MACRO–64 Assembler for OpenVMS AXP (TM) Systems Release Notes

June 1993

This document provides information about the latest release of the MACRO–64 Assembler for OpenVMS AXP Systems. It includes information on features, changes, and resolved problems.

Version T1.1 Build Number: 029

DIGITAL-INTERNAL FIELD TEST

Revision/Update Information:

Software Version: MACRO–64 Assembler for OpenVMS AXP Systems Version 1.1

Date: June 7, 1993

Digital Equipment Corporation
Maynard, Massachusetts

1993

© Digital Equipment Corporation 1993. The following are trademarks of Digital Equipment Corporation:

OpenVMS	AXP	Alpha AXP	
CDA	DEQNA	MicroVAX	VAX RMS
DDIF	Desktop–VMS	PrintServer 40	VAXserver
DEC	DIGITAL	Q–bus	VAXstation
DECdtm	GIGI	ReGIS	VMS
DECnet	HSC	ULTRIX	VT
DECUS	LiveLink	UNIBUS	XUI
DECwindows	LN03	VAX	
DECwriter	MASSBUS	VAXcluster	

Ada® is a registered trademark of the U.S. Government, Ada Joint Program Office.

Digital Equipment Corporation
Proprietary Information

This information is proprietary to Digital and is subject to the terms and conditions of your non-disclosure agreement.

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

Preface v

1 Programming Notes

1.1 Cross-Development Tools 1–1
1.2 Assembly-Time Using Calling-Standard Macros 1–1
1.3 STARLET.MLB 1–1
1.4 Example Programs 1–2
1.5 Using the Language Sensitive Editor 1–2
1.6 Integer Division 1–2
1.7 /MACHINE_CODE and Other AXP Languages 1–2

2 New and Modified Features

2.1 Register Symbols 2–1
2.2 Assembly-Time Improvement 2–4

3 Resolved Problems

3.1 .IDENT VAX MACRO Compatibility 3–1
3.2 Bad Error Location Indicator 3–1
3.3 Library-Macro Name Casing 3–1
3.4 Forward References to .RESTOREd Local Blocks 3–1
3.5 Module Name not Upcased with .TITLE 3–2
3.6 LSE Performance 3–2
3.7 Code address in MIX psect 3–2
3.8 Occasional ACCVIO with forward reference to temporary label 3–3

4 Known Problems and Restrictions

4.1 .BLKx Directives Do Not Appear in Machine-Code Listing 4–1
4.2 Instructions for MIX Psects 4–1
4.3 Debugger Does Not Scroll Repeat Blocks 4–1
4.4 Static .BYTE and .WORD Initializations with Shared Images 4–1
4.5 Debugger Information for Local Data Labels 4–1
4.6 Assembler gives no clues when out of memory 4–2
4.7 HELP MACRO/ALPHA_AXP doesn't work 4–2
4.8 HELP MACRO/ALPHA broken on OpenVMS V1.5-FT3 and V1.5-FT4 . . . 4–2
4.9 MACZERO diagnostic anomalies 4–2
4.10 TRAPB instruction encoding 4–2
4.11 Superfluous argument/scratch register conflict 4–2
4.12 No diagnostics in /PREPROCESS_ONLY mode 4–3

5 Known Documentation Errors

5.1 No reference manual examples in Section 3.6 in Bookreader format 5–1
5.2 Reference manual section B.7.1.1 errors 5–1
5.3 Reference manual section B.7.5 errors 5–1
5.4 $CALL SET_ARG_INFO=FALSE doesn't always mean you must set R25 5–1
5.5 Only BOOKBROWSER PAK works with MACRO-64 bookreader files 5–1
5.6 Listing Format is not documented 5–2
5.7 Table 6-1 is revised 5–2

Preface

Introduction

The MACRO–64 Assembler for OpenVMS AXP Systems provides a macro assembly language for the native Alpha AXP instruction set. MACRO–64 is available as a native assembler on the OpenVMS AXP Operating System. MACRO–64 is also available as a cross assembler hosted on the OpenVMS VAX operating system internally to Digital software development groups, and for a limited time to customers as part of an AXP cross-tool kit offered by Digital Services.

