A ASCII Character Set

Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII
00_{10}	00_{16}	NUL	32_{10}	20_{16}	SP	64_{10}	40_{16}	@	96_{10}	60_{16}	′
01_{10}	01_{16}	SOH	33_{10}	21_{16}	!	65_{10}	41_{16}	A	97_{10}	61_{16}	a
02_{10}	02_{16}	STX	34_{10}	22_{16}	"	66_{10}	42_{16}	B	98_{10}	62_{16}	b
03_{10}	03_{16}	ETX	35_{10}	23_{16}	#	67_{10}	43_{16}	C	99_{10}	63_{16}	c
04_{10}	04_{16}	EOT	36_{10}	24_{16}	$	68_{10}	44_{16}	D	100_{10}	64_{16}	d
05_{10}	05_{16}	ENQ	37_{10}	25_{16}	%	69_{10}	45_{16}	E	101_{10}	65_{16}	e
06_{10}	06_{16}	ACK	38_{10}	26_{16}	&	70_{10}	46_{16}	F	102_{10}	66_{16}	f
07_{10}	07_{16}	BEL	39_{10}	27_{16}	'	71_{10}	47_{16}	G	103_{10}	67_{16}	g
08_{10}	08_{16}	BS	40_{10}	28_{16}	(	72_{10}	48_{16}	H	104_{10}	68_{16}	h
09_{10}	09_{16}	HT	41_{10}	29_{16}	)	73_{10}	49_{16}	I	105_{10}	69_{16}	i
10_{10}	$0A_{16}$	LF	42_{10}	$2A_{16}$	*	74_{10}	$4A_{16}$	J	106_{10}	$6A_{16}$	j
11_{10}	$0B_{16}$	VT	43_{10}	$2B_{16}$	+	75_{10}	$4B_{16}$	K	107_{10}	$6B_{16}$	k
12_{10}	$0C_{16}$	FF	44_{10}	$2C_{16}$	,	76_{10}	$4C_{16}$	L	108_{10}	$6C_{16}$	l
13_{10}	$0D_{16}$	CR	45_{10}	$2D_{16}$	-	77_{10}	$4D_{16}$	M	109_{10}	$6D_{16}$	m
14_{10}	$0E_{16}$	SO	46_{10}	$2E_{16}$	.	78_{10}	$4E_{16}$	N	110_{10}	$6E_{16}$	n
15_{10}	$0F_{16}$	SI	47_{10}	$2F_{16}$	/	79_{10}	$4F_{16}$	O	111_{10}	$6F_{16}$	o
16_{10}	10_{16}	DLE	48_{10}	30_{16}	0	80_{10}	50_{16}	P	112_{10}	70_{16}	p
17_{10}	11_{16}	DC1	49_{10}	31_{16}	1	81_{10}	51_{16}	Q	113_{10}	71_{16}	q
18_{10}	12_{16}	DC2	50_{10}	32_{16}	2	82_{10}	52_{16}	R	114_{10}	72_{16}	r
19_{10}	13_{16}	DC3	51_{10}	33_{16}	3	83_{10}	53_{16}	S	115_{10}	73_{16}	s
20_{10}	14_{16}	DC4	52_{10}	34_{16}	4	84_{10}	54_{16}	T	116_{10}	74_{16}	t
21_{10}	15_{16}	NAK	53_{10}	35_{16}	5	85_{10}	55_{16}	U	117_{10}	75_{16}	u
22_{10}	16_{16}	SYN	54_{10}	36_{16}	6	86_{10}	56_{16}	V	118_{10}	76_{16}	v
23_{10}	17_{16}	ETB	55_{10}	37_{16}	7	87_{10}	57_{16}	W	119_{10}	77_{16}	w
24_{10}	18_{16}	CAN	56_{10}	38_{16}	8	88_{10}	58_{16}	X	120_{10}	78_{16}	x
25_{10}	19_{16}	EM	57_{10}	39_{16}	9	89_{10}	59_{16}	Y	121_{10}	79_{16}	y
26_{10}	$1A_{16}$	SUB	58_{10}	$3A_{16}$	:	90_{10}	$5A_{16}$	Z	122_{10}	$7A_{16}$	z
27_{10}	$1B_{16}$	ESC	59_{10}	$3B_{16}$	;	91_{10}	$5B_{16}$	[	123_{10}	$7B_{16}$	{
28_{10}	$1C_{16}$	FS	60_{10}	$3C_{16}$	<	92_{10}	$5C_{16}$	\	124_{10}	$7C_{16}$	|
29_{10}	$1D_{16}$	GS	61_{10}	$3D_{16}$	=	93_{10}	$5D_{16}$	]	125_{10}	$7D_{16}$	}
30_{10}	$1E_{16}$	RS	62_{10}	$3E_{16}$	>	94_{10}	$5E_{16}$	^	126_{10}	$7E_{16}$	~
31_{10}	$1F_{16}$	US	63_{10}	$3F_{16}$	?	95_{10}	$5F_{16}$	_	127_{10}	$7F_{16}$	DEL

B Hexadecimal/Decimal Conversion

The following table lists the decimal value for each possible hexadecimal value in each byte of a longword. The following sections contain instructions to use the table to convert hexadecimal numbers to decimal and decimal numbers to hexadecimal.

Hexadecimal to Decimal Conversion Table

8		7		6		5		4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

ZK-2013-GE

B.1 Hexadecimal to Decimal

For each integer position of the hexadecimal value, locate the corresponding column integer and record its decimal equivalent in the conversion table. Add the decimal equivalent to obtain the decimal value.

For example:

```
D0500AD0 (hex)  =        ?(dec)

D0000000        = 3,489,660,928
  500000        =     5,242,880
     A00        =         2,560
      D0        =           208

D0500AD0        = 3,494,906,576
```

B.2 Decimal to Hexadecimal

To determine the hexadecimal equivalent of a given decimal value, perform the following steps:

1. In the conversion table, locate the largest decimal value that does not exceed the decimal number to be converted.
2. Record the hexadecimal equivalent, followed by the number of zeros that corresponds to the integer column minus 1.
3. Subtract the table decimal value from the decimal number to be converted.
4. Repeat steps 1 to 3 until the subtraction balance equals zero. Add the hexadecimal equivalents to obtain the hexadecimal value.

For example:

```
22,466 (dec)    = ?(hex)

20,480          = 5000        22,466
 1,792          =  700       -20,480
   192          =   C0
     2          =    2         1,986
                              - 1,792
22,466          = 57C2
                                  194
                              -   192
                                    2
                              -     2
                                    0
```

B.3 Powers of 2 and 16

This section lists the decimal values of powers of 2 and 16. These values are useful in converting decimal numbers to hexadecimal.

Powers of 2		Powers of 16	
2**n	n	16**n	n
256	8	1	0
512	9	16	1
1024	10	256	2
2048	11	4096	3
4096	12	65536	4
8192	13	1048576	5
16384	14	16777216	6
32768	15	268435456	7
65536	16	4294967296	8
131072	17	68719476736	9
262144	18	1099511627776	10
524288	19	17592186044416	11
1048576	20	281474976710656	12
2097152	21	4503599627370496	13
4194304	22	72057594037927936	14
8388608	23	1152921504606846976	15
16777216	24		

C VAX MACRO Assembler Directives and Language Summary

This appendix summarizes the general assembler and macro directives (in alphabetical order), special characters, unary operators, binary operators, and addressing modes.

C.1 Assembler Directives

Table C–1 summarizes the VAX MACRO assembler directives:

Table C–1 Assembler Directives

Format	Operation
.ADDRESS address-list	Stores successive longwords of address data
.ALIGN keyword[,expression]	Aligns the location counter to the boundary specified by the keyword
.ALIGN integer[,expression]	Aligns location counter to the boundary specified by (2^integer)
.ASCIC string	Stores the ASCII string (enclosed in delimiters), preceded by a count byte
.ASCID string	Stores the ASCII string (enclosed in delimiters), preceded by a string descriptor
.ASCII string	Stores the ASCII string (enclosed in delimiters)
.ASCIZ string	Stores the ASCII string (enclosed in delimiters) followed by a 0 byte
.BLKA expression	Reserves longwords of address data
.BLKB expression	Reserves bytes for data
.BLKD expression	Reserves quadwords for double-precision floating-point data
.BLKF expression	Reserves longwords for single-precision floating-point data
.BLKG expression	Reserves quadwords for floating-point data
.BLKH expression	Reserves octawords for extended-precision floating-point data
.BLKL expression	Reserves longwords for data
.BLKO expression	Reserves octawords for data
.BLKQ expression	Reserves quadwords for data

(continued on next page)

