68HC12 INSTRUCTION LIST (reduced)

Loads, Stores, and Transfers

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Clear Memory Byte	CLR			X	X	X		m(ea) <= 0
Clear Accumulator A (B)	CLRA (B)						X	A <= 0
Load Accumulator A (B)	LDAA (B)	X	X	X	X	X		A <= [m(ea)]
Load Double Accumulator D	LDD	X	X	X	X	X		D <= [m(ea, ea+1)]
Load Effective Address into SP (X or Y)	LEAS (A,B)							SP <= ea
Store Accumulator A (B)	STAA (B)	X	X	X	X	X		m(ea) <= (A)
Store Double Accumulator D	STD	X	X	X	X	X		m(ea, ea+1) <= D
Transfer A to B	TAB						X	B <= (A)
Transfer A to CCR	TAP						X	CCR <= (A)
Transfer B to A	TBA						X	A <= B
Transfer CCR to A	TPA						X	A <= (CCR)
Exchange D with X (Y)	XGDX						X	D <=> (X)
Pull A (B) from Stack	PULA(B)						X	A <= [m(SP)], SP <= (SP)+1
Push A (B) onto Stack	PSHA(B)						X	SP <= (SP)-1, m(SP) <= A

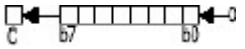
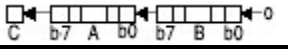
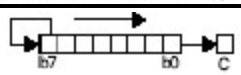
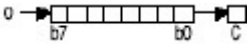
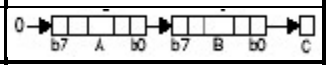
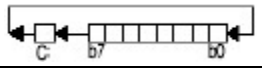
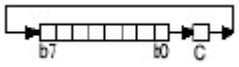
Arithmetic Operations

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Add Accumulators	ABA						X	A <= (A) + (B)
Add with Carry to A (B)	ADCA (B)	X	X	X	X	X		A <= (A) + [m(ea)]+(C)
Add Memory to A (B)	ADDA (B)	X	X	X	X	X		A <= (A) + [m(ea)]
Add Memory to D (16 Bit)	ADDD	X	X	X	X	X		D <= (D) + [m(ea,ea+1)]
Decrement Memory Byte	DEC			X	X	X		m(ea) <= [m(ea)] – 1
Decrement Accumulator A (B)	DECA (B)						X	A <= (A) – 1
Increment Memory Byte	INC			X	X	X		m(ea) <= [m(ea)] + 1
Increment Accumulator A (B)	INCA (B)						X	A <= (A) + 1
Subtract with Carry from A (B)	SBCA (B)	X	X	X	X	X		A <= (A) – [m(ea)] – C
Subtract Memory from A (B)	SUBA (B)	X	X	X	X	X		A <= (A) – [m(ea)]
Subtract Memory from D (16 Bit)	SUBD	X	X	X	X	X		D <= (D) – [m(ea,ea+1)]
Multiply (byte, unsigned)	MUL						X	D <= (A) x (B)
Multiply word, unsigned (signed)	EMUL(S)						X	Y:D <= (D) x (Y)
Unsigned (signed) 32 by 16 divide	EDIV(S)						X	X <= (Y:D) /.(X),Y <= quotient, D <= remainder
Fractional Divide (D < X)	FDIV						X	X <= (D) /.(X), D <= remainder
Integer Divide (unsigned)	IDIV						X	X <= (D) /.(X), D <= remainder

Logical Operations

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
AND A (B) with Memory	ANDA (B)	X	X	X	X	X		A <= A • [m(ea)]
Bit(s) Test A (B) with Memory	BITA (B)	X	X	X	X	X		A • [m(ea)]
One's Complement Memory Byte	COM			X	X	X		m(ea) <= [/m(ea)]
One's Complement A (B)	COMA (B)						X	A <= /A
OR A (B) with Memory (Exclusive)	EORA (B)	X	X	X	X	X		A <= A ⊕ [m(ea)]
OR A (B) with Memory (Inclusive)	ORAA (B)	X	X	X	X	X		A <= A + [m(ea)]