Reporting Problems

Please report all problems in the MACRO-64 notes conference MLTVAX::ASM$NOTES:ASM.NOTE.

Intended Audience

This manual is intended for system managers and programmers who need to be informed about features, changes, and resolved problems.

Document Structure

This manual is divided into the following chapters:

- Chapter 1, *Programming Notes*, includes some notes for programming in MACRO-64.
- Chapter 2, *New Features*, describes new features or implementations since V1.0-213, the production V1.0 release.
- Chapter 3, *Resolved Problems*, describes problems resolved since V1.0-213.
- Chapter 4, *Known Problems and Restrictions*, describes problems and restrictions that the development team is aware of, but has not resolved at the time of this release.
- Chapter 5, *Known Documentation Errors*, describes substantive documentation errors that the development and documentation teams are aware of, but have not corrected at the time of this release.

Associated Documents

The following documents are associated with MACRO–64:

- *MACRO–64 Assembler for OpenVMS AXP Systems Reference Manual*
- *MACRO–64 Assembler for OpenVMS AXP Systems Installation Guide*

Related Documents

The following documents are related to MACRO–64 programming:

1. *OpenVMS Calling Standard*
2. *Alpha Architecture Reference Manual*
3. *Alpha Architecture Handbook*
4. *Alpha System Reference Manual*

Conventions

The following conventions are used in these release notes:

Ctrl/x	A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
Return	In examples, a key name is shown enclosed in a box to indicate that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	In examples, a horizontal ellipsis indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
[ ]	In format descriptions, brackets indicate that whatever is enclosed within the brackets is optional; you can select none, one, or all of the choices. Brackets are not, however, optional in the syntax of a directory name in a file specification.
{ }	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in online versions of the book.
UPPERCASE TEXT	Uppercase letters indicate that you must enter a command (for example, enter OPEN/READ), or they indicate the name of a routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege.
-	Hyphens in coding examples indicate that additional arguments to the request are provided on the line that follows.
numbers	Unless otherwise noted, all numbers in the text are assumed to be decimal. Non-decimal radices—binary, octal, or hexadecimal—are explicitly indicated.

1
Programming Notes

This chapter contains some notes for programming in MACRO-64.

1.1 Cross-Development Tools

Prior to installing the cross-assembler on an OpenVMS VAX system, the VMS Alpha Cross-Development Tools must be installed on your system.

1.2 Assembly-Time Using Calling-Standard Macros

The calling standard macros provided with MACRO-64 in MACRO64.MLB are designed to make programming in conformance with the OpenVMS Calling Standard significantly easier. See Chapter 6 of the MACRO-64 Reference Manual for a description of these macros.

While in general these macros generate a minimum of instruction and data directive statements, the job these macros perform is complex and can involve several thousand lines of macro expansion assembly-time statements. When you use these macros you may notice increased assembly time and increased memory usage at assembly time. Nonetheless, use of the calling-standard macros should significantly speed your program development effort.

1.3 STARLET.MLB

While a large number of macros in STARLET.MLB work correctly with MACRO-64, those macros in STARLET.MLB that generate instruction statements are designed exclusively for use with the macro migration compiler, the MACRO-32 Compiler for OpenVMS AXP. These macros generate VAX instructions and are not suitable for use with MACRO-64. Therefore, MACRO-64 defines corresponding macros by the same names in MACRO64.MLB to prevent a series of potentially misleading diagnostics should you inadvertently use one of the VAX-instruction generating macros. Since MACRO-64 searches MACRO64.MLB before STARLET.MLB, the macro definitions in MACRO64.MLB occlude those of the same names in STARLET.MLB. If you attempt to use one of these macros with MACRO-64, the assembler displays an error message similar to the following:

```
Macro XXXXX is not supported for use with MACRO-64
```