C.1 Assembler Directives

Table C–1 (Cont.) Assembler Directives

Format	Operation
.BLKW expression	Reserves words for data
.BYTE expression-list	Generates successive bytes of data; each byte contains the value of the specified expression
.CROSS	Enables cross-referencing of all symbols
.CROSS symbol-list	Cross-references specified symbols
.DEBUG symbol-list	Makes symbol names known to the debugger
.DEFAULT DISPLACEMENT, keyword	Specifies the default displacement length for the relative addressing modes
.D_FLOATING literal-list	Generates 8-byte double-precision floating-point data
.DISABLE argument-list	Disables functions specified in argument-list
.DOUBLE literal-list	Equivalent to .D_FLOATING
.DSABL argument-list	Equivalent to .DISABLE
.ENABL argument-list	Equivalent to .ENABLE
.ENABLE argument-list	Enables functions specified in argument-list
.END [symbol]	Indicates logical end of source program; optional symbol specifies transfer address
.ENDC	Indicates end of conditional assembly block
.ENDM [macro-name]	Indicates end of macro definition
.ENDR	Indicates end of repeat block
.ENTRY symbol [,expression]	Procedure entry directive
.ERROR [expression] ;comment	Displays specified error message
.EVEN	Ensures that the current location counter has an even value (adds 1 if it is odd)
.EXTERNAL symbol-list	Indicates specified symbols are externally defined
.EXTRN symbol-list	Equivalent to .EXTERNAL
.F_FLOATING literal-list	Generates 4-byte single-precision floating-point data
.FLOAT literal-list	Equivalent to .F_FLOATING
.G_FLOATING literal-list	Generates 8-byte G_floating-point data

(continued on next page)

Table C–1 (Cont.) Assembler Directives

Format	Operation
.GLOBAL symbol-list	Indicates specified symbols are global symbols
.GLOBL	Equivalent to .GLOBAL
.H_FLOATING literal-list	Generates 16-byte extended-precision H_floating-point data
.IDENT string	Provides means of labeling object module with additional data
.IF condition [,] argument(s)	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 arguments specified
.IFF	Equivalent to .IF_FALSE
.IF_FALSE	Appears only within a conditional assembly block; begins block of code to be assembled if the original condition tests false
.IFT	Equivalent to .IF_TRUE
.IFTF	Equivalent to .IF_TRUE_FALSE
.IF_TRUE	Appears only within a conditional assembly block; begins block of code to be assembled if the original condition tests true
.IF_TRUE_FALSE	Appears only within a conditional assembly block; begins block of code to be assembled unconditionally
.IIF condition argument(s), statement	Acts as a 1-line conditional assembly block where the condition is tested for the argument specified; the statement is assembled only if the condition tests true
.IRP symbol,<argument list>	Replaces a formal argument with successive actual arguments specified in an argument list
.IRPC symbol,<BIT_STRING>	Replaces a formal argument with successive single characters specified in string
.LIBRARY macro-library-name	Specifies a macro library
.LINK "file-spec" [/qualifier[=(module-name[,...])],...]	Includes linker option records in object module
.LIST [argument-list]	Equivalent to .SHOW

(continued on next page)

Table C–1 (Cont.) Assembler Directives

Format	Operation
.LONG expression-list	Generates successive longwords of data; each longword contains the value of the specified expression
.MACRO macro-name [formal-argument-list]	Begins a macro definition
.MASK symbol [,expression]	Reserves a word for and copies a register save mask
.MCALL macro-name-list	Specifies the system or user-defined macros, or both, in libraries that are required to assemble the source program
.MDELETE macro-name-list	Deletes from memory the macro definitions of the macros in the list
.MEXIT	Exits from the expansion of a macro before the end of the macro is encountered
.NARG symbol	Determines the number of arguments in the current macro call
.NCHR symbol,<BIT_STRING>	Determines the number of characters in a specified character string
.NLIST [argument-list]	Equivalent to .NOSHOW
.NOCROSS	Disables cross-referencing of all symbols
.NOCROSS symbol-list	Disables cross-referencing of specified symbols
.NOSHOW	Decrements listing level count
.NOSHOW argument-list	Controls listing of macros and conditional assembly blocks
.NTYPE symbol,operand	Can appear only within a macro definition; equates the symbol to the addressing mode of the specified operand
.OCTA literal	Stores 16 bytes of data
.OCTA symbol	Stores 16 bytes of data
.ODD	Ensures that the current location counter has an odd value (adds 1 if it is even)
.OPDEF opcode value, operand-descriptor-list	Defines an opcode and its operand list
.PACKED decimal-string [,symbol]	Generates packed decimal data, 2 digits per byte

(continued on next page)

Table C–1 (Cont.) Assembler Directives

Format	Operation
.PAGE	Causes the assembly listing to skip to the top of the next page and to increment the page count
.PRINT [expression] ;comment	Displays the specified message
.PSECT	Begins or resumes the blank program section
.PSECT section-name argument list	Begins or resumes a user-defined program section
.QUAD literal	Stores 8 bytes of data
.QUAD symbol	Stores 8 bytes of data
.REF1 operand	Generates byte operand
.REF2 operand	Generates word operand
.REF4 operand	Generates longword operand
.REF8 operand	Generates quadword operand
.REF16 operand	Generates octaword operand
.REPEAT expression	Begins a repeat block; the section of code up to the next .ENDR directive is repeated the number of times specified by the expression
.REPT	Equivalent to .REPEAT
.RESTORE	Equivalent to .RESTORE_PSECT
.RESTORE_PSECT	Restores program section context from the program section context stack
.SAVE [LOCAL_BLOCK]	Equivalent to .SAVE_PSECT
.SAVE_PSECT [LOCAL_BLOCK]	Saves current program section context on the program section context stack
.SBTTL comment-string	Equivalent to .SUBTITLE
.SHOW	Increments listing level count
.SHOW argument-list	Controls listing of macros and conditional assembly blocks
.SIGNED_BYTE expression-list	Stores successive bytes of signed data
.SIGNED_WORD expression-list	Stores successive words of signed data
.SUBTITLE comment-string	Causes the specified string to be printed as part of the assembly listing page header; the string component of each .SUBTITLE is collected into a table of contents at the beginning of the assembly listing

(continued on next page)

Table C–1 (Cont.) Assembler Directives

Format	Operation
.TITLE module-name comment-string	Assigns the first 15 characters in the string as an object module name and causes the string to appear on each page of the assembly listing
.TRANSFER symbol	Directs the linker to redefine the value of the global symbol for use in a shareable image
.WARN [expression] ;comment	Displays specified warning message
.WEAK symbol-list	Indicates that each of the listed symbols has the weak attribute
.WORD expression-list	Generates successive words of data; each word contains the value of the corresponding specified expression

C.2 Special Characters

Table C–2 summarizes the VAX MACRO special characters:

Table C–2 Special Characters Used in VAX MACRO Statements

Character	Character Name	Functions
_	Underscore	Character in symbol names
$	Dollar sign	Character in symbol names
.	Period	Character in symbol names, current location counter, and decimal point
:	Colon	Label terminator
=	Equal sign	Direct assignment operator and macro keyword argument terminator
	Tab	Field terminator
	Space	Field terminator
#	Number sign	Immediate addressing mode indicator
@	At sign	Deferred addressing mode indicator and arithmetic shift operator
,	Comma	Field, operand, and item separator
;	Semicolon	Comment field indicator
+	Plus sign	Autoincrement addressing mode indicator, unary plus operator, and arithmetic addition operator
-	Minus sign	Autodecrement addressing mode indicator, unary minus operator, arithmetic subtraction operator, and line continuation indicator

(continued on next page)

Table C–2 (Cont.) Special Characters Used in VAX MACRO Statements

Character	Character Name	Functions
*	Asterisk	Arithmetic multiplication operator
/	Slash	Arithmetic division operator
&	Ampersand	Logical AND operator
!	Exclamation point	Logical inclusive OR operator
\	Backslash	Logical exclusive OR and numeric conversion indicator in macro arguments
^	Circumflex	Unary operator indicator and macro argument delimiter
[ ]	Square brackets	Index addressing mode and repeat count indicators
( )	Parentheses	Register deferred addressing mode indicators
<>	Angle brackets	Argument or expression grouping delimiters
?	Question mark	Created label indicator in macro arguments
'	Apostrophe	Macro argument concatenation indicator
%	Percent sign	Macro string operators

C.3 Operators

This section lists the VAX MACRO unary, binary, and macro string operators.

C.3.1 Unary Operators

Table C–3 summarizes the VAX MACRO unary operators:

Table C–3 Summary of Unary Operators

Unary Operator	Operator Name	Example	Effect
+	Plus sign	+A	Results in the positive value of A (default)
–	Minus sign	–A	Results in the negative (two's complement) value of A
^B	Binary	^B11000111	Specifies that 11000111 is a binary number
^D	Decimal	^D127	Specifies that 127 is a decimal number

(continued on next page)

Table C–3 (Cont.) Summary of Unary Operators

Unary Operator	Operator Name	Example	Effect
^O	Octal	^034	Specifies that 34 is an octal number
^X	Hexadecimal	^XFCF9	Specifies that FCF9 is a hexadecimal number
^A	ASCII	^A/ABC/	Produces an ASCII string; the characters between the matching delimiters are converted to ASCII representation
^M	Register mask	^M<R3,R4,R5>	Specifies the registers R3, R4, and R5 in the register mask
^F	Floating point	^F3.0	Specifies that 3.0 is a floating-point number
^C	Complement	^C24	Produces the one's complement value of 24 (decimal)

C.3.2 Binary Operators

Table C–4 summarizes the VAX MACRO binary operators:

Table C–4 Summary of Binary Operators

Binary Operator	Operator Name	Example	Operation
+	Plus sign	A+B	Addition
–	Minus sign	A–B	Subtraction
*	Asterisk	A*B	Multiplication
/	Slash	A/B	Division
@	At sign	A@B	Arithmetic Shift
&	Ampersand	A&B	Logical AND
!	Exclamation point	A!B	Logical inclusive OR
\	Backslash	A\B	Logical exclusive OR

C.3.3 Macro String Operators

Table C–5 summarizes the macro string operators. These operators can be used only in macros.

