AV-D827C-TE

VAX-11
Programming Card

digital

135

AV-D827C-TE

VAX-11
Programming Card

Prepared by Educational Services of
Digital Equipment Corporation

© Digital Equipment Corporation 1983
All Rights Reserved

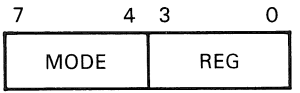
The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors which may appear in this document.

Printed in U.S.A.

VAX, VAX/VMS, PDP, and DEC are trademarks of Digital Equipment Corporation.

SUMMARY OF GENERAL MODE ADDRESSING

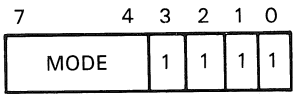
General Register Addressing



Hex	Dec	Name	Assembler	r	m	w	a	v	PC	SP	AP& FP	Indexable
0-3	0-3	literal	S^#literal	y	f	f	f	f	–	–	–	f
4	4	indexed	i[Rx]	y	y	y	y	y	f	y	y	f
5	5	register	Rn	y	y	y	f	y	u	uq	uo	f
6	6	register deferred	(Rn)	y	y	y	y	y	u	y	y	y
7	7	autodecrement	–(Rn)	y	y	y	y	y	u	y	y	ux
8	8	autoincrement	(Rn)+	y	y	y	y	y	p	y	y	ux

Hex	Dec	Name	Assembler	r	m	w	a	v	PC	SP	AP& FP	Indexable
9	9	autoincrement deferred	@(Rn)+	y	y	y	y	y	p	y	y	ux
A	10	byte displacement	B^D(Rn)	y	y	y	y	y	p	y	y	y
B	11	byte displacement deferred	@B^D(Rn)	y	y	y	y	y	p	y	y	y
C	12	word displacement	W^D(Rn)	y	y	y	y	y	p	y	y	y
D	13	word displacement deferred	@W^D(Rn)	y	y	y	y	y	p	y	y	y
E	14	longword displacement	L^D(Rn)	y	y	y	y	y	p	y	y	y
F	15	longword displacement deferred	@L^D(Rn)	y	y	y	y	y	p	y	y	y

Program Counter Addressing (reg = 15)



Hex	Dec	Name	Assembler	r	m	w	a	v	Indexable?
8	8	immediate	I^#constant	y	u	u	y	y	u
9	9	absolute	@#address	y	y	y	y	y	y
A	10	byte relative	B^address	y	y	y	y	y	y
B	11	byte relative deferred	@B^address	y	y	y	y	y	y
C	12	word relative	W^address	y	y	y	y	y	y
D	13	word relative deferred	@W^address	y	y	y	y	y	y
E	14	long word relative	L^address	y	y	y	y	y	y
F	15	long word relative deferred	@L^address	y	y	y	y	y	y

Key:

D	–	displacement
i	–	any indexable addressing mode
–	–	logically impossible
f	–	reserved addressing mode fault
p	–	Program Counter addressing
u	–	UNPREDICTABLE
uq	–	UNPREDICTABLE for quad, octa, D_floating, G_floating, and H_floating (and field if position + size greater than 32)
uo	–	UNPREDICTABLE for octa, and H format
ux	–	UNPREDICTABLE for index register same as base register
y	–	yes, always valid addressing mode
r	–	read access
m	–	modify access
w	–	write access
a	–	address access
v	–	field access
Rn	–	general register, n = 0–15
Rx	–	general register, x = 0–14

OPERAND SPECIFIER NOTATION

Operand specifiers are described in the following way:

<name>.<access type><data type>

where:

1. Name is a suggestive name for the operand in the context of the instruction. The name is often abbreviated.

2. Access type is a letter denoting the operand specifier access type:

   a – Calculate the effective address of the specified operand. Address is returned in a longword which is the actual instruction operand. Context of address calculation is given by <data type>; i.e. size to be used in autoincrement, autodecrement, and indexing.

   b – No operand reference. Operand specifier is a branch displacement. Size of branch displacement is given by <data type>.

m – Operand is read, potentially modified and written. Note that this is NOT an indivisible memory operation. Also note that if the operand is not actually modified, it may not be written back. However, modify type operands are always checked for both read and write accessibility.

r – Operand is read only.

v – Calculate the effective address of the specified operand. If the effective address is in memory, the address is returned in a longword which is the actual instruction operand. Context of address calculation is given by <data type>. If the effective address is Rn, the operand is in Rn or R[n + 1]'Rn.

w – Operand is written only.

3. Data type is a letter denoting the data type of the operand:

b – byte	o – octaword
d – D_floating	q – quadword
f – F_floating	w – word
g – G_floating	x – first data type specified by instruction
h – H_floating	y – second data type specified by instruction
i – longword	* – multiple longwords (used only on implied operands)

4. Implied operands, that is, locations accessed by the instruction but not specified in and operand, are denoted in enclosing brackets [ ].

Condition Codes Legend

. = conditionally cleared/set
– = not affected
0 = cleared
1 = set

OP	Mnemonic	Description	Arguments	Cond. Codes N Z V C
9D	ACBB	Add compare and branch byte	limit.rb, add.rb, index.mb, displ.bw	. . . -
6F	ACBD	Add compare and branch D_floating	limit.rd, add.rd, index.md, displ.bw	. . . -
4F	ACBF	Add compare and branch F_floating	limit.rf, add.rf, index.mf, displ.bw	. . . -
4FFD	ACBG	Add compare and branch G_floating	limit.rg, add.rg, index.mg, displ.bw	. . . -
6FFD	ACBH	Add compare and branch H_floating	limit.rh, add.rh, index.mh, displ.bw	. . . -
F1	ACBL	Add compare and branch long	limit.rl, add.rl, index.ml, displ.bw	. . . -
3D	ACBW	Add compare and branch word	limit.rw, add.rw, index.mw, displ.bw	. . . -
58	ADAWI	Add aligned word interlocked	add.rw, sum.mw	. . . .
80	ADDB2	Add byte 2-operand	add.rb, sum.mb	. . . .
81	ADDB3	Add byte 3-operand	add1.rb, add2.rb, sum.wb	. . . .
60	ADDD2	Add D_floating 2-operand	add.rd, sum.md	. . . 0
61	ADDD3	Add D_floating 3-operand	add1.rd, add2.rd, sum.wd	. . . 0
40	ADDF2	Add F_floating 2-operand	add.rf, sum.mf	. . . 0
41	ADDF3	Add F_floating 3-operand	add1.rf, add2.rf, sum.wf	. . . 0
40FD	ADDG2	Add G_floating 2-operand	add.rg, sum.mg	. . . 0

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
41FD	ADDG3	Add G_floating 3-operand	add1.rg, add2.rg, sum.wg	.	.	.	0
60FD	ADDH2	Add H_floating 2-operand	add.rh, sum.mh	.	.	.	0
61FD	ADDH3	Add H_floating 3-operand	add1.rh, add2.rh, sum.wh	.	.	.	0
C0	ADDL2	Add long 2-operand	add.rl, sum.ml	.	.	.	.
C1	ADDL3	Add long 3-operand	add1.rl, add2.rl, sum.wl	.	.	.	.
20	ADDP4	Add packed 4-operand	addlen.rw, addaddr.ab, sumlen.rw, sumaddr.ab, [R0-3.wl]	.	.	.	0
21	ADDP6	Add packed 6-operand	add1len.rw, add1addr.ab, add2len.rw, add2addr.ab, sumlen.rw, sumaddr.ab, [R0-5.wl]	.	.	.	0
A0	ADDW2	Add word 2-operand	add.rw, sum.mw	.	.	.	.
A1	ADDW3	Add word 3-operand	add1.rw, add2.rw, sum.ww	.	.	.	.
D8	ADWC	Add with carry	add.rl, sum.ml	.	.	.	.
F3	AOBLEQ	Add one and branch on less or equal	limit.rl, index.ml, displ.bb	.	.	.	-
F2	AOBLSS	Add one and branch on less	limit.rl, index.ml, displ.bb	.	.	.	-
78	ASHL	Arithmetic shift long	count.rb, src.rl, dst.wl	.	.	.	0
F8	ASHP	Arithmetic shift and round packed	count.rb, srclen.rw, srcaddr.ab, round.rb, dstlen.rw, dstaddr.ab, [R0-3.wl]	.	.	.	0

