Notation Used in Instruction Set Summary

CPU Register Notation

Accumulator A — A or a	Index Register Y — Y or y
Accumulator B — B or b	Stack Pointer — SP, sp, or s
Accumulator D — D or d	Program Counter — PC, pc, or p
Index Register X — X or x	Condition Code Register — CCR or c

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 integer in the range –16 . . . +15
oprx9 —	Any integer in the range –256 . . . +255
oprx16 —	Any integer 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 integer in the range $30-$39 or $40-$FF
xys —	X or Y or SP
xysp —	X or Y or SP or PC

Operators

+ —	Addition
– —	Subtraction
• —	Logical AND
+ —	Logical OR (inclusive)
⊕ —	Logical exclusive OR
× —	Multiplication
÷ —	Division
$\overline{M}$ —	Negation. One's complement (invert each bit of M)
: —	Concatenate Example: A : B means the 16-bit value formed by concatenating 8-bit accumulator A with 8-bit accumulator B. A is in the high-order position.

Operators (continued)

⇒ —	Transfer Example: (A) ⇒ M means the content of accumulator A is transferred to memory location M.
⇔ —	Exchange Example: D ⇔ X means exchange the contents of D with those of X.

Address Mode Notation

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

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.

Access Detail

Each code letter except (,), and comma equals one CPU cycle.Uppercase = 16-bit operation and lowercase = 8-bit operation.

`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 only)
`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); Page 2 prebyte treated as a separate 1-byte instruction
`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 (always an aligned-word read)
`t`	— 8-bit conditional read (or free cycle)
`T`	— 16-bit conditional read (or free cycle)
`x`	— 8-bit conditional write (or free cycle)
`()`	— Indicate a microcode loop
`,`	— Indicates where an interrupt could be honored

Special Cases

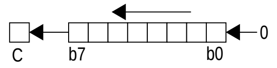
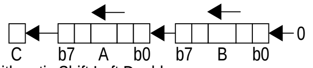
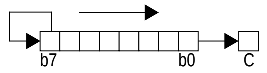
`PPP/P`	— Short branch, PPP if branch taken, P if not
`OPPP/OPO`	— Long branch, OPPP if branch taken, OPO if not

Condition Codes Columns

– —	Status bit not affected by operation.
0 —	Status bit cleared by operation.
1 —	Status bit set by operation.
∆ —	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)	Access Detail HCS12	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	Pf	– – – –	– – – –
ABY	(B) + (Y) ⇒ Y *Translates to* LEAY B,Y	IDX	19 ED	Pf	– – – –	– – – –
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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – 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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – 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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – 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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – 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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – –	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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – –	Δ Δ 0 –
ANDCC #*opr8i*	(CCR) • (M) ⇒ CCR Logical AND CCR with Memory	IMM	10 ii	P	⇓ ⇓ ⇓ ⇓	⇓ ⇓ ⇓ ⇓

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

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	S X H I	N Z V C
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	rPwO rPw rPwO frPwP 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	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – –	Δ Δ Δ Δ
BCC *rel8*	Branch if Carry Clear (if C = 0)	REL	24 rr	PPP/P[1]	– – – –	– – – –
BCLR *opr8a, msk8* BCLR *opr16a, msk8* BCLR *oprx0_xysp, msk8* BCLR *oprx9,xysp, msk8* BCLR *oprx16,xysp, msk8*	(M) • ($\overline{\text{mm}}$) ⇒ 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	rPwO rPwP rPwO rPwP frPwPO	– – – –	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 Does not change Accumulator or 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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – –	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 Does not change Accumulator or 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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – –	Δ Δ 0 –
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]	– – – –	– – – –

Note 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.

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	S X H I	N Z V C
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 rfPPP PrfPPP	– – – –	– – – –
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 rfPPP PrfPPP	– – – –	– – – –
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	rPwO rPwP rPwO rPwP frPwPO	– – – –	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	SPPP	– – – –	– – – –
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	gnSsPPP gnSsPPP gnSsPPP fgnSsPPP 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	– – – –
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	PwO Pw PwO PwP PIfw PIPw O O	– – – –	0 1 0 0
CLV	0 ⇒ V *Translates to* ANDCC #$FD	IMM	10 FD	P	– – – –	– – 0 –

Note 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.