Table C–5 Macro String Operators

Format	Function
%LENGTH(string)	Returns the length of the string
%LOCATE(string1,string2[,symbol])	Locates the substring string1 within string2 starting the search at the character position specified by symbol
%EXTRACT(symbol1,symbol2,string)	Extracts a substring from string that begins at character position specified by symbol1 and has a length specified by symbol2

C.4 Addressing Modes

Table C–6 summarizes the VAX MACRO addressing modes:

Table C–6 Summary of Addressing Modes

Type	Addressing Mode	Format	Hex Value	Description	Can Be Indexed?
General register	Register	Rn	5	Register contains the operand.	No
	Register deferred	(Rn)	6	Register contains the address of the operand.	Yes
	Autoincrement	(Rn)+	8	Register contains the address of the operand; the processor increments the register contents by the size of the operand data type.	Yes
	Autoincrement deferred	@(Rn)+	9	Register contains the address of the operand address; the processor increments the register contents by 4.	Yes
	Autodecrement	-(Rn)	7	The processor decrements the register contents by the size of the operand data type; the register then contains the address of the operand.	Yes
	Displacement	dis(Rn) B^dis(Rn) W^dis(Rn) L^dis(Rn)	 A C E	The sum of the contents of the register and the displacement is the address of the operand; B^, W^, and L^, respectively, indicate byte, word, and longword displacement.	Yes
	Displacement deferred	@dis(Rn) @B^dis(Rn) @W^dis(Rn) @L^dis(Rn)	 B D F	The sum of the contents of the register and the displacement is the address of the operand address; B^, W^, and L^, respectively, indicate, byte, word, and longword displacement.	Yes

Key:

Rn—Any general register R0 to R12. Note that the AP, FP, or SP register can be used in place of Rn.
Rx—Any general register R0 to R12. Note that the AP, FP, or SP register can be used in place of Rx. Rx cannot be the same as the Rn specified in the base-mode for certain base modes (see Section 5.3).
dis—An expression specifying a displacement.
address—An expression specifying an address.
literal—An expression, an integer constant, or a floating-point constant.

(continued on next page)

Table C–6 (Cont.) Summary of Addressing Modes

Type	Addressing Mode	Format	Hex Value	Description	Can Be Indexed?
	Literal	#literal S^#literal	 0-3	The literal specified is the operand; the literal is stored as a short literal.	No
Program Counter	Relative	address B^address W^address L^address	 A C E	The address specified is the address of the operand; the address is stored as a displacement from the PC; B^, W^, and L^, respectively, indicate byte, word, and longword displacement.	Yes
	Relative deferred	@address @B^address @W^address @L^address	 B D F	The address specified is the address of the operand address; the address specified is stored as a displacement from the PC; B^, W^, and L^ indicate byte, word, and longword displacement, respectively.	Yes
	Absolute	@#address	9	The address specified is the address of the operand; the address specified is stored as an absolute virtual address, not as a displacement.	Yes
	Immediate	#literal I^#literal	 8	The literal specified is the operand; the literal is stored as a byte, word, longword, or quadword.	No
	General	G^address	—	The address specified is the address of the operand; if the address is defined as relocatable, the linker stores the address as a displacement from the PC; if the address is defined as an absolute virtual address, the linker stores the address as an absolute value.	Yes

Key:

Rn—Any general register R0 to R12. Note that the AP, FP, or SP register can be used in place of Rn.
Rx—Any general register R0 to R12. Note that the AP, FP, or SP register can be used in place of Rx. Rx cannot be the same as the Rn specified in the base-mode for certain base modes (see Section 5.3).
dis—An expression specifying a displacement.
address—An expression specifying an address.
literal—An expression, an integer constant, or a floating-point constant.

(continued on next page)

Table C–6 (Cont.) Summary of Addressing Modes

Type	Addressing Mode	Format	Hex Value	Description	Can Be Indexed?
Index	Index	base-mode[Rx]	4	The base-mode specifies the base address, and the register specifies the index; the sum of the base address and the product of the contents of Rx and the size of the operand data type is the address of the operand; base mode can be any addressing mode except register, immediate, literal, index, or branch.	No
Branch	Branch	address	—	The address specified is the operand; this address is stored as a displacement from the PC; branch mode can only be used with the branch instructions.	No

Key:

Rn—Any general register R0 to R12. Note that the AP, FP, or SP register can be used in place of Rn.
Rx—Any general register R0 to R12. Note that the AP, FP, or SP register can be used in place of Rx. Rx cannot be the same as the Rn specified in the base-mode for certain base modes (see Section 5.3).
dis—An expression specifying a displacement.
address—An expression specifying an address.
literal—An expression, an integer constant, or a floating-point constant.

D Permanent Symbol Table Defined for Use with VAX MACRO

The permanent symbol table (PST) contains the symbols that VAX MACRO automatically recognizes. These symbols consist of both opcodes and assembler directives. Table D–1, Table D–2, and Table D–3 present the opcodes (instruction set) in alphabetical and numerical order. Section C.1 (in Appendix C) presents the assembler directives.

See Chapter 9 and Chapter 10 for detailed descriptions of the instruction set.

Table D–1 Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
9D	ACBB	Add compare and branch byte
6F	ACBD	Add compare and branch D_floating
4F	ACBF	Add compare and branch F_floating
4FFD	ACBG	Add compare and branch G_floating
6FFD	ACBH	Add compare and branch H_floating
F1	ACBL	Add compare and branch longword
3D	ACBW	Add compare and branch word
58	ADAWI	Add aligned word interlocked
80	ADDB2	Add byte 2 operand
81	ADDB3	Add byte 3 operand
60	ADDD2	Add D_floating 2 operand
61	ADDD3	Add D_floating 3 operand
40	ADDF2	Add F_floating 2 operand
41	ADDF3	Add F_floating 3 operand
40FD	ADDG2	Add G_floating 2 operand
41FD	ADDG3	Add G_floating 3 operand
60FD	ADDH2	Add H_floating 2 operand
61FD	ADDH3	Add H_floating 3 operand
C0	ADDL2	Add longword 2 operand
C1	ADDL3	Add longword 3 operand
20	ADDP4	Add packed 4 operand
21	ADDP6	Add packed 6 operand
A0	ADDW2	Add word 2 operand
A1	ADDW3	Add word 3 operand

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
D8	ADWC	Add with carry
F3	AOBLEQ	Add one and branch on less or equal
F2	AOBLSS	Add one and branch on less
78	ASHL	Arithmetic shift longword
F8	ASHP	Arithmetic shift and round packed
79	ASHQ	Arithmetic shift quadword
E1	BBC	Branch on bit clear
E5	BBCC	Branch on bit clear and clear
E7	BBCCI	Branch on bit clear and clear interlocked
E3	BBCS	Branch on bit clear and set
E0	BBS	Branch on bit set
E4	BBSC	Branch on bit set and clear
E2	BBSS	Branch on bit set and set
E6	BBSSI	Branch on bit set and set interlocked
1E	BCC	Branch on carry clear
1F	BCS	Branch on carry set
13	BEQL	Branch on equal
13	BEQLU	Branch on equal unsigned
18	BGEQ	Branch on greater or equal
1E	BGEQU	Branch on greater or equal unsigned
14	BGTR	Branch on greater
1A	BGTRU	Branch on greater unsigned
8A	BICB2	Bit clear byte 2 operand
8B	BICB3	Bit clear byte 3 operand
CA	BICL2	Bit clear longword 2 operand
CB	BICL3	Bit clear longword 3 operand
B9	BICPSW	Bit clear program status word
AA	BICW2	Bit clear word 2 operand
AB	BICW3	Bit clear word 3 operand
88	BISB2	Bit set byte 2 operand
89	BISB3	Bit set byte 3 operand
C8	BISL2	Bit set longword 2 operand
C9	BISL3	Bit set longword 3 operand
B8	BISPSW	Bit set program status word
A8	BISW2	Bit set word 2 operand

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
A9	BISW3	Bit set word 3 operand
93	BITB	Bit test byte
D3	BITL	Bit test longword
B3	BITW	Bit test word
E9	BLBC	Branch on low bit clear
E8	BLBS	Branch on low bit set
15	BLEQ	Branch on less or equal
1B	BLEQU	Branch on less or equal unsigned
19	BLSS	Branch on less
1F	BLSSU	Branch on less unsigned
12	BNEQ	Branch on not equal
12	BNEQU	Branch on not equal unsigned
03	BPT	Break point trap
11	BRB	Branch with byte displacement
31	BRW	Branch with word displacement
10	BSBB	Branch to subroutine with byte displacement
30	BSBW	Branch to subroutine with word displacement
1C	BVC	Branch on overflow clear
1D	BVS	Branch on overflow set
FA	CALLG	Call with general argument list
FB	CALLS	Call with stack
8F	CASEB	Case byte
CF	CASEL	Case longword
AF	CASEW	Case word
BD	CHME	Change mode to executive
BC	CHMK	Change mode to kernel
BE	CHMS	Change mode to supervisor
BF	CHMU	Change mode to user
94	CLRB	Clear byte
7C	CLRD	Clear D_floating
DF	CLRF	Clear F_floating
7C	CLRG	Clear G_floating
7CFD	CLRH	Clear H_floating
D4	CLRL	Clear longword
7CFD	CLRO	Clear octaword

