68HC12 and 68HCS12 Instruction Set

Textbook Appendix A

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)

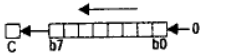
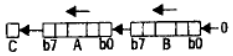
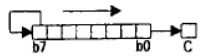
Machine Coding

`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.
`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.
`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.

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.

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
ABA	(A) + (B) ⇒ A Add Accumulators A and B	INH	18 06	2	–	–	Δ	–	Δ	Δ	Δ	Δ
ABX	(B) + (X) ⇒ X *Translates to* LEAX B,X	IDX	1A E5	2	–	–	–	–	–	–	–	–
ABY	(B) + (Y) ⇒ Y *Translates to* LEAY B,Y	IDX	19 ED	2	–	–	–	–	–	–	–	–
ADCA *opr*	(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	1 3 3 3 3 4 6 6	–	–	Δ	–	Δ	Δ	Δ	Δ
ADCB *opr*	(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	1 3 3 3 3 4 6 6	–	–	Δ	–	Δ	Δ	Δ	Δ
ADDA *opr*	(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	1 3 3 3 3 4 6 6	–	–	Δ	–	Δ	Δ	Δ	Δ
ADDB *opr*	(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	1 3 3 3 3 4 6 6	–	–	Δ	–	Δ	Δ	Δ	Δ
ADDD *opr*	(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	2 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	Δ	Δ

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
ANDA *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
ANDB *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
ANDCC *opr*	(CCR) • (M) ⇒ CCR Logical And CCR with Memory	IMM	10 ii	1	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
ASL *opr*  Arithmetic Shift Left		EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff	4 3 4 5 6 6	–	–	–	–	Δ	Δ	Δ	Δ
ASLA ASLB	Arithmetic Shift Left Accumulator A Arithmetic Shift Left Accumulator B	INH INH	48 58	1 1								
ASLD  Arithmetic Shift Left Double		INH	59	1	–	–	–	–	Δ	Δ	Δ	Δ
ASR *opr*  Arithmetic Shift Right		EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	77 hh ll 67 xb 67 xb ff 67 xb ee ff 67 xb 67 xb ee ff	4 3 4 5 6 6	–	–	–	–	Δ	Δ	Δ	Δ
ASRA ASRB	Arithmetic Shift Right Accumulator A Arithmetic Shift Right Accumulator B	INH INH	47 57	1 1								
BCC *rel*	Branch if Carry Clear (if C = 0)	REL	24 rr	3/1	–	–	–	–	–	–	–	–
BCLR *opr, msk*	(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	4 4 4 4 6	–	–	–	–	Δ	Δ	0	–
BCS *rel*	Branch if Carry Set (if C = 1)	REL	25 rr	3/1	–	–	–	–	–	–	–	–
BEQ *rel*	Branch if Equal (if Z = 1)	REL	27 rr	3/1	–	–	–	–	–	–	–	–
BGE *rel*	Branch if Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	2C rr	3/1	–	–	–	–	–	–	–	–
BGND	Place CPU in Background Mode see Background Mode section.	INH	00	5	–	–	–	–	–	–	–	–
BGT *rel*	Branch if Greater Than (if Z + (N ⊕ V) = 0) (signed)	REL	2E rr	3/1	–	–	–	–	–	–	–	–
BHI *rel*	Branch if Higher (if C + Z = 0) (unsigned)	REL	22 rr	3/1	–	–	–	–	–	–	–	–