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	S X H I	N Z V C
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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – –	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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – –	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	rPwO rPw rPwO frPwP 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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	Δ Δ Δ Δ
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 (branch) PPO (no branch)	– – – –	– – – –

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	S X H I	N Z V C
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` (branch) `PPO` (no branch)	– – – –	– – – –

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`
				`rPwO`
				`rPw`
				`rPwO`
				`frPwP`
				`fIfrPw`
				`fIPrPw`
				`O`
				`O`
					– – – –	Δ Δ Δ –

DES	(SP) – $0001 ⇒ SP *Translates to* LEAS –1,SP	IDX	`1B 9F`	`Pf`	– – – –	– – – –
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 by 16 Bit ⇒ 16 Bit Divide (unsigned)	INH	`11`	`ffffffffffO`	– – – –	Δ Δ Δ Δ
EDIVS	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D 32 by 16 Bit ⇒ 16 Bit Divide (signed)	INH	`18 14`	`OffffffffffO`	– – – –	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 by 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`
				`ORPf`
				`ORPO`
				`OfRPP`
				`OfIfRPf`
				`OfIPRPf`
					– – – –	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`
				`ORPf`
				`ORPO`
				`OfRPP`
				`OfIfRPf`
				`OfIPRPf`
					– – – –	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 by 16 Bit Multiply (unsigned)	INH	`13`	`ffO`	– – – –	Δ Δ – Δ

EMULS	(D) × (Y) ⇒ Y:D
	16 by 16 Bit Multiply (signed)
		INH	`18 13`	`OfO`
				(if followed by page 2 instruction)
				`OffO`
					– – – –	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`
				`rPf`
				`rPO`
				`rPf`
				`rPO`
				`frPP`
				`fIfrPf`
				`fIPrPf`
					– – – –	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`
				`rPf`
				`rPO`
				`rPf`
				`rPO`
				`frPP`
				`fIfrPf`
				`fIPrPf`
					– – – –	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`	`ORRffffffP`	– – – –	Δ Δ – Δ ? C Bit is undefined in HC12
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 by 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` (branch) `PPO` (no branch)	– – – –	– – – –
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` (branch) `PPO` (no branch)	– – – –	– – – –
IDIV	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 by 16 Bit Integer Divide (unsigned)	INH	`18 10`	`OffffffffffO`	– – – –	– Δ 0 Δ
IDIVS	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 by 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`
				`rPwO`
				`rPw`
				`rPwO`
				`frPwP`
				`fIfrPw`
				`fIPrPw`
				`O`
				`O`
					– – – –	Δ Δ Δ –

INS	(SP) + $0001 ⇒ SP *Translates to* LEAS 1,SP	IDX	`1B 81`	`Pf`	– – – –	– – – –
INX	(X) + $0001 ⇒ X Increment Index Register X	INH	`08`	`O`	– – – –	– Δ – –
INY	(Y) + $0001 ⇒ Y Increment Index Register Y	INH	`02`	`O`	– – – –	– Δ – –

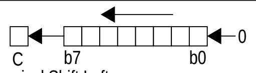
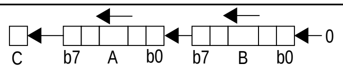
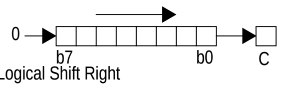
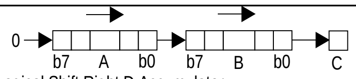
JMP *opr16a*
JMP *oprx0_xysp*
JMP *oprx9,xysp*
JMP *oprx16,xysp*
JMP [D,*xysp*]
JMP [*oprx16,xysp*]
	Routine address ⇒ PC
	
	Jump
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			`06 hh ll`
			`05 xb`
			`05 xb ff`
			`05 xb ee ff`
			`05 xb`
			`05 xb ee ff`
				`PPP`
				`PPP`
				`PPP`
				`fPPP`
				`fIfPPP`
				`fIfPPP`
					– – – –	– – – –

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.

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

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	S X H I	N Z V C
JSR *opr8a* JSR *opr16a* JSR *oprx0_xysp* JSR *oprx9,xysp* JSR *oprx16,xysp* JSR [D,*xysp*] JSR [*oprx16,xysp*]	(SP) – 2 ⇒ SP; $RTN_H:RTN_L \Rightarrow M_{(SP)}:M_{(SP+1)}$; 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	SPPP SPPP 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 $N \oplus V = 0$) (signed)	REL	18 2C qq rr	OPPP/OPO^1	– – – –	– – – –
LBGT *rel16*	Long Branch if Greater Than (if $Z + (N \oplus V) = 0$) (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 $Z + (N \oplus V) = 1$) (signed)	REL	18 2F qq rr	OPPP/OPO^1	– – – –	– – – –
LBLO *rel16*	Long Branch if Lower (if C = 1) (unsigned) *same function as LBC*S	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 $N \oplus V = 1$) (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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – –	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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – –	Δ Δ 0 –
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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	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.

