

The HCS12 has 6 addressing modes

Most of the HC12's instructions access data in memory

There are several ways for the HC12 to determine which address to access

Effective Address:

Memory address used by instruction (all modes except INH)

ADDRESSING MODE:

How the HC12 calculates the effective address

HCS12 ADDRESSING MODES:

INH	Inherent
IMM	Immediate
DIR	Direct
EXT	Extended
REL	Relative (used only with branch instructions)
IDX	Indexed (won't study indirect indexed mode)

The *Inherent* (INH) addressing mode

Inherent (INH) Addressing Mode

Instructions which work only with registers inside ALU

ABA ; Add B to A $(A) + (B) \rightarrow A$
 18 06

CLRA ; Clear A $0 \rightarrow A$
 87

ASRA ; Arithmetic Shift Right A
 47

TSTA ; Test A $(A) - 0x00$ Set CCR
 97

The HC12 does not access memory

There is no effective address

0x1000	18	0x2000	17	A		B	
	06		35				
	87		02	X			
	47		4A				
	97		C7				

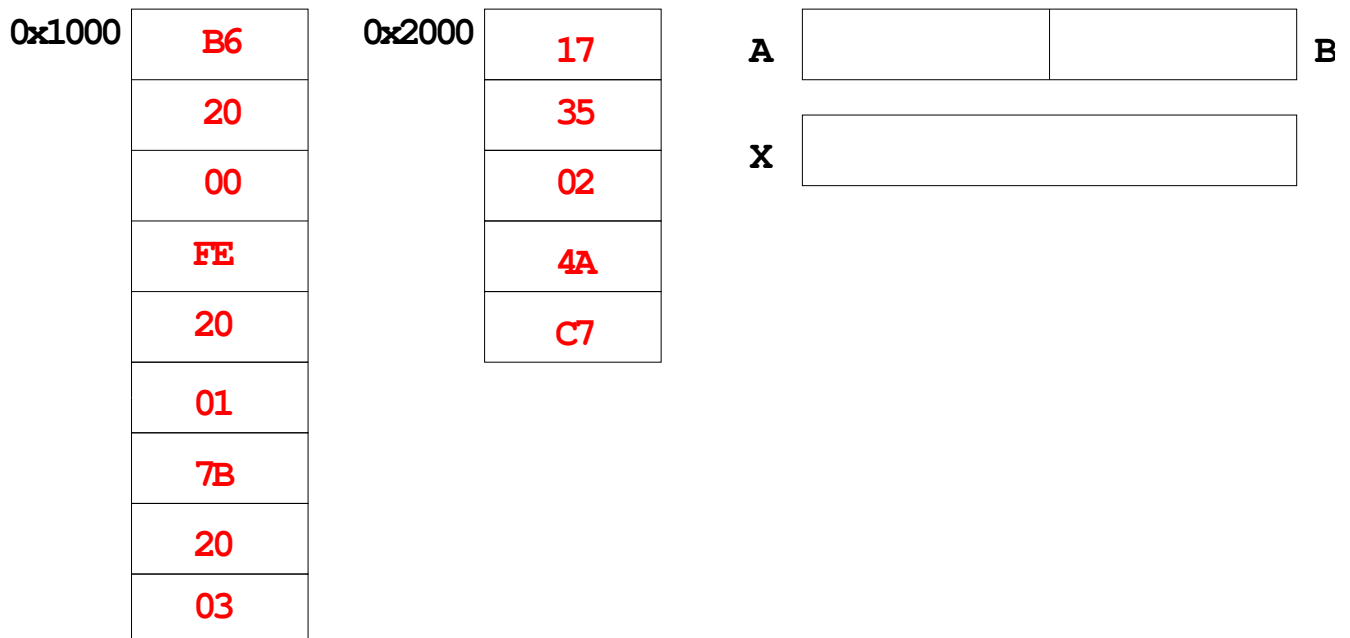
The *Extended* (EXT) addressing mode

Extended (EXT) Addressing Mode

Instructions which give the 16-bit address to be accessed

IDAA \$2000 B6 20 00	; (\$2000) -> A Effective Address: \$2000
IDX \$2001 FE 20 01	; (\$2001:\$2002) -> X Effective Address: \$2001
STAB \$2003 7B 20 03	; (B) -> \$2003 Effective Address: \$2003

