Order this document
by CPU12RG/D Rev. 1

Motorola Semiconductor Technical Data

CPU12 Reference Guide

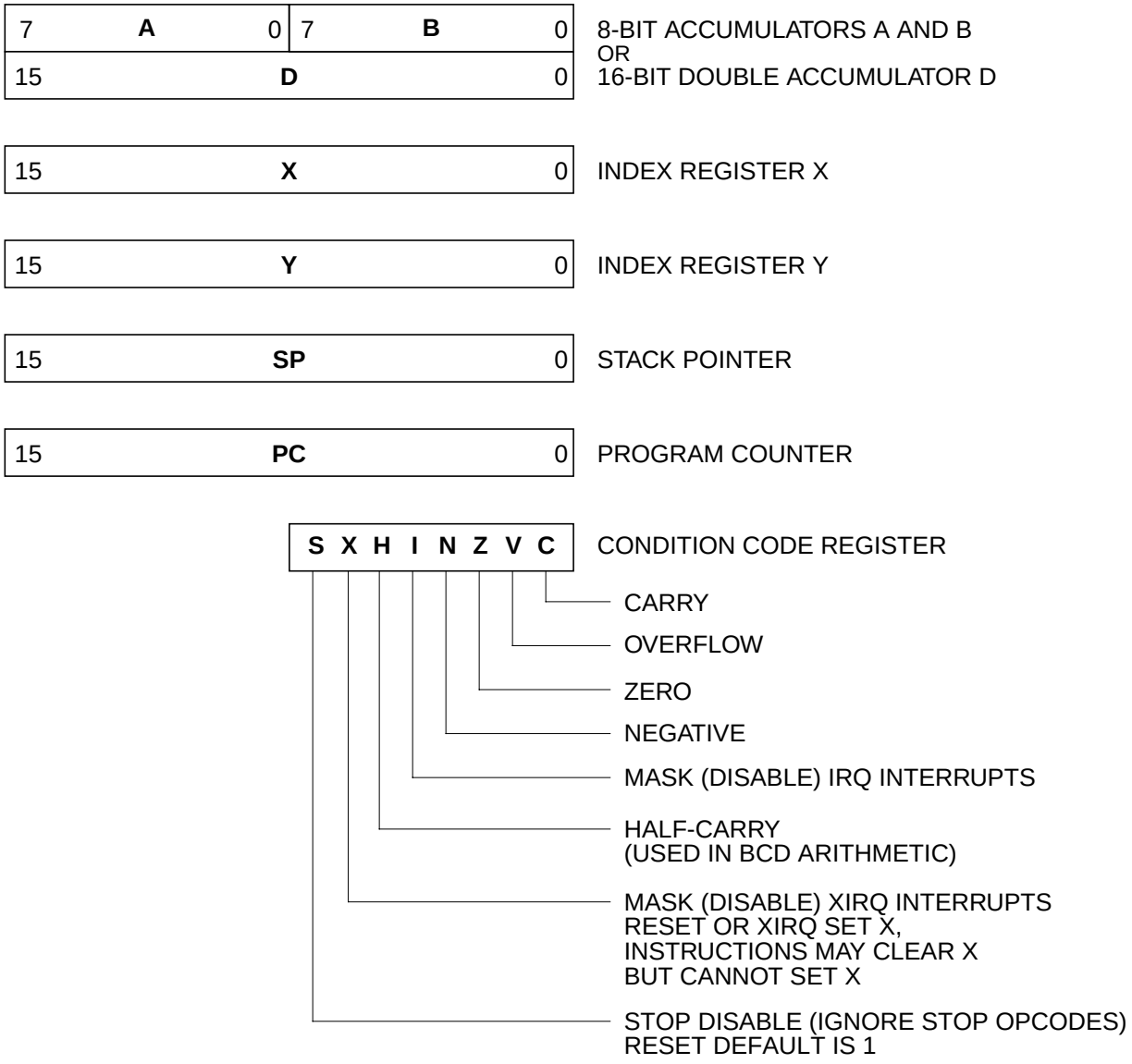
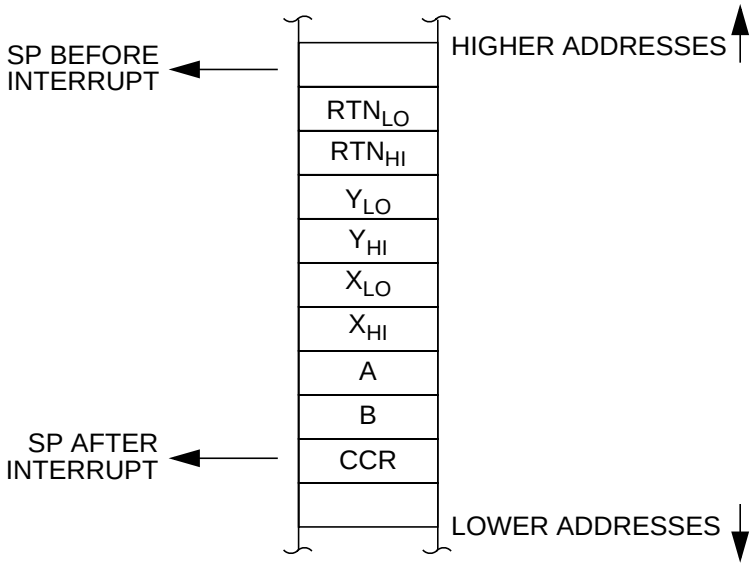


Figure 1. Programming Model

Stack and Memory Layout



STACK UPON ENTRY TO SERVICE ROUTINE
IF SP WAS ODD BEFORE INTERRUPT

SP +8	RTN_{LO}		SP +9
SP +6	Y_{LO}	RTN_{HI}	SP +7
SP +4	X_{LO}	Y_{HI}	SP +5
SP +2	A	X_{HI}	SP +3
SP	CCR	B	SP +1
SP –2			SP –1

STACK UPON ENTRY TO SERVICE ROUTINE
IF SP WAS EVEN BEFORE INTERRUPT

SP +9			SP +10
SP +7	RTN_{HI}	RTN_{LO}	SP +8
SP +5	Y_{HI}	Y_{LO}	SP +6
SP +4	X_{HI}	X_{LO}	SP +4
SP +1	B	A	SP +2
SP –1		CCR	SP

Interrupt Vector Locations

$FFFE, $FFFF	Power-On (POR) or External Reset
$FFFC, $FFFD	Clock Monitor Reset
$FFFA, $FFFB	Computer Operating Properly (COP Watchdog Reset
$FFF8, $FFF9	Unimplemented Opcode Trap
$FFF6, $FFF7	Software Interrupt Instruction (SWI)
$FFF4, $FFF5	XIRQ
$FFF2, $FFF3	IRQ
$FFC0–$FFF1	Device-Specific Interrupt Sources

Notation Used in Instruction Set Summary

Explanation of Italic Expressions in Source Form Column

abc — A or B or CCR

abcdxys — A or B or CCR or D or X or Y or SP. Some assemblers also allow T2 or T3.

abd — A or B or D

abdxys — A or B or D or X or Y or SP

dxys — D or X or Y or SP

msk8 — 8-bit mask, some assemblers require # symbol before value

opr8i — 8-bit immediate value

opr16i — 16-bit immediate value

opr8a — 8-bit address used with direct address mode

opr16a — 16-bit address value

oprx0_xysp — Indexed addressing postbyte code:

oprx3,–*xys*	Predecrement X or Y or SP by 1 . . . 8
oprx3,+*xys*	Preincrement X or Y or SP by 1 . . . 8
oprx3,*xys*–	Postdecrement X or Y or SP by 1 . . . 8
oprx3,*xys*+	Postincrement X or Y or SP by 1 . . . 8
oprx5,xysp	5-bit constant offset from X or Y or SP or PC
abd,*xysp*	Accumulator A or B or D offset from X or Y or SP or PC

oprx3 — Any positive integer 1 . . . 8 for pre/post increment/decrement

oprx5 — Any value in the range –16 . . . +15

oprx9 — Any value in the range –256 . . . +255

oprx16 — Any value in the range –32,768 . . . 65,535

page — 8-bit value for PPAGE, some assemblers require # 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 64K memory space

trapnum — Any 8-bit value in the range $30-$39 or $40-$FF

xys — X or Y or SP

xysp — X or Y or SP or PC

Address Modes

IMM — Immediate
IDX — Indexed (no extension bytes) includes:
5-bit constant offset
Pre/post increment/decrement by 1 . . . 8
Accumulator A, B, or D offset
IDX1 — 9-bit signed offset (1 extension byte)
IDX2 — 16-bit signed offset (2 extension bytes)
[D, IDX] — Indexed indirect (accumulator D offset)
[IDX2] — Indexed indirect (16-bit offset)
INH — Inherent (no operands in object code)
REL — 2’s complement relative offset (branches)

Machine Coding

`dd` — 8-bit direct address $0000 to $00FF. (High byte assumed to be $00).
`ee` — High-order byte of a 16-bit constant offset for indexed addressing.
`eb` — Exchange/Transfer post-byte. See **Table 3** on page 23.
`ff` — Low-order eight bits of a 9-bit signed constant offset for indexed addressing, or low-order byte of a 16-bit constant offset for indexed addressing.
`hh` — High-order byte of a 16-bit extended address.
`ii` — 8-bit immediate data value.
`jj` — High-order byte of a 16-bit immediate data value.
`kk` — Low-order byte of a 16-bit immediate data value.
`lb` — Loop primitive (DBNE) post-byte. See **Table 4** on page 24.
`ll` — Low-order byte of a 16-bit extended address.
`mm` — 8-bit immediate mask value for bit manipulation instructions. Set bits indicate bits to be affected.
`pg` — Program page (bank) number used in CALL instruction.
`qq` — High-order byte of a 16-bit relative offset for long branches.
`tn` — Trap number $30–$39 or $40–$FF.
`rr` — Signed relative offset $80 (–128) to $7F (+127). Offset relative to the byte following the relative offset byte, or low-order byte of a 16-bit relative offset for long branches.
`xb` — Indexed addressing post-byte. See **Table 1** on page 21 and **Table 2** on page 22.

Access Detail

Each code letter equals one CPU cycle. Uppercase = 16-bit operation and lowercase = 8-bit operation. For complex sequences see the *CPU12 Reference Manual* (CPU12RM/AD).

- `f` — Free cycle, CPU doesn't use bus
- `g` — Read PPAGE internally
- `I` — Read indirect pointer (indexed indirect)
- `i` — Read indirect PPAGE value (call indirect)
- `n` — Write PPAGE internally
- `O` — Optional program word fetch (P) if instruction is misaligned and has an odd number of bytes of object code — otherwise, appears as a free cycle (f)
- `P` — Program word fetch (always an aligned word read)
- `r` — 8-bit data read
- `R` — 16-bit data read
- `s` — 8-bit stack write
- `S` — 16-bit stack write
- `w` — 8-bit data write
- `W` — 16-bit data write
- `u` — 8-bit stack read
- `U` — 16-bit stack read
- `V` — 16-bit vector fetch
- `t` — 8-bit conditional read (or free cycle)
- `T` — 16-bit conditional read (or free cycle)
- `x` — 8-bit conditional write

Special Cases

- `PPP/P` — Short branch, PPP if branch taken, P if not
- `OPPP/OPO` — Long branch, OPPP if branch taken, OPO if not

Condition Codes Columns

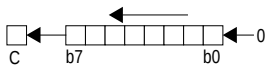
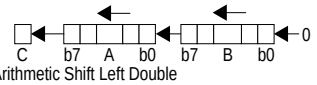
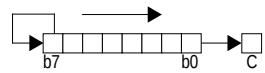
- – — Status bit not affected by operation.
- 0 — Status bit cleared by operation.
- 1 — Status bit set by operation.
- Δ — Status bit affected by operation.
- ⇓ — Status bit may be cleared or remain set, but is not set by operation.
- ⇑ — Status bit may be set or remain cleared, but is not cleared by operation.
- ? — Status bit may be changed by operation but the final state is not defined.
- ! — Status bit used for a special purpose.