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
7C	CLRQ	Clear quadword
B4	CLRW	Clear word
91	CMPB	Compare byte
29	CMPC3	Compare character 3 operand
2D	CMPC5	Compare character 5 operand
71	CMPD	Compare D_floating
51	CMPF	Compare F_floating
51FD	CMPG	Compare G_floating
71FD	CMPH	Compare H_floating
D1	CMPL	Compare longword
35	CMPP3	Compare packed 3 operand
37	CMPP4	Compare packed 4 operand
EC	CMPV	Compare field
B1	CMPW	Compare word
ED	CMPZV	Compare zero-extended field
0B	CRC	Calculate cyclic redundancy check
6C	CVTBD	Convert byte to D_floating
4C	CVTBF	Convert byte to F_floating
4CFD	CVTBG	Convert byte to G_floating
6CFD	CVTBH	Convert byte to H_floating
98	CVTBL	Convert byte to longword
99	CVTBW	Convert byte to word
68	CVTDB	Convert D_floating to byte
76	CVTDF	Convert D_floating to F_floating
32FD	CVTDH	Convert D_floating to H_floating
6A	CVTDL	Convert D_floating to longword
69	CVTDW	Convert D_floating to word
48	CVTFB	Convert F_floating to byte
56	CVTFD	Convert F_floating to D_floating
99FD	CVTFG	Convert F_floating to G_floating
98FD	CVTFH	Convert F_floating to H_floating
4A	CVTFL	Convert F_floating to longword
49	CVTFW	Convert F_floating to word
48FD	CVTGB	Convert G_floating to byte
33FD	CVTGF	Convert G_floating to F_floating

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
56FD	CVTGH	Convert G_floating to H_floating
4AFD	CVTGL	Convert G_floating to longword
49FD	CVTGW	Convert G_floating to word
68FD	CVTHB	Convert H_floating to byte
F7FD	CVTHD	Convert H_floating to D_floating
F6FD	CVTHF	Convert H_floating to F_floating
76FD	CVTHG	Convert H_floating to G_floating
6AFD	CVTHL	Convert H_floating to longword
69FD	CVTHW	Convert H_floating to word
F6	CVTLB	Convert longword to byte
6E	CVTLD	Convert longword to D_floating
4E	CVTLF	Convert longword to F_floating
4EFD	CVTLG	Convert longword to G_floating
6EFD	CVTLH	Convert longword to H_floating
F9	CVTLP	Convert longword to packed
F7	CVTLW	Convert longword to word
36	CVTPL	Convert packed to longword
08	CVTPS	Convert packed to leading separate
24	CVTPT	Convert packed to trailing
6B	CVTRDL	Convert rounded D_floating to longword
4B	CVTRFL	Convert rounded F_floating to longword
4BFD	CVTRGL	Convert rounded G_floating to longword
6BFD	CVTRHL	Convert rounded H_floating to longword
09	CVTSP	Convert leading separate to packed
26	CVTTP	Convert trailing to packed
33	CVTWB	Convert word to byte
6D	CVTWD	Convert word to D_floating
4D	CVTWF	Convert word to F_floating
4DFD	CVTWG	Convert word to G_floating
6DFD	CVTWH	Convert word to H_floating
32	CVTWL	Convert word to longword
97	DECB	Decrement byte
D7	DECL	Decrement longword
B7	DECW	Decrement word
86	DIVB2	Divide byte 2 operand

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
87	DIVB3	Divide byte 3 operand
66	DIVD2	Divide D_floating 2 operand
67	DIVD3	Divide D_floating 3 operand
46	DIVF2	Divide F_floating 2 operand
47	DIVF3	Divide F_floating 3 operand
46FD	DIVG2	Divide G_floating 2 operand
47FD	DIVG3	Divide G_floating 3 operand
66FD	DIVH2	Divide H_floating 2 operand
67FD	DIVH3	Divide H_floating 3 operand
C6	DIVL2	Divide longword 2 operand
C7	DIVL3	Divide longword 3 operand
27	DIVP	Divide packed
A6	DIVW2	Divide word 2 operand
A7	DIVW3	Divide word 3 operand
38	EDITPC	Edit packed to character
7B	EDIV	Extended divide
74	EMODD	Extended modulus D_floating
54	EMODF	Extended modulus F_floating
54FD	EMODG	Extended modulus G_floating
74FD	EMODH	Extended modulus H_floating
7A	EMUL	Extended multiply
EE	EXTV	Extract field
EF	EXTZV	Extract zero-extended field
EB	FFC	Find first clear bit
EA	FFS	Find first set bit
00	HALT	Halt
96	INCB	Increment byte
D6	INCL	Increment longword
B6	INCW	Increment word
0A	INDEX	Index calculation
5C	INSQHI	Insert into queue at head, interlocked
5D	INSQTI	Insert into queue at tail, interlocked
0E	INSQUE	Insert into queue
F0	INSV	Insert field
EDFD	IOTA	Generate compressed iota vector

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
17	JMP	Jump
16	JSB	Jump to subroutine
06	LDPCTX	Load program context
3A	LOCC	Locate character
39	MATCHC	Match characters
92	MCOMB	Move complemented byte
D2	MCOML	Move complemented longword
B2	MCOMW	Move complemented word
DB	MFPR	Move from processor register
31FD	MFVP	Move from vector processor
8E	MNEGB	Move negated byte
72	MNEGD	Move negated D_floating
52	MNEGF	Move negated F_floating
52FD	MNEGG	Move negated G_floating
72FD	MNEGH	Move negated H_floating
CE	MNEGL	Move negated longword
AE	MNEGW	Move negated word
9E	MOVAB	Move address of byte
7E	MOVAD	Move address of D_floating
DE	MOVAF	Move address of F_floating
7E	MOVAG	Move address of G_floating
7EFD	MOVAH	Move address of H_floating
DE	MOVAL	Move address of longword
7EFD	MOVAO	Move address of octaword
7E	MOVAQ	Move address of quadword
3E	MOVAW	Move address of word
90	MOVB	Move byte
28	MOVC3	Move character 3 operand
2C	MOVC5	Move character 5 operand
70	MOVD	Move D_floating
50	MOVF	Move F_floating
50FD	MOVG	Move G_floating
70FD	MOVH	Move H_floating
D0	MOVL	Move longword
7DFD	MOVO	Move data

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
34	MOVP	Move packed
DC	MOVPSL	Move program status longword
7D	MOVQ	Move quadword
2E	MOVTC	Move translated characters
2F	MOVTUC	Move translated until character
B0	MOVW	Move word
0A	MOVZBL	Move zero-extended byte to longword
9B	MOVZBW	Move zero-extended byte to word
3C	MOVZWL	Move zero-extended word to longword
DA	MTPR	Move to processor register
A9FD	MTVP	Move to vector processor
84	MULB2	Multiply byte 2 operand
85	MULB3	Multiply byte 3 operand
64	MULD2	Multiply D_floating 2 operand
65	MULD3	Multiply D_floating 3 operand
44	MULF2	Multiply F_floating 2 operand
45	MULF3	Multiply F_floating 3 operand
44FD	MULG2	Multiply G_floating 2 operand
45FD	MULG3	Multiply G_floating 3 operand
64FD	MULH2	Multiply H_floating 2 operand
65FD	MULH3	Multiply H_floating 3 operand
C4	MULL2	Multiply longword 2 operand
C5	MULL3	Multiply longword 3 operand
25	MULP	Multiply packed
A4	MULW2	Multiply word 2 operand
A5	MULW3	Multiply word 3 operand
01	NOP	No operation
75	POLYD	Evaluate polynomial D_floating
55	POLYF	Evaluate polynomial F_floating
55FD	POLYG	Evaluate polynomial G_floating
75FD	POLYH	Evaluate polynomial H_floating
BA	POPR	Pop registers
0C	PROBER	Probe read access
0D	PROBEW	Probe write access
9F	PUSHAB	Push address of byte

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
7F	PUSHAD	Push address of D_floating
DF	PUSHAF	Push address of F_floating
7F	PUSHAG	Push address of G_floating
7FFD	PUSHAH	Push address of H_floating
DF	PUSHAL	Push address of longword
7FFD	PUSHAO	Push address of octaword
7F	PUSHAQ	Push address of quadword
3F	PUSHAW	Push address of word
DD	PUSHL	Push longword
BB	PUSHR	Push registers
02	REI	Return from exception or interrupt
5E	REMQHI	Remove from queue at head, interlocked
5F	REMQTI	Remove from queue at tail, interlocked
0F	REMQUE	Remove from queue
04	RET	Return from called procedure
9C	ROTL	Rotate longword
05	RSB	Return from subroutine
D9	SBWC	Subtract with carry
2A	SCANC	Scan for character
3B	SKPC	Skip character
F4	SOBGEQ	Subtract one and branch on greater or equal
F5	SOBGTR	Subtract one and branch on greater
2B	SPANC	Span characters
82	SUBB2	Subtract byte 2 operand
83	SUBB3	Subtract byte 3 operand
62	SUBD2	Subtract D_floating 2 operand
63	SUBD3	Subtract D_floating 3 operand
42	SUBF2	Subtract F_floating 2 operand
43	SUBF3	Subtract F_floating 3 operand
42FD	SUBG2	Subtract G_floating 2 operand
43FD	SUBG3	Subtract G_floating 3 operand
62FD	SUBH2	Subtract H_floating 2 operand
63FD	SUBH3	Subtract H_floating 3 operand
C2	SUBL2	Subtract longword 2 operand
C3	SUBL3	Subtract longword 3 operand