Effective address is specified by the two bytes following op code



The *Direct* (DIR) addressing mode

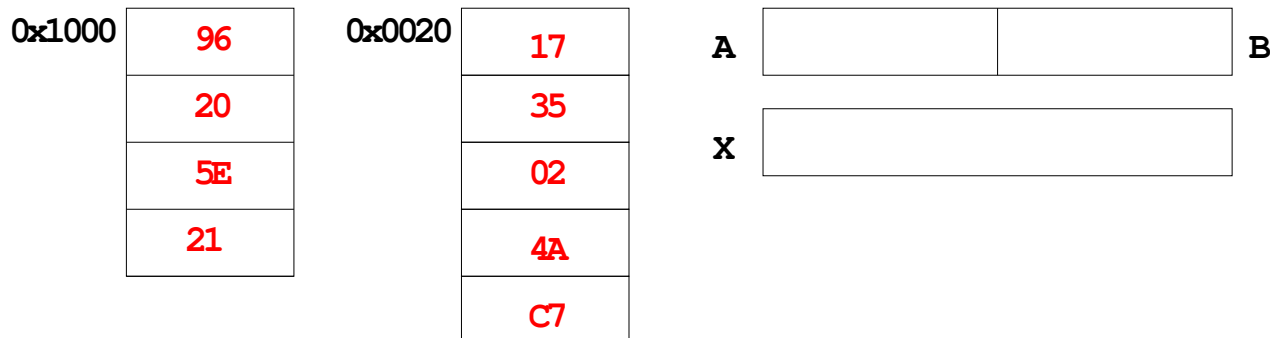
Direct (DIR) Addressing Mode

Instructions which give 8 LSB of address (8 MSB all 0)

LDAA \$20 ; (\$0020) -> A
 96 20 Effective Address: \$0020

STX \$21 ; (X) -> \$0021:\$0022
 5E 21 Effective Address: \$0021

8 LSB of effective address is specified by byte following op code



The *Immediate* (IMM) addressing mode

Immediate (IMM) Addressing Mode

Value to be used is part of instruction

LDAA #\$17 ; \$17 → A
 86 17 Effective Address: PC + 1

ADDA #10 ; (A) + \$0A → A
 8B 0A Effective Address: PC + 1

Effective address is the address following the op code

0x1000	B6	0x2000	17	A			B
	17		35				
	8B		02	X			
	0A		4A				
			C7				

The *Indexed* (IDX, IDX1, IDX2) addressing mode

Indexed (IDX) Addressing Mode

Effective address is obtained from X or Y register (or SP or PC)

Simple Forms

```
LDAA  0,X      ; Use (X) as address to get value to load into A
  A6 00      Effective address: contents of X

ADDA  5,Y      ; Use (Y) + 5 as address to get value to add to r
  AB 45      Effective address: contents of Y + 5

STAA  B,Y      ; Use (Y) + (B) as address to store contents of A into
  AB 45      Effective address: contents of Y + contents of B
```

More Complicated Forms

```
INC   2,X-      ; Post-decrement Indexed
               ; Increment the number at address (X),
               ; then subtract 2 from X
  62 3E      Effective address: contents of X

INC   4,+X      ; Pre-increment Indexed
               ; Add 4 to X
               ; then increment the number at address (X)
  62 23      Effective address: contents of X + 4
```

X		EFF ADDR	
Y		EFF ADDR	

Table 3-1. M68HC12 Addressing Mode Summary

Addressing Mode	Source Format	Abbreviation	Description
Inherent	INST (no externally supplied operands)	INH	Operands (if any) are in CPU registers
Immediate	INST #opr8i or INST #opr16i	IMM	Operand is included in instruction stream 8- or 16-bit size implied by context
Direct	INST opr8a	DIR	Operand is the lower 8 bits of an address in the range \$0000–\$00FF
Extended	INST opr16a	EXT	Operand is a 16-bit address
Relative	INST rel8 or INST rel16	REL	An 8-bit or 16-bit relative offset from the current pc is supplied in the instruction
Indexed (5-bit offset)	INST oprx5,xysp	IDX	5-bit signed constant offset from X, Y, SP, or PC
Indexed (pre-decrement)	INST oprx3,-xys	IDX	Auto pre-decrement x, y, or sp by 1 ~ 8
Indexed (pre-increment)	INST oprx3,+xys	IDX	Auto pre-increment x, y, or sp by 1 ~ 8
Indexed (post-decrement)	INST oprx3,xys-	IDX	Auto post-decrement x, y, or sp by 1 ~ 8
Indexed (post-increment)	INST oprx3,xys+	IDX	Auto post-increment x, y, or sp by 1 ~ 8
Indexed (accumulator offset)	INST abd,xysp	IDX	Indexed with 8-bit (A or B) or 16-bit (D) accumulator offset from X, Y, SP, or PC
Indexed (9-bit offset)	INST oprx9,xysp	IDX1	9-bit signed constant offset from X, Y, SP, or PC (lower 8 bits of offset in one extension byte)
Indexed (16-bit offset)	INST oprx16,xysp	IDX2	16-bit constant offset from X, Y, SP, or PC (16-bit offset in two extension bytes)
Indexed-Indirect (16-bit offset)	INST [oprx16,xysp]	[IDX2]	Pointer to operand is found at... 16-bit constant offset from X, Y, SP, or PC (16-bit offset in two extension bytes)
Indexed-Indirect (D accumulator offset)	INST [D,xysp]	[D,IDX]	Pointer to operand is found at... X, Y, SP, or PC plus the value in D