Shift and Rotate

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Arithmetic/Logical Shift Left Memory	ASL/LSL			X	X	X		
Arithmetic/Logical Shift Left A (B)	ASLA(B)						X	
Arithmetic/Logical Shift Left Double	ASLD/LSLD						X	
Arithmetic Shift Right Memory	ASR			X	X	X		
Arithmetic Shift Right A (B)	ASRA(B)						X	
Logical Shift Right A (B)	LSRA(B)						X	
Logical Shift Right Memory	LSR			X	X	X		
Logical Shift Right D	LSRD						X	
Rotate Left Memory	ROL			X	X	X		
Rotate Left A (B)	ROLA(B)						X	
Rotate Right A (B)	RORA(B)						X	
Rotate Right Memory	ROR			X	X	X		

Compare & Test

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Compare A to B	CBA						X	(A)-(B)
Compare A (B) to Memory	CMPA (B)	X	X	X	X	X		(A) - [m(ea)]
Compare D to Memory (16 Bit)	CPD	X	X	X	X	X		(D) - [m(ea,ea+1)]
Compare SP to Memory (16 Bit)	CPS	X	X	X	X	X		(SP) - [m(ea,ea+1)]
Compare X (Y) to Memory (16 Bit)	CPX	X	X	X	X	X		(X) - [m(ea,ea+1)]
Test memory for 0 or minus	TST			X	X	X		m(ea) - 0
Test A (B) for 0 or minus	TSTA (B)						X	(A)-0

Short Branches

Function	Mnemonic	REL	DIR	IDX	[IDX]	PC <= ea if
Branch ALWAYS	BRA	X				
Branch if Carry Clear	BCC	X				C = 0 ?
Branch if Carry Set	BCS	X				C = 1 ?
Branch if Equal Zero	BEQ	X				Z = 1 ?
Branch if Not Equal	BNE	X				Z = 0 ?
Branch if Minus	BMI	X				N = 1 ?
Branch if Plus	BPL	X				N = 0 ?
Branch if Bit(s) Clear in Memory Byte	BRCLR		X	X		[m(ea)]•mask=0
Branch if Bit(s) Set in Memory Byte	BRSET		X	X		[/m(ea)]•mask=0
Branch if Overflow Clear	BVC	X				V = 0 ?
Branch if Overflow Set	BVS	X				V = 1 ?
Branch if Greater Than	BGT	X				Signed >
Branch if Greater Than or Equal	BGE	X				Signed ≥
Branch if Less Than or Equal	BLE	X				Signed ≤
Branch if Less Than	BLT	X				Signed <
Branch if Higher	BHI	X				Unsigned >
Branch if Higher or Same (same as BCC)	BHS	X				Unsigned ≥
Branch if Lower or Same	BLS	X				Unsigned ≤
Branch if Lower (same as BCS)	BLO	X				Unsigned <
Branch Never	BRN	X				3-cycle NOP

Long branch mnemonic = **L** + Short branch mnemonic, e.g.: BRA → **L**BRA

Loop Primitive Instructions (counter ctr = A, B, or D)

Function	Mnemonic	REL	DIR	EXT	IDX	[IDX]	INH	Operation
Decrement counter & branch if =0	DBEQ	X						ctr <= (ctr)-1, if (ctr)=0 => PC <= ea
Decrement counter & branch if ≠0	DBNE	X						ctr <= (ctr)-1, if (ctr) ≠0 => PC <= ea
Increment counter & branch if =0	IBEQ	X						ctr <= (ctr)+1, if (ctr)=0 => PC <= ea
Increment counter & branch if ≠0	IBNE	X						ctr <= (ctr)+1, if (ctr) ≠0 => PC <= ea
Test counter & branch if =0	DBEQ	X						if (ctr)=0 => PC <= ea

Subroutine Calls and Returns

Function	Mnemonic	REL	DIR	EXT	IDX	[IDX]	INH	Operation
Branch to Subroutine	BSR	X						SP <= (SP)-2, m(SP) <= (PC), PC <= ea
Jump to Subroutine	JSR		X	X	X	X		SP <= (SP)-2, m(SP) <= (PC), PC <= ea
CALL a Subroutine (expanded memory)	CALL		X	X	X	X		SP <= (SP)-2, m(SP) <= (PC), PC <= ea SP <= (SP)-1, m(SP) <= (PPG), PC <= pg
Return from Subroutine	RTS						X	PC <= [m(SP)], SP <= (SP)+2
Return from call	RTC						X	PPG <= [m(SP)], SP <= (SP)+1, PC <= [m(SP)], SP <= (SP)+2

Function	Mnemonic	DIR	EXT	IDX	[IDX]	INH	Operation
Jump	JMP	X	X	X	X		PC <= ea

The **jump** instruction allows control to be passed to any address in the 64-Kbyte memory map.