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
BHS *rel*	Branch if Higher or Same (if C = 0) (unsigned) same function as BCC	REL	24 rr	3/1	–	–	–	–	–	–	–	–
BITA *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
BITB *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
BLE *rel*	Branch if Less Than or Equal (if Z + (N ⊕ V) = 1) (signed)	REL	2F rr	3/1	–	–	–	–	–	–	–	–
BLO *rel*	Branch if Lower (if C = 1) (unsigned) same function as BCS	REL	25 rr	3/1	–	–	–	–	–	–	–	–
BLS *rel*	Branch if Lower or Same (if C + Z = 1) (unsigned)	REL	23 rr	3/1	–	–	–	–	–	–	–	–
BLT *rel*	Branch if Less Than (if N ⊕ V = 1) (signed)	REL	2D rr	3/1	–	–	–	–	–	–	–	–
BMI *rel*	Branch if Minus (if N = 1)	REL	2B rr	3/1	–	–	–	–	–	–	–	–
BNE *rel*	Branch if Not Equal (if Z = 0)	REL	26 rr	3/1	–	–	–	–	–	–	–	–
BPL *rel*	Branch if Plus (if N = 0)	REL	2A rr	3/1	–	–	–	–	–	–	–	–
BRA *rel*	Branch Always (if 1 = 1)	REL	20 rr	3	–	–	–	–	–	–	–	–
BRCLR *opr, msk, rel*	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	4 5 4 6 8	–	–	–	–	–	–	–	–
BRN *rel*	Branch Never (if 1 = 0)	REL	21 rr	1	–	–	–	–	–	–	–	–
BRSET *opr, msk, rel*	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	4 5 4 6 8	–	–	–	–	–	–	–	–
BSET *opr, msk*	(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	4 4 4 4 6	–	–	–	–	Δ	Δ	0	–
BSR *rel*	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Subroutine address ⇒ PC Branch to Subroutine	REL	07 rr	4	–	–	–	–	–	–	–	–

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
BVC *rel*	Branch if Overflow Bit Clear (if V = 0)	REL	28 rr	3/1	–	–	–	–	–	–	–	–
BVS *rel*	Branch if Overflow Bit Set (if V = 1)	REL	29 rr	3/1	–	–	–	–	–	–	–	–
CALL *opr, page*	(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.)	EXT IDX IDX1 IDX2	4A hh ll pg 4B xb pg 4B xb ff pg 4B xb ee ff pg	8 8 8 9	–	–	–	–	–	–	–	–
CALL [D,*r*] CALL [*opr,r*]	Indirect modes get program address and new pg value based on pointer. *r* = X, Y, SP, or PC	[D,IDX] [IDX2]	4B xb 4B xb ee ff	10 10	–	–	–	–	–	–	–	–
CBA	(A) – (B) Compare 8-Bit Accumulators	INH	18 17	2	–	–	–	–	Δ	Δ	Δ	Δ
CLC	0 ⇒ C *Translates to* ANDCC #$FE	IMM	10 FE	1	–	–	–	–	–	–	–	0
CLI	0 ⇒ I *Translates to* ANDCC #$EF (enables I-bit interrupts)	IMM	10 EF	1	–	–	–	0	–	–	–	–
CLR *opr*	0 ⇒ M Clear Memory Location	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	79 hh ll 69 xb 69 xb ff 69 xb ee ff 69 xb 69 xb ee ff	3 2 3 3 5 5	–	–	–	–	0	1	0	0
CLRA CLRB	0 ⇒ A Clear Accumulator A 0 ⇒ B Clear Accumulator B	INH INH	87 C7	1 1								
CLV	0 ⇒ V *Translates to* ANDCC #$FD	IMM	10 FD	1	–	–	–	–	–	–	0	–
CMPA *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	Δ	Δ
CMPB *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	Δ	Δ