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
79	ASHQ	Arithmetic shift quad	count.rb, src.rq, dst.wq	.	.	.	0
E1	BBC	Branch on bit clear	pos.rl, base.vb, displ.bb, [field.rv]	-	-	-	-
E5	BBCC	Branch on bit clear and clear	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E7	BBCCI	Branch on bit clear and clear interlocked	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E3	BBCS	Branch on bit clear and set	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E0	BBS	Branch on bit set	pos.rl, base.vb, displ.bb, [field.rv]	-	-	-	-
E4	BBSC	Branch on bit set and clear	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E2	BBSS	Branch on bit set and set	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E6	BBSSI	Branch on bit set and set interlocked	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
1E	BCC	Branch on carry clear	displ.bb	-	-	-	-
1F	BCS	Branch on carry set	displ.bb	-	-	-	-
13	BEQL	Branch on equal	displ.bb	-	-	-	-
13	BEQLU	Branch on equal unsigned	displ.bb	-	-	-	-
18	BGEQ	Branch on greater or equal	displ.bb	-	-	-	-
1E	BGEQU	Branch on greater or equal unsigned	displ.bb	-	-	-	-
14	BGTR	Branch on greater	displ.bb	-	-	-	-
1A	BGTRU	Branch on greater unsigned	displ.bb	-	-	-	-

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
8A	BICB2	Bit clear byte 2-operand	mask.rb, dst.mb	.	.	0	-
8B	BICB3	Bit clear byte 3-operand	mask.rb, src.rb, dst.wb	.	.	0	-
CA	BICL2	Bit clear long 2-operand	mask.rl, dst.ml	.	.	0	-
CB	BICL3	Bit clear long 3-operand	mask.rl, src.rl, dst.wl	.	.	0	-
B9	BICPSW	Bit clear processor status word	mask.rw	.	.	.	.
AA	BICW2	Bit clear word 2-operand	mask.rw, dst.mw	.	.	0	-
AB	BICW3	Bit clear word 3-operand	mask.rw, src.rw, dst.ww	.	.	0	-
88	BISB2	Bit set byte 2-operand	mask.rb, dst.mb	.	.	0	-
89	BISB3	Bit set byte 3-operand	mask.rb, src.rb, dst.wb	.	.	0	-
C8	BISL2	Bit set long 2-operand	mask.rl, dst.ml	.	.	0	-
C9	BISL3	Bit set long 3-operand	mask.rl, src.rl, dst.wl	.	.	0	-
B8	BISPSW	Bit set processor status word	mask.rw	.	.	.	.
A8	BISW2	Bit set word 2-operand	mask.rw, dst.mw	.	.	0	-
A9	BISW3	Bit set word 3-operand	mask.rw, src.rw, dst.ww	.	.	0	-
93	BITB	Bit test byte	mask.rb, src.rb	.	.	0	-
D3	BITL	Bit test long	mask.rl, src.rl	.	.	0	-
B3	BITW	Bit test word	mask.rw, src.rw	.	.	0	-

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
E9	BLBC	Branch on low bit clear	src.rl, displ.bb	-	-	-	-
E8	BLBS	Branch on low	src.rl, displ.bb	-	-	-	-
15	BLEQ	Branch on less or equal	displ.bb	-	-	-	-
1B	BLEQU	Branch on less or equal unsigned	displ.bb	-	-	-	-
19	BLSS	Branch on less	displ.bb				
1F	BLSSU	Branch on less unsigned	displ.bb	-	-	-	-
12	BNEQ	Branch on not equal	displ.bb	-	-	-	-
12	BNEQU	Branch on not equal unsigned	displ.bb	-	-	-	-
03	BPT	Break point fault	[–(KSP).w*]	0	0	0	0
11	BRB	Branch with byte displacement	displ.bb	-	-	-	-
31	BRW	Branch with word displacement	displ.bw	-	-	-	-
10	BSBB	Branch to subroutine with byte displacement	displ.bb, [–(SP).wl]	-	-	-	-
30	BSBW	Branch to subroutine with word displacement	displ.bw, [–(SP).wl]	-	-	-	-
FDFF	BUGL	VMS bugcheck		0	0	0	0
FEFF	BUGW	VMS bugcheck		0	0	0	0
1C	BVC	Branch on overflow clear	displ.bb	-	-	-	-
1D	BVS	Branch on overflow set	displ.bb	-	-	-	-
FA	CALLG	Call with general argument list	arglist.ab,dst.ab, [–(SP).w*]	0	0	0	0

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
FB	CALLS	Call with argument list on stack	numarg.rl,dst.ab, [–(SP).w*]	0	0	0	0
8F	CASEB	Case byte	selector.rb, base.rb, limit.rb, displ.bw-list	.	.	0	.
CF	CASEL	Case long	selector.rl, base.rl, limit.rl, displ.bw-list	.	.	0	.
AF	CASEW	Case word	selector.rw, base.rw, limit.rw, displ.bw-list	.	.	0	.
BD	CHME	Change mode to executive	param.rw,[–(ySP.w*] y=MINU(E, PSL*current-mode*)	0	0	0	0
BC	CHMK	Change mode to kernel	param.rw,[–(KSP).w*]	0	0	0	0
BE	CHMS	Change mode to supervisor	param.rw,[–(ySP).w*] y=MINU(S, PSL*current-mode*)	0	0	0	0
BF	CHMU	Change mode to user	param.rw, [–(SP).w*]	0	0	0	0
94	CLRB	Clear byte	dst.wb	0	1	0	-
7C	CLRD	Clear D_floating	dst.wd	0	1	0	-
D4	CLRF	Clear F_floating	dst.wf	0	1	0	-
7C	CLRG	Clear G_floating	dst.wg	0	1	0	-
7CFD	CLRH	Clear H_floating	dst.wh	0	1	0	-

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
D4	CLRL	Clear long	dst.wl	0	1	0	-
7CFD	CLRO	Clear octaword	dst.wo	0	1	0	-
7C	CLRQ	Clear quad	dst.wq	0	1	0	-
B4	CLRW	Clear word	dst.ww	0	1	0	-
91	CMPB	Compare byte	src1.rb, src2.rb	.	.	0	.
29	CMPC3	Compare character 3-operand	len.rw, src1addr.ab, src2addr.ab, [R0-3.wl]	.	.	0	.
2D	CMPC5	Compare character 5-operand	src1len.rw, src1addr.ab, fill.rb, src2len.rw, src2addr.ab, [R0-3.wl]	.	.	0	.
71	CMPD	Compare D_floating	src1.rd, src2.rd	.	.	0	0
51	CMPF	Compare F_floating	src1.rf, src2.rf	.	.	0	0
51FD	CMPG	Compare G_floating	src1.rg, src2.rg	.	.	0	0
71FD	CMPH	Compare H_floating	src1.rh, src2.rh	.	.	0	0
D1	CMPL	Compare long	src1.rl, src2.r1	.	.	0	.
35	CMPP3	Compare packed 3-operand	len.rw, src1addr.ab, src2addr.ab, [R0-3.wl]	.	.	0	0
37	CMPP4	Compare packed 4-operand	src1len.rw, src1addr.ab, src2len.rw, src2addr.ab, [R0-3.wl]	.	.	0	0
EC	CMPV	Compare field	pos.rl, size.rb, base.vb, [field.rv], src.rl	.	.	0	.