Different types of indexed addressing modes
 (Note: We will not discuss indirect indexed mode)

INDEXED ADDRESSING MODES

(Does not include indirect modes)

	Example	Effective Address	Offset	Value in X After Done	Registers To Use
Constant Offset	<code>LDAA n,X</code>	$(X)+n$	0 to FFFF	(X)	X, Y, SP, PC
Constant Offset	<code>LDAA -n,X</code>	$(X)-n$	0 to FFFF	(X)	X, Y, SP, PC
Postincrement	<code>LDAA n,X+</code>	(X)	1 to 8	$(X)+n$	X, Y, SP
Preincrement	<code>LDAA n,+X</code>	$(X)+n$	1 to 8	$(X)+n$	X, Y, SP
Postdecrement	<code>LDAA n,X-</code>	(X)	1 to 8	$(X)-n$	X, Y, SP
Predecrement	<code>LDAA n,-X</code>	$(X)-n$	1 to 8	$(X)-n$	X, Y, SP
ACC Offset	<code>LDAA A,X</code> <code>LDAA B,X</code> <code>LDAA D,X</code>	$(X)+(A)$ $(X)+(B)$ $(X)+(D)$	0 to FF 0 to FF 0 to FFFF	(X)	X, Y, SP, PC

The data books list three different types of indexed modes:

- Table 3.2 of the **S12CPUV2 Reference Manual** shows details
- **IDX**: One byte used to specify address
 - Called the postbyte
 - Tells which register to use
 - Tells whether to use autoincrement or autodecrement
 - Tells offset to use
- **IDX1**: Two bytes used to specify address
 - First byte called the postbyte
 - Second byte called the extension
 - Postbyte tells which register to use, and sign of offset
 - Extension tells size of offset
- **IDX2**: Three bytes used to specify address
 - First byte called the postbyte
 - Next two bytes called the extension
 - Postbyte tells which register to use
 - Extension tells size of offset

Table 3-2. Summary of Indexed Operations

Postbyte Code (xb)	Source Code Syntax	Comments rr; 00 = X, 01 = Y, 10 = SP, 11 = PC
rr0nnnnn	,r n,r -n,r	5-bit constant offset n = -16 to +15 r can specify X, Y, SP, or PC
111rr0zs	n,r -n,r	Constant offset (9- or 16-bit signed) z- 0 = 9-bit with sign in LSB of postbyte(s) -256 ≤ n ≤ 255 1 = 16-bit -32,768 ≤ n ≤ 65,535 if z = s = 1, 16-bit offset indexed-indirect (see below) r can specify X, Y, SP, or PC
111rr011	[n,r]	16-bit offset indexed-indirect rr can specify X, Y, SP, or PC -32,768 ≤ n ≤ 65,535
rr1pnrrnn	n,-r n,+r n,r- n,r+	Auto predecrement, preincrement, postdecrement, or postincrement; p = pre-(0) or post-(1), n = -8 to -1, +1 to +8 r can specify X, Y, or SP (PC not a valid choice) +8 = 0111 ... +1 = 0000 -1 = 1111 ... -8 = 1000
111rr1aa	A,r B,r D,r	Accumulator offset (unsigned 8-bit or 16-bit) aa-00 = A 01 = B 10 = D (16-bit) 11 = see accumulator D offset indexed-indirect r can specify X, Y, SP, or PC
111rr111	[D,r]	Accumulator D offset indexed-indirect r can specify X, Y, SP, or PC

Indexed addressing mode instructions use a postbyte to specify index registers (X and Y), stack pointer (SP), or program counter (PC) as the base index register and to further classify the way the effective address is formed. A special group of instructions cause this calculated effective address to be loaded into an index register for further calculations:

- Load stack pointer with effective address (LEAS)
- Load X with effective address (LEAX)
- Load Y with effective address (LEAY)

The *Relative* (REL) addressing mode

Relative (REL) Addressing Mode

The relative addressing mode is used only in branch and long branch instructions.

