions, and generates a (P) error.</td>
<td>7.5</td>
</tr>
<tr>
<td>.REPT</td>
<td>Creates a counted repeat block.</td>
<td>7.7</td>
</tr>
</tbody>
</table>

\(^1\)MACRO–11's .PRINT directive is not the same as the RT–11 monitor .PRINT request; be careful not to confuse the two. Under RT–11, if you .MCALL .PRINT, you will get the RT–11 monitor .PRINT request; otherwise, you will get MACRO–11's .PRINT directive.

7.1 Defining Macros

By using macros, you can use a single line to insert a sequence of lines into a source program.

A macro definition is headed by a .MACRO directive (see Section 7.1.1) followed by the source lines. The source lines may optionally contain dummy arguments. If such arguments are used, each one is listed in the .MACRO directive.

A macro call (see Section 7.3) is the statement you use to call the macro into the source program. It consists of the macro name followed by the real arguments needed to replace any dummy arguments used in the macro.

Macro expansion is the insertion of the macro source lines into the main program. Included in this insertion is the replacement of the dummy arguments by the real arguments.

Macro directives provide the means to define macros and control macro expansions. Only one directive is allowed per source line. Each directive may have a blank operand field or one or more operands. Valid operands differ with each directive. This chapter describes the macro directives available in MACRO–11 and their arguments.

7.1.1 .MACRO Directive

Format:

```
[label:] .MACRO name, dummy argument list
```

where:

- **label** represents an optional statement label.
- **name** represents the user-assigned symbolic name of the macro. This name can be any valid symbol and can be used as a label elsewhere in the program.
- **dummy argument list** represents any valid separator (comma, space, and/or tab).
dummy argument list represents a number of valid symbols (see Section 3.2.2) that can appear anywhere in the body of the macro definition, even as a label. These dummy symbols can be used elsewhere in the program with no conflict of definition. Multiple dummy arguments specified in this directive can be separated by any valid separator. The detection of a duplicate or an invalid symbol in a dummy argument list terminates the scan and causes an error code (A) to be generated.

The first statement of a macro definition must be a .MACRO directive.

A comment can follow the dummy argument list in a .MACRO directive, as shown below:

```
.MACRO ABS A,B ;Defines macro ABS with two arguments.
```

Although it is acceptable for a label to appear on a .MACRO directive, this practice is discouraged, especially in the case of nested macro definitions, because invalid labels or labels constructed with the concatenation character will cause the macro directive to be ignored. This may result in improper termination of the macro definition.

### 7.1.2 .ENDM Directive

**Format:**

```
.ENDM [name]
```

**where:**

- **name** represents an optional argument specifying the name of the macro being terminated by the directive.

**Example:**

```
.ENDM ;Terminates the current
                    ;macro definition.
.ENDM ABS ;Terminates the current
                    ;macro definition named ABS.
```

The final statement of every macro definition must be a .ENDM directive.

If specified, the macro name in the .ENDM statement must match the name specified in the corresponding .MACRO directive. Otherwise, the statement is flagged with an error code (A) in the assembly listing. In either case, the current macro definition is terminated. Specifying the macro name in the .ENDM statement permits MACRO-11 to detect missing .ENDM statements or improperly nested macro definitions.

The .ENDM directive must not have a label. If a valid label is attached, the label is ignored; if an invalid label is attached, the directive is ignored.
The `.ENDM` directive can be followed by a comment field, as shown below:

```
.MACRO TYPMSG MESSGE       ;Type a message.
JSR R5, TYPMSG
.WORD MESSGE
.ENDM                        ;End of TYPMSG macro.
```

You can also use the `.ENDM` directive to terminate indefinite repeat blocks (see Section 7.6) and repeat blocks (see Section 7.7).

### 7.1.3 `.MEXIT` Directive

**Format:**

```
.MEXIT
```

The `.MEXIT` directive terminates a macro expansion before the end of the macro is encountered. This directive is also valid within repeat blocks (see Sections 7.6 and 7.7). It is most useful in nested macros. The `.MEXIT` directive terminates the current macro as though a `.ENDM` directive had been encountered. A `.MEXIT` also terminates any pending `.IF` directives.

Using the `.MEXIT` directive bypasses the complexities of nested conditional directives and alternate assembly paths, as shown in the following example:

```
.MACRO ALTR N,A,B
  
  .IF EQ N     ;Start conditional assembly block.
    
  .MEXIT      ;Terminate macro expansion.
  .ENDC       ;End conditional assembly block.
  
  .ENDM       ;Normal end of macro.
```

In an assembly where the dummy symbol `N` is replaced by 0 (zero) (see Table 6–6), the `.MEXIT` directive would assemble the conditional block and terminate the macro expansion. When macros are nested, a `.MEXIT` directive exits to the next higher level of macro expansion. A `.MEXIT` directive encountered outside a macro definition is flagged with an error code (O) in the assembly listing.

### 7.1.4 MACRO Definition Formatting

A form feed character within a macro definition causes a page eject during the assembly of the macro definition. However, no page eject is done when the macro is expanded.

Conversely, when the `.PAGE` directive is used in a macro definition, it is ignored during the assembly of the macro definition, but a page eject is performed when that macro expansion is listed.
7.2 Calling Macros

Format:

\[ \text{[label:] name real arguments} \]

where:

- **label** represents an optional statement label.
- **name** represents the name of the macro, as specified in the `.MACRO` directive (see Section 7.1.1).
- **real arguments** represent symbolic arguments which replace the dummy arguments listed in the `.MACRO` directive. When multiple arguments occur, they are separated by any valid separator. Arguments to the macro call are treated as character strings, their usage is determined by the macro definition.

A macro must be defined with the `.MACRO` directive (see Section 7.1.1) before the macro can be called and expanded within the source program.

When a macro name is the same as a user label, the appearance of the symbol in the operator field designates the symbol as a macro call; the appearance of the symbol in the operand field designates it as a label, as shown below:

ABS: MOV (RO),R1 ;ABS is defined as a label.

. .

BR ABS ;ABS is considered to be a label.

. .

ABS #4,ENT,LAR ;ABS is a macro call.

You can also assign a value to a symbol that has the same name as a macro, as illustrated in this example:

ABS = 100 ;ABS is a user symbol.

. .

ABS #4,ENT,LAR ;ABS is a macro call.

7.3 Arguments in Macro Definitions and Macro Calls

Multiple arguments within a macro definition or macro call must be separated by one of the valid separating characters described in Section 3.1.1. Macro definition arguments (dummy) and macro call arguments (real) normally maintain a strict positional relationship. That is, the first real argument in a macro call corresponds with the first dummy argument in a macro definition. Only the use of keyword arguments in a macro call can override this correspondence (see Section 7.3.6).
For example, the following macro definition and its associated macro call contain multiple arguments:

```
.MACRO REN A,B,C
  
  REN ALPHA,BETA,<C1,C2>
```

Arguments which themselves contain separating characters must be enclosed in paired angle brackets. For example, the macro call:

```
REN <MOV X,Y>,#44,WEV
```

uses the entire expression:

```
MOV X,Y
```

to replace all occurrences of the symbol A in the macro definition. Real arguments within a macro call are considered to be character strings and are treated as a single entity during the macro expansion.

The circumflex (\^) construction allows angle brackets to be passed as part of the argument. For example, this construction could have been used in the above macro call, as follows:

```
REN \^/<MOV X,Y>./,#44,WEV
```

passing the character string \(<MOV X,Y>\) as an argument.

Because of the use of the circumflex (\^) shown above, you must be careful when passing an argument beginning with a unary operator (\(^0\), \(^D\), \(^B\), \(^R\), \(^F\) ...). These arguments must be enclosed in angle brackets (as shown below) or MACRO\-11 will read the character following the circumflex as a delimiter.

```
REN \(<^0 411>,X,Y
```

The following macro call:

```
REN #44,WEV^-MOV X,Y/
```

contains only two arguments (#44 and WEV^-MOV X,Y/), because the circumflex is a unary operator (see Section 3.1.3) and it is not preceded by an argument separator.

As shown in the examples above, spaces can be used within bracketed argument constructions to increase the legibility of such expressions.

When 8-bit DEC Multinational character set (MCS) characters are used in argument strings, they must be enclosed in angle brackets (\(< >\)) or the argument delimiter (\(/\)) must be preceded by a circumflex (\(^\)) The following are valid uses of the MCS characters in the argument string:

```
<This string can contain MCS characters>

\/^This string can contain MCS characters/
```
7.3.1 Macro Nesting

Macro nesting occurs where the expansion of one macro includes a call to another. The depth of nesting allowed depends upon the amount of dynamic memory used by the source program being assembled.

To pass an argument containing valid argument delimiters to nested macros, enclose the argument in the macro definition within angle brackets, as shown in the coding sequence below. This extra set of angle brackets for each level of nesting is required in the macro definition, not in the macro call.

```assembly
MACRO LEVEL1 DUM1, DUM2
LEVEL2 <DUM1>
LEVEL2 <DUM2>
ENDM

MACRO LEVEL2 DUM3
DUM3
ADD #10, R0
MOV R0, (R1)+
ENDM
```

A call to the LEVEL1 macro, as shown below, for example:

```assembly
LEVEL1 <MOV X, R0>, <MOV R2, R0>
```

causes the following macro expansion to occur:

```assembly
MOV X, R0
ADD #10, R0
MOV R0, (R1)+
MOV R2, R0
ADD #10, R0
MOV R0, (R1)+
```

When macro definitions are nested, the inner definition cannot be called until the outer macro has been called and expanded. For example, in the following coding:

```assembly
MACRO LV1 A, B

MACRO LV2 C

ENDM
ENDM
```

the LV2 macro cannot be called and expanded until the LV1 macro has been expanded. Likewise, any macro defined within the LV2 macro definition cannot be called and expanded until LV2 has also been expanded.
7.3.2 Special Characters in Macro Arguments

If an argument does not contain spaces, tabs, semicolons, or commas, it can include special characters without enclosing them in a bracketed construction. For example:

```
.Macro  PUSH ARG
    MOV    ARG, -(SP)
.Endm

PUSH    X+3(%2)
```

generates the following code:

```
Mov    X+3(%2), -(SP)
```

7.3.3 Passing Numeric Arguments as Symbols

If the unary operator backslash (\) precedes an argument, the macro treats that argument as a numeric value in the current program radix. The ASCII characters representing this value are inserted in the macro expansion, and their function is defined in the context of the resulting code. The backslash operator cannot take a forward reference (the argument must be defined at the time it is used), and the argument cannot be a relocatable symbol.

The following example illustrates the use of the backslash operator:

```
1 .list    w
2  
3    ; Example of the use of the backslash (\) operator
4  
5 .macro   reserve x
6      .blkw    x
7 .endm
8 
9    ;Note difference in the way the macro gets expanded when backslash
10    ;is used and when it is not. (In this case the resulting binary
11    ;code is the same.)
12  
13    0000010: size=10
14    000000:  reserve size
15         : blkw   size
16    000020:  reserve \size
17         : blkw   10
18    000020:  .end
19    
```

Another, more complicated, example is given below:

```
.Macro  inc a,b
    con    a,\b
B=b+1
.Endm

.Macro  con a,b
    .word   4
A'\B:
.Endm

C=0  inc   x,c
```

The above macro call (`INC`) would thus expand to:

```
X0: .WORD 4
```

In this expanded code, the label `X0:` results from the concatenation of two real arguments. The single quote (') character in the label `A'B:` concatenates the real arguments `X` and `0` as they are passed during the expansion of the macro. This type of argument construction is described in more detail in Section 7.3.7.

A subsequent call to the same macro would generate the following code:

```
X1: .WORD 4
```

and so on, for later calls. The two macro definitions are necessary, because the symbol associated with dummy argument B (that is, C) cannot be updated in the CON macro definition, because the character 0 (zero) has replaced C in the argument string (`INC X, C`). In the CON macro definition, the number that is passed is treated as a string argument. (Where the value of the real argument is 0 (zero), only a single 0 character is passed to the macro expansion.)

Passing numeric values in this manner is useful in identifying source listings. For example, versions of programs created through conditional assemblies of a single source program can be identified through such coding as that shown below. Assume, for example, that the symbol `ID` in the macro call (`IDT`) has been equated elsewhere in the source program to the value 6.

```
.MACRO IDT SYM
.DEF /V01.'SYM/'
.ENDM

IDT \ID
```

The above macro call would then expand to:

```
.DEF /V01.6/
```

where 6 is the numeric value of the symbol `ID`.

### 7.3.4 Number of Arguments in Macro Calls

A macro can be defined with or without arguments. If more arguments appear in the macro call than in the macro definition, an error code (Q) is generated in the assembly listing. If fewer arguments appear in the macro call than in the macro definition, missing arguments are assumed to be null values. The conditional directives `.IF B` and `.IF NB` (see Table 6–6) can be used within the macro to detect missing arguments. The number of arguments can also be determined by using the `.NARG` directive (Section 7.4.1).

### 7.3.5 Creating Local Symbols Automatically

A label is often required in an expanded macro. In the conventional macro facilities thus far described, a label must be explicitly specified as an argument with each macro call. You must be careful in issuing subsequent calls to the same macro in order to avoid duplicating labels. This concern can be eliminated through a feature
of MACRO–11 that creates a unique symbol where a label is required in an expanded macro.

MACRO–11 can automatically create local symbols of the form n$, where n is an integer in the range 3000010 through 6553510, inclusive. Such local symbols are created by MACRO–11 in numerical order, as shown below:

```
30000$
30001$
...
65534$
65535$
```

This automatic generation occurs on each call of a macro whose definition contains a dummy argument preceded by the question mark (?) character, as shown in the macro definition below:

```
.MACRO ALPHA, A,?B     ;Contains dummy argument B preceded by
    ;question mark.
    TST    A
    BEQ    B
    ADD    #5,A
B:
    .ENDM
```

A local symbol is created automatically by MACRO–11 only when a real argument of the macro call is either null or missing, as shown in Example 1 below. If the real argument is specified in the macro call, however, MACRO–11 inhibits the generation of a local symbol and normal argument replacement occurs, as shown in Example 2 below. (Examples 1 and 2 are both expansions of the ALPHA macro defined above.)

**Example 1:** Create a Local Symbol for the Missing Argument

```
ALPHA R1     ;Second argument is missing.
TST R1
BEQ  30000$  ;Local symbol is created.
ADD #5,R1
```

```
30000$:
```

**Example 2:** Do Not Create a Local Symbol

```
ALPHA R2,XYZ ;Second argument XYZ is specified.
TST R2
BEQ XYZ ;Normal argument replacement occurs.
ADD #5,R2
```

```
XYZ:
```

Automatically created local symbols are restricted to the first $16_{10}$ arguments of a macro definition.

Automatically created local symbols resulting from the expansion of a macro, as described above, do not establish a local symbol block in their own right.

When a macro has several arguments earmarked for automatic local symbol generation, substituting a specific label for one such argument risks assembly errors because
MACRO-11 constructs its argument substitution list at the point of macro invocation. Therefore, the appearance of a label, the .ENABL LSB directive, or the .PSECT directive, in the macro expansion will create a new local symbol block. The new local symbol block could leave local symbol references in the previous block and their symbol definitions in the new one, causing error codes in the assembly listing. Furthermore, a later macro expansion that creates local symbols in the new block may duplicate one of the symbols in question, causing an additional error code (P) in the assembly listing.

7.3.6 Keyword Arguments

Format:

name=string

where:

name represents the dummy argument.

string represents the real symbolic argument.

The keyword argument cannot contain embedded argument separators unless delimited as described in Section 7.3.

Macros can be defined with, and/or called with, keyword arguments. When a keyword argument appears in the dummy argument list of a macro definition, the specified string becomes the default real argument at macro call. When a keyword argument appears in the real argument list of a macro call, however, the specified string becomes the real argument for the dummy argument that matches the specified name, whether or not the dummy argument was defined with a keyword. If a match fails, the entire argument specification is treated as the next positional real argument.

The DEC Multinational character set can be used in keyword arguments if enclosed in angle brackets (<>).

A keyword argument can be specified anywhere in the dummy argument list of a macro definition and is part of the positional ordering of argument. A keyword argument can also be specified anywhere in the real argument list of a macro call but, in this case, does not affect the positional ordering of the arguments.

1 .LIST ME
2 ;
3 ; Define a macro having keywords in dummy argument
4 ; list
5 ;
6 .MACRO TEST CONTROL=1,BLOCK,ADDRESS=TEMP
7 .WORD CONTROL
8 .WORD BLOCK
9 .WORD ADDRESS
10 .ENDM
11
12
13 ;
14 ; Now call several times
15 ;
7.3.7 Concatenation of Macro Arguments

The single quote or apostrophe character (') operates as a valid delimiting character in macro definitions. A single quote that precedes and/or follows a dummy argument in a macro definition is removed, and the substitution of the real argument occurs at that point. For example, in the following statements:

```
.MACRO DEF A,B,C
'A':  .ASCIZ /C/
      .BYTE '"A','B'
      .ENDM
```

when the macro DEF is called through the statement:

```
DEF     X,Y,<MACRO-11>
```

it expands as follows:

```
XY:   .ASCIZ /MACRO-11/
      .BYTE '"X','Y'
```

During expansion of the first line, the scan for the first argument terminates upon finding the first single quote (') character. Since A is a dummy argument, the single quote (') is removed. The scan then resumes with B; B is also noted as another dummy
argument. The two real arguments \( X \) and \( Y \) are then concatenated to form the label \( XY \). The third dummy argument is noted in the operand field of the \texttt{.ASCIIZ} directive, causing the real argument MACRO–11 to be substituted in this field.

When the arguments of the \texttt{.BYTE} directive are evaluated during expansion of the second line, the scan begins with the first single quote (') character. Since it is neither preceded nor followed by a dummy argument, this single quote remains in the macro expansion. The scan then encounters the second single quote, which is followed by a dummy argument and is therefore discarded. The scan of argument \( A \) is terminated upon encountering the comma (,). The third single quote is neither preceded nor followed by a dummy argument and again remains in the macro expansion. The fourth (and last) single quote is followed by another dummy argument and is likewise discarded. (Four single quote characters were necessary in the macro definition to generate two single quote characters in the macro expansion.)

7.4 Macro Attribute Directives: \texttt{.NARG}, \texttt{.NCHR}, and \texttt{.NTYPE}

MACRO–11 has three directives that let you determine certain attributes of macro arguments: \texttt{.NARG}, \texttt{.NCHR}, and \texttt{.NTYPE}. The use of these directives permits selective modifications of a macro expansion, depending on the nature of the arguments being passed. These directives are described below.

7.4.1 \texttt{.NARG} Directive

Format:

\[
\text{[label:] .NARG symbol}
\]

where:

- \texttt{label} represents an optional statement label.
- \texttt{symbol} represents any valid symbol. This symbol is equated to the number of nonkeyword arguments in the macro call currently being expanded. If a symbol is not specified, the \texttt{.NARG} directive is flagged with an error code (A) in the assembly listing.

The \texttt{.NARG} directive determines the number of nonkeyword arguments in the macro call currently being expanded. Hence, the \texttt{.NARG} directive can appear only within a macro definition; if it appears elsewhere, an error code (O) is generated in the assembly listing.

An example of the \texttt{.NARG} directive is shown in Figure 7–1.
Figure 7–1: Example of .NARG Directive

```assembly
.TITLE NARG

; Example of the .NARG directive

.MACRO NULL NUM
    .NARG SYM
    .IF EQ SYM
    .MEXIT
    .IFF
    .REPT NUM
    NOP
    .ENDR
    .ENDC
.ENDM

000000
    NULL
    .NARG SYM
    .IF EQ SYM
    .MEXIT
    .IFF
    .REPT 6
    NOP
    .ENDR
    .ENDC

000000 000000

000001
    NULL 6
    .NARG SYM
    .IF EQ SYM
    .MEXIT
    .IFF
    .REPT 6
    NOP
    .ENDR

000000 000240
000002 000240
000004 000240
000006 000240
000010 000240
000012 000240

000001 .END
```
7.4.2 .NCHR Directive

Format:

[label:] .NCHR symbol,<string>

where:

label represents an optional statement label.

symbol represents any valid symbol. This symbol is equated to the number of characters in the specified character string. If a symbol is not specified, the .NCHR directive is flagged with an error code (A) in the assembly listing.

<string> represents any valid separator (comma, space, and/or tab).

<string> represents a string of 7-bit ASCII or 8-bit DEC Multinational printing characters. If the character string contains a valid separator (comma, space, and/or tab), the whole string must be enclosed within angle brackets (<> ) or be delimited by the circumflex (^) construction (see Section 7.3). If the delimiting characters do not match or if the ending delimiter cannot be detected because of a syntactical error in the character string (thus prematurely terminating its evaluation), the .NCHR directive is flagged with an error code (A) in the assembly listing.

The .NCHR directive, which can appear anywhere in a MACRO–11 program, determines the number of characters in a specified character string. This directive is useful in calculating the length of macro arguments.

An example of the .NCHR directive is shown in Figure 7–2.
Figure 7–2: Example of .NCHR Directive

1 .TITLE NCHR
2
3 .LIST ME
4 :* ; Illustrate the .NCHR directive
5 ;
6
7 .MACRO STRING MESSAG
8 .NCHR $$$, MESSAG
9 .WORD $$$
10 .ASCII /MESSAG/
11 .EVEN
12 .ENDM
13
14 000000 MSG1: STRING <Hello>
15 000005 .NCHR $$$, Hello
16 000000 000005 .WORD $$$
17 000002 110 .ASCII /Hello/
18 000003 145
19 000004 154
20 000005 154
21 000006 157

7.4.3 .NTYPE Directive

Format:

[label:] .NTYPE symbol,aexp

where:

- label represents an optional statement label.
- symbol represents any valid symbol. This symbol is equated to the 6-bit addressing mode of the following expression (aexp). If a symbol is not specified, the .NTYPE directive is flagged with an error code (A) in the assembly listing.
- , represents any valid separator (comma, space, and/or tab).
- aexp represents any valid address expression, as used with an op code. If no argument is specified, an error code (O) will appear in the assembly listing.

The .NTYPE directive determines the addressing mode of a specified macro argument. Hence, the .NTYPE directive can appear only within a macro definition; if it appears elsewhere, it is flagged with an error code (O) in the assembly listing.

An example of a .NTYPE directive in a macro definition is shown in Figure 7–3.
Figure 7–3: Example of .NTYPE Directive in Macro Definition

1 .TITLE NTYPE
2
3 .LIST ME
4;
5 ; Illustrate the .NTYPE directive
6 ;
7
8 .MACRO SAVE ARG
9 .NTYPE $$$, ARG
10 .IF EQ $$$&70
11 MOV ARG,-(SP) ;Save in register mode
12 .IFF
13 MOV #ARG,-(SP) ;Save in non-register mode
14 .ENDC
15 .ENDM
16
17 000000 SAVE R1
18 000001 .NTYPE $$$, R1
19 000000 01016 IF EQ $$$&70
20 MOV R1,-(SP) ;Save in register mode
21 .IFF
22 MOV #R1,-(SP) ;Save in non-register mode
23 .ENDC
24
25 000002 SAVE TEMP
26 000067 .NTYPE $$$, TEMP
27 .IF EQ $$$&70
28 MOV TEMP,-(SP) ;Save in register mode
29 .IFF
30 000002 012746 MOV #TEMP,-(SP) ;Save in non-register mode
31 000006 .ENDC
32 000000 TEMP: .WORD 0
33 000001 .END

For additional information concerning addressing modes, refer to Chapter 5 and Section B.2.
7.5 .ERROR and .PRINT Directives

Format:

[1abel:] .ERROR [expr] :text

where:

label     represents an optional statement label.
expr      represents an optional expression whose value is output when the .ERROR
directive is encountered during assembly.
;         denotes the beginning of the text string.
text      represents the message associated with the .ERROR directive. The text can
          be 7-bit ASCII or 8-bit DEC Multinational characters.

The .ERROR directive writes a message to the listing file during assembly pass 2. A
common use of this directive is to warn you about a rejected or erroneous macro call
or an invalid set of conditions in a conditional assembly. If the listing file is not
specified, the .ERROR messages are written to the command output device.

Upon encountering a .ERROR directive anywhere in a source program, MACRO–11
writes a single line containing:

1. An error code (P)
2. The sequence number of the .ERROR directive statement
3. The value of the current location counter
4. The value of the expression, if one is specified
5. The source line containing the .ERROR directive.

For example, the following line tests an argument to be sure its value is at least 100,
and writes a message if it is not:

.IIF LT <A-100> .ERROR A ;Invalid macro argument

If A had a value of 76, a line in the following form would be written to the listing file:

<table>
<thead>
<tr>
<th>Seq. No.</th>
<th>Loc. No.</th>
<th>Exp. Value</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>512</td>
<td>005642</td>
<td>000076 .ERROR A ;Invalid macro argument</td>
</tr>
</tbody>
</table>

The .PRINT\(^1\) directive is identical in function to the .ERROR directive, except that it is
not flagged with the error code (P).

\(^1\) MACRO–11’s .PRINT directive is not the same as the RT–11 monitor .PRINT request; be careful not to confuse the two. Under RT–11, if you
.MCALL .PRINT, you will get the RT–11 monitor .PRINT request; otherwise, you will get MACRO–11’s .PRINT directive.
7.6 Indefinite Repeat Block Directives: .IRP and .IRPC

An indefinite repeat block is similar to a macro definition with only one dummy argument. At each expansion of the indefinite repeat range, this dummy argument is replaced with successive elements of a real argument list. Since the repeat directive and its associated range are coded inline within the source program, this type of macro definition and expansion does not require calling the macro by name, as required in the expansion of the conventional macros previously described in this chapter.

An indefinite repeat block can appear either within or outside another macro definition, indefinite repeat block, or repeat block. The rules for specifying indefinite repeat block arguments are the same as for specifying macro arguments (see Section 7.3).

7.6.1 .IRP Directive

Format:

[label:] .IRP sym,<argument list>

(range of indefinite repeat block)

.ENDR

where:

label represents an optional statement label.

Although it is valid for a label to appear on a .IRP directive, this practice is discouraged, especially in the case of nested macro definitions, because invalid labels or labels constructed with the concatenation character will cause the macro directive to be ignored. This may result in improper termination of the macro definition.

This also applies to .IRPC and .REPT.

sym represents a dummy argument that is replaced with successive real arguments from within the angle brackets. If no dummy argument is specified, the .IRP directive is flagged with an error code (A) in the assembly listing.

, represents any valid separator (comma, space, and/or tab).

<argument list> represents a list of real arguments enclosed within angle brackets that is to be used in the expansion of the indefinite repeat range. A real argument can consist of one or more 7-bit ASCII or 8-bit DEC Multinational characters; multiple arguments must be separated by any valid separator (comma, space, and/or tab). If no real arguments are specified, no action is taken.
range represents the block of code to be repeated once for each occurrence of a real argument in the list. The range can contain other macro definitions, repeat ranges and/or the .MEXIT directive (see Section 7.1.3).

.ENDR indicates the end of the indefinite repeat block range. You can also terminate an indefinite repeat block with .ENDM.

The .IRP directive replaces a dummy argument with successive real arguments specified in an argument string. This replacement process occurs during the expansion of an indefinite repeat block range.

Use the .MEXIT directive to leave a .IRP loop if you want to exit the loop before its normal completion.

An example of the .IRP directive is shown in Figure 7–4.

7.6.2 .IRPC Directive

Format:

[labell:] .IRPC sym,<string>

.
.
.
(range of indefinite repeat block)
.
.
.
.ENDR

where:

label represents an optional statement label (see discussion in Section 7.6.1).

sym represents a dummy argument that is replaced with successive real arguments from within the angle brackets. If no dummy argument is specified, the .IRPC directive is flagged with an error code (A) in the assembly listing.

, represents any valid separator (comma, space, and/or tab).

<string> represents a list of 7-bit ASCII or 8-bit DEC Multinational characters, enclosed within angle brackets, to be used in the expansion of the indefinite repeat range. Although the angle brackets are required only when the string contains separating characters, their use is recommended for legibility.

range represents the block of code to be repeated once for each occurrence of a character in the list. The range can contain macro definitions, repeat ranges, and/or the .MEXIT directive (see Section 7.1.3).

.ENDR indicates the end of the indefinite repeat block range. You can also terminate an indefinite repeat block with .ENDM.

The .IRPC directive does single character substitution, rather than argument substitution. On each iteration of the indefinite repeat range, the dummy argument is replaced with successive characters in the specified string.
Use the .MEXIT directive to leave a .IRPC loop, if you want to exit the loop before its normal completion.

An example of the .IRPC directive is shown in Figure 7–4.

Figure 7–4: Example of .IRP and .IRPC Directives

```
1 .TITLE IRPTST
2
3 .LIST ME
4 ; Illustrate the .IRP and .IRPC directives
5 ; by creating a pair of RAD50 tables
6 ;-
7
8 000000 REGS: .IRP  REG,<PC,SP,R5,R4,R3,R2,R1,R0>
9 000000  062170 .RAD50 /REG/
10 000002  074500 .RAD50 /PC/
11 000004  072770 .RAD50 /SP/
12 000006  072720 .RAD50 /R5/
13 000010  072650 .RAD50 /R4/
14 000012  072600 .RAD50 /R3/
15 000014  072530 .RAD50 /R2/
16 000016  072460 .RAD50 /R1/
17
18 000020 REGS2: .IRPC  NUM,<76543210>
19 000020  073110 .RAD50 /R7/
20 000022  073040 .RAD50 /R6/
21 000024  072770 .RAD50 /R5/
22 000026  072720 .RAD50 /R4/
23 000030  072650 .RAD50 /R3/
24 000032  072600 .RAD50 /R2/
25 000034  072530 .RAD50 /R1/
26 000036  072460 .RAD50 /R0/
27
28 000001 .END
29
```

7.7 Repeat Block Directive: .REPT, .ENDR

Format:

```
[label:] .REPT exp

(\i\i) (range of repeat block)

.ENDR
```

Macro Directives 7–21
where:

- **label** represents an optional statement label (see discussion in Section 7.6.1).
- **exp** represents any valid expression. This value controls the number of times the block of code is to be assembled within the program. When the expression value is less than or equal to zero, the repeat block is not assembled. If this expression is not an absolute value, the .REPT statement is flagged with an error code (A) in the assembly listing.
- **range** represents the block of code to be repeated. The repeat block can contain macro definitions, indefinite repeat blocks, other repeat blocks and/or the .MEXIT directive (see Section 7.1.3).
- **.ENDR** indicates the end of the repeat block range. You can also terminate a repeat block with .ENDM.

The .REPT directive duplicates a block of code, a certain number of times, in line with other source code.

Use the .MEXIT directive to leave a .REPT loop, if you want to exit the loop before its normal completion.

### 7.8 Macro Library Directive: .MCALL

**Format:**

```
.MCALL arg1, arg2, ... argn
```

where:

- **arg1, arg2, ... argn** represent the symbolic names of the macro definitions required in the assembly of the source program. The names must be separated by any valid separator (comma, space, and/or tab).

The .MCALL directive identifies any system and/or user-defined macro definitions that are not defined within the source program but which are required to assemble the program.

The .MCALL directive must appear before the first occurrence of a call to any externally defined macro if:

- Auto-Mcall mode is disabled (the default)
- The name of the macro being called is the same as one of MACRO's permanent symbols or directives, such as SUB, .ERROR, or .PRINT. Otherwise, MACRO will use the permanent symbol or directive instead of the macro from the library.

The /ML switch (see Section 8.1.3) under RSX-11M and the /LIBRARY qualifier (see Section 8.2.2) under IAS and RT-11, used with an input file specification, indicate to MACRO-11 that the file is a macro library. Additional macro libraries to be searched can also be specified in the MACRO-11 program itself, using the MACRO-11 .LIBRARY directive. See Section 6.10.1 for a description of the .LIBRARY directive. When a macro call is encountered in the source program, MACRO-11 first searches the user macro...
library for the named macro definitions and, if necessary, continues the search with the system macro library.

You can specify any number of user-supplied libraries\(^1\). For multiple library files, the search for the named macros begins with the last such file specified. The files are searched in reverse order until the required macro definitions are found, finishing, if necessary, with a search of the system macro library.

If any named macro is not found upon completion of the search, the `.MCALL` statement is flagged with an error code (U) in the assembly listing. Furthermore, a statement elsewhere in the source program that attempts to expand such an undefined macro is flagged with an error code (O) in the assembly listing.

The command strings to MACRO-11, through which file specifications are supplied, are described in detail in the applicable system manual (see the Associated Documents section in the Preface).

### 7.9 Macro Deletion Directive: `.MDELETE`

**Format:**

```
.MDELETE name1,name2,...,namen
```

where:

```
name1,name2,...,namen
```

represent valid macro names. When multiple names are specified, they are separated by any valid separator (comma, space, and/or tab).

The `.MDELETE` directive deletes the definitions of the specified macro(s), freeing virtual memory. If references are made to deleted macros, the referencing line is flagged with an op code (O) error.

An example of the `.MDELETE` directive is shown below.

```
.MDELETE .EXIT,EXIT$S
```

---

\(^1\) The number is restricted under RT-11. See Section 6.10.1.
Chapter 8

IAS/RSX–11M/RSX–11M–PLUS Operating Procedures

MACRO–11 assembles one or more ASCII source files containing MACRO–11 statements into a single relocatable binary object file. This binary object file contains the table of contents listing, the assembly listing, and the symbol table listing. An optional cross-reference listing of symbols and macros is available. A sample assembly listing is provided in Appendix H.

8.1 RSX–11M/RSX–11M–PLUS Operating Procedures

On RSX–11M and RSX–11M–PLUS systems, two command languages are available: the Monitor Console Routine (MCR) and the DIGITAL Command Language (DCL). When you log onto the system, you are given either MCR or DCL as the default command language. Your default command language is contained in your account file.

By pressing \( \text{CTRL/C} \) (echoed as \( ^{\text{C}} \)) at the monitor prompt, you can see the explicit prompt for the command language you are currently using:

\[
> ^{\text{C}} \\
\text{MCR}> \\
> ^{\text{C}} \\
\text{DCL}>
\]

You can switch from one command language to the other. To switch from DCL to MCR, type the following command:

\[
\text{DCL}> \text{SET TERMINAL MCR}
\]

To switch from MCR to DCL, type the following command:

\[
\text{MCR}> \text{SET /DCL=TI:}
\]

In addition to switching from one command language to the other, you can type a DCL command from a terminal set to MCR, and an MCR command from a terminal set to DCL, as shown below:

\[
\text{MCR}> \text{DCL cmd-string} \\
\text{DCL}> \text{MCR cmd-string}
\]
8.1.1 Running MACRO–11 Under RSX–11M/RSX–11M–PLUS

The following sections describe those MACRO–11 operating procedures that apply to both the Monitor Console Routine and the DIGITAL Command Language. You can use any of the four methods shown below to run MACRO–11:

- Direct MACRO–11 call
- Single assembly
- Install, run immediately, and remove on exit
- Indirect command processor

8.1.1.1 Direct MACRO–11 Call

MCR Format:

```
MCR> MAC
MAC> cmd-string
```

When you call MACRO–11 directly, the Monitor Console Routine (MCR) accepts MAC as input and runs MACRO–11. Since a command string is not present with the MCR line, MACRO–11 then asks for input with the prompting sequence MAC> and waits for command string input. After the assembly of the specified files has been completed, MACRO–11 again asks for command string input with the MAC> prompting sequence. This process repeats until you press [CTRL/Z].

DCL Format:

```
DCL> MACRO[/qualifier(s)]
File(s)? filespec[/qualifiers]...
```

DCL accepts MACRO as input and runs MACRO–11. In addition, you can include the qualifiers contained in Table 8–3. Since no file specifications are included in the DCL command line, MACRO–11 asks for input with the File(s)? prompt. You can then enter the name of one or more source files plus any of the qualifiers listed in Table 8–4. When you press RETURN, MACRO–11 does the assembly.