Instruction Set Summary

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
ABA	(A) + (B) ⇒ A Add Accumulators A and B	INH	18 06	OO	–	–	Δ	–	Δ	Δ	Δ	Δ
ABX	(B) + (X) ⇒ X *Translates to* LEAX B,X	IDX	1A E5	PP[1]	–	–	–	–	–	–	–	–
ABY	(B) + (Y) ⇒ Y *Translates to* LEAY B,Y	IDX	19 ED	PP[1]	–	–	–	–	–	–	–	–
ADCA *#opr8i* ADCA *opr8a* ADCA *opr16a* ADCA *oprx0_xysp* ADCA *oprx9,xysp* ADCA *oprx16,xysp* ADCA [D,*xysp*] ADCA [*oprx16,xysp*]	(A) + (M) + C ⇒ A Add with Carry to 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	Δ	–	Δ	Δ	Δ	Δ
ADCB *#opr8i* ADCB *opr8a* ADCB *opr16a* ADCB *oprx0_xysp* ADCB *oprx9,xysp* ADCB *oprx16,xysp* ADCB [D,*xysp*] ADCB [*oprx16,xysp*]	(B) + (M) + C ⇒ B Add with Carry to 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	Δ	–	Δ	Δ	Δ	Δ
ADDA *#opr8i* ADDA *opr8a* ADDA *opr16a* ADDA *oprx0_xysp* ADDA *oprx9,xysp* ADDA *oprx16,xysp* ADDA [D,*xysp*] ADDA [*oprx16,xysp*]	(A) + (M) ⇒ A Add without Carry to 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	Δ	–	Δ	Δ	Δ	Δ
ADDB *#opr8i* ADDB *opr8a* ADDB *opr16a* ADDB *oprx0_xysp* ADDB *oprx9,xysp* ADDB *oprx16,xysp* ADDB [D,*xysp*] ADDB [*oprx16,xysp*]	(B) + (M) ⇒ B Add without Carry to 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	Δ	–	Δ	Δ	Δ	Δ
ADDD *#opr16i* ADDD *opr8a* ADDD *opr16a* ADDD *oprx0_xysp* ADDD *oprx9,xysp* ADDD *oprx16,xysp* ADDD [D,*xysp*] ADDD [*oprx16,xysp*]	(A:B) + (M:M+1) ⇒ A:B Add 16-Bit to D (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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	Δ	Δ
ANDA *#opr8i* ANDA *opr8a* ANDA *opr16a* ANDA *oprx0_xysp* ANDA *oprx9,xysp* ANDA *oprx16,xysp* ANDA [D,*xysp*] ANDA [*oprx16,xysp*]	(A) • (M) ⇒ A Logical And A with Memory	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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–
ANDB *#opr8i* ANDB *opr8a* ANDB *opr16a* ANDB *oprx0_xysp* ANDB *oprx9,xysp* ANDB *oprx16,xysp* ANDB [D,*xysp*] ANDB [*oprx16,xysp*]	(B) • (M) ⇒ B Logical And B with Memory	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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–

Note 1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
ANDCC *#opr8i*	(CCR) • (M) ⇒ CCR Logical And CCR with Memory	IMM	10 ii	P	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
ASL *opr16a* ASL *oprx0_xysp* ASL *oprx9,xysp* ASL *oprx16,xysp* ASL [D,*xysp*] ASL [*oprx16,xysp*] ASLA ASLB	 Arithmetic Shift Left Arithmetic Shift Left Accumulator A Arithmetic Shift Left Accumulator 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
ASLD	 Arithmetic Shift Left Double	INH	59	O	–	–	–	–	Δ	Δ	Δ	Δ
ASR *opr16a* ASR *oprx0_xysp* ASR *oprx9,xysp* ASR *oprx16,xysp* ASR [D,*xysp*] ASR [*oprx16,xysp*] ASRA ASRB	 Arithmetic Shift Right Arithmetic Shift Right Accumulator A Arithmetic Shift Right Accumulator 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	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	–	–	–	–	Δ	Δ	Δ	Δ
BCC *rel8*	Branch if Carry Clear (if C = 0)	REL	24 rr	PPP/P[1]	–	–	–	–	–	–	–	–
BCLR *opr*8a, *msk8* BCLR *opr16a*, *msk8* BCLR *oprx*0_xysp, *msk8* BCLR *oprx9,xysp*, *msk8* BCLR *oprx16,xysp*, *msk8*	(M) • ($\overline{mm}$) ⇒ M Clear Bit(s) in Memory	DIR EXT IDX IDX1 IDX2	4D dd mm 1D hh ll mm 0D xb mm 0D xb ff mm 0D xb ee ff mm	rPOw rPPw rPOw rPwP frPwOP	–	–	–	–	Δ	Δ	0	–
BCS *rel8*	Branch if Carry Set (if C = 1)	REL	25 rr	PPP/P[1]	–	–	–	–	–	–	–	–
BEQ *rel8*	Branch if Equal (if Z = 1)	REL	27 rr	PPP/P[1]	–	–	–	–	–	–	–	–
BGE *rel8*	Branch if Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	2C rr	PPP/P[1]	–	–	–	–	–	–	–	–
BGND	Place CPU in Background Mode see *CPU12 Reference Manual*	INH	00	VfPPP	–	–	–	–	–	–	–	–
BGT *rel8*	Branch if Greater Than (if Z ✣ (N ⊕ V) = 0) (signed)	REL	2E rr	PPP/P[1]	–	–	–	–	–	–	–	–
BHI *rel8*	Branch if Higher (if C ✣ Z = 0) (unsigned)	REL	22 rr	PPP/P[1]	–	–	–	–	–	–	–	–
BHS *rel8*	Branch if Higher or Same (if C = 0) (unsigned) same function as BCC	REL	24 rr	PPP/P[1]	–	–	–	–	–	–	–	–
BITA *#opr8i* BITA *opr8a* BITA *opr16a* BITA *oprx0_xysp* BITA *oprx9,xysp* BITA *oprx16,xysp* BITA [D,*xysp*] BITA [*oprx16,xysp*]	(A) • (M) Logical And A with Memory	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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–
BITB *#opr8i* BITB *opr8a* BITB *opr16a* BITB *oprx0_xysp* BITB *oprx9,xysp* BITB *oprx16,xysp* BITB [D,*xysp*] BITB [*oprx16,xysp*]	(B) • (M) Logical And B with Memory	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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–

Notes: 1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
BLE *rel8*	Branch if Less Than or Equal (if Z ✣ (N ⊕ V) = 1) (signed)	REL	2F rr	PPP/P[1]	–	–	–	–	–	–	–	–
BLO *rel8*	Branch if Lower (if C = 1) (unsigned) same function as BCS	REL	25 rr	PPP/P[1]	–	–	–	–	–	–	–	–
BLS *rel8*	Branch if Lower or Same (if C ✣ Z = 1) (unsigned)	REL	23 rr	PPP/P[1]	–	–	–	–	–	–	–	–
BLT *rel8*	Branch if Less Than (if N ⊕ V = 1) (signed)	REL	2D rr	PPP/P[1]	–	–	–	–	–	–	–	–
BMI *rel8*	Branch if Minus (if N = 1)	REL	2B rr	PPP/P[1]	–	–	–	–	–	–	–	–
BNE *rel8*	Branch if Not Equal (if Z = 0)	REL	26 rr	PPP/P[1]	–	–	–	–	–	–	–	–
BPL *rel8*	Branch if Plus (if N = 0)	REL	2A rr	PPP/P[1]	–	–	–	–	–	–	–	–
BRA *rel8*	Branch Always (if 1 = 1)	REL	20 rr	PPP	–	–	–	–	–	–	–	–
BRCLR *opr8a, msk8, rel8* BRCLR *opr16a, msk8, rel8* BRCLR *oprx0_xysp, msk8, rel8* BRCLR *oprx9,xysp, msk8, rel8* BRCLR *oprx16,xysp, msk8, rel8*	Branch if (M) • (mm) = 0 (if All Selected Bit(s) Clear)	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 rffPPP frPffPPP	–	–	–	–	–	–	–	–
BRN *rel8*	Branch Never (if 1 = 0)	REL	21 rr	P	–	–	–	–	–	–	–	–
BRSET *opr8, msk8, rel8* BRSET *opr16a, msk8, rel8* BRSET *oprx0_xysp, msk8, rel8* BRSET *oprx9,xysp, msk8, rel8* BRSET *oprx16,xysp, msk8, rel8*	Branch if ($\overline{M}$) • (mm) = 0 (if All Selected Bit(s) Set)	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 rffPPP frPffPPP	–	–	–	–	–	–	–	–
BSET *opr8, msk8* BSET *opr16a, msk8* BSET *oprx0_xysp, msk8* BSET *oprx9,xysp, msk8* BSET *oprx16,xysp, msk8*	(M) ✣ (mm) ⇒ M Set Bit(s) in Memory	DIR EXT IDX IDX1 IDX2	4C dd mm 1C hh ll mm 0C xb mm 0C xb ff mm 0C xb ee ff mm	rPOw rPPw rPOw rPwP frPwOP	–	–	–	–	Δ	Δ	0	–
BSR *rel8*	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Subroutine address ⇒ PC Branch to Subroutine	REL	07 rr	PPPS	–	–	–	–	–	–	–	–
BVC *rel8*	Branch if Overflow Bit Clear (if V = 0)	REL	28 rr	PPP/P[1]	–	–	–	–	–	–	–	–
BVS *rel8*	Branch if Overflow Bit Set (if V = 1)	REL	29 rr	PPP/P[1]	–	–	–	–	–	–	–	–
CALL *opr16a*, *page* CALL *oprx0_xysp*, *page* CALL *oprx9,xysp*, *page* CALL *oprx16,xysp*, *page* CALL [D,*xysp*] CALL [*oprx16, xysp*]	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$ (SP) – 1 ⇒ SP; (PPG) ⇒ $M_{(SP)}$; pg ⇒ PPAGE register; Program address ⇒ PC Call subroutine in extended memory (Program may be located on another expansion memory page.) Indirect modes get program address and new pg value based on pointer.	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	gnfSsPPP gnfSsPPP gnfSsPPP fgnfSsPPP fIignSsPPP fIignSsPPP	–	–	–	–	–	–	–	–
CBA	(A) – (B) Compare 8-Bit Accumulators	INH	18 17	OO	–	–	–	–	Δ	Δ	Δ	Δ
CLC	0 ⇒ C *Translates to* ANDCC #$FE	IMM	10 FE	P	–	–	–	–	–	–	–	0
CLI	0 ⇒ I *Translates to* ANDCC #$EF (enables I-bit interrupts)	IMM	10 EF	P	–	–	–	0	–	–	–	–

Notes
1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
CLR *opr16a* CLR *oprx0_xysp* CLR *oprx9,xysp* CLR *oprx16,xysp* CLR [D,*xysp*] CLR [*oprx16,xysp*] CLRA CLRB	0 ⇒ M Clear Memory Location 0 ⇒ A Clear Accumulator A 0 ⇒ B Clear Accumulator 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	wOP Pw PwO PwP PIfPw PIPPw O O	–	–	–	–	0	1	0	0
CLV	0 ⇒ V *Translates to* ANDCC #$FD	IMM	10 FD	P	–	–	–	–	–	–	0	–
CMPA *#opr8i* CMPA *opr8a* CMPA *opr16a* CMPA *oprx0_xysp* CMPA *oprx9,xysp* CMPA *oprx16,xysp* CMPA [D,*xysp*] CMPA [*oprx16,xysp*]	(A) – (M) Compare Accumulator A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	81 ii 91 dd B1 hh ll A1 xb A1 xb ff A1 xb ee ff A1 xb A1 xb ee ff	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	Δ	Δ
CMPB *#opr8i* CMPB *opr8a* CMPB *opr16a* CMPB *oprx0_xysp* CMPB *oprx9,xysp* CMPB *oprx16,xysp* CMPB [D,*xysp*] CMPB [*oprx16,xysp*]	(B) – (M) Compare Accumulator B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C1 ii D1 dd F1 hh ll E1 xb E1 xb ff E1 xb ee ff E1 xb E1 xb ee ff	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	Δ	Δ
COM *opr16a* COM *oprx0_xysp* COM *oprx9,xysp* COM *oprx16,xysp* COM [D,*xysp*] COM [*oprx16,xysp*] COMA COMB	($\overline{M}$) ⇒ M *equivalent to* $FF – (M) ⇒ M 1's Complement Memory Location ($\overline{A}$) ⇒ A Complement Accumulator A ($\overline{B}$) ⇒ B Complement Accumulator 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	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	–	–	–	–	Δ	Δ	0	1
CPD *#opr16i* CPD *opr8a* CPD *opr16a* CPD *oprx0_xysp* CPD *oprx9,xysp* CPD *oprx16,xysp* CPD [D,*xysp*] CPD [*oprx16,xysp*]	(A:B) – (M:M+1) Compare D to Memory (16-Bit)	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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	Δ	Δ
CPS *#opr16i* CPS *opr8a* CPS *opr16a* CPS *oprx0_xysp* CPS *oprx9,xysp* CPS *oprx16,xysp* CPS [D,*xysp*] CPS [*oprx16,xysp*]	(SP) – (M:M+1) Compare SP to Memory (16-Bit)	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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	Δ	Δ
CPX *#opr16i* CPX *opr8a* CPX *opr16a* CPX *oprx0_xysp* CPX *oprx9,xysp* CPX *oprx16,xysp* CPX [D,*xysp*] CPX [*oprx16,xysp*]	(X) – (M:M+1) Compare X to Memory (16-Bit)	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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	Δ	Δ
CPY *#opr16i* CPY *opr8a* CPY *opr16a* CPY *oprx0_xysp* CPY *oprx9,xysp* CPY *oprx16,xysp* CPY [D,*xysp*] CPY [*oprx16,xysp*]	(Y) – (M:M+1) Compare Y to Memory (16-Bit)	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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	Δ	Δ