Note that you can still perform the operations with MACRO-64 that are performed by these STARLET macros. For instance, while the following STARLET macro invocation does not work with MACRO-64:

```
$ALLOC_S DEVNAM=OUTPUT_DEV_DESCR
```

you can use MACRO-64's $CALL macro instead as follows:

```
$CALL SYS$ALLOC, ARGS=<OUTPUT_DEV_DESCR/A>
```

That is, you can call system services from MACRO-64 as you would from other languages. See Chapter 6 of the MACRO-64 Reference Manual for information on the $CALL macro. Refer to the OpenVMS Systems Services manual for information on calling OpenVMS system services. Refer to SYS$EXAMPLES:MACRO64$WHAMI.M64 after installing MACRO-64 for an example of how to call system services from MACRO-64.

1.4 Example Programs

To assist in MACRO-64 programming, several example programs are placed in SYS$EXAMPLES:MACRO64$*.M64 on your system during installation. In order of increasing complexity, the example programs that are placed in the SYS$EXAMPLES directory are:

- MACRO64$HELLO.M64—A simple "Hello World" program that demonstrates the calling standard macros.
- MACRO64$WHAMI.M64—A program that displays the WHAMI internal processor register and demonstrates making system service calls from MACRO-64.
- MACRO64$PI.M64—A program that computes the mathematical constant PI to a specified number of digits, and demonstrates general MACRO-64 programming concepts and optimization techniques.

1.5 Using the Language Sensitive Editor

If you use the Language Sensitive Editor (LSE) with MACRO-64, and you wish LSE on the OpenVMS AXP Operating System to replicate the behavior of LSE on OpenVMS VAX, you should issue the LSE command: "SET COMMAND LANGUAGE VMS".

1.6 Integer Division

While the Alpha AXP Architecture does not provide an instruction for integer division, you can use the OTS$DIV_L and OTS$DIV_I routines to perform quadword and longword integer division, respectively. For an example of how to use these routines, see the MACRO64$PI.M64 example program in the SYS$EXAMPLES directory.

1.7 /MACHINE_CODE and Other AXP Languages

Many of the other OpenVMS AXP languages offer a /MACHINE_CODE command-line qualifier that shows the assembly language generated for your source program in the listing file. You can often use this feature to learn about AXP assembly language and optimization techniques. You can also use MACRO-64's /MACHINE_CODE command-line qualifier in conjunction with the /LIST to view the results of MACRO-64's optimization and automatic alignment features.

2
New and Modified Features

This chapter describes new features and feature changes that have been added since the MACRO-64 X1.1-016 release.

2.1 Register Symbols

MACRO-64 T1.1-029 adds the capability of user-defined register symbols.

MACRO-64 V1.0 defines the following permanent, registers symbols:

R0..R31—integer registers 0 thru 31
SP—integer register 30
FP—integer register 29
F0..F31—floating-point registers 0 thru 31

When MACRO-64 V1.0 operates in /NAMES=AS_IS mode, all of the above register symbols are defined in all uppercase and all lowercase.

With MACRO-64 V1.0, these register symbols are permanently reserved – you cannot use register symbols where you would use a MACRO-64 identifier. Furthermore, MACRO-64 V1.0 does not allow you to define your own register symbols.

MACRO-64 T1.1-029 defines the same register symbols as V1.0. However, while these register symbols remain reserved, they are not *permanently* reserved in that you can delete the definition of any of these predefined register symbols. Moreover, MACRO-64 T1.1-029 allows you to define your own register symbols.

To define a register symbol with MACRO-64 T1.1-029, use either the .DEFINE_IREG or .DEFINE_FREG directive for integer or floating-point registers, respectively. The syntax for these directives is as follows:

```
.DEFINE_IREG   new_regsym regnum
.DEFINE_FREG   new_regsym regnum
```

where new_regsym is either a MACRO-64 identifier or a currently-defined register symbol that is either predefined by MACRO-64 or previously defined by you, and regnum is either an assembly-time expression or a currently-defined register symbol.