8.1.1.2 Single Assembly

MCR Format:

```
MCR> MAC cmd-string
```

DCL Format:

```
DCL> MACRO cmd-string
```

When you do a single assembly, no prompting from MACRO–11 occurs, since the command line includes the command string input. MACRO–11 assembles the source files in the command string and exits when finished.
8.1.1.3 Install, Run Immediately, and Remove on Exit

Format:

> RUN $MAC
MAC> cmd-string

Use this method when MACRO-11 is not permanently installed in the system. On RSX-11M, the system must be generated for this type of call support. MAC is run from the system directory. MACRO-11 asks for command string input. The command string must have the MCR format, even if run from a DCL terminal. When MACRO-11 exits, it is removed from the system.

If the system has the “flying install” feature, the RUN $ calling format is not needed.

8.1.1.4 Indirect Command Processor

MCR Formats:

MCR> MAC
MAC> @filespec

or:

MCR> MAC @filespec

or:

MAC> RUN $MAC[/UIC=\(g,m\)]
MAC> @filespec

These commands use the indirect command processor, which effectively substitutes “@filespec” for the “cmd-string” input used in the other methods. In the commands shown above, the indirect command processor passes commands to MACRO-11. The file specified as @filespec contains MACRO-11 command strings. After this file is opened, command lines are read from the file until the end-of-file is detected. Three nested levels of indirect files are permitted in MACRO-11.

MCR and DCL Format:

DCL> @filespec

These forms use the indirect command processor to pass commands to the command language. This is the only form you can use with DCL. The indirect command file @filespec must contain one of the command lines to run MACRO-11 as listed in the other methods.

NOTE

MACRO-11 can be terminated by entering a [CTRL/Z] any time a request for command string input is pending.
8.1.2 Default RSX-11 File Specifications

MACRO-11 accepts as input or creates as output up to six types of files. When using the MACRO-11 assembler, you should keep in mind the default device, directory, name, and types listed in Table 8–1. Table 8–1 lists the default values for each file specification.

Table 8–1: RSX-11 File Specification Default Values

<table>
<thead>
<tr>
<th>File</th>
<th>Device</th>
<th>Directory</th>
<th>Filename</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object file</td>
<td>Your default volume</td>
<td>Current</td>
<td>None</td>
<td>.OBJ</td>
</tr>
<tr>
<td>Listing file</td>
<td>Device used for object file</td>
<td>Directory used in object file</td>
<td>None</td>
<td>.LST</td>
</tr>
<tr>
<td>Source</td>
<td>Your default volume</td>
<td>Current; used for source 1 or device of last source file specified</td>
<td>None</td>
<td>.MAC</td>
</tr>
<tr>
<td>User macro library</td>
<td>Your default volume</td>
<td>Current, if macro file is specified first; if not, directory of last source file</td>
<td>None</td>
<td>.MLB</td>
</tr>
<tr>
<td>System macro library</td>
<td>Library device</td>
<td>Library [1,1]</td>
<td>RSXMAC</td>
<td>.SML</td>
</tr>
<tr>
<td>Indirect command file</td>
<td>Your default volume</td>
<td>Current</td>
<td>None</td>
<td>.CMD</td>
</tr>
</tbody>
</table>

8.1.3 MCR Command String Format

In response to the `MAC>` prompting sequence printed by MACRO-11, type the output and input file specifications in the form shown below:

MAC> object,listing=src1,src2,...,srcn

where:

- `object` represents the binary object (output) file.
- `listing` represents the assembly listing (output) file containing the table of contents, the assembly listing, and the symbol table.
- `=` separates output file specifications from input file specifications.
- `src1,src2,...,srcn` represent the ASCII source (input) files containing the MACRO-11 source program or the user-supplied macro library files to be assembled.

Only two output file specifications in the command string are recognized by MACRO-11; any more than two output files are ignored. No limit is set on the number of source input files. If the entire command string is longer than 80 characters and less than or equal to 132 characters, a hyphen can be placed at the end of the first line as a continuation character.
A null specification in either of the output file specification fields signifies that the associated output file is not desired. A null specification in the input file field, however, is an error condition, resulting in the error message MAC—I_Illegal filename on the command output device (see Section 8.5). The absence of both the device name (dev:) and the name of the file (filename.type) from a file specification is the equivalent of a null specification.

NOTE
When no listing file is specified, any errors encountered in the source program are printed on the terminal from which MACRO-11 was started. When the /NL switch is used in the listing file specification without an argument, the errors and symbol table are written to the file specified.

Each file specification contains the following information:

```
filespec /switch:value ...
```

where:

- **filespec** is the standard file specification.
- **/switch** represents an ASCII name identifying a switch option. This switch option can be specified in three forms, as shown below, depending on the function desired:
  - `/switch` Enables the specified switch action.
  - `/noswitch` Negates the specified switch action.
  - `/=switch` Negates the specified switch action.

In addition, the switch identifier can be accompanied by ASCII character strings, octal numbers, or decimal numbers. The default assumption for a numeric value is octal. Decimal values must be followed by a decimal point (.)

Any numeric value preceded by a number sign (#) is regarded as an explicit octal declaration; this option is provided for command line documentation and ready identification of octal values.

Also, any numeric value can be preceded by a plus sign (+) or a minus (−) sign. The positive specification is the default assumption. If an explicit octal declaration is specified (#), the sign indicator, if included, must precede the number sign.

All switch values must be preceded by a colon (:).

The switch specifications are interpreted in the context of the program to which they apply. The switch options applicable to MACRO-11 are described in Table 8-2.

If MACRO-11 detects a syntax error in the command string, MACRO-11 writes the error message MAC—Command syntax error to the command output device, followed by a copy of the entire command string.
At assembly time, you may want to override certain MACRO-11 directives appearing in the source program or to provide MACRO-11 with information establishing how certain files are to be handled during assembly. You can do so through one or more switches, which can be selectively included as additional parameters in each file specification. The available switches for MACRO-11 file specifications under RSX-11M/RSX-11M-PLUS are listed in Table 8–2.

<table>
<thead>
<tr>
<th>Switch</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/LI:arg /NL:arg</td>
<td>Listing control switches; these options accept ASCII switch values (arg) which are equivalent in function and name to the arguments for the .LIST and .NLIST directives you can include in your source program (see Section 6.1.1). Arguments that you specify with the /LI:arg and /NL:arg switches override any arguments that you may have specified with the .LIST and .NLIST directives and remain in effect for the entire assembly process.</td>
</tr>
<tr>
<td>/EN:arg /DS:arg</td>
<td>Function control switches; these options accept ASCII switch values (arg) which are equivalent in function and name to the arguments for the .ENABL and .DSABL directives you can include in your source program (see Section 6.2.1). Arguments that you specify with the /EN:arg and /DS:arg switches override any arguments that you may have specified with the .ENABL and .DSABL directives and remain in effect for the entire assembly process.</td>
</tr>
<tr>
<td>/NL</td>
<td>The /NL switch, which takes no accompanying switch values, identifies an input file as a macro library file. As noted in Section 7.8, any macro that is defined externally must be identified by a .MCALL directive before it can be retrieved from a macro library file and assembled with the user program. In locating macro definitions, MACRO-11 performs a fixed search algorithm, beginning with the last specified user macro file, continuing in reverse order with each such specified file, and terminating, if necessary, with a search of the system macro library file. If a required macro definition is not found upon completion of the search, an error code (U) is written in the assembly listing. Therefore, a user macro library file must be specified in the command line or by using the MACRO-11 .LIBRARY directive (see Section 6.10.1) prior to the source file(s) that use macros defined in the library file. MACRO-11 does not prescan the command line for macro libraries; when a new source file is needed, MACRO-11 parses the next input file specification. If that file specification contains the /NL switch, it is appended to the front of the library file list. As a result, a user macro library file must be specified in the command line prior to the source files which require it, in order to resolve macro definitions.</td>
</tr>
<tr>
<td>/SP</td>
<td>Spool listing output (default value).</td>
</tr>
<tr>
<td>/NOSP</td>
<td>Do not spool output.</td>
</tr>
<tr>
<td>/CR:[arg]</td>
<td>Produce a cross-reference listing (see Section 8.3).</td>
</tr>
</tbody>
</table>
Switches for the object file are limited to /EN and /DS; when specified, they apply throughout the entire command string. Switch options for the listing file are limited to /LI, /ML, /SP, /CR, and /NOSP. Switches for input files are limited to /ML, /EN, and /DS; the option /ML applies only to the file immediately preceding the option so specified, whereas the /EN and /DS options, as noted above, are also applicable to subsequent files in the command string.

Do not specify the same switch more than once following a file specification. If you do, the values included with any duplicate switch specification override any previously specified values. If you want to include two or more values for the same switch, separate them by colons, as shown below:

/LI:SRC:MEB

8.1.4 DCL Operating Procedures

RSX-11M/RSX-11M-PLUS indicates its readiness to accept a command by prompting with the DCL prompt. In response to the prompt, enter the command string in one of the formats shown below:

> MACRO[/qualifiers]
   FILE? filespec[/qualifiers][,filespec[/qualifiers]...]

or:

[DCL]> MACRO[/qualifiers] filespec[/qualifiers][,filespec[/qualifiers]...]

where:

qualifiers affect either the entire command string (command qualifiers) or the filespec (parameter qualifiers). See Table 8–3 for a description of the command qualifiers and Table 8–4 for a description of the parameter qualifier.

filespec is the standard file specification shown in Section 8.4.

Use a comma (,) to separate file specifications. MACRO-11 concatenates all the files and then performs the assembly.

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/【NO】CROSS_REFERENCE</td>
<td>Suppresses or generates a cross-reference listing (see Section 8.3). When the cross-reference is generated, a listing file is also generated, whether or not the /LIST qualifier is present in the command string. The default is /NOCROSS_REFERENCE.</td>
</tr>
<tr>
<td>Qualifier</td>
<td>Function</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>/DISABLE: arg</td>
<td>Overrides the .DISABLE or .ENABLE assembler directives in the source program. When more than one argument is entered, arguments must be enclosed in parentheses and separated by commas.</td>
</tr>
<tr>
<td>/ENABLE: arg</td>
<td></td>
</tr>
<tr>
<td>/DISABLE: (arg,arg...)</td>
<td>You can specify any of the following arguments with the /DISABLE or /ENABLE qualifier:</td>
</tr>
<tr>
<td>/ENABLE: (arg,arg...)</td>
<td></td>
</tr>
<tr>
<td>ABSOLUTE</td>
<td>If enabled, MACRO–11 assembles all relative addresses (address mode 67) as absolute addresses (address mode 37). The default is Disabled.</td>
</tr>
<tr>
<td>AUTO_MCALL</td>
<td>If enabled, MACRO–11 searches all known macro libraries for a macro definition that matches any undefined symbols appearing in the op code field of a MACRO–11 statement. The default is Disabled. If MACRO–11 finds an unknown symbol in the op code field, it either declares an undefined symbol (U) error, or declares the symbol as an external symbol, depending upon the GLOBAL argument described below.</td>
</tr>
<tr>
<td>BINARY</td>
<td>If enabled, MACRO–11 produces absolute binary output in FILES–11 format. The default is Disabled.</td>
</tr>
<tr>
<td>CARD_FORMAT</td>
<td>If enabled, MACRO–11 treats columns 73 through the end of the line as comments. The default is Disabled.</td>
</tr>
<tr>
<td>CASE_MATCH</td>
<td>If enabled, MACRO–11 makes the conditional assembly directives .IF IDN and .IF DIF alphabetically case sensitive. The default is not case sensitive.</td>
</tr>
<tr>
<td>GLOBAL</td>
<td>If disabled, MACRO–11 flags all undefined symbol references with an error code (U) on the assembly listing. The default is Enabled; MACRO–11 treats all symbols that are undefined at the end of assembly pass 1 as default global references.</td>
</tr>
</tbody>
</table>
Table 8-3 (Cont.): RSX-11 DCL Command Qualifiers

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCAL</td>
<td>If enabled, MACRO-11 treats all symbols as local symbols. When enabled, all global symbols are flagged with the undefined symbol (U) error message. The default is Disabled.</td>
</tr>
<tr>
<td>LOWER_CASE</td>
<td>If disabled, MACRO-11 converts all lowercase ASCII input to uppercase. The default is Enabled.</td>
</tr>
<tr>
<td>REGISTER_DEFINITIONS</td>
<td>If disabled, MACRO-11 ignores the normal register definitions. The default is Enabled.</td>
</tr>
<tr>
<td>TRUNCATION</td>
<td>If enabled, MACRO-11 performs floating-point truncation. If disabled, MACRO-11 performs floating-point rounding. The default is Disabled.</td>
</tr>
<tr>
<td>/*[NO]LIST[:filespec]</td>
<td>Specifies whether or not MACRO-11 should create and print a listing file. You can include /LIST as a qualifier for either a command or a file specification. If /LIST qualifies the command, the listing file is both entered in your directory and printed on the line printer. If you do not include a file specification, the listing file has a .LST file type and is named after the last file named in the MACRO command. The listing file cannot be a library file. (The LINK command and all other language commands use the name of the first file named in the command as the default file name.) If /LIST qualifies a file specification, the file is entered in your directory but is not printed on the line printer. The listing file is named after the file it qualifies. The default is /NOLIST.</td>
</tr>
<tr>
<td>/*[NO]OBJECT[:filespec]</td>
<td>Specifies whether or not MACRO-11 should create an object module. If you do not include a file specification in the command line, MACRO-11 creates an object file with the same file name as the source file and a .OBJ extension. The default is /OBJECT.</td>
</tr>
<tr>
<td>/[NO]SHOW:arg</td>
<td>Overrides any .LIST and .NLIST assembler directives that may be included in the source file. You can use any of the following arguments with the /SHOW qualifier:</td>
</tr>
<tr>
<td></td>
<td>BINARY Controls the listing of macro expansion binary code.</td>
</tr>
</tbody>
</table>
### Table 8–3 (Cont.): RSX–11 DCL Command Qualifiers

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLS</td>
<td>Controls listing of macro calls and repeat range expansions.</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>Controls listing of comments.</td>
</tr>
<tr>
<td>CONDITIONALS</td>
<td>Controls listing of unsatisfied conditional coding.</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>Controls listing of the table of contents during assembly pass 1.</td>
</tr>
<tr>
<td>COUNTER</td>
<td>Controls listing of the current location counter field.</td>
</tr>
<tr>
<td>DEFINITIONS</td>
<td>Controls listing of macro definitions and repeat range expansions.</td>
</tr>
<tr>
<td>EXPANSIONS</td>
<td>Controls listing of macro expansions.</td>
</tr>
<tr>
<td>EXTENSIONS</td>
<td>Controls listing of binary expansions.</td>
</tr>
<tr>
<td>LISTING_DIRECTIVES</td>
<td>Controls listing of listing control directives without arguments,</td>
</tr>
<tr>
<td></td>
<td>that is, directives that alter the listing level counter.</td>
</tr>
<tr>
<td>OBJECT_BINARY</td>
<td>Controls listing of the generated binary code.</td>
</tr>
<tr>
<td>SEQUENCE_NUMBERS</td>
<td>Controls listing of source line sequence numbers.</td>
</tr>
<tr>
<td>SOURCE</td>
<td>Controls listing of source lines.</td>
</tr>
<tr>
<td>SYMBOLS</td>
<td>Controls listing of the symbol table resulting from the assembly.</td>
</tr>
<tr>
<td>/[NO]WIDE</td>
<td>When set to /WIDE, the listing is printed in 132-column format.</td>
</tr>
<tr>
<td></td>
<td>When set to /NOWIDE, the listing is printed in 80-column format. The default is /NOWIDE.</td>
</tr>
</tbody>
</table>

### Table 8–4: RSX–11 DCL Parameter Qualifier

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/LIBRARY</td>
<td>Specifies that an input file is a macro library file.</td>
</tr>
<tr>
<td></td>
<td>The assembler processes the files listed in the command line in reverse order. Therefore, a library file cannot be the last file in the command line.</td>
</tr>
</tbody>
</table>
8.1.5 MACRO–11 Command String Examples

Example 1:

The following commands assemble the source file FILNAM.MAC into a relocatable object module named FILNAM.OBJ:

```
MCR> MAC FILNAM=FILNAM
DCL> MACRO
    FILE? FILNAM
    DCL> MACRO FILNAM
```

Example 2:

The following commands assemble the source file FILNAM.MAC and produce an object file with the name TESTA.OBJ:

```
MCR> MAC TESTA=FILNAM
DCL> MACRO/OBJECT:TESTA FILNAM
```

Example 3:

The following commands concatenate and assemble the source files named FILNAM.MLB, TESTA.MAC, SPAN3.MAC, and SHELL.MAC and create an object file named SHELL.OBJ:

```
MCR> MAC SHELL=FILNAM/ML,TESTA,SPAN3,SHELL
DCL> MACRO FILNAM/LIBRARY,TESTA,SPAN3,SHELL
```

Example 4: The following commands produce an object module and an assembly listing. Any .LIST TTM or .LIST COM directives in the source file are ignored. The listing produced by this command includes no comments and is printed in wide format:

```
MCR> MAC FILNAM,FILNAM/ML:TTM:COM=FILNAM
DCL> MACRO/LIST/NOSHOW:COMMENTS/WIDE FILNAM
```

8.2 IAS MACRO–11 Operating Procedures

The following sections describe those MACRO–11 operating procedures that apply exclusively to the IAS system.

8.2.1 Running MACRO–11 Under IAS

The MACRO command used under IAS assembles one or more ASCII source files containing MACRO–11 statements into a relocatable binary object file. MACRO–11 also produces an assembly listing followed by a symbol table listing. A cross-reference listing can also be produced by means of the /CROSSREFERENCE qualifier (see Section 8.3 below).

You can call MACRO–11 directly from the terminal (interactive mode) or from a batch file (batch mode). For interactive mode, use the MACRO command, which can be issued whenever the IAS Program Development System (PDS) is at command level, a condition signified by the appearance of the prompt:

```
PDS>
```
For batch mode, use the $MACRO command.

When the assembly is completed, MACRO–11 terminates operations and returns control to PDS. (Refer to the IAS User’s Guide for further information about interactive and batch mode operations.)

8.2.2 IAS Command String

Formats:

Interactive Mode:

PDS> MACRO qualifiers  filespec /LIBRARY +...

or:

PDS> MACRO qualifiers

FILES?  filespec /LIBRARY    +...

Batch Mode:

$MACRO qualifications  filespec /LIBRARY +...

where:

filespec is the specification of an input file (see Section 8.4) that contains MACRO–11 source program code. When the program consists of multiple files, a plus sign (+) must be used to separate each file specification from the next. The “wild card” form of a file specification is not allowed.

/LIBRARY specifies that an input file is a macro library file. Library files hold the definitions of externally defined macros. As noted in Section 7.8, an externally defined macro must be identified in a .MCALL directive before it can be retrieved and assembled with your program. When MACRO–11 encounters a .MCALL directive, a search begins for the definitions of the macros listed.

The search order is important, because a macro might have two different definitions in library files LIB1 and LIB2. For example, if you need the definition in LIB1, you must place LIB1 after LIB2 in the command line, because MACRO–11 searches the last file specified in the command line first, then moves backwards through the given files until all have been searched.

If a macro’s definition is not found in any of the files named by the user, MACRO–11 automatically searches the system macro library; if the definition is still not found, an error code (U) is generated in the assembly listing.
specifies one or more of the following:

/OBJECT[:filespec] produces an object file as specified by filespec (see Section 8.4). The default is a file with the same filename as the last named source file and a .OBJ extension. /OBJECT is always the default condition.

/NOOBJECT does not produce an object file.

/LIST[:filespec] produces an assembly listing file according to filespec (see Section 8.4). If filespec is not specified, the listing is printed on the line printer. The default in interactive mode is /NOLIST and in batch mode is /LIST.

/NOLIST does not produce a listing file. The default in interactive mode is /NOLIST and in batch mode is /LIST.

When no listing file is specified, any errors encountered in the source program are displayed at the terminal from which MACRO-11 was initiated.

/CROSSREFERENCE[[:arg1...arg4]] produces a cross-reference listing. Arg1 through arg4 are described in Section 8.3. This qualifier can be abbreviated to /C.

A MACRO-11 command string can be specified by using any one of the three formats shown above for the interactive and batch modes. In interactive mode, if the input file specification (filespec) does not begin on the same line as the MACRO command and its qualifiers, PDS prints the following prompting message:

FILES?

then waits for you to specify the input file(s).

In batch mode, the $MACRO command and its arguments must appear on the same line unless the PDS line continuation symbol (−) is used.

### 8.2.3 IAS Indirect Command Files

Format:

```
@[filespec]
```

where:

- **@** specifies that the name that follows is an indirect file.
- **filespec** is the file specification (see Section 8.4) of a file that contains a command string. The default extension for the file name is .CMD.

You can use the indirect command file facility of PDS with MACRO-11 command strings. Create an ASCII file that contains the desired command strings (or portions thereof) in the forms shown in Section 8.2.2. When an indirect command file reference
is used in a MACRO-11 command string, the contents of the specified file are taken as all or part of the command string.

An indirect command file reference must always be the rightmost entry in the command (see Section 8.2.4 for examples).

**8.2.4 IAS Command String Examples**

The following examples show typical PDS MACRO-11 command strings.

**Example 1:**

```
PDS> MACRO /NOLIST
FILES? A+BOOT.MAC;3
```

In this example, the source files A.MAC and BOOT.MAC;3 are assembled to produce an object file called BOOT.OBJ. No listing is produced.

**Example 2:**

Where the indirect command file TEST.CMD contains the command string:

```
MACRO/OBJECT:MYFILE A+B
```

the command:

```
PDS> @TEST
```

assembles the two files A.MAC and B.MAC into an object file called MYFILE.OBJ.

**Example 3:**

Where the indirect command file INDO2.CMD contains the command string segment:

```
ATEST/LIBRARY+BTEST+SRT1.021
```

the command:

```
PDS> MACRO/LIST:DK1:TST @INDO2
```

assembles the files BTEST.MAC and SRT1.021, using the macro library file ATEST.MAC to produce an object file named SRT1.OBJ. A listing file named TST.LST is placed on disk unit 1.

**Example 4:**

```
$MACRO/LIST:DKO:MICR/NOOBJECT -
LIB1/LIBRARY+MICR.MAC;002
```

In this example, the library file is assembled with the file MICR.MAC;002. The program listing file named MICR.LST is placed on disk unit 0.

**8.3 Cross-Reference Processor (CREF)**

The CREF processor is used to produce a listing that includes cross-references to symbols that appear in the source program. The cross-reference listing is appended to the assembly listing. Such cross-references are helpful in debugging and in reading long programs.
A cross-reference listing can include up to four sections:

- User-defined symbols
- Macro symbols
- Register symbols
- Permanent symbols

To generate a cross-reference listing, specify the /CR switch in the MACRO–11 command string. Optional arguments can also be specified. The form of the switch is:

```
/CR [ SYM ]
     [ MAC ]
     [ REG ]
     [ PST ]
     [ SEC ]
     [ ERR ]
```

where:
- SYM specifies user-defined symbols (default).
- MAC specifies macro symbols (default).
- REG specifies register symbols.
- PST specifies permanent symbols.
- SEC specifies program sections.
- ERR specifies error lines (default).

If you want to generate listings for user-defined and macro symbols only, use /CR. No argument is necessary.

However, if an argument is specified, only that type of cross-reference listing is generated. For example:

```
/CR:SYM
```

produces a cross-reference listing of user-defined symbols only. No listing of macro symbols is generated. Thus, to produce all six types of cross-reference listings, you must specify all six arguments; the order in which they are specified is not significant. Use a colon to separate arguments, for example:

```
```

The CREF processor (CRF) is more fully described in the Utilities Reference Manual supplied with your system.
Figure 8–1 illustrates a complete cross-reference listing. In the listing, references are made in the form page-line. To make the listing more informative, the CREF processor uses the following signs:

- somewhere in the source program the symbol listed is defined by a direct assignment statement.

• destructive reference; the value of the symbol is changed (its previous contents destroyed) by the program instruction at the line number marked by the asterisk (*).

# symbol definition; the symbol is defined by a direct assignment statement, a colon sign (:), or a double colon sign (::) at the line number marked by the number sign (#).

**Figure 8–1: Sample IAS CREF Listing**

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>VALUE</th>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>R5OUNP</td>
<td>O000000</td>
<td>G</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>= *****</td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td>0000062</td>
<td>R</td>
</tr>
<tr>
<td>#</td>
<td></td>
<td>2-38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#2-70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REGISTER</th>
<th>SYMBOL</th>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>*2-49</td>
<td>*2-64</td>
</tr>
<tr>
<td>R1</td>
<td>*2-44</td>
<td>*2-49</td>
</tr>
<tr>
<td>R2</td>
<td>*2-66</td>
<td>*2-45</td>
</tr>
<tr>
<td>R3</td>
<td>*2-47</td>
<td>2-65</td>
</tr>
<tr>
<td>R4</td>
<td>*2-42</td>
<td>*2-43</td>
</tr>
<tr>
<td>SP</td>
<td>*2-42</td>
<td>*2-53</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REGISTER</th>
<th>SYMBOL</th>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNE</td>
<td>2-52</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>2-46</td>
<td>2-48</td>
</tr>
<tr>
<td>CLR</td>
<td>2-64</td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>2-51</td>
<td></td>
</tr>
<tr>
<td>DIV</td>
<td>2-65</td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>2-42</td>
<td>2-43</td>
</tr>
<tr>
<td>MOV</td>
<td>2-42</td>
<td>2-44</td>
</tr>
<tr>
<td>MOV</td>
<td>2-45</td>
<td>2-47</td>
</tr>
<tr>
<td>MOV</td>
<td>2-49</td>
<td>2-66</td>
</tr>
<tr>
<td>RETURN</td>
<td>2-66</td>
<td>2-64</td>
</tr>
<tr>
<td>.BYTE</td>
<td>2-70</td>
<td>2-71</td>
</tr>
<tr>
<td>.END</td>
<td>2-76</td>
<td>2-73</td>
</tr>
<tr>
<td>.GLOBAL</td>
<td>2-38</td>
<td>2-74</td>
</tr>
<tr>
<td>.IDENT</td>
<td>1-2</td>
<td></td>
</tr>
<tr>
<td>.NLIST</td>
<td>2-69</td>
<td></td>
</tr>
<tr>
<td>.PSECT</td>
<td>2-40</td>
<td></td>
</tr>
<tr>
<td>.SBTTL</td>
<td>2-25</td>
<td></td>
</tr>
<tr>
<td>.TITLE</td>
<td>1-1</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 8–1 Cont’d. on next page*

Format:

```
dev:[g,m]name.ext;ver
```

where:

- **dev:** is the name of the device where the desired file resides. A device name consists of two characters followed by a 1- or 2-digit device unit number (octal) and a colon (for example, DP1:, DK0:, DT3:). The default device is specified in Table 8–1. The default device under IAS is established initially by the system manager for each user and can be changed through the SET command.

- **[g,m]** is the User File Directory (UFD) code. This code consists of a group number (octal), a comma (, ) and an owner (member) number (octal) all enclosed in brackets ([ ]). An example of a UFD code is: [200,30]. The default UFD is equivalent to the User Identification Code (UIC) given at login time. Under IAS, the UFD can be changed through the SET DEFAULT command.

- **name** is a 1- to 9-character alphanumeric filename. There is no default.

- **.ext** is a 1- to 3-character alphanumeric filename extension or type that is preceded by a period (. ). An extension is normally used to identify the nature of the file. Default values depend on the context of the file specification and are as follows:
  - .CMD Indirect command (input) file
  - .LST A listing (print format) file
  - .MAC MACRO–11 source module (input file)
  - .OBJ MACRO–11 object module (output file)
  - .CRF Intermediate CREF input file created by MACRO–11

- **;ver** is an octal number between 1 and 77777 that is used to differentiate between versions of the same file. This number is prefixed by a semicolon (; ).
  - For input files, the default value is the highest version number of the file that exists.
  - For output files, the default value is the highest version number of the file that exists increased by 1. If no version number exists, the value 1 is used.
This is the general form for a file specification in IAS/RSX-11M/RSX-11M-PLUS systems. Detailed information is provided in the applicable system user's guide or operating procedures manual (see the Associated Documents section in the Preface).


MACRO–11 writes an error message to the command output device when one of the error conditions described below is detected. MACRO–11 writes below the error message the command line that caused the error. If the error is a .INCLUDE or a .LIBRARY directive file error, MACRO–11 writes both the source line and the command line that caused the error.

```
MAC -- Error message
MACRO-11 source line
MACRO-11 command line
```

These error messages reflect operational problems and should not be confused with the error codes (see Appendix D) produced by MACRO–11 during assembly.

All the error messages listed below, with the exception of the MAC—Command I/O error message, terminate the current assembly; MACRO–11 then attempts to restart by reading another command line. In the case of a command I/O error, however, MACRO–11 exits, since it is unable to obtain additional command line input.

MAC—Command file/open failure

Either the file from which MACRO–11 is reading a command could not be opened initially or between assemblies, or the indirect command file specified as @filename in the MACRO–11 command line could not be opened. See MAC—Open failure on input file.

MAC—Command I/O error

An error was returned by the file system during MACRO–11's attempt to read a command line. This is an unconditionally fatal error, causing MACRO–11 to exit. No MACRO–11 restart is attempted when this message appears.

MAC—Command syntax error

An error was detected in the syntax of the MACRO–11 command line.

MAC—Illegal filename

Neither the device name nor the filename was present in the input file specification (the input file specification was null), or a wild card convention (asterisk) was employed in an input or output file specification.

Wildcard options (*) are not permitted in MACRO–11 file specifications.

MAC—Illegal switch

An invalid switch was specified for a file, an invalid value was specified with a switch, or an invalid use of a switch was detected by MACRO–11.
MAC—INCLUDE directive file error

The file specified in the .INCLUDE statement either does not exist or is invalid, the device specified in the command line is not available, or the .INCLUDE stacking depth exceeds five.

MAC—Indirect command syntax error

The name of the indirect command file (@filename) specified in the MACRO-11 command line is syntactically incorrect.

MAC—Indirect file depth exceeded

An attempt to exceed the maximum allowable number of nested indirect command files has occurred. (Three levels of indirect command files are permitted in MACRO-11.)

MAC—Insufficient dynamic memory

There is not enough physical memory available for MACRO-11 to page its symbol table. Reinstall MACRO-11 in a larger partition, or see Section F.3.

MAC—Invalid format in macro library

The library file has been corrupted, or it was not produced by the Librarian utility program (LBR).

MAC—I/O error on input file

In reading a record from a source input file or macro library file, the file system detected an error; for example, a line containing more than 13210 characters was encountered. This message may also indicate that a device problem exists or that either a source file or a macro library file has been corrupted with incorrect data.

MAC—I/O error on macro library file

Same meaning as MAC—I/O error on input file, except that the file is a macro library file and not a source input file.

MAC—I/O error on output file

The file system detected an error while writing a record to the object output file or the listing output file. This message may also indicate that a device problem exists or that the device is full.

MAC—I/O error on work file

A read or write error occurred on the work file used to store the symbol table. This error is most likely caused by a problem on the device or by an attempt to write to a full device.

MAC—LIBRARY directive file error

The file specified in the .LIBRARY statement either does not exist or is invalid, the file specification in the .LIBRARY directive is for a nonrandom access device, the device specified in the command line is not available, or the .LIBRARY stacking depth exceeds the maximum depth allowed.
MAC—Open failure on input file

One of the following conditions exists:

- The specified device does not exist.
- The volume is not mounted.
- A problem exists with the device.
- The specified directory file does not exist.
- The specified file does not exist.
- You do not have access privilege to the file directory or to the file itself.

MAC—Open failure on output file

One of the following conditions exists:

- The specified device does not exist.
- The volume is not mounted.
- A problem exists with the device.
- The specified directory file does not exist.
- You do not have access privilege to the file directory.
- The volume is full, or the device is write protected.
- There is insufficient space for File Control Blocks.

MAC—64K storage limit exceeded

64K words of work file memory are available to MACRO-11. This message indicates that the assembler has generated so many symbols (about 13,000 to 14,000) that it has run out of space. Either the source program is too large to start with, or it contains a condition that leads to excessive size, such as a macro expansion that recursively calls itself without a terminating condition.
9.1 MACRO–11 Under RSTS

The only way a MACRO–11 program can run on a RSTS system is through either the RT–11 or RSX run-time systems.

9.1.1 RT–11 Through RSTS

There are two ways to run MACRO–11 under the RT–11 run-time system on RSTS:

- Use the RT–11 Emulator. This is done by typing `SW RT11`. The terminal will respond with the RT–11 prompt (a dot printed by the keyboard monitor). You can then use the RT–11 commands (see Section 9.2).

- Type the command `RUN $MACRO.SAV`. The terminal will respond with an asterisk (*) prompt. You can then enter a command string of the form:

  `objfil,lstfil=src1,...src6`

  where:

  - `objfil` is an object (output) file with the default extension .OBJ.
  - `lstfil` is a listing (output) file with the default extension .LST.
  - `src1,...src6` are source (input) files with the default extension .MAC. Six input files are allowed in this command.

9.1.2 RSX Through RSTS

To run MACRO–11 under the RSX run-time system on RSTS, type the command: `RUN $MAC.TSK`. The terminal will display:

  `MAC>

In response, enter a command string of the form:

  `objfil,lstfil=src1,...srcn`

where:

- `objfil` is an object (output) file with the default extension .OBJ.
- `lstfil` is a listing (output) file with the default extension .LST.
- `src1,...srcn` are source (input) files with the default extension .MAC.
NOTE
You can use other RSTS commands to call the RT-11 and RSX run-time systems, but they are site dependent and so are not mentioned here.

9.2 Running MACRO–11 Under RT–11

The following sections describe those MACRO–11 operating procedures that apply only to the RT–11 system. Table 9–1 lists the default file specifications for RT–11.

Table 9–1: RT–11 Default File Specification Values

<table>
<thead>
<tr>
<th>File</th>
<th>Device</th>
<th>Filename</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>DK:</td>
<td>Must specify</td>
<td>.OBJ</td>
</tr>
<tr>
<td>Listing</td>
<td>Same as for object file</td>
<td>Must specify</td>
<td>.LST</td>
</tr>
<tr>
<td>CREF</td>
<td>Logical device name CF:, if it has been defined; otherwise, DK:</td>
<td>CREF</td>
<td>.TMP</td>
</tr>
<tr>
<td>Work</td>
<td>Logical device name WF:, if it has been defined; otherwise, DK:</td>
<td>WRK</td>
<td>.TMP</td>
</tr>
<tr>
<td>First source</td>
<td>DK:</td>
<td>Must specify</td>
<td>.MAC</td>
</tr>
<tr>
<td>Additional source</td>
<td>Same as for preceding source file</td>
<td>Must specify</td>
<td>.MAC</td>
</tr>
<tr>
<td>System macro library</td>
<td>System device SY:</td>
<td>SYSMAC</td>
<td>.SML</td>
</tr>
<tr>
<td>User macro library</td>
<td>DK: if first file; otherwise, same as for preceding source file</td>
<td>Must specify</td>
<td>.MLB</td>
</tr>
</tbody>
</table>

9.2.1 RT–11 Command String (CSI) Format

To call the MACRO–11 assembler from the system device, respond to the system prompt (a dot printed by the keyboard monitor) by typing:

```
.R MACRO
```

When the assembler responds with an asterisk (*), it is ready to accept command string (CSI) input.