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
DAA	Adjust Sum to BCD Decimal Adjust Accumulator A	INH	18 07	OfO	–	–	–	–	Δ	Δ	?	Δ
DBEQ *abdxys, rel9*	(cntr) – 1⇒ cntr if (cntr) = 0, then Branch else Continue to next instruction Decrement Counter and Branch if = 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	PPP	–	–	–	–	–	–	–	–
DBNE *abdxys, rel9*	(cntr) – 1 ⇒ cntr If (cntr) not = 0, then Branch; else Continue to next instruction Decrement Counter and Branch if ≠ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	PPP	–	–	–	–	–	–	–	–
DEC *opr16a* DEC *oprx0_xysp* DEC *oprx9,xysp* DEC *oprx16,xysp* DEC [D,*xysp*] DEC [*oprx16,xysp*] DECA DECB	(M) – $01 ⇒ M Decrement Memory Location (A) – $01 ⇒ A Decrement A (B) – $01 ⇒ B Decrement 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	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	–	–	–	–	Δ	Δ	Δ	–
DES	(SP) – $0001 ⇒ SP *Translates to* LEAS –1,SP	IDX	1B 9F	PP[1]	–	–	–	–	–	–	–	–
DEX	(X) – $0001 ⇒ X Decrement Index Register X	INH	09	O	–	–	–	–	–	Δ	–	–
DEY	(Y) – $0001 ⇒ Y Decrement Index Register Y	INH	03	O	–	–	–	–	–	Δ	–	–
EDIV	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D 32 × 16 Bit ⇒ 16 Bit Divide (unsigned)	INH	11	fffffffffO	–	–	–	–	Δ	Δ	Δ	Δ
EDIVS	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D 32 × 16 Bit ⇒ 16 Bit Divide (signed)	INH	18 14	OfffffffffO	–	–	–	–	Δ	Δ	Δ	Δ
EMACS *opr16a* [2]	$(M_{(X)}:M_{(X+1)}) \times (M_{(Y)}:M_{(Y+1)}) + (M{\sim}M+3) \Rightarrow M{\sim}M+3$ 16 × 16 Bit ⇒ 32 Bit Multiply and Accumulate (signed)	Special	18 12 hh ll	ORROfffRRfWWP	–	–	–	–	Δ	Δ	Δ	Δ
EMAXD *oprx0_xysp* EMAXD *oprx9,xysp* EMAXD *oprx16,xysp* EMAXD [D,*xysp*] EMAXD [*oprx16,xysp*]	MAX((D), (M:M+1)) ⇒ D MAX of 2 Unsigned 16-Bit Values N, Z, V and C status 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	ORfP ORPO OfRPP OfIfRfP OfIPRfP	–	–	–	–	Δ	Δ	Δ	Δ
EMAXM *oprx0_xysp* EMAXM *oprx9,xysp* EMAXM *oprx16,xysp* EMAXM [D,*xysp*] EMAXM [*oprx16,xysp*]	MAX((D), (M:M+1)) ⇒ M:M+1 MAX of 2 Unsigned 16-Bit Values N, Z, V and C status 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_xysp* EMIND *oprx9,xysp* EMIND *oprx16,xysp* EMIND [D,*xysp*] EMIND [*oprx16,xysp*]	MIN((D), (M:M+1)) ⇒ D MIN of 2 Unsigned 16-Bit Values N, Z, V and C status 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	ORfP ORPO OfRPP OfIfRfP OfIPRfP	–	–	–	–	Δ	Δ	Δ	Δ
EMINM *oprx0_xysp* EMINM *oprx9,xysp* EMINM *oprx16,xysp* EMINM [D,*xysp*] EMINM [*oprx16,xysp*]	MIN((D), (M:M+1)) ⇒ M:M+1 MIN of 2 Unsigned 16-Bit Values N, Z, V and C status 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	(D) × (Y) ⇒ Y:D 16 × 16 Bit Multiply (unsigned)	INH	13	ffO	–	–	–	–	Δ	Δ	–	Δ

Notes
1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.
2. *opr16a* is an extended address specification. Both X and Y point to source operands.

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
EMULS	(D) × (Y) ⇒ Y:D 16 × 16 Bit Multiply (signed)	INH	18 13	Ofo	–	–	–	–	Δ	Δ	–	Δ
EORA *#opr8i* EORA *opr8a* EORA *opr16a* EORA *oprx0_xysp* EORA *oprx9,xysp* EORA *oprx16,xysp* EORA [D,*xysp*] EORA [*oprx16,xysp*]	(A) ⊕ (M) ⇒ A Exclusive-OR A with Memory	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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–
EORB *#opr8i* EORB *opr8a* EORB *opr16a* EORB *oprx0_xysp* EORB *oprx9,xysp* EORB *oprx16,xysp* EORB [D,*xysp*] EORB [*oprx16,xysp*]	(B) ⊕ (M) ⇒ B Exclusive-OR B with Memory	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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–
ETBL *oprx0_xysp*	(M:M+1)+ [(B)×((M+2:M+3) – (M:M+1))] ⇒ D 16-Bit Table Lookup and Interpolate Initialize B, and index before ETBL. <ea> points at first table entry (M:M+1) and B is fractional part of lookup value (no indirect addr. modes or extensions allowed)	IDX	18 3F xb	ORRfffffP	–	–	–	–	Δ	Δ	–	?
EXG *abcdxys,abcdxys*	(r1) ⇔ (r2) (if r1 and r2 same size) *or* $00:(r1) ⇒ r2 (if r1=8-bit; r2=16-bit) *or* ($r1_{low}$) ⇔ (r2) (if r1=16-bit; r2=8-bit) r1 and r2 may be A, B, CCR, D, X, Y, or SP	INH	B7 eb	P	–	–	–	–	–	–	–	–
FDIV	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 × 16 Bit Fractional Divide	INH	18 11	Offffffffffo	–	–	–	–	–	Δ	Δ	Δ
IBEQ *abdxys, rel9*	(cntr) + 1⇒ cntr If (cntr) = 0, then Branch else Continue to next instruction Increment Counter and Branch if = 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	PPP	–	–	–	–	–	–	–	–
IBNE *abdxys, rel9*	(cntr) + 1⇒ cntr if (cntr) not = 0, then Branch; else Continue to next instruction Increment Counter and Branch if ≠ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	PPP	–	–	–	–	–	–	–	–
IDIV	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 × 16 Bit Integer Divide (unsigned)	INH	18 10	Offffffffffo	–	–	–	–	–	Δ	0	Δ
IDIVS	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 × 16 Bit Integer Divide (signed)	INH	18 15	Offffffffffo	–	–	–	–	Δ	Δ	Δ	Δ
INC *opr16a* INC *oprx0_xysp* INC *oprx9,xysp* INC *oprx16,xysp* INC [D,*xysp*] INC [*oprx16,xysp*] INCA INCB	(M) + $01 ⇒ M Increment Memory Byte (A) + $01 ⇒ A Increment Acc. A (B) + $01 ⇒ B Increment Acc. 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	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	–	–	–	–	Δ	Δ	Δ	–
INS	(SP) + $0001 ⇒ SP *Translates to* LEAS 1,SP	IDX	1B 81	PP[1]	–	–	–	–	–	–	–	–
INX	(X) + $0001 ⇒ X Increment Index Register X	INH	08	O	–	–	–	–	–	Δ	–	–
INY	(Y) + $0001 ⇒ Y Increment Index Register Y	INH	02	O	–	–	–	–	–	Δ	–	–

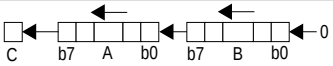
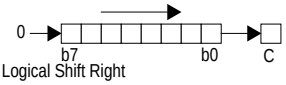
Note 1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
JMP *opr16a* JMP *oprx0_xysp* JMP *oprx9,xysp* JMP *oprx16,xysp* JMP [D,*xysp*] JMP [*oprx16,xysp*]	Subroutine address ⇒ PC Jump	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_xysp* JSR *oprx9,xysp* JSR *oprx16,xysp* JSR [D,*xysp*] JSR [*oprx16,xysp*]	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$; Subroutine address ⇒ PC Jump to Subroutine	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	PPPS PPPS PPPS PPPS fPPPS fIfPPPS fIfPPPS	–	–	–	–	–	–	–	–
LBCC *rel16*	Long Branch if Carry Clear (if C = 0)	REL	18 24 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBCS *rel16*	Long Branch if Carry Set (if C = 1)	REL	18 25 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBEQ *rel16*	Long Branch if Equal (if Z = 1)	REL	18 27 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBGE *rel16*	Long Branch Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	18 2C qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBGT *rel16*	Long Branch if Greater Than (if Z ✣ (N ⊕ V) = 0) (signed)	REL	18 2E qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBHI *rel16*	Long Branch if Higher (if C ✣ Z = 0) (unsigned)	REL	18 22 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBHS *rel16*	Long Branch if Higher or Same (if C = 0) (unsigned) same function as LBCC	REL	18 24 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBLE *rel16*	Long Branch if Less Than or Equal (if Z ✣ (N ⊕ V) = 1) (signed)	REL	18 2F qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBLO *rel16*	Long Branch if Lower (if C = 1) (unsigned) *same function as LBCS*	REL	18 25 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBLS *rel16*	Long Branch if Lower or Same (if C ✣ Z = 1) (unsigned)	REL	18 23 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBLT *rel16*	Long Branch if Less Than (if N ⊕ V = 1) (signed)	REL	18 2D qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBMI *rel16*	Long Branch if Minus (if N = 1)	REL	18 2B qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBNE *rel16*	Long Branch if Not Equal (if Z = 0)	REL	18 26 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBPL *rel16*	Long Branch if Plus (if N = 0)	REL	18 2A qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBRA *rel16*	Long Branch Always (if 1=1)	REL	18 20 qq rr	OPPP	–	–	–	–	–	–	–	–
LBRN *rel16*	Long Branch Never (if 1 = 0)	REL	18 21 qq rr	OPO	–	–	–	–	–	–	–	–
LBVC *rel16*	Long Branch if Overflow Bit Clear (if V=0)	REL	18 28 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LBVS *rel16*	Long Branch if Overflow Bit Set (if V = 1)	REL	18 29 qq rr	OPPP/OPO[1]	–	–	–	–	–	–	–	–
LDAA *#opr8i* LDAA *opr8a* LDAA *opr16a* LDAA *oprx0_xysp* LDAA *oprx9,xysp* LDAA *oprx16,xysp* LDAA [D,*xysp*] LDAA [*oprx16,xysp*]	(M) ⇒ A Load Accumulator 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–
LDAB *#opr8i* LDAB *opr8a* LDAB *opr16a* LDAB *oprx0_xysp* LDAB *oprx9,xysp* LDAB *oprx16,xysp* LDAB [D,*xysp*] LDAB [*oprx16,xysp*]	(M) ⇒ B Load Accumulator 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–

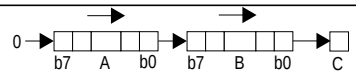
Note 1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
LDD *#opr16i* LDD *opr8a* LDD *opr16a* LDD *oprx0_xysp* LDD *oprx9,xysp* LDD *oprx16,xysp* LDD [D,*xysp*] LDD [*oprx16,xysp*]	(M:M+1) ⇒ A:B Load Double Accumulator D (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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	0	–
LDS *#opr16i* LDS *opr8a* LDS *opr16a* LDS *oprx0_xysp* LDS *oprx9,xysp* LDS *oprx16,xysp* LDS [D,*xysp*] LDS [*oprx16,xysp*]	(M:M+1) ⇒ SP Load Stack Pointer	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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	0	–
LDX *#opr16i* LDX *opr8a* LDX *opr16a* LDX *oprx0_xysp* LDX *oprx9,xysp* LDX *oprx16,xysp* LDX [D,*xysp*] LDX [*oprx16,xysp*]	(M:M+1) ⇒ X Load Index Register 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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	0	–
LDY *#opr16i* LDY *opr8a* LDY *opr16a* LDY *oprx0_xysp* LDY *oprx9,xysp* LDY *oprx16,xysp* LDY [D,*xysp*] LDY [*oprx16,xysp*]	(M:M+1) ⇒ Y Load Index Register 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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	0	–
LEAS *oprx0_xysp* LEAS *oprx9,xysp* LEAS *oprx16,xysp*	Effective Address ⇒ SP Load Effective Address into SP	IDX IDX1 IDX2	1B xb 1B xb ff 1B xb ee ff	PP[1] PO PP	–	–	–	–	–	–	–	–
LEAX *oprx0_xysp* LEAX *oprx9,xysp* LEAX *oprx16,xysp*	Effective Address ⇒ X Load Effective Address into X	IDX IDX1 IDX2	1A xb 1A xb ff 1A xb ee ff	PP[1] PO PP	–	–	–	–	–	–	–	–
LEAY *oprx0_xysp* LEAY *oprx9,xysp* LEAY *oprx16,xysp*	Effective Address ⇒ Y Load Effective Address into Y	IDX IDX1 IDX2	19 xb 19 xb ff 19 xb ee ff	PP[1] PO PP	–	–	–	–	–	–	–	–
LSL *opr16a* LSL *oprx0_xysp* LSL *oprx9,xysp* LSL *oprx16,xysp* LSL [D,*xysp*] LSL [*oprx16,xysp*] LSLA LSLB	 Logical Shift Left same function as ASL Logical Shift Accumulator A to Left Logical Shift Accumulator B to Left	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
LSLD	 Logical Shift Left D Accumulator same function as ASLD	INH	59	O	–	–	–	–	Δ	Δ	Δ	Δ
LSR *opr16a* LSR *oprx0_xysp* LSR *oprx9,xysp* LSR *oprx16,xysp* LSR [D,*xysp*] LSR [*oprx16,xysp*] LSRA LSRB	 Logical Shift Right Logical Shift Accumulator A to Right Logical Shift Accumulator B to Right	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	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	–	–	–	–	0	Δ	Δ	Δ