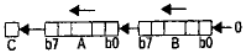
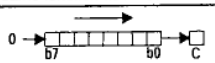
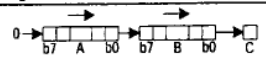
Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
COM *opr*	(M) ⇒ M *equivalent to* $FF – (M) ⇒ M	EXT	71 hh ll	4	–	–	–	–	Δ	Δ	0	1
	1's Complement Memory Location	IDX	61 xb	3								
		IDX1	61 xb ff	4								
		IDX2	61 xb ee ff	5								
		[D,IDX]	61 xb	6								
		[IDX2]	61 xb ee ff	6								
COMA	(A) ⇒ A Complement Accumulator A	INH	41	1								
COMB	(B) ⇒ B Complement Accumulator B	INH	51	1								
CPD *opr*	(A:B) – (M:M+1)	IMM	8C jj kk	2	–	–	–	–	Δ	Δ	Δ	Δ
	Compare D to Memory (16-Bit)	DIR	9C dd	3								
		EXT	BC hh ll	3								
		IDX	AC xb	3								
		IDX1	AC xb ff	3								
		IDX2	AC xb ee ff	4								
		[D,IDX]	AC xb	6								
		[IDX2]	AC xb ee ff	6								
CPS *opr*	(SP) – (M:M+1)	IMM	8F jj kk	2	–	–	–	–	Δ	Δ	Δ	Δ
	Compare SP to Memory (16-Bit)	DIR	9F dd	3								
		EXT	BF hh ll	3								
		IDX	AF xb	3								
		IDX1	AF xb ff	3								
		IDX2	AF xb ee ff	4								
		[D,IDX]	AF xb	6								
		[IDX2]	AF xb ee ff	6								
CPX *opr*	(X) – (M:M+1)	IMM	8E jj kk	2	–	–	–	–	Δ	Δ	Δ	Δ
	Compare X to Memory (16-Bit)	DIR	9E dd	3								
		EXT	BE hh ll	3								
		IDX	AE xb	3								
		IDX1	AE xb ff	3								
		IDX2	AE xb ee ff	4								
		[D,IDX]	AE xb	6								
		[IDX2]	AE xb ee ff	6								
CPY *opr*	(Y) – (M:M+1)	IMM	8D jj kk	2	–	–	–	–	Δ	Δ	Δ	Δ
	Compare Y to Memory (16-Bit)	DIR	9D dd	3								
		EXT	BD hh ll	3								
		IDX	AD xb	3								
		IDX1	AD xb ff	3								
		IDX2	AD xb ee ff	4								
		[D,IDX]	AD xb	6								
		[IDX2]	AD xb ee ff	6								
DAA	Adjust Sum to BCD	INH	18 07	3	–	–	–	–	Δ	Δ	?	Δ
	Decimal Adjust Accumulator A											
DBEQ *cntr, rel*	(cntr) – 1⇒ cntr	REL	04 lb rr	3	–	–	–	–	–	–	–	–
	If (cntr) = 0, then Branch	(9-bit)										
	else Continue to next instruction											
	Decrement Counter and Branch if = 0											
	(cntr = A, B, D, X, Y, or SP)											
DBNE *cntr, rel*	(cntr) – 1 ⇒ cntr	REL	04 lb rr	3	–	–	–	–	–	–	–	–
	If (cntr) not = 0, then Branch;	(9-bit)										
	else Continue to next instruction											
	Decrement Counter and Branch if ≠ 0											
	(cntr = A, B, D, X, Y, or SP)											

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
DEC *opr*	(M) – $01 ⇒ M	EXT	73 hh ll	4	–	–	–	–	Δ	Δ	Δ	–
	Decrement Memory Location	IDX	63 xb	3								