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	S X H I	N Z V C
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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	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	Pf 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	Pf 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	Pf 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	rPwO rPw rPwO 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	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	– – – –	0 Δ Δ Δ
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	OrPf OrPO OfrPP OfIfrPf OfIPrPf	– – – –	Δ Δ Δ Δ

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

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	S X H I	N Z V C

MAXM *oprx0_xysp*
MAXM *oprx9,xysp*
MAXM *oprx16,xysp*
MAXM [D,*xysp*]
MAXM [*oprx16,xysp*]
	MAX((A), (M)) $\Rightarrow$ 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) $\Rightarrow M_{(Y)}$; (X) + 4 $\Rightarrow$ X; (Y) + 1 $\Rightarrow$ 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)) $\Rightarrow$ 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`
				`OrPf`
				`OrPO`
				`OfrPP`
				`OfIfrPf`
				`OfIPrPf`
					– – – –	Δ Δ Δ Δ

MINM *oprx0_xysp*
MINM *oprx9,xysp*
MINM *oprx16,xysp*
MINM [D,*xysp*]
MINM [*oprx16,xysp*]
	MIN((A), (M)) $\Rightarrow$ 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) $\Rightarrow$ A:B 8 by 8 Unsigned Multiply	INH	`12`	`O`	– – – –	– – – Δ

NEG *opr16a*
NEG *oprx0_xysp*
NEG *oprx9,xysp*
NEG *oprx16,xysp*
NEG [D,*xysp*]
NEG [*oprx16,xysp*]
NEGA
 	
NEGB
	0 – (M) $\Rightarrow$ M *equivalent to* $(\overline{M})$ + 1 $\Rightarrow$ M
	Two's Complement Negate
	
	0 – (A) $\Rightarrow$ A *equivalent to* $(\overline{A})$ + 1 $\Rightarrow$ A
	Negate Accumulator A
	0 – (B) $\Rightarrow$ 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`
				`rPwO`
				`rPw`
				`rPwO`
				`frPwP`
				`fIfrPw`
				`fIPrPw`
				`O`
				
				`O`
					– – – –	Δ Δ Δ Δ

NOP	No Operation	INH	`A7`	`O`	– – – –	– – – –

ORAA #*opr8i*
ORAA *opr8a*
ORAA *opr16a*
ORAA *oprx0_xysp*
ORAA *oprx9,xysp*
ORAA *oprx16,xysp*
ORAA [D,*xysp*]
ORAA [*oprx16,xysp*]
	(A) + (M) $\Rightarrow$ 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`
				`rPf`
				`rPO`
				`rPf`
				`rPO`
				`frPP`
				`fIfrPf`
				`fIPrPf`
					– – – –	Δ Δ 0 –

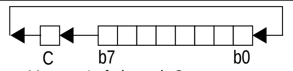
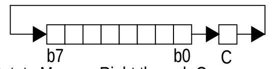
ORAB #*opr8i*
ORAB *opr8a*
ORAB *opr16a*
ORAB *oprx0_xysp*
ORAB *oprx9,xysp*
ORAB *oprx16,xysp*
ORAB [D,*xysp*]
ORAB [*oprx16,xysp*]
	(B) + (M) $\Rightarrow$ B
	Logical OR B with Memory
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			`CA ii`
			`DA dd`
			`FA hh ll`
			`EA xb`
			`EA xb ff`
			`EA xb ee ff`
			`EA xb`
			`EA xb ee ff`
				`P`
				`rPf`
				`rPO`
				`rPf`
				`rPO`
				`frPP`
				`fIfrPf`
				`fIPrPf`
					– – – –	Δ Δ 0 –