Note 1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

Instruction Set Summary (Continued)

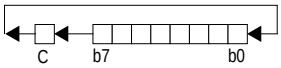
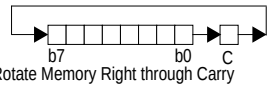
Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
LSRD	 Logical Shift Right D Accumulator	INH	49	o	–	–	–	–	0	Δ	Δ	Δ
MAXA *oprx0_xysp* MAXA *oprx9,xysp* MAXA *oprx16,xysp* MAXA [D,*xysp*] MAXA [*oprx16,xysp*]	MAX((A), (M)) ⇒ A MAX of 2 Unsigned 8-Bit Values N, Z, V and C status 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	OrfP OrPO OfrPP OfIfrfP OfIPrfP	–	–	–	–	Δ	Δ	Δ	Δ
MAXM *oprx0_xysp* MAXM *oprx9,xysp* MAXM *oprx16,xysp* MAXM [D,*xysp*] MAXM [*oprx16,xysp*]	MAX((A), (M)) ⇒ M MAX of 2 Unsigned 8-Bit Values N, Z, V and C status 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	μ (grade) ⇒ $M_{(Y)}$; (X) + 4 ⇒ X; (Y) + 1 ⇒ Y; A unchanged if (A) < P1 or (A) > P2 then μ = 0, else μ = MIN[((A) – P1)×S1, (P2 – (A))×S2, $FF] where: A = current crisp input value; X points at 4-byte data structure that describes a trapezoidal membership function (P1, P2, S1, S2); Y points at fuzzy input (RAM location). See *CPU12 Reference Manual* for special cases.	Special	01	RRfOw	–	–	?	–	?	?	?	?
MINA *oprx0_xysp* MINA *oprx9,xysp* MINA *oprx16,xysp* MINA [D,*xysp*] MINA [*oprx16,xysp*]	MIN((A), (M)) ⇒ A MIN of 2 Unsigned 8-Bit Values N, Z, V and C status 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	OrfP OrPO OfrPP OfIfrfP OfIPrfP	–	–	–	–	Δ	Δ	Δ	Δ
MINM *oprx0_xysp* MINM *oprx9,xysp* MINM *oprx16,xysp* MINM [D,*xysp*] MINM [*oprx16,xysp*]	MIN((A), (M)) ⇒ M MIN of 2 Unsigned 8-Bit Values N, Z, V and C status 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
MOVB *#opr8, opr16a*[1] MOVB *#opr8i, oprx0_xysp*[1] MOVB *opr16a, opr16a*[1] MOVB *opr16a, oprx0_xysp*[1] MOVB *oprx0_xysp, opr16a*[1] MOVB *oprx0_xysp, oprx0_xysp*[1]	$(M_1) \Rightarrow M_2$ Memory to Memory Byte-Move (8-Bit)	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*[1] MOVW *#opr16i, oprx0_xysp*[1] MOVW *opr16a, opr16a*[1] MOVW *opr16a, oprx0_xysp*[1] MOVW *oprx0_xysp, opr16a*[1] MOVW *oprx0_xysp, oprx0_xysp*[1]	$(M:M+1_1) \Rightarrow M:M+1_2$ Memory to Memory Word-Move (16-Bit)	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	(A) × (B) ⇒ A:B 8 × 8 Unsigned Multiply	INH	12	ffO	–	–	–	–	–	–	–	Δ
NEG *opr16a* NEG *oprx0_xysp* NEG *oprx9,xysp* NEG *oprx16,xysp* NEG [D,*xysp*] NEG [*oprx16,xysp*] NEGA NEGB	0 – (M) ⇒ M *or* $(\overline{M}) + 1 \Rightarrow M$ Two's Complement Negate 0 – (A) ⇒ A *equivalent to* $(\overline{A}) + 1 \Rightarrow A$ Negate Accumulator A 0 – (B) ⇒ B *equivalent to* $(\overline{B}) + 1 \Rightarrow B$ Negate Accumulator 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	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	–	–	–	–	Δ	Δ	Δ	Δ
NOP	No Operation	INH	A7	O	–	–	–	–	–	–	–	–

Note 1. The first operand in the source code statement specifies the source for the move.

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
ORAA *#opr8i* ORAA *opr8a* ORAA *opr16a* ORAA *oprx0_xysp* ORAA *oprx9,xysp* ORAA *oprx16,xysp* ORAA [D,*xysp*] ORAA [*oprx16,xysp*]	(A) ✣ (M) ⇒ A Logical OR A with Memory	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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–
ORAB *#opr8i* ORAB *opr8a* ORAB *opr16a* ORAB *oprx0_xysp* ORAB *oprx9,xysp* ORAB *oprx16,xysp* ORAB [D,*xysp*] ORAB [*oprx16,xysp*]	(B) ✣ (M) ⇒ B Logical OR B with Memory	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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	0	–
ORCC *#opr8i*	(CCR) ✣ M ⇒ CCR Logical OR CCR with Memory	IMM	14 ii	P	⇑	–	⇑	⇑	⇑	⇑	⇑	⇑
PSHA	(SP) – 1 ⇒ SP; (A) ⇒ $M_{(SP)}$ Push Accumulator A onto Stack	INH	36	Os	–	–	–	–	–	–	–	–
PSHB	(SP) – 1 ⇒ SP; (B) ⇒ $M_{(SP)}$ Push Accumulator B onto Stack	INH	37	Os	–	–	–	–	–	–	–	–
PSHC	(SP) – 1 ⇒ SP; (CCR) ⇒ $M_{(SP)}$ Push CCR onto Stack	INH	39	Os	–	–	–	–	–	–	–	–
PSHD	(SP) – 2 ⇒ SP; (A:B) ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Push D Accumulator onto Stack	INH	3B	OS	–	–	–	–	–	–	–	–
PSHX	(SP) – 2 ⇒ SP; (X_H:X_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Push Index Register X onto Stack	INH	34	OS	–	–	–	–	–	–	–	–
PSHY	(SP) – 2 ⇒ SP; (Y_H:Y_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Push Index Register Y onto Stack	INH	35	OS	–	–	–	–	–	–	–	–
PULA	($M_{(SP)}$) ⇒ A; (SP) + 1 ⇒ SP Pull Accumulator A from Stack	INH	32	ufO	–	–	–	–	–	–	–	–
PULB	($M_{(SP)}$) ⇒ B; (SP) + 1 ⇒ SP Pull Accumulator B from Stack	INH	33	ufO	–	–	–	–	–	–	–	–
PULC	($M_{(SP)}$) ⇒ CCR; (SP) + 1 ⇒ SP Pull CCR from Stack	INH	38	ufO	Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ
PULD	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ A:B; (SP) + 2 ⇒ SP Pull D from Stack	INH	3A	UfO	–	–	–	–	–	–	–	–
PULX	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ X_H:X_L; (SP) + 2 ⇒ SP Pull Index Register X from Stack	INH	30	UfO	–	–	–	–	–	–	–	–
PULY	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ Y_H:Y_L; (SP) + 2 ⇒ SP Pull Index Register Y from Stack	INH	31	UfO	–	–	–	–	–	–	–	–

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
REV (add if interrupted)	MIN-MAX rule evaluation Find smallest rule input (MIN). Store to rule outputs unless fuzzy output is already larger (MAX). For rule weights see REVW. Each rule input is an 8-bit offset from the base address in Y. Each rule output is an 8-bit offset from the base address in Y. $FE separates rule inputs from rule outputs. $FF terminates the rule list. REV may be interrupted.	Special	18 3A	Orf(ttx)O[1] ff + Orf	–	–	?	–	?	?	Δ	?
REVW (add 2 at end of ins if wts) (add if interrupted)	MIN-MAX rule evaluation Find smallest rule input (MIN), Store to rule outputs unless fuzzy output is already larger (MAX). Rule weights supported, optional. Each rule input is the 16-bit address of a fuzzy input. Each rule output is the 16-bit address of a fuzzy output. The value $FFFE separates rule inputs from rule outputs. $FFFF terminates the rule list. REVW may be interrupted.	Special	18 3B	ORf(tTx)O[2] (rffRf)[2] fff + ORft	–	–	?	–	?	?	Δ	!
ROL *opr16a* ROL *oprx0_xysp* ROL *oprx9,xysp* ROL *oprx16,xysp* ROL [D,*xysp*] ROL [*oprx16,xysp*] ROLA ROLB	 Rotate Memory Left through Carry Rotate A Left through Carry Rotate B Left through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	75 hh ll 65 xb 65 xb ff 65 xb ee ff 65 xb 65 xb ee ff 45 55	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	–	–	–	–	Δ	Δ	Δ	Δ
ROR *opr16a* ROR *oprx0_xysp* ROR *oprx9,xysp* ROR *oprx16,xysp* ROR [D,*xysp*] ROR [*oprx16,xysp*] RORA RORB	 Rotate Memory Right through Carry Rotate A Right through Carry Rotate B Right through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	76 hh ll 66 xb 66 xb ff 66 xb ee ff 66 xb 66 xb ee ff 46 56	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	–	–	–	–	Δ	Δ	Δ	Δ
RTC	($M_{(SP)}$) ⇒ PPAGE; (SP) + 1 ⇒ SP; ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ PC_H:PC_L; (SP) + 2 ⇒ SP Return from Call	INH	0A	uUnPPP	–	–	–	–	–	–	–	–
RTI (if interrupt pending)	($M_{(SP)}$) ⇒ CCR; (SP) + 1 ⇒ SP ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ B:A; (SP) + 2 ⇒ SP ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ X_H:X_L; (SP) + 4 ⇒ SP ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ PC_H:PC_L; (SP) – 2 ⇒ SP ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ Y_H:Y_L; (SP) + 4 ⇒ SP Return from Interrupt	INH	0B	uUUUUPPP uUUUUVfPPP	Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ
RTS	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ PC_H:PC_L; (SP) + 2 ⇒ SP Return from Subroutine	INH	3D	UfPPP	–	–	–	–	–	–	–	–
SBA	(A) – (B) ⇒ A Subtract B from A	INH	18 16	OO	–	–	–	–	Δ	Δ	Δ	Δ

Notes:
1. The 3-cycle loop in parentheses is executed once for each element in the rule list. When an interrupt occurs, there is a 2-cycle exit sequence, a 4-cycle re-entry sequence, then execution resumes with a prefetch of the last antecedent or consequent being processed at the time of the interrupt.
2. The 3-cycle loop in parentheses expands to 5 cycles for separators when weighting is enabled. The loop is executed once for each element in the rule list. When an interrupt occurs, there is a 2-cycle exit sequence, a 4-cycle re-entry sequence, then execution resumes with a prefetch of the last antecedent or consequent being processed at the time of the interrupt.