If you specify an identifier as the first argument to .DEFINE_IREG or .DEFINE_FREG, that identifier becomes a register symbol. Thereafter, it cannot be used as a MACRO-64 identifier. Specifically, the user-defined register symbol is only allowed where a register is allowed. In this sense, the user-defined register symbol is reserved until the end of the assembly unit, or until you delete its definition (see .UNDEFINE_REG below), which ever occurs first. If you specify a register symbol as the first argument to either .DEFINE_IREG or .DEFINE_FREG, the register number you specify with the second argument must be the

same as the value already assigned to that register symbol, or the assembler issues a warning that you are redefining a register symbol. If you wish to redefine a register symbol without a warning, first delete the current definition, then assign the new register number to the register symbol.

The second argument to .DEFINE_IREG and .DEFINE_FREG can be either an integer expression or a currently-defined register symbol. An integer expression indicates the register number to assign to register symbol. The expression can contain no forward or external references and must be in the range of zero to 31. Alternatively, you can define a register symbol in terms of another, currently-defined register symbol. A currently-defined register symbol is a predefined register symbol or a register symbol that you have previously defined. In this case, the new register symbol you specify with the first argument receives the current value of the register symbol you specify with the second argument. However, you cannot define an integer register in terms of a floating-point register, or vice versa.

Examples:

```
.DEFINE_IREG A0     16      ; A0 is integer register 16
.DEFINE_IREG A1     R17     ; A1 is integer register 17,
                            ;  defined in terms of the
                            ;  predefined R17 register symbol
.DEFINE_IREG PTR    A0      ; PTR is integer register 16,
                            ;  defined in terms of the
                            ;  previously-defined A0 register
                            ;  symbol
.DEFINE_FREG $F0    0       ; $F0 is floating register 0
.DEFINE_FREG $F1    F1      ; $F1 is floating register 1,
                            ;  defined in terms of the
                            ;  predefined F1 register symbol
.DEFINE_FREG RADIUS $F1     ; RADIUS is floating register 1,
                            ;  defined in terms of the
                            ;  previously defined $F1
                            ;  register symbol
.DEFINE_IREG X1     R5      ; X1 is integer register 5,
                            ;  defined in terms of the
                            ;  predefined R5 register symbol
.DEFINE_IREG X1     5       ; 2nd definition is the same
                            ;  value, so no diagnostic
                            ;  results
.DEFINE_IREG X1     7       ; Warning: redefinition with a
                            ;  different value
.DEFINE_FREG X2     F5      ; Error: cannot define an integer
                            ;  register in terms of a
                            ;  floating register

LDQ         R1, (PTR)       ; LDQ R1, (R16)
LDG         $F1, (A1)       ; LDG F1, (R17)
```

You can delete a register symbol definition with the .UNDEFINE_REG directive. The syntax for the .UNDEFINE_REG directive is as follows:

```
.UNDEFINE_REG regsym
```

where regsym is either a currently-defined register symbol or a MACRO-64 identifier. If you specify a MACRO-64 identifier that is not currently defined as a register symbol, the .UNDEFINE_REG directive has no effect. If you specify a register symbol that is currently defined, the .UNDEFINED_REG directive causes it to no longer be a register symbol. Starting with the statement that

follows the .UNDEFINE_REG directive, the symbol can be used as MACRO-64 identifier.

Examples:

```
.DEFINE_IREG  X1    R5      ; X1 is integer register 5
.UNDEFINE_REG X1            ; X1 is an identifier again
.DEFINE_IREG  X1    7       ; X1 is integer register 7 - no
                            ;  redefinition and no warning

$ROUTINE F0                 ; Error: F0 is a register and
                            ;  cannot be used as an
                            ;  identifier
.UNDEFINE_REG F0            ; F0 is no longer a register
$ROUTINE F0                 ; Ok now
```