		IDX1	63 xb ff	4								
		IDX2	63 xb ee ff	5								
		[D,IDX]	63 xb	6								
		[IDX2]	63 xb ee ff	6								
DECA	(A) – $01 ⇒ A Decrement A	INH	43	1								
DECB	(B) – $01 ⇒ B Decrement B	INH	53	1								
DES	(SP) – $0001 ⇒ SP	IDX	1B 9F	2	–	–	–	–	–	–	–	–
	Translates to LEAS –1,SP											
DEX	(X) – $0001 ⇒ X	INH	09	1	–	–	–	–	–	Δ	–	–
	Decrement Index Register X											
DEY	(Y) – $0001 ⇒ Y	INH	03	1	–	–	–	–	–	Δ	–	–
	Decrement Index Register Y											
EDIV	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D	INH	11	11	–	–	–	–	Δ	Δ	Δ	Δ
	32 × 16 Bit ⇒ 16 Bit Divide (unsigned)											
EDIVS	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D	INH	18 14	12	–	–	–	–	Δ	Δ	Δ	Δ
	32 × 16 Bit ⇒ 16 Bit Divide (signed)											
EMACS *sum*	$(M_{(X)}:M_{(X+1)}) \times (M_{(Y)}:M_{(Y+1)}) + (M\sim M+3) \Rightarrow M\sim M+3$	Special	18 12 hh ll	13	–	–	–	–	Δ	Δ	Δ	Δ
	16 × 16 Bit ⇒ 32 Bit											
	Multiply and Accumulate (signed)											
EMAXD *opr*	MAX((D), (M:M+1)) ⇒ D	IDX	18 1A xb	4	–	–	–	–	Δ	Δ	Δ	Δ
	MAX of 2 Unsigned 16-Bit Values	IDX1	18 1A xb ff	4								
		IDX2	18 1A xb ee ff	5								
	N, Z, V and C status bits reflect result of	[D,IDX]	18 1A xb	7								
	internal compare ((D) – (M:M+1))	[IDX2]	18 1A xb ee ff	7								
EMAXM *opr*	MAX((D), (M:M+1)) ⇒ M:M+1	IDX	18 1E xb	4	–	–	–	–	Δ	Δ	Δ	Δ
	MAX of 2 Unsigned 16-Bit Values	IDX1	18 1E xb ff	5								
		IDX2	18 1E xb ee ff	6								
	N, Z, V and C status bits reflect result of	[D,IDX]	18 1E xb	7								
	internal compare ((D) – (M:M+1))	[IDX2]	18 1E xb ee ff	7								
EMIND *opr*	MIN((D), (M:M+1)) ⇒ D	IDX	18 1B xb	4	–	–	–	–	Δ	Δ	Δ	Δ
	MIN of 2 Unsigned 16-Bit Values	IDX1	18 1B xb ff	4								
		IDX2	18 1B xb ee ff	5								
	N, Z, V and C status bits reflect result of	[D,IDX]	18 1B xb	7								
	internal compare ((D) – (M:M+1))	[IDX2]	18 1B xb ee ff	7								
EMINM *opr*	MIN((D), (M:M+1)) ⇒ M:M+1	IDX	18 1F xb	4	–	–	–	–	Δ	Δ	Δ	Δ
	MIN of 2 Unsigned 16-Bit Values	IDX1	18 1F xb ff	5								
		IDX2	18 1F xb ee ff	6								
	N, Z, V and C status bits reflect result of	[D,IDX]	18 1F xb	7								
	internal compare ((D) – (M:M+1))	[IDX2]	18 1F xb ee ff	7								
EMUL	(D) × (Y) ⇒ Y:D	INH	13	3	–	–	–	–	Δ	Δ	–	Δ
	16 × 16 Bit Multiply (unsigned)											
EMULS	(D) × (Y) ⇒ Y:D	INH	18 13	3	–	–	–	–	Δ	Δ	–	Δ
	16 × 16 Bit Multiply (signed)											

