Table 1-1 Addressing Mode Summary

Addressing Mode	Source Form	Abbreviation	Description
Indexed (preincrement)	INST oprx3,+xys	IDX	Effective address is the value in X, Y, or SP autoincremented by 1 to 8.
Indexed (postdecrement)	INST oprx3,xys–	IDX	Effective address is the value in X, Y, or SP. The value is postdecremented by 1 to 8.
Indexed (postincrement)	INST oprx3,xys+	IDX	Effective address is the value in X, Y, or SP. The value is postincremented by 1 to 8.
Indexed (accumulator offset)	INST abd,xysp	IDX	Effective address is the value in X, Y, SP, or PC plus the value in A, B, or D.
Indexed (9-bit offset)	INST oprx9,xysp	IDX1	Effective address is the value in X, Y, SP, or PC plus a 9-bit signed constant offset.
Indexed (16-bit offset)	INST oprx16,xysp	IDX2	Effective address is the value in X, Y, SP, or PC plus a 16-bit constant offset.
Indexed-indirect (16-bit offset)	INST [oprx16,xysp]	[IDX2]	The value in X, Y, SP, or PC plus a 16-bit constant offset points to the effective address.
Indexed-indirect (D accumulator offset)	INST [D,xysp]	[D,IDX]	The value in X, Y, SP, or PC plus the value in D points to the effective address.

1.8 Instruction Set Overview

All memory and I/O are mapped in a common 64K byte address space, allowing the same set of instructions to access memory, I/O, and control registers. Load, store, transfer, exchange, and move instructions facilitate movement of data to and from memory and peripherals.

There are instructions for signed and unsigned addition, division and multiplication with 8-bit, 16-bit, and some larger operands.

Special arithmetic and logic instructions aid stacking operations, indexing, BCD calculation, and condition code register manipulation. There are also dedicated instructions for multiply and accumulate operations, table interpolation, and specialized mathematical calculations for fuzzy logic operations.

A summary of the CPU instruction set is given in **Table 1-2** below. A detailed overview of the entire instruction set is covered in **Section 4** of this guide along with an instruction-by-instruction detailed description in **Appendix A**.

Table 1-2 Instruction Set Summary

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
ABA	Add B to A; (A)+(B)⇒A	INH	18 06	OO	– – Δ – Δ Δ Δ Δ
ABXSame as LEAX B,X	Add B to X; (X)+(B)⇒X	IDX	1A E5	Pf	– – – – – – – –
ABYSame as LEAY B,Y	Add B to Y; (Y)+(B)⇒Y	IDX	19 ED	Pf	– – – – – – – –
ADCA #*opr8i* ADCA *opr8a* ADCA *opr16a* ADCA *oprx0_xysppc* ADCA *oprx9,xysppc* ADCA *oprx16,xysppc* ADCA [D,*xysppc*] ADCA [*oprx16,xysppc*]	Add with carry to A; (A)+(M)+C⇒A or (A)+imm+C⇒A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	89 ii 99 dd B9 hh ll A9 xb A9 xb ff A9 xb ee ff A9 xb A9 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – Δ – Δ Δ Δ Δ

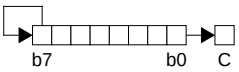
Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
ADCB #*opr8i* ADCB *opr8a* ADCB *opr16a* ADCB *oprx0_xysppc* ADCB *oprx9,xysppc* ADCB *oprx16,xysppc* ADCB [D,*xysppc*] ADCB [*oprx16,xysppc*]	Add with carry to B; (B)+(M)+C⇒B or (B)+imm+C⇒B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C9 ii D9 dd F9 hh ll E9 xb E9 xb ff E9 xb ee ff E9 xb E9 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – Δ – Δ Δ Δ Δ

ADDA #*opr8i*
ADDA *opr8a*
ADDA *opr16a*
ADDA *oprx0_xysppc*
ADDA *oprx9,xysppc*
ADDA *oprx16,xysppc*
ADDA [D,*xysppc*]
ADDA [*oprx16,xysppc*]
	Add to A; (A)+(M)⇒A
	or (A)+imm⇒A
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			8B ii
			9B dd
			BB hh ll
			AB xb
			AB xb ff
			AB xb ee ff
			AB xb
			AB xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					– – Δ – Δ Δ Δ Δ

ADDB #*opr8i*
ADDB *opr8a*
ADDB *opr16a*
ADDB *oprx0_xysppc*
ADDB *oprx9,xysppc*
ADDB *oprx16,xysppc*
ADDB [D,*xysppc*]
ADDB [*oprx16,xysppc*]
	Add to B; (B)+(M)⇒B
	or (B)+imm⇒B
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CB ii
			DB dd
			FB hh ll
			EB xb
			EB xb ff
			EB xb ee ff
			EB xb
			EB xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					– – Δ – Δ Δ Δ Δ

ADDD #*opr16i*
ADDD *opr8a*
ADDD *opr16a*
ADDD *oprx0_xysppc*
ADDD *oprx9,xysppc*
ADDD *oprx16,xysppc*
ADDD [D,*xysppc*]
ADDD [*oprx16,xysppc*]
	Add to D; (A:B)+(M:M+1)⇒A:B
	or (A:B)+imm⇒A:B
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			C3 jj kk
			D3 dd
			F3 hh ll
			E3 xb
			E3 xb ff
			E3 xb ee ff
			E3 xb
			E3 xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					– – – – Δ Δ Δ Δ

ANDA #*opr8i* ANDA *opr8a* ANDA *opr16a* ANDA *oprx0_xysppc* ANDA *oprx9,xysppc* ANDA *oprx16,xysppc* ANDA [D,*xysppc*] ANDA [*oprx16,xysppc*]	AND with A; (A)•(M)⇒A or (A)•imm⇒A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	84 ii 94 dd B4 hh ll A4 xb A4 xb ff A4 xb ee ff A4 xb A4 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – – Δ Δ 0 –
ANDB #*opr8i* ANDB *opr8a* ANDB *opr16a* ANDB *oprx0_xysppc* ANDB *oprx9,xysppc* ANDB *oprx16,xysppc* ANDB [D,*xysppc*] ANDB [*oprx16,xysppc*]	AND with B; (B)•(M)⇒B or (B)•imm⇒B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C4 ii D4 dd F4 hh ll E4 xb E4 xb ff E4 xb ee ff E4 xb E4 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – – Δ Δ 0 –
ANDCC #*opr8i*	AND with CCR; (CCR)•imm⇒CCR	IMM	10 ii	P	⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓
ASL *opr16a*Same as LSL ASL *oprx0_xysp* ASL *oprx9,xysppc* ASL *oprx16,xysppc* ASL [D,*xysppc*] ASL [*oprx16,xysppc*] ASLASame as LSLA ASLBSame as LSLB	Arithmetic shift left M C b7 … b0 ← 0 Arithmetic shift left A Arithmetic shift left B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff 48 58	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – – Δ Δ Δ Δ
ASLDSame as LSLD	Arithmetic shift left D C b7 A b0 b7 B b0 ← 0	INH	59	O	– – – – Δ Δ Δ Δ

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
ASR *opr16a* ASR *oprx0_xysppc* ASR *oprx9,xysppc* ASR *oprx16,xysppc* ASR [D,*xysppc*] ASR [*oprx16,xysppc*] ASRA ASRB	Arithmetic shift right M  Arithmetic shift right A Arithmetic shift right B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	77 hh ll 67 xb 67 xb ff 67 xb ee ff 67 xb 67 xb ee ff 47 57	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – – Δ Δ Δ Δ
BCC *rel8*Same as BHS	Branch if C clear; if C=0, then (PC)+2+rel⇒PC	REL	24 rr	PPP (branch) P (no branch)	– – – – – – – –
BCLR *opr8a, msk8* BCLR *opr16a, msk8* BCLR *oprx0_xysppc, msk8* BCLR *oprx9,xysppc, msk8* BCLR *oprx16,xysppc, msk8*	Clear bit(s) in M; (M)•mask byte⇒M	DIR EXT IDX IDX1 IDX2	4D dd mm 1D hh ll mm 0D xb mm 0D xb ff mm 0D xb ee ff mm	rPwO rPwP rPwO rPwP frPwPO	– – – – Δ Δ 0 –
BCS *rel8*Same as BLO	Branch if C set; if C=1, then (PC)+2+rel⇒PC	REL	25 rr	PPP (branch) P (no branch)	– – – – – – – –
BEQ *rel8*	Branch if equal; if Z=1, then (PC)+2+rel⇒PC	REL	27 rr	PPP (branch) P (no branch)	– – – – – – – –
BGE *rel8*	Branch if ≥ 0, signed; if N⊕V=0, then (PC)+2+rel⇒PC	REL	2C rr	PPP (branch) P (no branch)	– – – – – – – –
BGND	Enter background debug mode	INH	00	VfPPP	– – – – – – – –
BGT *rel8*	Branch if > 0, signed; if Z | (N⊕V)=0, then (PC)+2+rel⇒PC	REL	2E rr	PPP (branch) P (no branch)	– – – – – – – –
BHI *rel8*	Branch if higher, unsigned; if C | Z=0, then (PC)+2+rel⇒PC	REL	22 rr	PPP (branch) P (no branch)	– – – – – – – –
BHS *rel8*Same as BCC	Branch if higher or same, unsigned; if C=0, then (PC)+2+rel⇒PC	REL	24 rr	PPP (branch) P (no branch)	– – – – – – – –
BITA #*opr8i* BITA *opr8a* BITA *opr16a* BITA *oprx0_xysppc* BITA *oprx9,xysppc* BITA *oprx16,xysppc* BITA [D,*xysppc*] BITA [*oprx16,xysppc*]	Bit test A; (A)•(M) or (A)•imm	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	85 ii 95 dd B5 hh ll A5 xb A5 xb ff A5 xb ee ff A5 xb A5 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – – Δ Δ 0 –
BITB #*opr8i* BITB *opr8a* BITB *opr16a* BITB *oprx0_xysppc* BITB *oprx9,xysppc* BITB *oprx16,xysppc* BITB [D,*xysppc*] BITB [*oprx16,xysppc*]	Bit test B; (B)•(M) or (B)•imm	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C5 ii D5 dd F5 hh ll E5 xb E5 xb ff E5 xb ee ff E5 xb E5 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – – Δ Δ 0 –
BLE *rel8*	Branch if ≤ 0, signed; if Z | (N⊕V)=1, then (PC)+2+rel⇒PC	REL	2F rr	PPP (branch) P (no branch)	– – – – – – – –
BLO *rel8*Same as BCS	Branch if lower, unsigned; if C=1, then (PC)+2+rel⇒PC	REL	25 rr	PPP (branch) P (no branch)	– – – – – – – –
BLS *rel8*	Branch if lower or same, unsigned; if C | Z=1, then (PC)+2+rel⇒PC	REL	23 rr	PPP (branch) P (no branch)	– – – – – – – –
BLT *rel8*	Branch if < 0, signed; if N⊕V=1, then (PC)+2+rel⇒PC	REL	2D rr	PPP (branch) P (no branch)	– – – – – – – –
BMI *rel8*	Branch if minus; if N=1, then (PC)+2+rel⇒PC	REL	2B rr	PPP (branch) P (no branch)	– – – – – – – –
BNE *rel8*	Branch if not equal to 0; if Z=0, then (PC)+2+rel⇒PC	REL	26 rr	PPP (branch) P (no branch)	– – – – – – – –
BPL *rel8*	Branch if plus; if N=0, then (PC)+2+rel⇒PC	REL	2A rr	PPP (branch) P (no branch)	– – – – – – – –
BRA *rel8*	Branch always	REL	20 rr	PPP	– – – – – – – –