ORCC *#opr8i*	(CCR) + M $\Rightarrow$ CCR Logical OR CCR with Memory	IMM	`14 ii`	`P`	⇑ – ⇑ ⇑	⇑ ⇑ ⇑ ⇑
PSHA	(SP) – 1 $\Rightarrow$ SP; (A) $\Rightarrow M_{(SP)}$ Push Accumulator A onto Stack	INH	`36`	`Os`	– – – –	– – – –
PSHB	(SP) – 1 $\Rightarrow$ SP; (B) $\Rightarrow M_{(SP)}$ Push Accumulator B onto Stack	INH	`37`	`Os`	– – – –	– – – –
PSHC	(SP) – 1 $\Rightarrow$ SP; (CCR) $\Rightarrow M_{(SP)}$ Push CCR onto Stack	INH	`39`	`Os`	– – – –	– – – –
PSHD	(SP) – 2 $\Rightarrow$ SP; (A:B) $\Rightarrow M_{(SP)}{:}M_{(SP+1)}$ Push D Accumulator onto Stack	INH	`3B`	`OS`	– – – –	– – – –
PSHX	(SP) – 2 $\Rightarrow$ SP; $(X_H{:}X_L) \Rightarrow M_{(SP)}{:}M_{(SP+1)}$ Push Index Register X onto Stack	INH	`34`	`OS`	– – – –	– – – –
PSHY	(SP) – 2 $\Rightarrow$ SP; $(Y_H{:}Y_L) \Rightarrow M_{(SP)}{:}M_{(SP+1)}$ Push Index Register Y onto Stack	INH	`35`	`OS`	– – – –	– – – –
PULA	$(M_{(SP)}) \Rightarrow$ A; (SP) + 1 $\Rightarrow$ SP Pull Accumulator A from Stack	INH	`32`	`ufO`	– – – –	– – – –
PULB	$(M_{(SP)}) \Rightarrow$ B; (SP) + 1 $\Rightarrow$ SP Pull Accumulator B from Stack	INH	`33`	`ufO`	– – – –	– – – –
PULC	$(M_{(SP)}) \Rightarrow$ CCR; (SP) + 1 $\Rightarrow$ SP Pull CCR from Stack	INH	`38`	`ufO`	Δ ⇓ Δ Δ	Δ Δ Δ Δ
PULD	$(M_{(SP)}{:}M_{(SP+1)}) \Rightarrow$ A:B; (SP) + 2 $\Rightarrow$ SP Pull D from Stack	INH	`3A`	`UfO`	– – – –	– – – –
PULX	$(M_{(SP)}{:}M_{(SP+1)}) \Rightarrow X_H{:}X_L$; (SP) + 2 $\Rightarrow$ SP Pull Index Register X from Stack	INH	`30`	`UfO`	– – – –	– – – –
PULY	$(M_{(SP)}{:}M_{(SP+1)}) \Rightarrow Y_H{:}Y_L$; (SP) + 2 $\Rightarrow$ SP Pull Index Register Y from Stack	INH	`31`	`UfO`	– – – –	– – – –

REV	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(t,tx)O`
				(exit + re-entry replaces comma above if interrupted)
				`ff + Orf(t,`
					– – ? –	? ? Δ ?

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`	`ORf(t,Tx)O`
				(loop to read weight if enabled)
				`(r,RfRf)`
				(exit + re-entry replaces comma above if interrupted)
				`ffff + ORf(t,`
					– – ? –	? ? Δ !