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
22	SUBP4	Subtract packed 4 operand
23	SUBP6	Subtract packed 6 operand
A2	SUBW2	Subtract word 2 operand
A3	SUBW3	Subtract word 3 operand
07	SVPCTX	Save process context
95	TSTB	Test byte
73	TSTD	Test D_floating
53	TSTF	Test F_floating
53FD	TSTG	Test G_floating
73FD	TSTH	Test H_floating
D5	TSTL	Test longword
B5	TSTW	Test word
35FD	VGATHL	Gather longword vector from memory to vector register
37FD	VGATHQ	Gather quadword vector from memory to vector register
34FD	VLDL	Load longword vector from memory to vector register
36FD	VLDQ	Load quadword vector from memory to vector register
87FD	VSADDD	Vector scalar add D_floating
85FD	VSADDF	Vector scalar add F_floating
83FD	VSADDG	Vector scalar add G_floating
81FD	VSADDL	Vector scalar add longword
CDFD	VSBICL	Vector scalar bit clear longword
C9FD	VSBISL	Vector scalar bit set longword
9DFD	VSCATL	Scatter longword vector from vector register to memory
9FFD	VSCATQ	Scatter quadword vector from vector register to memory
C7FD	VSCMPD	Vector scalar compare D_floating
C5FD	VSCMPF	Vector scalar compare F_floating
C3FD	VSCMPG	Vector scalar compare G_floating
C1FD	VSCMPL	Vector scalar compare longword
AFFD	VSDIVD	Vector scalar divide D_floating
ADFD	VSDIVF	Vector scalar divide F_floating
ABFD	VSDIVG	Vector scalar divide G_floating
EFFD	VSMERGE	Vector scalar merge

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
A7FD	VSMULD	Vector scalar multiply D_floating
A5FD	VSMULF	Vector scalar multiply F_floating
A3FD	VSMULG	Vector scalar multiply G_floating
A1FD	VSMULL	Vector scalar multiply longword
E5FD	VSSLLL	Vector scalar shift left logical longword
E1FD	VSSRLL	Vector scalar shift right logical longword
8FFD	VSSUBD	Vector scalar subtract D_floating
8DFD	VSSUBF	Vector scalar subtract F_floating
8BFD	VSSUBG	Vector scalar subtract G_floating
89FD	VSSUBL	Vector scalar subtract longword
9CFD	VSTL	Store longword vector from vector register to memory
9EFD	VSTQ	Store quadword vector from vector register to memory
E9FD	VSXORL	Vector scalar exclusive-OR longword
A8FD	VSYNC	Synchronize vector memory access
86FD	VVADDD	Vector vector add D_floating
84FD	VVADDF	Vector vector add F_floating
82FD	VVADDG	Vector vector add G_floating
80FD	VVADDL	Vector vector add longword
CCFD	VVBICL	Vector vector bit clear longword
C8FD	VVBISL	Vector vector bit set longword
C6FD	VVCMPD	Vector vector compare D_floating
C4FD	VVCMPF	Vector vector compare F_floating
C2FD	VVCMPG	Vector vector compare G_floating
C0FD	VVCMPL	Vector vector compare longword
ECFD	VVCVT	Vector convert
AEFD	VVDIVD	Vector vector divide D_floating
ACFD	VVDIVF	Vector vector divide F_floating
AAFD	VVDIVG	Vector vector divide G_floating
EEFD	VVMERGE	Vector vector merge
A6FD	VVMULD	Vector vector multiply F_floating
A4FD	VVMULF	Vector vector multiply F_floating
A2FD	VVMULG	Vector vector multiply G_floating
A0FD	VVMULL	Vector vector multiply longword
E4FD	VVSLLL	Vector vector shift left logical longword

(continued on next page)

Table D–1 (Cont.) Opcodes (Alphabetic Order) and Functions

Hex Value	Mnemonic	Functional Name
E0FD	VVSRLL	Vector vector shift right logical longword
8EFD	VVSUBD	Vector vector subtract D_floating
8CFD	VVSUBF	Vector vector subtract F_floating
8AFD	VVSUBG	Vector vector subtract G_floating
88FD	VVSUBL	Vector vector subtract longword
E8FD	VVXORL	Vector vector exclusive-OR longword
FC	XFC	Extended function call
8C	XORB2	Exclusive-OR byte 2 operand
8D	XORB3	Exclusive-OR byte 3 operand
CC	XORL2	Exclusive-OR longword 2 operand
CD	XORL3	Exclusive-OR longword 3 operand
AC	XORW2	Exclusive-OR word 2 operand
AD	XORW3	Exclusive-OR word 3 operand

Table D–2 One_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
00	HALT	30	BSBW
01	NOP	31	BRW
02	REI	32	CVTWL
03	BPT	33	CVTWB
04	RET	34	MOVP
05	RSB	35	CMPP3
06	LDPCTX	36	CVTPL
07	SVPCTX	37	CMPP4
08	CVTPS	38	EDITPC
09	CVTSP	39	MATCHC
0A	INDEX	3A	LOCC
0B	CRC	3B	SKPC
0C	PROBER	3C	MOVZWL
0D	PROBEW	3D	ACBW
0E	INSQUE	3E	MOVAW
0F	REMQUE	3F	PUSHAW
10	BSBB	40	ADDF2
11	BRB	41	ADDF3

(continued on next page)

Table D–2 (Cont.) One_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
12	BNEQ, BNEQU	42	SUBF2
13	BEQL, BEQLU	43	SUBF3
14	BGTR	44	MULF2
15	BLEQ	45	MULF3
16	JSB	46	DIVF2
17	JMP	47	DIVF3
18	BGEQ	48	CVTFB
19	BLSS	49	CVTFW
1A	BGTRU	4A	CVTFL
1B	BLEQU	4B	CVTRFL
1C	BVC	4C	CVTBF
1D	BVS	4D	CVTWF
1E	BGEQU, BCC	4E	CVTLF
1F	BLSSU, BCS	4F	ACBF
20	ADDP4	50	MOVF
21	ADDP6	51	CMPF
22	SUBP4	52	MNEGF
23	SUBP6	53	TSTF
24	CVTPT	54	EMODF
25	MULP	55	POLYF
26	CVTTP	56	CVTFD
27	DIVP	57	Reserved to Digital
28	MOVC3	58	ADAWI
29	CMPC3	59	Reserved to Digital
2A	SCANC	5A	Reserved to Digital
2B	SPANC	5B	Reserved to Digital
2C	MOVC5	5C	INSQHI
2D	CMPC5	5D	INSQTI
2E	MOVTC	5E	REMQHI
2F	MOVTUC	5F	REMQTI
60	ADDD2	90	MOVB
61	ADDD3	91	CMPB

(continued on next page)

Table D–2 (Cont.) One_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
62	SUBD2	92	MCOMB
63	SUBD3	93	BITB
64	MULD2	94	CLRB
65	MULD3	95	TSTB
66	DIVD2	96	INCB
67	DIVD3	97	DECB
68	CVTDB	98	CVTBL
69	CVTDW	99	CVTBW
6A	CVTDL	9A	MOVZBL
6B	CVTRDL	9B	MOVZBW
6C	CVTBD	9C	ROTL
6D	CVTWD	9D	ACBB
6E	CVTLD	9E	MOVAB
6F	ACBD	9F	PUSHAB
70	MOVD	A0	ADDW2
71	CMPD	A1	ADDW3
72	MNEGD	A2	SUBW2
73	TSTD	A3	SUBW3
74	EMODD	A4	MULW2
75	POLYD	A5	MULW3
76	CVTDF	A6	DIVW2
77	Reserved to Digital	A7	DIVW3
78	ASHL	A8	BISW2
79	ASHQ	A9	BISW3
7A	EMUL	AA	BICW2
7B	EDIV	AB	BICW3
7C	CLRQ, CLRD, CLRG	AC	XORW2
7D	MOVQ	AD	XORW3
7E	MOVAQ, MOVAD, MOVAG	AE	MNEGW
7F	PUSHAQ, PUSHAD, PUSHAG	AF	CASEW
80	ADDB2	B0	MOVW
81	ADDB3	B1	CMPW
82	SUBB2	B2	MCOMW
83	SUBB3	B3	BITW
84	MULB2	B4	CLRW

(continued on next page)

Table D–2 (Cont.) One_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
85	MULB3	B5	TSTW
86	DIVB2	B6	INCW
87	DIVB3	B7	DECW
88	BISB2	B8	BISPSW
89	BISB3	B9	BICPSW
8A	BICB2	BA	POPR
8B	BICB3	BB	PUSHR
8C	XORB2	BC	CHMK
8D	XORB3	BD	CHME
8E	MNEGB	BE	CHMS
8F	CASEB	BF	CHMU
C0	ADDL2	E0	BBS
C1	ADDL3	E1	BBC
C2	SUBL2	E2	BBSS
C3	SUBL3	E3	BBCS
C4	MULL2	E4	BBSC
C5	MULL3	E5	BBCC
C6	DIVL2	E6	BBSSI
C7	DIVL3	E7	BBCCI
C8	BISL2	E8	BLBS
C9	BISL3	E9	BLBC
CA	BICL2	EA	FFS
CB	BICL3	EB	FFC
CC	XORL2	EC	CMPV
CD	XORL3	ED	CMPZV
CE	MNEGL	EE	EXTV
CF	CASEL	EF	EXTZV
D0	MOVL	F0	INSV
D1	CMPL	F1	ACBL
D2	MCOML	F2	AOBLSS
D3	BITL	F3	AOBLEQ
D4	CLRL, CLRF	F4	SOBGEQ
D5	TSTL	F5	SOBGTR
D6	INCL	F6	CVTLB
D7	DECL	F7	CVTLW