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
BRCLR *opr8a*, *msk8*, *rel8* BRCLR *opr16a*, *msk8*, *rel8* BRCLR *oprx0_xysppc*, *msk8*, *rel8* BRCLR *oprx9,xysppc*, *msk8*, *rel8* BRCLR *oprx16,xysppc*, *msk8*, *rel8*	Branch if bit(s) clear; if (M)•(mask byte)=0, then (PC)+2+rel⇒PC	DIR EXT IDX IDX1 IDX2	4F dd mm rr 1F hh ll mm rr 0F xb mm rr 0F xb ff mm rr 0F xb ee ff mm rr	rPPP rfPPP rPPP rfPPP PrfPPP	– – – – – – – –
BRN *rel8*	Branch never	REL	21 rr	P	– – – – – – – –
BRSET *opr8*, *msk8*, *rel8* BRSET *opr16a*, *msk8*, *rel8* BRSET *oprx0_xysppc*, *msk8*, *rel8* BRSET *oprx9,xysppc*, *msk8*, *rel8* BRSET *oprx16,xysppc*, *msk8*, *rel8*	Branch if bit(s) set; if ($\overline{M}$)•(mask byte)=0, then (PC)+2+rel⇒PC	DIR EXT IDX IDX1 IDX2	4E dd mm rr 1E hh ll mm rr 0E xb mm rr 0E xb ff mm rr 0E xb ee ff mm rr	rPPP rfPPP rPPP rfPPP PrfPPP	– – – – – – – –
BSET *opr8*, *msk8* BSET *opr16a*, *msk8* BSET *oprx0_xysppc*, *msk8* BSET *oprx9,xysppc*, *msk8* BSET *oprx16,xysppc*, *msk8*	Set bit(s) in M (M) | mask byte⇒M	DIR EXT IDX IDX1 IDX2	4C dd mm 1C hh ll mm 0C xb mm 0C xb ff mm 0C xb ee ff mm	rPwO rPwP rPwO rPwP frPwPO	– – – – Δ Δ 0 –
BSR *rel8*	Branch to subroutine; (SP)–2⇒SP RTN_H:RTN_L⇒M_{SP}:M_{SP+1} (PC)+2+rel⇒PC	REL	07 rr	SPPP	– – – – – – – –
BVC *rel8*	Branch if V clear; if V=0, then (PC)+2+rel⇒PC	REL	28 rr	PPP (branch) P (no branch)	– – – – – – – –
BVS *rel8*	Branch if V set; if V=1, then (PC)+2+rel⇒PC	REL	29 rr	PPP (branch) P (no branch)	– – – – – – – –
CALL *opr16a*, *page* CALL *oprx0_xysppc*, *page* CALL *oprx9,xysppc*, *page* CALL *oprx16,xysppc*, *page* CALL [D,*xysppc*] CALL [*oprx16*, *xysppc*]	Call subroutine in expanded memory (SP)–2⇒SP RTN_H:RTN_L⇒M_{SP}:M_{SP+1} (SP)–1⇒SP; (PPG)⇒M_{SP} pg⇒PPAGE register subroutine address⇒PC	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	4A hh ll pg 4B xb pg 4B xb ff pg 4B xb ee ff pg 4B xb 4B xb ee ff	gnSsPPP gnSsPPP gnSsPPP fgnSsPPP fIignSsPPP fIignSsPPP	– – – – – – – –
CBA	Compare A to B; (A)–(B)	INH	18 17	OO	– – – – Δ Δ Δ Δ
CLCSame as ANDCC #$FE	Clear C bit	IMM	10 FE	P	– – – – – – – 0
CLISame as ANDCC #$EF	Clear I bit	IMM	10 EF	P	– – – 0 – – – –
CLR *opr16a* CLR *oprx0_xysppc* CLR *oprx9,xysppc* CLR *oprx16,xysppc* CLR [D,*xysppc*] CLR [*oprx16,xysppc*] CLRA CLRB	Clear M; $00⇒M Clear A; $00⇒A Clear B; $00⇒B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	79 hh ll 69 xb 69 xb ff 69 xb ee ff 69 xb 69 xb ee ff 87 C7	PwO Pw PwO PwP PIfw PIPw O O	– – – – 0 1 0 0
CLVSame as ANDCC #$FD	Clear V	IMM	10 FD	P	– – – – – – 0 –

CMPA #*opr8i*
CMPA *opr8a*
CMPA *opr16a*
CMPA *oprx0_xysppc*
CMPA *oprx9,xysppc*
CMPA *oprx16,xysppc*
CMPA [D,*xysppc*]
CMPA [*oprx16,xysppc*]
	Compare A
	(A)–(M) or (A)–imm
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			81 ii
			91 dd
			B1 hh ll
			A1 xb
			A1 xb ff
			A1 xb ee ff
			A1 xb
			A1 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					– – – – Δ Δ Δ Δ

CMPB #*opr8i*
CMPB *opr8a*
CMPB *opr16a*
CMPB *oprx0_xysppc*
CMPB *oprx9,xysppc*
CMPB *oprx16,xysppc*
CMPB [D,*xysppc*]
CMPB [*oprx16,xysppc*]
	Compare B
	(B)–(M) or (B)–imm
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			C1 ii
			D1 dd
			F1 hh ll
			E1 xb
			E1 xb ff
			E1 xb ee ff
			E1 xb
			E1 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					– – – – Δ Δ Δ Δ