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

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	S X H I	N Z V C
ROL *opr16a*		EXT	`75 hh ll`	`rPwO`	– – – –	Δ Δ Δ Δ
ROL *oprx0_xysp*		IDX	`65 xb`	`rPw`		
ROL *oprx9,xysp*		IDX1	`65 xb ff`	`rPwO`		
ROL *oprx16,xysp*	Rotate Memory Left through Carry	IDX2	`65 xb ee ff`	`frPwP`		
ROL [D,*xysp*]		[D,IDX]	`65 xb`	`fIfrPw`		
ROL [*oprx16,xysp*]		[IDX2]	`65 xb ee ff`	`fIPrPw`		
ROLA	Rotate A Left through Carry	INH	`45`	`O`		
ROLB	Rotate B Left through Carry	INH	`55`	`O`		
ROR *opr16a*		EXT	`76 hh ll`	`rPwO`	– – – –	Δ Δ Δ Δ
ROR *oprx0_xysp*		IDX	`66 xb`	`rPw`		
ROR *oprx9,xysp*		IDX1	`66 xb ff`	`rPwO`		
ROR *oprx16,xysp*	Rotate Memory Right through Carry	IDX2	`66 xb ee ff`	`frPwP`		
ROR [D,*xysp*]		[D,IDX]	`66 xb`	`fIfrPw`		
ROR [*oprx16,xysp*]		[IDX2]	`66 xb ee ff`	`fIPrPw`		
RORA	Rotate A Right through Carry	INH	`46`	`O`		
RORB	Rotate B Right through Carry	INH	`56`	`O`		
RTC	$(M_{(SP)}) \Rightarrow$ PPAGE; $(SP) + 1 \Rightarrow SP$;	INH	`0A`	`uUnfPPP`	– – – –	– – – –
	$(M_{(SP)}:M_{(SP+1)}) \Rightarrow PC_H:PC_L$;					
	$(SP) + 2 \Rightarrow SP$					
	Return from Call					
RTI	$(M_{(SP)}) \Rightarrow$ CCR; $(SP) + 1 \Rightarrow SP$	INH	`0B`	`uUUUUPPP`	Δ ⇓ Δ Δ	Δ Δ Δ Δ
	$(M_{(SP)}:M_{(SP+1)}) \Rightarrow B:A$; $(SP) + 2 \Rightarrow SP$			(with interrupt pending)		
	$(M_{(SP)}:M_{(SP+1)}) \Rightarrow X_H:X_L$; $(SP) + 4 \Rightarrow SP$			`uUUUUVfPPP`		
	$(M_{(SP)}:M_{(SP+1)}) \Rightarrow PC_H:PC_L$; $(SP) – 2 \Rightarrow SP$					
	$(M_{(SP)}:M_{(SP+1)}) \Rightarrow Y_H:Y_L$; $(SP) + 4 \Rightarrow SP$					
	Return from Interrupt					
RTS	$(M_{(SP)}:M_{(SP+1)}) \Rightarrow PC_H:PC_L$;	INH	`3D`	`UfPPP`	– – – –	– – – –
	$(SP) + 2 \Rightarrow SP$					
	Return from Subroutine					
SBA	$(A) – (B) \Rightarrow A$	INH	`18 16`	`OO`	– – – –	Δ Δ Δ Δ
	Subtract B from A					
SBCA #*opr8i*	$(A) – (M) – C \Rightarrow A$	IMM	`82 ii`	`P`	– – – –	Δ Δ Δ Δ
SBCA *opr8a*	Subtract with Borrow from A	DIR	`92 dd`	`rPf`		
SBCA *opr16a*		EXT	`B2 hh ll`	`rPO`		
SBCA *oprx0_xysp*		IDX	`A2 xb`	`rPf`		
SBCA *oprx9,xysp*		IDX1	`A2 xb ff`	`rPO`		
SBCA *oprx16,xysp*		IDX2	`A2 xb ee ff`	`frPP`		
SBCA [D,*xysp*]		[D,IDX]	`A2 xb`	`fIfrPf`		
SBCA [*oprx16,xysp*]		[IDX2]	`A2 xb ee ff`	`fIPrPf`		
SBCB #*opr8i*	$(B) – (M) – C \Rightarrow B$	IMM	`C2 ii`	`P`	– – – –	Δ Δ Δ Δ
SBCB *opr8a*	Subtract with Borrow from B	DIR	`D2 dd`	`rPf`		
SBCB *opr16a*		EXT	`F2 hh ll`	`rPO`		
SBCB *oprx0_xysp*		IDX	`E2 xb`	`rPf`		
SBCB *oprx9,xysp*		IDX1	`E2 xb ff`	`rPO`		
SBCB *oprx16,xysp*		IDX2	`E2 xb ee ff`	`frPP`		
SBCB [D,*xysp*]		[D,IDX]	`E2 xb`	`fIfrPf`		
SBCB [*oprx16,xysp*]		[IDX2]	`E2 xb ee ff`	`fIPrPf`		
SEC	$1 \Rightarrow C$	IMM	`14 01`	`P`	– – – –	– – – 1
	Translates to ORCC #$01					
SEI	$1 \Rightarrow I$; (inhibit I interrupts)	IMM	`14 10`	`P`	– – – 1	– – – –
	Translates to ORCC #$10					
SEV	$1 \Rightarrow V$	IMM	`14 02`	`P`	– – – –	– – 1 –
	Translates to ORCC #$02					
SEX *abc,dxys*	$00:(r1) ⇒ r2 if r1, bit 7 is 0 *or*	INH	`B7 eb`	`P`	– – – –	– – – –
	$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					
STAA *opr8a*	$(A) \Rightarrow M$	DIR	`5A dd`	`Pw`	– – – –	Δ Δ 0 –