Format:

```
dev:obj,dev:list,dev:cref/s:arg=dev:src1,src2,...,dev:srcn/s:arg
```

where:
dev: is any valid RT–11 device for output; any file-structured device for input. If dev: is omitted, DK: is assumed.

obj is the file specification of the binary object file that the assembly process produces; the device for this file should not be TT: or LP:.

list is the file specification of the assembly and symbol listing that the assembly process produces.

cref is the file specification of the CREF temporary cross-reference file that the assembly process produces. Omission of dev:cref does not preclude a cross-reference listing, however. If you specify /C without a CREF filename, MACRO–11 uses a default name for the CREF temporary file on logical device name CF:, if it has been defined, or on DK:.

/s:arg is a set of file specification options and arguments (see Table 9–2).

src1,src2,...srcn represent the ASCII source (input) files containing the MACRO–11 source program or the user-supplied macro library files to be assembled. You can specify as many as six source files.

The following command string calls for an assembly that uses one source file plus the system macro library to produce an object file BINF.OBJ and a listing. The listing goes directly to the line printer.

*DK: BINF.OBJ, LP:=DK:SRC.MAC

All output file specifications are optional. The system does not produce an output file (except for the CREF temporary file, if you include the /C option), unless the command string contains a specification for that file.

The system determines the file type of an output file specification by its position in the command string, as determined by the number of commas in the string. For example, to omit the object file, you must begin the command string with a comma. The following command produces a listing, including a cross-reference table, but not a binary object file.

*,LP://C=SRC1,SRC2

Notice that you need not include a comma after the final output file specification in the command string.

Table 9–1 lists the default values for each file specification.

Some assemblies need more symbol table space than available memory can contain. When this occurs, the system automatically creates a temporary work file called WRK.TMP to provide extended symbol table space.

MACRO–11 writes WRK.TMP to the logical name WF:, if it has been defined. Otherwise, MACRO–11 puts the work file on DK:. To assign the logical name WF: to a device, enter the following command:

.ASSIGN dev: WF

where:

dev: is the file-structured device that will hold WRK.TMP.
The default size of \texttt{WRK\_TMP} is 200\textsubscript{g} blocks. You can increase the size to a maximum of 400\textsubscript{g} blocks with a customization patch. Refer to the file \texttt{CUSTOM\_TXT} on your RT--11 distribution kit.

### 9.2.2 RT--11 CSI Command Line Options

At assembly time, you may need to override certain MACRO directives appearing in the source programs. You may also need to direct MACRO--11 on the handling of certain files during assembly. You can satisfy these needs by using the switches described in Table 9--2.

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(/L:\text{arg}) (/N:\text{arg})</td>
<td>Listing control options; these options accept ASCII values (\text{arg}) which are equivalent in function and name to the arguments for the .LIST and .NLIST directives you can include in your source program (see Section 6.1.1). Arguments that you specify with the (/L:\text{arg}) and (/N:\text{arg}) options override any arguments that you may have specified with the .LIST and .NLIST directives and remain in effect for the entire assembly process.</td>
</tr>
<tr>
<td>(/E:\text{arg}) (/D:\text{arg})</td>
<td>Function control options; these options accept ASCII values (\text{arg}) which are equivalent in function and name to the arguments for the .ENABL and .DSABL directives you can include in your source program (see Section 6.2.1). Arguments that you specify with the (/E:\text{arg}) and (/D:\text{arg}) options override any arguments that you may have specified with the .ENABL and .DSABL directives and remain in effect for the entire assembly process.</td>
</tr>
</tbody>
</table>
| \(/M\) | Indicates input file is a MACRO library file. When the assembler encounters a .MCALL directive in the source code, it searches macro libraries according to their order of appearance in the command string, starting from the right. When it locates a macro record whose name matches that given in the .MCALL, it assembles the macro as indicated by that definition. Thus, if two or more macro libraries contain definitions of the same macro name, the macro library that appears rightmost in the command string takes precedence. Consider the following command string:  

```
*(output file specification)=ALIB/M,BLIB/M,XIZ
```

Assume that each of the two macro libraries, ALIB\_MLB and BLIB\_MLB, contains a macro called .BIG, but with different definitions. Then, if source file XIZ contains a macro call .MCALL .BIG, the system includes the definition of .BIG in the program as it appears in the macro library BLIB. If the command string does not include the standard system macro library SYSMAC\_SML, the system automatically includes it as the first source file in the command string. Therefore, if macro library ALIB\_MLB contains a definition of a macro called .READ, that definition of .READ overrides the standard .READ macro definition in SYSMAC\_SML. |
| \(/C:\text{arg}\) | Controls contents of cross-reference table. |
The \%/w option affects only the source file to which it is appended. The other options affect the entire command string.

### 9.2.3 RT–11 Digital Command Language (DCL) Format

You can enter the MACRO DCL command in response to the monitor prompt (.) to run MACRO–11 under RT–11.

**Format:**

\[
\text{MACRO} \left[ \begin{array}{l}
/\text{CROSSREFERENCE}[:\text{type}[:\ldots:\text{type}]] \\
/\text{DISABLE}[:\text{type}[:\ldots:\text{type}]] \\
/\text{ENABLE}[:\text{type}[:\ldots:\text{type}]] \\
/\text{LIST}[:\text{filespec}] \\
/\text{ALLOCATE}[:\text{size}] \\
/\text{NOOBJECT}[:\text{filespec}] \\
/\text{ALLOCATE}[:\text{size}] \\
/\text{NO}SHOW[:\text{type}[:\ldots:\text{type}]] \\
\end{array} \right] \text{filespecs} [/\text{LIBRARY}]
\]

**where:**

- `/ALLOCATE:size` reserves space for output file; a size of –1 reserves the largest possible space.

- `/CROSSREFERENCE[:\text{type}[:\ldots:\text{type}]]`

  produces CREF listing; \text{type} can be:
  - C Control section names
  - E Error codes
  - M Macro names
  - P Permanent names
  - R Register symbols
  - S user-defined symbols
  - blank equivalent to :E:M:S

- `/DISABLE[:\text{type}[:\ldots:\text{type}]]`

  specifies .DSABL directives; \text{type} can be:
  - ABS Produces absolute binary output
  - AMA Assembles absolute addresses as relative addresses
  - CDR Treats source columns beyond 72 as a comment
  - DBG Writes internal symbol director (ISD) records
  - FTP Truncates floating point
  - GBL Assumes undefined symbols are globals
  - LC Accepts lowercase characters in source programs
  - LSB Defines local symbol block
  - MCL Enables or disables automatic .MCALL
  - PNC Enables or disables binary output
  - REG Defines default register mnemonics

- `/ENABLE[:\text{type}[:\ldots:\text{type}]]`

  specifies .ENABL directives; \text{type} can be any of the types listed under `/DISABLE`.

- `/LIBRARY`

  identifies a macro library file.
/LIST[filespec] writes program listing to the printer or to filespec.

[NO]OBJECT[filespec] [does not] generate a .OBJ file; output filespec defaults to input filespec.

[NO]SHOW:type[...type] specifies MACRO-11 .LIST and .NLIST directives; type can be:

- BEX Extended binary code
- BIN Generated binary code
- CND Unsatisfied conditionals and .IF and .ENDC statements
- COM Comments
- LOC Location counter
- MC Macro calls, repeat range expansions
- MD Macro definitions, repeat range expansions
- ME Macro expansions
- MEB Macro expansions, binary code
- SEQ Source line sequence numbers
- SRC Source code
- SYM Symbol table
- TOC Table of contents
- TTM Wide or narrow listing format

### 9.3 Cross-Reference (CREF) Table Generation Option

A cross-reference (CREF) table lists all or a subset of the symbols in a source program, identifying the statements that define and use symbols.

#### 9.3.1 Obtaining a Cross-Reference Table

To obtain a CREF table you must include the /C:arg option in the command string. Usually you include the /C:arg option with the assembly listing file specification, but it can appear anywhere on the command line.

If the command string does not include a CREF file specification but does include /C, MACRO-11 automatically writes a temporary file CREF.TMP to logical name GF:, if it has been defined. Otherwise, MACRO-11 uses DK:. If you want to use a device other than DK: for the temporary CREF workfile, include the dev:cref field in the command string, or assign the logical name GF: to the device you want to use.

A complete CREF listing contains the following sections:

- A cross reference of program symbols—labels used in the program and symbols followed by an operator.
- A cross reference of register symbols—R0, R1, R2, R3, R4, R5, SP, and PC.
- A cross reference of MACRO symbols—symbols defined by .MACRO and .MCALL directives.
- A cross reference of permanent symbols—all operation mnemonics and assembler directives.
- A cross reference of program sections—the names you specify as operands of .CSECT or .PSECT directives.

- A cross reference of errors—all flagged errors from the assembly, grouped and listed by type.

You can include any or all of these sections on the cross-reference listing by specifying the appropriate arguments with the /C option. These arguments are listed and described in Table 9–3.

<table>
<thead>
<tr>
<th>Argument</th>
<th>CREF Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>User defined symbols</td>
</tr>
<tr>
<td>R</td>
<td>Register symbols</td>
</tr>
<tr>
<td>M</td>
<td>MACRO symbolic names</td>
</tr>
<tr>
<td>P</td>
<td>Permanent symbols including instructions and directives</td>
</tr>
<tr>
<td>C</td>
<td>Control and program sections</td>
</tr>
<tr>
<td>E</td>
<td>Error code grouping</td>
</tr>
</tbody>
</table>

**NOTE**

Specifying /C with no arguments is equivalent to specifying /C:S:M:E. That special case excepted, you must explicitly request each CREF section by including its arguments. No cross-reference file is written if you omit the /C option, even if the command string includes a CREF file specification.

### 9.3.2 Handling Cross-Reference Table Files

When you request a cross-reference listing with the /C option, MACRO–11 generates a temporary file CREF.TMP and writes this file to logical device name CF:, if it is defined. Otherwise, MACRO–11 writes CREF.TMP to DK:.

If the device MACRO–11 attempts to use for CREF.TMP is write-protected, or if it contains insufficient free space for the temporary file, you can specify another device for the file in your command string. To use another device, you can specify a third output file in the command string; that is, include a dev:cref specification for the CREF temporary file in addition to the file specifications for the binary and listing files. (You must still include the /C option to control the form and content of the listing. The dev:cref specification is ignored if the /C option is not also present in the command string.)

MACRO–11 then uses dev:cref instead of CF:CREF.TMP or DK:CREF.TMP. In any case, CREF deletes the file automatically after producing the CREF listing.

For example, with the following command string MACRO–11 uses RK2:TEMP.TMP as the temporary CREF file:

```
* ,LP:, RK2:TEMP.TMP=SOURCE/C
```
Another way to assign an alternative device for the CREF.TMP file is to assign the logical name CF: to the device you want to use for CREF.TMP, prior to running MACRO-11:

`.ASSIGN dev: CF`

This method is convenient if you intend to do several assemblies, as it relieves you from having to include a dev:cref specification for the CREF file in each command string. If you enter the `ASSIGN dev: CF` command, and later include a cref file specification in a command string, the specification in the command string prevails for that assembly only.

The system lists requested cross-reference tables following the MACRO assembly listing. Each table begins on a new page.

The system prints symbols and also symbol values, control sections, and error codes, if applicable, beginning at the left margin of the page. References to each symbol are listed on the same line, left-to-right across the page. The system lists references in the form P-L; where P is the page on which the symbol, control section, or error code appears, and L is the line number on the page.

A number sign (#) next to a reference indicates a symbol definition. An asterisk (*) next to a reference indicates a destructive reference—an operation that alters the contents of the addressed location.

### 9.3.3 MACRO-11 Error Messages Under RT-11

MACRO-11 writes an error message to the command output device when one of the error conditions described below is detected. MACRO-11 writes below the error message the command line that caused the error. If the error is a `.INCLUDE` or a `.LIBRARY` directive file error, MACRO-11 writes both the source line and the command line that caused the error.

```
?MACRO-s-Error message
MACRO-11 source line
MACRO-11 command line
```

The s in the error message represents the letter code that indicates the severity level of the error.

These error messages reflect operational problems and should not be confused with the error codes (see Appendix D) produced by MACRO-11 during assembly.
<table>
<thead>
<tr>
<th>Message and Meaning</th>
<th>How to Respond</th>
</tr>
</thead>
<tbody>
<tr>
<td>?MACRO–F–Device full (&lt;\text{dev:}&gt;)</td>
<td>• Delete unnecessary files from the output volume, perhaps transferring them to a backup volume.</td>
</tr>
<tr>
<td></td>
<td>• Use another volume with more space.</td>
</tr>
<tr>
<td></td>
<td>• Specify an explicit output file size by using the /ALLOCATE option or include the file size in square brackets as part of the output file specification.</td>
</tr>
<tr>
<td></td>
<td>• Consolidate free space on the volume by using the monitor’s SQUEEZE command.</td>
</tr>
<tr>
<td></td>
<td>• Refer to other techniques for gaining file space in the \textit{RT–11 System Message Manual}.</td>
</tr>
<tr>
<td>?MACRO–F–File not found (&lt;\text{dev:filnam.typ}&gt;)</td>
<td>Correct any file specification errors in the command line and enter it again.</td>
</tr>
<tr>
<td>?MACRO–F–.INCLUDE directive file error</td>
<td>1. Check for a typing error in the command line. Use file specifications that are valid with the .INCLUDE directive.</td>
</tr>
<tr>
<td></td>
<td>2. Enter the command line again, specifying an available device, or load the device handler.</td>
</tr>
<tr>
<td></td>
<td>3. Make sure that the .INCLUDE stacking depth does not exceed five.</td>
</tr>
<tr>
<td>?MACRO–F–Insufficient memory</td>
<td>• Use the SHOW command to find out what device handlers are loaded, then use the UNLOAD command to remove those that are not necessary. After unloading any unnecessary handlers, you may need to unload, then reload, handlers that you plan to use so that free space is concatenated. Be careful not to unload any handler being used by a foreground or system job.</td>
</tr>
</tbody>
</table>

MACRO–11 lacks the minimum amount of memory (16K words) necessary to run.
<table>
<thead>
<tr>
<th>Message and Meaning</th>
<th>How to Respond</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Terminate and unload the foreground job or a system job.</td>
<td></td>
</tr>
<tr>
<td>• If you are using the FB monitor, SET USR SWAP (see the RT-11 System User’s Guide) to allow USR swapping.</td>
<td></td>
</tr>
<tr>
<td>• Create a new monitor with SYSGEN (see the RT-11 System Generation Guide) containing only those features that you absolutely need.</td>
<td></td>
</tr>
<tr>
<td>• If you have extended memory available, use VBGEXE to run MACRO-11.</td>
<td></td>
</tr>
<tr>
<td>?MACRO-F-Internal error</td>
<td>This error should not occur. If you get this error, please send an SPR to DIGITAL along with a method of reproducing the problem.</td>
</tr>
<tr>
<td>MACRO-11 detected an unexpected condition while checking its internal tables.</td>
<td></td>
</tr>
<tr>
<td>?MACRO-F-Invalid command</td>
<td>Correct and retype the command line.</td>
</tr>
<tr>
<td>The command line contains a syntax error or specifies more than six input files.</td>
<td></td>
</tr>
<tr>
<td>?MACRO-F-Invalid device</td>
<td>Install the device or substitute another.</td>
</tr>
<tr>
<td>The device specified in the command line is not on the system.</td>
<td></td>
</tr>
<tr>
<td>?MACRO-F-Invalid macro library</td>
<td>Obtain a new copy of SYSMAC.SML from your distribution kit. If you have modified SYSMAC.SML, carefully check the procedures you used.</td>
</tr>
<tr>
<td>The library file has been corrupted, or it was not produced by the RT-11 librarian, LIBR.</td>
<td></td>
</tr>
<tr>
<td>?MACRO-F-Invalid option: /x</td>
<td>Check for a typing error in the command line. Use only a valid listing control or a functional control (or CREF) option.</td>
</tr>
<tr>
<td>The specified option was not recognized by the program.</td>
<td></td>
</tr>
<tr>
<td>?MACRO-F-I/O error on <a href="">dev:filnam.typ</a></td>
<td>• Be sure the device is on line and write enabled.</td>
</tr>
<tr>
<td>A hardware error occurred during a read from or write to the specified file.</td>
<td></td>
</tr>
<tr>
<td>• Refer to other procedures for recovery from hard error conditions listed in the RT-11 System Message Manual.</td>
<td></td>
</tr>
<tr>
<td>Message and Meaning</td>
<td>How to Respond</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>?MACRO-F- I/O error on workfile</td>
<td>• Be sure the device is on line and write enabled.</td>
</tr>
<tr>
<td></td>
<td>• Be sure there is enough contiguous free space on the output volume to</td>
</tr>
<tr>
<td></td>
<td>accommodate the workfile. If not, use the monitor's SQUEEZE command, or</td>
</tr>
<tr>
<td></td>
<td>delete unnecessary files.</td>
</tr>
<tr>
<td></td>
<td>• Refer to other procedures for recovery from hard error conditions listed in</td>
</tr>
<tr>
<td></td>
<td>the RT-11 System Message Manual.</td>
</tr>
<tr>
<td>?MACRO-F-.LIBRARY directive file error</td>
<td>1. Check for a typing error in the command line. Use file specifications that</td>
</tr>
<tr>
<td>1. The file specified in the .LIBRARY</td>
<td>are valid with the .LIBRARY directive.</td>
</tr>
<tr>
<td>2. The file specification in the .LIBRARY</td>
<td>2. Make sure that the file specification used in the .LIBRARY directive is</td>
</tr>
<tr>
<td>2. The file specification in the .LIBRARY</td>
<td>for a random-access device.</td>
</tr>
<tr>
<td>3. The device specified in the command</td>
<td>3. Enter the command line again, specifying an available device.</td>
</tr>
<tr>
<td>4. The .LIBRARY stacking depth exceeds the maximum depth allowed.</td>
<td>4. Make sure that the .LIBRARY stacking depth does not exceed the maximum</td>
</tr>
<tr>
<td></td>
<td>depth allowed.</td>
</tr>
<tr>
<td>?MACRO-F-Protected file already exists <a href="">dev:filnam.typ</a></td>
<td>Use the monitor UNPROTECT command to change the protection level of the</td>
</tr>
<tr>
<td>An attempt was made to create a file</td>
<td>existing file, or use a different name to create the new file.</td>
</tr>
<tr>
<td>having the same name as an existing protected file.</td>
<td></td>
</tr>
<tr>
<td>?MACRO-F-Storage limit exceeded (64K)</td>
<td>Check the program logic for a condition that leads to excessive size, such</td>
</tr>
<tr>
<td>MACRO’s virtual symbol table can store symbols and macros up to 64K words (400K</td>
<td>as a macro expansion that recursively calls itself without a terminating</td>
</tr>
<tr>
<td>blocks) in any combination. The program contains more than 64K of one or both of</td>
<td>condition. If necessary, reduce the requirements of the source program by</td>
</tr>
<tr>
<td>these elements.</td>
<td>segmenting it into separate modules, and assemble each separately.</td>
</tr>
<tr>
<td>Message and Meaning</td>
<td>How to Respond</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| ?MACRO-F-Workfile space exceeded | - Increase the size of the workfile by patching location WRKSIZ. (Refer to the file CUSTOM.TXT on your distribution kit.) The default size of the workfile is 200₈ blocks; it can be patched to a maximum size of 400₈ blocks.  
- Check the program logic for a condition that leads to excessive size, such as a macro expansion that recursively calls itself without a terminating condition. You may also have a missing .ENDM or .ENDR statement. If necessary, reduce the requirements of the source program by segmenting it into separate modules, and assemble each separately. |
| ?MACRO-W-I/O error on CREF file: CREF aborted | - Delete unnecessary files from the output volume, perhaps transferring them to a backup volume.  
- Use another volume with more space.  
- Include the CREF workfile specification in the MACRO-11 command line, and include the file size in square brackets as part of the file specification.  
- Consolidate free space on the volume by using the monitor's SQUEEZE command.  
- Refer to other techniques for gaining file space in the RT-11 System Message Manual. |

A.1 DEC Multinational Character Set

Empty positions are reserved for future standardizations.

Table A-1: DEC Multinational Character Set

<table>
<thead>
<tr>
<th>Left Byte Octal</th>
<th>Right Byte Octal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Character</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>000</td>
<td>00</td>
<td>0</td>
<td>NUL</td>
<td>Null; tape feed; CTRL/@</td>
</tr>
<tr>
<td>000400</td>
<td>001</td>
<td>01</td>
<td>1</td>
<td>SOH</td>
<td>Start of heading; SOM, start of message; CTRL/A</td>
</tr>
<tr>
<td>001000</td>
<td>002</td>
<td>02</td>
<td>2</td>
<td>STX</td>
<td>Start of text; EOA, end of address; CTRL/B</td>
</tr>
<tr>
<td>001400</td>
<td>003</td>
<td>03</td>
<td>3</td>
<td>ETX</td>
<td>End of text; EOM, end of message; CTRL/C</td>
</tr>
<tr>
<td>002000</td>
<td>004</td>
<td>04</td>
<td>4</td>
<td>EOT</td>
<td>End of transmission (END); shuts off TWX terminals; CTRL/D</td>
</tr>
<tr>
<td>002400</td>
<td>005</td>
<td>05</td>
<td>5</td>
<td>ENQ</td>
<td>Enquiry (ENQRY); WRU; CTRL/E</td>
</tr>
<tr>
<td>003000</td>
<td>006</td>
<td>06</td>
<td>6</td>
<td>ACK</td>
<td>Acknowledge; RU; CTRL/F</td>
</tr>
<tr>
<td>003400</td>
<td>007</td>
<td>07</td>
<td>7</td>
<td>BEL</td>
<td>Rings the bell; CTRL/G</td>
</tr>
<tr>
<td>004000</td>
<td>010</td>
<td>08</td>
<td>8</td>
<td>BS</td>
<td>Backspace; FEO, format effector; backspaces some terminals; CTRL/H</td>
</tr>
<tr>
<td>004400</td>
<td>011</td>
<td>09</td>
<td>9</td>
<td>HT</td>
<td>Horizontal tab; CTRL/I</td>
</tr>
<tr>
<td>005000</td>
<td>012</td>
<td>0A</td>
<td>10</td>
<td>LF</td>
<td>Line feed or Line space (new line); CTRL/J</td>
</tr>
<tr>
<td>005400</td>
<td>013</td>
<td>0B</td>
<td>11</td>
<td>VT</td>
<td>Vertical tab (VTAB); CTRL/K</td>
</tr>
<tr>
<td>006000</td>
<td>014</td>
<td>0C</td>
<td>12</td>
<td>FF</td>
<td>Form feed to top of next page (PAGE); CTRL/L</td>
</tr>
<tr>
<td>006400</td>
<td>015</td>
<td>0D</td>
<td>13</td>
<td>CR</td>
<td>Carriage return to beginning of line; CTRL/M</td>
</tr>
<tr>
<td>007000</td>
<td>016</td>
<td>0E</td>
<td>14</td>
<td>SO</td>
<td>Shift out; changes ribbon color to red; CTRL/N</td>
</tr>
<tr>
<td>007400</td>
<td>017</td>
<td>0F</td>
<td>15</td>
<td>SI</td>
<td>Shift in; changes ribbon color to black; CTRL/O</td>
</tr>
<tr>
<td>010000</td>
<td>020</td>
<td>10</td>
<td>16</td>
<td>DLE</td>
<td>Data link escape; DC0; CTRL/P</td>
</tr>
<tr>
<td>010400</td>
<td>021</td>
<td>11</td>
<td>17</td>
<td>DC1</td>
<td>Device control 1; turns transmitter (READER) on; XON; CTRL/Q</td>
</tr>
<tr>
<td>011000</td>
<td>022</td>
<td>12</td>
<td>18</td>
<td>DC2</td>
<td>Device control 2; turns punch or auxiliary on; TAPE; AUX ON; CTRL/R</td>
</tr>
<tr>
<td>011400</td>
<td>023</td>
<td>13</td>
<td>19</td>
<td>DC3</td>
<td>Device control 3; turns transmitter (READER) off; XOFF; CTRL/S</td>
</tr>
</tbody>
</table>
### Table A–1 (Cont.): DEC Multinational Character Set

<table>
<thead>
<tr>
<th>Left Byte Octal</th>
<th>Right Byte Octal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Character</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>012000</td>
<td>024</td>
<td>14</td>
<td>20</td>
<td>DC4</td>
<td>Device control 4; turns punch or auxiliary off; AUX OFF; CTRL/T</td>
</tr>
<tr>
<td>012400</td>
<td>025</td>
<td>15</td>
<td>21</td>
<td>NAK</td>
<td>Negative acknowledge; ERR; ERROR; CTRL/U</td>
</tr>
<tr>
<td>013000</td>
<td>026</td>
<td>16</td>
<td>22</td>
<td>SYN</td>
<td>Synchronous file (SYNC); CTRL/V</td>
</tr>
<tr>
<td>013400</td>
<td>027</td>
<td>17</td>
<td>23</td>
<td>ETB</td>
<td>End of transmission block; LEM, logical end of medium; CTRL/W</td>
</tr>
<tr>
<td>014000</td>
<td>030</td>
<td>18</td>
<td>24</td>
<td>CAN</td>
<td>Cancel (CANC); CTRL/X</td>
</tr>
<tr>
<td>014400</td>
<td>031</td>
<td>19</td>
<td>25</td>
<td>EM</td>
<td>End of medium; CTRL/Y</td>
</tr>
<tr>
<td>015000</td>
<td>032</td>
<td>1A</td>
<td>26</td>
<td>SUB</td>
<td>Substitute; CTRL/Z</td>
</tr>
<tr>
<td>015400</td>
<td>033</td>
<td>1B</td>
<td>27</td>
<td>ESC</td>
<td>Escape; CTRL/\</td>
</tr>
<tr>
<td>016000</td>
<td>034</td>
<td>1C</td>
<td>28</td>
<td>FS</td>
<td>File separator; CTRL/\</td>
</tr>
<tr>
<td>016400</td>
<td>035</td>
<td>1D</td>
<td>29</td>
<td>GS</td>
<td>Group separator; CTRL/\</td>
</tr>
<tr>
<td>017000</td>
<td>036</td>
<td>1E</td>
<td>30</td>
<td>RS</td>
<td>Record separator; CTRL/*</td>
</tr>
<tr>
<td>017400</td>
<td>037</td>
<td>1F</td>
<td>31</td>
<td>US</td>
<td>Unit separator; CTRL/~</td>
</tr>
<tr>
<td>020000</td>
<td>040</td>
<td>20</td>
<td>32</td>
<td>SP</td>
<td>Space</td>
</tr>
<tr>
<td>020400</td>
<td>041</td>
<td>21</td>
<td>33</td>
<td>!</td>
<td>Exclamation mark</td>
</tr>
<tr>
<td>021000</td>
<td>042</td>
<td>22</td>
<td>34</td>
<td>&quot;</td>
<td>Double quote</td>
</tr>
<tr>
<td>021400</td>
<td>043</td>
<td>23</td>
<td>35</td>
<td>#</td>
<td>Number sign</td>
</tr>
<tr>
<td>022000</td>
<td>044</td>
<td>24</td>
<td>36</td>
<td>$</td>
<td>Dollar sign</td>
</tr>
<tr>
<td>022400</td>
<td>045</td>
<td>25</td>
<td>37</td>
<td>%</td>
<td>Percent sign</td>
</tr>
<tr>
<td>023000</td>
<td>046</td>
<td>26</td>
<td>38</td>
<td>&amp;</td>
<td>Ampersand</td>
</tr>
<tr>
<td>023400</td>
<td>047</td>
<td>27</td>
<td>39</td>
<td>'</td>
<td>Single quote; apostrophe; accent acute</td>
</tr>
<tr>
<td>024000</td>
<td>050</td>
<td>28</td>
<td>40</td>
<td>(</td>
<td>Left parenthesis</td>
</tr>
<tr>
<td>024400</td>
<td>051</td>
<td>29</td>
<td>41</td>
<td>)</td>
<td>Right parenthesis</td>
</tr>
<tr>
<td>025000</td>
<td>052</td>
<td>2A</td>
<td>42</td>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td>025400</td>
<td>053</td>
<td>2B</td>
<td>43</td>
<td>+</td>
<td>Plus sign</td>
</tr>
<tr>
<td>026000</td>
<td>054</td>
<td>2C</td>
<td>44</td>
<td>,</td>
<td>Comma</td>
</tr>
<tr>
<td>026400</td>
<td>055</td>
<td>2D</td>
<td>45</td>
<td>-</td>
<td>Minus sign or hyphen</td>
</tr>
<tr>
<td>027000</td>
<td>056</td>
<td>2E</td>
<td>46</td>
<td>.</td>
<td>Period</td>
</tr>
<tr>
<td>027400</td>
<td>057</td>
<td>2F</td>
<td>47</td>
<td>/</td>
<td>Slash</td>
</tr>
<tr>
<td>030000</td>
<td>060</td>
<td>30</td>
<td>48</td>
<td>0</td>
<td>Number zero</td>
</tr>
<tr>
<td>030400</td>
<td>061</td>
<td>31</td>
<td>49</td>
<td>1</td>
<td>Number one</td>
</tr>
<tr>
<td>031000</td>
<td>062</td>
<td>32</td>
<td>50</td>
<td>2</td>
<td>Number two</td>
</tr>
<tr>
<td>031400</td>
<td>063</td>
<td>33</td>
<td>51</td>
<td>3</td>
<td>Number three</td>
</tr>
<tr>
<td>032000</td>
<td>064</td>
<td>34</td>
<td>52</td>
<td>4</td>
<td>Number four</td>
</tr>
<tr>
<td>032400</td>
<td>065</td>
<td>35</td>
<td>53</td>
<td>5</td>
<td>Number five</td>
</tr>
<tr>
<td>033000</td>
<td>066</td>
<td>36</td>
<td>54</td>
<td>6</td>
<td>Number six</td>
</tr>
<tr>
<td>033400</td>
<td>067</td>
<td>37</td>
<td>55</td>
<td>7</td>
<td>Number seven</td>
</tr>
<tr>
<td>034000</td>
<td>070</td>
<td>38</td>
<td>56</td>
<td>8</td>
<td>Number eight</td>
</tr>
<tr>
<td>034400</td>
<td>071</td>
<td>39</td>
<td>57</td>
<td>9</td>
<td>Number nine</td>
</tr>
<tr>
<td>035000</td>
<td>072</td>
<td>3A</td>
<td>58</td>
<td>:</td>
<td>Colon</td>
</tr>
<tr>
<td>035400</td>
<td>073</td>
<td>3B</td>
<td>59</td>
<td>;</td>
<td>Semicolon</td>
</tr>
<tr>
<td>036000</td>
<td>074</td>
<td>3C</td>
<td>60</td>
<td>&lt;</td>
<td>Left angle bracket</td>
</tr>
<tr>
<td>036400</td>
<td>075</td>
<td>3D</td>
<td>61</td>
<td>=</td>
<td>Equal sign</td>
</tr>
<tr>
<td>Left Byte Octal</td>
<td>Right Byte Octal</td>
<td>Hex</td>
<td>Decimal</td>
<td>Character</td>
<td>Remarks</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------</td>
<td>------</td>
<td>---------</td>
<td>-----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>037000</td>
<td>076</td>
<td>3E</td>
<td>62</td>
<td>&gt;</td>
<td>Right angle bracket</td>
</tr>
<tr>
<td>037400</td>
<td>077</td>
<td>3F</td>
<td>63</td>
<td>?</td>
<td>Question mark</td>
</tr>
<tr>
<td>040000</td>
<td>100</td>
<td>40</td>
<td>64</td>
<td>@</td>
<td>At sign</td>
</tr>
<tr>
<td>040400</td>
<td>101</td>
<td>41</td>
<td>65</td>
<td>A</td>
<td>Uppercase A</td>
</tr>
<tr>
<td>041000</td>
<td>102</td>
<td>42</td>
<td>66</td>
<td>B</td>
<td>Uppercase B</td>
</tr>
<tr>
<td>041400</td>
<td>103</td>
<td>43</td>
<td>67</td>
<td>C</td>
<td>Uppercase C</td>
</tr>
<tr>
<td>042000</td>
<td>104</td>
<td>44</td>
<td>68</td>
<td>D</td>
<td>Uppercase D</td>
</tr>
<tr>
<td>042400</td>
<td>105</td>
<td>45</td>
<td>69</td>
<td>E</td>
<td>Uppercase E</td>
</tr>
<tr>
<td>043000</td>
<td>106</td>
<td>46</td>
<td>70</td>
<td>F</td>
<td>Uppercase F</td>
</tr>
<tr>
<td>043400</td>
<td>107</td>
<td>47</td>
<td>71</td>
<td>G</td>
<td>Uppercase G</td>
</tr>
<tr>
<td>044000</td>
<td>110</td>
<td>48</td>
<td>72</td>
<td>H</td>
<td>Uppercase H</td>
</tr>
<tr>
<td>044400</td>
<td>111</td>
<td>49</td>
<td>73</td>
<td>I</td>
<td>Uppercase I</td>
</tr>
<tr>
<td>045000</td>
<td>112</td>
<td>4A</td>
<td>74</td>
<td>J</td>
<td>Uppercase J</td>
</tr>
<tr>
<td>045400</td>
<td>113</td>
<td>4B</td>
<td>75</td>
<td>K</td>
<td>Uppercase K</td>
</tr>
<tr>
<td>046000</td>
<td>114</td>
<td>4C</td>
<td>76</td>
<td>L</td>
<td>Uppercase L</td>
</tr>
<tr>
<td>046400</td>
<td>115</td>
<td>4D</td>
<td>77</td>
<td>M</td>
<td>Uppercase M</td>
</tr>
<tr>
<td>047000</td>
<td>116</td>
<td>4E</td>
<td>78</td>
<td>N</td>
<td>Uppercase N</td>
</tr>
<tr>
<td>047400</td>
<td>117</td>
<td>4F</td>
<td>79</td>
<td>O</td>
<td>Uppercase O</td>
</tr>
<tr>
<td>050000</td>
<td>120</td>
<td>50</td>
<td>80</td>
<td>P</td>
<td>Uppercase P</td>
</tr>
<tr>
<td>050400</td>
<td>121</td>
<td>51</td>
<td>81</td>
<td>Q</td>
<td>Uppercase Q</td>
</tr>
<tr>
<td>051000</td>
<td>122</td>
<td>52</td>
<td>82</td>
<td>R</td>
<td>Uppercase R</td>
</tr>
<tr>
<td>051400</td>
<td>123</td>
<td>53</td>
<td>83</td>
<td>S</td>
<td>Uppercase S</td>
</tr>
<tr>
<td>052000</td>
<td>124</td>
<td>54</td>
<td>84</td>
<td>T</td>
<td>Uppercase T</td>
</tr>
<tr>
<td>052400</td>
<td>125</td>
<td>55</td>
<td>85</td>
<td>U</td>
<td>Uppercase U</td>
</tr>
<tr>
<td>053000</td>
<td>126</td>
<td>56</td>
<td>86</td>
<td>V</td>
<td>Uppercase V</td>
</tr>
<tr>
<td>053400</td>
<td>127</td>
<td>57</td>
<td>87</td>
<td>W</td>
<td>Uppercase W</td>
</tr>
<tr>
<td>054000</td>
<td>130</td>
<td>58</td>
<td>88</td>
<td>X</td>
<td>Uppercase X</td>
</tr>
<tr>
<td>054400</td>
<td>131</td>
<td>59</td>
<td>89</td>
<td>Y</td>
<td>Uppercase Y</td>
</tr>
<tr>
<td>055000</td>
<td>132</td>
<td>5A</td>
<td>90</td>
<td>Z</td>
<td>Uppercase Z</td>
</tr>
<tr>
<td>055400</td>
<td>133</td>
<td>5B</td>
<td>91</td>
<td>[</td>
<td>Left square bracket</td>
</tr>
<tr>
<td>056000</td>
<td>134</td>
<td>5C</td>
<td>92</td>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>056400</td>
<td>135</td>
<td>5D</td>
<td>93</td>
<td>]</td>
<td>Right square bracket</td>
</tr>
<tr>
<td>057000</td>
<td>136</td>
<td>5E</td>
<td>94</td>
<td>‘</td>
<td>Circumflex; appears as up arrow (↑) on some terminals</td>
</tr>
<tr>
<td>057400</td>
<td>137</td>
<td>5F</td>
<td>95</td>
<td>_</td>
<td>Underscore; appears as left arrow (←) on some terminals</td>
</tr>
<tr>
<td>060000</td>
<td>140</td>
<td>60</td>
<td>96</td>
<td>,</td>
<td>Accent grave</td>
</tr>
<tr>
<td>060400</td>
<td>141</td>
<td>61</td>
<td>97</td>
<td>a</td>
<td>Lowercase a</td>
</tr>
<tr>
<td>061000</td>
<td>142</td>
<td>62</td>
<td>98</td>
<td>b</td>
<td>Lowercase b</td>
</tr>
<tr>
<td>061400</td>
<td>143</td>
<td>63</td>
<td>99</td>
<td>c</td>
<td>Lowercase c</td>
</tr>
<tr>
<td>062000</td>
<td>144</td>
<td>64</td>
<td>100</td>
<td>d</td>
<td>Lowercase d</td>
</tr>
<tr>
<td>062400</td>
<td>145</td>
<td>65</td>
<td>101</td>
<td>e</td>
<td>Lowercase e</td>
</tr>
<tr>
<td>063000</td>
<td>146</td>
<td>66</td>
<td>102</td>
<td>f</td>
<td>Lowercase f</td>
</tr>
<tr>
<td>063400</td>
<td>147</td>
<td>67</td>
<td>103</td>
<td>g</td>
<td>Lowercase g</td>
</tr>
<tr>
<td>064000</td>
<td>150</td>
<td>68</td>
<td>104</td>
<td>h</td>
<td>Lowercase h</td>
</tr>
</tbody>
</table>
Table A-1 (Cont.): DEC Multinational Character Set