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
COM *opr16a* COM *oprx0_xysppc* COM *oprx9,xysppc* COM *oprx16,xysppc* COM [D,*xysppc*] COM [*oprx16,xysppc*] COMA COMB	Complement M; ($\overline{M}$)=$FF–(M)⇒M Complement A; ($\overline{A}$)=$FF–(A)⇒A Complement B; ($\overline{B}$)=$FF–(B)⇒B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	71 hh ll 61 xb 61 xb ff 61 xb ee ff 61 xb 61 xb ee ff 41 51	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – – Δ Δ 0 1
CPD #*opr16i* CPD *opr8a* CPD *opr16a* CPD *oprx0_xysppc* CPD *oprx9,xysppc* CPD *oprx16,xysppc* CPD [D,*xysppc*] CPD [*oprx16,xysppc*]	Compare D (A:B)–(M:M+1) or (A:B)–imm	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8C jj kk 9C dd BC hh ll AC xb AC xb ff AC xb ee ff AC xb AC xb ee ff	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – – Δ Δ Δ Δ
CPS #*opr16i* CPS *opr8a* CPS *opr16a* CPS *oprx0_xysppc* CPS *oprx9,xysppc* CPS *oprx16,xysppc* CPS [D,*xysppc*] CPS [*oprx16,xysppc*]	Compare SP (SP)–(M:M+1) or (SP)–imm	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8F jj kk 9F dd BF hh ll AF xb AF xb ff AF xb ee ff AF xb AF xb ee ff	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – – Δ Δ Δ Δ
CPX #*opr16i* CPX *opr8a* CPX *opr16a* CPX *oprx0_xysppc* CPX *oprx9,xysppc* CPX *oprx16,xysppc* CPX [D,*xysppc*] CPX [*oprx16,xysppc*]	Compare X (X)–(M:M+1) or (X)–imm	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8E jj kk 9E dd BE hh ll AE xb AE xb ff AE xb ee ff AE xb AE xb ee ff	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – – Δ Δ Δ Δ
CPY #*opr16i* CPY *opr8a* CPY *opr16a* CPY *oprx0_xysppc* CPY *oprx9,xysppc* CPY *oprx16,xysppc* CPY [D,*xysppc*] CPY [*oprx16,xysppc*]	Compare Y (Y)–(M:M+1) or (Y)–imm	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8D jj kk 9D dd BD hh ll AD xb AD xb ff AD xb ee ff AD xb AD xb ee ff	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – – Δ Δ Δ Δ
DAA	Decimal adjust A for BCD	INH	18 07	OfO	– – – – Δ Δ ? Δ
DBEQ *abdxysp*, *rel9*	Decrement and branch if equal to 0 (counter)–1⇒counter if (counter)=0, then branch	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	– – – – – – – –
DBNE *abdxysp*, *rel9*	Decrement and branch if not equal to 0; (counter)–1⇒counter; if (counter)≠0, then branch	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	– – – – – – – –
DEC *opr16a* DEC *oprx0_xysppc* DEC *oprx9,xysppc* DEC *oprx16,xysppc* DEC [D,*xysppc*] DEC [*oprx16,xysppc*] DECA DECB	Decrement M; (M)–1⇒M Decrement A; (A)–1⇒A Decrement B; (B)–1⇒B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	73 hh ll 63 xb 63 xb ff 63 xb ee ff 63 xb 63 xb ee ff 43 53	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – – Δ Δ Δ –
DESSame as LEAS –1,SP	Decrement SP; (SP)–1⇒SP	IDX	1B 9F	Pf	– – – – – – – –
DEX	Decrement X; (X)–1⇒X	INH	09	O	– – – – – Δ – –
DEY	Decrement Y; (Y)–1⇒Y	INH	03	O	– – – – – Δ – –
EDIV	Extended divide, unsigned; 32 by 16 to 16-bit; (Y:D)÷(X)⇒Y; remainder⇒D	INH	11	fffffffffO	– – – – Δ Δ Δ Δ

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
EDIVS	Extended divide,signed; 32 by 16 to 16-bit; (Y:D)÷(X)⇒Y remainder⇒D	INH	18 14	`OffffffffffO`	– – – – Δ Δ Δ Δ
EMACS *opr16a*	Extended multiply and accumulate, signed; $(M_X:M_{X+1})\times(M_Y:M_{Y+1})+$ (M~M+3)⇒M~M+3; 16 by 16 to 32-bit	Special	18 12 hh ll	`ORROfffRRfWWP`	– – – – Δ Δ Δ Δ

EMAXD *oprx0_xysppc*
EMAXD *oprx9,xysppc*
EMAXD *oprx16,xysppc*
EMAXD [D,*xysppc*]
EMAXD [*oprx16,xysppc*]
	Extended maximum in D; put larger of 2
	unsigned 16-bit values in D
	MAX[(D), (M:M+1)]⇒D
	N, Z, V, C bits reflect result of internal compare [(D)–(M:M+1)]
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			18 1A xb
			18 1A xb ff
			18 1A xb ee ff
			18 1A xb
			18 1A xb ee ff
				`ORPf`
				`ORPO`
				`OfRPP`
				`OfIfRPf`
				`OfIPRPf`
					– – – – Δ Δ Δ Δ

EMAXM *oprx0_xysppc*
EMAXM *oprx9,xysppc*
EMAXM *oprx16,xysppc*
EMAXM [D,*xysppc*]
EMAXM [*oprx16,xysppc*]
	Extended maximum in M; put larger of 2
	unsigned 16-bit values in M
	MAX[(D), (M:M+1)]⇒M:M+1
	N, Z, V, C bits reflect result of internal compare [(D)–(M:M+1)]
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			18 1E xb
			18 1E xb ff
			18 1E xb ee ff
			18 1E xb
			18 1E xb ee ff
				`ORPW`
				`ORPWO`
				`OfRPWP`
				`OfIfRPW`
				`OfIPRPW`
					– – – – Δ Δ Δ Δ

EMIND *oprx0_xysppc*
EMIND *oprx9,xysppc*
EMIND *oprx16,xysppc*
EMIND [D,*xysppc*]
EMIND [*oprx16,xysppc*]
	Extended minimum in D; put smaller of
	2 unsigned 16-bit values in D
	MIN[(D), (M:M+1)]⇒D
	N, Z, V, C bits reflect result of internal compare [(D)–(M:M+1)]
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			18 1B xb
			18 1B xb ff
			18 1B xb ee ff
			18 1B xb
			18 1B xb ee ff
				`ORPf`
				`ORPO`
				`OfRPP`
				`OfIfRPf`
				`OfIPRPf`
					– – – – Δ Δ Δ Δ

EMINM *oprx0_xysppc*
EMINM *oprx9,xysppc*
EMINM *oprx16,xysppc*
EMINM [D,*xysppc*]
EMINM [*oprx16,xysppc*]
	Extended minimum in M; put smaller of
	2 unsigned 16-bit values in M
	MIN[(D), (M:M+1)]⇒M:M+1
	N, Z, V, C bits reflect result of internal compare [(D)–(M:M+1)]
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			18 1F xb
			18 1F xb ff
			18 1F xb ee ff
			18 1F xb
			18 1F xb ee ff
				`ORPW`
				`ORPWO`
				`OfRPWP`
				`OfIfRPW`
				`OfIPRPW`
					– – – – Δ Δ Δ Δ

EMUL	Extended multiply, unsigned (D)×(Y)⇒Y:D; 16 by 16 to 32-bit	INH	13	`ffO`	– – – – Δ Δ – Δ
EMULS	Extended multiply, signed (D)×(Y)⇒Y:D; 16 by 16 to 32-bit	INH	18 13	`OfO` `OffO` (if followed by page 2 instruction)	– – – – Δ Δ – Δ

EORA #*opr8i*
EORA *opr8a*
EORA *opr16a*
EORA *oprx0_xysppc*
EORA *oprx9,xysppc*
EORA *oprx16,xysppc*
EORA [D,*xysppc*]
EORA [*oprx16,xysppc*]
	Exclusive OR A
	(A)⊕(M)⇒A
	or (A)⊕imm⇒A
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			88 ii
			98 dd
			B8 hh ll
			A8 xb
			A8 xb ff
			A8 xb ee ff
			A8 xb
			A8 xb ee ff
				`P`
				`rPf`
				`rPO`
				`rPf`
				`rPO`
				`frPP`
				`fIfrPf`
				`fIPrPf`
					– – – – Δ Δ 0 –

EORB #*opr8i*
EORB *opr8a*
EORB *opr16a*
EORB *oprx0_xysppc*
EORB *oprx9,xysppc*
EORB *oprx16,xysppc*
EORB [D,*xysppc*]
EORB [*oprx16,xysppc*]
	Exclusive OR B
	(B)⊕(M)⇒B
	or (B)⊕imm⇒B
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			C8 ii
			D8 dd
			F8 hh ll
			E8 xb
			E8 xb ff
			E8 xb ee ff
			E8 xb
			E8 xb ee ff
				`P`
				`rPf`
				`rPO`
				`rPf`
				`rPO`
				`frPP`
				`fIfrPf`
				`fIPrPf`
					– – – – Δ Δ 0 –

ETBL *oprx0_xysppc*	Extended table lookup and interpolate, 16-bit; (M:M+1)+ [(B)×((M+2:M+3)–(M:M+1))]⇒D	IDX	18 3F xb	`ORRfffffffP`	– – – – Δ Δ – Δ
Before executing ETBL, initialize B with fractional part of lookup value; initialize index register to point to first table entry (M:M+1). No extensions or indirect addressing allowed.					
EXG *abcdxysp,abcdxysp*	Exchange register contents (r1)⇔(r2) r1 and r2 same size $00:(r1)⇒r2r1=8-bit; r2=16-bit $(r1_L)$⇔(r2)r1=16-bit; r2=8-bit	INH	B7 eb	`P`	– – – – – – – –
FDIV	Fractional divide; (D)÷(X)⇒X remainder⇒D; 16 by 16-bit	INH	18 11	`OffffffffffO`	– – – – – Δ Δ Δ