Branch instruction: One byte following op code specifies how far to branch

Treat the offset as a signed number; add the offset to the address following the current instruction to get the address of the instruction to branch to

BRA **20 35** $PC + 2 + 0035 \rightarrow PC$

BRA **20 C7** $PC + 2 + FFC7 \rightarrow PC$
 $PC + 2 - 0039 \rightarrow PC$

Long branch instruction: Two bytes following op code specifies how far to branch

Treat the offset as an unsigned number; add the offset to the address following the current instruction to get the address of the instruction to branch to

LBEO **18 27 02 1A** If $Z = 1$ then $PC + 4 + 021A \rightarrow PC$
 If $Z = 0$ then $PC + 4 \rightarrow PC$

When writing assembly language program, you don't have to calculate offset

You indicate what address you want to go to, and the assembler calculates the offset

\$1020 **BRA** \$1030 ; Branch to instruction at address \$1030

0x1020	20	PC	
	0E		

Summary of HCS12 addressing modes

ADDRESSING MODES

Name	Example	Op Code	Effective Address
INH Inherent	ABA	18 06	None
IMM Immediate	LDAA #\$35	86 35	PC + 1
DIR Direct	LDAA \$35	96 35	0x0035
EXT Extended	LDAA \$2035	B6 20 35	0x0935
IDX Indexed IDX1 IDX2	LDAA 3,X LDAA 30,X LDAA 300,X	A6 03 A6 E0 13 A6 E2 01 2C	X + 3
IDX Indexed Postincrement	LDAA 3,X+	A6 32	X (X+3 -> X)
IDX Indexed Preincrement	LDAA 3,+X	A6 22	X+3 (X+3 -> X)
IDX Indexed Postdecrement	LDAA 3,X-	A6 3D	X (X-3 -> X)
IDX Indexed Predecrement	LDAA 3,-X	A6 2D	X-3 (X-3 -> X)
REL Relative	BRA \$1050 LBRA \$1F00	20 23 18 20 0E CF	PC + 2 + Offset PC + 4 + Offset

A few instructions have **two** effective addresses:

- **MOVB #\$AA,\$1C00** Move byte 0xAA (IMM) to address \$1C00 (EXT)
- **MOVW 0,X,0,Y** Move word from address pointed to by X (IDX) to address pointed to by Y (IDX)

A few instructions have **three** effective addresses:

- **BRSET F00,\$#03,LABEL** Branch to LABEL (REL) if bits \$#03 (IMM) of variable F00 (EXT) are set.

Using X and Y as Pointers

- Registers X and Y are often used to point to data.
- To initialize pointer use

```
ldx    #table

not
```

```
ldx    table
```

- For example, the following loads the address of `table` (\$1000) into X; i.e., X will point to `table`:

```
ldx    #table    ; Address of table => X
```

The following puts the first two bytes of `table` (\$0C7A) into X. X will **not** point to `table`:

```
ldx    table    ; First two bytes of table => X
```

- To step through `table`, need to increment pointer after use

```
ldaa    0,x
inx
```

or

```
ldaa    1,x+
```

	Data	Address
table	0C	\$1000
	7A	\$1001
	D5	\$1002
	00	\$1003
	61	\$1004
	62	\$1005
	63	\$1006
	64	\$1007

```
org    $1000
table: dc.b  12,122,-43,0
       dc.b  'a'
       dc.b  'b'
       dc.b  'c'
       dc.b  'd'
```

Which branch instruction should you use?

Branch if A > B

Is 0xFF > 0x00?

If unsigned, 0xFF = 255 and 0x00 = 0,
so 0xFF > 0x00

If signed, 0xFF = -1 and 0x00 = 0,
so 0xFF < 0x00

Using unsigned numbers: BHI (checks C bit of OCR)

Using signed numbers: BGT (checks V bit of OCR)

For unsigned numbers, use branch instructions which check C bit

For signed numbers, use branch instructions which check V bit

Hand Assembling a Program

To hand-assemble a program, do the following:

1. Start with the **org** statement, which shows where the first byte of the program will go into memory.
(E.g., **org \$2000** will put the first instruction at address \$2000.)
2. Look at the first instruction. Determine the addressing mode used.
(E.g., **ldab #10** uses IMM mode.)
3. Look up the instruction in the **HCS12 S12CPUV2 Reference Manual**, find the appropriate Addressing Mode, and the Object Code for that addressing mode.
(E.g., **ldab IMM** has object code **C6 ii.**)
 - Table A-1 of the **S12CPUV2 Reference Manual** has a concise summary of the instructions, addressing modes, op-codes, and cycles.
4. Put in the object code for the instruction, and put in the appropriate operand. Be careful to convert decimal operands to hex operands if necessary.
(E.g., **ldab #10** becomes **C6 0A.**)
5. Add the number of bytes of this instruction to the address of the instruction to determine the address of the next instruction.
(E.g., **\$2000 + 2 = \$2002** will be the starting address of the next instruction.)

```
        org    $2000
        ldab   #10
loop:   clra
        dbne   b,loop
        swi
```