(continued on next page)

Table D–2 (Cont.) One_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
D8	ADWC	F8	ASHP
D9	SBWC	F9	CVTLP
DA	MTPR	FA	CALLG
DB	MFPR	FB	CALLS
DC	MOVPSL	FC	XFC
DD	PUSHL	FD	ESCD to Digital
DE	MOVAL, MOVA	FE	ESCE to Digital
DF	PUSHAL, PUSHAF	FF	ESCF to Digital

Table D–3 Two_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
00FD	Reserved to Digital	30FD	Reserved to Digital
01FD	Reserved to Digital	31FD	MFVP
02FD	Reserved to Digital	32FD	CVTDH
03FD	Reserved to Digital	33FD	CVTGF
04FD	Reserved to Digital	34FD	VLDL
05FD	Reserved to Digital	35FD	VGATHL
06FD	Reserved to Digital	36FD	VLDQ
07FD	Reserved to Digital	37FD	VGATHQ
08FD	Reserved to Digital	38FD	Reserved to Digital
09FD	Reserved to Digital	39FD	Reserved to Digital
0AFD	Reserved to Digital	3AFD	Reserved to Digital
0BFD	Reserved to Digital	3BFD	Reserved to Digital
0CFD	Reserved to Digital	3CFD	Reserved to Digital
0DFD	Reserved to Digital	3DFD	Reserved to Digital
0EFD	Reserved to Digital	3EFD	Reserved to Digital
0FFD	Reserved to Digital	3FFD	Reserved to Digital
10FD	Reserved to Digital	40FD	ADDG2
11FD	Reserved to Digital	41FD	ADDG3
12FD	Reserved to Digital	42FD	SUBG2
13FD	Reserved to Digital	43FD	SUBG3
14FD	Reserved to Digital	44FD	MULG2
15FD	Reserved to Digital	45FD	MULG3
16FD	Reserved to Digital	46FD	DIVG2

(continued on next page)

Table D–3 (Cont.) Two_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
17FD	Reserved to Digital	47FD	DIVG3
18FD	Reserved to Digital	48FD	CVTGB
19FD	Reserved to Digital	49FD	CVTGW
1AFD	Reserved to Digital	4AFD	CVTGL
1BFD	Reserved to Digital	4BFD	CVTRGL
1CFD	Reserved to Digital	4CFD	CVTBG
1DFD	Reserved to Digital	4DFD	CVTWG
1EFD	Reserved to Digital	4EFD	CVTLG
1FFD	Reserved to Digital	4FFD	ACBG
20FD	Reserved to Digital	50FD	MOVG
21FD	Reserved to Digital	51FD	CMPG
22FD	Reserved to Digital	52FD	MNEGG
23FD	Reserved to Digital	53FD	TSTG
24FD	Reserved to Digital	54FD	EMODG
25FD	Reserved to Digital	55FD	POLYG
26FD	Reserved to Digital	56FD	CVTGH
27FD	Reserved to Digital	57FD	Reserved to Digital
28FD	Reserved to Digital	58FD	Reserved to Digital
29FD	Reserved to Digital	59FD	Reserved to Digital
2AFD	Reserved to Digital	5AFD	Reserved to Digital
2BFD	Reserved to Digital	5BFD	Reserved to Digital
2CFD	Reserved to Digital	5CFD	Reserved to Digital
2DFD	Reserved to Digital	5DFD	Reserved to Digital
2EFD	Reserved to Digital	5EFD	Reserved to Digital
2FFD	Reserved to Digital	5FFD	Reserved to Digital
60FD	ADDH2	90FD	Reserved to Digital
61FD	ADDH3	91FD	Reserved to Digital
62FD	SUBH2	92FD	Reserved to Digital
63FD	SUBH3	93FD	Reserved to Digital
64FD	MULH2	94FD	Reserved to Digital
65FD	MULH3	95FD	Reserved to Digital
66FD	DIVH2	96FD	Reserved to Digital
67FD	DIVH3	97FD	Reserved to Digital
68FD	CVTHB	98FD	CVTFH
69FD	CVTHW	99FD	CVTFG

(continued on next page)

Table D–3 (Cont.) Two_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
6AFD	CVTHL	9AFD	Reserved to Digital
6BFD	CVTRHL	9BFD	Reserved to Digital
6CFD	CVTBH	9CFD	VSTL
6DFD	CVTWH	9DFD	VSCATL
6EFD	CVTLH	9EFD	VSTQ
6FFD	ACBH	9FFD	VSCATQ
70FD	MOVH	A0FD	VVMULL
71FD	CMPH	A1FD	VSMULL
72FD	MNEGH	A2FD	VVMULG
73FD	TSTH	A3FD	VSMULG
74FD	EMODH	A4FD	VVMULF
75FD	POLYH	A5FD	VSMULF
76FD	CVTHG	A6FD	VVMULD
77FD	Reserved to Digital	A7FD	VSMULD
78FD	Reserved to Digital	A8FD	VSYNC
79FD	Reserved to Digital	A9FD	MTVP
7AFD	Reserved to Digital	AAFD	VVDIVG
7BFD	Reserved to Digital	ABFD	VSDIVG
7CFD	CLRH, CLRO	ACFD	VVDIVF
7DFD	MOVO	ADFD	VSDIVF
7EFD	MOVAH, MOVAO	AEFD	VVDIVD
7FFD	PUSHAH, PUSHAO	AFFD	VSDIVD
80FD	VVADDL	B0FD	Reserved to Digital
81FD	VSADDL	B1FD	Reserved to Digital
82FD	VVADDG	B2FD	Reserved to Digital
83FD	VSADDG	B3FD	Reserved to Digital
84FD	VVADDF	B4FD	Reserved to Digital
85FD	VSADDF	B5FD	Reserved to Digital
86FD	VVADDD	B6FD	Reserved to Digital
87FD	VSADDD	B7FD	Reserved to Digital
88FD	VVSUBL	B8FD	Reserved to Digital
89FD	VSSUBL	B9FD	Reserved to Digital
8AFD	VVSUBG	BAFD	Reserved to Digital
8BFD	VSSUBG	BBFD	Reserved to Digital
8CFD	VVSUBF	BCFD	Reserved to Digital

(continued on next page)

Table D–3 (Cont.) Two_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
8DFD	VSSUBF	BDFD	Reserved to Digital
8EFD	VVSUBD	BEFD	Reserved to Digital
8FFD	VSSUBD	BFFD	Reserved to Digital
C0FD	VVCMPL	E0FD	VVSRLL
C1FD	VSCMPL	E1FD	VSSRLL
C2FD	VVCMPG	E2FD	Illegal Vector Opcode
C3FD	VSCMPG	E3FD	Illegal Vector Opcode
C4FD	VVCMPF	E4FD	VVSLLL
C5FD	VSCMPF	E5FD	VSSLLL
C6FD	VVCMPD	E6FD	Illegal Vector Opcode
C7FD	VSCMPD	E7FD	Illegal Vector Opcode
C8FD	VVBISL	E8FD	VVXORL
C9FD	VSBISL	E9FD	VSXORL
CAFD	Illegal Vector Opcode	EAFD	Illegal Vector Opcode
CBFD	Illegal Vector Opcode	EBFD	Illegal Vector Opcode
CCFD	VVBICL	ECFD	VVCVT
CDFD	VSBICL	EDFD	IOTA
CEFD	Illegal Vector Opcode	EEFD	VVMERGE
CFFD	Illegal Vector Opcode	EFFD	VSMERGE
D0FD	Reserved to Digital	F0FD	Reserved to Digital
D1FD	Reserved to Digital	F1FD	Reserved to Digital
D2FD	Reserved to Digital	F2FD	Reserved to Digital
D3FD	Reserved to Digital	F3FD	Reserved to Digital
D4FD	Reserved to Digital	F4FD	Reserved to Digital
D5FD	Reserved to Digital	F5FD	Reserved to Digital
D6FD	Reserved to Digital	F6FD	CVTHF
D7FD	Reserved to Digital	F7FD	CVTHD
D8FD	Reserved to Digital	F8FD	Reserved to Digital
D9FD	Reserved to Digital	F9FD	Reserved to Digital
DAFD	Reserved to Digital	FAFD	Reserved to Digital
DBFD	Reserved to Digital	FBFD	Reserved to Digital
DCFD	Reserved to Digital	FCFD	Reserved to Digital
DDFD	Reserved to Digital	FCFE	Reserved to Digital

(continued on next page)

Table D–3 (Cont.) Two_Byte Opcodes (Numeric Order)

Hex Value	Mnemonic	Hex Value	Mnemonic
DEFD	Reserved to Digital	FCFF	Reserved to Digital
DFFD	Reserved to Digital	FDFF	BUGL
		FEFF	BUGW
		FFFF	Reserved for all time

E Exceptions That May Occur During Instruction Execution

Exceptions can be grouped into the following six classes:

- Arithmetic traps and faults
- Memory management exceptions
- Exceptions detected during operand reference
- Tracing
- Serious system failures

E.1 Arithmetic Traps and Faults

This section contains the descriptions of the exceptions that occur as the result of performing an arithmetic or conversion operation. They are mutually exclusive and are all assigned the same vector in the system control block (SCB) and the same signal "reason" code. Each exception indicates that an instruction has been completed (trap) or backed up (fault). An appropriate distinguishing exception type code is pushed onto the stack as a longword. Table E–1 lists the arithmetic exception type codes.