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
EORA *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
EORB *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
ETBL *opr*	(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 allowed)	IDX	18 3F xb	10	–	–	–	–	Δ	Δ	–	?
EXG *r1, r2*	(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	1	–	–	–	–	–	–	–	–
FDIV	(D) ÷ (X) ⇒ X; r ⇒ D 16 × 16 Bit Fractional Divide	INH	18 11	12	–	–	–	–	–	Δ	Δ	Δ
IBEQ *cntr, rel*	(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	3	–	–	–	–	–	–	–	–
IBNE *cntr, rel*	(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	3	–	–	–	–	–	–	–	–
IDIV	(D) ÷ (X) ⇒ X; r ⇒ D 16 × 16 Bit Integer Divide (unsigned)	INH	18 10	12	–	–	–	–	–	Δ	0	Δ
IDIVS	(D) ÷ (X) ⇒ X; r ⇒ D 16 × 16 Bit Integer Divide (signed)	INH	18 15	12	–	–	–	–	Δ	Δ	Δ	Δ

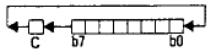
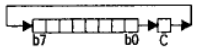
Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
INC *opr*	(M) + $01 ⇒ M Increment Memory Byte	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	72 hh ll 62 xb 62 xb ff 62 xb ee ff 62 xb 62 xb ee ff	4 3 4 5 6 6	–	–	–	–	Δ	Δ	Δ	–
INCA INCB	(A) + $01 ⇒ A Increment Acc. A (B) + $01 ⇒ B Increment Acc. B	INH INH	42 52	1 1								
INS	(SP) + $0001 ⇒ SP *Translates to* LEAS 1,SP	IDX	1B 81	2	–	–	–	–	–	–	–	–
INX	(X) + $0001 ⇒ X Increment Index Register X	INH	08	1	–	–	–	–	–	Δ	–	–
INY	(Y) + $0001 ⇒ Y Increment Index Register Y	INH	02	1	–	–	–	–	–	Δ	–	–
JMP *opr*	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	3 3 3 4 6 6	–	–	–	–	–	–	–	–
JSR *opr*	(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	4 4 4 4 5 7 7	–	–	–	–	–	–	–	–
LBCC *rel*	Long Branch if Carry Clear (if C = 0)	REL	18 24 qq rr	4/3	–	–	–	–	–	–	–	–
LBCS *rel*	Long Branch if Carry Set (if C = 1)	REL	18 25 qq rr	4/3	–	–	–	–	–	–	–	–
LBEQ *rel*	Long Branch if Equal (if Z = 1)	REL	18 27 qq rr	4/3	–	–	–	–	–	–	–	–
LBGE *rel*	Long Branch Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	18 2C qq rr	4/3	–	–	–	–	–	–	–	–
LBGT *rel*	Long Branch if Greater Than (if Z + (N ⊕ V) = 0) (signed)	REL	18 2E qq rr	4/3	–	–	–	–	–	–	–	–
LBHI *rel*	Long Branch if Higher (if C + Z = 0) (unsigned)	REL	18 22 qq rr	4/3	–	–	–	–	–	–	–	–
LBHS *rel*	Long Branch if Higher or Same (if C = 0) (unsigned) same function as LBCC	REL	18 24 qq rr	4/3	–	–	–	–	–	–	–	–
LBLE *rel*	Long Branch if Less Than or Equal (if Z + (N ⊕ V) = 1) (signed)	REL	18 2F qq rr	4/3	–	–	–	–	–	–	–	–
LBLO *rel*	Long Branch if Lower (if C = 1) (unsigned) *same function as LBCS*	REL	18 25 qq rr	4/3	–	–	–	–	–	–	–	–
LBLS *rel*	Long Branch if Lower or Same (if C + Z = 1) (unsigned)	REL	18 23 qq rr	4/3	–	–	–	–	–	–	–	–
LBLT *rel*	Long Branch if Less Than (if N ⊕ V = 1) (signed)	REL	18 2D qq rr	4/3	–	–	–	–	–	–	–	–
LBMI *rel*	Long Branch if Minus (if N = 1)	REL	18 2B qq rr	4/3	–	–	–	–	–	–	–	–
LBNE *rel*	Long Branch if Not Equal (if Z = 0)	REL	18 26 qq rr	4/3	–	–	–	–	–	–	–	–
LBPL *rel*	Long Branch if Plus (if N = 0)	REL	18 2A qq rr	4/3	–	–	–	–	–	–	–	–
LBRA *rel*	Long Branch Always (if 1=1)	REL	18 20 qq rr	4	–	–	–	–	–	–	–	–

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
LBRN *rel*	Long Branch Never (if 1 = 0)	REL	18 21 qq rr	3	–	–	–	–	–	–	–	–
LBVC *rel*	Long Branch if Overflow Bit Clear (if V=0)	REL	18 28 qq rr	4/3	–	–	–	–	–	–	–	–
LBVS *rel*	Long Branch if Overflow Bit Set (if V = 1)	REL	18 29 qq rr	4/3	–	–	–	–	–	–	–	–
LDAA *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
LDAB *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
LDD *opr*	(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	2 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
LDS *opr*	(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	2 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
LDX *opr*	(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	2 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
LDY *opr*	(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	2 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	–
LEAS *opr*	Effective Address ⇒ SP Load Effective Address into SP	IDX IDX1 IDX2	1B xb 1B xb ff 1B xb ee ff	2 2 2	–	–	–	–	–	–	–	–

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
LEAX *opr*	Effective Address ⇒ X Load Effective Address into X	IDX IDX1 IDX2	1A xb 1A xb ff 1A xb ee ff	2 2 2	–	–	–	–	–	–	–	–
LEAY *opr*	Effective Address ⇒ Y Load Effective Address into Y	IDX IDX1 IDX2	19 xb 19 xb ff 19 xb ee ff	2 2 2	–	–	–	–	–	–	–	–
LSL *opr*	 Logical Shift Left same function as ASL	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff	4 3 4 5 6 6	–	–	–	–	Δ	Δ	Δ	Δ
LSLA	Logical Shift Accumulator A to Left	INH	48	1								
LSLB	Logical Shift Accumulator B to Left	INH	58	1								
LSLD	 Logical Shift Left D Accumulator same function as ASLD	INH	59	1	–	–	–	–	Δ	Δ	Δ	Δ
LSR *opr*	 Logical Shift Right	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	74 hh ll 64 xb 64 xb ff 64 xb ee ff 64 xb 64 xb ee ff	4 3 4 5 6 6	–	–	–	–	0	Δ	Δ	Δ
LSRA	Logical Shift Accumulator A to Right	INH	44	1								
LSRB	Logical Shift Accumulator B to Right	INH	54	1								
LSRD	 Logical Shift Right D Accumulator	INH	49	1	–	–	–	–	0	Δ	Δ	Δ
MAXA	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	4 4 5 7 7	–	–	–	–	Δ	Δ	Δ	Δ
MAXM	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	4 5 6 7 7	–	–	–	–	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 instruction details for special cases.	Special	01	5	–	–	?	–	?	?	?	?