Table A-1. Instruction Set Summary (Sheet 7 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
LBGT <i>rel16</i>	Long Branch if Greater Than (if $Z + (N \oplus V) = 0$) (signed)	REL	18 2E qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBHI <i>rel16</i>	Long Branch if Higher (if $C + Z = 0$) (unsigned)	REL	18 22 qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBHS <i>rel16</i>	Long Branch if Higher or Same (if $C = 0$) (unsigned) same function as LBCC	REL	18 24 qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBLT <i>rel16</i>	Long Branch if Less Than or Equal (if $Z + (N \oplus V) = 1$) (signed)	REL	18 2F qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBLO <i>rel16</i>	Long Branch if Lower (if $C = 1$) (unsigned) same function as LBCCS	REL	18 25 qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBLS <i>rel16</i>	Long Branch if Lower or Same (if $C + Z = 1$) (unsigned)	REL	18 23 qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBLT <i>rel16</i>	Long Branch if Less Than (if $N \oplus V = 1$) (signed)	REL	18 2D qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBMI <i>rel16</i>	Long Branch if Minus (if $N = 1$)	REL	18 2B qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBNE <i>rel16</i>	Long Branch if Not Equal (if $Z = 0$)	REL	18 26 qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBPL <i>rel16</i>	Long Branch if Plus (if $N = 0$)	REL	18 2A qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBRA <i>rel16</i>	Long Branch Always (if 1=1)	REL	18 20 qq rr	OPPP	OPPP	----	----
LBRN <i>rel16</i>	Long Branch Never (if 1 = 0)	REL	18 21 qq rr	OPO	OPO	----	----
LBVC <i>rel16</i>	Long Branch if Overflow Bit Clear (if $V=0$)	REL	18 28 qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LBVS <i>rel16</i>	Long Branch if Overflow Bit Set (if $V = 1$)	REL	18 29 qq rr	OPPP/OPO ¹	OPPP/OPO ¹	----	----
LDAA # <i>opr8i</i> LDAA <i>opr8a</i> LDAA <i>opr16a</i> LDAA <i>opr0_xysp</i> LDAA <i>opr09_xysp</i> LDAA <i>opr16_xysp</i> LDAA [<i>D_xysp</i>] LDAA [<i>opr16_xysp</i>]	(M) $\Rightarrow$ 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	P rPf rOP rPf rPO frPP fIfrfP fIPrfP	----	$\Delta \Delta 0-$
LDAB # <i>opr8i</i> LDAB <i>opr8a</i> LDAB <i>opr16a</i> LDAB <i>opr0_xysp</i> LDAB <i>opr09_xysp</i> LDAB <i>opr16_xysp</i> LDAB [<i>D_xysp</i>] LDAB [<i>opr16_xysp</i>]	(M) $\Rightarrow$ 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	P rPf rOP rPf rPO frPP fIfrfP fIPrfP	----	$\Delta \Delta 0-$
LDD # <i>opr16i</i> LDD <i>opr8a</i> LDD <i>opr16a</i> LDD <i>opr0_xysp</i> LDD <i>opr09_xysp</i> LDD <i>opr16_xysp</i> LDD [<i>D_xysp</i>] LDD [<i>opr16_xysp</i>]	(M:M+1) $\Rightarrow$ 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	OP RfP ROP RfP RPO frPP fIfrfP fIPrfP	----	$\Delta \Delta 0-$
LDS # <i>opr16i</i> LDS <i>opr8a</i> LDS <i>opr16a</i> LDS <i>opr0_xysp</i> LDS <i>opr09_xysp</i> LDS <i>opr16_xysp</i> LDS [<i>D_xysp</i>] LDS [<i>opr16_xysp</i>]	(M:M+1) $\Rightarrow$ 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	OP RfP ROP RfP RPO frPP fIfrfP fIPrfP	----	$\Delta \Delta 0-$
LDX # <i>opr16i</i> LDX <i>opr8a</i> LDX <i>opr16a</i> LDX <i>opr0_xysp</i> LDX <i>opr09_xysp</i> LDX <i>opr16_xysp</i> LDX [<i>D_xysp</i>] LDX [<i>opr16_xysp</i>]	(M:M+1) $\Rightarrow$ 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	OP RfP ROP RfP RPO frPP fIfrfP fIPrfP	----	$\Delta \Delta 0-$

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.