When you use any of the OpenVMS Calling Standard macros described in Chapter 6 of the MACRO-64 Reference Manual, the following register symbols are defined for you:

```
.define_ireg $IA0   R16     ; Integer argument 0
.define_ireg $IA1   R17     ; Integer argument 1
.define_ireg $IA2   R18     ; Integer argument 2
.define_ireg $IA3   R19     ; Integer argument 3
.define_ireg $IA4   R20     ; Integer argument 4
.define_ireg $IA5   R21     ; Integer argument 5
.define_freg $FA0   F16     ; Floating-point argument 0
.define_freg $FA1   F17     ; Floating-point argument 1
.define_freg $FA2   F18     ; Floating-point argument 2
.define_freg $FA3   F19     ; Floating-point argument 3
.define_freg $FA4   F20     ; Floating-point argument 4
.define_freg $FA5   F21     ; Floating-point argument 5
.define_ireg $AI    R25     ; Argument-information register
.define_ireg $RA    R26     ; Return-address register
.define_ireg $PV    R27     ; Procedure-value register
.define_ireg $FP    R29     ; Frame pointer
.define_ireg $SP    R30     ; Stack pointer
```

Note that SP and FP remain predefined by MACRO-64 whether you use the OpenVMS Calling Standard macros or not. $SP and $FP are defined by the OpenVMS Calling Standard macros for consistency with the other register definitions that correspond to OpenVMS Calling Standard register conventions.

The SYS$EXAMPLES:MACRO64$PI.M64 has been enhanced to show the use of user-defined register symbols.

The macros described in Chapter 6 of the MACRO-64 Reference Manual accept user-defined register symbols in lieu of the predefined register symbols. In writing your own macros, you may wish to allow for user-defined register symbols as well. To help you to do so, the %type( ) lexical operator with MACRO-64 T1.0-029 returns MACRO64$TYPE_GENREG and MACRO64$TYPE_FLTREG for user-defined integer and floating-point registers, respectively. In addition, MACRO-64 T1.1-029 supplies you with two, new lexical operators: %ireg( ) and %freg( ). The new lexical operators are described below.

%IREG

Lexical operator for obtaining the integer register number associated with a symbol.

Format

%IREG(symbol)

Arguments

symbol—The single argument is of type string. It specifies a symbol which may or may not be currently defined as an integer-register symbol.

Description—If the specified symbol is currently defined as an integer-register symbol, %IREG returns the decimal number of the integer register. Otherwise, %IREG returns 32.

Examples

```
; Is SRC_REG the same as SP?
.IF EQ, <%IREG(SRC_REG)>, <%IREG(SP)>
```

If SRC_REG has been defined as integer register R16, the statements appear as follows after lexical processing:

```
; Is SRC_REG the same as SP?
.IF EQ, <16>, <30>
```

%FREG

Lexical operator for obtaining the floating-point register number associated with a symbol.

Format

%FREG(symbol)

Arguments

symbol—The single argument is of type string. It specifies a symbol which may or may not be currently defined as an floating-point-register symbol.

Description—If the specified symbol is currently defined as a floating-point-register symbol, %FREG returns the decimal number of the floating-point register. Otherwise, %FREG returns 32.

Examples

```
; Is TARG_REG the same as F31?
.IF EQ, <%FREG(TARG_REG)>, <%FREG(31)>
```

If TARG_REG has been defined as floating-point register F5, the statements appear as follows after lexical processing:

```
; Is TARG_REG the same as F31?
.IF EQ, <5>, <31>
```

2.2 Assembly-Time Improvement

Elapsed assembly time with MACRO-64 T1.1-029 has been reduced over that of MACRO-64 V1.0-213. The amount of the reduction will vary depending on your source programs. The MACRO-64 development team has observed an approximate 15 percent reduction in elapsed assembly time.

3
Resolved Problems

This chapter lists problems corrected since the V1.0-213 release.

3.1 .IDENT VAX MACRO Compatibility

The .IDENT directive has been modified to accept VAX MACRO syntax in addition to MACRO-64 syntax.

3.2 Bad Error Location Indicator

Under certain circumstances, MACRO-64 V1.0-213 generates a diagnostic location indicator with an erroneous column for diagnostic messages that are cited against macro expansion lines. The symptom is that the pointer to the token of interest in the listing file or diagnostics file is sometimes wildly wrong. However, the correct line is always cited.