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
MINA	MIN((A), (M)) ⇒ A	IDX	18 19 xb	4	–	–	–	–	Δ	Δ	Δ	Δ
	MIN of Two Unsigned 8-Bit Values	IDX1	18 19 xb ff	4								
		IDX2	18 19 xb ee ff	5								
	N, Z, V and C status bits reflect result of	[D,IDX]	18 19 xb	7								
	internal compare ((A) – (M)).	[IDX2]	18 19 xb ee ff	7								
MINM	MIN((A), (M)) ⇒ M	IDX	18 1D xb	4	–	–	–	–	Δ	Δ	Δ	Δ
	MIN of Two Unsigned 8-Bit Values	IDX1	18 1D xb ff	5								
		IDX2	18 1D xb ee ff	6								
	N, Z, V and C status bits reflect result of	[D,IDX]	18 1D xb	7								
	internal compare ((A) – (M)).	[IDX2]	18 1D xb ee ff	7								
MOVB *opr1, opr2*	$(M_1) \Rightarrow M_2$	IMM-EXT	18 0B ii hh ll	4	–	–	–	–	–	–	–	–
	Memory to Memory Byte-Move (8-Bit)	IMM-IDX	18 08 xb ii	4								
		EXT-EXT	18 0C hh ii hh ll	6								
		EXT-IDX	18 09 xb hh ll	5								
		IDX-EXT	18 0D xb hh ll	5								
		IDX-IDX	18 0A xb xb	5								
MOVW *opr1, opr2*	$(M:M+1_1) \Rightarrow M:M+1_2$	IMM-EXT	18 03 jj kk hh ll	5	–	–	–	–	–	–	–	–
	Memory to Memory Word-Move (16-Bit)	IMM-IDX	18 00 xb jj kk	4								
		EXT-EXT	18 04 hh ll hh ll	6								
		EXT-IDX	18 01 xb hh ll	5								
		IDX-EXT	18 05 xb hh ll	5								
		IDX-IDX	18 02 xb xb	5								
MUL	(A) × (B) ⇒ A:B	INH	12	3	–	–	–	–	–	–	–	Δ
	8 × 8 Unsigned Multiply											
NEG *opr*	0 – (M) ⇒ M *or* $(\overline{M}) + 1 \Rightarrow M$	EXT	70 hh ll	4	–	–	–	–	Δ	Δ	Δ	Δ
	Two's Complement Negate	IDX	60 xb	3								
		IDX1	60 xb ff	4								
		IDX2	60 xb ee ff	5								
		[D,IDX]	60 xb	6								
		[IDX2]	60 xb ee ff	6								
NEGA	0 – (A) ⇒ A *equivalent to* $(\overline{A}) + 1 \Rightarrow B$	INH	40	1								
	Negate Accumulator A											
NEGB	0 – (B) ⇒ B *equivalent to* $(\overline{B}) + 1 \Rightarrow B$	INH	50	1								
	Negate Accumulator B											
NOP	No Operation	INH	A7	1	–	–	–	–	–	–	–	–
ORAA *opr*	(A) + (M) ⇒ A	IMM	8A ii	1	–	–	–	–	Δ	Δ	0	–
	Logical OR A with Memory	DIR	9A dd	3								
		EXT	BA hh ll	3								
		IDX	AA xb	3								
		IDX1	AA xb ff	3								
		IDX2	AA xb ee ff	4								
		[D,IDX]	AA xb	6								
		[IDX2]	AA xb ee ff	6								
ORAB *opr*	(B) + (M) ⇒ B	IMM	CA ii	1	–	–	–	–	Δ	Δ	0	–
	Logical OR B with Memory	DIR	DA dd	3								
		EXT	FA hh ll	3								
		IDX	EA xb	3								
		IDX1	EA xb ff	3								
		IDX2	EA xb ee ff	4								
		[D,IDX]	EA xb	6								
		[IDX2]	EA xb ee ff	6								
ORCC *opr*	(CCR) + M ⇒ CCR	IMM	14 ii	1	⇑	–	⇑	⇑	⇑	⇑	⇑	⇑
	Logical OR CCR with Memory											