Table A-1. Instruction Set Summary (Sheet 3 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
BLS <i>rel8</i>	Branch if Lower or Same (if C + Z = 1) (unsigned)	REL	23 rr	PPP/P ¹	PPP/P ¹	----	----
BLT <i>rel8</i>	Branch if Less Than (if N ⊕ V = 1) (signed)	REL	2D rr	PPP/P ¹	PPP/P ¹	----	----
BMI <i>rel8</i>	Branch if Minus (if N = 1)	REL	2B rr	PPP/P ¹	PPP/P ¹	----	----
BNE <i>rel8</i>	Branch if Not Equal (if Z = 0)	REL	26 rr	PPP/P ¹	PPP/P ¹	----	----
BPL <i>rel8</i>	Branch if Plus (if N = 0)	REL	2A rr	PPP/P ¹	PPP/P ¹	----	----
BRA <i>rel8</i>	Branch Always (if 1 = 1)	REL	20 rr	PPP	PPP	----	----
BRCLR <i>opr8a, msk8, rel8</i> BRCLR <i>opr16a, msk8, rel8</i> BRCLR <i>oprnx0, xysp, msk8, rel8</i> BRCLR <i>oprnx9, xysp, msk8, rel8</i> BRCLR <i>oprnx16, xysp, msk8, rel8</i>	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	rPPP rfPPP rPPP rfPPP frPfPPP	----	----
BRN <i>rel8</i>	Branch Never (if 1 = 0)	REL	21 rr	P	P	----	----
BRSET <i>opr8, msk8, rel8</i> BRSET <i>opr16a, msk8, rel8</i> BRSET <i>oprnx0, xysp, msk8, rel8</i> BRSET <i>oprnx9, xysp, msk8, rel8</i> BRSET <i>oprnx16, xysp, msk8, rel8</i>	Branch if (M̄) • (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	rPPP rfPPP rPPP rfPPP frPfPPP	----	----
BSET <i>opr8, msk8</i> BSET <i>opr16a, msk8</i> BSET <i>oprnx0, xysp, msk8</i> BSET <i>oprnx9, xysp, msk8</i> BSET <i>oprnx16, xysp, msk8</i>	(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	rPwO rPwP rPwO rPwP frPwOP	----	Δ Δ 0 -
BSR <i>rel8</i>	(SP) - 2 ⇒ SP; RTN ₀ ; RTN ₁ ⇒ M _{(SP);M_(SP+1)} Subroutine address ⇒ PC Branch to Subroutine	REL	07 rr	SPPP	PPPS	----	----
BVC <i>rel8</i>	Branch if Overflow Bit Clear (if V = 0)	REL	28 rr	PPP/P ¹	PPP/P ¹	----	----
BVS <i>rel8</i>	Branch if Overflow Bit Set (if V = 1)	REL	29 rr	PPP/P ¹	PPP/P ¹	----	----
CALL <i>opr16a, page</i> CALL <i>oprnx0, xysp, page</i> CALL <i>oprnx9, xysp, page</i> CALL <i>oprnx16, xysp, page</i> CALL [D, xysp] CALL [oprnx16, xysp]	(SP) - 2 ⇒ SP; RTN ₀ ; RTN ₁ ⇒ 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	gnfSsPPP gnfSsPPP gnfSsPPP fgnfSsPPP fIignSsPPP fIignSsPPP	----	----
CBA	(A) - (B) Compare 8-Bit Accumulators	INH	18 17	OO	OO	----	Δ Δ Δ Δ
CLC	0 ⇒ C Translates to ANDCC #SFE	IMM	10 FE	P	P	----	---0
CLI	0 ⇒ I Translates to ANDCC #SEF (enables I-bit interrupts)	IMM	10 EF	P	P	---0	----
CLR <i>opr16a</i> CLR <i>oprnx0, xysp</i> CLR <i>oprnx9, xysp</i> CLR <i>oprnx16, xysp</i> CLR [D, xysp] CLR [oprnx16, 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 ee ff 87 C7	PwO Pw PwO PwP PIfPw PIfPw O O	wOP Pw PwO PwP PIfPw PIfPw O O	----	0 1 0 0
CLV	0 ⇒ V Translates to ANDCC #SFD	IMM	10 FD	P	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.