Table E–1 Arithmetic Exception Type Codes

Exception Type	Mnemonic	Decimal Value	Hexa-decimal Value
Traps			
integer overflow	SS$_INTOVF	1	1
integer divide-by-zero	SS$_INTDIV	2	2
floating overflow	SS$_FLTOVF	3	3
floating or decimal divide-by-zero	SS$_FLTDIV	4	4
floating underflow	SS$_FLTUND	5	5
decimal overflow	SS$_DECOVF	6	6
subscript range	SS$_SUBRNG	7	7
Faults			
floating underflow	SS$_FLTOVF_F	8	8
floating divide-by-zero	SS$_FLTDIV_F	9	9
floating underflow	SS$_FLTUND_F	10	A

(continued on next page)

Exceptions That May Occur During Instruction Execution

E.1 Arithmetic Traps and Faults

E.1.1 Integer Overflow Trap

An integer overflow trap is an exception indicating that the last instruction executed had an integer overflow, which set the processor status longword (PSL) V bit, and that the integer overflow was enabled (the IV bit in the PSL was set). The stored result is the low-order part of the correct result. The N and Z bits in the PSL are set according to the stored result. The type code pushed onto the stack is 1 (SS$_INTOVF).

E.1.2 Integer Divide-by-Zero Trap

An integer divide-by-zero trap is an exception indicating that the last instruction executed had an integer zero divisor. The stored result is equal to the dividend, and condition code V bit in the PSL is set. The type code pushed onto the stack is 2 (SS$_INTDIV).

E.1.3 Floating Overflow Trap

A floating overflow trap is an exception indicating that the last instruction executed resulted in an exponent greater than the largest representable exponent for the data type after normalization and rounding. The stored result contains a one in the sign field and zeros in the exponent and fraction fields. This is a reserved operand. It causes a reserved operand fault if used in a subsequent floating point instruction. The N and V condition code bits in the PSL are set, and the Z and C bits in the PSL are cleared. The type code pushed onto the stack is 3 (SS$_FLTOVF).

E.1.4 Divide-by-Zero Trap

A floating divide-by-zero trap is an exception indicating that the last instruction executed had a floating zero divisor. The stored result is the reserved operand described previously for the floating overflow trap. The condition codes are set as they are for the floating overflow trap.

A decimal string divide-by-zero trap is an exception indicating that the last instruction executed had a decimal-string zero divisor. The destination, R0 to R5, and condition codes are UNPREDICTABLE. The zero divisor can be either +0 or -0.

The type code pushed onto the stack for both types of divide-by-zero is 4 (SS$_FLTDIV).

E.1.5 Floating Underflow Trap

A floating underflow trap is an exception indicating that the last instruction executed resulted in an exponent less than the smallest representable exponent for the data type after normalization and rounding, and that floating underflow was enabled (FU set). The stored result is zero. The N, V, and C condition codes bits in the PSL are cleared, and the Z bit in the PSL is set, except for the polynomial evaluation instruction POLYx. In POLYx, the trap occurs on completion of the instruction, which may be many operations after the underflow. The condition codes are set on the final result in POLYx. The type code pushed onto the stack is 5 (SS$_FLTUND).

E.1.6 Decimal String Overflow Trap

A decimal string overflow trap is an exception indicating that the last instruction executed had a decimal-string result too large for the destination string provided, and that decimal overflow was enabled (the DV bit in the PSL was set). The V condition code bit in the PSL is always set. The type code pushed onto the stack is 6 (SS$_DECOVF).

E.1.7 Subscript-Range Trap

A subscript range trap is an exception indicating that the last instruction was an INDEX instruction with a subscript operand that failed the range check. The value of the subscript operand is lower than the low operand or greater than the high operand. The result is stored in indexout, and the condition codes are set as if the subscript were within range. The type code pushed onto the stack is 7 (SS$_SUBRNG).

E.1.8 Floating Overflow Fault

A floating overflow fault is an exception indicating that the last instruction executed resulted in an exponent greater than the largest representable exponent for the data type after normalization and rounding. The destination was unaffected, and the saved condition codes are UNPREDICTABLE. The saved program counter (PC) points to the instruction causing the fault. The POLYx instruction is suspended with the first-part-done bit (FPD) set. The type code pushed onto the stack is 8 (SS$_FLTOVF_F).

E.1.9 Divide-by-Zero Floating Fault

A floating divide-by-zero fault is an exception indicating that the last instruction executed had a floating zero divisor. The quotient operand was unaffected and the saved condition codes are UNPREDICTABLE. The saved PC points to the instruction causing the fault. The type code pushed onto the stack is 9 (SS$_FLTDIV_F).

E.1.10 Floating Underflow Fault

A floating underflow fault is an exception indicating that the last instruction executed resulted in an exponent less than the smallest representable exponent for the data type after normalization and rounding, and that floating underflow was enabled (the FU bit was set). The destination operand is unaffected. The saved condition codes are UNPREDICTABLE. The saved PC points to the instruction causing the fault. The POLYx instruction is suspended with FPD set. The type code pushed onto the stack is 10 (SS$_FLTUND_F).

E.2 Memory Management Exceptions

A memory management exception can be either an access control violation fault or a translation not valid fault.

E.2.1 Access Control Violation Fault

An access control violation fault is an exception indicating that the process attempted a reference not allowed at the current access mode.

E.2.2 Translation Not Valid Fault

A translation not valid fault is an exception indicating that the process attempted a reference to a page for which the valid bit in the page table had not been set.

Note that if a process attempts to reference a page for which the page table entry specifies both translation not valid fault and access control violation, an access control violation fault occurs.

E.3 Exceptions Detected During Operand Reference

Two exceptions are possible during operand reference: the reserved addressing mode fault and the reserved operand exception.

E.3.1 Reserved Addressing Mode Fault

A reserved addressing mode fault is an exception indicating that an operand specifier attempted to use an addressing mode that is disallowed. No parameters are pushed.

E.3.2 Reserved Operand Exception

A reserved operand exception is an exception indicating that an accessed operand has a format reserved for future use by Digital. No parameters are pushed onto the stack. This exception always backs up the saved PC to point to the opcode. The exception service routine may determine the type of operand by examining the opcode using the saved PC.

Note that only the changes made by instruction fetch and the changes made because of operand specifier evaluation may be restored. Therefore, some instructions are not restartable. These exceptions are labeled as aborts rather than as faults. The saved PC is always restored properly unless the instruction attempted to modify it in a manner that results in UNPREDICTABLE results.

The reserved operand exceptions are caused by the following:

- Bit field too wide
- Invalid combination of bits in PSL restored by the return from interrupt (REI) instruction (fault)
- Invalid combination of bits in PSW mask longword during a return from procedure (RET) instruction (fault)
- Invalid combination of bits in the bit set PSW (BISPSW) or bit clear PSW (BICPSW) instructions (fault)
- Invalid call procedure with stack argument list (CALLS) or call procedure with general argument list (CALLG) instructions entry mask (fault)
- Invalid register number in the move from processor register (MFPR) instruction or move to processor register (MTPR) instruction (fault)
- Invalid PCB contents in the load processor context (LDPCTX) instruction for some implementations (abort)
- Unaligned operand in the add aligned word interlocked (ADAWI) instruction (fault)
- Invalid register contents in the move to processor register (MTPR) instruction for some implementations (fault)
- Invalid operand addresses in insert and remove queue interlocked (INSQHI, INSQTI, REMQHI, or REMQTI) instructions (fault)
- A floating point number that has the sign bit set and the exponent zero in the polynomial evaluation (POLY) instruction table (fault)
- POLY degree too large (fault)
- Decimal string too long (abort)
- Invalid digit in convert trailing numeric to packed (CVTTP) or convert separate numeric to packed (CVTSP) instructions (abort)
- Reserved pattern operator in the edit packed to character string (EDITPC) instruction (fault)
- Incorrect source string length at completion of EDITPC (abort)

E.4 Exceptions Occurring as the Consequence of an Instruction

The following exceptions may occur as a consequence of instruction execution:

- Reserved or privileged instruction fault
- Opcode reserved to customers fault
- Instruction emulation exceptions
- Compatibility mode exception
- Change mode trap
- Breakpoint fault

Each is described in the following subsections.

E.4.1 Reserved or Privileged Instruction Fault

A reserved or privileged instruction fault occurs when the processor encounters an opcode that is not specifically defined or requires higher privileges than the current mode. No parameters are pushed onto the stack. Opcode FFFF (hex) will always fault.

E.4.2 Operand Reserved to Customers Fault

An opcode reserved to customers fault is an exception that occurs when an opcode reserved to customers is executed. The operation is identical to the reserved or privileged instruction fault, except that the event is caused by a different set of opcodes and faults through a different vector. All opcodes reserved to customers start with FC (hex), which is the XFC instruction. If the special instruction must generate a unique exception, one of the reserved-to-customer vectors should be used. An example might be an unrecognized second byte of the instruction.