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
SBCA *#opr8i* SBCA *opr8a* SBCA *opr16a* SBCA *oprx0_xysp* SBCA *oprx9,xysp* SBCA *oprx16,xysp* SBCA [D,*xysp*] SBCA [*oprx16,xysp*]	(A) – (M) – C ⇒ A Subtract with Borrow from 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	Δ	Δ
SBCB *#opr8i* SBCB *opr8a* SBCB *opr16a* SBCB *oprx0_xysp* SBCB *oprx9,xysp* SBCB *oprx16,xysp* SBCB [D,*xysp*] SBCB [*oprx16,xysp*]	(B) – (M) – C ⇒ B Subtract with Borrow from 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	Δ	Δ
SEC	1 ⇒ C *Translates to* ORCC #$01	IMM	14 01	P	–	–	–	–	–	–	–	1
SEI	1 ⇒ I; (inhibit I interrupts) *Translates to* ORCC #$10	IMM	14 10	P	–	–	–	1	–	–	–	–
SEV	1 ⇒ V *Translates to* ORCC #$02	IMM	14 02	P	–	–	–	–	–	–	1	–
SEX *abc,dxys*	$00:(r1) ⇒ r2 if r1, bit 7 is 0 *or* $FF:(r1) ⇒ r2 if r1, bit 7 is 1 Sign Extend 8-bit r1 to 16-bit r2 r1 may be A, B, or CCR r2 may be D, X, Y, or SP *Alternate mnemonic for* TFR r1, r2	INH	B7 eb	P	–	–	–	–	–	–	–	–
STAA *opr8a* STAA *opr16a* STAA *oprx0_xysp* STAA *oprx9,xysp* STAA *oprx16,xysp* STAA [D,*xysp*] STAA [*oprx16,xysp*]	(A) ⇒ M Store Accumulator A to Memory	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 wOP Pw PwO PwP PIfPw PIPPw	–	–	–	–	Δ	Δ	0	–
STAB *opr8a* STAB *opr16a* STAB *oprx0_xysp* STAB *oprx9,xysp* STAB *oprx16,xysp* STAB [D,*xysp*] STAB [*oprx16,xysp*]	(B) ⇒ M Store Accumulator B to Memory	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 wOP Pw PwO PwP PIfPw PIPPw	–	–	–	–	Δ	Δ	0	–
STD *opr8a* STD *opr16a* STD *oprx0_xysp* STD *oprx9,xysp* STD *oprx16,xysp* STD [D,*xysp*] STD [*oprx16,xysp*]	(A) ⇒ M, (B) ⇒ M+1 Store Double Accumulator	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 WOP PW PWO PWP PIfPW PIPPW	–	–	–	–	Δ	Δ	0	–
STOP (entering STOP) (exiting STOP) (continue) (if STOP disabled)	(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 If S control bit = 1, the STOP instruction is disabled and acts like a two-cycle NOP. Registers stacked to allow quicker recovery by interrupt.	INH	18 3E	OOSSSfSs fVfPPP fO OO	–	–	–	–	–	–	–	–

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
STS *opr8a* STS *opr16a* STS *oprx0_xysp* STS *oprx9,xysp* STS *oprx16,xysp* STS [D,*xysp*] STS [*oprx16,xysp*]	(SP_H:SP_L) ⇒ M:M+1 Store Stack Pointer	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 WOP PW PWO PWP PIfPW PIPPW	–	–	–	–	Δ	Δ	0	–
STX *opr8a* STX *opr16a* STX *oprx0_xysp* STX *oprx9,xysp* STX *oprx16,xysp* STX [D,*xysp*] STX [*oprx16,xysp*]	(X_H:X_L) ⇒ M:M+1 Store Index Register X	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 WOP PW PWO PWP PIfPW PIPPW	–	–	–	–	Δ	Δ	0	–
STY *opr8a* STY *opr16a* STY *oprx0_xysp* STY *oprx9,xysp* STY *oprx16,xysp* STY [D,*xysp*] STY [*oprx16,xysp*]	(Y_H:Y_L) ⇒ M:M+1 Store Index Register Y	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 WOP PW PWO PWP PIfPW PIPPW	–	–	–	–	Δ	Δ	0	–
SUBA #*opr8i* SUBA *opr8a* SUBA *opr16a* SUBA *oprx0_xysp* SUBA *oprx9,xysp* SUBA *oprx16,xysp* SUBA [D,*xysp*] SUBA [*oprx16,xysp*]	(A) – (M) ⇒ A Subtract Memory from Accumulator 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	Δ	Δ
SUBB #*opr8i* SUBB *opr8a* SUBB *opr16a* SUBB *oprx0_xysp* SUBB *oprx9,xysp* SUBB *oprx16,xysp* SUBB [D,*xysp*] SUBB [*oprx16,xysp*]	(B) – (M) ⇒ B Subtract Memory from Accumulator 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 rfP rOP rfP rPO frPP fIfrfP fIPrfP	–	–	–	–	Δ	Δ	Δ	Δ
SUBD #*opr16i* SUBD *opr8a* SUBD *opr16a* SUBD *oprx0_xysp* SUBD *oprx9,xysp* SUBD *oprx16,xysp* SUBD [D,*xysp*] SUBD [*oprx16,xysp*]	(D) – (M:M+1) ⇒ D Subtract Memory from D (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	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	–	–	–	–	Δ	Δ	Δ	Δ
SWI	(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; (SWI Vector) ⇒ PC Software Interrupt	INH	3F	VSPSSPSsP[1]	–	–	–	1	–	–	–	–
TAB	(A) ⇒ B Transfer A to B	INH	18 0E	OO	–	–	–	–	Δ	Δ	0	–
TAP	(A) ⇒ CCR *Translates to* TFR A , CCR	INH	B7 02	P	Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ
TBA	(B) ⇒ A Transfer B to A	INH	18 0F	OO	–	–	–	–	Δ	Δ	0	–

Note 1. The CPU also uses the SWI processing sequence for hardware interrupts and unimplemented opcode traps. A variation of the sequence (VfPPP) is used for resets.

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
TBEQ *abdxys,rel9*	If (cntr) = 0, then Branch; else Continue to next instruction Test Counter and Branch if Zero (cntr = A, B, D, X,Y, or SP)	REL (9-bit)	`04 lb rr`	`PPP`	–	–	–	–	–	–	–	–
TBL *oprx0_xysp*	(M) + [(B) × ((M+1) – (M))] ⇒ A 8-Bit Table Lookup and Interpolate Initialize B, and index before TBL. <ea> points at first 8-bit table entry (M) and B is fractional part of lookup value. (no indirect addressing modes or extensions allowed)	IDX	`18 3D xb`	`Orrrrrfp`	–	–	–	–	Δ	Δ	–	?
TBNE *abdxys,rel9*	If (cntr) not = 0, then Branch; else Continue to next instruction Test Counter and Branch if Not Zero (cntr = A, B, D, X,Y, or SP)	REL (9-bit)	`04 lb rr`	`PPP`	–	–	–	–	–	–	–	–
TFR *abcdxys,abcdxys*	(r1) ⇒ r2 *or* $00:(r1) ⇒ r2 *or* (r1[7:0]) ⇒ r2 Transfer Register to Register r1 and r2 may be A, B, CCR, D, X, Y, or SP	INH	`B7 eb`	`P`	– or Δ	– ⇓	– Δ	– Δ	– Δ	– Δ	– Δ	– Δ
TPA	(CCR) ⇒ A *Translates to* TFR CCR ,A	INH	`B7 20`	`P`	–	–	–	–	–	–	–	–
TRAP *trapnum*	(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 Unimplemented opcode trap	INH	`18 tn` `tn = $30–$39` `or` `$40–$FF`	`OfVSPSSPSsP`	–	–	–	1	–	–	–	–
TST *opr16a* TST *oprx0_xysp* TST *oprx9,xysp* TST *oprx16,xysp* TST [D,*xysp*] TST [*oprx16,xysp*] TSTA TSTB	(M) – 0 Test Memory for Zero or Minus (A) – 0 Test A for Zero or Minus (B) – 0 Test B for Zero or Minus	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`	`rOP` `rfP` `rPO` `frPP` `fIfrfP` `fIPrfP` `O` `O`	–	–	–	–	Δ	Δ	0	0
TSX	(SP) ⇒ X *Translates to* TFR SP,X	INH	`B7 75`	`P`	–	–	–	–	–	–	–	–
TSY	(SP) ⇒ Y *Translates to* TFR SP,Y	INH	`B7 76`	`P`	–	–	–	–	–	–	–	–
TXS	(X) ⇒ SP *Translates to* TFR X,SP	INH	`B7 57`	`P`	–	–	–	–	–	–	–	–
TYS	(Y) ⇒ SP *Translates to* TFR Y,SP	INH	`B7 67`	`P`	–	–	–	–	–	–	–	–
WAI (before interrupt) (when interrupt comes)	(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) ⇒ $M_{(SP)}$; WAIT for interrupt	INH	`3E`	`OSSSfSsf` `VfPPP`	– or – or –	– – 1	– – –	– 1 1	– – –	– – –	– – –	– – –

Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
WAV (add if interrupt)	$\sum_{i=1}^{B} S_i F_i \Rightarrow Y{:}D$ $\sum_{i=1}^{B} F_i \Rightarrow X$ Calculate Sum of Products and Sum of Weights for Weighted Average Calculation Initialize B, X, and Y before WAV. B specifies 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-bits. If interrupted, six extra bytes of stack used for intermediate values	Special	18 3C	Off(frrffff)O SSS + UUUrr	–	–	?	–	?	Δ	?	?
wavr pseudo-instruction	*see* WAV Resume executing an interrupted WAV instruction (recover intermediate results from stack rather than initializing them to zero)	Special	3C		–	–	?	–	?	Δ	?	?
XGDX	(D) ⇔ (X) *Translates to* EXG D, X	INH	B7 C5	P	–	–	–	–	–	–	–	–
XGDY	(D) ⇔ (Y) *Translates to* EXG D, Y	INH	B7 C6	P	–	–	–	–	–	–	–	–

Table 1. Indexed Addressing Mode Postbyte Encoding (xb)

00 0,X 5b const	10 –16,X 5b const	20 1,+X pre-inc	30 1,X+ post-inc	40 0,Y 5b const	50 –16,Y 5b const	60 1,+Y pre-inc	70 1,Y+ post-inc	80 0,SP 5b const	90 –16,SP 5b const	A0 1,+SP pre-inc	B0 1,SP+ post-inc	C0 0,PC 5b const	D0 –16,PC 5b const	E0 n,X 9b const	F0 n,SP 9b const
01 1,X 5b const	11 –15,X 5b const	21 2,+X pre-inc	31 2,X+ post-inc	41 1,Y 5b const	51 –15,Y 5b const	61 2,+Y pre-inc	71 2,Y+ post-inc	81 1,SP 5b const	91 –15,SP 5b const	A1 2,+SP pre-inc	B1 2,SP+ post-inc	C1 1,PC 5b const	D1 –15,PC 5b const	E1 –n,X 9b const	F1 –n,SP 9b const
02 2,X 5b const	12 –14,X 5b const	22 3,+X pre-inc	32 3,X+ post-inc	42 2,Y 5b const	52 –14,Y 5b const	62 3,+Y pre-inc	72 3,Y+ post-inc	82 2,SP 5b const	92 –14,SP 5b const	A2 3,+SP pre-inc	B2 3,SP+ post-inc	C2 2,PC 5b const	D2 –14,PC 5b const	E2 n,X 16b const	F2 n,SP 16b const
03 3,X 5b const	13 –13,X 5b const	23 4,+X pre-inc	33 4,X+ post-inc	43 3,Y 5b const	53 –13,Y 5b const	63 4,+Y pre-inc	73 4,Y+ post-inc	83 3,SP 5b const	93 –13,SP 5b const	A3 4,+SP pre-inc	B3 4,SP+ post-inc	C3 3,PC 5b const	D3 –13,PC 5b const	E3 [n,X] 16b indr	F3 [n,SP] 16b indr
04 4,X 5b const	14 –12,X 5b const	24 5,+X pre-inc	34 5,X+ post-inc	44 4,Y 5b const	54 –12,Y 5b const	64 5,+Y pre-inc	74 5,Y+ post-inc	84 4,SP 5b const	94 –12,SP 5b const	A4 5,+SP pre-inc	B4 5,SP+ post-inc	C4 4,PC 5b const	D4 –12,PC 5b const	E4 A,X A offset	F4 A,SP A offset
05 5,X 5b const	15 –11,X 5b const	25 6,+X pre-inc	35 6,X+ post-inc	45 5,Y 5b const	55 –11,Y 5b const	65 6,+Y pre-inc	75 6,Y+ post-inc	85 5,SP 5b const	95 –11,SP 5b const	A5 6,+SP pre-inc	B5 6,SP+ post-inc	C5 5,PC 5b const	D5 –11,PC 5b const	E5 B,X B offset	F5 B,SP B offset
06 6,X 5b const	16 –10,X 5b const	26 7,+X pre-inc	36 7,X+ post-inc	46 6,Y 5b const	56 –10,Y 5b const	66 7,+Y pre-inc	76 7,Y+ post-inc	86 6,SP 5b const	96 –10,SP 5b const	A6 7,+SP pre-inc	B6 7,SP+ post-inc	C6 6,PC 5b const	D6 –10,PC 5b const	E6 D,X D offset	F6 D,SP D offset
07 7,X 5b const	17 –9,X 5b const	27 8,+X pre-inc	37 8,X+ post-inc	47 7,Y 5b const	57 –9,Y 5b const	67 8,+Y pre-inc	77 8,Y+ post-inc	87 7,SP 5b const	97 –9,SP 5b const	A7 8,+SP pre-inc	B7 8,SP+ post-inc	C7 7,PC 5b const	D7 –9,PC 5b const	E7 [D,X] D indirect	F7 [D,SP] D indirect
08 8,X 5b const	18 –8,X 5b const	28 8,–X pre-dec	38 8,X– post-dec	48 8,Y 5b const	58 –8,Y 5b const	68 8,–Y pre-dec	78 8,Y– post-dec	88 8,SP 5b const	98 –8,SP 5b const	A8 8,–SP pre-dec	B8 8,SP– post-dec	C8 8,PC 5b const	D8 –8,PC 5b const	E8 n,Y 9b const	F8 n,PC 9b const
09 9,X 5b const	19 –7,X 5b const	29 7,–X pre-dec	39 7,X– post-dec	49 9,Y 5b const	59 –7,Y 5b const	69 7,–Y pre-dec	79 7,Y– post-dec	89 9,SP 5b const	99 –7,SP 5b const	A9 7,–SP pre-dec	B9 7,SP– post-dec	C9 9,PC 5b const	D9 –7,PC 5b const	E9 –n,Y 9b const	F9 –n,PC 9b const
0A 10,X 5b const	1A –6,X 5b const	2A 6,–X pre-dec	3A 6,X– post-dec	4A 10,Y 5b const	5A –6,Y 5b const	6A 6,–Y pre-dec	7A 6,Y– post-dec	8A 10,SP 5b const	9A –6,SP 5b const	AA 6,–SP pre-dec	BA 6,SP– post-dec	CA 10,PC 5b const	DA –6,PC 5b const	EA n,Y 16b const	FA n,PC 16b const
0B 11,X 5b const	1B –5,X 5b const	2B 5,–X pre-dec	3B 5,X– post-dec	4B 11,Y 5b const	5B –5,Y 5b const	6B 5,–Y pre-dec	7B 5,Y– post-dec	8B 11,SP 5b const	9B –5,SP 5b const	AB 5,–SP pre-dec	BB 5,SP– post-dec	CB 11,PC 5b const	DB –5,PC 5b const	EB [n,Y] 16b indr	FB [n,PC] 16b indr
0C 12,X 5b const	1C –4,X 5b const	2C 4,–X pre-dec	3C 4,X– post-dec	4C 12,Y 5b const	5C –4,Y 5b const	6C 4,–Y pre-dec	7C 4,Y– post-dec	8C 12,SP 5b const	9C –4,SP 5b const	AC 4,–SP pre-dec	BC 4,SP– post-dec	CC 12,PC 5b const	DC –4,PC 5b const	EC A,Y A offset	FC A,PC A offset
0D 13,X 5b const	1D –3,X 5b const	2D 3,–X pre-dec	3D 3,X– post-dec	4D 13,Y 5b const	5D –3,Y 5b const	6D 3,–Y pre-dec	7D 3,Y– post-dec	8D 13,SP 5b const	9D –3,SP 5b const	AD 3,–SP pre-dec	BD 3,SP– post-dec	CD 13,PC 5b const	DD –3,PC 5b const	ED B,Y B offset	FD B,PC B offset
0E 14,X 5b const	1E –2,X 5b const	2E 2,–X pre-dec	3E 2,X– post-dec	4E 14,Y 5b const	5E –2,Y 5b const	6E 2,–Y pre-dec	7E 2,Y– post-dec	8E 14,SP 5b const	9E –2,SP 5b const	AE 2,–SP pre-dec	BE 2,SP– post-dec	CE 14,PC 5b const	DE –2,PC 5b const	EE D,Y D offset	FE D,PC D offset
0F 15,X 5b const	1F –1,X 5b const	2F 1,–X pre-dec	3F 1,X– post-dec	4F 15,Y 5b const	5F –1,Y 5b const	6F 1,–Y pre-dec	7F 1,Y– post-dec	8F 15,SP 5b const	9F –1,SP 5b const	AF 1,–SP pre-dec	BF 1,SP– post-dec	CF 15,PC 5b const	DF –1,PC 5b const	EF [D,Y] D indirect	FF [D,PC] D indirect