Stack and Index Register Instructions

Function	Mnemonic	IMM	DIR	EXT	IDX	[IDX]	INH	Operation
Decrement Index Register X (Y)	DEX (Y)						X	X <= (X) - 1
Increment Index Register X (Y)	INX (Y)						X	X <= (X) + 1
Load Index Register X (Y)	LDX(Y)	X	X	X	X	X		X <= [m(ea,ea+1)]
Pull X (Y) from Stack	PULX						X	X <= [m(SP,SP+1)] SP <= (SP) + 2
Push X (Y) onto Stack	PSHX (Y)						X	m(SP,SP+1) <= (X) SP <= (SP) - 2
Store Index Register X (Y)	STX (X)	X	X	X	X	X		m(ea,ea+1) <= X
Add Accumulator B to X (Y)	ABX (Y)						X	X <= (X) + (B)
Decrement Stack Pointer	DES						X	SP <= (SP) - 1
Increment Stack Pointer	INS						X	SP <= (SP) + 1
Load Stack Pointer	LDS	X	X	X	X	X		SP <= [m(ea,ea+1)]
Store Stack Pointer	STS	X	X	X	X	X		m(ea,ea+1) <= (SP)
Transfer SP to X (Y)	TSX (Y)						X	X <= (SP)
Transfer X (Y) to SP	TXS (Y)						X	SP <= (X)
Exchange D with X (Y)	XGDX (Y)						X	(D) <=> (X)

Function	Mnemonic	INH	Operation
Return from Interrupt	RTI	X	$(M_{(SP)}) \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow B : A; (SP) + \$0002 \Rightarrow S$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow X_H : X_L; (SP) + \$0004 \Rightarrow S$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L; (SP) + \$0002 \Rightarrow$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow Y_H : Y_L; (SP) + \$0004 \Rightarrow S$
Software Interrupt	SWI	X	
Wait for Interrupt	WAI	X	

Interrupt Handling

The software interrupt (SWI) instruction is similar to a JSR instruction, except the contents of all working CPU registers are saved on the stack rather than just the return address. SWI is unusual in that it is requested by the software program as opposed to other interrupts that are requested asynchronously to the executing program.

Reference Guide for TST Instruction

TST *opr16a*	(M) – 0	EXT	F7 hh ll	rPO	rOP	----	ΔΔ00
TST *oprx0_xysp*	Test Memory for Zero or Minus	IDX	E7 xb	rPf	rfP		
TST *oprx9,xysp*		IDX1	E7 xb ff	rPO	rPO		
TST *oprx16,xysp*		IDX2	E7 xb ee ff	frPP	frPP		
TST [D,*xysp*]		[D,IDX]	E7 xb	fIfrPf	fIfrfP		
TST [*oprx16,xysp*]		[IDX2]	E7 xb ee ff	fIPrPf	fIPrfP		
TSTA	(A) – 0 Test A for Zero or Minus	INH	97	O	O		
TSTB	(B) – 0 Test B for Zero or Minus	INH	D7	O	O		

Reference Guide for JSR Instruction

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail HC12	S X H I	N Z V C
JSR *opr8a*	(SP) – 2 ⇒ SP;	DIR	17 dd	SPPP	PPPS	----	----
JSR *opr16a*	RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$;	EXT	16 hh ll	SPPP	PPPS		
JSR *oprx0_xysp*	Subroutine address ⇒ PC	IDX	15 xb	PPPS	PPPS		
JSR *oprx9,xysp*		IDX1	15 xb ff	PPPS	PPPS		
JSR *oprx16,xysp*	Jump to Subroutine	IDX2	15 xb ee ff	fPPPS	fPPPS		
JSR [D,*xysp*]		[D,IDX]	15 xb	fIfPPPS	fIfPPPS		
JSR [*oprx16,xysp*]		[IDX2]	15 xb ee ff	fIfPPPS	fIfPPPS		