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
PSHA	(SP) – 1 ⇒ SP; (A) ⇒ $M_{(SP)}$	INH	36	2	–	–	–	–	–	–	–	–
	Push Accumulator A onto Stack											
PSHB	(SP) – 1 ⇒ SP; (B) ⇒ $M_{(SP)}$	INH	37	2	–	–	–	–	–	–	–	–
	Push Accumulator B onto Stack											
PSHC	(SP) – 1 ⇒ SP; (CCR) ⇒ $M_{(SP)}$	INH	39	2	–	–	–	–	–	–	–	–
	Push CCR onto Stack											
PSHD	(SP) – 2 ⇒ SP; (A:B) ⇒ $M_{(SP)}$:$M_{(SP+1)}$	INH	3B	2	–	–	–	–	–	–	–	–
	Push D Accumulator onto Stack											
PSHX	(SP) – 2 ⇒ SP; (X_H:X_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$	INH	34	2	–	–	–	–	–	–	–	–
	Push Index Register X onto Stack											
PSHY	(SP) – 2 ⇒ SP; (Y_H:Y_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$	INH	35	2	–	–	–	–	–	–	–	–
	Push Index Register Y onto Stack											
PULA	($M_{(SP)}$) ⇒ A; (SP) + 1 ⇒ SP	INH	32	3	–	–	–	–	–	–	–	–
	Pull Accumulator A from Stack											
PULB	($M_{(SP)}$) ⇒ B; (SP) + 1 ⇒ SP	INH	33	3	–	–	–	–	–	–	–	–
	Pull Accumulator B from Stack											
PULC	($M_{(SP)}$) ⇒ CCR; (SP) + 1 ⇒ SP	INH	38	3	Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ
	Pull CCR from Stack											
PULD	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ A:B; (SP) + 2 ⇒ SP	INH	3A	3	–	–	–	–	–	–	–	–
	Pull D from Stack											
PULX	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ X_H:X_L; (SP) + 2 ⇒ SP	INH	30	3	–	–	–	–	–	–	–	–
	Pull Index Register X from Stack											
PULY	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ Y_H:Y_L; (SP) + 2 ⇒ SP	INH	31	3	–	–	–	–	–	–	–	–
	Pull Index Register Y from Stack											
REV	MIN-MAX rule evaluation	Special	18 3A	3**	–	–	–	–	–	–	Δ	–
	Find smallest rule input (MIN).			per rule byte								
	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.											

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
REVW	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	3** per rule byte; 5 per wt.	–	–	?	–	?	?	Δ	!
ROL *opr*	 Rotate Memory Left through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	75 hh ll 65 xb 65 xb ff 65 xb ee ff 65 xb 65 xb ee ff	4 3 4 5 6 6	–	–	–	–	Δ	Δ	Δ	Δ
ROLA	Rotate A Left through Carry	INH	45	1								
ROLB	Rotate B Left through Carry	INH	55	1								
ROR *opr*	 Rotate Memory Right through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	76 hh ll 66 xb 66 xb ff 66 xb ee ff 66 xb 66 xb ee ff	4 3 4 5 6 6	–	–	–	–	Δ	Δ	Δ	Δ
RORA	Rotate A Right through Carry	INH	46	1								
RORB	Rotate B Right through Carry	INH	56	1								
RTC	$(M_{(SP)}) \Rightarrow$ PPAGE; (SP) + 1 ⇒ SP; $(M_{(SP)}:M_{(SP+1)}) \Rightarrow PC_H:PC_L$; (SP) + 2 ⇒ SP Return from Call	INH	0A	6	–	–	–	–	–	–	–	–
RTI	$(M_{(SP)}) \Rightarrow$ 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 Return from Interrupt	INH	0B	8	Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ
RTS	$(M_{(SP)}:M_{(SP+1)}) \Rightarrow PC_H:PC_L$; (SP) + 2 ⇒ SP Return from Subroutine	INH	3D	5	–	–	–	–	–	–	–	–
SBA	(A) – (B) ⇒ A Subtract B from A	INH	18 16	2	–	–	–	–	Δ	Δ	Δ	Δ