CMPA #opr8i CMPA opr8a CMPA opr16a CMPA oprnx0, xysp CMPA oprnx9, xysp CMPA oprnx16, xysp CMPA [D, xysp] CMPA [oprnx16, 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 fIfrPf	P rPf rOP rPf rPO frPP fIfrPf fIfrPf	----	Δ Δ Δ Δ
--	--	---	--	---	---	------	---------

Table A-1. Instruction Set Summary (Sheet 4 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12 M68HC12	S X H I	N Z V C
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 1l E1 xb E1 xb ff E1 xb ee ff E1 xb ee ff	P rPf rPO rPF rPO frPP fIFrPf fIPrPf	P rPf rOP 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	(M) ⇒ M equivalent to \$FF – (M) ⇒ M 1's Complement Memory Location (A) ⇒ A Complement Accumulator A (B) ⇒ B Complement Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	71 hh 1l 61 xb 61 xb ff 61 xb ee ff 61 xb 61 xb ee ff 41 51	rPwO rPw rPwO frPP fIFrPf fIPrPf O O	rOPw rPw rPOw frPPw fIFrPf fIPrPf 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 1l AC xb AC xb ff AC xb ee ff AC xb AC xb ee ff	PO RPF RPO RPF RPO frPP fIFrPf fIPrPf	OP RPF ROP 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 1l AF xb AF xb ff AF xb ee ff AF xb AF xb ee ff	PO RPF RPO RPF RPO frPP fIFrPf fIPrPf	OP RPF ROP 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 1l AE xb AE xb ff AE xb ee ff AE xb AE xb ee ff	PO RPF RPO RPF RPO frPP fIFrPf fIPrPf	OP RPF ROP 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 1l AD xb AD xb ff AD xb ee ff AD xb AD xb ee ff	PO RPF RPO RPF RPO frPP fIFrPf fIPrPf	OP RPF ROP RPF RPO frPP fIFrPf fIPrPf	---- Δ Δ Δ Δ
DAA	Adjust Sum to BCD Decimal Adjust Accumulator A	INH	18 07	OfO	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 1b rr	PPP (branch) PPO (no branch)	PPP	---- ----
DBNE abdxys, rel9	(cntr) – 1 ⇒ cntr If (cntr) not = 0, then Branch; else Continue to next instruction Decrement Counter and Branch if ≠ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr	PPP (branch) PPO (no branch)	PPP	---- ----

DBNE Decrement and Branch if Not Equal to Zero DBNE

Operation (counter) – 1 $\Rightarrow$ counter
 If (counter) not = 0, then (PC) + \$0003 + rel $\Rightarrow$ PC

Subtracts one from the counter register A, B, D, X, Y, or SP. Branches to a relative destination if the counter register does not reach zero. Rel is a 9-bit two's complement offset for branching forward or backward in memory. Branching range is \$100 to \$0FF (–256 to +255) from the address following the last byte of object code in the instruction.

CCR

Effects

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Code and CPU Cycles

Source Form	Address Mode	Machine Code (Hex)	CPU Cycles
DBNE <i>abdxysp, rel9</i>	REL (9-bit)	04 1b rr	PPP (branch) PPO (no branch)

Loop Primitive Postbyte (1b) Coding				
Source Form	Postbyte ¹	Object Code	Counter Register	Offset
DBNE A, <i>rel9</i>	0010 X000	04 20 rr	A	Positive
DBNE B, <i>rel9</i>	0010 X001	04 21 rr	B	
DBNE D, <i>rel9</i>	0010 X100	04 24 rr	D	
DBNE X, <i>rel9</i>	0010 X101	04 25 rr	X	
DBNE Y, <i>rel9</i>	0010 X110	04 26 rr	Y	
DBNE SP, <i>rel9</i>	0010 X111	04 27 rr	SP	
DBNE A, <i>rel9</i>	0011 X000	04 30 rr	A	Negative
DBNE B, <i>rel9</i>	0011 X001	04 31 rr	B	
DBNE D, <i>rel9</i>	0011 X100	04 34 rr	D	
DBNE X, <i>rel9</i>	0011 X101	04 35 rr	X	
DBNE Y, <i>rel9</i>	0011 X110	04 36 rr	Y	
DBNE SP, <i>rel9</i>	0011 X111	04 37 rr	SP	

NOTES:

- Bits 7:6:5 select DBEQ or DBNE; bit 4 is the offset sign bit; bit 3 is not used; bits 2:1:0 select the counter register.