STAA *opr16a*	Store Accumulator A to Memory	EXT	`7A hh ll`	`PwO`		
STAA *oprx0_xysp*		IDX	`6A xb`	`Pw`		
STAA *oprx9,xysp*		IDX1	`6A xb ff`	`PwO`		
STAA *oprx16,xysp*		IDX2	`6A xb ee ff`	`PwP`		
STAA [D,*xysp*]		[D,IDX]	`6A xb`	`PIfw`		
STAA [*oprx16,xysp*]		[IDX2]	`6A xb ee ff`	`PIPw`		
STAB *opr8a*	$(B) \Rightarrow M$	DIR	`5B dd`	`Pw`	– – – –	Δ Δ 0 –
STAB *opr16a*	Store Accumulator B to Memory	EXT	`7B hh ll`	`PwO`		
STAB *oprx0_xysp*		IDX	`6B xb`	`Pw`		
STAB *oprx9,xysp*		IDX1	`6B xb ff`	`PwO`		
STAB *oprx16,xysp*		IDX2	`6B xb ee ff`	`PwP`		
STAB [D,*xysp*]		[D,IDX]	`6B xb`	`PIfw`		
STAB [*oprx16,xysp*]		[IDX2]	`6B xb ee ff`	`PIPw`		
STD *opr8a*	$(A) \Rightarrow M$, $(B) \Rightarrow M+1$	DIR	`5C dd`	`PW`	– – – –	Δ Δ 0 –
STD *opr16a*	Store Double Accumulator	EXT	`7C hh ll`	`PWO`		
STD *oprx0_xysp*		IDX	`6C xb`	`PW`		
STD *oprx9,xysp*		IDX1	`6C xb ff`	`PWO`		
STD *oprx16,xysp*		IDX2	`6C xb ee ff`	`PWP`		
STD [D,*xysp*]		[D,IDX]	`6C xb`	`PIfW`		
STD [*oprx16,xysp*]		[IDX2]	`6C xb ee ff`	`PIPW`		
STOP	$(SP) – 2 \Rightarrow SP$;	INH	`18 3E`	(entering STOP)	– – – –	– – – –
	$RTN_H:RTN_L \Rightarrow M_{(SP)}:M_{(SP+1)}$;			`OOSSSSsf`		
	$(SP) – 2 \Rightarrow SP$; $(Y_H:Y_L) \Rightarrow M_{(SP)}:M_{(SP+1)}$;			(exiting STOP)		
	$(SP) – 2 \Rightarrow SP$; $(X_H:X_L) \Rightarrow M_{(SP)}:M_{(SP+1)}$;			`fVfPPP`		
	$(SP) – 2 \Rightarrow SP$; $(B:A) \Rightarrow M_{(SP)}:M_{(SP+1)}$;					
	$(SP) – 1 \Rightarrow SP$; $(CCR) \Rightarrow M_{(SP)}$;			(continue)		
	STOP All Clocks			`ff`		
	Registers stacked to allow quicker recovery by interrupt.					
				(if STOP disabled)		
	If S control bit = 1, the STOP instruction is disabled and acts like a two-cycle NOP.			`OO`		
STS *opr8a*	$(SP_H:SP_L) \Rightarrow M:M+1$	DIR	`5F dd`	`PW`	– – – –	Δ Δ 0 –
STS *opr16a*	Store Stack Pointer	EXT	`7F hh ll`	`PWO`		
STS *oprx0_xysp*		IDX	`6F xb`	`PW`		
STS *oprx9,xysp*		IDX1	`6F xb ff`	`PWO`		
STS *oprx16,xysp*		IDX2	`6F xb ee ff`	`PWP`		
STS [D,*xysp*]		[D,IDX]	`6F xb`	`PIfW`		
STS [*oprx16,xysp*]		[IDX2]	`6F xb ee ff`	`PIPW`		
STX *opr8a*	$(X_H:X_L) \Rightarrow M:M+1$	DIR	`5E dd`	`PW`	– – – –	Δ Δ 0 –
STX *opr16a*	Store Index Register X	EXT	`7E hh ll`	`PWO`		
STX *oprx0_xysp*		IDX	`6E xb`	`PW`		
STX *oprx9,xysp*		IDX1	`6E xb ff`	`PWO`		
STX *oprx16,xysp*		IDX2	`6E xb ee ff`	`PWP`		
STX [D,*xysp*]		[D,IDX]	`6E xb`	`PIfW`		
STX [*oprx16,xysp*]		[IDX2]	`6E xb ee ff`	`PIPW`		
STY *opr8a*	$(Y_H:Y_L) \Rightarrow M:M+1$	DIR	`5D dd`	`PW`	– – – –	Δ Δ 0 –
STY *opr16a*	Store Index Register Y	EXT	`7D hh ll`	`PWO`		
STY *oprx0_xysp*		IDX	`6D xb`	`PW`		
STY *oprx9,xysp*		IDX1	`6D xb ff`	`PWO`		
STY *oprx16,xysp*		IDX2	`6D xb ee ff`	`PWP`		
STY [D,*xysp*]		[D,IDX]	`6D xb`	`PIfW`		
STY [*oprx16,xysp*]		[IDX2]	`6D xb ee ff`	`PIPW`		
SUBA #*opr8i*	$(A) – (M) \Rightarrow A$	IMM	`80 ii`	`P`	– – – –	Δ Δ Δ Δ
SUBA *opr8a*	Subtract Memory from Accumulator A	DIR	`90 dd`	`rPf`		
SUBA *opr16a*		EXT	`B0 hh ll`	`rPO`		
SUBA *oprx0_xysp*		IDX	`A0 xb`	`rPf`		
SUBA *oprx9,xysp*		IDX1	`A0 xb ff`	`rPO`		
SUBA *oprx16,xysp*		IDX2	`A0 xb ee ff`	`frPP`		
SUBA [D,*xysp*]		[D,IDX]	`A0 xb`	`fIfrPf`		
SUBA [*oprx16,xysp*]		[IDX2]	`A0 xb ee ff`	`fIPrPf`		