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
B1	CMPW	Compare word	src1.rw, src2.rw	.	.	0	.
ED	CMPZV	Compare zero-extended field	pos.rl, size.rb, base.vb, [field.rv], src.rl	.	.	0	.
0B	CRC	Calculate cyclic redundancy check	tbl.ab, initialcrc.rl, strlen.rw, stream.ab, [R0-3.wl]	.	.	0	0
6C	CVTBD	Convert byte to D_floating	src.rb, dst.wd	.	.	.	0
4C	CVTBF	Convert byte to F_floating	src.rb, dst.wf	.	.	.	0
4CFD	CVTBG	Convert byte to G_floating	src.rb, dst.wg	.	.	.	0
6CFD	CVTBH	Convert byte to H_floating	src.rb, dst.wh	.	.	.	0
98	CVTBL	Convert byte to long	src.rb, dst.wl	.	.	.	0
99	CVTBW	Convert byte to word	src.rb, dst.ww	.	.	.	0
68	CVTDB	Convert D_floating to byte	src.rd, dst.wb	.	.	.	0
76	CVTDF	Convert D_floating to F_floating	src.rd, dst.wf	.	.	.	0
32FD	CVTDH	Convert D_floating to H_floating	src.rd, dst.wh	.	.	.	0
6A	CVTDL	Convert D_floating to long	src.rd, dst.wl	.	.	.	0
69	CVTDW	Convert D_floating to word	src.rd, dst.ww	.	.	.	0
48	CVTFB	Convert F_floating to byte	src.rf, dst.wb	.	.	.	0
56	CVTFD	Convert F_floating to D_floating	src.rf, dst.wd	.	.	.	0
99FD	CVTFG	Convert F_floating to G_floating	src.rf, dst.wg	.	.	.	0

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
98FD	CVTFH	Convert F_floating to H_floating	src.rf, dst.wh	.	.	.	0
4A	CVTFL	Convert F_floating to long	src.rf, dst.wl	.	.	.	0
49	CVTFW	Convert F_floating to word	src.rf, dst.ww	.	.	.	0
48FD	CVTGB	Convert G_floating to byte	src.rg, dst.wb	.	.	.	0
33FD	CVTGF	Convert G_floating to F_floating	src.rg, dst.wf	.	.	.	0
56FD	CVTGH	Convert G_floating to H_floating	src.rg, dst.wh	.	.	.	0
4AFD	CVTGL	Convert G_floating to longword	src.rg, dst.wl	.	.	.	0
49FD	CVTGW	Convert G_floating to word	src.rg, dst.ww	.	.	.	0
68FD	CVTHB	Convert H_floating to byte	src.rh, dst.wb	.	.	.	0
F7FD	CVTHD	Convert H_floating to D_floating	srd.rh, dst.wd	.	.	.	0
F6FD	CVTHF	Convert H_floating to F_floating	src.rh, dst.wf	.	.	.	0
76FD	CVTHG	Convert H_floating to G_floating	srd.rh, dst.wg	.	.	.	0
6AFD	CVTHL	Convert H_floating to longword	srd.rh, dst.wl	.	.	.	0
69FD	CVTHW	Convert H_floating to word	src.rh, dst.ww	.	.	.	0
F6	CVTLB	Convert long to byte	src.rl, dst.wb	.	.	.	0
6E	CVTLD	Convert long to D_floating	src.rl, dst.wd	.	.	.	0
4E	CVTLF	Convert long to F_floating	src.rl, dst.wf	.	.	.	0
4EFD	CVTLG	Convert longword to G_floating	src.rl, dst.wg	.	.	.	0

OP	Mnemonic	Description	Arguments	Cond. Codes N Z V C
6EFD	CVTLH	Convert longword to H_floating	src.rl, dst.wh	. . . 0
F9	CVTLP	Convert long to packed	src.rl, dstlen.rw, dstaddr.ab, [R0-3.wl]	. . . 0
F7	CVTLW	Convert long to word	src.rl, dst.ww	. . . 0
36	CVTPL	Convert packed to long	srclen.rw, srcaddr.ab, [R0-3.wl], dst.wl	. . . 0
08	CVTPS	Convert packed to leading separate	srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab, [R0-3.wl]	. . . 0
24	CVTPT	Convert packed to trailing	srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab, [R0-3.wl]	. . . 0
6B	CVTRDL	Convert rounded D_floating to long	src.rd, dst.wl	. . . 0
4B	CVTRFL	Convert rounded F_floating to long	src.rf, dst.wl	. . . 0
4BFD	CVTRGL	Convert rounded G_floating to long	src.rg, dst.wl	. . . 0
6BFD	CVTRHL	Convert rounded H_floating to long	src.rh, dst.wl	. . . 0
09	CVTSP	Convert leading separate to packed	srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab, [R0-3.wl]	. . . 0
26	CVTTP	Convert trailing to packed	srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab, [R0-3.wl]	. . . 0
33	CVTWB	Convert word to byte	src.rw, dst.wb	. . . 0
6D	CVTWD	Convert word to D_floating	src.rw, dst.wd	. . . 0

OP	Mnemonic	Description	Arguments	Cond. Codes N Z V C
4D	CVTWF	Convert word to F_floating	src.rw, dst.wf	. . . 0
4DFD	CVTWG	Convert word to G_floating	src.rw, dst.wg	. . . 0
6DFD	CVTWH	Convert word to H_floating	src.rw, dst.wh	. . . 0
32	CVTWL	Convert word to long	src.rw, dst.wl	. . . 0
97	DECB	Decrement byte	dif.mb	. . . .
D7	DECL	Decrement long	dif.ml	. . . .
B7	DECW	Decrement word	dif.mw	. . . .
86	DIVB2	Divide byte 2-operand	divr.rb, quo.mb	. . . 0
87	DIVB3	Divide byte 3-operand	divr.rb, divd.rb, quo.wb	. . . 0
66	DIVD2	Divide D_floating 2-operand	divr.rd, quo.md	. . . 0
67	DIVD3	Divide D_floating 3-operand	divr.rd, divd.rd, quo.wd	. . . 0
46	DIVF2	Divide F_floating 2-operand	divr.rf, quo.mf	. . . 0
47	DIVF3	Divide F_floating 3-operand	divr.rf, divd.rf, quo.wf	. . . 0
46FD	DIVG2	Divide G_floating 2-operand	divr.rg, quo.mg	. . . 0
47FD	DIVG3	Divide G_floating 3-operand	divr.rg, divd.rg, quo.wg	. . . 0
66FD	DIVH2	Divide H_floating 2-operand	divr.rh, quo.mh	. . . 0
67FD	DIVH3	Divide H_floating 3-operand	divr.rh, divd.rh, quo.wh	. . . 0
C6	DIVL2	Divide long 2-operand	divr.rl, quo.ml	. . . 0

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
C7	DIVL3	Divide long 3-operand	divr.rl, divd.rl, quo.wl	.	.	.	0
27	DIVP	Divide packed	divrlen.rw, divraddr.ab, divdlen.rw, divdaddr.ab, quolen.rw, quoaddr.ab, [R0-5.wl, – 16(SP): – 1(SP).wb]	.	.	.	0
A6	DIVW2	Divide word 2-operand	divr.rw, quo.mw	.	.	.	0
A7	DIVW3	Divide word 3-operand	divr.rw, divd.rw, quo.ww	.	.	.	0
38	EDITPC	Edit packed to character string	srclen.rw, srcaddr.ab, pattern.ab, dstaddr.ab, [R0-5.wl]	.	.	.	.
7B	EDIV	Extended divide	divr.rl, divd.rq, quo.wl, rem.wl	.	.	.	0
74	EMODD	Extended modulus D_floating	mulr.rd, mulrx.rb, muld.rd, int.wl, fract.wd	.	.	.	0
54	EMODF	Extended modulus F_floating	mulr.rf, mulrx.rb, muld.rf, int.wl, fract.wf	.	.	.	0
54FD	EMODG	Extended modulus G_floating	mulr.rg, mulrx.rw, muld.rg, int.wl, fract.wg	.	.	.	0
74FD	EMODH	Extended modulus H_floating	mulr.rh, mulrx.rw, muld.rh, int.wl, fract.wh	.	.	.	0
7A	EMUL	Extended multiply	mulr.rl, muld.rl, add.rl, prod.wq	.	.	0	0
FD	ESCD	Escape D		.	.	.	.
FE	ESCE	Escape E		.	.	.	.
FF	ESCF	Escape F		.	.	.	.
EE	EXTV	Extract field	pos.rl, size.rb, base.vb, [field.rv], dst.wl	.	.	0	-