Key to Table 1

Table 2. Indexed Addressing Mode Summary

Postbyte Code (xb)	Operand Syntax	Comments
rr0nnnnn	,r n,r –n,r	**5-bit constant offset** n = –16 to +15 rr can specify X, Y, SP, or PC
111rr0zs	n,r –n,r	**Constant offset** (9- or 16-bit signed) z- 0 = 9-bit with sign in LSB of postbyte (s) 1 = 16-bit if z = s = 1, 16-bit offset indexed-indirect (see below) rr can specify X, Y, SP, or PC
rr1pnnnn	n,–r n,+r n,r– n,r+	**Auto pre-decrement /increment or Auto post-decrement/increment**; p = pre-(0) or post-(1), n = –8 to –1, +1 to +8 rr can specify X, Y, or SP (PC not a valid choice)
111rr1aa	A,r B,r D,r	**Accumulator offset** (unsigned 8-bit or 16-bit) aa - 00 = A 01 = B 10 = D (16-bit) 11 = see accumulator D offset indexed-indirect rr can specify X, Y, SP, or PC
111rr011	[n,r]	**16-bit offset indexed-indirect** rr can specify X, Y, SP, or PC
111rr111	[D,r]	**Accumulator D offset indexed-indirect** rr can specify X, Y, SP, or PC

Table 3. Transfer and Exchange Postbyte Encoding

TRANSFERS									
⇓LS	MS⇒	0	1	2	3	4	5	6	7
0		A ⇒ A	B ⇒ A	CCR ⇒ A	$TMP3_L$ ⇒ A	B ⇒ A	X_L ⇒ A	Y_L ⇒ A	SP_L ⇒ A
1		A ⇒ B	B ⇒ B	CCR ⇒ B	$TMP3_L$ ⇒ B	B ⇒ B	X_L ⇒ B	Y_L ⇒ B	SP_L ⇒ B
2		A ⇒ CCR	B ⇒ CCR	CCR ⇒ CCR	$TMP3_L$ ⇒ CCR	B ⇒ CCR	X_L ⇒ CCR	Y_L ⇒ CCR	SP_L ⇒ CCR
3		sex:A ⇒ TMP2	sex:B ⇒ TMP2	sex:CCR ⇒ TMP2	TMP3 ⇒ TMP2	D ⇒ TMP2	X ⇒ TMP2	Y ⇒ TMP2	SP ⇒ TMP2
4		sex:A ⇒ D SEX A,D	sex:B ⇒ D SEX B,D	sex:CCR ⇒ D SEX CCR,D	TMP3 ⇒ D	D ⇒ D	X ⇒ D	Y ⇒ D	SP ⇒ D
5		sex:A ⇒ X SEX A,X	sex:B ⇒ X SEX B,X	sex:CCR ⇒ X SEX CCR,X	TMP3 ⇒ X	D ⇒ X	X ⇒ X	Y ⇒ X	SP ⇒ X
6		sex:A ⇒ Y SEX A,Y	sex:B ⇒ Y SEX B,Y	sex:CCR ⇒ Y SEX CCR,Y	TMP3 ⇒ Y	D ⇒ Y	X ⇒ Y	Y ⇒ Y	SP ⇒ Y
7		sex:A ⇒ SP SEX A,SP	sex:B ⇒ SP SEX B,SP	sex:CCR ⇒ SP SEX CCR,SP	TMP3 ⇒ SP	D ⇒ SP	X ⇒ SP	Y ⇒ SP	SP ⇒ SP
EXCHANGES									
⇓LS	MS⇒	8	9	A	B	C	D	E	F
0		A ⇔ A	B ⇔ A	CCR ⇔ A	$TMP3_L$ ⇒ A $00:A ⇒ TMP3	B ⇒ A A ⇒ B	X_L ⇒ A $00:A ⇒ X	Y_L ⇒ A $00:A ⇒ Y	SP_L ⇒ A $00:A ⇒ SP
1		A ⇔ B	B ⇔ B	CCR ⇔ B	$TMP3_L$ ⇒ B $FF:B ⇒ TMP3	B ⇒ B $FF ⇒ A	X_L ⇒ B $FF:B ⇒ X	Y_L ⇒ B $FF:B ⇒ Y	SP_L ⇒ B $FF:B ⇒ SP
2		A ⇔ CCR	B ⇔ CCR	CCR ⇔ CCR	$TMP3_L$ ⇒ CCR $FF:CCR ⇒ TMP3	B ⇒ CCR $FF:CCR ⇒ D	X_L ⇒ CCR $FF:CCR ⇒ X	Y_L ⇒ CCR $FF:CCR ⇒ Y	SP_L ⇒ CCR $FF:CCR ⇒ SP
3		$00:A ⇒ TMP2 $TMP2_L$ ⇒ A	$00:B ⇒ TMP2 $TMP2_L$ ⇒ B	$00:CCR ⇒ TMP2 $TMP2_L$ ⇒ CCR	TMP3 ⇔ TMP2	D ⇔ TMP2	X ⇔ TMP2	Y ⇔ TMP2	SP ⇔ TMP2
4		$00:A ⇒ D	$00:B ⇒ D	$00:CCR ⇒ D B ⇒ CCR	TMP3 ⇔ D	D ⇔ D	X ⇔ D	Y ⇔ D	SP ⇔ D
5		$00:A ⇒ X X_L ⇒ A	$00:B ⇒ X X_L ⇒ B	$00:CCR ⇒ X X_L ⇒ CCR	TMP3 ⇔ X	D ⇔ X	X ⇔ X	Y ⇔ X	SP ⇔ X
6		$00:A ⇒ Y Y_L ⇒ A	$00:B ⇒ Y Y_L ⇒ B	$00:CCR ⇒ Y Y_L ⇒ CCR	TMP3 ⇔ Y	D ⇔ Y	X ⇔ Y	Y ⇔ Y	SP ⇔ Y
7		$00:A ⇒ SP SP_L ⇒ A	$00:B ⇒ SP SP_L ⇒ B	$00:CCR ⇒ SP SP_L ⇒ CCR	TMP3 ⇔ SP	D ⇔ SP	X ⇔ SP	Y ⇔ SP	SP ⇔ SP

TMP2 and TMP3 registers are for factory use only.

CPU12 Reference Guide

Table 4. Loop Primitive Postbyte Encoding (lb)

00 A DBEQ (+)	10 A DBEQ (–)	20 A DBNE (+)	30 A DBNE (–)	40 A TBEQ (+)	50 A TBEQ (–)	60 A TBNE (+)	70 A TBNE (–)	80 A IBEQ (+)	90 A IBEQ (–)	A0 A IBNE (+)	B0 A IBNE (–)
01 B DBEQ (+)	11 B DBEQ (–)	21 B DBNE (+)	31 B DBNE (–)	41 B TBEQ (+)	51 B TBEQ (–)	61 B TBNE (+)	71 B TBNE (–)	81 B IBEQ (+)	91 B IBEQ (–)	A1 B IBNE (+)	B1 B IBNE (–)
02 —	12 —	22 —	32 —	42 —	52 —	62 —	72 —	82 —	92 —	A2 —	B2 —
03 —	13 —	23 —	33 —	43 —	53 —	63 —	73 —	83 —	93 —	A3 —	B3 —
04 D DBEQ (+)	14 D DBEQ (–)	24 D DBNE (+)	34 D DBNE (–)	44 D TBEQ (+)	54 D TBEQ (–)	64 D TBNE (+)	74 D TBNE (–)	84 D IBEQ (+)	94 D IBEQ (–)	A4 D IBNE (+)	B4 D IBNE (–)
05 X DBEQ (+)	15 X DBEQ (–)	25 X DBNE (+)	35 X DBNE (–)	45 X TBEQ (+)	55 X TBEQ (–)	65 X TBNE (+)	75 X TBNE (–)	85 X IBEQ (+)	95 X IBEQ (–)	A5 X IBNE (+)	B5 X IBNE (–)
06 Y DBEQ (+)	16 Y DBEQ (–)	26 Y DBNE (+)	36 Y DBNE (–)	46 Y TBEQ (+)	56 Y TBEQ (–)	66 Y TBNE (+)	76 Y TBNE (–)	86 Y IBEQ (+)	96 Y IBEQ (–)	A6 Y IBNE (+)	B6 Y IBNE (–)
07 SP DBEQ (+)	17 SP DBEQ (–)	27 SP DBNE (+)	37 SP DBNE (–)	47 SP TBEQ (+)	57 SP TBEQ (–)	67 SP TBNE (+)	77 SP TBNE (–)	87 SP IBEQ (+)	97 SP IBEQ (–)	A7 SP IBNE (+)	B7 SP IBNE (–)

Key to Table 4

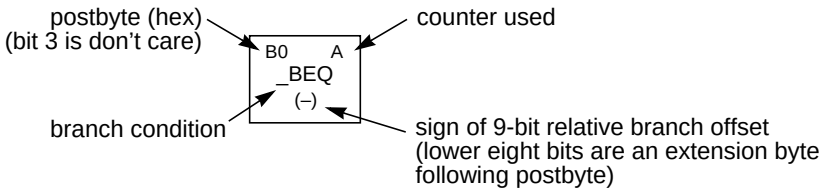


Table 5. Branch/Complementary Branch

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	$Z + (N \oplus V) = 0$	r≤m	BLE	2F	Signed
r≥m	BGE	2C	$N \oplus V = 0$	r<m	BLT	2D	Signed
r=m	BEQ	27	$Z = 1$	r≠m	BNE	26	Signed
r≤m	BLE	2F	$Z + (N \oplus V) = 1$	r>m	BGT	2E	Signed
r<m	BLT	2D	$N \oplus V = 1$	r≥m	BGE	2C	Signed
r>m	BHI	22	$C + Z = 0$	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	$C = 0$	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	$Z = 1$	r≠m	BNE	26	Unsigned
r≤m	BLS	23	$C + Z = 1$	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	$C = 1$	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
r=0	BEQ	27	$Z = 1$	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

For 16-bit offset long branches preceed opcode with a $18 page prebyte.