Core User Guide — S12CPU15UG V1.2

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C																
STY <i>opr8a</i> STY <i>opr16a</i> STY <i>opr0_xysppc</i> STY <i>opr9_xysppc</i> STY <i>opr16_xysppc</i> STY [D, <i>xysppc</i>] STY [<i>opr16_xysppc</i>]	Store Y (Y _H :Y _L)⇒M:M+1	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5D dd 7D hh ll 6D xb 6D xb ff 6D xb ee ff 6D xb 6D xb ee ff	PW PW0 PW PW0 PWP PIfW PIPW	<table><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table>	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-														
-	-	-	-	-	-	-	-														
SUBA # <i>opr8i</i> SUBA <i>opr8a</i> SUBA <i>opr16a</i> SUBA <i>opr0_xysppc</i> SUBA <i>opr9_xysppc</i> SUBA <i>opr16_xysppc</i> SUBA [D, <i>xysppc</i>] SUBA [<i>opr16_xysppc</i>]	Subtract from A (A)-(M)⇒A or (A)-imm⇒A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	80 ii 90 dd B0 hh ll A0 xb A0 xb ff A0 xb ee ff A0 xb A0 xb ee ff	P rPf rP0 rPf rP0 frPP fIfrPf fIPrPf	<table><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table>	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-														
-	-	-	-	-	-	-	-														
SUBB # <i>opr8i</i> SUBB <i>opr8a</i> SUBB <i>opr16a</i> SUBB <i>opr0_xysppc</i> SUBB <i>opr9_xysppc</i> SUBB <i>opr16_xysppc</i> SUBB [D, <i>xysppc</i>] SUBB [<i>opr16_xysppc</i>]	Subtract from B (B)-(M)⇒B or (B)-imm⇒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 rP0 rPf rP0 frPP fIfrPf fIPrPf	<table><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table>	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-														
-	-	-	-	-	-	-	-														
SUBD # <i>opr16i</i> SUBD <i>opr8a</i> SUBD <i>opr16a</i> SUBD <i>opr0_xysppc</i> SUBD <i>opr9_xysppc</i> SUBD <i>opr16_xysppc</i> SUBD [D, <i>xysppc</i>] SUBD [<i>opr16_xysppc</i>]	Subtract from D (A:B)-(M:M+1)⇒A:B or (A:B)-imm⇒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	P0 RPf RP0 RPf RP0 fRPP fIfrPf fIPrPf	<table><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table>	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-														
-	-	-	-	-	-	-	-														
SWI	Software interrupt; (SP)-2⇒SP RTN _H :RTN _L ⇒M _{SP} :M _{SP+1} (SP)-2⇒SP; (Y _H :Y _L)⇒M _{SP} :M _{SP+1} (SP)-2⇒SP; (X _H :X _L)⇒M _{SP} :M _{SP+1} (SP)-2⇒SP; (B:A)⇒M _{SP} :M _{SP+1} (SP)-1⇒SP; (CCR)⇒M _{SP} ; 1⇒I (SWI vector)⇒PC	INH	3F	VSPSSPS* <																	

68HC12 Cycles

- 68HC12 works on 48 MHz clock
- A processor cycle takes 2 clock cycles – P clock is 24 MHz
- Each processor cycle takes 41.7 ns ($1/24 \mu s$) to execute
- An instruction takes from 1 to 12 processor cycles to execute
- You can determine how many cycles an instruction takes by looking up the CPU cycles for that instruction in the S12CPUV2 Reference Manual.
 - For example, LDAA using the IMM addressing mode shows one CPU cycle (of type P).
 - LDAA using the EXT addressing mode shows three CPU cycles (of type rPO).
 - Section 6.6 of the S12CPUV2 Reference Manual explains what the HCS12 is doing during each of the different types of CPU cycles.

			org	\$2000	;	Inst	Mode	Cycles
000			ldab	#10	;	LDAB	(IMM)	1
2000 C6 0A			clra		;	CLRA	(INH)	1
2002 87	loop:		dbne	b,loop	;	DBNE	(REL)	3
2003 04 31 FC			swi		;	SWI		9
2006 3F								

The program executes the `ldab #10` instruction once (which takes one cycle). It then goes through loop 10 times (which has two instructions, one with one cycle and one with three cycles), and finishes with the `swi` instruction (which takes 9 cycles).

Total number of cycles:

$$1 + 10 \times (1 + 3) + 9 = 50$$

$$50 \text{ cycles} = 50 \times 41.7 \text{ ns/cycle} = 2.08 \mu s$$

LDAB

Load B

LDAB

Operation (M) ⇒ B
or
imm ⇒ B

Loads B with either the value in M or an immediate value.

CCR Effects

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise
Z: Set if result is \$00; cleared otherwise
V: Cleared

Code and CPU Cycles

Source Form	Address Mode	Machine Code (Hex)	CPU Cycles
LDAB #opr8i	IMM	C6 ii	P
LDAB opr8a	DIR	D6 dd	rPf
LDAB opr16a	EXT	F6 hh ll	rP0
LDAB oprx0_xysppc	IDX	E6 xb	rPf
LDAB oprx9_xysppc	IDX1	E6 xb ff	rP0
LDAB oprx16_xysppc	IDX2	E6 xb ee ff	frPP
LDAB [D,xysppc]	[D,IDX]	E6 xb	fIfrPf
LDAB [oprx16,xysppc]	[IDX2]	E6 xb ee ff	fIPrPf