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	S X H I	N Z V C
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 rPf rPO rPf rPO frPP fIfrPf fIPrPf	– – – –	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	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	– – – –	Δ Δ Δ Δ
SWI	(SP) – 2 ⇒ SP; $RTN_H:RTN_L \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 2 ⇒ SP; $(Y_H:Y_L) \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 2 ⇒ SP; $(X_H:X_L) \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 2 ⇒ SP; $(B:A) \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 1 ⇒ SP; $(CCR) \Rightarrow M_{(SP)}$ 1 ⇒ I; (SWI Vector) ⇒ PC Software Interrupt	INH	3F	VSPSSPSsP* (for Reset) VfPPP	– – – 1 1 1 – 1	– – – – – – – –
*The CPU also uses the SWI microcode sequence for hardware interrupts and unimplemented opcode traps. Reset uses the VfPPP variation of this sequence.						
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 –
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 (branch) PPO (no branch)	– – – –	– – – –
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	ORfffP	– – – –	Δ Δ – Δ ? C Bit is undefined in HC12
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 (branch) PPO (no branch)	– – – –	– – – –
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; $RTN_H:RTN_L \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 2 ⇒ SP; $(Y_H:Y_L) \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 2 ⇒ SP; $(X_H:X_L) \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 2 ⇒ SP; $(B:A) \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 1 ⇒ SP; $(CCR) \Rightarrow M_{(SP)}$ 1 ⇒ I; (TRAP Vector) ⇒ PC Unimplemented opcode trap	INH	18 tn tn = $30–$39 or $40–$FF	OVSPSSPSsP	– – – 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	rPO rPf rPO frPP fIfrPf fIPrPf 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	(SP) – 2 ⇒ SP; $RTN_H:RTN_L \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 2 ⇒ SP; $(Y_H:Y_L) \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 2 ⇒ SP; $(X_H:X_L) \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 2 ⇒ SP; $(B:A) \Rightarrow M_{(SP)}:M_{(SP+1)}$; (SP) – 1 ⇒ SP; $(CCR) \Rightarrow M_{(SP)}$; WAIT for interrupt	INH	3E	OSSSSsf (after interrupt) fVfPPP	– – – – or – – – 1 or – 1 – 1	– – – – – – – – – – – –
WAV	$\sum_{i=1}^{B} S_iF_i \Rightarrow Y:D$ and $\sum_{i=1}^{B} F_i \Rightarrow X$ 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	Of(frr,ffff)O (add if interrupt) 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	UUUrr,ffff (frr,ffff)O (exit + re-entry replaces comma above if interrupted) SSS + UUUrr,	– – ? –	? 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	– – – –	– – – –