Memory Expansion

Some M68HC12 derivatives support >4 megabytes of program memory.

Memory precedence

— Highest —
On-chip registers (usually $0000 or $1000)
BDM ROM (only when BDM active)
On-chip RAM
On-chip EEPROM
On-chip program memory (FLASH or ROM)
Expansion windows (on MCUs with expanded memory)
Other external memory
— Lowest —

CPU sees 64 Kbytes of address space (CPU_ADDR [15:0])

PPAGE 8-bit register to select 1 of 256 —16 Kbyte program pages
DPAGE 8-bit register to select 1 of 256 — 4 Kbyte data pages
EPAGE 8-bit register to select 1 of 256 — 1 Kbyte extra pages

Extended address is 22 bits (EXT_ADDR [21:0])

Program expansion window works with CALL and RTC instructions to simplify program access to extended memory space. Data and extra expansion windows (when present) use traditional banked expansion memory techniques.

Program window

If CPU_ADDR [15:0] = $8000–BFFF and PWEN = 1
Then EXT_ADDR [21:0] = PPAGE [7:0]:CPU_ADDR [13:0]

Program window works with CALL/RTC to automate bank switching. 256 pages (banks) of 16 Kbytes each = 4 M.

Data window

If CPU_ADDR [15:0] = $7000–7FFF and DWEN = 1
Then EXT_ADDR [21:0] = 1:1:DPAGE [7:0]:CPU_ADDR [11:0]

User program controls DPAGE value

Extra window

If CPU_ADDR [15:0] = $0000–03FF and EWDIR = 1 and EWEN = 1

or CPU_ADDR [15:0] = $0400–07FF and EWDIR = 0 and EWEN = 1

Then EXT_ADDR [21:0] = 1:1:1:1:EPAGE [7:0]:CPU_ADDR [9:0]

User program controls EPAGE value

CPU address not in any enabled window

EXT_ADDR [21:0] = 1:1:1:1:1:1:CPU_ADDR [15:0]

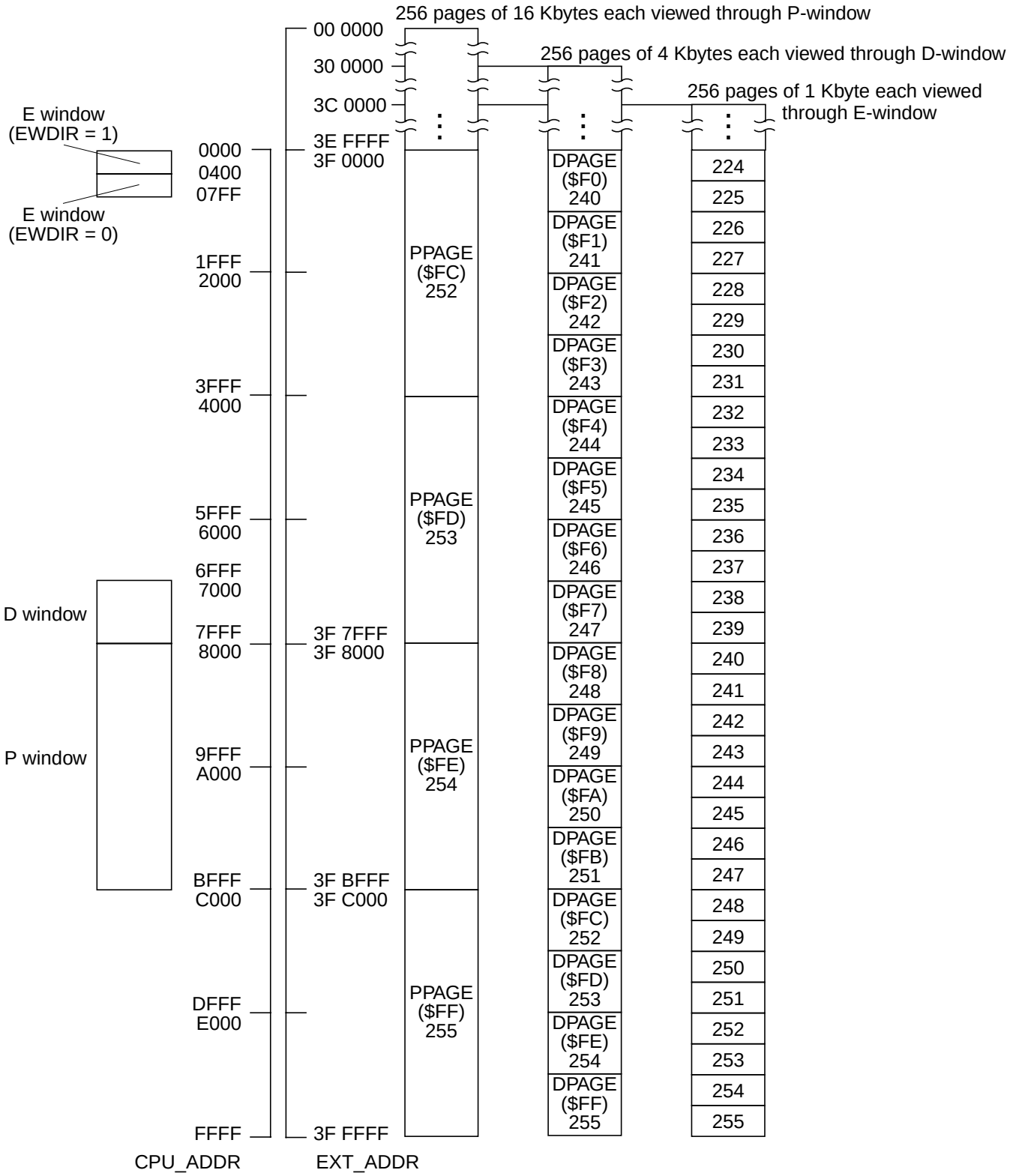

256 pages of 16 Kbytes each viewed through P-window
256 pages of 4 Kbytes each viewed through D-window
256 pages of 1 Kbyte each viewed through E-window
E window (EWDIR = 1)
E window (EWDIR = 0)
D window
P window
0000
0400
07FF
1FFF
2000
3FFF
4000
5FFF
6000
6FFF
7000
7FFF
8000
9FFF
A000
BFFF
C000
DFFF
E000
FFFF
CPU_ADDR
00 0000
30 0000
3C 0000
3E FFFF
3F 0000
3F 7FFF
3F 8000
3F BFFF
3F C000
3F FFFF
EXT_ADDR
PPAGE ($FC) 252
PPAGE ($FD) 253
PPAGE ($FE) 254
PPAGE ($FF) 255
DPAGE ($F0) 240
DPAGE ($F1) 241
DPAGE ($F2) 242
DPAGE ($F3) 243
DPAGE ($F4) 244
DPAGE ($F5) 245
DPAGE ($F6) 246
DPAGE ($F7) 247
DPAGE ($F8) 248
DPAGE ($F9) 249
DPAGE ($FA) 250
DPAGE ($FB) 251
DPAGE ($FC) 252
DPAGE ($FD) 253
DPAGE ($FE) 254
DPAGE ($FF) 255
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255

Table 6. CPU12 Opcode Map (Sheet 1 of 2)

00 *5	10 1	20 3	30 3	40 1	50 1	60 3-6	70 4	80 1	90 3	A0 3-6	B0 3	C0 1	D0 3	E0 3-6	F0 3
BGND	ANDCC	BRA	PULX	NEGA	NEGB	NEG	NEG	SUBA	SUBA	SUBA	SUBA	SUBB	SUBB	SUBB	SUBB
IH 1	IM 2	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
01 5	11 11	21 1	31 3	41 1	51 1	61 3-6	71 4	81 1	91 3	A1 3-6	B1 3	C1 1	D1 3	E1 3-6	F1 3
MEM	EDIV	BRN	PULY	COMA	COMB	COM	COM	CMPA	CMPA	CMPA	CMPA	CMPB	CMPB	CMPB	CMPB
IH 1	IH 1	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
02 1	12 3	22 3/1	32 3	42 1	52 1	62 3-6	72 4	82 1	92 3	A2 3-6	B2 3	C2 1	D2 3	E2 3-6	F2 3
INY	MUL	BHI	PULA	INCA	INCB	INC	INC	SBCA	SBCA	SBCA	SBCA	SBCB	SBCB	SBCB	SBCB
IH 1	IH 1	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
03 1	13 3	23 3/1	33 3	43 1	53 1	63 3-6	73 4	83 2	93 3	A3 3-6	B3 3	C3 2	D3 3	E3 3-6	F3 3
DEY	EMUL	BLS	PULB	DECA	DECB	DEC	DEC	SUBD	SUBD	SUBD	SUBD	ADDD	ADDD	ADDD	ADDD
IH 1	IH 1	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3
04 3	14 1	24 3/1	34 2	44 1	54 1	64 3-6	74 4	84 1	94 3	A4 3-6	B4 3	C4 1	D4 3	E4 3-6	F4 3
loop‡	ORCC	BCC	PSHX	LSRA	LSRB	LSR	LSR	ANDA	ANDA	ANDA	ANDA	ANDB	ANDB	ANDB	ANDB
RL 3	IM 2	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
05 3-6	15 4-7	25 3/1	35 2	45 1	55 1	65 3-6	75 4	85 1	95 3	A5 3-6	B5 3	C5 1	D5 3	E5 3-6	F5 3
JMP	JSR	BCS	PSHY	ROLA	ROLB	ROL	ROL	BITA	BITA	BITA	BITA	BITB	BITB	BITB	BITB
ID 2-4	ID 2-4	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
06 3	16 4	26 3/1	36 2	46 1	56 1	66 3-6	76 4	86 1	96 3	A6 3-6	B6 3	C6 1	D6 3	E6 3-6	F6 3
JMP	JSR	BNE	PSHA	RORA	RORB	ROR	ROR	LDAA	LDAA	LDAA	LDAA	LDAB	LDAB	LDAB	LDAB
EX 3	EX 3	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
07 4	17 4	27 3/1	37 2	47 1	57 1	67 3-6	77 4	87 1	97 1	A7 1	B7 1	C7 1	D7 1	E7 3-6	F7 3
BSR	JSR	BEQ	PSHB	ASRA	ASRB	ASR	ASR	CLRA	TSTA	NOP	TFR/EXG	CLRB	TSTB	TST	TST
RL 2	DI 2	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IH 1	IH 1	IH 1	IH 2	IH 1	IH 1	ID 2-4	EX 3
08 1	18 -	28 3/1	38 3	48 1	58 1	68 3-6	78 4	88 1	98 3	A8 3-6	B8 3	C8 1	D8 3	E8 3-6	F8 3
INX	page 2	BVC	PULC	ASLA	ASLB	ASL	ASL	EORA	EORA	EORA	EORA	EORB	EORB	EORB	EORB
IH 1	- -	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
09 1	19 2	29 3/1	39 2	49 1	59 1	69 2-5	79 3	89 1	99 3	A9 3-6	B9 3	C9 1	D9 3	E9 3-6	F9 3
DEX	LEAY	BVS	PSHC	LSRD	ASLD	CLR	CLR	ADCA	ADCA	ADCA	ADCA	ADCB	ADCB	ADCB	ADCB
IH 1	ID 2-4	RL 2	IH 1	IH 1	IH 1	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
0A 6	1A 2	2A 3/1	3A 3	4A 8	5A 2	6A 2-5	7A 3	8A 1	9A 3	AA 3-6	BA 3	CA 1	DA 3	EA 3-6	FA 3
RTC	LEAX	BPL	PULD	CALL	STAA	STAA	STAA	ORAA	ORAA	ORAA	ORAA	ORAB	ORAB	ORAB	ORAB
IH 1	ID 2-4	RL 2	IH 1	EX 4	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
0B 8	1B 2	2B 3/1	3B 2	4B 8-10	5B 2	6B 2-5	7B 3	8B 1	9B 3	AB 3-6	BB 3	CB 1	DB 3	EB 3-6	FB 3
RTI	LEAS	BMI	PSHD	CALL	STAB	STAB	STAB	ADDA	ADDA	ADDA	ADDA	ADDB	ADDB	ADDB	ADDB
IH 1	ID 2-4	RL 2	IH 1	ID 2-5	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3	IM 2	DI 2	ID 2-4	EX 3
0C 4-6	1C 4	2C 3/1	3C *+9	4C 4	5C 2	6C 2-5	7C 3	8C 2	9C 3	AC 3-6	BC 3	CC 2	DC 3	EC 3-6	FC 3
BSET	BSET	BGE	wavr	BSET	STD	STD	STD	CPD	CPD	CPD	CPD	LDD	LDD	LDD	LDD
ID 3-5	EX 4	RL 2	SP 1	DI 3	DI 2	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3
0D 4-6	1D 4	2D 3/1	3D 5	4D 4	5D 2	6D 2-5	7D 3	8D 2	9D 3	AD 3-6	BD 3	CD 2	DD 3	ED 3-6	FD 3
BCLR	BCLR	BLT	RTS	BCLR	STY	STY	STY	CPY	CPY	CPY	CPY	LDY	LDY	LDY	LDY
ID 3-5	EX 4	RL 2	IH 1	DI 3	DI 2	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3
0E 4-8	1E 5	2E 3/1	3E *8	4E 4	5E 2	6E 2-5	7E 3	8E 2	9E 3	AE 3-6	BE 3	CE 2	DE 3	EE 3-6	FE 3
BRSET	BRSET	BGT	WAI	BRSET	STX	STX	STX	CPX	CPX	CPX	CPX	LDX	LDX	LDX	LDX
ID 4-6	EX 5	RL 2	IH 1	DI 4	DI 2	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3
0F 4-8	1F 5	2F 3/1	3F 9	4F 4	5F 2	6F 2-5	7F 3	8F 2	9F 3	AF 3-6	BF 3	CF 2	DF 3	EF 3-6	FF 3
BRCLR	BRCLR	BLE	SWI	BRCLR	STS	STS	STS	CPS	CPS	CPS	CPS	LDS	LDS	LDS	LDS
ID 4-6	EX 5	RL 2	IH 1	DI 4	DI 2	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3	IM 3	DI 2	ID 2-4	EX 3