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
IBEQ *abdxysp*, *rel9*	Increment and branch if equal to 0 (counter)+1⇒counter If (counter)=0, then branch	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	– – – – – – – –
IBNE *abdxysp*, *rel9*	Increment and branch if not equal to 0 (counter)+1⇒counter If (counter)≠0, then branch	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	– – – – – – – –
IDIV	Integer divide, unsigned; (D)÷(X)⇒X Remainder⇒D; 16 by 16-bit	INH	18 10	OfffffffffffO	– – – – – Δ 0 Δ
IDIVS	Integer divide, signed; (D)÷(X)⇒X Remainder⇒D; 16 by 16-bit	INH	18 15	OfffffffffffO	– – – – Δ Δ Δ Δ
INC *opr16a* INC *oprx0_xysppc* INC *oprx9,xysppc* INC *oprx16,xysppc* INC [D,*xysppc*] INC [*oprx16,xysppc*] INCA INCB	Increment M; (M)+1⇒M Increment A; (A)+1⇒A Increment B; (B)+1⇒B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	72 hh ll 62 xb 62 xb ff 62 xb ee ff 62 xb 62 xb ee ff 42 52	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – – Δ Δ Δ –
INSSame as LEAS 1,SP	Increment SP; (SP)+1⇒SP	IDX	1B 81	Pf	– – – – – – – –
INX	Increment X; (X)+1⇒X	INH	08	O	– – – – – Δ – –
INY	Increment Y; (Y)+1⇒Y	INH	02	O	– – – – – Δ – –
JMP *opr16a* JMP *oprx0_xysppc* JMP *oprx9,xysppc* JMP *oprx16,xysppc* JMP [D,*xysppc*] JMP [*oprx16,xysppc*]	Jump Subroutine address⇒PC	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	06 hh ll 05 xb 05 xb ff 05 xb ee ff 05 xb 05 xb ee ff	PPP PPP PPP fPPP fIfPPP fIfPPP	– – – – – – – –
JSR *opr8a* JSR *opr16a* JSR *oprx0_xysppc* JSR *oprx9,xysppc* JSR *oprx16,xysppc* JSR [D,*xysppc*] JSR [*oprx16,xysppc*]	Jump to subroutine (SP)–2⇒SP RTN_H:RTN_L⇒M_{SP}:M_{SP+1} Subroutine address⇒PC	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	17 dd 16 hh ll 15 xb 15 xb ff 15 xb ee ff 15 xb 15 xb ee ff	SPPP SPPP PPPS PPPS fPPPS fIfPPPS fIfPPPS	– – – – – – – –
LBCC *rel16*Same as LBHS	Long branch if C clear; if C=0, then (PC)+4+rel⇒PC	REL	18 24 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBCS *rel16*Same as LBLO	Long branch if C set; if C=1, then (PC)+4+rel⇒PC	REL	18 25 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBEQ *rel16*	Long branch if equal; if Z=1, then (PC)+4+rel⇒PC	REL	18 27 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBGE *rel16*	Long branch if ≥ 0, signed If N⊕V=0, then (PC)+4+rel⇒PC	REL	18 2C qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBGT *rel16*	Long branch if > 0, signed If Z | (N⊕V)=0, then (PC)+4+rel⇒PC	REL	18 2E qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBHI *rel16*	Long branch if higher, unsigned If C | Z=0, then (PC)+4+rel⇒PC	REL	18 22 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBHS *rel16*Same as LBCC	Long branch if higher or same, unsigned; If C=0, (PC)+4+rel⇒PC	REL	18 24 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBLE *rel16*	Long branch if ≤ 0, signed; if Z | (N⊕V)=1, then (PC)+4+rel⇒PC	REL	18 2F qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBLO *rel16*Same as LBCS	Long branch if lower, unsigned; if C=1, then (PC)+4+rel⇒PC	REL	18 25 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBLS *rel16*	Long branch if lower or same, unsigned; If C | Z=1, then (PC)+4+rel⇒PC	REL	18 23 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBLT *rel16*	Long branch if < 0, signed If N⊕V=1, then (PC)+4+rel⇒PC	REL	18 2D qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
LBMI *rel16*	Long branch if minus If N=1, then (PC)+4+rel⇒PC	REL	18 2B qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBNE *rel16*	Long branch if not equal to 0 If Z=0, then (PC)+4+rel⇒PC	REL	18 26 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBPL *rel16*	Long branch if plus If N=0, then (PC)+4+rel⇒PC	REL	18 2A qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBRA *rel16*	Long branch always	REL	18 20 qq rr	OPPP	– – – – – – – –
LBRN *rel16*	Long branch never	REL	18 21 qq rr	OPO	– – – – – – – –
LBVC *rel16*	Long branch if V clear If V=0,then (PC)+4+rel⇒PC	REL	18 28 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBVS *rel16*	Long branch if V set If V=1,then (PC)+4+rel⇒PC	REL	18 29 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –

LDAA #*opr8i*
LDAA *opr8a*
LDAA *opr16a*
LDAA *oprx0_xysppc*
LDAA *oprx9,xysppc*
LDAA *oprx16,xysppc*
LDAA [D,*xysppc*]
LDAA [*oprx16,xysppc*]
	Load A
	(M)⇒A
	or imm⇒A
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			86 ii
			96 dd
			B6 hh ll
			A6 xb
			A6 xb ff
			A6 xb ee ff
			A6 xb
			A6 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					– – – – Δ Δ 0 –

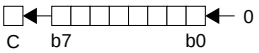
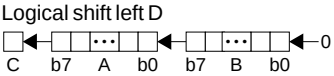
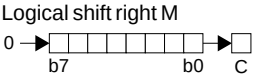
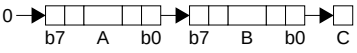
LDAB #*opr8i*
LDAB *opr8a*
LDAB *opr16a*
LDAB *oprx0_xysppc*
LDAB *oprx9,xysppc*
LDAB *oprx16,xysppc*
LDAB [D,*xysppc*]
LDAB [*oprx16,xysppc*]
	Load B
	(M)⇒B
	or imm⇒B
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			C6 ii
			D6 dd
			F6 hh ll
			E6 xb
			E6 xb ff
			E6 xb ee ff
			E6 xb
			E6 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					– – – – Δ Δ 0 –

LDD #*opr16i*
LDD *opr8a*
LDD *opr16a*
LDD *oprx0_xysppc*
LDD *oprx9,xysppc*
LDD *oprx16,xysppc*
LDD [D,*xysppc*]
LDD [*oprx16,xysppc*]
	Load D
	(M:M+1)⇒A:B
	or imm⇒A:B
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CC jj kk
			DC dd
			FC hh ll
			EC xb
			EC xb ff
			EC xb ee ff
			EC xb
			EC xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					– – – – Δ Δ 0 –

LDS #*opr16i*
LDS *opr8a*
LDS *opr16a*
LDS *oprx0_xysppc*
LDS *oprx9,xysppc*
LDS *oprx16,xysppc*
LDS [D,*xysppc*]
LDS [*oprx16,xysppc*]
	Load SP
	(M:M+1)⇒SP
	or imm⇒SP
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CF jj kk
			DF dd
			FF hh ll
			EF xb
			EF xb ff
			EF xb ee ff
			EF xb
			EF xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					– – – – Δ Δ 0 –

LDX #*opr16i*
LDX *opr8a*
LDX *opr16a*
LDX *oprx0_xysppc*
LDX *oprx9,xysppc*
LDX *oprx16,xysppc*
LDX [D,*xysppc*]
LDX [*oprx16,xysppc*]
	Load X
	(M:M+1)⇒X
	or imm⇒X
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CE jj kk
			DE dd
			FE hh ll
			EE xb
			EE xb ff
			EE xb ee ff
			EE xb
			EE xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					– – – – Δ Δ 0 –

LDY #*opr16i*
LDY *opr8a*
LDY *opr16a*
LDY *oprx0_xysppc*
LDY *oprx9,xysppc*
LDY *oprx16,xysppc*
LDY [D,*xysppc*]
LDY [*oprx16,xysppc*]
	Load Y
	(M:M+1)⇒Y
	or imm⇒Y
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CD jj kk
			DD dd
			FD hh ll
			ED xb
			ED xb ff
			ED xb ee ff
			ED xb
			ED xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					– – – – Δ Δ 0 –