This problem has been corrected in MACRO-64 T1.1-029.

3.3 Library-Macro Name Casing

If you replace the supplied copies of MACRO64.MLB and STARLET.MLB and you run MACRO-64 in case sensitive mode (/NAMES=AS_IS), it is possible to read a macro from the macro library that has different name casing from the macro name you specify. This situation is inherent in the way the librarian and the system routines deal with macro-name case sensitivity. When this happens, MACRO-64 V1.0-213 *usually* detects the problem and issues a MACCASEMATCH diagnostic. However, under certain unusual circumstances, MACRO-64 V1.0-213 MACRO-64 fails to detect and diagnose this problem.

MACRO-64 T1.1-029 has been corrected to detect and diagnose all situations where the case of the name of the macro read in from a macro library does not match the case of the specified macro name when MACRO-64 is operating in /NAMES=AS_IS mode.

3.4 Forward References to .RESTOREd Local Blocks

MACRO-64 V1.0-213 does not correctly process a forward reference to a temporary label that is defined in a section of a local block that occurs after the local block is restored with a .RESTORE directive. In the following example, MACRO-64 V1.0-213 fails to correctly resolve the forward reference to the definition of the 10$ label.

```
        BR   10$
        .SAVE LOCAL_BLOCK
        .RESTORE
10$:
```

MACRO-64 V1.0-213 incorrectly issues a diagnostic for the above example indicating that the label 10$ is undefined.

MACRO-64 T1.1-029 corrects this problem.

3.5 Module Name not Upcased with .TITLE

MACRO-64 V1.0-213 defines the module name in the object module using the same alphabetic case you use to specify the module name with the .TITLE directive. MACRO-64 should define the module name in all upper case in the object module, regardless of the alphabetic case you use with the .TITLE directive.

MACRO-64 T1.1-029 corrects this problem.

3.6 LSE Performance

If you install MACRO-64 V1.0-213 on your system and you have previously installed the Language Sensitive Editor (LSE), you may experience slow LSE startup in all language environments due to duplicate keywords stored in the LSE system environment file after installing MACRO-64.

This problem has been corrected in MACRO-64 T1.1-029. However, installing MACRO-64 T1.1-029 does not remove the duplicate keywords placed in your LSE system environment file by MACRO-64 V1.0-213. The following steps remove the duplicate keywords:

1. Invoke LSE with the following command:

```
$ LSE /NOSYSTEM_ENVIRONMENT -
_$ /ENVIRONMENT = SYS$LIBRARY:LSE$SYSTEM_ENVIRONMENT
```

2. Issue the following LSE commands:

```
COMMAND LANGUAGE VMSLSE
DELETE KEYWORDS KEYS
DELETE KEYWORDS DEFAULT_KEYWORDS_LIST
SAVE ENVIRONMENT /ALL SYS$LIBRARY:LSE$SYSTEM_ENVIRONMENT.ENV
```

   If you have defined your own version of the KEYS and DEFAULT_ KEYWORDS_LIST keyword-list names that are deleted above, you will want to redefine these keyword-list names prior to saving the environment file with the LSE "SAVE" command from above.

3.7 Code address in MIX psect

MACRO-64 V1.0-213 incorrectly issues a INVLCA diagnostic when you reference a local code address that is defined in a MIX psect with either the .LOCAL_ LINKAGE_PAIR or .LOCAL_CODE_ADDRESS directives.

MACRO-64 T1.1-029 corrects this problem.

3.8 Occasional ACCVIO with forward reference to temporary label

Under certain circumstances, MACRO-64 X1.1-016 access violates when processing a temporary label. You may see the access violation go away by adding or removing the /LIST qualifier, or any other change in processing that affects how the assembler allocates heap storage.

MACRO-64 T1.1-029 corrects this problem.

4
Known Problems and Restrictions

This chapter describes problems and restrictions known, but not resolved at the time of release.

