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

ADDRESSING MODE:

How the HC12 calculates the effective address

HC12 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

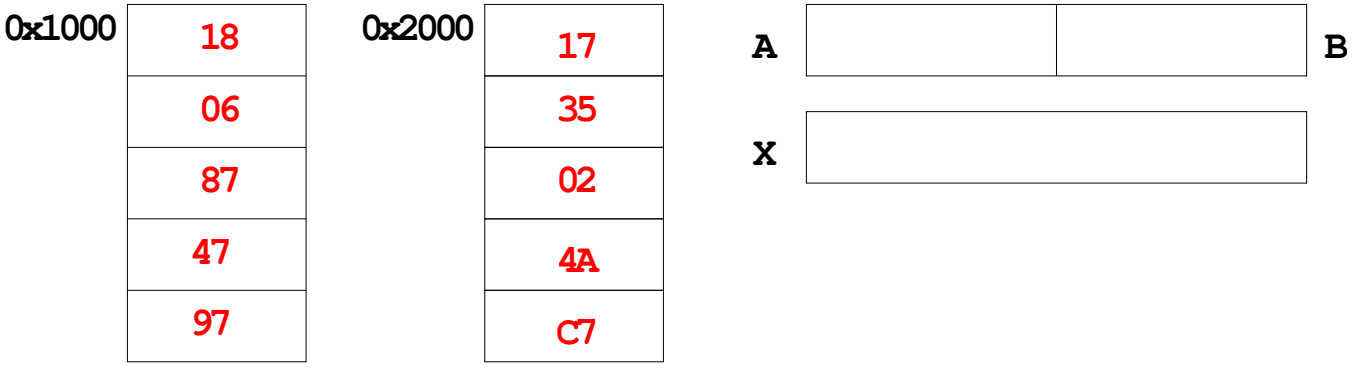
Inherent (INH) Addressing Mode

Instructions which work only with registers inside ALU

```
ABA      ; Add B to A    (A) + (B) -> A
  18 06

CLRA     ; Clear A       0 -> 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



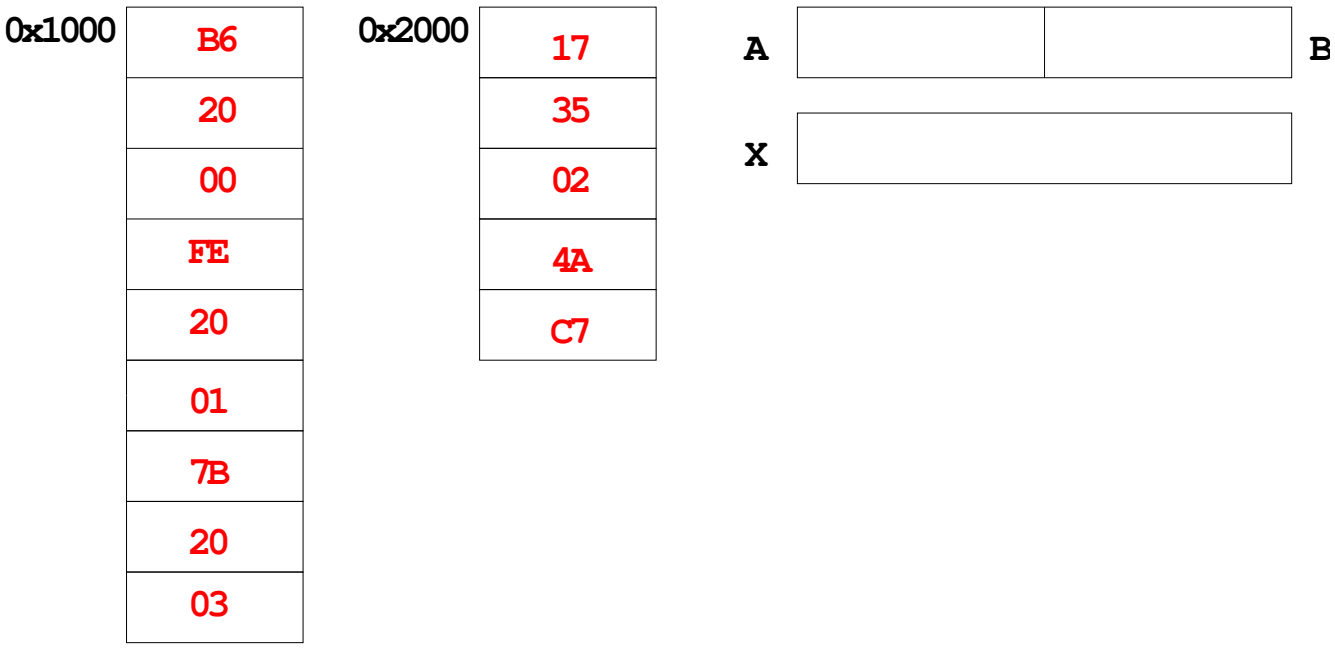
The *Extended* (EXT) addressing mode

Extended (EXT) Addressing Mode

Instructions which give the 16–bit address to be accessed