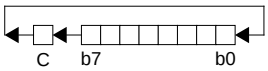
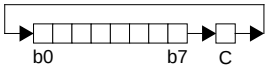
Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
LEAS *oprx0_xysppc* LEAS *oprx9,xysppc* LEAS *oprx16,xysppc*	Load effective address into SP EA⇒SP	IDX IDX1 IDX2	1B xb 1B xb ff 1B xb ee ff	Pf PO PP	– – – – – – – –
LEAX *oprx0_xysppc* LEAX *oprx9,xysppc* LEAX *oprx16,xysppc*	Load effective address into X EA⇒X	IDX IDX1 IDX2	1A xb 1A xb ff 1A xb ee ff	Pf PO PP	– – – – – – – –
LEAY *oprx0_xysppc* LEAY *oprx9,xysppc* LEAY *oprx16,xysppc*	Load effective address into Y EA⇒Y	IDX IDX1 IDX2	19 xb 19 xb ff 19 xb ee ff	Pf PO PP	– – – – – – – –
LSL *opr16a*Same as ASL LSL *oprx0_xysppc* LSL *oprx9,xysppc* LSL *oprx16,xysppc* LSL [D,*xysppc*] LSL [*oprx16,xysppc*] LSLASame as ASLA LSLBSame as ASLB	Logical shift left M  Logical shift left A Logical shift left B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff 48 58	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	– – – – Δ Δ Δ Δ
LSLDSame as ASLD	Logical shift left D  	INH	59	O	– – – – Δ Δ Δ Δ
LSR *opr16a* LSR *oprx0_xysppc* LSR *oprx9,xysppc* LSR *oprx16,xysppc* LSR [D,*xysppc*] LSR [*oprx16,xysppc*] LSRA LSRB	Logical shift right M  Logical shift right A Logical shift right B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	74 hh ll 64 xb 64 xb ff 64 xb ee ff 64 xb 64 xb ee ff 44 54	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – – 0 Δ Δ Δ
LSRD	Logical shift right D  	INH	49	O	– – – – 0 Δ Δ Δ
MAXA *oprx0_xysppc* MAXA *oprx9,xysppc* MAXA *oprx16,xysppc* MAXA [D,*xysppc*] MAXA [*oprx16,xysppc*]	Maximum in A; put larger of 2 unsigned 8-bit values in A MAX[(A), (M)]⇒A N, Z, V, C bits reflect result of internal compare [(A)–(M)]	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 18 xb 18 18 xb ff 18 18 xb ee ff 18 18 xb 18 18 xb ee ff	OrPf OrPO OfrPP OfIfrPf OfIPrPf	– – – – Δ Δ Δ Δ
MAXM *oprx0_xysppc* MAXM *oprx9,xysppc* MAXM *oprx16,xysppc* MAXM [D,*xysppc*] MAXM [*oprx16,xysppc*]	Maximum in M; put larger of 2 unsigned 8-bit values in M MAX[(A), (M)]⇒M N, Z, V, C bits reflect result of internal compare [(A)–(M)]	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1C xb 18 1C xb ff 18 1C xb ee ff 18 1C xb 18 1C xb ee ff	OrPw OrPwO OfrPwP OfIfrPw OfIPrPw	– – – – Δ Δ Δ Δ
MEM	Determine grade of membership; μ (grade)⇒M_Y; (X)+4⇒X; (Y)+1⇒Y If (A)<P1 or (A)>P2, then μ=0; else μ= MIN[((A)–P1)×S1, (P2–(A))×S2, $FF] (A)=current crisp input value; X points at 4 data bytes (P1, P2, S1, S2) of a trapezoidal membership function; Y points at fuzzy input (RAM location)	Special	01	RRfOw	– – ? – ? ? ? ?
MINA *oprx0_xysppc* MINA *oprx9,xysppc* MINA *oprx16,xysppc* MINA [D,*xysppc*] MINA [*oprx16,xysppc*]	Minimum in A; put smaller of 2 unsigned 8-bit values in A MIN[(A), (M)]⇒A N, Z, V, C bits reflect result of internal compare [(A)–(M)]	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 19 xb 18 19 xb ff 18 19 xb ee ff 18 19 xb 18 19 xb ee ff	OrPf OrPO OfrPP OfIfrPf OfIPrPf	– – – – Δ Δ Δ Δ
MINM *oprx0_xysppc* MINM *oprx9,xysppc* MINM *oprx16,xysppc* MINM [D,*xysppc*] MINM [*oprx16,xysppc*]	Minimum in N; put smaller of two unsigned 8-bit values in M MIN[(A), (M)]⇒M N, Z, V, C bits reflect result of internal compare [(A)–(M)]	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1D xb 18 1D xb ff 18 1D xb ee ff 18 1D xb 18 1D xb ee ff	OrPw OrPwO OfrPwP OfIfrPw OfIPrPw	– – – – Δ Δ Δ Δ

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C

MOVB #*opr8*, *opr16a*
MOVB #*opr8i*, *oprx0_xysppc*
MOVB *opr16a*, *opr16a*
MOVB *opr16a*, *oprx0_xysppc*
MOVB *oprx0_xysppc*, *opr16a*
MOVB *oprx0_xysppc*, *oprx0_xysppc*
	Move byte
	Memory-to-memory 8-bit byte-move
	$(M_1) \Rightarrow M_2$
	First operand specifies byte to move
		IMM-EXT
		IMM-IDX
		EXT-EXT
		EXT-IDX
		IDX-EXT
		IDX-IDX
			18 0B ii hh ll
			18 08 xb ii
			18 0C hh ll hh ll
			18 09 xb hh ll
			18 0D xb hh ll
			18 0A xb xb
				OPwP
				OPwO
				OrPwPO
				OPrPw
				OrPwP
				OrPwO
					– – – – – – – –

MOVW #*oprx16*, *opr16a*
MOVW #*opr16i*, *oprx0_xysppc*
MOVW *opr16a*, *opr16a*
MOVW *opr16a*, *oprx0_xysppc*
MOVW *oprx0_xysppc*, *opr16a*
MOVW *oprx0_xysppc*, *oprx0_xysppc*
	Move word
	Memory-to-memory 16-bit word-move
	$(M_1{:}M_1{+}1) \Rightarrow M_2{:}M_2{+}1$
	First operand specifies word to move
		IMM-EXT
		IMM-IDX
		EXT-EXT
		EXT-IDX
		IDX-EXT
		IDX-IDX
			18 03 jj kk hh ll
			18 00 xb jj kk
			18 04 hh ll hh ll
			18 01 xb hh ll
			18 05 xb hh ll
			18 02 xb xb
				OPWPO
				OPPW
				ORPWPO
				OPRPW
				ORPWP
				ORPWO
					– – – – – – – –

MUL	Multiply, unsigned $(A) \times (B) \Rightarrow A{:}B$; 8 by 8-bit	INH	12	O	– – – – – – – Δ
NEG *opr16a* NEG *oprx0_xysppc* NEG *oprx9,xysppc* NEG *oprx16,xysppc* NEG [D,*xysppc*] NEG [*oprx16,xysppc*] NEGA NEGB	Negate M; $0-(M) \Rightarrow M$ or $(\overline{M})+1 \Rightarrow M$ Negate A; $0-(A) \Rightarrow A$ or $(\overline{A})+1 \Rightarrow A$ Negate B; $0-(B) \Rightarrow B$ or $(\overline{B})+1 \Rightarrow B$	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	70 hh ll 60 xb 60 xb ff 60 xb ee ff 60 xb 60 xb ee ff 40 50	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – – Δ Δ Δ Δ
NOP	No operation	INH	A7	O	– – – – – – – –
ORAA #*opr8i* ORAA *opr8a* ORAA *opr16a* ORAA *oprx0_xysppc* ORAA *oprx9,xysppc* ORAA *oprx16,xysppc* ORAA [D,*xysppc*] ORAA [*oprx16,xysppc*]	OR accumulator A $(A) \mid (M) \Rightarrow A$ or $(A) \mid imm \Rightarrow A$	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8A ii 9A dd BA hh ll AA xb AA xb ff AA xb ee ff AA xb AA xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – – Δ Δ 0 –
ORAB #*opr8i* ORAB *opr8a* ORAB *opr16a* ORAB *oprx0_xysppc* ORAB *oprx9,xysppc* ORAB *oprx16,xysppc* ORAB [D,*xysppc*] ORAB [*oprx16,xysppc*]	OR accumulator B $(B) \mid (M) \Rightarrow B$ or $(B) \mid imm \Rightarrow B$	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CA ii DA dd FA hh ll EA xb EA xb ff EA xb ee ff EA xb EA xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – – Δ Δ 0 –
ORCC #*opr8i*	OR CCR; $(CCR) \mid imm \Rightarrow CCR$	IMM	14 ii	P	⇑ – ⇑ ⇑ ⇑ ⇑ ⇑ ⇑
PSHA	Push A; $(SP)-1 \Rightarrow SP$; $(A) \Rightarrow M_{SP}$	INH	36	Os	– – – – – – – –
PSHB	Push B; $(SP)-1 \Rightarrow SP$; $(B) \Rightarrow M_{SP}$	INH	37	Os	– – – – – – – –
PSHC	Push CCR; $(SP)-1 \Rightarrow SP$; $(CCR) \Rightarrow M_{SP}$	INH	39	Os	– – – – – – – –
PSHD	Push D $(SP)-2 \Rightarrow SP$; $(A{:}B) \Rightarrow M_{SP}{:}M_{SP+1}$	INH	3B	OS	– – – – – – – –
PSHX	Push X $(SP)-2 \Rightarrow SP$; $(X_H{:}X_L) \Rightarrow M_{SP}{:}M_{SP+1}$	INH	34	OS	– – – – – – – –
PSHY	Push Y $(SP)-2 \Rightarrow SP$; $(Y_H{:}Y_L) \Rightarrow M_{SP}{:}M_{SP+1}$	INH	35	OS	– – – – – – – –
PULA	Pull A $(M_{SP}) \Rightarrow A$; $(SP)+1 \Rightarrow SP$	INH	32	ufO	– – – – – – – –
PULB	Pull B $(M_{SP}) \Rightarrow B$; $(SP)+1 \Rightarrow SP$	INH	33	ufO	– – – – – – – –
PULC	Pull CCR $(M_{SP}) \Rightarrow CCR$; $(SP)+1 \Rightarrow SP$	INH	38	ufO	Δ ⇓ Δ Δ Δ Δ Δ
PULD	Pull D $(M_{SP}{:}M_{SP+1}) \Rightarrow A{:}B$; $(SP)+2 \Rightarrow SP$	INH	3A	UfO	– – – – – – – –