<table>
<thead>
<tr>
<th>Left Byte Octal</th>
<th>Right Byte Octal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Character</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>064400</td>
<td>151</td>
<td>69</td>
<td>105</td>
<td>i</td>
<td>Lowercase i</td>
</tr>
<tr>
<td>065000</td>
<td>152</td>
<td>6A</td>
<td>106</td>
<td>j</td>
<td>Lowercase j</td>
</tr>
<tr>
<td>065400</td>
<td>153</td>
<td>6B</td>
<td>107</td>
<td>k</td>
<td>Lowercase k</td>
</tr>
<tr>
<td>066000</td>
<td>154</td>
<td>6C</td>
<td>108</td>
<td>l</td>
<td>Lowercase l</td>
</tr>
<tr>
<td>066400</td>
<td>155</td>
<td>6D</td>
<td>109</td>
<td>m</td>
<td>Lowercase m</td>
</tr>
<tr>
<td>067000</td>
<td>156</td>
<td>6E</td>
<td>110</td>
<td>n</td>
<td>Lowercase n</td>
</tr>
<tr>
<td>067400</td>
<td>157</td>
<td>6F</td>
<td>111</td>
<td>o</td>
<td>Lowercase o</td>
</tr>
<tr>
<td>070000</td>
<td>160</td>
<td>70</td>
<td>112</td>
<td>p</td>
<td>Lowercase p</td>
</tr>
<tr>
<td>070400</td>
<td>161</td>
<td>71</td>
<td>113</td>
<td>q</td>
<td>Lowercase q</td>
</tr>
<tr>
<td>071000</td>
<td>162</td>
<td>72</td>
<td>114</td>
<td>r</td>
<td>Lowercase r</td>
</tr>
<tr>
<td>071400</td>
<td>163</td>
<td>73</td>
<td>115</td>
<td>s</td>
<td>Lowercase s</td>
</tr>
<tr>
<td>072000</td>
<td>164</td>
<td>74</td>
<td>116</td>
<td>t</td>
<td>Lowercase t</td>
</tr>
<tr>
<td>072400</td>
<td>165</td>
<td>75</td>
<td>117</td>
<td>u</td>
<td>Lowercase u</td>
</tr>
<tr>
<td>073000</td>
<td>166</td>
<td>76</td>
<td>118</td>
<td>v</td>
<td>Lowercase v</td>
</tr>
<tr>
<td>073400</td>
<td>167</td>
<td>77</td>
<td>119</td>
<td>w</td>
<td>Lowercase w</td>
</tr>
<tr>
<td>074000</td>
<td>170</td>
<td>78</td>
<td>120</td>
<td>x</td>
<td>Lowercase x</td>
</tr>
<tr>
<td>074400</td>
<td>171</td>
<td>79</td>
<td>121</td>
<td>y</td>
<td>Lowercase y</td>
</tr>
<tr>
<td>075000</td>
<td>172</td>
<td>7A</td>
<td>122</td>
<td>z</td>
<td>Lowercase z</td>
</tr>
<tr>
<td>075400</td>
<td>173</td>
<td>7B</td>
<td>123</td>
<td>(</td>
<td>Left brace</td>
</tr>
<tr>
<td>076000</td>
<td>174</td>
<td>7C</td>
<td>124</td>
<td>)</td>
<td>Vertical bar</td>
</tr>
<tr>
<td>076400</td>
<td>175</td>
<td>7D</td>
<td>125</td>
<td></td>
<td>Right brace</td>
</tr>
<tr>
<td>077000</td>
<td>176</td>
<td>7E</td>
<td>126</td>
<td>~</td>
<td>Tilde</td>
</tr>
<tr>
<td>077400</td>
<td>177</td>
<td>7F</td>
<td>127</td>
<td>DEL</td>
<td>Delete, Rubout</td>
</tr>
<tr>
<td>100000</td>
<td>200</td>
<td>80</td>
<td>128</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>100400</td>
<td>201</td>
<td>81</td>
<td>129</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>101000</td>
<td>202</td>
<td>82</td>
<td>130</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>101400</td>
<td>203</td>
<td>83</td>
<td>131</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>102000</td>
<td>204</td>
<td>84</td>
<td>132</td>
<td>IND</td>
<td>Index</td>
</tr>
<tr>
<td>102400</td>
<td>205</td>
<td>85</td>
<td>133</td>
<td>NEL</td>
<td>Next line</td>
</tr>
<tr>
<td>103000</td>
<td>206</td>
<td>86</td>
<td>134</td>
<td>SSA</td>
<td>Start selected area</td>
</tr>
<tr>
<td>103400</td>
<td>207</td>
<td>87</td>
<td>135</td>
<td>ESA</td>
<td>End selected area</td>
</tr>
<tr>
<td>104000</td>
<td>210</td>
<td>88</td>
<td>136</td>
<td>HTS</td>
<td>Horizontal tab set</td>
</tr>
<tr>
<td>104400</td>
<td>211</td>
<td>89</td>
<td>137</td>
<td>HTJ</td>
<td>Horizontal tab justify</td>
</tr>
<tr>
<td>105000</td>
<td>212</td>
<td>8A</td>
<td>138</td>
<td>VTS</td>
<td>Vertical tab set</td>
</tr>
<tr>
<td>105400</td>
<td>213</td>
<td>8B</td>
<td>139</td>
<td>PLD</td>
<td>Partial line down</td>
</tr>
<tr>
<td>106000</td>
<td>214</td>
<td>8C</td>
<td>140</td>
<td>PLU</td>
<td>Partial line up</td>
</tr>
<tr>
<td>106400</td>
<td>215</td>
<td>8D</td>
<td>141</td>
<td>RI</td>
<td>Reverse index</td>
</tr>
<tr>
<td>107000</td>
<td>216</td>
<td>8E</td>
<td>142</td>
<td>SS2</td>
<td>Single shift G2</td>
</tr>
<tr>
<td>107400</td>
<td>217</td>
<td>8F</td>
<td>143</td>
<td>SS3</td>
<td>Single shift G3</td>
</tr>
<tr>
<td>110000</td>
<td>220</td>
<td>90</td>
<td>144</td>
<td>DCS</td>
<td>Device control string</td>
</tr>
<tr>
<td>110400</td>
<td>221</td>
<td>91</td>
<td>145</td>
<td>PU1</td>
<td>Private use 1</td>
</tr>
<tr>
<td>111000</td>
<td>222</td>
<td>92</td>
<td>146</td>
<td>PU2</td>
<td>Private use 2</td>
</tr>
<tr>
<td>111400</td>
<td>223</td>
<td>93</td>
<td>147</td>
<td>STS</td>
<td>Set transmit state</td>
</tr>
<tr>
<td>112000</td>
<td>224</td>
<td>94</td>
<td>148</td>
<td>CCH</td>
<td>Cancel character</td>
</tr>
<tr>
<td>112400</td>
<td>225</td>
<td>95</td>
<td>149</td>
<td>MW</td>
<td>Message waiting</td>
</tr>
<tr>
<td>Left Byte Octal</td>
<td>Right Byte Octal</td>
<td>Hex</td>
<td>Decimal</td>
<td>Character</td>
<td>Remarks</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------</td>
<td>-----</td>
<td>---------</td>
<td>-----------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>113000</td>
<td>226</td>
<td>96</td>
<td>150</td>
<td>SPA</td>
<td>Start protected area</td>
</tr>
<tr>
<td>113400</td>
<td>227</td>
<td>97</td>
<td>151</td>
<td>EPA</td>
<td>End protected area</td>
</tr>
<tr>
<td>114000</td>
<td>230</td>
<td>98</td>
<td>152</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>114400</td>
<td>231</td>
<td>99</td>
<td>153</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>115000</td>
<td>232</td>
<td>9A</td>
<td>154</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>115400</td>
<td>233</td>
<td>9B</td>
<td>155</td>
<td>CSI</td>
<td>Control sequence introduction</td>
</tr>
<tr>
<td>116000</td>
<td>234</td>
<td>9C</td>
<td>156</td>
<td>ST</td>
<td>String terminator</td>
</tr>
<tr>
<td>116400</td>
<td>235</td>
<td>9D</td>
<td>157</td>
<td>OSC</td>
<td>Operating system command</td>
</tr>
<tr>
<td>117000</td>
<td>236</td>
<td>9E</td>
<td>158</td>
<td>PM</td>
<td>Privacy message</td>
</tr>
<tr>
<td>117400</td>
<td>237</td>
<td>9F</td>
<td>159</td>
<td>APC</td>
<td>Application program command</td>
</tr>
<tr>
<td>120000</td>
<td>240</td>
<td>A0</td>
<td>160</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>120400</td>
<td>241</td>
<td>A1</td>
<td>161</td>
<td>¡</td>
<td>Inverted exclamation mark</td>
</tr>
<tr>
<td>121000</td>
<td>242</td>
<td>A2</td>
<td>162</td>
<td>¢</td>
<td>Cent sign</td>
</tr>
<tr>
<td>121400</td>
<td>243</td>
<td>A3</td>
<td>163</td>
<td>£</td>
<td>British pound</td>
</tr>
<tr>
<td>122000</td>
<td>244</td>
<td>A4</td>
<td>164</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>122400</td>
<td>245</td>
<td>A5</td>
<td>165</td>
<td>¥</td>
<td>Japanese yen</td>
</tr>
<tr>
<td>123000</td>
<td>246</td>
<td>A6</td>
<td>166</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>123400</td>
<td>247</td>
<td>A7</td>
<td>167</td>
<td>§</td>
<td>Section sign</td>
</tr>
<tr>
<td>124000</td>
<td>250</td>
<td>A8</td>
<td>168</td>
<td>Ø</td>
<td>General currency</td>
</tr>
<tr>
<td>124400</td>
<td>251</td>
<td>A9</td>
<td>169</td>
<td>©</td>
<td>Copyright</td>
</tr>
<tr>
<td>125000</td>
<td>252</td>
<td>AA</td>
<td>170</td>
<td>′</td>
<td>Feminine ordinal</td>
</tr>
<tr>
<td>125400</td>
<td>253</td>
<td>AB</td>
<td>171</td>
<td>«</td>
<td>Double open angle bracket</td>
</tr>
<tr>
<td>126000</td>
<td>254</td>
<td>AC</td>
<td>172</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>126400</td>
<td>255</td>
<td>AD</td>
<td>173</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>127000</td>
<td>256</td>
<td>AE</td>
<td>174</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>127400</td>
<td>257</td>
<td>AF</td>
<td>175</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>130000</td>
<td>260</td>
<td>B0</td>
<td>176</td>
<td>°</td>
<td>Degree</td>
</tr>
<tr>
<td>130400</td>
<td>261</td>
<td>B1</td>
<td>177</td>
<td>±</td>
<td>Plus or minus</td>
</tr>
<tr>
<td>131000</td>
<td>262</td>
<td>B2</td>
<td>178</td>
<td>²</td>
<td>Superscript 2</td>
</tr>
<tr>
<td>131400</td>
<td>263</td>
<td>B3</td>
<td>179</td>
<td>³</td>
<td>Superscript 3</td>
</tr>
<tr>
<td>132000</td>
<td>264</td>
<td>B4</td>
<td>180</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>132400</td>
<td>265</td>
<td>B5</td>
<td>181</td>
<td>µ</td>
<td>Micro</td>
</tr>
<tr>
<td>133000</td>
<td>266</td>
<td>B6</td>
<td>182</td>
<td>¶</td>
<td>Pilcrow</td>
</tr>
<tr>
<td>133400</td>
<td>267</td>
<td>B7</td>
<td>183</td>
<td>·</td>
<td>Middle dot</td>
</tr>
<tr>
<td>134000</td>
<td>270</td>
<td>B8</td>
<td>184</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>134400</td>
<td>271</td>
<td>B9</td>
<td>185</td>
<td>¹</td>
<td>Superscript 1</td>
</tr>
<tr>
<td>135000</td>
<td>272</td>
<td>BA</td>
<td>186</td>
<td>Ω</td>
<td>Masculine ordinal</td>
</tr>
<tr>
<td>135400</td>
<td>273</td>
<td>BB</td>
<td>187</td>
<td>»</td>
<td>Double close angle bracket</td>
</tr>
<tr>
<td>136000</td>
<td>274</td>
<td>BC</td>
<td>188</td>
<td>¼</td>
<td>One-fourth</td>
</tr>
<tr>
<td>136400</td>
<td>275</td>
<td>BD</td>
<td>189</td>
<td>½</td>
<td>One-half</td>
</tr>
<tr>
<td>137000</td>
<td>276</td>
<td>BE</td>
<td>190</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>137400</td>
<td>277</td>
<td>BF</td>
<td>191</td>
<td>′</td>
<td>Inverted question mark</td>
</tr>
<tr>
<td>140000</td>
<td>300</td>
<td>C0</td>
<td>192</td>
<td>Å</td>
<td>Uppercase A grave</td>
</tr>
<tr>
<td>140400</td>
<td>301</td>
<td>C1</td>
<td>193</td>
<td>Å</td>
<td>Uppercase A acute</td>
</tr>
<tr>
<td>141000</td>
<td>302</td>
<td>C2</td>
<td>194</td>
<td>Å</td>
<td>Uppercase A circumflex</td>
</tr>
</tbody>
</table>
Table A–1 (Cont.): DEC Multinational Character Set

<table>
<thead>
<tr>
<th>Left Byte Octal</th>
<th>Right Byte Octal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Character</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>141400</td>
<td>303</td>
<td>C3</td>
<td>195</td>
<td>Â</td>
<td>Uppercase A tilde</td>
</tr>
<tr>
<td>142000</td>
<td>304</td>
<td>C4</td>
<td>196</td>
<td>Ä</td>
<td>Uppercase A umlaut</td>
</tr>
<tr>
<td>142400</td>
<td>305</td>
<td>C5</td>
<td>197</td>
<td>À</td>
<td>Uppercase A ring</td>
</tr>
<tr>
<td>143000</td>
<td>306</td>
<td>C6</td>
<td>198</td>
<td>Æ</td>
<td>Uppercase AE diphthong</td>
</tr>
<tr>
<td>143400</td>
<td>307</td>
<td>C7</td>
<td>199</td>
<td>Ç</td>
<td>Uppercase C cedilla</td>
</tr>
<tr>
<td>144000</td>
<td>310</td>
<td>C8</td>
<td>200</td>
<td>É</td>
<td>Uppercase E grave</td>
</tr>
<tr>
<td>144400</td>
<td>311</td>
<td>C9</td>
<td>201</td>
<td>È</td>
<td>Uppercase E acute</td>
</tr>
<tr>
<td>145000</td>
<td>312</td>
<td>CA</td>
<td>202</td>
<td>Ê</td>
<td>Uppercase E circumflex</td>
</tr>
<tr>
<td>145400</td>
<td>313</td>
<td>CB</td>
<td>203</td>
<td>É</td>
<td>Uppercase E umlaut</td>
</tr>
<tr>
<td>146000</td>
<td>314</td>
<td>CC</td>
<td>204</td>
<td>I</td>
<td>Uppercase I grave</td>
</tr>
<tr>
<td>146400</td>
<td>315</td>
<td>CD</td>
<td>205</td>
<td>I</td>
<td>Uppercase I acute</td>
</tr>
<tr>
<td>147000</td>
<td>316</td>
<td>CE</td>
<td>206</td>
<td>I</td>
<td>Uppercase I circumflex</td>
</tr>
<tr>
<td>147400</td>
<td>317</td>
<td>CF</td>
<td>207</td>
<td>I</td>
<td>Uppercase I umlaut</td>
</tr>
<tr>
<td>150000</td>
<td>320</td>
<td>D0</td>
<td>208</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>150400</td>
<td>321</td>
<td>D1</td>
<td>209</td>
<td>Ñ</td>
<td>Uppercase N tilde</td>
</tr>
<tr>
<td>151000</td>
<td>322</td>
<td>D2</td>
<td>210</td>
<td>Ö</td>
<td>Uppercase O grave</td>
</tr>
<tr>
<td>151400</td>
<td>323</td>
<td>D3</td>
<td>211</td>
<td>Ø</td>
<td>Uppercase O acute</td>
</tr>
<tr>
<td>152000</td>
<td>324</td>
<td>D4</td>
<td>212</td>
<td>Ù</td>
<td>Uppercase O circumflex</td>
</tr>
<tr>
<td>152400</td>
<td>325</td>
<td>D5</td>
<td>213</td>
<td>Ø</td>
<td>Uppercase O tilde</td>
</tr>
<tr>
<td>153000</td>
<td>326</td>
<td>D6</td>
<td>214</td>
<td>Ù</td>
<td>Uppercase O umlaut</td>
</tr>
<tr>
<td>153400</td>
<td>327</td>
<td>D7</td>
<td>215</td>
<td>OE</td>
<td>Uppercase OE ligature</td>
</tr>
<tr>
<td>154000</td>
<td>330</td>
<td>D8</td>
<td>216</td>
<td>Ø</td>
<td>Uppercase O slash</td>
</tr>
<tr>
<td>154400</td>
<td>331</td>
<td>D9</td>
<td>217</td>
<td>Ù</td>
<td>Uppercase U grave</td>
</tr>
<tr>
<td>155000</td>
<td>332</td>
<td>DA</td>
<td>218</td>
<td>Ü</td>
<td>Uppercase U acute</td>
</tr>
<tr>
<td>155400</td>
<td>333</td>
<td>DB</td>
<td>219</td>
<td>Ù</td>
<td>Uppercase U circumflex</td>
</tr>
<tr>
<td>156000</td>
<td>334</td>
<td>DC</td>
<td>220</td>
<td>Ü</td>
<td>Uppercase U umlaut</td>
</tr>
<tr>
<td>156400</td>
<td>335</td>
<td>DD</td>
<td>221</td>
<td>Ý</td>
<td>Uppercase Y umlaut</td>
</tr>
<tr>
<td>157000</td>
<td>336</td>
<td>DE</td>
<td>222</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>157400</td>
<td>337</td>
<td>DF</td>
<td>223</td>
<td>b</td>
<td>German small sharp s</td>
</tr>
<tr>
<td>160000</td>
<td>340</td>
<td>E0</td>
<td>224</td>
<td>å</td>
<td>Lowercase a grave</td>
</tr>
<tr>
<td>160400</td>
<td>341</td>
<td>E1</td>
<td>225</td>
<td>á</td>
<td>Lowercase a acute</td>
</tr>
<tr>
<td>161000</td>
<td>342</td>
<td>E2</td>
<td>226</td>
<td>å</td>
<td>Lowercase a circumflex</td>
</tr>
<tr>
<td>161400</td>
<td>343</td>
<td>E3</td>
<td>227</td>
<td>å</td>
<td>Lowercase a tilde</td>
</tr>
<tr>
<td>162000</td>
<td>344</td>
<td>E4</td>
<td>228</td>
<td>å</td>
<td>Lowercase a umlaut</td>
</tr>
<tr>
<td>162400</td>
<td>345</td>
<td>E5</td>
<td>229</td>
<td>å</td>
<td>Lowercase a ring</td>
</tr>
<tr>
<td>163000</td>
<td>346</td>
<td>E6</td>
<td>230</td>
<td>æ</td>
<td>Lowercase ae diphthong</td>
</tr>
<tr>
<td>163400</td>
<td>347</td>
<td>E7</td>
<td>231</td>
<td>ç</td>
<td>Lowercase c cedilla</td>
</tr>
<tr>
<td>164000</td>
<td>350</td>
<td>E8</td>
<td>232</td>
<td>è</td>
<td>Lowercase e grave</td>
</tr>
<tr>
<td>164400</td>
<td>351</td>
<td>E9</td>
<td>233</td>
<td>é</td>
<td>Lowercase e acute</td>
</tr>
<tr>
<td>165000</td>
<td>352</td>
<td>EA</td>
<td>234</td>
<td>ê</td>
<td>Lowercase e circumflex</td>
</tr>
<tr>
<td>165400</td>
<td>353</td>
<td>EB</td>
<td>235</td>
<td>ê</td>
<td>Lowercase e umlaut</td>
</tr>
<tr>
<td>166000</td>
<td>354</td>
<td>EC</td>
<td>236</td>
<td>i</td>
<td>Lowercase i grave</td>
</tr>
<tr>
<td>166400</td>
<td>355</td>
<td>ED</td>
<td>237</td>
<td>i</td>
<td>Lowercase i acute</td>
</tr>
<tr>
<td>167000</td>
<td>356</td>
<td>EE</td>
<td>238</td>
<td>i</td>
<td>Lowercase i circumflex</td>
</tr>
<tr>
<td>167400</td>
<td>357</td>
<td>EF</td>
<td>239</td>
<td>i</td>
<td>Lowercase i umlaut</td>
</tr>
<tr>
<td>Left</td>
<td>Right</td>
<td>Hex</td>
<td>Decimal</td>
<td>Character</td>
<td>Remarks</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td>---------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Octal</td>
<td>Octal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170000</td>
<td>360</td>
<td>F0</td>
<td>240</td>
<td>Ń</td>
<td>Reserved</td>
</tr>
<tr>
<td>170400</td>
<td>361</td>
<td>F1</td>
<td>241</td>
<td>ñ</td>
<td>Lowercase n tilde</td>
</tr>
<tr>
<td>171000</td>
<td>362</td>
<td>F2</td>
<td>242</td>
<td>ò</td>
<td>Lowercase o grave</td>
</tr>
<tr>
<td>171400</td>
<td>363</td>
<td>F3</td>
<td>243</td>
<td>ò</td>
<td>Lowercase o acute</td>
</tr>
<tr>
<td>172000</td>
<td>364</td>
<td>F4</td>
<td>244</td>
<td>ò</td>
<td>Lowercase o circumflex</td>
</tr>
<tr>
<td>172400</td>
<td>365</td>
<td>F5</td>
<td>245</td>
<td>ò</td>
<td>Lowercase o tilde</td>
</tr>
<tr>
<td>173000</td>
<td>366</td>
<td>F6</td>
<td>246</td>
<td>ò</td>
<td>Lowercase o umlaut</td>
</tr>
<tr>
<td>173400</td>
<td>367</td>
<td>F7</td>
<td>247</td>
<td>Ï</td>
<td>Lowercase oe ligature</td>
</tr>
<tr>
<td>174000</td>
<td>370</td>
<td>F8</td>
<td>248</td>
<td>ø</td>
<td>Lowercase o slash</td>
</tr>
<tr>
<td>174400</td>
<td>371</td>
<td>F9</td>
<td>249</td>
<td>ü</td>
<td>Lowercase u grave</td>
</tr>
<tr>
<td>175000</td>
<td>372</td>
<td>FA</td>
<td>250</td>
<td>û</td>
<td>Lowercase u acute</td>
</tr>
<tr>
<td>175400</td>
<td>373</td>
<td>FB</td>
<td>251</td>
<td>û</td>
<td>Lowercase u circumflex</td>
</tr>
<tr>
<td>176000</td>
<td>374</td>
<td>FC</td>
<td>252</td>
<td>Ü</td>
<td>Lowercase u umlaut</td>
</tr>
<tr>
<td>176400</td>
<td>375</td>
<td>FD</td>
<td>253</td>
<td>Y</td>
<td>Lowercase y umlaut</td>
</tr>
<tr>
<td>177000</td>
<td>376</td>
<td>FE</td>
<td>254</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>177400</td>
<td>377</td>
<td>FF</td>
<td>255</td>
<td></td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### A.2 Radix−50 Character Set

<table>
<thead>
<tr>
<th>Character</th>
<th>Octal Equivalent</th>
<th>Radix−50 Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>040</td>
<td>000</td>
</tr>
<tr>
<td>A−Z</td>
<td>101−132</td>
<td>001−032</td>
</tr>
<tr>
<td>$</td>
<td>044</td>
<td>033</td>
</tr>
<tr>
<td>.</td>
<td>056</td>
<td>034</td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td>035</td>
</tr>
<tr>
<td>0−9</td>
<td>060−071</td>
<td>036−047</td>
</tr>
</tbody>
</table>

The maximum Radix−50 octal value is therefore:

\[47 \times 50^2 + 47 \times 50 + 47 = 174777\]

Table A−3 provides a convenient means of translating between the ASCII character set and its Radix−50 equivalents. For example, given the ASCII string `X2B`, the Radix−50 equivalent is (arithmetic is performed in octal):

\[
x=113000 \\
2=002400 \\
B=000002 \\
X2B=115402
\]

### Table A−3: Radix−50 Character Equivalents

<table>
<thead>
<tr>
<th>Single Character or First Character</th>
<th>Second Character</th>
<th>Third Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>Space</td>
<td>Space</td>
</tr>
<tr>
<td>A</td>
<td>003100</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>006200</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>011300</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>014400</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>017500</td>
<td>E</td>
</tr>
<tr>
<td>F</td>
<td>022600</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>025700</td>
<td>G</td>
</tr>
<tr>
<td>H</td>
<td>031000</td>
<td>H</td>
</tr>
<tr>
<td>I</td>
<td>034100</td>
<td>I</td>
</tr>
<tr>
<td>J</td>
<td>037200</td>
<td>J</td>
</tr>
<tr>
<td>K</td>
<td>042300</td>
<td>K</td>
</tr>
<tr>
<td>L</td>
<td>045400</td>
<td>L</td>
</tr>
<tr>
<td>M</td>
<td>050500</td>
<td>M</td>
</tr>
<tr>
<td>N</td>
<td>053600</td>
<td>N</td>
</tr>
<tr>
<td>O</td>
<td>056700</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>062000</td>
<td>P</td>
</tr>
</tbody>
</table>

|                  |                  | 00000           |
|                  |                  | 000001          |
|                  |                  | 000002          |
|                  |                  | 000003          |
|                  |                  | 000004          |
|                  |                  | 000005          |
|                  |                  | 000006          |
|                  |                  | 000007          |
|                  |                  | 000010          |
|                  |                  | 000011          |
|                  |                  | 000012          |
|                  |                  | 000013          |
|                  |                  | 000014          |
|                  |                  | 000015          |
|                  |                  | 000016          |
|                  |                  | 000017          |
|                  |                  | 000020          |
Table A–3 (Cont.): Radix–50 Character Equivalents

<table>
<thead>
<tr>
<th>Single Character or First Character</th>
<th>Second Character</th>
<th>Third Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q 065100</td>
<td>Q 001250</td>
<td>Q 000021</td>
</tr>
<tr>
<td>R 070200</td>
<td>R 001320</td>
<td>R 000022</td>
</tr>
<tr>
<td>S 073300</td>
<td>S 001370</td>
<td>S 000023</td>
</tr>
<tr>
<td>T 076400</td>
<td>T 001440</td>
<td>T 000024</td>
</tr>
<tr>
<td>U 101500</td>
<td>U 001510</td>
<td>U 000025</td>
</tr>
<tr>
<td>V 104600</td>
<td>V 001560</td>
<td>V 000026</td>
</tr>
<tr>
<td>W 107700</td>
<td>W 001630</td>
<td>W 000027</td>
</tr>
<tr>
<td>X 113000</td>
<td>X 001700</td>
<td>X 000030</td>
</tr>
<tr>
<td>Y 116100</td>
<td>Y 001750</td>
<td>Y 000031</td>
</tr>
<tr>
<td>Z 121200</td>
<td>Z 002020</td>
<td>Z 000032</td>
</tr>
<tr>
<td>$ 124300</td>
<td>$ 002070</td>
<td>$ 000033</td>
</tr>
<tr>
<td>. 127400</td>
<td>. 002140</td>
<td>. 000034</td>
</tr>
<tr>
<td>Unused 132500</td>
<td>Unused 002210</td>
<td>Unused 000035</td>
</tr>
<tr>
<td>0 135600</td>
<td>0 002260</td>
<td>0 000036</td>
</tr>
<tr>
<td>1 140700</td>
<td>1 002330</td>
<td>1 000037</td>
</tr>
<tr>
<td>2 144000</td>
<td>2 002400</td>
<td>2 000040</td>
</tr>
<tr>
<td>3 147100</td>
<td>3 002450</td>
<td>3 000041</td>
</tr>
<tr>
<td>4 152200</td>
<td>4 002520</td>
<td>4 000042</td>
</tr>
<tr>
<td>5 155300</td>
<td>5 002570</td>
<td>5 000043</td>
</tr>
<tr>
<td>6 160400</td>
<td>6 002640</td>
<td>6 000044</td>
</tr>
<tr>
<td>7 163500</td>
<td>7 002710</td>
<td>7 000045</td>
</tr>
<tr>
<td>8 166600</td>
<td>8 002760</td>
<td>8 000046</td>
</tr>
<tr>
<td>9 171700</td>
<td>9 003030</td>
<td>9 000047</td>
</tr>
</tbody>
</table>

A.3 DEC Multinational Character Set

Figure A–1 contains the DEC multinational character set; empty positions are reserved for future standardizations.
Figure A-1: DEC Multinational Character Set

<table>
<thead>
<tr>
<th>ASCII CHARACTER SET</th>
<th>ESC MULTINATIONAL CHARACTER SET</th>
<th>DEC SUPPLEMENTAL GRAPHIC SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>1</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>2</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>3</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>4</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>5</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>6</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>7</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>8</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>9</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>A</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>B</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>C</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>D</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>E</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>F</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>G</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>H</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>I</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>J</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>K</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>L</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>M</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>N</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>O</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>P</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>Q</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>R</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>S</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>T</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>U</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>V</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>W</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>X</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>Y</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>Z</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>0</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>1</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>2</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>3</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>4</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>5</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>6</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>7</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>8</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>9</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>A</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>B</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>C</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>D</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>E</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>F</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>G</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>H</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>I</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>J</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>K</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>L</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>M</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>N</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>O</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>P</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>Q</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>R</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>S</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>T</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>U</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>V</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>W</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>X</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>Y</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
<tr>
<td>Z</td>
<td>11:04:09</td>
<td>11:04:09</td>
</tr>
</tbody>
</table>

KEY:
- ASCII CHARACTER SET
- ESC MULTINATIONAL CHARACTER SET
- DEC SUPPLEMENTAL GRAPHIC SET
Appendix B
MACRO–11 Assembly Language and Assembler Directives

B.1 Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>Label terminator</td>
</tr>
<tr>
<td>=</td>
<td>Direct assignment operator</td>
</tr>
<tr>
<td>%</td>
<td>Register term indicator</td>
</tr>
<tr>
<td>TAB</td>
<td>Item terminator or field terminator</td>
</tr>
<tr>
<td>SP</td>
<td>Item terminator or field terminator</td>
</tr>
<tr>
<td>#</td>
<td>Immediate expression indicator</td>
</tr>
<tr>
<td>@</td>
<td>Deferred addressing indicator</td>
</tr>
<tr>
<td>(</td>
<td>Initial register indicator</td>
</tr>
<tr>
<td>)</td>
<td>Terminal register indicator</td>
</tr>
<tr>
<td>, (comma)</td>
<td>Operand field separator</td>
</tr>
<tr>
<td>;</td>
<td>Comment field indicator</td>
</tr>
<tr>
<td>+</td>
<td>Arithmetic addition operator or autoincrement indicator</td>
</tr>
<tr>
<td>-</td>
<td>Arithmetic subtraction operator or autodecrement indicator</td>
</tr>
<tr>
<td>*</td>
<td>Arithmetic multiplication operator</td>
</tr>
<tr>
<td>/</td>
<td>Arithmetic division operator</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical AND operator</td>
</tr>
<tr>
<td>!</td>
<td>Logical OR operator</td>
</tr>
<tr>
<td>'</td>
<td>Double ASCII character indicator</td>
</tr>
<tr>
<td>' (single quote)</td>
<td>Single ASCII character indicator or concatenation indicator</td>
</tr>
<tr>
<td>.</td>
<td>Assembly location counter</td>
</tr>
<tr>
<td>&lt;</td>
<td>Initial argument indicator</td>
</tr>
<tr>
<td>&gt;</td>
<td>Terminal argument indicator</td>
</tr>
<tr>
<td>–</td>
<td>Universal unary operator or argument indicator</td>
</tr>
<tr>
<td>\</td>
<td>Macro call numeric argument indicator</td>
</tr>
<tr>
<td>vertical tab</td>
<td>Source line terminator</td>
</tr>
</tbody>
</table>
### B.2 Summary of Address Mode Syntax

<table>
<thead>
<tr>
<th>Format</th>
<th>Name</th>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Register</td>
<td>0n</td>
<td>Register R contains the operand.</td>
</tr>
<tr>
<td>@R or (ER)</td>
<td>Register Deferred</td>
<td>1n</td>
<td>Register R contains the address of the operand.</td>
</tr>
<tr>
<td>(ER) +</td>
<td>Autoincrement</td>
<td>2n</td>
<td>The contents of the register specified as (ER) are incremented after being used as the address of the operand.</td>
</tr>
<tr>
<td>@ER +</td>
<td>Autoincrement Deferred</td>
<td>3n</td>
<td>The register specified as (ER) contains the pointer to the address of the operand; the register (ER) is incremented after use.</td>
</tr>
<tr>
<td>-(ER)</td>
<td>Autodecrement</td>
<td>4n</td>
<td>The contents of the register specified as (ER) are decremented before being used as the address of the operand.</td>
</tr>
<tr>
<td>@-(ER)</td>
<td>Autodecrement Deferred</td>
<td>5n</td>
<td>The contents of the register specified as (ER) are decremented before being used as the pointer to the address of the operand.</td>
</tr>
<tr>
<td>E(ER)</td>
<td>Index</td>
<td>6n</td>
<td>The expression E, plus the contents of the register specified as (ER), form the address of the operand.</td>
</tr>
<tr>
<td>@E(ER)</td>
<td>Index Deferred</td>
<td>7n</td>
<td>The expression E, plus the contents of the register specified as (ER), yield a pointer to the address of the operand.</td>
</tr>
<tr>
<td>E</td>
<td>Immediate</td>
<td>27</td>
<td>The expression E is the operand itself.</td>
</tr>
<tr>
<td>@E</td>
<td>Absolute</td>
<td>37</td>
<td>The expression E is the address of the operand.</td>
</tr>
<tr>
<td>E</td>
<td>Relative</td>
<td>67</td>
<td>The address of the operand E, relative to the instruction, follows the instruction.</td>
</tr>
<tr>
<td>@E</td>
<td>Relative Deferred</td>
<td>77</td>
<td>The address of the operand is pointed to by E, whose address, relative to the instruction, follows the instruction.</td>
</tr>
</tbody>
</table>

1Symbols used in the table:
- \( n \) is an integer, 0 to 7, representing a register number.
- \( R \) is a register expression.
- \( E \) is an expression.
- \( ER \) is a register expression or an expression whose value is in the range 0 to 7.
B.3 Assembler Directives

The MACRO-11 assembler directives are summarized in the following table. For a detailed description of each directive, refer to the appropriate sections in the body of the manual.