Table 6. CPU12 Opcode Map (Sheet 2 of 2)

00 4 MOVW IM-ID 5	10 12 IDIV IH 2	20 4 LBRA RL 4	30 10 TRAP IH 2	40 10 TRAP IH 2	50 10 TRAP IH 2	60 10 TRAP IH 2	70 10 TRAP IH 2	80 10 TRAP IH 2	90 10 TRAP IH 2	A0 10 TRAP IH 2	B0 10 TRAP IH 2	C0 10 TRAP IH 2	D0 10 TRAP IH 2	E0 10 TRAP IH 2	F0 10 TRAP IH 2
01 5 MOVW EX-ID 5	11 12 FDIV IH 2	21 3 LBRN RL 4	31 10 TRAP IH 2	41 10 TRAP IH 2	51 10 TRAP IH 2	61 10 TRAP IH 2	71 10 TRAP IH 2	81 10 TRAP IH 2	91 10 TRAP IH 2	A1 10 TRAP IH 2	B1 10 TRAP IH 2	C1 10 TRAP IH 2	D1 10 TRAP IH 2	E1 10 TRAP IH 2	F1 10 TRAP IH 2
02 5 MOVW ID-ID 4	12 13 EMACS SP 4	22 4/3 LBHI RL 4	32 10 TRAP IH 2	42 10 TRAP IH 2	52 10 TRAP IH 2	62 10 TRAP IH 2	72 10 TRAP IH 2	82 10 TRAP IH 2	92 10 TRAP IH 2	A2 10 TRAP IH 2	B2 10 TRAP IH 2	C2 10 TRAP IH 2	D2 10 TRAP IH 2	E2 10 TRAP IH 2	F2 10 TRAP IH 2
03 5 MOVW IM-EX 6	13 3 EMULS IH 2	23 4/3 LBLS RL 4	33 10 TRAP IH 2	43 10 TRAP IH 2	53 10 TRAP IH 2	63 10 TRAP IH 2	73 10 TRAP IH 2	83 10 TRAP IH 2	93 10 TRAP IH 2	A3 10 TRAP IH 2	B3 10 TRAP IH 2	C3 10 TRAP IH 2	D3 10 TRAP IH 2	E3 10 TRAP IH 2	F3 10 TRAP IH 2
04 6 MOVW EX-EX 6	14 12 EDIVS IH 2	24 4/3 LBCC RL 4	34 10 TRAP IH 2	44 10 TRAP IH 2	54 10 TRAP IH 2	64 10 TRAP IH 2	74 10 TRAP IH 2	84 10 TRAP IH 2	94 10 TRAP IH 2	A4 10 TRAP IH 2	B4 10 TRAP IH 2	C4 10 TRAP IH 2	D4 10 TRAP IH 2	E4 10 TRAP IH 2	F4 10 TRAP IH 2
05 5 MOVW ID-EX 5	15 12 IDIVS IH 2	25 4/3 LBCS RL 4	35 10 TRAP IH 2	45 10 TRAP IH 2	55 10 TRAP IH 2	65 10 TRAP IH 2	75 10 TRAP IH 2	85 10 TRAP IH 2	95 10 TRAP IH 2	A5 10 TRAP IH 2	B5 10 TRAP IH 2	C5 10 TRAP IH 2	D5 10 TRAP IH 2	E5 10 TRAP IH 2	F5 10 TRAP IH 2
06 2 ABA IH 2	16 2 SBA IH 2	26 4/3 LBNE RL 4	36 10 TRAP IH 2	46 10 TRAP IH 2	56 10 TRAP IH 2	66 10 TRAP IH 2	76 10 TRAP IH 2	86 10 TRAP IH 2	96 10 TRAP IH 2	A6 10 TRAP IH 2	B6 10 TRAP IH 2	C6 10 TRAP IH 2	D6 10 TRAP IH 2	E6 10 TRAP IH 2	F6 10 TRAP IH 2
07 3 DAA IH 2	17 2 CBA IH 2	27 4/3 LBEQ RL 4	37 10 TRAP IH 2	47 10 TRAP IH 2	57 10 TRAP IH 2	67 10 TRAP IH 2	77 10 TRAP IH 2	87 10 TRAP IH 2	97 10 TRAP IH 2	A7 10 TRAP IH 2	B7 10 TRAP IH 2	C7 10 TRAP IH 2	D7 10 TRAP IH 2	E7 10 TRAP IH 2	F7 10 TRAP IH 2
08 4 MOVB IM-ID 4	18 4-7 MAXA ID 3-5	28 4/3 LBVC RL 4	38 10 TRAP IH 2	48 10 TRAP IH 2	58 10 TRAP IH 2	68 10 TRAP IH 2	78 10 TRAP IH 2	88 10 TRAP IH 2	98 10 TRAP IH 2	A8 10 TRAP IH 2	B8 10 TRAP IH 2	C8 10 TRAP IH 2	D8 10 TRAP IH 2	E8 10 TRAP IH 2	F8 10 TRAP IH 2
09 5 MOVB EX-ID 5	19 4-7 MINA ID 3-5	29 4/3 LBVS RL 4	39 10 TRAP IH 2	49 10 TRAP IH 2	59 10 TRAP IH 2	69 10 TRAP IH 2	79 10 TRAP IH 2	89 10 TRAP IH 2	99 10 TRAP IH 2	A9 10 TRAP IH 2	B9 10 TRAP IH 2	C9 10 TRAP IH 2	D9 10 TRAP IH 2	E9 10 TRAP IH 2	F9 10 TRAP IH 2
0A 5 MOVB ID-ID 4	1A 4-7 EMAXD ID 3-5	2A 4/3 LBPL RL 4	3A *3n REV SP 2	4A 10 TRAP IH 2	5A 10 TRAP IH 2	6A 10 TRAP IH 2	7A 10 TRAP IH 2	8A 10 TRAP IH 2	9A 10 TRAP IH 2	AA 10 TRAP IH 2	BA 10 TRAP IH 2	CA 10 TRAP IH 2	DA 10 TRAP IH 2	EA 10 TRAP IH 2	FA 10 TRAP IH 2
0B 4 MOVB IM-EX 5	1B 4-7 EMIND ID 3-5	2B 4/3 LBMI RL 4	3B *3n REVW SP 2	4B 10 TRAP IH 2	5B 10 TRAP IH 2	6B 10 TRAP IH 2	7B 10 TRAP IH 2	8B 10 TRAP IH 2	9B 10 TRAP IH 2	AB 10 TRAP IH 2	BB 10 TRAP IH 2	CB 10 TRAP IH 2	DB 10 TRAP IH 2	EB 10 TRAP IH 2	FB 10 TRAP IH 2
0C 6 MOVB EX-EX 6	1C 4-7 MAXM ID 3-5	2C 4/3 LBGE RL 4	3C *8B WAV SP 2	4C 10 TRAP IH 2	5C 10 TRAP IH 2	6C 10 TRAP IH 2	7C 10 TRAP IH 2	8C 10 TRAP IH 2	9C 10 TRAP IH 2	AC 10 TRAP IH 2	BC 10 TRAP IH 2	CC 10 TRAP IH 2	DC 10 TRAP IH 2	EC 10 TRAP IH 2	FC 10 TRAP IH 2
0D 5 MOVB ID-EX 5	1D 4-7 MINM ID 3-5	2D 4/3 LBLT RL 4	3D 8 TBL ID 3	4D 10 TRAP IH 2	5D 10 TRAP IH 2	6D 10 TRAP IH 2	7D 10 TRAP IH 2	8D 10 TRAP IH 2	9D 10 TRAP IH 2	AD 10 TRAP IH 2	BD 10 TRAP IH 2	CD 10 TRAP IH 2	DD 10 TRAP IH 2	ED 10 TRAP IH 2	FD 10 TRAP IH 2
0E 2 TAB IH 2	1E 4-7 EMAXM ID 3-5	2E 4/3 LBGT RL 4	3E *9+5 STOP IH 2	4E 10 TRAP IH 2	5E 10 TRAP IH 2	6E 10 TRAP IH 2	7E 10 TRAP IH 2	8E 10 TRAP IH 2	9E 10 TRAP IH 2	AE 10 TRAP IH 2	BE 10 TRAP IH 2	CE 10 TRAP IH 2	DE 10 TRAP IH 2	EE 10 TRAP IH 2	FE 10 TRAP IH 2
0F 2 TBA IH 2	1F 4-7 EMINM ID 3-5	2F 4/3 LBLE RL 4	3F 10 ETBL ID 3	4F 10 TRAP IH 2	5F 10 TRAP IH 2	6F 10 TRAP IH 2	7F 10 TRAP IH 2	8F 10 TRAP IH 2	9F 10 TRAP IH 2	AF 10 TRAP IH 2	BF 10 TRAP IH 2	CF 10 TRAP IH 2	DF 10 TRAP IH 2	EF 10 TRAP IH 2	FF 10 TRAP IH 2

* Refer to instruction summary for more information.

‡ The opcode $04 corresponds to one of the loop primitive instructions DBEQ, DBNE, IBEQ, IBNE, TBEQ, or TBNE.

CPU12 Reference Guide

Table 7. Hexadecimal to ASCII Conversion

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
$00	NUL	$20	SP *space*	$40	@	$60	` *grave*
$01	SOH	$21	!	$41	A	$61	a
$02	STX	$22	“ quote	$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 *beep*	$27	‘ *apost.*	$47	G	$67	g
$08	BS *back sp*	$28	(	$48	H	$68	h
$09	HT *tab*	$29	)	$49	I	$69	i
$0A	LF *linefeed*	$2A	*	$4A	J	$6A	j
$0B	VT	$2B	+	$4B	K	$6B	k
$0C	FF	$2C	, *comma*	$4C	L	$6C	l
$0D	CR *return*	$2D	- *dash*	$4D	M	$6D	m
$0E	SO	$2E	. *period*	$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	ESCAPE	$3B	;	$5B	[	$7B	{
$1C	FS	$3C	<	$5C	\	$7C	|
$1D	GS	$3D	=	$5D	]	$7D	}
$1E	RS	$3E	>	$5E	^	$7E	~
$1F	US	$3F	?	$5F	_ *under*	$7F	DEL *delete*

Hexadecimal to Decimal Conversion

To convert a hexadecimal number (up to four hexadecimal digits) to decimal, look up the decimal equivalent of each hexadecimal digit in **Table 8**. The decimal equivalent of the original hexadecimal number is the sum of the weights found in the table for all hexadecimal digits.

Table 8. Hexadecimal to/from Decimal Conversion

15 Bit 8				7 Bit 0			
15 12		11 8		7 4		3 0	
4th Hex Digit		3rd Hex Digit		2nd Hex Digit		1st Hex Digit	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,484	E	224	E	14
F	61,440	F	3,840	F	240	F	15

Decimal to Hexadecimal Conversion

To convert a decimal number (up to $65{,}535_{10}$) to hexadecimal, find the largest decimal number in **Table 8** that is less than or equal to the number you are converting. The corresponding hexadecimal digit is the most significant hexadecimal digit of the result. Subtract the decimal number found from the original decimal number to get the *remaining decimal value*. Repeat the procedure using the remaining decimal value for each subsequent hexadecimal digit.

CPU12 Reference Guide

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution, P.O. Box 5405, Denver, Colorado 80217, 1-800-441-2447 or 1-303-675-2140. Customer Focus Center, 1-800-521-6274

JAPAN: Nippon Motorola Ltd.: SPD, Strategic Planning Office, 141, 4-32-1 Nishi-Gotanda, Shinigawa-Ku, Tokyo, Japan. 03-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

Mfax™, Motorola Fax Back System: RMFAX0@email.sps.mot.com; http://sps.motorola.com/mfax/; TOUCHTONE, 1-602-244-6609; US and Canada ONLY, 1-800-774-1848

HOME PAGE: http://motorola.com/sps/

MOTOROLA

CPU12RG/D