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
EF	EXTZV	Extract zero-extended field	pos.rl, size.rb, base.vb, [field.rv], dst.wl	.	.	0	-
EB	FFC	Find first clear bit	startpos.rl, size.rb, base.vb, [field.rv], findpos.wl	0	.	0	0
EA	FFS	Find first set bit	startpos.rl, size.rb, base.vb, [field.rv], findpos.wl	0	.	0	0
00	HALT	Halt (kernel mode only)	[–(KSP).w*]	.	.	.	.
96	INCB	Increment byte	sum.mb	.	.	.	.
D6	INCL	Increment long	sum.ml	.	.	.	.
B6	INCW	Increment word	sum.mw	.	.	.	.
0A	INDEX	Index calculation	subscript.rl, low.rl, high.rl, size.rl, entry.rl, addr.wl	.	.	0	0
5C	INSQHI	Insert at head of queue, interlocked	entry.ab, header.aq	0	.	0	.
5D	INSQTI	Insert at tail of queue, interlocked	entry.ab, header.aq	0	.	0	.
0E	INSQUE	Insert into queue	entry.ab, addr.wl	.	.	0	.
F0	INSV	Insert field	src.rl, pos.rl, size.rb, base.vb, [field.wv]	-	-	-	-
17	JMP	Jump	dst.ab	-	-	-	-
16	JSB	Jump to subroutine	dst.ab, [–(SP) + .wl]	-	-	-	-

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
06	LDPCTX	Load process context (kernel mode only)	[PCB.r*, –(KSP).w*]	-	-	-	-
3A	LOCC	Locate character	char.rb, len.rw, addr.ab, [R0-1.wl]	0	.	0	0
39	MATCHC	Match characters	len1.rw, addr1.ab, len2.rw, addr2.ab, [R0-3.wl]	0	.	0	0
92	MCOMB	Move complemented byte	src.rb, dst.wb	.	.	0	-
D2	MCOML	Move complemented long	src.rl, dst.wl	.	.	0	-
B2	MCOMW	Move complemented word	src.rw, dst.ww	.	.	0	-
DB	MFPR	Move from processor register (kernel mode only)	procreg.rl, dst.wl	.	.	0	-
8E	MNEGB	Move negated byte	src.rb, dst.wb	.	.	.	.
72	MNEGD	Move negated D_floating	src.rd, dst.wd	.	.	0	0
52	MNEGF	Move negated F_floating	src.rf, dst.wf	.	.	0	0
52FD	MNEGG	Move negated G_floating	src.rg, dst.wg	.	.	0	0
72FD	MNEGH	Move negated H_floating	src.rh, dst.wh	.	.	0	0
CE	MNEGL	Move negated long	src.rl, dst.wl	.	.	.	.
AE	MNEGW	Move negated word	src.rw, dst.ww	.	.	.	.
9E	MOVAB	Move address of byte	src.ab, dst.wl	.	.	0	-

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
7E	MOVAD	Move address of D_floating	src.aq, dst.wl	.	.	0	-
DE	MOVAF	Move address of F_floating	src.al, dst.wl	.	.	0	-
7E	MOVAG	Move address of G_floating	src.aq, dst.wl	.	.	0	-
7EFD	MOVAH	Move address of H_floating	src.ao, dst.wl	.	.	0	-
DE	MOVAL	Move address of long	src.al, dst.wl	.	.	0	-
7EFD	MOVAO	Move address of octaword	src.ao, dst.wl	.	.	0	-
7E	MOVAQ	Move address of quad	src.aq, dst.wl	.	.	0	-
3E	MOVAW	Move address of word	src.aw, dst.wl	.	.	0	-
90	MOVB	Move byte	src.rb, dst.wb	.	.	0	-
28	MOVC3	Move character 3-operand	len.rw, srcaddr.ab, dstaddr.ab, [R0-5.wl]	0	1	0	0
2C	MOVC5	Move character 5-operand	srclen.rw, srcaddr.ab, fill.rb, dstlen.rw, dstaddr.ab, [R0-5.wl]	.	.	0	.
70	MOVD	Move D_floating	src.rd, dst.wd	.	.	0	-
50	MOVF	Move F_floating	src.rf, dst.wf	.	.	0	-
50FD	MOVG	Move G_floating	src.rg, dst.wg	.	.	0	-
70FD	MOVH	Move H_floating	src.rh, dst.wh	.	.	0	-
D0	MOVL	Move long	src.rl, dst.wl	.	.	0	-
7DFD	MOVO	Move octaword	src.ro, dst.wo	.	.	0	-

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
34	MOVP	Move packed	len.rw, srcaddr.ab, dstaddr.ab, [R0-3.wl]	.	.	0	-
DC	MOVPSL	Move processor status longword	dst.wl	-	-	-	-
7D	MOVQ	Move quad	src.rq, dst.wq	.	.	0	-
2E	MOVTC	Move translated characters	srclen.rw, srcaddr.ab, fill.rb, tbladdr.ab, dstlen.rw, dstaddr.ab, [R0-5.wl]	.	.	0	.
2F	MOVTUC	Move translated until character	srclen.rw, srcaddr.ab, escape.rb, tbladdr.ab, dstlen.rw, dstaddr.ab, [R0-5.wl]	.	.	.	.
B0	MOVW	Move word	src.rw, dst.ww	.	.	0	-
9A	MOVZBL	Move zero-extended byte to long	src.rb, dst.wl	0	.	0	-
9B	MOVZBW	Move zero-extended byte to word	src.rb, dst.ww	0	.	0	-
3C	MOVZWL	Move zero-extended word to long	src.rw, dst.wl	0	.	0	-
DA	MTPR	Move to processor register (kernel mode only)	src.rl, procreg.wl	.	.	0	-
84	MULB2	Multiply byte 2-operand	mulr.rb, prod.mb	.	.	.	0
85	MULB3	Multiply byte 3-operand	mulr.rb, muld.rb, prod.wb	.	.	.	0
64	MULD2	Multiply D_floating 2-operand	mulr.rd, prod.md	.	.	.	0
65	MULD3	Multiply D_floating 3-operand	mulr.rd, muld.rd, prod.wd	.	.	.	0
44	MULF2	Multiply F_floating 2-operand	mulr.rf, prod.mf	.	.	.	0

OP	Mnemonic	Description	Arguments	Cond. Codes N Z V C
45	MULF3	Multiply F_floating 3-operand	mulr.rf, muld.rf, prod.wf	. . . 0
44FD	MULG2	Multiply G_floating 2-operand	mulr.rg, prod.mg	. . . 0
45FD	MULG3	Multiply G_floating 3-operand	mulr.rg, muld.rg, prod.wg	. . . 0
64FD	MULH2	Multiply G_floating 2-operand	mulr.rh, prod.mh	. . . 0
65FD	MULH3	Multiply H_floating 3-operand	mulr.rh, muld.rh, prod.wh	. . . 0
C4	MULL2	Multiply long 2-operand	mulr.rl, prod.ml	. . . 0
C5	MULL3	Multiply long 3-operand	mulr.rl, muld.rl, prod.wl	. . . 0
25	MULP	Multiply packed	mulrlen.rw, mulradr.ab, muldlen.rw, muldadr.ab, prodlen.rw, prodadr.ab, [R0-5.wl]	. . . 0
A4	MULW2	Multiply word 2-operand	mulr.rw, prod.mw	. . . 0
A5	MULW3	Multiply word 3-operand	mulr.rw, muld.rw, prod.ww	. . . 0
01	NOP	No operation		- - - -
75	POLYD	Evaluate polynomial D_floating	arg.rd, degree.rw, tbladdr.ab, [R0-5.wl]	. . . 0
55	POLYF	Evaluate polynomial F_floating	arg.rf, degree.rw, tbladdr.ab, [R0-3.wl]	. . . 0
55FD	POLYG	Evaluate polynomial G_floating	arg.rg, degree.rw, tbladdr.ab, [R0-5.wl]	. . . 0
75FD	POLYH	Evaluate polynomial H_floating	arg.rh, degree.rw, tbladdr.ab, [R0-5.wl, −16(SP):−1(SP).wl]	. . . 0