<table>
<thead>
<tr>
<th>Form</th>
<th>Reference</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>6.3.3</td>
<td>Followed by one ASCII character, a single quote (apostrophe) generates a word which contains the 7-bit ASCII representation of the character in the low-order byte and zero in the high-order byte. Single quote is also used as a concatenation indicator in the expansion of macro arguments.</td>
</tr>
<tr>
<td>&quot;</td>
<td>6.3.3</td>
<td>Followed by two ASCII characters, a double quote generates a word which contains the 7-bit ASCII representation of the two characters. The first character is stored in the low-order byte; the second character is stored in the high-order byte.</td>
</tr>
<tr>
<td>&quot;Bn</td>
<td>6.4.1.2</td>
<td>A temporary radix control, causes the value n to be treated as a binary number.</td>
</tr>
<tr>
<td>&quot;Cexpr</td>
<td>6.4.2.1</td>
<td>A temporary numeric control, causes the expression's value to be one's complemented.</td>
</tr>
<tr>
<td>&quot;Dn</td>
<td>6.4.1.2</td>
<td>A temporary radix control, causes the value n to be treated as a decimal number.</td>
</tr>
<tr>
<td>&quot;Fn</td>
<td>6.4.2.3</td>
<td>A temporary numeric control, causes the value n to be treated as a 16-bit floating-point number.</td>
</tr>
<tr>
<td>&quot;On</td>
<td>6.4.1.2</td>
<td>A temporary radix control, causes the value n to be treated as an octal number.</td>
</tr>
<tr>
<td>&quot;Rccc</td>
<td>6.3.7</td>
<td>Converts ccc to Radix-50 form.</td>
</tr>
<tr>
<td>&quot;Xn</td>
<td>6.4.1.2</td>
<td>A temporary radix control, causes the value n to be treated as a hexadecimal number. The value n must begin with a digit, which may be 0.</td>
</tr>
<tr>
<td>.ASCII /string/</td>
<td>6.3.4</td>
<td>Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters), one character per byte.</td>
</tr>
<tr>
<td>.ASCIZ /string/</td>
<td>6.3.5</td>
<td>Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters), one character per byte, with a zero byte terminating the specified string.</td>
</tr>
<tr>
<td>.ASECTION</td>
<td>6.7.2</td>
<td>Begins or resumes the absolute program section.</td>
</tr>
<tr>
<td>.BLKB exp</td>
<td>6.5.3</td>
<td>Reserves a block of storage space whose length in bytes is determined by the specified expression.</td>
</tr>
<tr>
<td>Form</td>
<td>Reference</td>
<td>Operation</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>.BLKW exp</td>
<td>6.5.3</td>
<td>Reserves a block of storage space whose length in words is determined by the specified expression.</td>
</tr>
<tr>
<td>.BYTE exp1,exp2,...</td>
<td>6.3.1</td>
<td>Generates successive bytes of data; each byte contains the value of the corresponding specified expression.</td>
</tr>
<tr>
<td>.CROSS sym1,sym2,...</td>
<td>6.2.2</td>
<td>Enables the cross-reference listing for the specified symbol list. If a symbol list is not specified, this directive reenables the cross-reference listing for all symbols in the program.</td>
</tr>
<tr>
<td>.CSECT [name]</td>
<td>6.7.2</td>
<td>Begins or resumes named or unnamed relocatable program section. This directive is provided for compatibility with other PDP-11 assemblers.</td>
</tr>
<tr>
<td>.DSABL arg</td>
<td>6.2.1</td>
<td>Disables the function specified by the argument.</td>
</tr>
<tr>
<td>.ENABL arg</td>
<td>6.2.1</td>
<td>Enables the function specified by the argument.</td>
</tr>
<tr>
<td>.END [exp]</td>
<td>6.6</td>
<td>Indicates the logical end of the source program. The optional argument specifies the transfer address where program execution is to begin.</td>
</tr>
<tr>
<td>.ENDC</td>
<td>6.9.1</td>
<td>Indicates the end of a conditional assembly block.</td>
</tr>
<tr>
<td>.ENDM [name]</td>
<td>7.1.2</td>
<td>Indicates the end of the current repeat block, indefinite repeat block, or macro definition. The optional name, if used, must be identical to the name specified in the macro definition.</td>
</tr>
<tr>
<td>.ENDR</td>
<td>7.7</td>
<td>Indicates the end of the current repeat block. This directive is provided for compatibility with other PDP-11 assemblers.</td>
</tr>
<tr>
<td>.ERROR exp;text</td>
<td>7.5</td>
<td>A user-called error directive, causes output to the listing file or the command output device containing the optional expression and the statement containing the directive.</td>
</tr>
<tr>
<td>.EVEN</td>
<td>6.5.1</td>
<td>Ensures that the current location counter contains an even address by adding 1 if it is odd.</td>
</tr>
<tr>
<td>.FLT2 arg1, arg2,...</td>
<td>6.4.2.2</td>
<td>Generates successive 2-word floating-point equivalents for the floating-point numbers specified as arguments.</td>
</tr>
<tr>
<td>.FLT4 arg1, arg2,...</td>
<td>6.4.2.2</td>
<td>Generates successive 4-word floating-point equivalents for the floating-point numbers specified as arguments.</td>
</tr>
<tr>
<td>.GLOBL sym1,sym2,...</td>
<td>6.8.1</td>
<td>Defines the listed symbol(s) as global symbol(s).</td>
</tr>
<tr>
<td>Form</td>
<td>Reference</td>
<td>Operation</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>.IDENT /string/</td>
<td>6.1.4</td>
<td>Provides a means of labeling the object module with the program version number. The version number is the Radix-50 string appearing between the paired delimiting characters.</td>
</tr>
<tr>
<td>.IF cond, arg1, arg2,...</td>
<td>6.9.1</td>
<td>Begins a conditional assembly block of source code, which is included in the assembly only if the stated condition is met with respect to the specified argument(s).</td>
</tr>
<tr>
<td>.IFF</td>
<td>6.9.2</td>
<td>Appears only within a conditional assembly block, indicating the beginning of a section of code to be assembled if the condition upon entering the block tests false.</td>
</tr>
<tr>
<td>.IFT</td>
<td>6.9.2</td>
<td>Appears only within a conditional assembly block, indicating the beginning of a section of code to be assembled if the condition upon entering the block tests true.</td>
</tr>
<tr>
<td>.IFTF</td>
<td>6.9.2</td>
<td>Appears only within a conditional assembly block, indicating the beginning of a section of code to be assembled unconditionally.</td>
</tr>
<tr>
<td>.IF cond, arg, statement</td>
<td>6.9.3</td>
<td>Acts as a 1-line conditional assembly block where the condition is tested for the specified argument. The statement is assembled only if the condition tests true.</td>
</tr>
<tr>
<td>.INCLUDE filespec</td>
<td>6.10.2</td>
<td>Inserts a specified source file within the source file currently being used.</td>
</tr>
<tr>
<td>.IRP sym, &lt;arg1, arg2,...&gt;</td>
<td>7.6.1</td>
<td>Indicates the beginning of an indefinite repeat block in which the specified symbol is replaced with successive elements of the real argument list enclosed within angle brackets.</td>
</tr>
<tr>
<td>.IRPC sym, &lt;string&gt;</td>
<td>7.6.2</td>
<td>Indicates the beginning of an indefinite repeat block in which the specified symbol takes on the value of successive characters, optionally enclosed within angle brackets.</td>
</tr>
<tr>
<td>.LIBRARY filespec</td>
<td>6.10.1</td>
<td>Adds a specified file name to a macro library list that is searched.</td>
</tr>
<tr>
<td>.LIMIT</td>
<td>6.5.4</td>
<td>Reserves two words into which the Linker or Task Builder inserts the low and high addresses of the task image.</td>
</tr>
<tr>
<td>Form</td>
<td>Reference</td>
<td>Operation</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>.LIST [arg]</td>
<td>6.1.1</td>
<td>Without an argument, the .LIST directive increments the listing level count by 1. With an argument, this directive does not alter the listing level count, but formats the assembly listing according to the specified argument.</td>
</tr>
<tr>
<td>.MACRO name, arg1, arg2, ...</td>
<td>7.1.1</td>
<td>Indicates the start of a macro definition having the specified name and the following dummy arguments.</td>
</tr>
<tr>
<td>.MCALL arg1, arg2, ...</td>
<td>7.8</td>
<td>Specifies the symbolic names of the user or system macro definitions required in the assembly of the current user program, but which are not defined within the program.</td>
</tr>
<tr>
<td>.MDELETE name1, name2, ...</td>
<td>7.9</td>
<td>Deletes the definitions of the specified macro(s), freeing virtual memory.</td>
</tr>
<tr>
<td>.MEXIT</td>
<td>7.1.3</td>
<td>Causes an exit from the current macro expansion or indefinite repeat block.</td>
</tr>
<tr>
<td>.NARG symbol</td>
<td>7.4.1</td>
<td>Appearing only within a macro definition, equates the specified symbol to the number of arguments in the macro call currently being expanded.</td>
</tr>
<tr>
<td>.NCHR symbol,&lt;string&gt;</td>
<td>7.4.2</td>
<td>Appearing anywhere in a source program, equates the specified symbol to the number of characters in the specified string.</td>
</tr>
<tr>
<td>.NLIST [arg]</td>
<td>6.1.1</td>
<td>Without an argument, decrements the listing level count by 1. With an argument, this directive suppresses that portion of the listing specified by the argument.</td>
</tr>
<tr>
<td>.NOCROSS sym1, sym2, ...</td>
<td>6.2.2</td>
<td>Disables the cross-reference listing for the listed symbols. If a symbol list is not specified, this directive disables the cross-reference listing for all symbols in the program.</td>
</tr>
<tr>
<td>.NTYPE symbol,aexp</td>
<td>7.4.3</td>
<td>Appearing only within a macro definition, equates the symbol to the 6-bit addressing mode of the specified address expression.</td>
</tr>
<tr>
<td>.ODD</td>
<td>6.5.2</td>
<td>Ensures that the current location counter contains an odd address by adding 1 if it is even.</td>
</tr>
<tr>
<td>.PACKED</td>
<td>6.3.8</td>
<td>Causes a decimal number of $31_{10}$ digits or less to be packed two digits per byte.</td>
</tr>
<tr>
<td>.PAGE</td>
<td>6.1.5</td>
<td>Causes the assembly listing to skip to the top of the next page and to increment the page count.</td>
</tr>
<tr>
<td>Form</td>
<td>Reference</td>
<td>Operation</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>.PRINT exp;text</td>
<td>7.5</td>
<td>User-called message directive; causes output to the listing file or the command output device containing the optional expression and the statement containing the directive.</td>
</tr>
<tr>
<td>.PSEC name,attr,...</td>
<td>6.7.1</td>
<td>Begins or resumes a named or att attr unnamed program section having the specified attributes.</td>
</tr>
<tr>
<td>.RAD50 /string/</td>
<td>6.3.6</td>
<td>Generates a block of data containing the Radix-50 equivalent of the character string enclosed within delimiting characters.</td>
</tr>
<tr>
<td>.RADIX n</td>
<td>6.4.1.1</td>
<td>Alters the current program radix to n, where n is 2, 8, 10, or 16.</td>
</tr>
<tr>
<td>.REM comment-character</td>
<td>6.1.6</td>
<td>Allows a programmer to insert a block of comments into a MACRO-11 source program without having to precede the comment lines with the comment character (;).</td>
</tr>
<tr>
<td>.REPT exp</td>
<td>7.7</td>
<td>Begins a repeat block; causes the section of code up to the next .ENDM or .ENDR directive to be repeated the number of times specified as exp.</td>
</tr>
<tr>
<td>.RESTORE</td>
<td>6.7.4</td>
<td>Retrieves a previously .SAVEed program section context from the top of the program section context stack leaving the current program section in effect.</td>
</tr>
<tr>
<td>.SAVE</td>
<td>6.7.3</td>
<td>Stores the current program section context on the top of the program section context stack leaving the current program section in effect.</td>
</tr>
<tr>
<td>.SBTTL string</td>
<td>6.1.3</td>
<td>Causes the specified string to be printed as part of the assembly listing page header. The string component of each .SBTTL directive is collected into a table of contents at the beginning of the assembly listing.</td>
</tr>
<tr>
<td>.TITLE string</td>
<td>6.1.2</td>
<td>Assigns the first six Radix-50 characters in the string as an object module name and causes the string to appear on each page of the assembly listing.</td>
</tr>
<tr>
<td>.WEAK sym1,sym2,...</td>
<td>6.8.2</td>
<td>Specifies symbols that are defined either externally in another module or globally in the current module.</td>
</tr>
<tr>
<td>.WORD exp1,exp2,...</td>
<td>6.3.2</td>
<td>Generates successive words of data; each word contains the value of the corresponding specified expression.</td>
</tr>
</tbody>
</table>
Appendix C
Permanent Symbol Table

The mnemonics for the PDP-11 operation (op) codes and MACRO-11 assembler directives are stored in the Permanent Symbol Table. The Permanent Symbol Table contains the symbols that are automatically recognized by MACRO-11.

For a detailed description of the op codes, see the PDP-11 Processor Handbook.

C.1 Op Codes

<table>
<thead>
<tr>
<th>Instruction Mnemonic</th>
<th>Octal Value</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>005500</td>
<td>Add Carry</td>
</tr>
<tr>
<td>ADCB</td>
<td>105500</td>
<td>Add Carry (Byte)</td>
</tr>
<tr>
<td>ADD</td>
<td>060000</td>
<td>Add Source To Destination</td>
</tr>
<tr>
<td>ASH</td>
<td>072000</td>
<td>Shift Arithmetically</td>
</tr>
<tr>
<td>ASHC</td>
<td>073000</td>
<td>Arithmetic Shift Combined</td>
</tr>
<tr>
<td>ASL</td>
<td>006300</td>
<td>Arithmetic Shift Left</td>
</tr>
<tr>
<td>ASLB</td>
<td>106300</td>
<td>Arithmetic Shift Left (Byte)</td>
</tr>
<tr>
<td>ASR</td>
<td>006200</td>
<td>Arithmetic Shift Right</td>
</tr>
<tr>
<td>ASRB</td>
<td>106200</td>
<td>Arithmetic Shift Right (Byte)</td>
</tr>
<tr>
<td>BCC</td>
<td>103000</td>
<td>Branch If Carry Is Clear</td>
</tr>
<tr>
<td>BCS</td>
<td>103400</td>
<td>Branch If Carry Is Set</td>
</tr>
<tr>
<td>BEQ</td>
<td>001400</td>
<td>Branch If Equal</td>
</tr>
<tr>
<td>BGE</td>
<td>002000</td>
<td>Branch If Greater Than Or Equal</td>
</tr>
<tr>
<td>BGT</td>
<td>003000</td>
<td>Branch If Greater Than</td>
</tr>
<tr>
<td>BHI</td>
<td>101000</td>
<td>Branch If Higher</td>
</tr>
<tr>
<td>BHIS</td>
<td>103000</td>
<td>Branch If Higher Or Same</td>
</tr>
<tr>
<td>BIC</td>
<td>040000</td>
<td>Bit Clear</td>
</tr>
<tr>
<td>BICB</td>
<td>140000</td>
<td>Bit Clear (Byte)</td>
</tr>
<tr>
<td>BIS</td>
<td>050000</td>
<td>Bit Set</td>
</tr>
<tr>
<td>BISB</td>
<td>150000</td>
<td>Bit Set (Byte)</td>
</tr>
<tr>
<td>Instruction Mnemonic</td>
<td>Octal Value</td>
<td>Operation</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>BIT</td>
<td>030000</td>
<td>Bit Test</td>
</tr>
<tr>
<td>BITB</td>
<td>130000</td>
<td>Bit Test (Byte)</td>
</tr>
<tr>
<td>BLE</td>
<td>003400</td>
<td>Branch If Less Than Or Equal</td>
</tr>
<tr>
<td>BLO</td>
<td>103400</td>
<td>Branch If Lower</td>
</tr>
<tr>
<td>BLOS</td>
<td>101400</td>
<td>Branch If Lower Or Same</td>
</tr>
<tr>
<td>BLT</td>
<td>002400</td>
<td>Branch If Less Than</td>
</tr>
<tr>
<td>BMI</td>
<td>100400</td>
<td>Branch If Minus</td>
</tr>
<tr>
<td>BNE</td>
<td>001000</td>
<td>Branch If Not Equal</td>
</tr>
<tr>
<td>BPL</td>
<td>100000</td>
<td>Branch If Plus</td>
</tr>
<tr>
<td>BPT</td>
<td>000003</td>
<td>Breakpoint Trap</td>
</tr>
<tr>
<td>BR</td>
<td>000400</td>
<td>Branch Unconditional</td>
</tr>
<tr>
<td>BVC</td>
<td>102000</td>
<td>Branch If Overflow Is Clear</td>
</tr>
<tr>
<td>BVS</td>
<td>102400</td>
<td>Branch If Overflow Is Set</td>
</tr>
<tr>
<td>CALL</td>
<td>004700</td>
<td>Jump To Subroutine (JSR PC,xxx)</td>
</tr>
<tr>
<td>CALLR</td>
<td>000100</td>
<td>Jump (JMP addr)</td>
</tr>
<tr>
<td>CCC</td>
<td>000257</td>
<td>Clear All Condition Codes</td>
</tr>
<tr>
<td>CLC</td>
<td>000241</td>
<td>Clear C Condition Code Bit</td>
</tr>
<tr>
<td>CLN</td>
<td>000250</td>
<td>Clear N Condition Code Bit</td>
</tr>
<tr>
<td>CLR</td>
<td>005000</td>
<td>Clear Destination</td>
</tr>
<tr>
<td>CLRB</td>
<td>105000</td>
<td>Clear Destination (Byte)</td>
</tr>
<tr>
<td>CLV</td>
<td>000242</td>
<td>Clear V Condition Code Bit</td>
</tr>
<tr>
<td>CLZ</td>
<td>000244</td>
<td>Clear Z Condition Code Bit</td>
</tr>
<tr>
<td>CMP</td>
<td>020000</td>
<td>Compare Source To Destination</td>
</tr>
<tr>
<td>CMPB</td>
<td>120000</td>
<td>Compare Source To Destination (Byte)</td>
</tr>
<tr>
<td>COM</td>
<td>005100</td>
<td>Complement Destination</td>
</tr>
<tr>
<td>COMB</td>
<td>105100</td>
<td>Complement Destination (Byte)</td>
</tr>
<tr>
<td>DEC</td>
<td>005300</td>
<td>Decrement Destination</td>
</tr>
<tr>
<td>DECB</td>
<td>105300</td>
<td>Decrement Destination (Byte)</td>
</tr>
<tr>
<td>DIV</td>
<td>071000</td>
<td>Divide</td>
</tr>
<tr>
<td>EMT</td>
<td>104000</td>
<td>Emulator Trap</td>
</tr>
<tr>
<td>FADD</td>
<td>075000</td>
<td>Floating Add</td>
</tr>
<tr>
<td>Instruction Mnemonic</td>
<td>Octal Value</td>
<td>Operation</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>FDIV</td>
<td>075030</td>
<td>Floating Divide</td>
</tr>
<tr>
<td>FMUL</td>
<td>075020</td>
<td>Floating Multiply</td>
</tr>
<tr>
<td>FSUB</td>
<td>075010</td>
<td>Floating Subtract</td>
</tr>
<tr>
<td>HALT</td>
<td>000000</td>
<td>Halt</td>
</tr>
<tr>
<td>INC</td>
<td>005200</td>
<td>Increment Destination</td>
</tr>
<tr>
<td>INCB</td>
<td>105200</td>
<td>Increment Destination (Byte)</td>
</tr>
<tr>
<td>IOT</td>
<td>000004</td>
<td>Input/Output Trap</td>
</tr>
<tr>
<td>JMP</td>
<td>000100</td>
<td>Jump</td>
</tr>
<tr>
<td>JSR</td>
<td>004000</td>
<td>Jump To Subroutine</td>
</tr>
<tr>
<td>MARK</td>
<td>006400</td>
<td>Mark</td>
</tr>
<tr>
<td>MED6X</td>
<td>076600</td>
<td>PDP-11/60 Maintenance</td>
</tr>
<tr>
<td>MFPI</td>
<td>006500</td>
<td>Move From Previous Instruction Space</td>
</tr>
<tr>
<td>MFPS</td>
<td>106700</td>
<td>Move From PS (LSI-11, LSI-11/23, LSI-11/2)</td>
</tr>
<tr>
<td>MFPT</td>
<td>000007</td>
<td>Move From Processor Type</td>
</tr>
<tr>
<td>MOV</td>
<td>010000</td>
<td>Move Source To Destination</td>
</tr>
<tr>
<td>MOVB</td>
<td>110000</td>
<td>Move Source To Destination (Byte)</td>
</tr>
<tr>
<td>MTPS</td>
<td>006600</td>
<td>Move To Previous Instruction Space</td>
</tr>
<tr>
<td>MTPS</td>
<td>106400</td>
<td>Move To PS (LSI-11, LSI-11/23, LSI-11/2)</td>
</tr>
<tr>
<td>MUL</td>
<td>070000</td>
<td>Multiply</td>
</tr>
<tr>
<td>NEG</td>
<td>005400</td>
<td>Negate Destination</td>
</tr>
<tr>
<td>NEGB</td>
<td>105400</td>
<td>Negate Destination (Byte)</td>
</tr>
<tr>
<td>NOP</td>
<td>000240</td>
<td>No Operation</td>
</tr>
<tr>
<td>RESET</td>
<td>000005</td>
<td>Reset External Bus</td>
</tr>
<tr>
<td>RETURN</td>
<td>000207</td>
<td>Return From Subroutine (RTS PC)</td>
</tr>
<tr>
<td>ROL</td>
<td>006100</td>
<td>Rotate Left</td>
</tr>
<tr>
<td>ROLB</td>
<td>106100</td>
<td>Rotate Left (Byte)</td>
</tr>
<tr>
<td>ROR</td>
<td>006000</td>
<td>Rotate Right</td>
</tr>
<tr>
<td>RORB</td>
<td>106000</td>
<td>Rotate Right (Byte)</td>
</tr>
<tr>
<td>RTI</td>
<td>000002</td>
<td>Return From Interrupt (permits trace trap)</td>
</tr>
<tr>
<td>RTS</td>
<td>000200</td>
<td>Return From Subroutine</td>
</tr>
<tr>
<td>RTT</td>
<td>000006</td>
<td>Return From Interrupt (inhibits trace trap)</td>
</tr>
<tr>
<td>Instruction Mnemonic</td>
<td>Octal Value</td>
<td>Operation</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>SBC</td>
<td>005600</td>
<td>Subtract Carry</td>
</tr>
<tr>
<td>SBCB</td>
<td>105600</td>
<td>Subtract Carry (Byte)</td>
</tr>
<tr>
<td>SCC</td>
<td>000277</td>
<td>Set All Condition Code Bits</td>
</tr>
<tr>
<td>SEC</td>
<td>000261</td>
<td>Set C Condition Code Bit</td>
</tr>
<tr>
<td>SEN</td>
<td>000270</td>
<td>Set N Condition Code Bit</td>
</tr>
<tr>
<td>SEV</td>
<td>000262</td>
<td>Set V Condition Code Bit</td>
</tr>
<tr>
<td>SEZ</td>
<td>000264</td>
<td>Set Z Condition Code Bit</td>
</tr>
<tr>
<td>SOB</td>
<td>077000</td>
<td>Subtract One And Branch</td>
</tr>
<tr>
<td>SUB</td>
<td>160000</td>
<td>Subtract Source From Destination</td>
</tr>
<tr>
<td>SWAB</td>
<td>000300</td>
<td>Swap Bytes</td>
</tr>
<tr>
<td>SXT</td>
<td>006700</td>
<td>Sign Extend</td>
</tr>
<tr>
<td>TRAP</td>
<td>104400</td>
<td>Trap</td>
</tr>
<tr>
<td>TST</td>
<td>005700</td>
<td>Test Destination</td>
</tr>
<tr>
<td>TSTB</td>
<td>105700</td>
<td>Test Destination (Byte)</td>
</tr>
<tr>
<td>TSTSET</td>
<td>007200</td>
<td>Test Destination And Set Low Bit</td>
</tr>
<tr>
<td>WAIT</td>
<td>000001</td>
<td>Wait For Interrupt</td>
</tr>
<tr>
<td>WRTLCK</td>
<td>007300</td>
<td>Read/Lock Destination. Write/Unlock R0 Into Destination</td>
</tr>
<tr>
<td>XOR</td>
<td>074000</td>
<td>Exclusive OR</td>
</tr>
</tbody>
</table>
### C.2 Commercial Instruction Set (CIS) Op Codes

Every operation listed in the CIS table has two instruction mnemonics. The suffix I, attached to every second mnemonic, indicates that the addresses are inline. CIS instructions take no arguments.

<table>
<thead>
<tr>
<th>Instruction Mnemonic</th>
<th>Octal Value</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDN</td>
<td>076050</td>
<td>Add Numeric</td>
</tr>
<tr>
<td>ADDNI</td>
<td>076150</td>
<td>Add Numeric</td>
</tr>
<tr>
<td>ADDP</td>
<td>076070</td>
<td>Add Packed</td>
</tr>
<tr>
<td>ADDPI</td>
<td>076170</td>
<td>Add Packed</td>
</tr>
<tr>
<td>ASHN</td>
<td>076056</td>
<td>Arithmetic Shift Numeric</td>
</tr>
<tr>
<td>ASHNI</td>
<td>076156</td>
<td>Arithmetic Shift Numeric</td>
</tr>
<tr>
<td>ASHP</td>
<td>076076</td>
<td>Arithmetic Shift Packed</td>
</tr>
<tr>
<td>ASHPI</td>
<td>076176</td>
<td>Arithmetic Shift Packed</td>
</tr>
<tr>
<td>CMPC</td>
<td>076044</td>
<td>Compare Character String</td>
</tr>
<tr>
<td>CMPCI</td>
<td>076144</td>
<td>Compare Character String</td>
</tr>
<tr>
<td>CMPN</td>
<td>076052</td>
<td>Compare Numeric</td>
</tr>
<tr>
<td>CMPNI</td>
<td>076152</td>
<td>Compare Numeric</td>
</tr>
<tr>
<td>CMPP</td>
<td>076072</td>
<td>Compare Packed</td>
</tr>
<tr>
<td>CMPPI</td>
<td>076172</td>
<td>Compare Packed</td>
</tr>
<tr>
<td>CVTLN</td>
<td>076057</td>
<td>Convert Long To Numeric</td>
</tr>
<tr>
<td>CVTLNI</td>
<td>076157</td>
<td>Convert Long To Numeric</td>
</tr>
<tr>
<td>CVTLP</td>
<td>076077</td>
<td>Convert Long To Packed</td>
</tr>
<tr>
<td>CVTLPi</td>
<td>076177</td>
<td>Convert Long To Packed</td>
</tr>
<tr>
<td>CVTNP</td>
<td>076055</td>
<td>Convert Numeric To Packed</td>
</tr>
<tr>
<td>CVTNPi</td>
<td>076155</td>
<td>Convert Numeric To Packed</td>
</tr>
<tr>
<td>CVTPN</td>
<td>076054</td>
<td>Convert Packed To Numeric</td>
</tr>
<tr>
<td>CVTPNI</td>
<td>076154</td>
<td>Convert Packed To Numeric</td>
</tr>
<tr>
<td>DIVP</td>
<td>076075</td>
<td>Divide Decimal</td>
</tr>
<tr>
<td>DIVPI</td>
<td>076175</td>
<td>Divide Decimal</td>
</tr>
<tr>
<td>LOCC</td>
<td>076040</td>
<td>Locate Character</td>
</tr>
<tr>
<td>LOCCI</td>
<td>076140</td>
<td>Locate Character</td>
</tr>
<tr>
<td>L2Dn(^1)</td>
<td>07602n</td>
<td>Load 2 Descriptors @(Rn)+</td>
</tr>
</tbody>
</table>

\(^1\) \(n = 0 \) to \(7\).
<table>
<thead>
<tr>
<th>Instruction Mnemonic</th>
<th>Octal Value</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3Dn¹</td>
<td>07606n</td>
<td>Load 3 Descriptors @(Rn)+</td>
</tr>
<tr>
<td>MATC</td>
<td>076045</td>
<td>Match Character</td>
</tr>
<tr>
<td>MATCI</td>
<td>076145</td>
<td>Match Character</td>
</tr>
<tr>
<td>MOVC</td>
<td>076030</td>
<td>Move Character</td>
</tr>
<tr>
<td>MOVCI</td>
<td>076130</td>
<td>Move Character</td>
</tr>
<tr>
<td>MOVRC</td>
<td>076031</td>
<td>Move Reverse Justified Character</td>
</tr>
<tr>
<td>MOVRCI</td>
<td>076131</td>
<td>Move Reverse Justified Character</td>
</tr>
<tr>
<td>MOVTC</td>
<td>076032</td>
<td>Move Translated Character</td>
</tr>
<tr>
<td>MOVTCI</td>
<td>076132</td>
<td>Move Translated Character</td>
</tr>
<tr>
<td>MULP</td>
<td>076074</td>
<td>Multiply Decimal</td>
</tr>
<tr>
<td>MULPI</td>
<td>076174</td>
<td>Multiply Decimal</td>
</tr>
<tr>
<td>SCANC</td>
<td>076042</td>
<td>Scan Character</td>
</tr>
<tr>
<td>SCANCI</td>
<td>076142</td>
<td>Scan Character</td>
</tr>
<tr>
<td>SKPC</td>
<td>076041</td>
<td>Skip Character</td>
</tr>
<tr>
<td>SKPCI</td>
<td>076141</td>
<td>Skip Character</td>
</tr>
<tr>
<td>SPANC</td>
<td>076043</td>
<td>Span Character</td>
</tr>
<tr>
<td>SPANCI</td>
<td>076143</td>
<td>Span Character</td>
</tr>
<tr>
<td>SUBN</td>
<td>076051</td>
<td>Subtract Numeric</td>
</tr>
<tr>
<td>SUBNI</td>
<td>076151</td>
<td>Subtract Numeric</td>
</tr>
<tr>
<td>SUBP</td>
<td>076071</td>
<td>Subtract Packed</td>
</tr>
<tr>
<td>SUBPI</td>
<td>076171</td>
<td>Subtract Packed</td>
</tr>
</tbody>
</table>

¹n = 0 to 7.
## C.3 Floating-Point Processor Op Codes

<table>
<thead>
<tr>
<th>Instruction Mnemonic</th>
<th>Octal Value</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSD</td>
<td>170600</td>
<td>Make Absolute Double</td>
</tr>
<tr>
<td>ABSF</td>
<td>170600</td>
<td>Make Absolute Floating</td>
</tr>
<tr>
<td>ADDD</td>
<td>172000</td>
<td>Add Double</td>
</tr>
<tr>
<td>ADDF</td>
<td>172000</td>
<td>Add Floating</td>
</tr>
<tr>
<td>CFCC</td>
<td>170000</td>
<td>Copy Floating Condition Codes</td>
</tr>
<tr>
<td>CLRDF</td>
<td>170400</td>
<td>Clear Double</td>
</tr>
<tr>
<td>CLRFC</td>
<td>170400</td>
<td>Clear Floating</td>
</tr>
<tr>
<td>CMPD</td>
<td>173400</td>
<td>Compare Double</td>
</tr>
<tr>
<td>CMPF</td>
<td>173400</td>
<td>Compare Floating</td>
</tr>
<tr>
<td>DIVD</td>
<td>174400</td>
<td>Divide Double</td>
</tr>
<tr>
<td>DIVF</td>
<td>174400</td>
<td>Divide Floating</td>
</tr>
<tr>
<td>LDCDF</td>
<td>177400</td>
<td>Load And Convert From Double To Floating</td>
</tr>
<tr>
<td>LDCFD</td>
<td>177400</td>
<td>Load And Convert From Floating To Double</td>
</tr>
<tr>
<td>LDCID</td>
<td>177000</td>
<td>Load And Convert Integer To Double</td>
</tr>
<tr>
<td>LDCIF</td>
<td>177000</td>
<td>Load And Convert Integer To Floating</td>
</tr>
<tr>
<td>LDCLD</td>
<td>177000</td>
<td>Load And Convert Long Integer To Double</td>
</tr>
<tr>
<td>LDCLF</td>
<td>177000</td>
<td>Load And Convert Long Integer To Floating</td>
</tr>
<tr>
<td>LDD</td>
<td>172400</td>
<td>Load Double</td>
</tr>
<tr>
<td>LDEXP</td>
<td>176400</td>
<td>Load Exponent</td>
</tr>
<tr>
<td>LDF</td>
<td>172400</td>
<td>Load Floating</td>
</tr>
<tr>
<td>LDFPS</td>
<td>170100</td>
<td>Load FPPs Program Status</td>
</tr>
<tr>
<td>MFPD</td>
<td>106500</td>
<td>Move From Previous Data Space</td>
</tr>
<tr>
<td>MODD</td>
<td>171400</td>
<td>Multiply And Integerize Double</td>
</tr>
<tr>
<td>MODF</td>
<td>171400</td>
<td>Multiply And Integerize Floating</td>
</tr>
<tr>
<td>MTPD</td>
<td>106600</td>
<td>Move To Previous Data Space</td>
</tr>
<tr>
<td>MULD</td>
<td>171000</td>
<td>Multiply Double</td>
</tr>
<tr>
<td>MULF</td>
<td>171000</td>
<td>Multiply Floating</td>
</tr>
<tr>
<td>NEGD</td>
<td>170700</td>
<td>Negate Double</td>
</tr>
<tr>
<td>NEGF</td>
<td>170700</td>
<td>Negate Floating</td>
</tr>
<tr>
<td>Instruction Mnemonic</td>
<td>Octal Value</td>
<td>Operation</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>SETD</td>
<td>170011</td>
<td>Set Double Mode</td>
</tr>
<tr>
<td>SETF</td>
<td>170001</td>
<td>Set Floating Mode</td>
</tr>
<tr>
<td>SETI</td>
<td>170002</td>
<td>Set Integer Mode</td>
</tr>
<tr>
<td>SETL</td>
<td>170012</td>
<td>Set Long Integer Mode</td>
</tr>
<tr>
<td>SPL</td>
<td>000230</td>
<td>Set Priority Level</td>
</tr>
<tr>
<td>STA0</td>
<td>170005</td>
<td>Diagnostic Floating Point</td>
</tr>
<tr>
<td>STB0</td>
<td>170006</td>
<td>Diagnostic Floating Point</td>
</tr>
<tr>
<td>STCDF</td>
<td>176000</td>
<td>Store And Convert From Double To Floating</td>
</tr>
<tr>
<td>STCDI</td>
<td>175400</td>
<td>Store And Convert From Double To Integer</td>
</tr>
<tr>
<td>STCDL</td>
<td>175400</td>
<td>Store And Convert From Double To Long Integer</td>
</tr>
<tr>
<td>STCFD</td>
<td>176000</td>
<td>Store And Convert From Floating To Double</td>
</tr>
<tr>
<td>STCFI</td>
<td>175400</td>
<td>Store And Convert From Floating To Integer</td>
</tr>
<tr>
<td>STCFL</td>
<td>175400</td>
<td>Store And Convert From Floating To Long Integer</td>
</tr>
<tr>
<td>STD</td>
<td>174000</td>
<td>Store Double</td>
</tr>
<tr>
<td>STEXP</td>
<td>175000</td>
<td>Store Exponent</td>
</tr>
<tr>
<td>STF</td>
<td>174000</td>
<td>Store Floating</td>
</tr>
<tr>
<td>STFPS</td>
<td>170200</td>
<td>Store FPPs Program Status</td>
</tr>
<tr>
<td>STST</td>
<td>170300</td>
<td>Store FPPs Status</td>
</tr>
<tr>
<td>SUBD</td>
<td>173000</td>
<td>Subtract Double</td>
</tr>
<tr>
<td>SUBF</td>
<td>173000</td>
<td>Subtract Floating</td>
</tr>
<tr>
<td>TSTD</td>
<td>170500</td>
<td>Test Double</td>
</tr>
<tr>
<td>TSTF</td>
<td>170500</td>
<td>Test Floating</td>
</tr>
</tbody>
</table>
### C.4 MACRO–11 Directives

The MACRO–11 directives that follow are described in greater detail in Appendix B.