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
PULX	Pull X $(M_{SP}:M_{SP+1}) \Rightarrow X_H:X_L$; (SP)+2⇒SP	INH	30	Ufo	– – – – – – – –
PULY	Pull Y $(M_{SP}:M_{SP+1}) \Rightarrow Y_H:Y_L$; (SP)+2⇒SP	INH	31	Ufo	– – – – – – – –
REV	Rule evaluation, unweighted; find smallest rule input; store to rule outputs unless fuzzy output is larger	Special	18 3A	Orf(t^tx)o* ff+Orft^**	– – ? – ? ? Δ ?
*The t^tx loop is executed once for each element in the rule list. The ^ denotes a check for pending interrupt requests. **These are additional cycles caused by an interrupt: ff is the exit sequence and Orft^ is the re-entry sequence.					
REVW	Rule evaluation, weighted; rule weights optional; find smallest rule input; store to rule outputs unless fuzzy output is larger	Special	18 3B	ORf(t^Tx)O* or ORf(r^ffRf)O** ffff+ORft^*** ffff+ORfr^****	– – ? – ? ? Δ !
*With weighting not enabled, the t^Tx loop is executed once for each element in the rule list. The ^ denotes a check for pending interrupt requests. **With weighting enabled, the t^Tx loop is replaced by r^ffRf. ***Additional cycles caused by an interrupt when weighting is not enabled: ffff is the exit sequence and ORft^ is the re-entry sequence. **** Additional cycles caused by an interrupt when weighting is enabled: ffff is the exit sequence and ORfr^ is the re-entry sequence.					
ROL *opr16a* ROL *oprx0_xysppc* ROL *oprx9,xysppc* ROL *oprx16,xysppc* ROL [D,*xysppc*] ROL [*oprx16,xysppc*] ROLA ROLB	Rotate left M  Rotate left A Rotate left B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	75 hh ll 65 xb 65 xb ff 65 xb ee ff 65 xb 65 xb ee ff 45 55	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – – Δ Δ Δ Δ
ROR *opr16a* ROR *oprx0_xysppc* ROR *oprx9,xysppc* ROR *oprx16,xysppc* ROR [D,*xysppc*] ROR [*oprx16,xysppc*] RORA RORB	Rotate right M  Rotate right A Rotate right B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	76 hh ll 66 xb 66 xb ff 66 xb ee ff 66 xb 66 xb ee ff 46 56	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – – Δ Δ Δ Δ
RTC	Return from call; $(M_{SP}) \Rightarrow$ PPAGE (SP)+1⇒SP; $(M_{SP}:M_{SP+1}) \Rightarrow PC_H:PC_L$ (SP)+2⇒SP	INH	0A	uUnfPPP	– – – – – – – –
RTI	Return from interrupt $(M_{SP}) \Rightarrow$ CCR; (SP)+1⇒SP $(M_{SP}:M_{SP+1}) \Rightarrow$ B:A;(SP)+2⇒SP $(M_{SP}:M_{SP+1}) \Rightarrow X_H:X_L$;(SP)+4⇒SP $(M_{SP}:M_{SP+1}) \Rightarrow PC_H:PC_L$;(SP)+2⇒SP $(M_{SP}:M_{SP+1}) \Rightarrow Y_H:Y_L$;(SP)+4⇒SP	INH	0B	uUUUUPPP or uUUUUfVfPPP*	Δ ⇓ Δ Δ Δ Δ Δ Δ
*RTI takes 11 cycles if an interrupt is pending.					
RTS	Return from subroutine $(M_{SP}:M_{SP+1}) \Rightarrow PC_H:PC_L$; (SP)+2⇒SP	INH	3D	UfPPP	– – – – – – – –
SBA	Subtract B from A; (A)–(B)⇒A	INH	18 16	OO	– – – – Δ Δ Δ Δ
SBCA #*opr8i* SBCA *opr8a* SBCA *opr16a* SBCA *oprx0_xysppc* SBCA *oprx9,xysppc* SBCA *oprx16,xysppc* SBCA [D,*xysppc*] SBCA [*oprx16,xysppc*]	Subtract with carry from A (A)–(M)–C⇒A or (A)–imm–C⇒A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	82 ii 92 dd B2 hh ll A2 xb A2 xb ff A2 xb ee ff A2 xb A2 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – – Δ Δ Δ Δ

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
SBCB #*opr8i* SBCB *opr8a* SBCB *opr16a* SBCB *oprx0_xysppc* SBCB *oprx9,xysppc* SBCB *oprx16,xysppc* SBCB [D,*xysppc*] SBCB [*oprx16,xysppc*]	Subtract with carry from B (B)–(M)–C⇒B or (B)–imm–C⇒B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C2 ii D2 dd F2 hh ll E2 xb E2 xb ff E2 xb ee ff E2 xb E2 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – – Δ Δ Δ Δ
SECSame as ORCC #$01	Set C bit	IMM	14 01	P	– – – – – – – 1
SEISame as ORCC #$10	Set I bit	IMM	14 10	P	– – – 1 – – – –
SEVSame as ORCC #$02	Set V bit	IMM	14 02	P	– – – – – – 1 –
SEX *abc,dxysp*Same as TFR r1, r2	Sign extend; 8-bit r1 to 16-bit r2 $00:(r1)⇒r2 if bit 7 of r1 is 0 $FF:(r1)⇒r2 if bit 7 of r1 is 1	INH	B7 eb	P	– – – – – – – –
STAA *opr8a* STAA *opr16a* STAA *oprx0_xysppc* STAA *oprx9,xysppc* STAA *oprx16,xysppc* STAA [D,*xysppc*] STAA [*oprx16,xysppc*]	Store accumulator A (A)⇒M	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5A dd 7A hh ll 6A xb 6A xb ff 6A xb ee ff 6A xb 6A xb ee ff	Pw PwO Pw PwO PwP PIfw PIPw	– – – – Δ Δ 0 –
STAB *opr8a* STAB *opr16a* STAB *oprx0_xysppc* STAB *oprx9,xysppc* STAB *oprx16,xysppc* STAB [D,*xysppc*] STAB [*oprx16,xysppc*]	Store accumulator B (B)⇒M	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5B dd 7B hh ll 6B xb 6B xb ff 6B xb ee ff 6B xb 6B xb ee ff	Pw PwO Pw PwO PwP PIfw PIPw	– – – – Δ Δ 0 –
STD *opr8a* STD *opr16a* STD *oprx0_xysppc* STD *oprx9,xysppc* STD *oprx16,xysppc* STD [D,*xysppc*] STD [*oprx16,xysppc*]	Store D (A:B)⇒M:M+1	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5C dd 7C hh ll 6C xb 6C xb ff 6C xb ee ff 6C xb 6C xb ee ff	PW PWO PW PWO PWP PIfW PIPW	– – – – Δ Δ 0 –
STOP	Stop processing; (SP)–2⇒SP RTN_H:RTN_L⇒M_{SP}:M_{SP+1} (SP)–2⇒SP; (Y_H:Y_L)⇒M_{SP}:M_{SP+1} (SP)–2⇒SP; (X_H:X_L)⇒M_{SP}:M_{SP+1} (SP)–2⇒SP; (B:A)⇒M_{SP}:M_{SP+1} (SP)–1⇒SP; (CCR)⇒M_{SP} Stop all clocks	INH	18 3E	00SSSSsf (enter stop mode) fVfPPP (exit stop mode) ff (continue stop mode) 00 (if stop mode disabled by S=1)	– – – – – – – –
STS *opr8a* STS *opr16a* STS *oprx0_xysppc* STS *oprx9,xysppc* STS *oprx16,xysppc* STS [D,*xysppc*] STS [*oprx16,xysppc*]	Store SP (SP_H:SP_L)⇒M:M+1	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5F dd 7F hh ll 6F xb 6F xb ff 6F xb ee ff 6F xb 6F xb ee ff	PW PWO PW PWO PWP PIfW PIPW	– – – – Δ Δ 0 –
STX *opr8a* STX *opr16a* STX *oprx0_xysppc* STX *oprx9,xysppc* STX *oprx16,xysppc* STX [D,*xysppc*] STX [*oprx16,xysppc*]	Store X (X_H:X_L)⇒M:M+1	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5E dd 7E hh ll 6E xb 6E xb ff 6E xb ee ff 6E xb 6E xb ee ff	PW PWO PW PWO PWP PIfW PIPW	– – – – Δ Δ 0 –

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
STY *opr8a* STY *opr16a* STY *oprx0_xysppc* STY *oprx9,xysppc* STY *oprx16,xysppc* STY [D,*xysppc*] STY [*oprx16,xysppc*]	Store Y $(Y_H:Y_L) \Rightarrow M:M+1$	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5D dd 7D hh ll 6D xb 6D xb ff 6D xb ee ff 6D xb 6D xb ee ff	PW PWO PW PWO PWP PIfW PIPW	– – – – Δ Δ 0 –

SUBA #*opr8i*
SUBA *opr8a*
SUBA *opr16a*
SUBA *oprx0_xysppc*
SUBA *oprx9,xysppc*
SUBA *oprx16,xysppc*
SUBA [D,*xysppc*]
SUBA [*oprx16,xysppc*]
	Subtract from A
	$(A)-(M) \Rightarrow A$
	or $(A)-imm \Rightarrow A$
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			80 ii
			90 dd
			B0 hh ll
			A0 xb
			A0 xb ff
			A0 xb ee ff
			A0 xb
			A0 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					– – – – Δ Δ Δ Δ

SUBB #*opr8i*
SUBB *opr8a*
SUBB *opr16a*
SUBB *oprx0_xysppc*
SUBB *oprx9,xysppc*
SUBB *oprx16,xysppc*
SUBB [D,*xysppc*]
SUBB [*oprx16,xysppc*]
	Subtract from B
	$(B)-(M) \Rightarrow B$
	or $(B)-imm \Rightarrow B$
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			C0 ii
			D0 dd
			F0 hh ll
			E0 xb
			E0 xb ff
			E0 xb ee ff
			E0 xb
			E0 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					– – – – Δ Δ Δ Δ