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
BA	POPR	Pop registers	mask.rw, [(SP)+.r*]	-	-	-	-
0C	PROBER	Prode read access	mode.rb, len.rw, base.ab	0	.	0	-
0D	PROBEW	Probe write access	mode.rb, len.rw, base.ab	0	.	0	-
9F	PUSHAB	Push address of byte	src.ab, [–(SP).wl]	.	.	0	-
7F	PUSHAD	Push address of D_floating	src.aq, [–(SP).wl]	.	.	0	-
DF	PUSHAF	Push address of F_floating	src.al, [–(SP).wl]	.	.	0	-
7F	PUSHAG	Push address of G_floating	src.aq, [–(SP).wl]	.	.	0	-
7FFD	PUSHAH	Push address of H_floating	src.ao, [–(SP).wl]	.	.	0	-
DF	PUSHAL	Push address of long	src.al, [–(SP).wl]	.	.	0	-
7FFD	PUSHAO	Push address of octaword	src.ao, [–(SP).wl]	.	.	0	-
7F	PUSHAQ	Push address of quad	src.aq, [–(SP).wl]	.	.	0	-
3F	PUSHAW	Push address of word	src.aw, [–(SP).wl]	.	.	0	-
DD	PUSHL	Push long	src.rl, [–(SP).wl]	.	.	0	-
BB	PUSHR	Push registers	mask.rw, [–(SP).w*]	-	-	-	-
02	REI	Return from exception or interrupt	[(SP)+.r*]	.	.	.	.
5E	REMQHI	Remove from head of queue, interlocked	header.aq, addr.wl	0	.	.	.

OP	Mnemonic	Description	Arguments	Cond. Codes N Z V C
5F	REMQTI	Remove from tail of queue, interlocked	header.aq, addr.wl	0 . . .
0F	REMQUE	Remove from queue	entry.ab, addr.wl	. . . .
04	RET	Return from procedure	[(SP)+.r*]	. . . .
9C	ROTL	Rotate long	count.rb, src.rl, dst.wl	. . 0 -
05	RSB	Return from subroutine	[(SP)+.rl]	- - - -
57	Reserved	Reserved		
5A	Reserved	Reserved		
5B	Reserved	Reserved		
77	Reserved	Reserved		
FE	Reserved	Reserved		
FF	Reserved	Reserved		
D9	SBWC	Subtract with carry	sub.rl, dif.ml	. . . .
2A	SCANC	Scan for character	len.rw, addr.ab, tbladdr.ab, mask.rb, [R0-3.wl]	0 . 0 0
3B	SKPC	Skip character	char.rb, len.rw, addr.ab, [R0-1.wl]	0 . 0 0
F4	SOBGEQ	Subtract one and branch on greater or equal	index.ml, displ.bb	. . . -

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
F5	SOBGTR	Subtract one and branch on greater	index.ml, displ.bb	.	.	.	-
2B	SPANC	Span characters	len.rw, addr.ab, tbladdr.ab, mask.rb, [R0-3.wl]	0	.	0	0
82	SUBB2	Subtract byte 2-operand	sub.rb, dif.mb	.	.	.	.
83	SUBB3	Subtract byte 3-operand	sub.rb, min.rb, dif.wb	.	.	.	.
62	SUBD2	Subtract D_floating 2-operand	sub.rd, dif.md	.	.	.	0
63	SUBD3	Subtract D_floating 3-operand	sub.rd, min.rd, dif.wd	.	.	.	0
42	SUBF2	Subtract F_floating 2-operand	sub.rf, dif.mf	.	.	.	0
43	SUBF3	Subtract F_floating 3-operand	sub.rf, min.rf, dif.wf	.	.	.	0
42FD	SUBG2	Subtract G_floating 2-operand	sub.rg, dif.mg	.	.	.	0
43FD	SUBG3	Subtract G_floating 3-operand	sub.rg, min.rg, dif.wg	.	.	.	0
62FD	SUBH2	Subtract H_floating 2-operand	sub.rh, dif.mh	.	.	.	0
63FD	SUBH3	Subtract H_floating 3-operand	sub.rh, min.rh, dif.wh	.	.	.	0
C2	SUBL2	Subtract long 2-operand	sub.rl, dif.ml	.	.	.	.
C3	SUBL3	Subtract long 3-operand	sub.rl, min.rl, dif.wl	.	.	.	.
22	SUBP4	Subtract packed 4-operand	sublen.rw, subaddr.ab, diflen.rw, difaddr.ab, [R0-3.wl]	.	.	.	0

OP	Mnemonic	Description	Arguments	Cond. Codes N	Z	V	C
23	SUBP6	Subtract packed 6-operand	sublen.rw, subaddr.ab, minlen.rw, minaddr.ab, diflen.rw, difaddr.ab, [R0-5.wl]	.	.	.	0
A2	SUBW2	Subtract word 2-operand	sub.rw, dif.mw	.	.	.	.
A3	SUBW3	Subtract word 3-operand	sub.rw, min.rw, dif.ww	.	.	.	.
07	SVPCTX	Save process context (kernel mode only)	[(SP)+.r*, –(KSP).w*]	-	-	-	-
95	TSTB	Test byte	src.rb	.	.	0	0
73	TSTD	Test D_floating	src.rd	.	.	0	0
53	TSTF	Test F_floating	src.rf	.	.	0	0
53FD	TSTG	Test G_floating	src.rg	.	.	0	0
73FD	TSTH	Test H_floating	src.rh	.	.	0	0
D5	TSTL	Test long	src.rl	.	.	0	0
B5	TSTW	Test word	src.rw	.	.	0	0
FC	XFC	Extended function call	user defined operands	0	0	0	0
8C	XORB2	Exclusive or byte 2-operand	mask.rb, dst.mb	.	.	0	-

OP	Mnemonic	Description	Arguments	Cond. Codes N Z V C
8D	XORB3	Exclusive or byte 3-operand	mask.rb, src.rb, dst.wb	. . 0 -
CC	XORL2	Exclusive or long 2-operand	mask.rl., dst.ml	. . 0 -
CD	XORL3	Exclusive or long 3-operand	mask.rl, src.rl, dst.wl	. . 0 -
AC	XORW2	Exclusive or word 2-operand	mask.rw, dst.mw	. . 0 -
AD	XORW3	Exclusive or word 3-operand	mask.rw, src.rw, dst.ww	. . 0 -