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ASCII</td>
<td>Translates character string to ASCII equivalents.</td>
</tr>
<tr>
<td>.ASCIIZ</td>
<td>Translates character string to ASCII equivalents; inserts zero byte as last character.</td>
</tr>
<tr>
<td>.ASECT</td>
<td>Begins absolute program section (provided for compatibility with other PDP–11 assemblers).</td>
</tr>
<tr>
<td>.BLKB</td>
<td>Reserves byte block in accordance with value of specified argument.</td>
</tr>
<tr>
<td>.BLKW</td>
<td>Reserves word block in accordance with value of specified argument.</td>
</tr>
<tr>
<td>.BYTE</td>
<td>Generates successive byte data in accordance with specified arguments.</td>
</tr>
<tr>
<td>.CROSS</td>
<td>Enables cross-reference listing for specified symbols; enables cross-reference for all symbols.</td>
</tr>
<tr>
<td>.CSECT</td>
<td>Begins relocatable program section (provided for compatibility with other PDP–11 assemblers).</td>
</tr>
<tr>
<td>.DSABL</td>
<td>Disables specified function.</td>
</tr>
<tr>
<td>.ENABL</td>
<td>Enables specified function.</td>
</tr>
<tr>
<td>.END</td>
<td>Defines logical end of source program.</td>
</tr>
<tr>
<td>.ENDC</td>
<td>Defines end of conditional assembly block.</td>
</tr>
<tr>
<td>.ENDM</td>
<td>Defines end of macro definition, repeat block, or indefinite repeat block.</td>
</tr>
<tr>
<td>.ENDR</td>
<td>Defines end of current repeat block (provided for compatibility with other PDP–11 assemblers).</td>
</tr>
<tr>
<td>.ERROR</td>
<td>Outputs diagnostic message to listing file or command output device.</td>
</tr>
<tr>
<td>.EVEN</td>
<td>Word aligns the current location counter.</td>
</tr>
<tr>
<td>.FLT2</td>
<td>Generates two words of storage for each floating-point argument.</td>
</tr>
<tr>
<td>.FLT4</td>
<td>Generates four words of storage for each floating-point argument.</td>
</tr>
<tr>
<td>.GLOBL</td>
<td>Declares global attribute for specified symbol(s).</td>
</tr>
<tr>
<td>.IDENT</td>
<td>Labels object module with specified program version number.</td>
</tr>
<tr>
<td>.IF</td>
<td>Begins conditional assembly block.</td>
</tr>
<tr>
<td>.IFF</td>
<td>Begins subconditional assembly block (if conditional assembly block test is false).</td>
</tr>
<tr>
<td>.IFT</td>
<td>Begins subconditional assembly block (if conditional assembly block test is true).</td>
</tr>
<tr>
<td>.IFTF</td>
<td>Begins subconditional assembly block (whether conditional assembly block test is true or false).</td>
</tr>
<tr>
<td>Directive</td>
<td>Function</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>.IIF</td>
<td>Assembles immediate conditional assembly statement (if specified condition is satisfied).</td>
</tr>
<tr>
<td>.INCLUDE</td>
<td>Inserts specified source file within source file currently being used.</td>
</tr>
<tr>
<td>.IRP</td>
<td>Begins indefinite repeat block; replaces specified symbol with specified successive real arguments.</td>
</tr>
<tr>
<td>.IRPC</td>
<td>Begins indefinite repeat block; replaces specified symbol with value of successive characters in specified string.</td>
</tr>
<tr>
<td>.LIBRARY</td>
<td>Adds a specified file name to a macro library list that is searched.</td>
</tr>
<tr>
<td>.LIMIT</td>
<td>Reserves two words of storage for high and low addresses of task image.</td>
</tr>
<tr>
<td>.LIST</td>
<td>Controls listing level count and format of assembly listing.</td>
</tr>
<tr>
<td>.MACRO</td>
<td>Denotes start of macro definition.</td>
</tr>
<tr>
<td>.MCALL</td>
<td>Identifies required macro definition(s) for assembly.</td>
</tr>
<tr>
<td>.MDELETE</td>
<td>Deletes the definitions of the specified macro(s).</td>
</tr>
<tr>
<td>.MEXIT</td>
<td>Exits from current macro definition or indefinite repeat block.</td>
</tr>
<tr>
<td>.NARG</td>
<td>Equates specified symbol to the number of nonkeyword arguments in the macro expansion.</td>
</tr>
<tr>
<td>.NCHR</td>
<td>Equates specified symbol to the number of characters in the specified character string.</td>
</tr>
<tr>
<td>.NLIST</td>
<td>Controls listing level count and suppresses specified portions of the assembly listing.</td>
</tr>
<tr>
<td>.NOCROSS</td>
<td>Disables cross-reference listing for specified symbols; disables cross-reference listing for all symbols.</td>
</tr>
<tr>
<td>.NTYPE</td>
<td>Equates specified symbols to the addressing mode of the specified argument.</td>
</tr>
<tr>
<td>.OCT</td>
<td>Byte aligns the current location counter.</td>
</tr>
<tr>
<td>.PACKED</td>
<td>Generates packed decimal data, two digits per byte.</td>
</tr>
<tr>
<td>.PAGE</td>
<td>Advances form to top of next page.</td>
</tr>
<tr>
<td>.PRINT</td>
<td>Prints specified message on command output device.</td>
</tr>
<tr>
<td>.PSECT</td>
<td>Begins specified program section having specified attributes.</td>
</tr>
<tr>
<td>.RAD50</td>
<td>Generates data block having Radix–50 equivalents of specified character string.</td>
</tr>
<tr>
<td>.RADIX</td>
<td>Changes current program radix to specified radix.</td>
</tr>
<tr>
<td>.REM</td>
<td>Inserts a block of comments into a MACRO–11 source program without having to precede comment lines with the comment character (;).</td>
</tr>
<tr>
<td>.REPT</td>
<td>Begins repeat block and replicates it according to the value of the specified expression.</td>
</tr>
<tr>
<td>Directive</td>
<td>Function</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>.RESTORE</td>
<td>Stores the current program section context on the top of the program section context stack.</td>
</tr>
<tr>
<td>.SAVE</td>
<td>Retrieves the program section from the top of the program section context stack.</td>
</tr>
<tr>
<td>.SBTTL</td>
<td>Prints specified subtitle text as the second line of the assembly listing page header.</td>
</tr>
<tr>
<td>.TITLE</td>
<td>Prints specified title text as object module name in the first line of the assembly listing page header.</td>
</tr>
<tr>
<td>.WEAK</td>
<td>Specifies symbols that are either defined externally in another module or are defined globally in the current module.</td>
</tr>
<tr>
<td>.WORD</td>
<td>Generates successive word data in accordance with specified arguments.</td>
</tr>
</tbody>
</table>
Appendix D

Error Messages

An error code is printed as the first character in a source line containing an error. This error code identifies the error condition detected during the processing of the line. For example:

Q 26 000236 010102 MOV R1, R2, A

The extraneous argument A in the MOV instruction above causes the line to be flagged with a Q (syntax) error.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Assembly error. Because many different conditions produce this error message, the directives which may yield a general assembly error have been categorized below to reflect these error conditions:</td>
</tr>
<tr>
<td></td>
<td><strong>CATEGORY 1: INVALID ARGUMENT SPECIFIED</strong></td>
</tr>
<tr>
<td>.ENABL/.DSABL</td>
<td>Table 6–3 contains a list of the valid arguments for this directive.</td>
</tr>
<tr>
<td>.IF/.IFF</td>
<td>An invalid conditional test (see Table 6–6), an invalid argument expression value, or no conditional argument is specified in the directive.</td>
</tr>
<tr>
<td>.IRP/.IRPC</td>
<td>No dummy argument is specified in the directive.</td>
</tr>
<tr>
<td>.LIST/.NLIST</td>
<td>Table 6–2 contains a list of the valid arguments for this directive.</td>
</tr>
<tr>
<td>.MACRO</td>
<td>There is an invalid or duplicate symbol in the dummy argument list.</td>
</tr>
<tr>
<td>.NARG/.NCHR</td>
<td>No symbol is specified in the directive.</td>
</tr>
<tr>
<td>.NTYPE</td>
<td>Other than a valid argument (see Table 6–4) is specified with the directive, or the attribute arguments of a previously declared program section change (see Section 6.7.1.1).</td>
</tr>
<tr>
<td>.PSECT</td>
<td>A value other than 2, 8, 10, or 16 is specified as a new radix.</td>
</tr>
<tr>
<td>.RADIX</td>
<td>Program name is not specified in the directive, or first non-blank character following the directive is a non-Radix–50 character.</td>
</tr>
</tbody>
</table>
Error Code | Meaning
--- | ---
CATEGORY 2: UNMATCHED DELIMITER/INVALID ARGUMENT CONSTRUCTION
 | Character string or argument string delimiters do not match, or an invalid character is used as a delimiter, or an invalid argument construction is used in the directive.
 | Character string delimiters do not match, or an invalid character is used as a delimiter in the directive.

CATEGORY 3: GENERAL ADDRESSING ERRORS
This type of error results from one of several possible conditions:

- Permissible range of a branch instruction (from \(-128\,_{10}\) to \(+127\,_{10}\) words) has been exceeded.
- A statement makes invalid use of the current location counter. For example, a \(=\)expression statement attempts to force the current location counter to cross program section (.PSECT) boundaries.
- A statement contains an invalid address expression:

In cases where an absolute address expression is required, specifying a global symbol, a relocatable value, or a complex relocatable value (see Section 3.9) results in an invalid address expression. For example, this error occurs with .BLKB/.BLKW/.REPT if other than an absolute value or an expression which reduces to an absolute value is specified with the directive.

If an undefined symbol is made a default global reference by the .ENABL GBL directive (see Section 6.2.1) during pass 1, any attempt to redefine the symbol during pass 2 will result in an invalid address expression.

In cases where a relocatable address expression is required, either a relocatable or absolute value is permissible, but a global symbol or a complex relocatable value in the statement results in an invalid address expression.

- Multiple expressions are not separated by a comma. This condition causes the next symbol to be evaluated as part of the current expression.
- .SAVE—The stack is full when the .SAVE directive is issued.
- .RESTORE—The stack is empty when the .RESTORE directive is issued.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B</strong></td>
<td>Bounding error. Instructions or word data are being assembled at an odd address. The location counter is incremented by 1. Insert a <code>.EVEN</code> statement before the statement that generates the error.</td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>Doubly-defined symbol referenced. Reference was made to a symbol which is defined more than once. Remove one of the definitions or rename one of the symbols to something else.</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>End directive not found. When the end-of-file is reached during source input and the <code>.END</code> directive has not yet been encountered, MACRO-11 generates this error code, ends assembly pass 1, and proceeds with assembly pass 2. Put a <code>.END</code> directive at the end of the program. This error is also caused by assembler stack overflow. In this case MACRO-11 places a question mark (?) into the line at the point where the overflow occurred.</td>
</tr>
<tr>
<td><strong>I</strong></td>
<td>Invalid character detected. Invalid characters which are also nonprinting are replaced by a question mark (?) on the listing. The character is then ignored. Delete the characters, or replace them with characters in the valid MACRO-11 character set.</td>
</tr>
<tr>
<td><strong>L</strong></td>
<td>Input line is greater than 132₁₀ characters. This error condition is caused only during macro expansion when longer real arguments, replacing the dummy arguments, cause a line to exceed 132₁₀ characters. Rewrite the macro so this does not occur.</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label. Rename one of the labels to something else.</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td>A number contains a digit that is not in the current program radix. The number is evaluated as a decimal value. Change the erroneous digit so the number is valid in the current program radix, or redefine the current radix with the <code>.RADIX</code> directive or one of the temporary radix operators.</td>
</tr>
</tbody>
</table>

**CATEGORY 4: INVALID FORWARD REFERENCE**

This type of error results from either of two possible conditions:

- A global assignment statement (`symbol==expression` or `symbol==:expression`) contains a forward reference to another symbol.
- An expression defining the value of the current location counter contains a forward reference.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>Op code error. Directive out of context. Permissible nesting level depth for conditional assemblies has been exceeded. Attempt to expand a macro which was unidentified after a .MCALL search. Check syntax and context. Make sure conditional nesting does not exceed 16 levels.</td>
</tr>
<tr>
<td>P</td>
<td>Phase error. A label’s definition of value varies from one assembly pass to another, or a multiple definition of a local symbol has occurred within a local symbol block. This situation may occur if you define a local symbol block using the .ENABLE LSB directive, then attempt to define a local symbol in a program section other than that which was in effect when the block was entered. An error code P also appears if a .ERROR directive is assembled. You may also be using the conditional tests .IF P1 and .IF P2 incorrectly. Check and correct program logic.</td>
</tr>
<tr>
<td>Q</td>
<td>Questionable syntax. Arguments are missing, too many arguments are specified, or the instruction scan was not completed. Verify that instruction syntax is correct. Also, be sure the program does not contain a carriage return with no line feed or a colon instead of a semicolon at the beginning of a comment.</td>
</tr>
<tr>
<td>R</td>
<td>Register error. An invalid use of or reference to a register has been made, or an attempt has been made to redefine a standard register symbol without first issuing the .DISABLE REG directive.</td>
</tr>
<tr>
<td>T</td>
<td>Truncation error. A number generated more than 16 bits in a word. You may get this error if you specify an octal or decimal value but the radix is set to hexadecimal or if you specify an octal value but the radix is set to decimal.</td>
</tr>
<tr>
<td>U</td>
<td>Undefined symbol. An undefined symbol was encountered during the evaluation of an expression; such an undefined symbol is assigned a value of zero. Other possible conditions which result in this error code include unsatisfied macro names in the list of .MCALL arguments and a direct assignment (symbol=expression or symbol:=expression) statement which contains a forward reference to a symbol whose definition also contains a forward reference; also, a local symbol may have been referenced that does not exist in the current local symbol block.</td>
</tr>
<tr>
<td>Z</td>
<td>Instruction error. The instruction so flagged is not compatible among all members of the PDP-11 family. See Section 5.3 for details.</td>
</tr>
</tbody>
</table>
Appendix E
Sample Coding Standard

Local user requirements must be met in a coding standard, but following this model as closely as possible helps you and DIGITAL by simplifying communication and software maintenance. Remember that this is a sample and may not entirely apply to your system.

E.1 Line Format

Source lines are from 1 to 80 characters in the following format:

1. Label Field—If present, begins in column 1. This field should be coded in uppercase only.

2. Operation field—Begins in column 9 (tab stop 1). This field should be coded in uppercase only.

3. Operand field—Begins in column 17 (tab stop 2). This field should be coded in uppercase only.

4. Comment field—Begins in column 33 (tab stop 4). If the operand field extends beyond column 33 (tab stop 4), leave a space and start the comment. This field should be coded in uppercase and lowercase to increase readability.

E.2 Comments

To make the program easier to understand, use comments to explain the logic behind the instructions. In general, you should use a comment per line of code. However, if a particularly difficult or obscure section of code is used, precede that section with a longer explanation.

Comments that are too long for the comment field can be continued on the following line. Begin the new line with a semicolon, space over to the column the comment began in, and continue writing. All comments should be written in uppercase and lowercase to increase readability.

If a lengthy text is needed for an explanation, begin the comment with a line containing only the characters ;+ and end it with a line containing only the characters ;-. The lines between these delimiters should each begin with a semicolon and a space. For example:

;+
; The invert routine accepts
; a list of random numbers and
; applies the Kolmogorov Algorithm
; to alphabetize them.
;->
E.3 Naming Standards

E.3.1 Registers

For the general purpose registers, use the default names:

R0=%0 ;REG 0
R1=%1 ;REG 1
R2=%2 ;REG 2
R3=%3 ;REG 3
R4=%4 ;REG 4
R5=%5 ;REG 5
SP=%6 ;Stack pointer (REG 6)
PC=%7 ;Program counter (REG 7)

For hardware registers, use the hardware definition. Examples are PS (Program Status Register) and SWR (Switch Register).

For device registers, use the hardware notation. For example, the control status register for the RK disk is RKCS.

E.3.2 Processor Priority

Test or alter the processor priority by using the symbols:

PR0, PR1, PR2, . . . , PR7

which should be equated to their corresponding priority bit pattern.

E.3.3 Symbols

The following chart diagrams the syntax of the five major types of symbol names:

<table>
<thead>
<tr>
<th>symbol</th>
<th>pos-1</th>
<th>pos-2</th>
<th>pos-3</th>
<th>pos-4</th>
<th>pos-5</th>
<th>pos-6</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonglobal symbol</td>
<td>letter</td>
<td>a-num/</td>
<td>a-num/</td>
<td>a-num/</td>
<td>a-num/</td>
<td>a-num/</td>
<td>&gt; =1</td>
</tr>
<tr>
<td>global symbol</td>
<td>$.</td>
<td>a-num/</td>
<td>a-num/</td>
<td>a-num/</td>
<td>a-num/</td>
<td>a-num/</td>
<td>&gt; =1</td>
</tr>
<tr>
<td>global offset</td>
<td>letter</td>
<td>$.</td>
<td>a-num</td>
<td>a-num/</td>
<td>a-num/</td>
<td>a-num/</td>
<td>&gt; =3</td>
</tr>
<tr>
<td>global bit pattern</td>
<td>letter</td>
<td>a-num</td>
<td>$.</td>
<td>a-num</td>
<td>a-num/</td>
<td>a-num/</td>
<td>&gt; =4</td>
</tr>
<tr>
<td>local symbol</td>
<td>number</td>
<td>$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; =2</td>
</tr>
</tbody>
</table>

NOTES:

letter is A-Z.
a-num is an alphanumeric character.

1 Symbols that are branch targets are also called labels, but in this appendix the term symbol includes labels.
<table>
<thead>
<tr>
<th>symbol</th>
<th>pos-1</th>
<th>pos-2</th>
<th>pos-3</th>
<th>pos-4</th>
<th>pos-5</th>
<th>pos-6</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>is the absence of a character in the position.</td>
</tr>
<tr>
<td>$/.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>are reserved for DIGITAL-supplied software; do not use $ or . in your global symbols to avoid possible conflict with globals, for example, in distributed libraries.</td>
</tr>
<tr>
<td>number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>is in the range 0 to 65535_{10}.</td>
</tr>
</tbody>
</table>

### E.3.3.1 Symbol Examples

**Nonglobal Symbols:**

- A1B
- ZXCJ1
- INSRT

**Global Address Symbols:**

- $JIM
- .VECTR
- $SEC

**Global Absolute Offset Symbols:**

- A$JIM
- A$XT
- A.ENT

**Global Bit Pattern Symbols:**

- A1$20
- B3.6
- JI.M

**Local Symbols:**

- 37$
- 271$
- 6$

### E.3.3.2 Local Symbols

When defining target symbols for branches that exist solely for positional reference, use local symbols of the form:

- `<number>`$:

Define local symbols so the numbers proceed sequentially down the page and from page to page.
E.3.3.3 Global Symbols
Restrict your use of global symbols, within reason, to those cases where reference to
the code occurs external to the code.

Never put a .GLOBL statement in a program without showing cause.

E.3.3.4 Macro Names (RSX–11)
In a macro name, the last two characters (last character possibly being null) have
special significance: the next to last character is a $, the last character specifies the
mode of the macro.

For example, in the three RSX–11 macro forms inline, stack, and p-section, the inline
form has no suffix, the stack has an $ suffix, and the p-section a c. Thus the RSX–11
Queue I/O macro can be written as any of:

```
QIO$
QI0S$
QI0SC
```

depending on the form required. These are not reserved letters.

E.3.3.5 General Symbols
Make frequently used bit patterns such as carriage return and line feed conventional
symbols as they are needed, for example:

```
CR = 015
LF = 012
```

E.4 Program Modules
There are no assembler limits on program size. However, since the virtual memory
capacity of a computer is finite, keep programs as compact as possible by:

- Creating them for a single function
- Writing them in accordance with the memory allocation guidelines in Appendix F

Code areas are different from data areas. Code is read-only, but data can be read-only
or read-write; read-only data should be segregated from read-write data. Both areas,
code and data, should have explanatory comments.

E.4.1 The Module Preface
Put each program module in a separate file. For easy reference, the file name should
be similar to the name of the module. The availability of File Control Services and
File Control Primitives simplify version number maintenance.

E.4.2 The Module
Below is a list of the information that is included in the example MACRO–11 module
(see Section E.4.3). The information is formatted as follows. The first six items appear
on the same page and do not have explicit headings.
1. A .NLIST statement, followed by any .ENABL/.DSABL or .NLIST/.LIST options that are relevant to the assembly of this module, followed by a matching .LIST statement. The .NLIST statement has a comment appended to it specifying the module edit level.

2. A .TITLE statement that specifies the name of the module. If a module contains more than one routine, .SBTTL statements are used.

3. Several .SBTTL statements giving the name, general function, and version number of the module. The .SBTTL directive inserts this information in the table of contents for quick reference.

4. A .IDENT statement that specifies the version number of the module (see Section E.8).

5. A copyright statement, and a disclaimer, followed by a form feed. The copyright, even though a comment, should be all uppercase. This ensures that the copyright will be presented correctly, even on a terminal that has only uppercase.

6. The name of the program or software package that the module is a part of.

7. The name of the author.

8. The date of module creation.

9. A 1- or 2-line abstract of the function(s) of the module.

10. A description of all external references made by the module, one per line, in alphabetical order.

11. A chronological edit trail of modifications to the module that includes the following:
   - Edit number
   - Editor's identification
   - Edit date
   - Description of the modification made

**NOTE**

Items 7 through 11 should appear on the same page.

12. Any references to external files by the .LIBRARY and .INCLUDE directives.

13. .MCALLS to any externally defined macros.

14. A list of the definitions of all equated symbols used in the module. These definitions should appear one per line and in alphabetical order.

15. All local macro definitions, preferably in alphabetical order.

16. All local data. The comments in this section should include:
   - Description of each element (type, size, and so forth)
• Organization (functional, alphabetical, adjacent, and so forth)
• Adjacency requirements (if any)

17. A form feed, followed by a .SBTTL statement describing the routine that follows.

18. A routine header, giving the following information:
• Routine name
• Description
• Inputs
• Calling sequence
• Outputs
• Side effects, register usage, and so forth

**NOTE**
Repeat items 17 and 18 for every routine within the module.

**E.4.3 Module Example**

```plaintext
.NLIST
.ENABLE GBL
.LIST MEB
.TITLE MACINI - Once-only code for the MACRO-11 assembler
.SBTTL
.IDENT /Y05.01/

******************************************************************************
:* *
:* COPYRIGHT (c) 1982, 1983 *
:* BY DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS. *
:* ALL RIGHTS RESERVED. *
:* *
:* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
:* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
:* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
:* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
:* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
:* TRANSFERRED. *
:* *
:* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
:* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
:* CORPORATION. *
:* *
:* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
:* SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL. *
:* *
:******************************************************************************
```

---

MACRO-11: The PDP-11 macro assembler for RT/RSX/VMS and RSTS/E

Author: Joe Worrall

Created: 21-Aug-82

Abstract: MACINI contains code only executed once per task invocation.

<table>
<thead>
<tr>
<th>Externals</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$LIBID</td>
<td>File-ID of the system library account (LB:[1,1])</td>
</tr>
<tr>
<td>$POSID</td>
<td>File-ID of the P/OS library account (LB:[1,5])</td>
</tr>
<tr>
<td>$STABF</td>
<td>Workfile statistics buffer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edit</th>
<th>Who</th>
<th>Date</th>
<th>Description of modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Jrw</td>
<td>25-Aug-82</td>
<td>Handle P/OS .PARSE module.</td>
</tr>
<tr>
<td>002</td>
<td>Jrw</td>
<td>05-Sep-82</td>
<td>Allow recursive FINIT$'s.</td>
</tr>
<tr>
<td>003</td>
<td>Jrw</td>
<td>10-Nov-82</td>
<td>Setup statistics buffer.</td>
</tr>
</tbody>
</table>

External file references:
.LIBRARY /MACLIB/ ;Add MACLIB.MLB to macro library list
.INCLUDE /MACPRE/ ;Include MACPRE.MAC in assembly

External library ".MCALL's" for this module:

.MCALL FINIT$

Equated symbols:
... Equated symbols ...

Local macros:
... Local macros ...

Local data:
... Local data ...

.SBTTL $INIT - Handle once only code for MACRO-11 assembler
$INIT

This routine is a collection of all the code, only executed once in any one run of the MACRO-11 task. It's collected here because:

- It's logical to keep it in one place
- It keeps the code out of the root, keeping the assembler SMALL.

INPUTS: n/a

CALL: CALL $INIT

OUTPUTS:

- Record management, statistics, and FCS buffers are setup. If the system contains EIS support, the DIV and MUL routine vectors are setup to point to the hardware instructions.

EFFECTS: R0 - R5 Destroyed!

... Begin module code ...

E.4.4 Modularity

No other characteristic has more impact on the ultimate engineering success of a system than does modularity. Adherence to a set of call and return conventions helps achieve this modularity.

E.4.4.1 Calling Conventions (Inter-Module/Intra-Module)

Transfer of Control

Macros exist for call and return. The actual transfer is via a JSR PC instruction. For register save routines, a JSR Rn,SAVE is permitted.

The CALL macro is:

CALL subr-name

The RETURN macro is:

RETURN

Register Conventions

On entry, a subroutine minimally saves all registers it intends to alter except result registers. On exit, it restores these registers. (The preservation of the register state is assumed across calls.)

Argument Passing

Any registers can be used, but their use should follow a coherent pattern. For example, if passing three arguments, use R0, R1, and R2 rather than R0, R2, and R5. Saving and restoring occurs in one place.
E.4.4.2 Exiting
All subroutine exits occur through a single RETURN macro.

E.4.4.3 Success/Failure Indication
The C-bit is used to return the success/failure indicator, where success equals 0, and failure equals 1. The argument registers can be used to return values or additional success/failure data.

E.4.4.4 Module Checking Routines
Modules are responsible for verifying the validity of arguments passed to them. The design of a module's calling sequence should aim at minimizing the validity checks by minimizing invalid combinations. Programmers may add test code to perform additional testing during checkout. All code should aim at discovering an error as close (in terms of instruction executions) to its occurrence as possible.

E.5 Code Format
E.5.1 Program Flow
Programs are organized on the listing so that they flow down the page, even at the cost of an extra branch or jump. All unconditional branch and jump instructions should be followed by a blank line. This causes these instructions to stand out in the source code, allowing the code to be traced more easily. For example:

```
Process

Test

BBB

AAA

Common
```

MLD-1256-87
appears on the listing as:

```
TST
BNE  BBB
AAA:  . . . .
       . . . .
       . . . .
       . . . .
       BR  CMN
BBB:  . . . .
       . . . .
       . . . .
       . . . .
       . . . .
       . . . .
       BR  CMN
```

rather than:

```
TST
BNE  BBB
AAA:  . . . .
       . . . .
       . . . .
       . . . .
       . . . .
       CMN
BBB:  . . . .
       . . . .
       . . . .
       . . . .
       . . . .
       BR  CMN
```

**E.5.2 Common Exits**

A common exit appears as the last code sequence on the listing. Thus, the flow chart:

```
PR1
  ↓
  PR2
  ↓
  PR3
  ↓
  Exit
  ↓
  PR4
```

MLO-1257-87
appears on the listing as:

PR1:

....
....
....
BR   EXIT

PR2:

....
....
....
BR   EXIT

PR3:

....
....
....
BR   EXIT

PR4:

....
....
....

EXIT:

and not as:

PR1:

....
....
....

EXIT:

....
....
....

PR2:

....
....
....
BR   EXIT

PR3:

....
....
....
BR   EXIT

PR4:

....
....
....
BR   EXIT
E.5.3 Code with Interrupts Inhibited

Code executed with interrupts inhibited is flagged by a 3-semicolon (;;;) comment delimiter, for example:

```assembly
..ERTZ: ;Enable by returning
;by system subroutines,
BIS   #PR7,PS    ; inhibit interrupts
BIT   #PR7,2(SP) c
BEQ   10$       o
RTT   m
        m
10$:
        e
        n
        t
        s
```

E.5.4 Code in System State

RSX-11M executive subroutines and other privileged code executed in system state is flagged by a 2-semicolon (;;) comment delimiter, for example:

```assembly
; Switch to system state, ...
... ; and exit.
CALL   $SWSTK,EXIT ; Inhibit context switching
... ; Return in system state
... ...
RETURN ; Go back to user state (EXIT)
EXIT:  ; User state code
```

E.6 Instruction Usage

E.6.1 Forbidden Instructions

You should avoid certain instruction combinations because they make a program hard to read, debug, and maintain. Avoid the following programming practices:

- The use of instructions or index words as literals of the previous instruction. For example:

  ```assembly
  MOV   @PC,REGISTER
  BIC   SRC,DST
  ```

  uses the bit clear instruction as a literal. This may seem to be a very neat way to save a word, but the practice can easily confuse the next person who has to work on the program. To compound the problem, the instruction will not execute properly if I/D space is enabled. In that case, @PC is a D-space reference.
• The use of the MOV instruction instead of a JMP instruction to transfer program control to another location. For example:

\[
\text{MOV} \quad \#\text{ALPHA}, \text{PC}
\]

transfers control to location ALPHA. Besides taking longer to execute, the use of MOV instead of JMP makes it nearly impossible to pick up someone else's program and tell where transfers of control take place. As a more general issue, other operations such as ADD and SUB from PC should be discouraged.

• The seemingly clever use of all single-word instructions where one double-word instruction could be used, which would execute faster and not consume any additional memory. Consider the following instruction sequence:

\[
\begin{align*}
\text{CMP} & \quad -(\text{R1}), -(\text{R1}) \\
\text{CMP} & \quad -(\text{R1}), -(\text{R1})
\end{align*}
\]

The intent of this instruction sequence is to subtract 8 from register R1 (not to set condition codes). This can be accomplished in approximately 1/3 the time via a SUB instruction at no additional cost in memory space. The practice can also cause a memory management fault on a mapped system if the value in R1 happens to look like an unmapped address.

• Self-relative address arithmetic (+.n) is absolutely forbidden in branch instructions; its use in other contexts must be avoided if at all possible and practical.

**E.6.2 Conditional Branches**

When using the PDP-11 conditional branch instructions, you must make the correct choice between the signed and the unsigned branches.

<table>
<thead>
<tr>
<th>Signed</th>
<th>Unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGE</td>
<td>BHIS (BCC)</td>
</tr>
<tr>
<td>BLT</td>
<td>BLO</td>
</tr>
<tr>
<td>BGT</td>
<td>BHI</td>
</tr>
<tr>
<td>BLE</td>
<td>BLOS (BCS)</td>
</tr>
</tbody>
</table>

A common pitfall is to use a signed branch (for example, BGT) when comparing two memory addresses. This works until the two addresses have opposite signs; that is, one of them goes across the 16K (100000<sub>8</sub>) bound. This type of coding error usually results from relinking the program at different addresses and/or changing the size of the program.


E.7 Program Source Files

Source creation and maintenance are done in base levels. A base level is the point at which the program source files have been frozen. From the freeze point to the next base level, corrections are not made directly to the base level itself. Rather, a file of corrections is accumulated for each file in the base level. Whenever an updated source file is desired, the correction file is applied to the base file.

The accumulation of corrections proceeds until a logical breaking point has occurred (a milestone or significant implementation point has been reached). At this time, all accumulated corrections are applied to the previous base level to create a new base level, and correction files are started for the new base level.

E.8 PDP–11 Version Number Standard

The PDP–11 Version Number Standard applies to all modules, parameter files, complete programs, and libraries which are written as part of the PDP–11 Software Development effort. It is used to provide unique identification of all released, prereleased, and in-house software.

The version number is limited in that only six characters of identification are used. Future implementations of the Macro Assembler, Linker, and Librarian should provide for at least nine characters, and possibly twelve. It is expected that this standard will be improved as the need arises.

Version Identifier Format:

\[ <\text{version}> <\text{edit}> <\text{patch}> \]

where:

- \(<\text{version}>\) consists of two decimal digits which represent the release number of a program. The version number starts at 00 and is incremented to reflect the number of major changes in the program.

- \(<\text{edit}>\) consists of two decimal digits which represent the number of alterations made to the source program. The edit number begins at 01 (is blank if there are no edits) and is incremented with each alteration.

- \(<\text{patch}>\) is a letter between B and Z which represents the number of alterations made to the binary form of the program. The patch number begins at B (is blank if there are no patches) and changes alphabetically with each patch.

These fields are interrelated. When \(<\text{version}>\) is changed, then \(<\text{patch}>\) and \(<\text{edit}>\) must be reset to blank. It is intended that when \(<\text{edit}>\) is incremented, then \(<\text{patch}>\) will be reset to blank, because the various bugs have been fixed.
E.8.1 Displaying the Version Identifier

The visible output of the version identifier should appear as:

```
Program
    Name <key-letter> <version> . <edit> <patch>
```

where the following Key Letters have been identified:

- X: in-house experimental version
- Y: field test, prerelease, or in-house release version
- V: released or frozen version

'X' corresponds roughly to individual support, 'Y' to group support, and 'V' to company support.

The dot (.) which separates <version> from <edit> is not used if both <edit> and <patch> are null. When a version identifier is displayed as part of program identification, then the format is:

```
programname <space><key-letter><version> . <edit><patch>
```

Examples:

- PIP V05.00
- LINK V08.00
- MACRO V05.00

E.8.2 Use of the Version Number in the Program

All sources must contain the version number in a .IDENT directive. In programs (or libraries) which consist of more than one module, each module must have a version number. The version number of the program or library is not necessarily related to the version numbers of the constituent modules; it is perfectly reasonable, for example, that the first version of a new FORTRAN library, V00, contain an existing SIN routine, for example V05.01.

Parameter files are also required to contain the version number in a .IDENT directive. Because the assembler records the last .IDENT seen, parameter files must precede the program.

Entities which consist of a collection of modules or programs (for example, the FORTRAN Library) have an identification module in the first position. An identification module exists solely to provide identification. For example:

```
:OTS identification
.TITLE FTNLIB
.IDENT /V02.00/
.END
```

is an identification module.
Appendix F
Allocating Virtual Memory

This appendix is intended for the MACRO–11 user who wants to avoid the problem of thrashing by optimizing the allocation of virtual memory. If you have a small system, you should pay particular attention to these conventions.