4.1 .BLKx Directives Do Not Appear in Machine-Code Listing

The .BLKx directives do not appear in the machine-code listing with /MACHINE_CODE. However, the storage is correctly allocated and the offsets within the psect are correctly maintained.

4.2 Instructions for MIX Psects

Instructions which occur in psects with the MIX attribute appear in the machine listing as initial value specifiers for the .LONG directive.

4.3 Debugger Does Not Scroll Repeat Blocks

The debugger does not scroll the source window within a repeat block definition when you are executing instructions in a repeat block. Instead, the debugger treats the entire repeat block as a single statement.

4.4 Static .BYTE and .WORD Initializations with Shared Images

References in a .BYTE or .WORD directive to a global value defined in a shared image contain unpredictable results at image activation time.

4.5 Debugger Information for Local Data Labels

No local symbol debug information is generated for some local data labels having a runtime address bound to a storage location whose storage is allocated with one of the .BLKx directives. Specifically, this problem occurs when a .BLKx directive (e.g.: .BLKQ) is not preceded by a data location which is statically initialized with a MACRO-64 data initialization directive (e.g.: .WORD, .LONG). For example, no debug information is generated for the DBG1 in the following source code:

```
.psect a,noexe
DBG1: .BLKQ
```

A workaround to generate the proper debugger (DST) information for local labels is to place one or more data declarations which are statically initialized prior to the .BLKx directive. For example, the following insures local symbol debug information generation for the local data label DBG1 in the previous code segment:

```
.psect a,noexe
.QUAD 0
DBG1: .BLKQ
```

4.6 Assembler gives no clues when out of memory

If the assembler runs out of virtual memory with a very large assembly unit, the assembler aborts with a "%LIB-F-INSVIRMEM, insufficient virtual memory" diagnostic. The assembler should provide some indication of what it was trying to do when the failure occurred.

4.7 HELP MACRO/ALPHA_AXP doesn't work

HELP MACRO/ALPHA_AXP doesn't work. You must instead type HELP MACRO /ALPHA.

4.8 HELP MACRO/ALPHA broken on OpenVMS V1.5-FT3 and V1.5-FT4

If you type "HELP MACRO/ALPHA" on FT3 or FT4 of OpenVMS V1.5 after installing MACRO-64, two help sections are displayed for /ALPHA and help leaves you at the top of the help-library hierarchy. To navigate the MACRO-64 help, you must type "HELP MACRO/ALPHA topic", where "topic" is one of the MACRO-64 topics displayed when you type HELP MACRO/ALPHA.

This problem lies with MACRO-64. However, OpenVMS V1.5-213 has been changed such that "HELP MACRO/ALPHA" works correctly after installing MACRO-64.

4.9 MACZERO diagnostic anomalies

When two expressions specified with the .IF directive cannot be evaluated, two anomalies can occur. First, the diagnostic location indication can be that of the comma between the two expressions. This happens for both the first and the second expressions. Second, a spurious EOLEXP diagnostic is generated.

4.10 TRAPB instruction encoding

TRAPB instructions should should encode the unused registers as R31. MACRO-64 encodes them as R0. The workaround is to redefine the TRAPB instruction using the supplied $OPEF macro as follows:

```
$opdef TRAPB, format=custom, -
    encoding=<26:31=^x18,21:25=31,16:20=31,0:15=0>
```

If you use MACRO-64's optimization features, a side effect of redefining TRAPB this way is that certain optimization opportunities may be missed.

4.11 Superfluous argument/scratch register conflict

Currently, $CALL issues a diagnostic if you specify an argument-source or argument-target register that you also specify or default as a scratch register. (The default scratch registers are R0, R1, F0, and F1.) The diagnostic is legitimate if $CALL actually needs to use a scratch register to load an argument from memory and subsequently store it on the stack. However, the diagnostic is superfluous if $CALL does not need a scratch register.