Post Byte Encoding, xb, for Indexed Addressing

ed Addressing Mode Postbyte Encoding (xb)

60	70	80	90	A0	B0	C0	D0	E0	F0							
1,+Y	1,Y+	0,SP	−16,SP	1,+SP	1,SP+	0,PC	−16,PC	n,X	n,SP							
pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	9b const	9b const							
61	71	81	91	A1	B1	C1	D1	E1	F1							
2,+Y	2,Y+	1,SP	−15,SP	2,+SP	2,SP+	1,PC	−15,PC	−n,X	−n,SP							
pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	9b const	9b const							
62	72	82	92	A2	B2	C2	D2	E2	F2							
3,+Y	3,Y+	2,SP	−14,SP	3,+SP	3,SP+	2,PC	−14,PC	n,X	n,SP							
pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	16b const	16b const							
63	73	83	93	A3	B3	C3	D3	E3	F3							
4,+Y	4,Y+	3,SP	−13,SP	4,+SP	4,SP+	3,PC	−13,PC	[n,X]	[n,SP]							
pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	16b indr	16b indr							
64	74	84	94	A4	B4	C4	D4	E4	F4							
5,+Y	5,Y+	4,SP	−12,SP	5,+SP	5,SP+	4,PC	−12,PC	A,X	A,SP							
pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	A offset	A offset							
65	75	85	95	A5	B5	C5	D5	E5	F5							
6,+Y	6,Y+	5,SP	−11,SP	6,+SP	6,SP+	5,PC	−11,PC	B,X	B,SP							
pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	B offset	B offset							
66	76	86	96	A6	B6	C6	D6	E6	F6							
7,+Y	7,Y+	6,SP	−10,SP	7,+SP	7,SP+	6,PC	−10,PC	D,X	D,SP							
pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	D offset	D offset							
67	77	87	97	A7	B7	C7	D7	E7	F7							
8,+Y	8,Y+	7,SP	−9,SP	8,+SP	8,SP+	7,PC	−9,PC	[D,X]	[D,SP]							
pre-inc	post-inc	5b const	5b const	pre-inc	post-inc	5b const	5b const	D indirect	D indirect							
68	78	88	98	A8	B8	C8	D8	E8	F8							
8,−Y	8,Y−	8,SP	−8,SP	8,−SP	8,SP−	8,PC	−8,PC	n,Y	n,PC							
pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	9b const	9b const							
69	79	89	99	A9	B9	C9	D9	E9	F9							
7,−Y	7,Y−	9,SP	−7,SP	7,−SP	7,SP−	9,PC	−7,PC	−n,Y	−n,PC							
pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	9b const	9b const							
6A	7A	8A	9A	AA	BA	CA	DA	EA	FA							
6,−Y	6,Y−	10,SP	−6,SP	6,−SP	6,SP−	10,PC	−6,PC	n,Y	n,PC							
pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	16b const	16b const							
6B	7B	8B	9B	AB	BB	CB	DB	EB	FB							
5,−Y	5,Y−	11,SP	−5,SP	5,−SP	5,SP−	11,PC	−5,PC	[n,Y]	[n,PC]							
pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	16b indr	16b indr							
6C	7C	8C	9C	AC	BC	CC	DC	EC	FC							
4,−Y	4,Y−	12,SP	−4,SP	4,−SP	4,SP−	12,PC	−4,PC	A,Y	A,PC							
pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	A offset	A offset							
6D	7D	8D	9D	AD	BD	CD	DD	ED	FD							
3,−Y	3,Y−	13,SP	−3,SP	3,−SP	3,SP−	13,PC	−3,PC	B,Y	B,PC							
pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	B offset	B offset							
6E	7E	8E	9E	AE	BE	CE	DE	EE	FE							
2,−Y	2,Y−	14,SP	−2,SP	2,−SP	2,SP−	14,PC	−2,PC	D,Y	D,PC							
pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	D offset	D offset							
6F	7F	8F	9F	AF	BF	CF	DF	EF	FF							
1,−Y	1,Y−	15,SP	−1,SP	1,−SP	1,SP−	15,PC	−1,PC	[D,Y]	[D,PC]							
pre-dec	post-dec	5b const	5b const	pre-dec	post-dec	5b const	5b const	D indirect	D indirect							

le 1