This appendix discusses the following topics:

• General hints and space-saving guidelines
• Macro definitions and expansions
• Operational techniques

This discussion assumes that you have used modular programming, as advised in Appendix E. Modular programming results in bodies of code that are small, distinct, and highly functional. Using such code, which presents many advantages, one can usually avoid the problem of insufficient dynamic memory during assembly.

F.1 GeneralHints and Space Saving Guidelines

Working memory is shared by a number of MACRO–11’s tables, each of which is allocated space on demand (64K words of dynamically pageable storage are available to the assembler). The tables and their corresponding entry sizes are as follows:

• User-defined symbols—five words
• Local symbols—three words
• Program sections—six words
• Macro names—five words
• Macro text—nine words
• Source files—six words

In addition, several scratch pad tables are used during the assembly process, as follows:

• Expression analysis—five words
• Object code generation—five words
• Macro argument processing—three words
• .MCALL argument processing—five words

This information can serve as a guide for estimating dynamic storage requirements and for determining ways to reduce such requirements.
For example, the use of local symbols whenever possible is highly encouraged, since their internal representation requires 25 percent less dynamic storage than that required for regular user-defined symbols. The usage of local symbols can often be maximized by extending the scope of local symbol blocks through the .ENABL LSB/.DSABL LSB MACRO-11 directives (see Sections 3.5 and 6.2.1).

Since MACRO-11 does not support a purge function, once a symbol is defined, it permanently occupies its dynamic memory allocation. Numerous instances occur during conditional assemblies and repeat loops when a temporarily assigned symbol is used as a count or offset indicator. If possible, the symbols so used should be reused.

In keeping with the same principle, special treatment should be given to the definition of commonly used symbols. Instead of simply appending a prefix file which defines all possibly used symbols for each assembly, group symbols into logical classes. Each class can then become a shortened prefix file or a macro in a library (see Section F.2 below). In either case, selective definition of symbolic assignments is achieved, resulting in fewer defined (but unreferenced) symbols.

An example of this idea is seen in the definition of IAS and RSX-11M standard symbols. The RSX system macro library, for example, supplies several macros used to define distinct classes of symbols. These groupings and associated macro names are as follows:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRERR$</td>
<td>Directive return status codes</td>
</tr>
<tr>
<td>FILIO$</td>
<td>File-related I/O function codes</td>
</tr>
<tr>
<td>IOERR$</td>
<td>I/O return status codes</td>
</tr>
<tr>
<td>SPcio$</td>
<td>Special I/O function codes</td>
</tr>
</tbody>
</table>

**F.2 Macro Definitions and Expansions**

Dynamic storage is used most heavily for the storage of macro text. Upon macro definition or the issuance of a .MACALL directive, the entire macro body is stored, including all comments appearing in the macro definition. For this reason, comments should not be included as part of the macro text. Under RSX-11, a Librarian function switch (/SZ) is available to compress macro source text by removing all trailing blanks and tabs, blank lines, and comments. The RSX-11 system macro library (RSXMAC.SML) has already been compressed. User-supplied macro libraries (.MLB) and macro definition prefix files should also be compressed. For additional information regarding these two utility tasks, consult the applicable RSX-11M or RSX-11M-PLUS Utilities Manual (see the Associated Documents section in the Preface).

It often seems practical to include a file of commonly used macro definitions in each assembly. This practice, however, may produce the undesirable allocation of valuable dynamic storage for unnecessary macros. This waste of memory can be avoided by making the file of macro definitions a user-supplied macro library file (see Table 8–1). In that case, the names of desired macros must be listed as arguments in the .MACALL directive (see Section 7.8), or the automatic MACRO call, .ENABL NCL, must be enabled (see Section 6.2.1).
You can delete macro definitions after they have been called by using the .MDELETE request (see Section 7.9). This practice not only frees storage space, it also eliminates the overhead and the dynamic memory wasted by calling a useless macro. Alternatively, certain types of macros can be redefined to null after they have been called. The practice of deleting macros or redefining macros to null applies mainly to those that define symbolic assignments, as shown in the example below. The redefinition process can be accomplished as follows:

```
.MACRO DEFIN
SYM1 = VAL1 ;Define symbolic assignments.
SYM2 = VAL2

OFF1 = SYMBOL ;Define symbolic offsets.
OFF2 = OFF1+SIZ1
OFF3 = OFF2+SIZ2

OFFN = OFFM+SIZM

.MACRO DEFIN ;Macro null redefinition.
.ENDM

.ENDM DEFIN
```

Macros that are to be deleted or redefined should be defined (or read via the .MCALL directive) and called before all other macro definition and/or .MCALL processing. This procedure ensures more efficient use of dynamic memory.

### F.3 Operational Techniques

When, despite your adherence to the guidelines discussed above, performance still falls below expectations, several additional measures can be taken to increase dynamic memory.

The first measure involves shifting the burden of symbol definition from MACRO-11 to the Linker or Task Builder. In most cases, the definition of system I/O and File Control Services (FCS) symbols (and user-defined symbols of the same nature) is not necessary during the assembly process, since such symbols are defaulted to global references (Appendix D, error code A). The Linker or Task Builder attempts to resolve all global references from user-specified default libraries and/or the system object library (SYSLIB). Furthermore, by applying the selective search option for object modules consisting only of global symbol definitions, the actual additional burden to the Linker is minimal.

The second way is to produce only one output file (either object or listing), as opposed to two. The additional memory required to support the second output file is allocated from available dynamic memory at the start of each assembly.
Appendix G

Writing Position-Independent Code

G.1 Introduction to Position-Independent Code

The output of a MACRO-11 assembly is a relocatable object module. The Task Builder or Linker binds one or more modules together to create an executable task image. Once created, if the program is to run, it must be loaded at the virtual address specified at link time. This is because the Task Builder or Linker has to modify some instructions to reflect the memory locations in which the program is to run. Such a body of code is considered position-dependent (dependent on the virtual addresses to which it is bound).

All PDP-11 processors offer addressing modes that make it possible to write code that does not depend on the virtual addresses to which it is bound. Such code is termed position-independent and to run can be loaded at any virtual address. Position-independent code can improve system efficiency, both in use of virtual address space and in conservation of physical memory.

In multiprogramming systems like IAS, RSX-11M, and RSX-11M-PLUS, it is important that many tasks be able to share a single physical copy of common code, for example, a library routine. To make the optimum use of a task's virtual address space, shared code should be position-independent. Position-dependent code can also be shared, but it must appear in the same virtual locations in every task using it. This restricts the placement of such code by the Task Builder or Linker and can result in the loss of virtual addressing space.

The construction of position-independent code is closely linked to the proper usage of PDP-11 addressing modes. The remainder of this Appendix assumes that you are familiar with the addressing modes described in Chapter 5.

All addressing modes involving only register references are position-independent. These modes are as follows:

- $R$  register mode
- $(R)$  register deferred mode
- $(R) +$  autoincrement mode
- $(R) +$  autoincrement deferred mode
- $-(R)$  autodecrement mode
- $-(R)$  autodecrement deferred mode

When you use these addressing modes, your code is guaranteed position-independent, provided the contents of the registers have been supplied such that they are not dependent upon a particular virtual memory location.
The relative addressing modes are position-independent when a relocatable address is referenced from a relocatable instruction. These modes are as follows:

- **A** relative mode
- **QA** relative deferred mode

Relative modes are not position-independent when an absolute address (that is a non-relocatable address) is referenced from a relocatable instruction. In this case, absolute addressing (**#A**) can be used to make the reference position-independent.

Index modes can be either position-independent or position-dependent, according to their use in the program. These modes are as follows:

- **X(R)** index mode
- **@X(R)** index deferred mode

If the base, X, is an absolute value (for example, a control block offset), the reference is position-independent. For example:

```
MOV 2(SP), RO ;Position-independent
N=4
MOV N(SP), RO ;Position-independent
```

However, if X is a relocatable address, the reference is position-dependent. For example:

```
CLR ADDR(RI) ;Position-dependent
```

Immediate mode can be either position-independent or not, according to its usage. Immediate mode references are formatted as follows:

```
#N  immediate mode
```

When an absolute expression defines the value of N, the code is position-independent. When a relocatable expression defines N, the code is position-dependent. That is, immediate mode references are position-independent only when N is an absolute value.

Absolute mode addressing is position-independent only in those cases where an absolute virtual location is being referenced. Absolute mode addressing references are formatted as follows:

```
@#A  absolute mode
```

An example of a position-independent absolute reference is a reference to the directive status word ($DSW) from a relocatable instruction. For example:

```
MOV @#$DSW, RO ;Retrieve directive status
```

### G.2 Examples

The RSX-11M library routine **PWRUP** is a FORTRAN-callable subroutine that establishes or removes a user power failure Asynchronous System Trap (AST) entry point address. Embedded within the routine is the AST entry point that saves all registers, effects a call to the user-specified entry point, restores all registers on return, and executes an
AST exit directive. The following examples are excerpts from this routine. The first example, Figure G–1, has been modified to illustrate position-dependent references. The second example, Figure G–2, is the position-independent version.

**Figure G–1: Example of Position-Dependent Code**

```
;* Position-dependent code example
;
PWRUP:: CLR -(SP) ;Assume success
; Perform further initialization...
    MOV $OTSV,R4 ;Point R4 at object time system save area
    ;the above reference to $OTSV is position-dependent
    MOV (SP)+,R2 ;Retrieve AST entry point address
    BNE 10$ ;Branch if one was specified
    CLR -(SP) ;If none, specify no power fail routine
10$:
    MOV R2,F.PF(R4) ;Set the AST entry point
    MOV #BA, -(SP) ;the above reference to BA is position-dependent
20$:
; Continue processing...
;
; AST service routine
;
BA: MOV RO, -(SP) ;Preserve RO
; Rest of routine follows...
```
Figure G–2: Example of Position-Independent Code

; Position independent code example
; Assume success

PWRUP:: CLR - (SP) ; Perform necessary initialization...

MOV @#$OTS V,R4 ; Point R4 at object time system save area
; the above reference to $OTS V is position-independent
MOV (SP)+,R2 ; Retrieve AST entry point address
BNE 10$ ; Branch if one was specified
CLR -(SP) ; If none, specify no power fail routine

10$: MOV R2,F.PF(R4) ; Set the AST entry point
MOV PC, - (SP) ; Push our PC to relocate our AST service addr
ADD #BA-, (SP) ; Relocate our AST service address now
; the above reference to BA is position-dependent

20$: ; Continue processing...

; AST service routine

BA: MOV RO, - (SP) ; Preserve RO

; Rest of routine follows...

The position-dependent version of the subroutine contains a relative reference to an absolute symbol ($OTS V) and a literal reference to a relocatable symbol (BA). Both references are bound by the Task Builder to fixed memory locations. Therefore, the routine will not execute properly as part of a resident library if its location in virtual memory is not the same as the location specified at link time.

In the position-independent version, the reference to $OTS V has been changed to an absolute reference. In addition, the necessary code has been added to compute the virtual location of BA, based upon the value of the program counter. In this case, the value is obtained by adding the value of the program counter to the fixed displacement between the current location and the specified symbol. Thus, execution of the modified routine is not affected by its location in the image's virtual address space.
The MACRO-11 Assembler provides a way of checking whether the code is position-independent. In an assembly listing, MACRO-11 inserts a single quote (') character following the contents of any word which requires the Task Builder or Linker to perform a relocation operation and, therefore, may not be position-independent code. Cases that are flagged by a single quote in the assembly listing are as follows:

- Absolute mode references, when the reference is relocatable. References are not flagged when they are absolute. For example:
  
  MOV   @#ADDR,R1     ;PIC only if ADDR is absolute.

- Index and index deferred mode references, when the offset is relocatable. For example:
  
  MOV   ADDR(R1),R5   ;Non-PIC if ADDR is relocatable.
  MOV   @ADDR(R1),R5  ;Non-PIC if ADDR is relocatable.

- Relative and relative deferred mode references, when the specified address is relocatable with respect to another program section. For example:
  
  MOV   ADDR1,R1      ;Non-PIC when ADDR1 is absolute.
  MOV   @ADDR1,R1

- Immediate mode references to relocatable addresses.
  
  MOV   #ADDR,R1      ;Non-PIC when ADDR is relocatable.

In one case, MACRO-11 does not flag a potential position-dependent reference. This occurs where a relative reference is made to an absolute virtual location from a relocatable instruction (see the MOV $0TSV,R4 instruction in Figure G-1).

References requiring more than simple relocation at link time are indicated in the assembly listing. Simple global references are flagged with the letter G. Statements that contain multiple global references or require complex relocation are flagged with the letter C (see Section 3.9 and Chapter 4). It is difficult to state with certainty whether or not a C-flagged statement is position-independent. However, in general, position dependence can be decided by applying the guidelines discussed earlier in this Appendix to the resulting address value produced at link time.
Sample Assembly and Cross-Reference Listing

2- 1 RAD50 unpack routine

RAD50 unpack routine

1  .TITLE  RSOUNP
2  .IDENT  '03/
3
4  ; Copyright (c) 1979, 1987 by
5  ; Digital Equipment Corporation, Maynard, Mass.
6  ;
7  ; This software is furnished under a license and may be used and copied
8  ; only in accord with the terms of such license and with the
9  ; inclusion of the above copyright notice. This software or any other
10  ; copies thereof may not be provided or otherwise made available to any
11  ; other person. No title to and ownership of the software is hereby
12  ; transferred.
13  ; The information in this software is subject to change without notice
14  ; and should not be construed as a commitment by Digital Equipment
15  ; Corporation.
16  ;
17  ; Digital assumes no responsibility for the use or reliability of its
18  ; software on equipment which is not supplied by Digital.
19  ;
20  ; Update history:
21  ;
22  ; D.H. Cutler  10-Feb-73
23  ; SGW  25-Mar-87

RSOUNP MACRO V05.04  Wednesday 25-Mar-87  16:49

1 .ENTRY RSOUNP
2
3 ; Unpack a 6 char RAD50 symbol to ASCII
4 ;
5 ; Enter with R2 -> Output ASCII string
6 ; SYMBOL, SYMBOL+2 -> RAD50 symbol to unpack
7 ;
8 ; Return with R2 -> Past output string
9 ; RO, R1, R3 destroyed
10 
11 .GLOBAL SYMBOL
12
13 000000 .PROCURE PURE1,I
14 00446  RSOUNP: MOV  R4,-(SP) ; Save R4
15 00449  012704 MOV  #SYMBOL,R4 ; Point at RAD50 symbol buffer
16 0044D  0800006 012401 MOV  (R4)+,R1 ; Get next RAD50 word
17 00452  012703 MOV  #50+50,R3 ; Set divisor for high character
18 00455  003100 004767 CALL  10$ ; Unpack and store the character
19 0045B  000030 012703 MOV  #50,R3 ; Now set divisor for middle character
20 0045E  000200 004767 CALL  10$ ; Unpack and store the character
21 00463  000024 001000 MOV  R1,R0 ; Copy remaining character
22 00466  00032 004767 CALL  11$ ; Translate and store it
23 0046F  000018

Sample Assembly and Cross-Reference Listing  H-1
27 000036 020427  CMP  R4,#SYMBO8+4  ;Test if last word done
000044G
28 000042 001191  BNE  18  ;Branch if no
29 000044 012804  MOV  (SP)+,R4  ;Restore R4
30 000046 000207 38:  RETURN  ;Return to caller
31 32  ; Translate RAD60 character code to ASCII
33 34  0 = space
35 36  1-32 = A-Z
37 38  33 = 
39 40  34 = 
41 42  35 = unused code
43 44  36-47 = 0-9
45 46 000060 005000 10#:  CLR  RO  ;Divide RAD60 word, get
47 48 000062 071003  DIV  R3,RO  ;remainder (RAD60 char) in R0
49 50 000064 118F22 11#:  MOV  TABLE(RO),(R2)+ ;Get ASCII equivalent of RAD60
51 000062 51  RETURN
52 53 .LIST  BEX
54 55 000069 040  TABLE:  .BYTE  ' ', 'A', 'B', 'C', 'D', 'E', 'F', 'G'
56 57 000072 110  .BYTE  'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O'
58 59 000102 120  .BYTE  'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W'
60 61 000112 130  .BYTE  'X', 'Y', 'Z', '$', '$', '$', '$', '$'
62 63 .EXIT
64 .B80030  MACRO  V06.04  Wednesday 25-Mar-87 16:49  Page 2-1
65 .RAD60 unpack routine
66 67 000122 082  .BYTE  '2', '3', '4', '5', '6', '7', '8', '9'
68 69 000001  . END
70 71 .B80030  MACRO  V06.04  Wednesday 25-Mar-87 16:49  Page 2-2
72 .Symbol table
73 .B80030 0000008G 002 SYMBOL= ****** G  TABLE  0000062R  002
74 75  . ABS  000000 000  (RW,I,GR,AB,OCR)
76 77  000000 001  (RW,I,LCL,REL,COD)
78 79  .PUBLIC  000132 002  (RW,I,LCL,REL,COD)
80 81 Errors detected: 0
82 *** Assembler statistics
83 Work file reads: 0
84 Work file writes: 0
85 Size of work file: 64 Words ( 1 Pages)
86 Size of core pool: 17820 Words ( 70 Pages)
87 Operating system: ET-11
88 Elapsed time: 00:00:02.45
90 .B80030  MACRO  V06.04  Wednesday 25-Mar-87 16:49  Page 8-2
91 Cross reference table (CREF V06.04)
92 .B80030 2-18#  SYMBOL  2-14 2-19  2-27
93 TABLE  2-42 2-46#
94 .B80030  MACRO  V06.04  Wednesday 25-Mar-87 16:49  Page 8-1
95 Cross reference table (CREF V06.04)
96 .B80030 2-18#  SYMBOL  2-14 2-19  2-27
97 TABLE  2-42 2-46#
98 .B80030  MACRO  V06.04  Wednesday 25-Mar-87 16:49  Page 8-1
99 Cross reference table (CREF V06.04)
100 .B80030 2-18#  SYMBOL  2-14 2-19  2-27
101 TABLE  2-42 2-46#
102 .B80030  MACRO  V06.04  Wednesday 25-Mar-87 16:49  Page 8-1
103 Cross reference table (CREF V06.04)
104 .B80030 2-18#  SYMBOL  2-14 2-19  2-27
105 TABLE  2-42 2-46#
106 .B80030  MACRO  V06.04  Wednesday 25-Mar-87 16:49  Page 8-1
107 Cross reference table (CREF V06.04)
108 .B80030 2-18#  SYMBOL  2-14 2-19  2-27
109 TABLE  2-42 2-46#
110 Errors detected: 0
111 .ABS  0-0
112 .PUBLIC  2-16

Appendix I

Obsolete MACRO–11 Directives, Syntax, and Command Line Options

1.1 Obsolete Directives and Syntax

Although supported in older versions of MACRO–11, the following directives and syntax are not supported in the current release. Table 1–1 shows both the old directives and syntax and the new syntax to use. All MACRO–11 code that contains the old directives and syntax should be updated to use the new syntax.

Table 1–1: Old and New Directives and Syntax

<table>
<thead>
<tr>
<th>Syntax no longer supported</th>
<th>New syntax to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>.EOT</td>
<td>None</td>
</tr>
<tr>
<td>.IFZ xxx or .IFEQ xxx</td>
<td>.IF EQ,xxx</td>
</tr>
<tr>
<td>.IF Z,xxx</td>
<td>.IF EQ,xxx</td>
</tr>
<tr>
<td>.IFNZ xxx or .IFNE xxx</td>
<td>.IF NE,xxx</td>
</tr>
<tr>
<td>.IF NZ,xxx</td>
<td>.IF NE,xxx</td>
</tr>
<tr>
<td>.IFL xxx or .IFLT xxx</td>
<td>.IF LT,xxx</td>
</tr>
<tr>
<td>.IF L,xxx</td>
<td>.IF LT,xxx</td>
</tr>
<tr>
<td>.IFG xxx or .IFGT xxx</td>
<td>.IF GT,xxx</td>
</tr>
<tr>
<td>.IF G,xxx</td>
<td>.IF GT,xxx</td>
</tr>
<tr>
<td>.IFLE xxx</td>
<td>.IF LE,xxx</td>
</tr>
<tr>
<td>.IFDF xxx</td>
<td>.IF DF,xxx</td>
</tr>
<tr>
<td>.IFNDF xxx</td>
<td>.IF NDF,xxx</td>
</tr>
</tbody>
</table>

1.2 Obsolete Command Line Option

DIGITAL no longer supports the MACRO–11 command line option /P[ASS] :n. This option was originally created to speed up assemblies in some cases by scanning a given file with only one pass of the assembler. However, DIGITAL has found that the /P[ASS] :n switch has many side effects; it has caused more problems than can be documented reasonably.

Although the /P[ASS] :n option is still accepted by MACRO–11, DIGITAL no longer accepts SPRs relating to the option and has removed all documentation for it. You should update any assembly command files containing the /P[ASS] :n option by removing the references to the option.
Appendix J

Release Notes

This appendix explains the changes that have been made to MACRO-11 for each version release since Version 5.0. The new features mentioned are documented in Chapters 1 through 9 of this manual. Previous versions of this appendix assigned some of the changes to the wrong version numbers of the software. Accordingly, this appendix has been rearranged so that each change is listed under the version number when the change was actually made. If you have Version 5.5 of MACRO-11, you need not worry about when the changes happened, because all the changes will be in place. If you are still using an older version of MACRO-11, however, this appendix can help you determine whether or not a problem in that version has been fixed in a later version of the software.

NOTE
The version numbers of MACRO-11 and its host operating system are completely independent of each other. MACRO-11 V5.3 was distributed with RT-11 V5.2, for example. Either version number may change without affecting the other. Be careful not to confuse the two.

J.1 Changes—All Versions of MACRO-11

J.1.1 V5.5 Update Changes

- Hexadecimal support was added:
  *`X, for temporary hexadecimal radix
  .RADIX 16, for permanent radix change
  .LIST HEX, to produce a hexadecimal listing
- Two new conditionals were added: .IF P1 and .IF P2.
- MACRO-11 now accepts more than 254₁₀ program sections, although only the first 254 appear in the symbol table. Previously, more than 254 program sections caused an assembly error.
- If a .PAGE directive is issued with the listing already at top-of-page, the .PAGE directive is ignored. In previous versions of MACRO-11, the page number was incremented, even though a new physical page was not printed.
- A .PAGE directive in an unexpanded macro is now ignored.
- The XOR instruction is now flagged with the Z error for certain addressing modes (execution may be different on different processors).
• Space between a macro name and the opening left angle bracket of an argument list is now optional. In previous versions, a space or tab was required. For example, if BUILD is a macro name, BUILD<A,B,C> is now valid; previously, it had to be written as BUILD <A,B,C>.

• In previous versions of MACRO-11, it was possible to change the value of a symbol that was assigned a value by using the =: operator, even though values assigned with =: are supposed to be permanent. MACRO-11 now retains a symbol's permanent attribute in all cases.

• .SBTTL lines in listings now include up to 80 characters.

• The .IRPC directive now accepts arguments of up to 124 characters; the previous limit was 96 characters.

• The error ?MACRO-F-Internal error (on RT-11/RSTS) or MAC—Internal error (on IAS/RSX) was added. If you get this error, please submit an SPR to DIGITAL along with a method of duplicating the problem.

J.1.2 V5.4 Update Changes

In previous versions of MACRO-11, the assembler parsed the arguments of .IF conditional statements even if the .IF statements were within unsatisfied conditional code blocks. This could cause assembly errors when there should be none. MACRO-11 no longer parses the arguments of conditional statements within blocks of code that do not get assembled.

J.1.3 V5.3 Update Changes

• MACRO-11 did not mark symbolic expressions as complex when they contained a symbol from a relocatable psect and a symbol from an absolute psect. That resulted in incorrect Linker output if the base of the absolute psect was not zero. Those symbolic expressions are now correctly marked as complex.

• MACRO-11 did not mark symbolic expressions as complex when they contained symbols from different absolute psects. That resulted in incorrect Linker output if the base of either psect was not zero. Those symbolic expressions are now correctly marked as complex.

• When MACRO-11 directly assigned the current location counter symbol (.) to a global symbol in an absolute psect, MACRO-11 incorrectly bound the global symbol to the .ABS. psect. MACRO-11 now correctly binds the global symbol to the absolute psect in which the assignment occurred.

• If MACRO-11 encountered a label containing invalid characters, MACRO-11 would hang in an infinite loop if there existed a macro with the same name as the valid part of the label name. MACRO-11 now correctly returns an error.
J.1.4 V5.2 Update Changes

- MACRO-11 does not allow the colon (:) character as a delimiter for .ASCII or .ASCIZ strings. This is now documented in Chapter 6.
- MACRO-11 now provides support for the 8-bit DEC Multinational character set (MCS). A chart showing the MCS is located in Appendix A.

The following directives support the MCS. For specific support information, consult the description of each directive.

<table>
<thead>
<tr>
<th>Macro Directive</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ASCII</td>
<td>6.3.4</td>
</tr>
<tr>
<td>.ASCIZ</td>
<td>6.3.5</td>
</tr>
<tr>
<td>.ERROR</td>
<td>7.5</td>
</tr>
<tr>
<td>.IF</td>
<td>6.9.1</td>
</tr>
<tr>
<td>.IF DIF</td>
<td>6.9.2</td>
</tr>
<tr>
<td>.IF IDN</td>
<td></td>
</tr>
<tr>
<td>.IFF</td>
<td></td>
</tr>
<tr>
<td>.IFF DIF</td>
<td>7.6.1</td>
</tr>
<tr>
<td>.IFF IDN</td>
<td></td>
</tr>
<tr>
<td>.IRP</td>
<td></td>
</tr>
<tr>
<td>.IRPC</td>
<td>7.6.2</td>
</tr>
<tr>
<td>.NCHR</td>
<td>7.4.2</td>
</tr>
<tr>
<td>.PRINT</td>
<td>7.5</td>
</tr>
<tr>
<td>.REM</td>
<td>6.1.6</td>
</tr>
<tr>
<td>.SBTTL</td>
<td>6.1.3</td>
</tr>
<tr>
<td>.TITLE</td>
<td>6.1.2</td>
</tr>
</tbody>
</table>

Further information on the 8-bit DEC Multinational character set is located in sections:

- 2.2.4 Comment field
- 6.3.3 ASCII conversion characters
- 7.3 Arguments in macro definitions and macro calls
- 7.3.6 Keyword arguments

J.1.5 V5.1 Update Changes

- MACRO-11 processed some index deferred arguments as floating-point numbers by default. MACRO-11 now processes all index deferred arguments as octal by default.
• MACRO–11 did not mark internal displaced relocatable statements as relocatable with a single quote (') in the assembly listing. They are now marked correctly.

• MACRO–11 set bit 3, an unused bit, in all .PSECT object records. MACRO–11 no longer sets bit 3. That change makes object files created with the new version of MACRO–11 different from object files created with previous versions of MACRO–11. As a result, they will have different PAT checksums, and a binary comparison of the files will show differences. However, the resulting task or .SAV image files will be the same.

J.1.6 V5.0 Update Changes

• The op code CALLR addr (Call-Return) has been added to the permanent symbol table (PST). This op code is equivalent to the JMP addr op code. The CALLR addr op code was added to complement the CALL addr op code, which is equivalent to the JSR PC, addr op code.

• The previous version of MACRO–11 used a range of 64$ to 127$ for automatic local symbol generation. MACRO–11 now uses a range of 30000$ to 65535$ when generating local symbols.

• Most assembler generated listing text is now in uppercase and lowercase. This change was made to increase the readability of MACRO–11 code. Lines of code that include the .SBTTL or the .TITLE directive are not converted to uppercase.

• Lines of code that include the .SBTTL directive are listed in the table of contents of an assembly listing, even if a .NLIST statement is in effect at the time the .SBTTL lines are encountered. You can specify the .NLIST directive with the TOC argument to prevent the table of contents from being printed.

• The symbol table is printed at the end of an assembly, even if the .NLIST directive is in effect. You can specify the .NLIST directive with the SYM argument to prevent the symbol table from being printed.

• All page headers include the day of the week.

• The assembler statistics information that appears at the end of the assembly listing file has been updated to include the following additional information:
  - Total number of virtual workfile reads
  - Total number of virtual workfile writes
  - Maximum amount of virtual memory used (in words and pages)
  - Size of physical memory free space (in words and pages)
  - Operating system and environment that the assembler is running under
  - Total elapsed assembly time
  - MACRO–11 command line

• The program section (.PSECT) synopsis that is printed after the symbol table in the listing file includes the program section attributes.
• The maximum number of relocatable terms in a complex expression has been changed. The maximum size of a .OBJ record that MACRO–11 can produce was increased from $42_{10}$ bytes to $128_{10}$ bytes.

Do not compare .OBJ files that have been created by different versions of MACRO–11 when verifying whether your code generation is correct. Changes that have been made for this version of MACRO–11 (mentioned above) invalidate a direct comparison of assembler .OBJ output. Verify code generation by linking or taskbuilding the .OBJ files involved and then comparing the .SAV or the .TSK image files.

**NOTE**

.ZOBJ files produced by this version of MACRO–11 are different from those produced by older versions. If you use the PAT (object file patch utility), checksums must be recomputed on any object patches assembled with this new version of MACRO–11.

• The default for the LC argument has been changed from .DSABL LC to .ENABL LC.

• The following .ENABL/.DSABL options have been added:
  — .ENABL LCM/.DSABL LCM
  — .ENABL MCL/.DSABL MCL

• The following directives have been added to MACRO–11 and documented in this manual.
  — .CROSS
  — .INCLUDE
  — .LIBRARY
  — .MDELETE
  — .NOCROSS
  — .REM
  — .WEAK

**J.2 Changes—MACRO–11/RSX Version Only**

**J.2.1 V5.5 Update Changes**

There were no RSX–specific changes made to MACRO–11 V5.5.

**J.2.2 V5.4 Update Changes**

There were no RSX–specific changes made to MACRO–11 V5.4.
J.2.3 V5.3 Update Changes

- Previous versions of MACRO-11 would hang in an infinite loop if they encountered a record with an invalid record size. That problem has been fixed.

- MACRO-11 now fully supports RSX logical names by calling the .CSI4 SYSLIB parsing routine.

J.2.4 V5.2 Update Changes

There were no RSX-specific changes made to MACRO-11 V5.2.

J.2.5 V5.1 Update Changes

- Previous versions of MACRO-11 would exit with SUCCESS exit status even though errors were reported. That problem has been fixed.

- If MACRO-11 detected an I/O error while reading a command file, MACRO-11 would produce an odd-address trap. Now, MACRO-11 reports the error message MAC—Command I/O error.

J.2.6 V5.0 Update Changes

- The cross-reference options SEC and ERR have been added.

NOTE

The RSX-11 CREF program (CRF) has been updated to include support for these two new macro cross-reference options. Only the new RSX-11 CRF version (V2) distributed with RSX-11M V4.1 and RSX-11M-PLUS V2.1 should be used with this version of MACRO-11.

- The default for the command line option /[-]SP has been modified from /SP to /-SP. The new default may be modified by the system manager by using the TKB GBLPAT option described in the MACRO-11/RSX Task Build command file.

J.3 Changes—MACRO-11/RT-11 Version Only

J.3.1 V5.5 Update Changes

- In previous versions the error message ?MACRO-F-I/O Error on workfile could occur either because of an actual I/O error or because the workfile was full. A new error message was added, ?MACRO-F-Workfile space exceeded, and the I/O error message reserved for I/O errors only.

You can increase the size of the RT-11 MACRO-11 workfile to a maximum of 400₉ blocks with a customization patch. Refer to the file CUSTOM.TXT on your distribution kit for the address of the location to patch. If your program requires
workfile space greater than 400 blocks, you will get the error message \textit{MACRO-F-Storage limit exceeded (64K)}. This limit cannot be increased.

- In previous versions, if you requested a CREF listing of only error codes (/C:E) but your program had no errors, CREF would hang. CREF now handles this case properly.

### J.3.2 V5.4 Update Changes

- Invalid nonprinting characters in a MACRO-11 source file were not being detected. MACRO-11 now detects invalid nonprinting characters and flags them with an I error.
- CREF did not produce a correct cross-reference listing of a MACRO-11 source with a page length of more than 999 lines. CREF now handles pages of more than 999 lines correctly.

### J.3.3 V5.3 Update Changes

- When running in memory configurations smaller than 8K words, MACRO-11 sometimes trapped with an invalid EMT error, indicating that the input \texttt{MAC} file was not found when in fact the file did exist, or displayed spurious assembly errors. MACRO-11 now runs correctly in memory configurations smaller than 8K words.

### J.3.4 V5.2 Update Changes

There were no RT-11-specific changes made to MACRO-11 V5.2.

### J.3.5 V5.1 Update Changes

There were no RT-11-specific changes made to MACRO-11 V5.1.

### J.3.6 V5.0 Update Changes

- The message:

  \begin{verbatim}
  Errors detected: 0
  \end{verbatim}

  is no longer printed on the console terminal. MACRO-11 prints the message on the terminal only if errors have been detected in the module being assembled.
- If the first character in a MACRO-11/RT-11 command line is a semicolon (\texttt{;}), the line is treated as a comment and is ignored. This change was made to maintain compatibility with the RSX-11 version of MACRO-11.
• RSX–11 style command line switches may be used in addition to the 1-character options:

/M can be represented as /M[LIB]
/E can be represented as /E[NABL]
/D can be represented as /D[SABL]
/P can be represented as /P[ASS]
/L can be represented as /L[IST]
/N can be represented as /N[LIST]

• The default file extension for macro libraries has been changed to .MLB to conform with RSX–11. The RT–11 V5 LIBR program defaults its macro library output to the .MLB extension, also.

• Prior to this release of MACRO–11, if you specified more than one .MLB file on a command line and each file had a definition of the same macro, the first specified macro library would be used for the macro definition if called in the source program. This has been modified to work the same as the RSX–11 macro assembler. The RT–11 macro assembler now scans .MLB files from the last specified file (either in the MACRO–11 command line or by using the .LIBRARY directive) to the first specified file. The assembler then scans the system default macro library, SY:SYSMAC.SML.