4.12 No diagnostics in /PREPROCESS_ONLY mode

In /PREPROCESS_ONLY mode, MACRO-64 performs normal syntatic and semantic processing of your source program after the preprocessing stage in order to process normal symbol definitions and correctly maintain the current location counter ("."). However, most diagnostics that might be produced in these latter assembly stages would not be meaningful to you in /PREPROCESS_ONLY mode. Therefore, when MACRO-64 operates in /PREPROCESS_ONLY mode, the intended behavior is that all diagnostics arising from the latter assembly stages be suppressed. Unfortunately, preprocessing diagnostics, which would be meaningful in /PREPROCESS_ONLY mode, are also suppressed.

5
Known Documentation Errors

This chapter describes substantive, documentation errors known, but not resolved at the time of release.

5.1 No reference manual examples in Section 3.6 in Bookreader format

Section 3.6 of the reference manual in bookreader format has lost all its examples, and consequently, all of its substance. The hard-copy version of the reference manual is unaffected.

5.2 Reference manual section B.7.1.1 errors

Section B.7.1.1 in the reference manual contains the following errors:

- In item 1, .DISABLE_ALIGN_DATA should be .DISABLE ALIGN_DATA.
- In item 1, "with padding" should be "without padding."
- In item 2, .ENABLE_ALIGN_DATA should be .ENABLE ALIGN_DATA.

5.3 Reference manual section B.7.5 errors

Section B.7.5 contains 2 code examples. Both examples are in error in that the label definition "20$:" must follow the .ENDR directive. Currently the 20$: label definition precedes the .ENDR directive.

5.4 $CALL SET_ARG_INFO=FALSE doesn't always mean you must set R25

On Page 6-12, the reference manual states that if you specify SET_ARG_ INFO=FALSE with $CALL that you must set R25 yourself. That is not true in all cases. The reference manual should state that you must set R25 yourself before invoking $CALL if you wish to conform with the OpenVMS Calling Standard. If you do not need to conform with the OpenVMS Calling Standard and if the called routine doesn't need argument information in R25, you can specify SET_ ARG_INFO=FALSE and do nothing special with R25. This avoids the overhead involved when either you or $CALL load argument information into R25 at the expense of calling standard conformance.

5.5 Only BOOKBROWSER PAK works with MACRO-64 bookreader files

There are supposed to be two LMF PAKs which let you access the MACRO-64 bookreader files: MACRO64 and BOOKBROWSER. Unfortunately, while the MACRO64 PAK allows correct operation of the assembler, the MACRO64 PAK does not allow access to the MACRO-64 documentation in bookreader format as it should.

5.5 Only BOOKBROWSER PAK works with MACRO-64 bookreader files

The BOOKBROWSER PAK is a generic bookreader PAK which grants access to a number of bookreader files for a variety of products. The BOOKBROWSER PAK correctly allows access to the MACRO-64 bookreader files. The BOOKBROWSER PAK is available with the VMS On-Line documentation CD.

5.6 Listing Format is not documented

The MACRO-64 listing-file format should be documented in the reference manual. It is not.

5.7 Table 6-1 is revised

Table 6-1 on page 6-12 of the reference manual accurately describes the behavior of MACRO-64 V1.0-213. However, due to corrections in the $CALL macro and due to OpenVMS Calling Standard changes incorporated into the $CALL macro, table 6-12 is no longer accurate for MACRO-64 T1.1-029. Below is updated version of table 6-1 from page 6-12 of the reference manual.

Arg Qual	Arg Type	LOAD Instr	STORE Instr	AI Encod	RegArg Signat	MemArg Signat
/A	Address	LDA	STQ	I64	I32	I32
/D	D-float	LDQ	STQ	FD	FD	Q
/F	F-float	LDF	STF	FF	FF	I32
/G	G-float	LDG	STG	FG	FG	Q
/L	Longwrd	LDL	STQ	I64	I32	I32
/Q	Quadwrd	LDQ	STQ	I64	Q	Q
/S	S-float	LDS	STS	FS	FS	I32
/T	T-float	LDT	STT	FT	FT	Q
/UL	Unsignd Longwrd	LDL / ZAP #^xF0	STQ	I64	U32	I32