SUBD #*opr16i*
SUBD *opr8a*
SUBD *opr16a*
SUBD *oprx0_xysppc*
SUBD *oprx9,xysppc*
SUBD *oprx16,xysppc*
SUBD [D,*xysppc*]
SUBD [*oprx16,xysppc*]
	Subtract from D
	$(A:B)-(M:M+1) \Rightarrow A:B$
	or $(A:B)-imm \Rightarrow A:B$
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			83 jj kk
			93 dd
			B3 hh ll
			A3 xb
			A3 xb ff
			A3 xb ee ff
			A3 xb
			A3 xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					– – – – Δ Δ Δ Δ

SWI	Software interrupt; $(SP)-2 \Rightarrow SP$ $RTN_H:RTN_L \Rightarrow M_{SP}:M_{SP+1}$ $(SP)-2 \Rightarrow SP$; $(Y_H:Y_L) \Rightarrow M_{SP}:M_{SP+1}$ $(SP)-2 \Rightarrow SP$; $(X_H:X_L) \Rightarrow M_{SP}:M_{SP+1}$ $(SP)-2 \Rightarrow SP$; $(B:A) \Rightarrow M_{SP}:M_{SP+1}$ $(SP)-1 \Rightarrow SP$; $(CCR) \Rightarrow M_{SP}$;$1 \Rightarrow I$ (SWI vector)$\Rightarrow$PC	INH	3F	VSPSSPSsP*	– – – 1 – – – –
*The CPU also uses VSPSSPSsP for hardware interrupts and unimplemented opcode traps.					
TAB	Transfer A to B; $(A) \Rightarrow B$	INH	18 0E	OO	– – – – Δ Δ 0 –
TAP	Transfer A to CCR; $(A) \Rightarrow CCR$ Assembled as TFR A, CCR	INH	B7 02	P	Δ ⇓ Δ Δ Δ Δ Δ Δ
TBA	Transfer B to A; $(B) \Rightarrow A$	INH	18 0F	OO	– – – – Δ Δ 0 –
TBEQ *abdxysp,rel9*	Test and branch if equal to 0 If (counter)=0, then $(PC)+2+rel \Rightarrow PC$	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	– – – – – – – –
TBL *oprx0_xysppc*	Table lookup and interpolate, 8-bit $(M)+[(B) \times ((M+1)-(M))] \Rightarrow A$	IDX	18 3D xb	ORfffP	– – – – Δ Δ – Δ
TBNE *abdxysp,rel9*	Test and branch if not equal to 0 If (counter)≠0, then $(PC)+2+rel \Rightarrow PC$	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	– – – – – – – –
TFR *abcdxysp,abcdxysp*	Transfer from register to register (r1)⇒r2r1 and r2 same size $00:(r1) \Rightarrow r2r1$=8-bit; r2=16-bit $(r1_L) \Rightarrow r2r1$=16-bit; r2=8-bit	INH	B7 eb	P	– – – – – – – – or Δ ⇓ Δ Δ Δ Δ Δ Δ
TPASame as TFR CCR ,A	Transfer CCR to A; $(CCR) \Rightarrow A$	INH	B7 20	P	– – – – – – – –

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
TRAP *trapnum*	Trap unimplemented opcode; (SP)–2⇒SP RTN_H:RTN_L⇒M_{SP}:M_{SP+1} (SP)–2⇒SP; (Y_H:Y_L)⇒M_{SP}:M_{SP+1} (SP)–2⇒SP; (X_H:X_L)⇒M_{SP}:M_{SP+1} (SP)–2⇒SP; (B:A)⇒M_{SP}:M_{SP+1} (SP)–1⇒SP; (CCR)⇒M_{SP} 1⇒I; (trap vector)⇒PC	INH	18 tn tn = $30–$39 or tn = $40–$FF	OVSPSSPSsP	– – – 1 – – – –
TST *opr16a* TST *oprx0_xysppc* TST *oprx9,xysppc* TST *oprx16,xysppc* TST [D,*xysppc*] TST [*oprx16,xysppc*] TSTA TSTB	Test M; (M)–0 Test A; (A)–0 Test B; (B)–0	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	F7 hh ll E7 xb E7 xb ff E7 xb ee ff E7 xb E7 xb ee ff 97 D7	rPO rPf rPO frPP fIfrPf fIPrPf O O	– – – – Δ Δ 0 0
TSXSame as TFR SP,X	Transfer SP to X; (SP)⇒X	INH	B7 75	P	– – – – – – – –
TSYSame as TFR SP,Y	Transfer SP to Y; (SP)⇒Y	INH	B7 76	P	– – – – – – – –
TXSSame as TFR X,SP	Transfer X to SP; (X)⇒SP	INH	B7 57	P	– – – – – – – –
TYSSame as TFR Y,SP	Transfer Y to SP; (Y)⇒SP	INH	B7 67	P	– – – – – – – –
WAI	Wait for interrupt; (SP)–2⇒SP RTN_H:RTN_L⇒M_{SP}:M_{SP+1} (SP)–2⇒SP; (Y_H:Y_L)⇒M_{SP}:M_{SP+1} (SP)–2⇒SP; (X_H:X_L)⇒M_{SP}:M_{SP+1} (SP)–2⇒SP; (B:A)⇒M_{SP}:M_{SP+1} (SP)–1⇒SP; (CCR)⇒M_{SP}	INH	3E	OSSSSsf (before interrupt) fVfPPP (after interrupt)	– – – – – – – – or – – – 1 – – – – or – 1 – 1 – – – –
WAV	Calculate weighted average; sum of products (SOP) and sum of weights (SOW)* $\sum_{i=1}^{B} S_i F_i \Rightarrow Y{:}D$ $\sum_{i=1}^{B} F_i \Rightarrow X$	Special	18 3C	Of(frr^ffff)O** SSS+UUUrr^***	– – ? – ? Δ ? ?
*Initialize B, X, and Y: B=number of elements; X points at first element in S_i list; Y points at first element in F_i list. All S_i and F_i elements are 8-bit values. **The frr^ffff sequence is the loop for one iteration of SOP and SOW accumulation. The ^ denotes a check for pending interrupt requests. ***Additional cycles caused by an interrupt: SSS is the exit sequence and UUUrr^ is the re-entry sequence. Intermediate values use six stack bytes.					
wavr*	Resume executing interrupted WAV	Special	3C	UUUrr^ffff(frr^ffff)O** SSS+UUUrr^***	– – ? – ? Δ ? ?
*wavr is a pseudoinstruction that recovers intermediate results from the stack rather than initializing them to 0. **The frr^ffff sequence is the loop for one iteration of SOP and SOW recovery. The ^ denotes a check for pending interrupt requests. ***These are additional cycles caused by an interrupt: SSS is the exit sequence and UUUrr^ is the re-entry sequence.					
XGDXSame as EXG D, X	Exchange D with X; (D)⇔(X)	INH	B7 C5	P	– – – – – – – –
XGDYSame as EXG D, Y	Exchange D with Y; (D)⇔(Y)	INH	B7 C6	P	– – – – – – – –

1.8.1 Register and Memory Notation

Table 1-3 Register and Memory Notation

A or *a*	Accumulator A
An	Bit n of accumulator A
B or *b*	Accumulator B
Bn	Bit n of accumulator B
D or *d*	Accumulator D
Dn	Bit n of accumulator D
X or *x*	Index register X
X_H	High byte of index register X
X_L	Low byte of index register X
Xn	Bit n of index register X
Y or *y*	Index register Y
Y_H	High byte of index register Y
Y_L	Low byte of index register Y
Yn	Bit n of index register Y
SP or *sp*	Stack pointer
SPn	Bit n of stack pointer
PC or *pc*	Program counter
PC_H	High byte of program counter
PC_L	Low byte of program counter
CCR or *c*	Condition code register
M	Address of 8-bit memory location
Mn	Bit n of byte at memory location M
Rn	Bit n of the result of an arithmetic or logical operation
In	Bit n of the intermediate result of an arithmetic or logical operation
RTN_H	High byte of return address
RTN_L	Low byte of return address
( )	Contents of

1.8.2 Source Form Notation

The **Source Form** column of the summary in **Table 1-2** gives essential information about assembler source forms. For complete information about writing source files for a particular assembler, refer to the documentation provided by the assembler vendor.

Everything in the **Source Form** column, *except expressions in italic characters*, is literal information which must appear in the assembly source file exactly as shown. The initial 3- to 5-letter mnemonic is always a literal expression. All commas, pound signs (#), parentheses, square brackets ([or]), plus signs (+), minus signs (–), and the register designation (A, B, D), are literal characters.

The groups of italic characters shown in **Table 1-4** represent variable information to be supplied by the programmer. These groups can include any alphanumeric character or the underscore character, but cannot

include a space or comma. For example, the groups *xysppc* and *oprx0_xysppc* are both valid, but the two groups *oprx0 xysppc* are not valid because there is a space between them.