Numeric Order

00	HALT	10	BSBB	1E	BCC	2C	MOVC5	3A	LOCC
01	NOP	11	BRB	1E	BGEQU	2D	CMPC5	3B	SKPC
02	REI	12	BNEQ	1F	BCS	2E	MOVTC	3C	MOVZWL
03	BPT	12	BNEQU	1F	BLSSU	2F	MOVTUC	3D	ACBW
04	RET	13	BEQL	20	ADDP4	30	BSBW	3E	MOVAW
05	RSB	13	BEQLU	21	ADDP6	31	BRW	3F	PUSHAW
06	LDPCTX	14	BGTR	22	SUBP4	32	CVTWL	40	ADDF2
07	SVPCTX	15	BLEQ	23	SUBP6	32FD	CVTDH	40FD	ADDG2
08	CVTPS	16	JSB	24	CVTPT	33	CVTWB	41	ADDF3
09	CVTSP	17	JMP	25	MULP	33FD	CVTGF	41FD	ADDG3
0A	INDEX	18	BGEQ	26	CVTTP	34	MOVP	42	SUBF2
0B	CRC	19	BLSS	27	DIVP	35	CMPP3	42FD	SUBG2
0C	PROBER	1A	BGTRU	28	MOVC3	36	CVTPL	43	SUBF3
0D	PROBEW	1B	BLEQU	29	CMPC3	37	CMPP4	43FD	SUBG3
0E	INSQUE	1C	BVC	2A	SCANC	38	EDITPC	44	MULF2
0F	REMQUE	1D	BVS	2B	SPANC	39	MATCHC	44FD	MULG2

Numeric Order

45	MULF3	4E	CVTLF	57	Reserved	65	MULD3	6E	CVTLD
45FD	MULG3	4EFD	CVTLG	58	ADAWI	65FD	MULH3	6EFD	CVTLH
46	DIVF2	4F	ACBF	5A	Reserved	66	DIVD2	6F	ACBD
46FD	DIVG2	4FFD	ACBG	5B	Reserved	66FD	DIVH2	6FFD	ACBH
47	DIVF3	50	MOVF	5C	INSQHI	67	DIVD3	70	MOVD
47FD	DIVG3	50FD	MOVG	5D	INSQTI	67FD	DIVH3	70FD	MOVH
48	CVTFB	51	CMPF	5E	REMQHI	68	CVTDB	71	CMPD
48FD	CVTGB	51FD	CMPG	5F	REMQTI	68FD	CVTHB	71FD	CMPH
49	CVTFW	52	MNEGF	60	ADDD2	69	CVTDW	72	MNEGD
49FD	CVTGW	52FD	MNEGG	60FD	ADDH2	69FD	CVTHW	72FD	MNEGH
4A	CVTFL	53	TSTF	61	ADDD3	6A	CVTDL	73	TSTD
4AFD	CVTGL	53FD	TSTG	61FD	ADDH3	6AFD	CVTHL	73FD	TSTH
4B	CVTRFL	54	EMODF	62	SUBD2	6B	CVTRDL	74	EMODD
4BFD	CVTRGL	54FD	EMODG	62FD	SUBH2	6BFD	CVTRHL	74FD	EMODH
4C	CVTBF	55	POLYF	63	SUBD3	6C	CVTBD	75	POLYD
4CFD	CVTBG	55FD	POLYG	63FD	SUBH3	6CFD	CVTBH	75FD	POLYH
4D	CVTWF	56	CVTFD	64	MULD2	6D	CVTWD	76	CVTDF
4DFD	CVTWG	56FD	CVTGH	64FD	MULH2	6DFD	CVTWH	76FD	CVTHG

Numeric Order

77	Reserved	7F	PUSHAG	8E	MNEGB	9E	MOVAB	B0	MOVW
78	ASHL	7F	PUSHAQ	8F	CASEB	9F	PUSHAB	B1	CMPW
79	ASHQ	7FFD	PUSHAH	90	MOVB	A0	ADDW2	B2	MCOMW
7A	EMUL	7FFD	PUSHAO	91	CMPB	A1	ADDW3	B3	BITW
7B	EDIV	80	ADDB2	92	MCOMB	A2	SUBW2	B4	CLRW
7C	CLRD	81	ADDB3	93	BITB	A3	SUBW3	B5	TSTW
7C	CLRG	82	SUBB2	94	CLRB	A4	MULW2	B6	INCW
7C	CLRQ	83	SUBB3	95	TSTB	A5	MULW3	B7	DECW
7CFD	CLRH	84	MULB2	96	INCB	A6	DIVW2	B8	BISPSW
7CFD	CLRO	85	MULB3	97	DECB	A7	DIVW3	B9	BICPSW
7D	MOVQ	86	DIVB2	98	CVTBL	A8	BISW2	BA	POPR
7DFD	MOVO	87	DIVB3	98FD	CVTFH	A9	BISW3	BB	PUSHR
7E	MOVAD	88	BISB2	99	CVTBW	AA	BICW2	BC	CHMK
7E	MOVAG	89	BISB3	99FD	CVTFG	AB	BICW3	BD	CHME
7E	MOVAQ	8A	BICB2	9A	MOVZBL	AC	XORW2	BE	CHMS
7EFD	MOVAH	8B	BICB3	9B	MOVZBW	AD	XORW3	BF	CHMU
7EFD	MOVAO	8C	XORB2	9C	ROTL	AE	MNEGW	C0	ADDL2
7F	PUSHAD	8D	XORB3	9D	ACBB	AF	CASEW	C1	ADDL3

Numeric Order

C2	SUBL2	D4	CLRF	E3	BBCS	F5	SOBGTR
C3	SUBL3	D4	CLRL	E4	BBSC	F6	CVTLB
C4	MULL2	D5	TSTL	E5	BBCC	F6FD	CVTHF
C5	MULL3	D6	INCL	E6	BBSSI	F7	CVTLW
C6	DIVL2	D7	DECL	E7	BBCCI	F7FD	CVTHD
C7	DIVL3	D8	ADWC	E8	BLBS	F8	ASHP
C8	BISL2	D9	SBWC	E9	BLBC	F9	CVTLP
C9	BISL3	DA	MTPR	EA	FFS	FA	CALLG
CA	BICL2	DB	MFPR	EB	FFC	FB	CALLS
CB	BICL3	DC	MOVPSL	EC	CMPV	FC	XFC
CC	XORL2	DD	PUSHL	ED	CMPZV	FD	ESCD
CD	XORL3	DE	MOVAF	EE	EXTV	FDFF	BUGL
CE	MNEGL	DE	MOVAL	EF	EXTZV	FE	ESCE
CF	CASEL	DF	PUSHAF	F0	INSV	FE	Reserved
D0	MOVL	DF	PUSHAL	F1	ACBL	FEFF	BUGW
D1	CMPL	E0	BBS	F2	AOBLSS	FF	ESCF
D2	MCOML	E1	BBC	F3	AOBLEQ	FF	Reserved
D3	BITL	E2	BBSS	F4	SOBGEQ		

Hexadecimal – ASCII Conversion

HEX Code	ASCII Char	HEX Code	ASCII Char	HEX Code	ASCII Char	HEX Code	ASCII Char
00	NUL	20	SP	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m

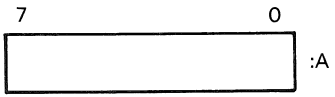
HEX Code	ASCII Char	HEX Code	ASCII Char	HEX Code	ASCII Char	HEX Code	ASCII Char
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL

Hexadecimal to Decimal Conversion Table

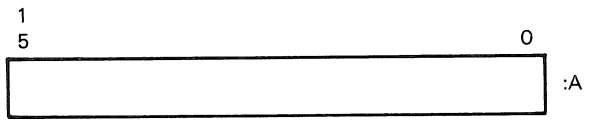
8		7		6		5		4		3		2		1
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	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	2,415,929,104	9	150,994,994	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	12,631,488	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	251,685,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