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
SBCA *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	Δ	Δ
SBCB *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	Δ	Δ
SEC	1 ⇒ C *Translates to* ORCC #$01	IMM	14 01	1	–	–	–	–	–	–	–	1
SEI	1 ⇒ I; (inhibit I interrupts) *Translates to* ORCC #$10	IMM	14 10	1	–	–	–	1	–	–	–	–
SEV	1 ⇒ V *Translates to* ORCC #$02	IMM	14 02	1	–	–	–	–	–	–	1	–
SEX *r1, r2*	$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	1	–	–	–	–	–	–	–	–
STAA *opr*	(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	2 3 2 3 3 5 5	–	–	–	–	Δ	Δ	0	–
STAB *opr*	(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	2 3 2 3 3 5 5	–	–	–	–	Δ	Δ	0	–
STD *opr*	(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	2 3 2 3 3 5 5	–	–	–	–	Δ	Δ	0	–

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
STOP	(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	9** +5 or +2**	–	–	–	–	–	–	–	–
STS *opr*	(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	2 3 2 3 3 5 5	–	–	–	–	Δ	Δ	0	–
STX *opr*	(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	2 3 2 3 3 5 5	–	–	–	–	Δ	Δ	0	–
STY *opr*	(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	2 3 2 3 3 5 5	–	–	–	–	Δ	Δ	0	–
SUBA *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	Δ	Δ
SUBB *opr*	(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	1 3 3 3 3 4 6 6	–	–	–	–	Δ	Δ	Δ	Δ

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
SUBD *opr*	(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	2 3 3 3 3 4 6 6	–	–	–	–	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	9	–	–	–	1	–	–	–	–
TAB	(A) ⇒ B Transfer A to B	INH	18 0E	2	–	–	–	–	Δ	Δ	0	–
TAP	(A) ⇒ CCR *Translates to* TFR A , CCR	INH	B7 02	1	Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ
TBA	(B) ⇒ A Transfer B to A	INH	18 0F	2	–	–	–	–	Δ	Δ	0	–
TBEQ *cntr, rel*	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	3	–	–	–	–	–	–	–	–
TBL *opr*	(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 allowed.)	IDX	18 3D xb	8	–	–	–	–	Δ	Δ	–	?
TBNE *cntr, rel*	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	3	–	–	–	–	–	–	–	–
TFR *r1, r2*	(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	1	– or Δ	– ⇓	– Δ	– Δ	– Δ	– Δ	– Δ	– Δ
TPA	(CCR) ⇒ A *Translates to* TFR CCR , A	INH	B7 20	1	–	–	–	–	–	–	–	–

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
TRAP	(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	10	–	–	–	1	–	–	–	–
TST *opr*	(M) – 0 Test Memory for Zero or Minus	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	F7 hh ll E7 xb E7 xb ff E7 xb ee ff E7 xb E7 xb ee ff	3 3 3 4 6 6	–	–	–	–	Δ	Δ	0	0
TSTA	(A) – 0 Test A for Zero or Minus	INH	97	1								
TSTB	(B) – 0 Test B for Zero or Minus	INH	D7	1								
TSX	(SP) ⇒ X *Translates to* TFR SP,X	INH	B7 75	1	–	–	–	–	–	–	–	–
TSY	(SP) ⇒ Y *Translates to* TFR SP,Y	INH	B7 76	1	–	–	–	–	–	–	–	–
TXS	(X) ⇒ SP *Translates to* TFR X,SP	INH	B7 57	1	–	–	–	–	–	–	–	–
TYS	(Y) ⇒ SP *Translates to* TFR Y,SP	INH	B7 67	1	–	–	–	–	–	–	–	–
WAI	(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	8** (in) + 5 (int)	– or – or –	– – 1	– – –	– 1 1	– – –	– – –	– – –	– – –
WAV	$\sum_{i=1}^{B} S_i F_i \Rightarrow Y{:}D$ $\sum_{i=1}^{B} F_i \Rightarrow X$ 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	8** per lable	–	–	?	–	?	Δ	?	?

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
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	1	–	–	–	–	–	–	–	–
XGDY	(D) ⇔ (Y) *Translates to* EXG D, Y	INH	B7 C6	1	–	–	–	–	–	–	–	–

NOTES:

*Each cycle (~) is typically 125 ns for an 8-MHz bus (16-MHz oscillator).

**Refer to detailed instruction descriptions for additional information.