The XFC fault is intended primarily for use with writable control store to implement installation-dependent instructions. The method used to enable and disable the handling of an XFC fault in user-written microcode is implementation dependent. Some implementations may transfer control to microcode without checking bits <1:0> of the exception vector.

E.4.3 Instruction Emulation Exceptions

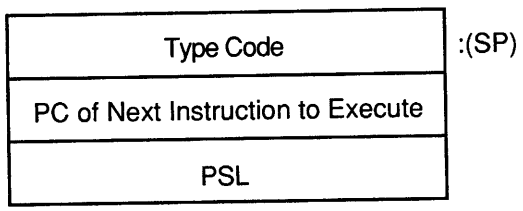
When a subset processor executes a string instruction that is omitted from its instruction set, an emulation exception results. An emulation exception can occur through either of two system control block (SCB) vectors, depending on whether or not the first-part-done (FPD) bit in the program status longword was set at the beginning of the instruction. If the FPD bit is clear, a subset emulation trap occurs through the SCB vector at offset CB (hex), and a subset emulation trap frame is pushed onto the current stack. If the FPD bit is set, a suspended emulation fault

occurs through the SCB vector at offset CC (hex), and the PC and the PSL are pushed onto the current stack.

E.4.4 Compatibility Mode Exception

A compatibility mode exception is an exception that occurs when the processor is in compatibility mode. A longword of information containing a code that indicates the exception type is pushed onto the stack. Figure E–1 shows the stack frame, which is the same as that for arithmetic exceptions.

Figure E–1 Compatibility Mode Exception Stack Frame



ZK–6351–GE

The compatibility type codes are shown in Table E–2.

Table E–2 Compatibility Mode Exception Type Codes

Exception Type	Decimal Value
Faults	
reserved opcode	0
BPT instruction	1
IOT instruction	2
EMT instruction	3
TRAP instruction	4
illegal instruction	5
Aborts	
odd address	6

All other exceptions in compatibility mode, including the access control violation fault, the translation not valid fault, and the machine check abort, occur by means of the regular native-mode vector.

E.4.5 Change Mode Trap

A change mode trap is an exception occurring when one of the change mode instructions (CHMK, CHME, CHMS, or CHMU) is executed. The instruction operand is pushed onto the exception stack.

E.4.6 Breakpoint Fault

A breakpoint fault is an exception that occurs when the breakpoint instruction (BPT) is executed. The BPT instruction pushes the current PSL onto the stack.

To proceed from a breakpoint fault, a debugger or tracing program does the following:

1 Restores the original contents of the location containing the BPT instruction.

2 Sets the T bit in the PSL saved by the BPT fault. The PSL is on the stack.

3 Resumes operation of the main instruction stream.

When the instruction that has a breakpoint completes execution, a *trace exception* occurs. At this point, the tracing program takes control and does the following:

1 Reinserts the BPT instruction.

2 Restores the T bit to its original state (usually zero).

3 Resumes operation of the main instruction stream.

Note that if both tracing and breakpointing are in progress (if the PSL T bit was set at the time of the BPT), both the BPT restoration and a normal trace exception should be processed on the trace exception by the trace handler.

E.5 Trace Fault

Program tracing is used for many purposes. Debugging programs and evaluating program performance are the most common uses of program tracing.

A trace fault is an exception that occurs between instructions when trace is enabled. One trace fault occurs before the execution of each traced instruction. The address in the PC saved when a trace fault occurs is the address of the instruction after the trace fault that would normally be executed. The trace exception for an instruction takes precedence over all other exceptions. The detection of reserved instruction faults occurs after the trace fault. If a trace fault and a memory management fault (or an odd address abort during a compatibility mode instruction fetch) occur simultaneously, exceptions are taken in UNPREDICTABLE order.

To ensure that exactly one trace occurs per instruction despite other traps and faults, the PSL contains the trace enable (T) and trace pending (TP) bits.

The PSL TP bit generates a fault before any other processing at the start of the next instruction.

The following are rules of operation for trace:

1 At the beginning of an instruction, if the trace pending (TP) bit is set, it is cleared and a trace fault is taken.

2 The value of the trace enable (T) bit is loaded into the trace pending (TP) bit.

3 The detection of interrupts and other exceptions can occur during instruction execution. In this case, TP is cleared before the exception or interrupt is initiated. The system saves the entire PSL including the T bit and TP bit on interrupt or exception initiation and restores the PSL at the end with an REI. This makes interrupts and benign exceptions totally transparent to the executing program.

   The following are conditions and results that might occur during instruction execution or before the next instruction:

   a. If the instruction faults or an interrupt is serviced, the PSL TP bit is cleared before the PSL is saved on the stack. The saved PC (the next lower word on the stack after the saved PSL) is set to the start of the faulting or interrupted instruction. Instruction execution is resumed at step 1.

   b. If the instruction aborts or takes an arithmetic trap, the PSL TP bit is not changed before the PSL is saved on the stack.

   c. If an interrupt is serviced after instruction completion and arithmetic traps but before the presence of tracing is checked at the start of the next instruction, the PSL TP bit is not changed before the PSL is saved on the stack.

E.5.1 Trace Operation When Entering a Change Mode Instruction

The routine entered by a change mode (CHMx) instruction is not traced because change mode clears T and TP in the new PSL that is used for whichever new mode is entered. However, if the T bit was set in the old PSW (the one to be saved) at the beginning of the change mode instruction, the system sets both the T and the TP bit in the saved PSL. Trace faults resume with the instruction that follows other returns from interrupt (REI) in the routine entered by the CHMx instruction. An instruction following an REI faults if T was set when the REI was executed, or if the TP bit in the saved PSL is set. In both cases, TP is set after the REI.

E.5.2 Trace Operation Upon Return From Interrupt

Note that a trace fault that occurs for an instruction following an REI instruction that had set the TP will be taken with the new PSL restored by the REI instruction. Thus, special care must be taken if exception or interrupt routines are traced.

E.5.3 Trace Operation After a BISPSW Instruction

If the T bit is set by a BISPSW instruction, trace faults begin with the second instruction after the BISPSW.

E.5.4 Trace Operation After a CALLS or CALLG Instruction

The CALLS and CALLG instructions save a clear T bit, although the T bit in the PSL is unchanged. This is done so that a debugger or trace program proceeding from a BPT fault does not get a spurious trace from the RET that matches the CALL.

E.6 Serious System Failures

The following are possible serious system failures:

- Kernel stack not valid abort
- Interrupt stack not valid halt
- Machine check exception

These system failures are described in the following sections.

E.6.1 Kernel Stack Not Valid Abort

The kernel stack not valid abort is an exception indicating that the kernel stack was not valid while the processor was pushing information onto it during the initiation of an exception or interrupt. This is usually an indication of a stack overflow or other operating system error. During this process, the attempted exception is transformed into an abort that uses the interrupt stack. Only the PSL and PC of the original exception are pushed onto the interrupt stack. The interrupt priority level (IPL) is raised to 1F (hex). If the exception vector bits <1:0> are not both 1, the operation of the processor is UNDEFINED.

Software can abort the process without aborting the system. However, because of the lost information, the process cannot be continued. If the kernel stack is not valid during the normal execution of an instruction (including CHMx or REI), the normal memory management fault is initiated.

E.6.2 Interrupt Stack Not Valid Halt

An interrupt stack not valid halt results when the interrupt stack was not valid or a memory error occurred while the processor was pushing information onto the interrupt stack during the initiation of an exception or interrupt. No further interrupt requests are acknowledged on the processor. The processor leaves the PC, the PSL, and the reason for the halt in registers so that they are available to a debugger, to the normal bootstrap routine, or to an optional watchdog bootstrap routine. A watchdog bootstrap routine can cause the processor to leave the halted state.

E.6.3 Machine Check Exception

A machine check exception indicates that the processor detected an internal error. As is usual for exceptions, a machine check is taken regardless of current interrupt priority level (IPL). The machine check exception vector (bits 0 to 1) must specify 1 or the operation of the processor is UNDEFINED. The exception is taken on the interrupt stack, and IPL is raised to 1F (hex).

The processor pushes a machine check stack frame onto the interrupt stack, consisting of a count longword, an implementation-dependent number of error report longwords, and a PC, and a PSL. The count longword reports the number of bytes of error report pushed. For example, if 4 longwords of error report are pushed, the count longword will contain 16 (decimal).

Software can decide, on the basis of the information presented, whether to abort the current process if the machine check came from the process. The machine check includes any uncorrected bus and memory errors and any other processor-detected errors. Some processor errors cannot ensure the state of the machine at all. For such errors, the state is preserved as well as possible, given the circumstances.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal[1]	————	USASSB Order Processing - WMO/E15 *or* U.S. Area Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

VAX MACRO and Instruction Set Reference Manual
AA–LA89B–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

-- Do Not Tear - Fold Here and Tape ----------

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-- Do Not Tear - Fold Here ----------

Cut Along Dotted Line

Reader's Comments

VAX MACRO and Instruction Set Reference Manual
AA–LA89B–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

-- Do Not Tear - Fold Here and Tape ----------

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-- Do Not Tear - Fold Here ----------

Cut Along Dotted Line