DATA TYPES

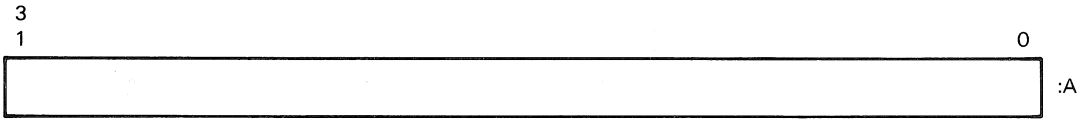
BYTE



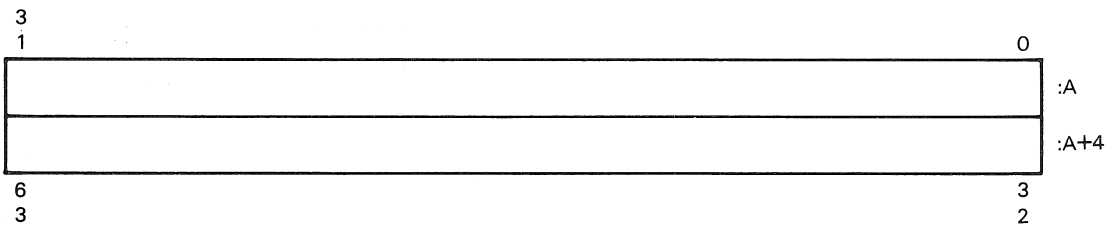
WORD



LONGWORD

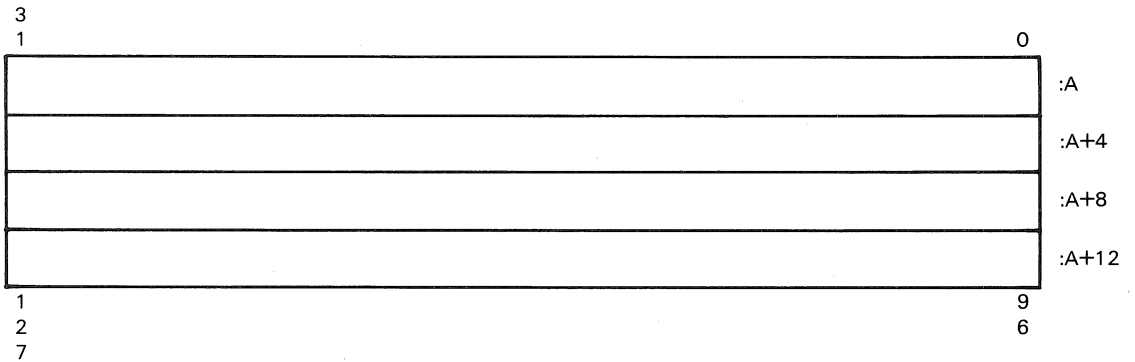


QUADWORD

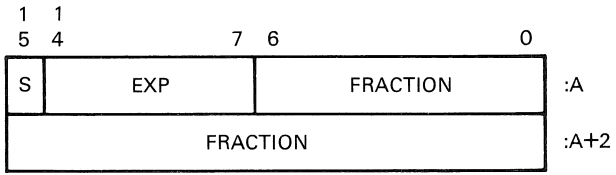


MK4675

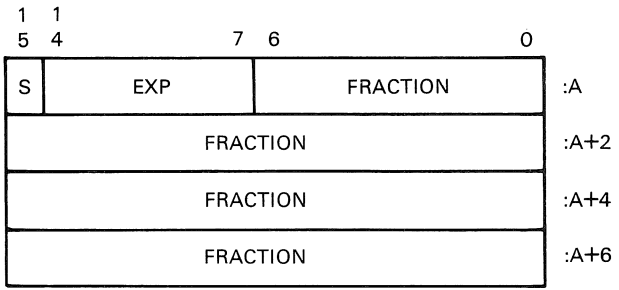
OCTAWORD



F __ FLOATING

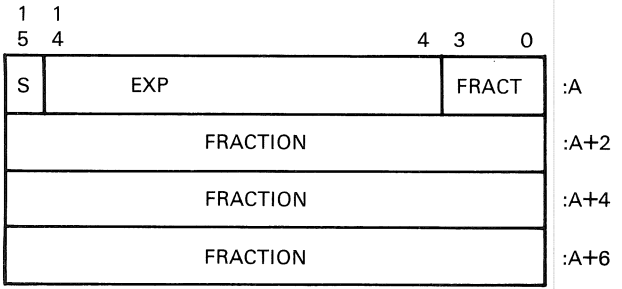


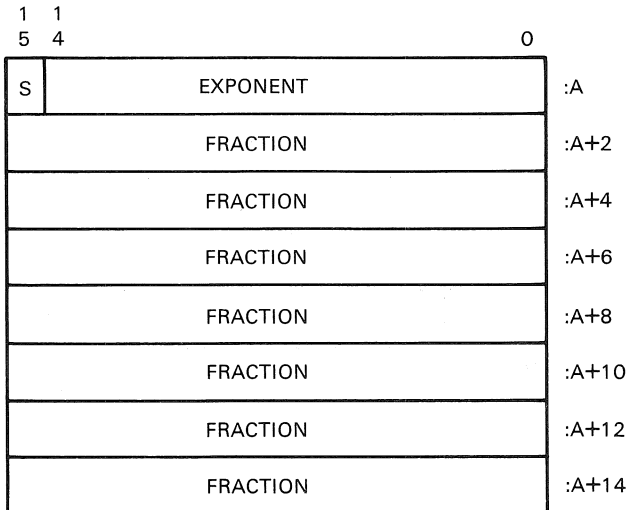
D __ FLOATING



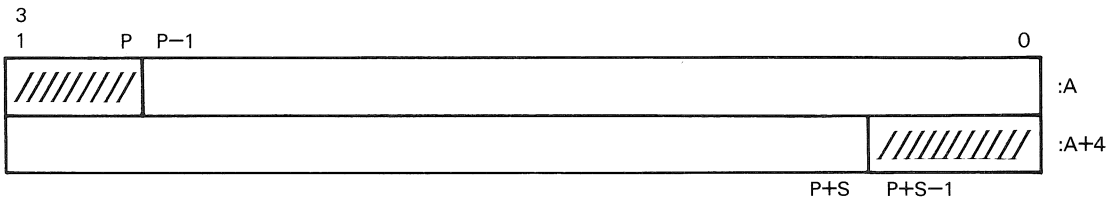
MK4676

G __ FLOATING




15 14
0
S
EXPONENT
:A
FRACTION
:A+2
FRACTION
:A+4
FRACTION
:A+6
FRACTION
:A+8
FRACTION
:A+10
FRACTION
:A+12
FRACTION
:A+14
MK4679

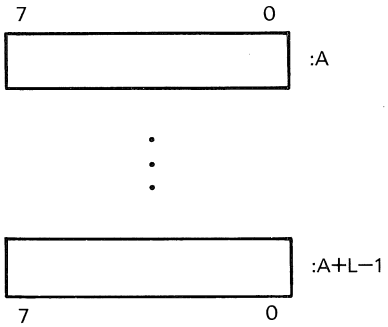
Variable Length Bit Field



P = STARTING LOCATION OF BIT FIELD
S = SIZE OF FIELD

MK4680

Character String



Trailing Numeric String

Representation of Least Significant Digit and Sign

	Zoned Numeric Format			Overpunch Format			
Digit	Decimal	Hex	ASCII Char	Decimal	Hex	ASCII Character Norm	Alt.
0	48	30	0	123	7B	{	0 [?
1	49	31	1	65	41	A	1
2	50	32	2	66	42	B	2
3	51	33	3	67	43	C	3
4	52	34	4	68	44	D	4
5	53	35	5	69	45	E	5
6	54	36	6	70	46	F	6
7	55	37	7	71	47	G	7
8	56	38	8	72	48	H	8
9	57	39	9	73	49	I	9

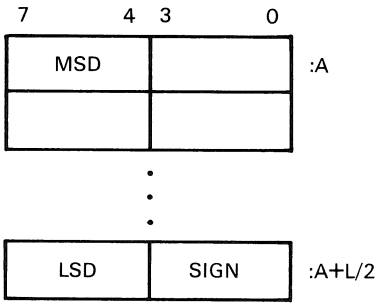
Zoned Numeric Format				Overpunch Format			
Digit	**Decimal**	**Hex**	**ASCII Char**	**Decimal**	**Hex**	**ASCII Character Norm**	**Alt.**
−0	112	70	p	125	7D	}	]!:
−1	113	71	q	74	4A	J	
−2	114	72	r	75	4B	K	
−3	115	73	s	76	4C	L	
−4	116	74	t	77	4D	M	
−5	117	75	u	78	4E	N	
−6	118	76	v	79	4F	O	
−7	119	77	w	80	50	P	
−8	120	78	x	81	51	Q	
−9	121	79	y	82	52	R	

Leading Separate Numeric String Sign Representation

Sign	Decimal	Hex	ASCII Character
+	43	2B	+
+	32	20	<blank>
-	45	2D	-

The preferred representation for "+" is ASCII "+".