Table 1-4 Source Form Notation

abc	Register designator for A, B, or CCR
abcdxysp	Register designator for A, B, CCR, D, X, Y, or SP
abd	Register designator for A, B, or D
abdxysp	Register designator for A, B, D, X, Y, or SP
dxysp	Register designator for D, X, Y, or SP
msk8	8-bit mask value Some assemblers require the # symbol before the mask value.
opr8i	8-bit immediate value
opr16i	16-bit immediate value
opr8a	8-bit address value used with direct address mode
opr16a	16-bit address value
oprx0_xysp	Indexed addressing postbyte code: *oprx3,–xysp* — Predecrement X , Y, or SP by 1–8 *oprx3,+xysp* — Preincrement X , Y, or SP by 1–8 *oprx3,xysp–* — Postdecrement X, Y, or SP by 1–8 *oprx3,xysp+* — Postincrement X, Y, or SP by 1–8 *oprx5,xysppc* — 5-bit constant offset from X, Y, SP, or PC *abd,xysppc* — Accumulator A, B, or D offset from X, Y, SP, or PC
oprx3	Any positive integer from 1 to 8 for pre/post increment/decrement
oprx5	Any integer from –16 to +15
oprx9	Any integer from –256 to +255
oprx16	Any integer from –32,768 to +65,535
page	8-bit value for PPAGE register Some assemblers require the # symbol before this value.
rel8	Label of branch destination within –256 to +255 locations
rel9	Label of branch destination within –512 to +511 locations
rel16	Any label within the 64-Kbyte memory space
trapnum	Any 8-bit integer from $30 to $39 or from $40 to $FF
xysp	Register designator for X or Y or SP
xysppc	Register designator for X or Y or SP or PC

1.8.3 Operation Notation

Table 1-5 Operation Notation

+	Add
–	Subtract
•	AND
|	OR
⊕	Exclusive OR
×	Multiply
÷	Divide
:	Concatenate
⇒	Transfer
⇔	Exchange

1.8.4 Address Mode Notation

Table 1-6 Address Mode Notation

INH	Inherent; no operands in instruction stream
IMM	Immediate; operand immediate value in instruction stream
DIR	Direct; operand is lower byte of address from $0000 to $00FF
EXT	Operand is a 16-bit address
REL	Two's complement relative offset; for branch instructions
IDX	Indexed (no extension bytes); includes: 5-bit constant offset from X, Y, SP or PC Pre/post increment/decrement by 1–8 Accumulator A, B, or D offset
IDX1	9-bit signed offset from X, Y, SP, or PC; 1 extension byte
IDX2	16-bit signed offset from X, Y, SP, or PC; 2 extension bytes
[IDX2]	Indexed-indirect; 16-bit offset from X, Y, SP, or PC
[D, IDX]	Indexed-indirect; accumulator D offset from X, Y, SP, or PC

1.8.5 Machine Code Notation

In the **Machine Code (Hex)** column of the summary in **Table 1-2**, digits 0–9 and upper case letters A–F represent hexadecimal values. Pairs of lower-case letters represent 8-bit values as shown in **Table 1-7**.

Table 1-7 Machine Code Notation

dd	8-bit direct address from $0000 to $00FF; high byte is $00
ee	High byte of a 16-bit constant offset for indexed addressing
eb	Exchange/transfer postbyte
ff	Low eight bits of a 9-bit signed constant offset in indexed addressing, or low byte of a 16-bit constant offset in indexed addressing
hh	High byte of a 16-bit extended address
ii	8-bit immediate data value
jj	High byte of a 16-bit immediate data value
kk	Low byte of a 16-bit immediate data value
lb	Loop primitive (DBNE) postbyte
ll	Low byte of a 16-bit extended address
mm	8-bit immediate mask value for bit manipulation instructions; bits that are set indicate bits to be affected
pg	Program page or bank number used in CALL instruction
qq	High byte of a 16-bit relative offset for long branches
tn	Trap number from $30 to $39 or from $40 to $FF
rr	Signed relative offset $80 (–128) to $7F (+127) relative to the byte following the relative offset byte, or low byte of a 16-bit relative offset for long branches
xb	Indexed addressing postbyte

1.8.6 Access Detail Notation

A single-letter code in the **Access Detail** column of **Table 1-2** represents a single CPU access cycle. An upper-case letter indicates a 16-bit access.

Table 1-8 Access Detail Notation

f	Free cycle. During an f cycle, the CPU does not use the bus. An f cycle is always one cycle of the system bus clock. An f cycle can be used by a queue controller or the background debug system to perform a single-cycle access without disturbing the CPU.
g	Read PPAGE register. A g cycle is used only in CALL instructions and is not visible on the external bus. Since PPAGE is an internal 8-bit register, a g cycle is never stretched.
I	Read indirect pointer. Indexed-indirect instructions use the 16-bit indirect pointer from memory to address the instruction operand. An I cycle is a 16-bit read that can be aligned or misaligned. An I cycle is extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. An I cycle is also stretched if it corresponds to a misaligned access to a memory that is not designed for single-cycle misaligned access.
i	Read indirect PPAGE value. An i cycle is used only in indexed-indirect CALL instructions. The 8-bit PPAGE value for the CALL destination is fetched from an indirect memory location. An i cycle is stretched only when controlled by a chip-select circuit that is programmed for slow memory.
n	Write PPAGE register. An n cycle is used only in CALL and RTC instructions to write the destination value of the PPAGE register and is not visible on the external bus. Since the PPAGE register is an internal 8-bit register, an n cycle is never stretched.
O	Optional cycle. An O cycle adjusts instruction alignment in the instruction queue. An O cycle can be a free cycle (f) or a program word access cycle (P). When the first byte of an instruction with an odd number of bytes is misaligned, the O cycle becomes a P cycle to maintain queue order. If the first byte is aligned, the O cycle is an f cycle. The $18 prebyte for a page-two opcode is treated as a special one-byte instruction. If the prebyte is misaligned, the O cycle at the beginning of the instruction becomes a P cycle to maintain queue order. If the prebyte is aligned, the O cycle is an f cycle. If the instruction has an odd number of bytes, it has a second O cycle at the end. If the first O cycle is a P cycle (prebyte misaligned), the second O cycle is an f cycle. If the first O cycle is an f cycle (prebyte aligned), the second O cycle is a P cycle. An O cycle that becomes a P cycle can be extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the program is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. An O cycle that becomes an f cycle is never stretched.
P	Program word access. Program information is fetched as aligned 16-bit words. A P cycle is extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the program is stored externally. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory.
r	8-bit data read. An r cycle is stretched only when controlled by a chip-select circuit programmed for slow memory.
R	16-bit data read. An R cycle is extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. An R cycle is also stretched if it corresponds to a misaligned access to a memory that is not designed for single-cycle misaligned access.
s	Stack 8-bit data. An s cycle is stretched only when controlled by a chip-select circuit programmed for slow memory.

Table 1-8 Access Detail Notation (Continued)

S	Stack 16-bit data. An S cycle is extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the SP is pointing to external memory. There can be additional stretching if the address space is assigned to a chip-select circuit programmed for slow memory. An S cycle is also stretched if it corresponds to a misaligned access to a memory that is not designed for single-cycle misaligned access. The internal RAM is designed to allow single cycle misaligned word access.
w	8-bit data write. A w cycle is stretched only when controlled by a chip-select circuit programmed for slow memory.
W	16-bit data write. A W cycle is extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. A W cycle is also stretched if it corresponds to a misaligned access to a memory that is not designed for single-cycle misaligned access.
u	Unstack 8-bit data. A W cycle is stretched only when controlled by a chip-select circuit programmed for slow memory.
U	Unstack 16-bit data. A U cycle is extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the SP is pointing to external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. A U cycle is also stretched if it corresponds to a misaligned access to a memory that is not designed for single-cycle misaligned access. The internal RAM is designed to allow single-cycle misaligned word access.
V	16-bit vector fetch. Vectors are always aligned 16-bit words. A V cycle is extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the program is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory.
t	8-bit conditional read. A t cycle is either a data read cycle or a free cycle, depending on the data and flow of the REVW instruction. A t cycle is stretched only when controlled by a chip-select circuit programmed for slow memory.
T	16-bit conditional read. A T cycle is either a data read cycle or a free cycle, depending on the data and flow of the REV or REVW instruction. A T cycle is extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. A T cycle is also stretched if it corresponds to a misaligned access to a memory that is not designed for single-cycle misaligned access.
x	8-bit conditional write. An x cycle is either a data write cycle or a free cycle, depending on the data and flow of the REV or REVW instruction. An x cycle is stretched only when controlled by a chip-select circuit programmed for slow memory.
Special Notation for Branch Taken/Not Taken	
PPP/P	A short branch requires three cycles if taken, one cycle if not taken. Since the instruction consists of a single word containing both an opcode and an 8-bit offset, the not-taken case is simple — the queue advances, another program word fetch is made, and execution continues with the next instruction. The taken case requires that the queue be refilled so that execution can continue at a new address. First, the effective address of the destination is determined, then the CPU performs three program word fetches from that address.
OPPP/OPO	A long branch requires four cycles if taken, three cycles if not taken. An O cycle is required because all long branches are page two opcodes and thus include the $18 prebyte. The prebyte is treated as a one-byte instruction. If the prebyte is misaligned, the O cycle is a P cycle; if the prebyte is aligned, the O cycle is an f cycle. As a result, both the taken and not-taken cases use one O cycle for the prebyte. In the not-taken case, the queue must advance so that execution can continue with the next instruction, and another O cycle is required to maintain the queue. The taken case requires that the queue be refilled so that execution can continue at a new address. First, the effective address of the destination is determined, then the CPU performs three program word fetches from that address.

1.8.7 Condition Code State Notation

Table 1-9 Condition Code State Notation

–	Not changed by operation
0	Cleared by operation
1	Set by operation
Δ	Set or cleared by operation
⇓	May be cleared or remain set, but not set by operation
⇑	May be set or remain cleared, but not cleared by operation
?	May be changed by operation but final state not defined
!	Used for a special purpose