• The default for the GBL argument has been changed from .DSABL GBL to .ENABL GBL.
Index

A

ABS
  argument for .ENABL/.DSABL, 6–14
  argument for .PSECT, 6–36
.ABS. default program section name, 6–37
Absolute expressions, 3–14
definition, 3–15
Addition operator, 3–4
Addressing modes, 5–1
  absolute, 5–6
  autodecrement, 5–4
  autodecrement deferred, 5–4
  autoincrement, 5–4
  autoincrement deferred, 5–4
difference between absolute and relative, 5–8
effect of .ENABL AMA, 5–8
  immediate, 5–6
  index, 5–5
  index deferred, 5–5
  register, 5–3
  register deferred, 5–3
  relative, 5–7
  relative deferred, 5–8
  summary, B–2
  table of, 5–2
A error
  .ASCII, 6–20
  .ASCII, 6–21
  .BLKB/.BLKW, 6–32
  .BYTE, 6–17
  .ENABL/.DSABL, 6–14
  .ENDM, 7–3
  for invalid floating point number, 6–28
  .IF, 6–46
  .IF, 6–51
  in bad expression, 3–15
  inconsistent current location counter
    attribute, 3–12
  invalid forward reference defining global,
    3–8
  .IRP, 7–20
  .IRPC, 7–20

A error (cont’d.)
  .LIST/.NLIST, 6–9
  .MACRO, 7–3
  .NARG, 7–13
  .NCHR, 7–15
  .NTYPE, 7–16
  on EMT and TRAP instructions, 5–9
  .PSECT, 6–35, 6–38
  .RAD50, 6–22
  .RADIX, 6–26
  .REPT, 7–22
  .RESTORE, 6–42
  .SAVE, 6–41
  single or double quote character storage,
    6–20
  .TITLE, 6–10
AMA
  argument for .ENABL/.DSABL, 6–14
Ampersand
  AND operator, 3–4
  special character in MACRO–11, 3–2
  special meaning within .IF DF/NDF
    conditional, 6–47
AND operator
  special meaning within .IF DF/NDF
    conditional, 6–47
  summary, 3–4
Angle brackets
  argument delimiter, 3–3
  enclose expressions, 3–14
  required for special .RAD50 values, 6–23
  spaces may increase readability of
    arguments, 7–6
  to insert special values in .ASCII, .ASCIZ
    strings, 6–21
  use in keyword arguments, 7–11
Apostrophe
  see Single quote
Argument delimiters
  angle brackets, 3–3
  circumflex, 3–3
  table of, 3–3
ASCII character set, A–1
ASCII character storage techniques, 6–19
.ASCII directive, 6–20
  changes current location counter, 6–31
  inserting special values with angle brackets, 6–21
summary, 6–1
.ASCIIZ directive, 6–21
  changes current location counter, 6–31
  inserting special values with angle brackets
  see .ASCII directive
summary, 6–1
.AASECT directive, 6–40
  assigns attributes to current location counter, 3–12
  default characteristics, 6–41
  special case of .PSECT, 6–40
summary, 6–1
terminates local symbol block, 3–10
Assembler directives
  see Directives
Asterisk
  in cross-reference table, 8–15
  multiplication operator, 3–4
  special character in MACRO–11, 3–2
At sign
  special character in MACRO–11, 3–2
  used in absolute addressing mode, 5–6
  used in autodecrement deferred mode, 5–4
  used in autoincrement deferred mode, 5–4
  used in index deferred mode, 5–5
  used in register deferred mode, 5–3
  used in relative deferred addressing mode, 5–8

**B**
  for temporary binary radix, 6–27
Backslash
  cannot take forward reference, 7–8
  cannot use with relative symbol, 7–8
  special character in MACRO–11, 3–2
  used to pass numeric argument as symbol, 7–8
B conditional assembly test, 6–46
  only comma valid as separator, 6–50
B error
  odd current location counter, 6–31

BEX
  argument for .LIST/.NLIST, 6–7
BIN
  argument for .LIST/.NLIST, 6–7
Binary operators
  ampersand, 3–4
  asterisk, 3–4
  exclamation mark, 3–4
  minus sign, 3–4
  plus sign, 3–4
  priority, 3–4
  slash, 3–4
  table of, 3–4
  use, 3–4
  used in expressions, 3–14
.BO, default program section name, 6–37
.BLKB directive, 6–32
  changes current location counter, 6–31
  preferred way to reserve space, 3–13
summary, 6–1
.BLKW directive, 6–32
  preferred way to reserve space, 3–13
summary, 6–1
Branch instructions, 5–8
.BYTE directive, 6–17
  changes current location counter, 6–31
  example using concatenated macro argument, 7–13
summary, 6–1

C
  C flag in assembly listing, 4–1
  \C one’s complement operator, 6–29
  represents pressing CTRL/C in command lines, 8–1
/C[R] option
  relationship to .CROSS/.NOCROSS, 6–16
Carriage return
  cannot follow single or double quote, 6–20
CDR
  argument for .ENABL/.DSABL, 6–14
Characters
  invalid, 3–3
Character set
  ASCII, A–1
  DEC multinational, A–1
  DEC multinational chart, A–10
Character set (cont’d.)
definition, 3–1
radix–50, A–8
Circumflex
construct for argument delimiter, 3–3
different meanings, 7–6
passing angle brackets as part of macro argument, 7–6
passing DEC multinational characters, 7–6
special character in MACRO–11, 3–2
universal unary operator, 3–4
CND
argument for .LIST/.NLIST, 6–7
Coding standard, E–1
Colon
invalid as .ASCII string delimiter, 6–21
invalid as .ASCIZ string delimiter, 6–22
invalid as .IDENT string delimiter, 6–12
invalid as .RAD50 string delimiter, 6–23
must precede switch value in RSX command string, 8–5
never as character string delimiter, 6–21
special character in MACRO–11, 3–1
terminates a label, 2–2
COM
argument for .LIST/.NLIST, 6–7
Comma
in macro argument, 7–8
separating character, 3–3
special character in MACRO–11, 3–2
used in operand field, 2–4
Command string examples (IAS), 8–14
Command string format (IAS), 8–12
Comment field
begins with semicolon, 2–4
definition of, 2–4
using .REM, 6–13
valid characters, 2–4
Commercial instruction set (list), C–4
Complement operator (‘C’), 6–29
Complex relocatable expressions, 3–14
definition, 3–16
maximum number of terms, 3–16
CON
argument for .PSECT, 6–37
cannot share data, 6–39
if section ends with odd address, 6–40
Concatenation of arguments example, 7–9
Concatenation of macro arguments, 7–12
Conditional assembly directives, 6–45
.IF, 6–45
.IFF, 6–48
.IFT, 6–48
.IFFT, 6–48
.IIF, 6–50
CREF
see Cross-reference
CRF
argument for .ENABL/.DSABL, 6–14
.CROSS directive, 6–16
relationship to /C[R] or /CROSS option, 6–16
summary, 6–1
/CROSS option
relationship to .CROSS/.NOCROSS, 6–16
Cross-reference listing
sample, H–1
Cross-reference processor
options with IAS/RSX, 8–14
options with RT–11, 9–7
use with IAS/RSX, 8–14
with RT–11, 9–6
Cross-reference table
special symbols, 8–15
.CSECT directive, 6–40
assigns attributes to current location counter, 3–12
default characteristics, 6–41
special case of .PSECT, 6–40
summary, 6–2
terminates local symbol block, 3–10
Current location counter, 3–11
cannot assign value with forward reference, 3–12
change with direct assignment statement, 3–12
changing attributes of, 3–12
effect of odd value, 6–31
list of statements that may leave as odd value, 6–31
using to reserve space, 3–13
Current location counter symbol (period), 3–7, 3–11
assign new value to, 3–12
in program sections, 6–39
D

for temporary decimal radix, 6–27
D
argument for .PSECT, 6–35
Data storage directives, 6–17
DCL command language (RSX), 8–1
DCL command qualifiers (RSX), 8–7
DCL operating procedures (RSX), 8–7
DEC multinational character set
chart, A–10
chart, A–1
use in keyword arguments, 7–11
using circumflex when passing as arguments, 7–6
Delimiters
See argument delimiters
D error
multiply-defined label reference, 2–3
DF conditional assembly test, 6–46
DIF
conditional assembly test, 6–46
effect of .ENABL/.DSABL LCM, 6–46
Direct assignment statements, 3–7
double equal colon sign, 3–7
double equal sign, 3–7
equal colon sign, 3–7
equal sign, 3–7
forward referencing, 3–8
may change current location counter, 6–31
requirements, 3–8
use of space character, 3–8
Directives
conditional assembly, 6–45
data storage, 6–17
file control, 6–51
function, 6–13
indefinite repeat, 7–19
listing control, 6–3
list of obsolete, I–1
macro, 7–1
macro attribute, 7–13, 7–15
overriding permanent definitions with
.MCALL, 7–22
summary, B–3, B–4, B–5, B–6, B–7, C–8
symbol control, 6–43
table of general, 6–1
Division operator, 3–4

Dollar sign
reserved for DIGITAL system symbols, 3–1,
3–5
Double colon
effect when defining a label, 3–6
special character in MACRO–11, 3–1
terminates a label, 2–2
Double equal colon sign
used in direct assignment statements, 3–7
Double equal colon sign
effect when defining a label, 3–6
special character in MACRO–11, 3–1
used in direct assignment statements, 3–7
Double equal sign colon
effect when defining a label, 3–6
special character in MACRO–11, 3–1
Double quote
component of a term, 3–14
for ASCII character storage, 6–19
special character in MACRO–11, 3–2
.DSABL directive, 6–14
summary, 6–2
table of symbolic arguments, 6–14
.DSABL FPT
disables floating point truncation, enables
rounding, 6–29
.DSABL GBL
effect on undefined symbols, 3–6, 3–14
.DSABL LC
effect on valid character set, 3–1
.DSABL LCM
effect on .IF IDN/.IF DIF, 6–46
.DSABL LSB
terminates local symbol block, 3–10
Dummy arguments in macro definition, 7–2
relationship to real arguments, 7–5

E

E error
.END, 6–34
EMT instructions, 5–9
.ENABL AMA
difference between absolute and relative
addressing, 5–8
.ENABL directive, 6–14
summary, 6–2
table of symbolic arguments, 6–14
.ENABL FPT
.ENABL FPT (cont'd.)
    enables floating point truncation, disables rounding, 6–29

.ENABL GBL
    effect on undefined symbols, 3–6

.ENABL LCM
    effect on .IF IDN/.IF DIF, 6–46

.ENABL LSB
    begins local symbol block, 3–10
    may confuse automatic local symbol generation in macro, 7–11

.ENABL MCL
    relationship to .LIBRARY, 6–51

.ENDED directive
    error if outside conditional block, 6–47
    not required with .IIF, 6–50
    summary, 6–2

.END directive, 6–34
    summary, 6–2

.ENDEDM directive, 7–3
    cannot have label, 7–3
    can terminate repeat blocks, 7–4
    summary, 7–1
    terminates macro definition, 7–3

.ENDED directive, 7–21
    summary, 7–1
    terminates .IRP, 7–20
    terminates .IRPC, 7–20
    terminates .REPT, 7–22

EQ conditional assembly test, 6–46

Equal colon sign
    used in direct assignment statements, 3–7

Equal sign
    in cross-reference table, 8–15
    special character in MACRO-11, 3-1
    used as character string delimiter, 6-21
    used in direct assignment statements, 3-7

Equal sign colon
    special character in MACRO-11, 3-1

Error codes

A (cont’d.)
    in bad expression, 3–15
    inconsistent current location counter attribute, 3–12
    invalid floating point number, 6–28
    invalid forward reference defining global, 3–8

    .IRP, 7–20
    .IRPC, 7–20
    .LIST/.NLIST, 6–9
    .MACRO, 7–3
    .NARG, 7–13
    .NCHR, 7–15
    .NTYPE, 7–16
    on EMT and TRAP instructions, 5–9

    .PSECT, 6–35, 6–38
    .RAD50, 6–22
    .REPT, 7–22
    .RESTORE, 6–42
    .SAVE, 6–41
    single or double quote character storage, 6–20

    .TITLE, 6–10

B
    from odd current location counter, 6–31

D
    multiply-defined label reference, 2–3

E
    .END, 6–34

I
    .ASCII, 6–20
    .ASCII, 6–21
    .BLKB/.BLKW, 6–32
    .BYTE, 6–17
    .ENDM, 7–3
    from .ENABL/.DSABL, 6–14
    .IF, 6–46
    .IIF, 6–51

N
    number not in current radix, 3–13

O
    .END, 6–34
    .ENDC, 6–47
    .IF directive nesting, 6–47
    .MCALL, 7–23
    .MDELETE, 7–23
    .MEXIT, 7–4
    .NARG, 7–13
Error codes
O (cont’d.)
  .NTYPE, 7–16
  with .IFF, .IFT, .IFTF, 6–48
P
  .ERROR, 7–18
  inconsistent program section attribute, 3–12
  multiple definition of local symbol, 3–11
  when defining local symbols, 6–15
Q
  .EVEN, 6–31
  for invalid floating point number, 6–28
  in bad expression, 3–15
  invalid syntax, 3–3
  .ODD, 6–32
  .TITLE, 6–10
  too many arguments in macro call, 7–9
R
  invalid redefinition of default register
  symbol, 3–9
T
  number more than 16 bits long, 3–13
U, 3–6
  invalid forward reference, 3–8
  .MCALL, 7–23
  relationship to .ENABL/.DSABL MCL or
  GBL, 6–15
  undefined symbol, 3–14
Z
  flags inconsistent instructions, 5–1
  table of applicable instructions, 5–3
  .ERROR directive, 7–18
  summary, 7–1
Error messages
  system messages for IAS/RSX, 8–18, 8–19, 8–20
  system messages for RT–11, 9–8, 9–9, 9–10, 9–11, 9–12
  .EVEN directive, 6–31
  summary, 6–2
Exclamation mark
  logical inclusive OR operator, 3–4
  special character in MACRO–11, 3–2
  special meaning within .IF DF/NDF
  conditional, 6–47
Expressions, 3–14
  components of a term, 3–14
  definition, 3–14
Expressions (cont’d.)
  evaluation rules, 3–14
  types, 3–14, 3–15
  value of global at assembly, 3–15
External expressions, 3–14
  definition, 3–16
F
  ‘F
  1-word floating point operator, 6–30
File control directives, 6–51
File specifications
  default for RSX–11M, 8–2
  defaults for RT–11, 9–2
  IAS/RSX, 8–17
Floating point numbers
  formats, 6–28
  single-word format, 3–13
  using ‘F operator, 3–13
Floating point processor op codes (list), C–6
  .FLT2 directive, 6–30
  summary, 6–2
  .FLT4 directive, 6–30
  summary, 6–2
Format
  of a MACRO–11 statement, 2–1
  recommended source line format, 2–5
Form feed
  cannot follow single or double quote, 6–20
  effect inside macro definition, 7–4
  generates new page in listing, 6–13
Forward reference
  invalid in current location counter
  assignment, 3–12
Forward referencing
  in direct assignments statements, 3–8
FPT
  argument for .ENABL/.DSABL, 6–14
Function directives, 6–13
G
  G
  flag in assembly listing, 4–1
GBL
  argument for .ENABL/.DSABL, 6–14
  argument for .PSECT, 6–36
  use for data sharing, 6–39
GE conditional assembly test, 6–46
Global expressions

Index–6
Global expressions (cont’d.)

definition, 3–16
Global symbols
creating with direct assignment statements, 3–7
defining, 3–6
function, 3–6
value at assembly time, 3–15
.GLOBL directive, 6–43
defines global user symbols, 3–6
summary, 6–2
GT conditional assembly test, 6–46

.H

HEX
argument for .LIST/.NLIST, 6–8

.I

I
argument for .PSECT, 6–35
IAS
command string examples, 8–14
command string format, 8–12
operating procedures, 8–11, 8–13
system error messages, 8–18, 8–19, 8–20
IAS file specification, 8–17
.IDENT directive, 6–12
summary, 6–2
IDN
conditional assembly test, 6–46
effect of .ENABL/.DSABL LCM, 6–46
.I
error
.ASCII, 6–20
.ASCIIZ, 6–21
invalid character, 3–3
.RAD50, 6–22

.IF
use to detect missing arguments, 7–9
.IF DF
logical AND, OR operators have special meaning, 6–47
.IF DIF
effect of .ENABL/.DSABL LCM, 6–14
.IF directive, 6–45
maximum nesting level, 6–47
summary, 6–2
table of valid condition tests, 6–46
.IFF directive, 6–48
summary, 6–2
.IF IDN
effect of .ENABL/.DSABL LCM, 6–14
.IF NB
use to detect missing arguments, 7–9
.IF NDF
logical AND, OR operators have special meaning, 6–47
.IFT directive, 6–48
summary, 6–2
.IFTF directive, 6–48
summary, 6–2
.IIF directive, 6–48
does not require .ENDC, 6–50
summary, 6–2
.INCLUDE directive, 6–52
default device and file type, 6–52
does implicit .PAGE, 6–52
maximum nesting level, 6–52
restriction on RT-11 systems, 6–52
summary, 6–2
Inclusive OR operator
summary, 3–4
Indefinite repeat directives, 7–19
Invalid characters, 3–3
.IRPC directive, 7–20
restriction using label, 7–20
summary, 7–1
.IRP directive, 7–19
summary, 7–1

.K

Keyword arguments, 7–11
order, 7–11
using DEC multinational character set, 7–11

.L

Label
definition of, 2–2
if same as macro name, 7–5
maximum length, 2–3
not recommended on .MACRO directive, 7–3
on line containing .PSECT, .ASECT, or .CSECT, 6–39
terminated with colon, 2–2
terminated with double colon, 2–2
terminates local symbol block, 3–10
Label (cont’d.)
user label may confuse automatic local
symbol generation in macro, 7–10
valid characters for, 2–3
valid formats, 2–2
LC
argument for .ENABL/.DSABL, 6–14
LCL
argument for .PSECT, 6–36
LCM
argument for .ENABL/.DSABL, 6–14
LE conditional assembly test, 6–46
Left angle bracket
invalid as .ASCII string delimiter, 6–21
invalid as .ASCIZ string delimiter, 6–22
invalid as .IDENT string delimiter, 6–12
invalid as .RAD50 string delimiter, 6–23
special character in MACRO-11, 3–2
Left parenthesis
special character in MACRO-11, 3–2
.LIBRARY directive, 6–51
default device and file type, 6–51
limit on number of files, 6–51
relationship to .ENABL MCL, 6–51
relationship to .MCALL, 6–51, 7–22
restriction on RT-11 systems, 6–51
summary, 6–2
/LIBRARY option
relationship to .MCALL, 7–22
.LIMIT directive, 6–33
summary, 6–2
Line feed
cannot follow single or double quote, 6–20
Linking, discussion, 4–1
.LIST directive, 6–6
overriding with command line options, 6–9
summary, 6–2
table of arguments, 6–7
Listing (sample), H–1
Listing control directives, 6–3
.IDENT, 6–12
overriding with command line options, 6–9
.PAGE, 6–12
.REM, 6–13
.SBTTL, 6–11
table of arguments, 6–7
.TITLE, 6–10
Listing format, 6–3
LOC
argument for .LIST/.NLIST, 6–8
Local symbol block
ways to delimit, 3–10
Local symbols
automatic generation limitations, 7–10
cautions with automatic generation, 7–10
creating automatically in macros, 7–9
definition, 3–10
generate automatically in macro expansion, 3–10
range of valid values, 7–10
range of values, 3–10
uses, 3–10
Location counter
see Current location counter
Logical AND operator
special meaning within .IF DF/NDF conditional, 6–47
summary, 3–4
Logical inclusive OR operator
special meaning within .IF/NDF conditional, 6–47
summary, 3–4
LSB
argument for .ENABL/.DSABL, 6–15
LT conditional assembly test, 6–46
M
Macro argument delimiters
table of, 3–3
.MACRO directive, 7–2
label not recommended, 7–3
summary, 7–1
Macro directives, 7–1
.ENDM, 7–3
.MACRO, 7–2
.MEXIT, 7–4
table, 7–1
Macros, 7–2
argument concatenation example, 7–9
arguments in definitions and calls, 7–5
attribute directives, 7–13, 7–15
begin with .MACRO, 7–2
calling, 7–5
concatenation of arguments, 7–12
creating local symbols automatically, 7–9
defining, 7–2
definition of terms, 7–2
Macros (cont'd.)

dummy arguments, 7-2
formatting of definitions, 7-4
if name is same as user label, 7-5
keyword arguments, 7-11
keywords can override positional
relationship, 7-5
nesting, 7-7
  maximum level, 7-7
number of arguments in calls, 7-9
passing numeric arguments as symbols, 7-8
relationship of dummy and real arguments,
  7-5
separators for arguments, 7-5
special characters in arguments, 7-8
special treatment of DEC multinational
characters in arguments, 7-6
Macro symbols, 3-5
rules, 3-5
.MAIN.
  default of .TITLE, 6-10
MC
  argument for .LIST/.NLIST, 6-8
.MCALL directive, 7-22
   overriding permanent symbol definitions,
      7-22
   relationship to .LIBRARY, 6-51, 7-22
      summary, 7-1
      when required, 7-22
MCL
  argument for .ENABL/.DSABL, 6-15
MCR command language (RSX), 8-1
MCR command string format, 8-4
MD
  argument for .LIST/.NLIST, 6-8
.MDELETE directive, 7-23
     summary, 7-1
ME
  argument for .LIST/.NLIST, 6-8
MEB
  argument for .LIST/.NLIST, 6-8
Memory
  allocation considerations, 6-40
  using efficiently, F-1
M error
  multiply-defined label, 2-3
  redefine permanently-assigned symbol, 3-7
.MEXIT directive, 7-4
  exit .IRP before normal completion, 7-20
.MEXIT directive (cont'd.)
  exit .IRPC before normal completion, 7-21
  exit .REPT before normal completion, 7-22
  summary, 7-1
  terminates macro before completion, 7-4
  valid in repeat blocks, 7-4
Minus sign
  complements switch in RSX command string,
     8-5
special character in MACRO-11, 3-2
  subtraction operator, 3-4
  unary minus operator, 3-4
/ML option
  relationship to .MCALL, 7-22
Modes
  see Addressing modes
Multiplication operator, 3-4

N

NAME
  argument for .PSECT, 6-35
.NARG directive, 7-13
     summary, 7-1
     use to detect missing arguments, 7-9
NB conditional assembly test, 6-46
  only comma valid as separator, 6-50
.NCHR directive, 7-15
     summary, 7-1
NDF conditional assembly test, 6-46
NE conditional assembly test, 6-46
N error
  number not in current radix, 3-13
Nesting macros, 7-7
  maximum level, 7-7
.NLIST directive, 6-6
  overriding with command line options, 6-9
     summary, 6-2
  table of arguments, 6-7
.NLIST TOC
  suppresses table of contents, 6-11
.NOCROSS directive, 6-16
  relationship to /C[R] or /CROSS option,
     6-16
     summary, 6-2
NOSAV
  argument for .PSECT, 6-37
.NTYPE directive, 7-16
     summary, 7-2

Index—9
Null
  cannot follow single or double quote, 6–20
Numbers, 3–13
  changing default radix, 3–13
  components of a term, 3–14
  floating point using .F, 3–13
  initial default is octal, 3–13
  never relocatable, 3–13
Number sign
  in cross-reference table, 8–15
  signifies octal number in RSX command string, 8–5
  special character in MACRO–11, 3–2
  used in absolute addressing mode, 5–6
  used in immediate addressing mode, 5–6

---

O

"O"
  for temporary octal radix, 6–27
Obsolete
  command line options, I–1
  directives, I–1
Octal radix
  initial default for numbers, 3–13
Odd address
  at end of program sections, 6–40
.ODD directive, 6–32
  changes current location counter, 6–31
  summary, 6–2
O error
  .END, 6–34
  .ENDC, 6–47
  .IF directive nesting, 6–47
  .IFF, .IFT, .IFTF, 6–48
  .MCALL, 7–23
  .MDELETE, 7–23
  .MEXIT, 7–4
  .NARG, 7–13
  .NTYPE, 7–16
Operand field
  definition of, 2–4
  valid formats, 2–4
  valid terminators, 2–4
Operating procedures (RSTS/RT–11), 9–1
Operator field
  definition of, 2–3
  implicit .WORD if blank, 6–18
  valid formats, 2–3
  valid terminators, 2–3
Options
  list of obsolete, I–1
OR operator
  special meaning within .IF/NDF conditional, 6–47
  summary, 3–4
OVR
  argument for .PSECT, 6–37
  if section ends with odd address, 6–40
  use for data sharing, 6–39

P

P1
  conditional assembly test, 6–46
P2
  conditional assembly test, 6–46
.PACKED directive, 6–24
  changes current location counter, 6–31
  summary, 6–2
.PAGE directive, 6–12
  implicit with .INCLUDE, 6–52
  inside macro definition, 7–4
  summary, 6–2
Page eject
  if .PAGE directive encountered, 6–12
  if form feed encountered, 6–13
  if more than 58 lines, 6–13
  if new source file, 6–13
  operation of form feed inside macro definition, 7–4
Pass 1, what happens, 1–1
Pass 2, what happens, 1–2
PC
  see Program counter
Percent sign
  defines register symbols, 3–8, 3–9
  special character in MACRO–11, 3–1
Period
  component of a term, 3–14
  current location counter symbol, 3–7, 3–11, 3–14
  assign new value to, 3–12
  makes expression relocatable, 3–15
  reserved for DIGITAL system symbols, 3–1, 3–5
  signified decimal number in RSX command string, 8–5
  special character in MACRO–11, 3–2

Index–10
Permanent symbols, 3–5
  overriding with .MCALL, 7–22
Permanent symbol table
  list, C–1
P error
  .ERROR, 7–18
  inconsistent program section attribute, 3–12
  multiple definition of local symbol, 3–11
  when defining local symbols, 6–15
PIC
  see Position-independent code
Plus sign
  addition operator, 3–4
  default switch value in RSX command string, 8–5
  special character in MACRO–11, 3–2
  unary plus operator, 3–4
PNC
  argument for .ENABL/.DSABL, 6–15
Position-independent code, G–1
  .PRINT directive, 7–18
    summary, 7–2
Priority of binary operators, 3–4
Processor differences, table of, 5–3
Program counter operation, 5–1
Program section directives, 6–34
Program sections
  context information maintained by MACRO–11, 6–38
  creating, 6–38
  default characteristics, 6–41
  default names, 6–37
  effect of ending with odd address, 6–40
  maximum number, 6–37
  memory allocation considerations, 6–40
  separating code and data, 6–40
  sharing code or data, 6–39
P-sect
  see Program sections
.PSECT directive, 6–35
  assigns attributes to current location counter, 3–12
  default characteristics, 6–41
  list of symbolic arguments, 6–35
  may confuse automatic local symbol generation in macro, 7–11
  summary, 6–2
  terminates local symbol block, 3–10

Q
  Q error
    .EVEN, 6–31
    in bad expression, 3–15
    invalid floating point number, 6–28
    invalid syntax, 3–3
    .ODD, 6–32
    .TITLE, 6–10
    too many arguments in macro call, 7–9
Question mark
  used to generate local symbols, 7–10

R
  .RAD50 directive, 6–22
    See also Radix–50 character set
    character equivalents, 6–23
    formula, 6–23
    inserting special values with angle brackets, 6–23
    summary, 6–2
    valid characters, 6–22
  Radix–50 character set, A–8
    See also .RAD50 directive
  Radix–50 character terminates operator field, 2–3
  Radix–50 storage
    See also .RAD50 directive
    temporary with ‘R, 6–24
  Radix control, 6–26
    changing default, 3–13
    list of temporary operators, 6–27
    temporary, 3–13
    when to use temporary, 6–26
  .RADIX directive, 6–26
    discussed, 3–13
    restriction if .RADIX 16, 6–26
    summary, 6–3
REG
  argument for .ENABL/.DSABL, 6–15
  Register symbols, 3–8
    default definitions, 3–8
    requirements, 3–9
REL
  argument for .PSECT, 6–36
  Release notes, J–1
  Relocatable expressions, 3–14
    definition, 3–15
  Relocation, discussion, 4–1

Index–11
Relocation and linking, 4–1
.REM directive, 6–13
  summary, 6–3
.REP directive, 7–21
  restriction using label, 7–20
  summary, 7–2
Requirements
  for direct assignment statements, 3–8
  for register symbols, 3–9
R error
  invalid redefinition of default register
  symbol, 3–9
  invalid register symbol, 3–9
.RESTORE directive, 6–41
  assigns attributes to current location counter,
  3–12
  summary, 6–3
  terminates local symbol block, 3–10
Right angle bracket
  special character in MACRO–11, 3–2
Right parenthesis
  special character in MACRO–11, 3–2
RO
  argument for .PSECT, 6–35
  R operator, 6–24
RSTS operating procedures, 9–1
RSX–11M
  default file specifications, 8–2
  file specification, 8–17
  file specification switches, 8–6
  operating procedures, 8–1
  system error messages, 8–18, 8–19, 8–20
  under RSTS, 9–1
RT–11
  CSI command line format, 9–2
  CSI command line options, 9–4
  DCL command line format, 9–5
  default file specifications, 9–2
  operating procedures, 9–2
  system error messages, 9–8, 9–9, 9–10, 9–11,
  9–12
  under RSTS, 9–1
Rubout
  cannot follow single or double quote, 6–20
RW
  argument for .PSECT, 6–35
SAV (cont’d.)
  argument for .PSECT, 6–37
  .SAVE directive, 6–41
    maximum number, 6–41
    summary, 6–3
  SBTTL directive, 6–11
    generates table of contents, 6–11
    summary, 6–3
    text appears in listing heading, 6–3
Search order of symbol tables, 3–6
Semicolon
  begins comment field, 2–4
  in macro argument, 7–8
  special character in MACRO–11, 3–2
  used as character string delimiter, 6–21
Separating characters
  comma, 3–3
  space, 3–3
  table of, 3–3
SEQ
  argument for .LIST/.NLIST, 6–8
Sharing code or data, 6–39
Single quote
  component of a term, 3–14
  example of concatenation, 7–9
  flag in assembly listing, 4–1
  for ASCII character storage, 6–19
  special character in MACRO–11, 3–2
  use to concatenate macro arguments, 7–12
Slash
  division operator, 3–4
  special character in MACRO–11, 3–2
Space
  delimiter in expressions, 3–15
  in direct assignment statements, 3–8
  in macro argument, 7–8
  separating character, 3–3
  special character in MACRO–11, 3–2
  terminates operator field, 2–3
  used in operand field, 2–4
  valid in angle bracket arguments, 7–6
Special characters
  ampersand, 3–2
  angle brackets, 3–14, 6–21, 6–23
  asterisk, 3–2
  at sign, 3–2, 5–3, 5–4, 5–5, 5–6, 5–8
  backslash, 3–2
  carriage return, 6–20
  circumflex, 3–2, 7–6
  colon, 3–1, 6–12, 6–21, 6–22, 6–23

Index–12
Special characters

colon (cont'd.)

label terminator, 2–2
coma, 3–2
used in operand field, 2–4
dollar sign
reserved for DIGITAL system symbols, 2–3, 3–1, 3–5
double colon, 3–1, 3–6
label terminator, 2–2
double equal sign, 3–1, 3–6
double equal sign colon, 3–1, 3–6
double quote, 3–2, 3–14, 6–19
equal sign, 3–1, 6–21
equal sign colon, 3–1
exclamation mark, 3–2
form feed, 6–13, 6–20
in macro arguments, 7–8
left angle bracket, 3–2, 6–12, 6–21, 6–22, 6–23
left parenthesis, 3–2
line feed, 6–20
minus sign, 3–2
null, 6–20
number or pound sign, 3–2
number sign, 5–6
percent sign, 3–1
defines register symbols, 3–8, 3–9
period, 3–2, 3–15
assign new value to current location counter, 3–12
current location counter, 3–14
current location counter symbol, 3–7
reserved for DIGITAL system symbols, 2–3, 3–1, 3–5
symbol for current location counter, 3–11
plus sign, 3–2
right angle bracket, 3–2
right parenthesis, 3–2
rubout, 6–20
semicolon, 3–2, 6–21
begins comment field, 2–4
single quote, 3–2, 3–14, 4–1, 6–19
slash, 3–2
space, 3–2, 3–15
in direct assignment statements, 3–8
terminates operator field, 2–3
used in operand field, 2–4

Special characters

tab, 3–2
terminates operator field, 2–3
used in operand field, 2–4
table, B–1
table of, 3–1
SRC
argument for .LIST/.NLIST, 6–8
Standard for coding programs, E–1
Standards and conventions, 2–1
Starting address of program specify with .END directive, 6–34
Statement format, 2–1
Subtraction operator, 3–4
SYM
argument for .LIST/.NLIST, 6–8
Symbol control directives, 6–43
Symbols
assumed value of undefined, 3–14
components of a term, 3–14
macro, 3–5
rules, 3–5
order of symbol table searches, 3–6
permanent, 3–5
types, 3–5
user-defined, 3–5
rules, 3–5
Symbol table
list of permanent, C–1

T

Tab
in macro argument, 7–8
special character in MACRO–11, 3–2
terminates operator field, 2–3
used in operand field, 2–4
Table of contents
generated by .SBTTL, 6–11
Terms
components of expressions, 3–14
definition and possible elements, 3–14
T error
number more than 16 bits long, 3–13
.TITLE directive, 6–10
defaults to .MAIN, 6–10
result if more than one, 6–10
summary, 6–3
text appears in listing heading, 6–3

Index–13
TOC
argument for .LIST/.NLIST, 6-8
TRAP instructions, 5-9
TTM
argument for .LIST/.NLIST, 6-8

U
invalid forward reference, 3-8
.relationship to .ENABL/.DSABL GBL, 6-14
.argument to .ENABL/.DSABL MCL or GBL, 6-15
.undefined symbol, 3-6, 3-14

Unary operators
'B, 6-27
'C, 6-29
circumflex, 3-4
components of a term, 3-14
'D, 6-27
double quote, 6-19
'F, 6-30
minus sign, 3-4
'O, 6-27
plus sign, 3-4
single quote, 6-19
table of, 3-4
treatment of multiple, 3-14
use, 3-3, 3-4
'X, 6-27

Undeclared symbols
assumed value, 3-14
User-defined symbols, 3-5
rules, 3-5

V
Virtual memory
allocating, F-1

W
.WEAK directive, 6-44
.summary, 6-3
.supported only in RT-11, 6-45
.WORD directive, 6-18
.compared with .FLT2, .FLT4, 6-30
.implicit, 2-1, 2-4
.implicit if blank operator field, 6-18
.operation described, 3-11
## HOW TO ORDER

**ADDITIONAL DOCUMENTATION**

<table>
<thead>
<tr>
<th>From</th>
<th>Call</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alaska, Hawaii, or New Hampshire</td>
<td>603–884–6660</td>
<td>Digital Equipment Corporation P.O. Box CS2008</td>
</tr>
<tr>
<td>Rest of U.S.A. and Puerto Rico*</td>
<td>800–258–1710</td>
<td>Nashua, NH 03061</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*Prepaid orders from Puerto Rico must be placed with DIGITAL's local subsidiary (809–754–7575)</td>
</tr>
<tr>
<td>Canada</td>
<td>800–267–6219</td>
<td>Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6</td>
</tr>
<tr>
<td></td>
<td>(for software</td>
<td></td>
</tr>
<tr>
<td></td>
<td>documentation)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>613–592–5111</td>
<td>Attn: Direct Order desk</td>
</tr>
<tr>
<td></td>
<td>(for hardware</td>
<td></td>
</tr>
<tr>
<td></td>
<td>documentation)</td>
<td></td>
</tr>
<tr>
<td>Internal orders</td>
<td></td>
<td>Software Distribution Center (SDC)</td>
</tr>
<tr>
<td>(for software</td>
<td></td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td>documentation)</td>
<td></td>
<td>Westminster, MA 01473</td>
</tr>
<tr>
<td>Internal orders</td>
<td>617–234–4323</td>
<td>Publishing &amp; Circulation Serv. (P&amp;CS)</td>
</tr>
<tr>
<td>(for hardware</td>
<td></td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td>documentation)</td>
<td></td>
<td>Northboro, MA 01532</td>
</tr>
</tbody>
</table>
Your comments and suggestions will help us improve the quality of our future documentation. Please note that this form is for comments on documentation only.

I rate this manual’s:

<table>
<thead>
<tr>
<th></th>
<th>Excellent</th>
<th>Good</th>
<th>Fair</th>
<th>Poor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (product works as described)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completeness (enough information)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clarity (easy to understand)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization (structure of subject matter)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Figures (useful)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Examples (useful)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index (ability to find topic)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page layout (easy to find information)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What I like best about this manual: ____________________________________________________________

What I like least about this manual: ____________________________________________________________

I found the following errors in this manual:

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

My additional comments or suggestions for improving this manual: ____________________________________________
__________________________________________________________________________
__________________________________________________________________________

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

☐ Administrative Support  ☐ Scientist/Engineer
☐ Computer Operator       ☐ Software Support
☐ Educator/Trainer        ☐ System Manager
☐ Programmer/Analyst      ☐ Other (please specify) ____________________________
☐ Sales                   ____________________________

Name/Title ___________________________________________ Dept. ______________________
Company ___________________________________________ Date ______________________
Mailing Address ___________________________________ Phone ______________________

10/97
BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
CORPORATE USER PUBLICATIONS
MLO5-5/E45
146 MAIN STREET
MAYNARD, MA 01754-2571