Packed Decimal String



MK4678

Packed Decimal Sign Nibble Representation

Sign	Decimal	Hex
+	10, 12, 14 or 15	A, C, E, or F
−	11 or 13	B or D

The preferred sign representation is 12 for "+" and 13 for "−".

ASSEMBLER NOTATION FOR ADDRESSING MODES

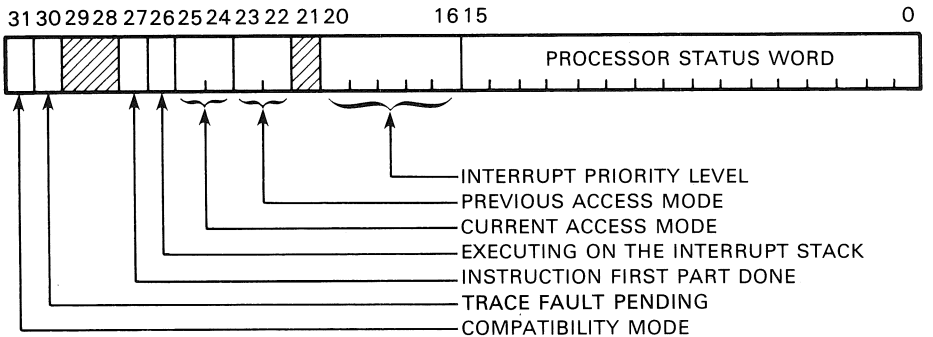
S^#5	forced short literal
#5	optimized short literal
R10	register
(R10)	register deferred
–(R10)	autodecrement
(R10)+	autoincrement
#START	immediate
I^#1	forced immediate
@(R10)+	autoincrement deferred
@#START	absolute
1(R10)	optimized byte displacement
0(R10)	optimized register deferred
@1(R10)	optimized byte displacement deferred
@(R10)	implied byte displacement deferred
START	optimized pc relative
@START	optimized pc relative deferred

1234(R10)	optimized word displacement
@1234(R10)	optimized word displacement deferred
12345678(R10)	longword displacement
@12345678(R10)	longword displacement deferred
B^12(R10)	forced byte displacement
B^START	forced byte pc relative
@B^12(R10)	forced byte displacement deferred
@B^START	forced byte pc relative deferred
W^12(R10)	forced word displacement
W^START	forced word pc relative
@W^12(R10)	forced word displacement deferred
@W^START	forced word pc relative deferred
L^12(R10)	forced longword displacement
L^START	forced longword pc relative
@L^12(R10)	forced longword displacement deferred
@L^START	forced longword pc relative deferred
(R10)[R11]	register deferred index
–(R10)[R11]	autodecrement indexed
(R10)+[R11]	autoincrement indexed
@(R10)+[R11]	autoincrement deferred indexed

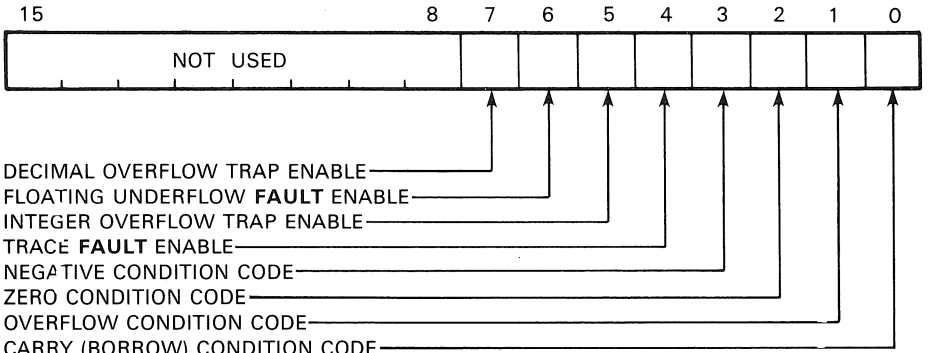
@#START[R11]	absolute indexed
1(R10)[R11]	optimized byte displacement indexed
0(R10)[R11]	optimized register deferred indexed
@1(R10)[R11]	optimized byte displacement deferred indexed
@(R10)[R11]	implied byte displacement deferred indexed
1234(R10)[R11]	optimized word displacement indexed
@1234(R10)[R11]	optimized word displacement deferred indexed
12345678(R10)[R11]	longword displacement indexed
@12345678(R10)[R11]	longword displacement deferred indexed
START[R11]	optimized pc relative indexed
@START[R11]	optimized pc relative deferred indexed
B^12(R10)[R11]	forced byte displacement indexed
B^START[R11]	forced byte pc relative indexed
@B^12(R10)[R11]	forced byte displacement deferred indexed
@B^START[R11]	forced byte pc relative deferred indexed
W^12(R10)[R11]	forced word displacement indexed
W^START[R11]	forced word pc relative indexed
@W^12(R10)[R11]	forced word displacement deferred indexed

@W^START[R11]	forced word pc relative deferred indexed
L^12(R10)[R11]	forced longword displacement indexed
L^START[R11]	forced longword pc relative indexed
@L^12(R10)[R11]	forced longword displacement deferred indexed
@L^START[R11]	forced longword pc relative deferred

PROCESSOR STATUS LONGWORD



PROCESSOR STATUS WORD



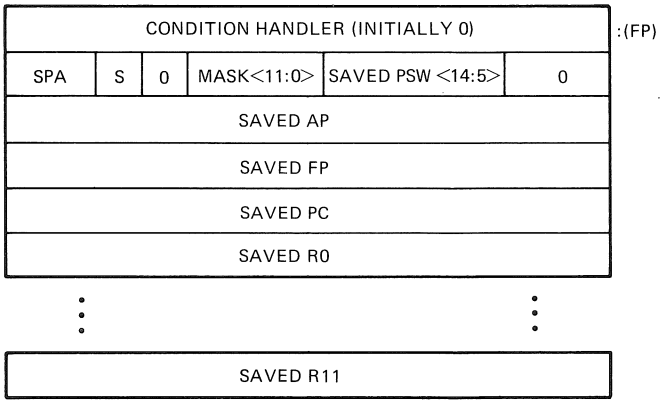
POWERS OF 2

2^n	n
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16
131072	17
262144	18
524288	19
1048576	20
2097152	21
4194304	22
8388608	23
16777216	24

POWERS OF 16

16^n	n
1	0
16	1
256	2
4096	3
65536	4
1048576	5
16777216	6
268435456	7
4294967296	8
68719476736	9
1099511627776	10
17592186044416	11
281474976710656	12
4503599627370496	13
72057594037927936	14
1152921504606846976	15

STACK FRAME FORMAT



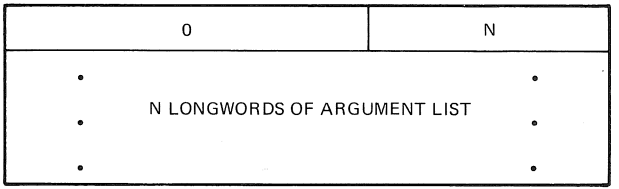
(0 TO 3 BYTES SPECIFIED BY SPA, STACK POINTER ALIGNMENT)

S = SET IF CALLS; CLEAR IF CALLG.

TK-10860

ARGUMENT LIST FORMAT:

The argument count N is an unsigned byte contained in the first byte of the list. The high order 24 bits of the first longword are reserved for future use and must be zero.



Digital Equipment Corporation • Bedford, MA 01730