```
LDAA  $2000      ; ($2000) -> A
  B6 20 00         Effective Address:  $2000

LDX   $2001      ; ($2001:$2002) -> X
  FE 20 01         Effective Address:  $2001

STAB  $2003      ; (B) -> $2003
  7B 20 03         Effective Address:  $2003
```

Effective address is specified by the two bytes following op code



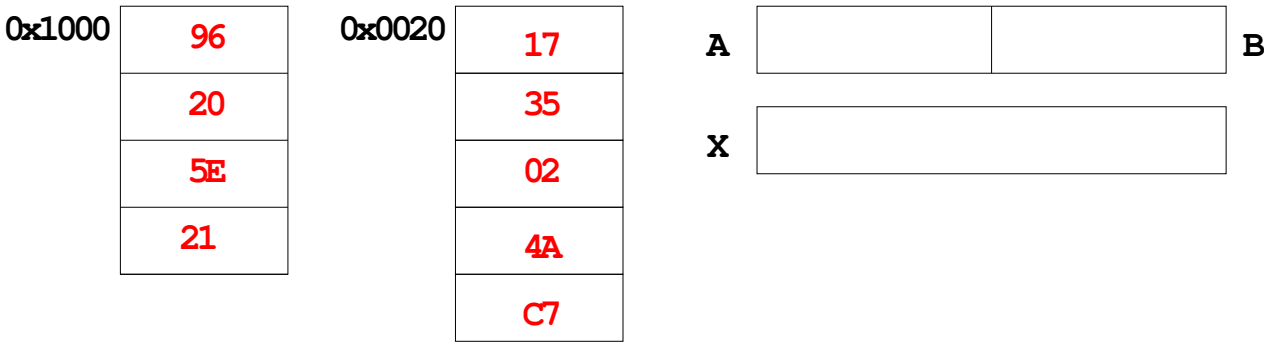
The *Direct* (DIR) addressing mode

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



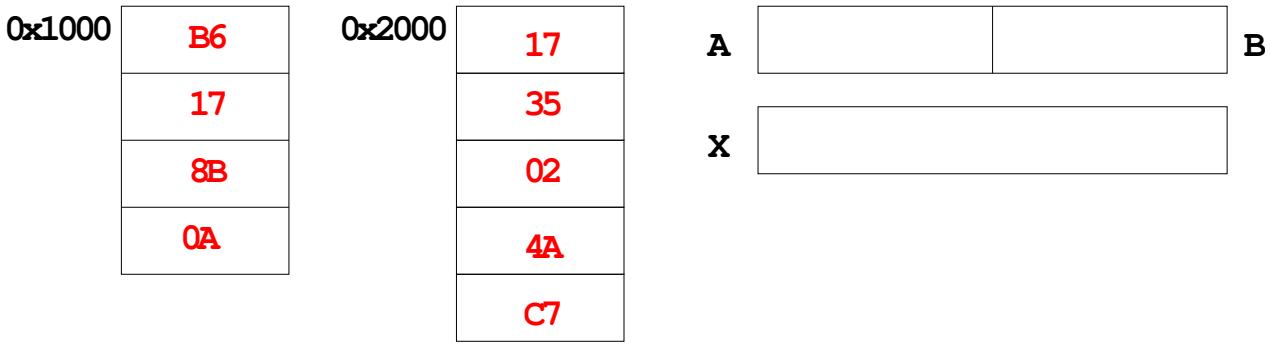
The *Immediate* (IMM) addressing mode

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



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 put in A
 A6 00           Effective address:  contents of X

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

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 [ ]

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	LDAA n,X	(X)+n	0 to FFFF	(X)	X, Y, SP, PC
Constant Offset	LDAA −n,X	(X)−n	0 to FFFF	(X)	X, Y, SP, PC
Postincrement	LDAA n,X+	(X)	1 to 8	(X)+n	X, Y, SP
Preincrement	LDAA n,+X	(X)+n	1 to 8	(X)+n	X, Y, SP
Postdecrement	LDAA n,X−	(X)	1 to 8	(X)−n	X, Y, SP
Predecrement	LDAA n,−X	(X)−n	1 to 8	(X)−n	X, Y, SP

ACC Offset	LDAA A,X
	LDAA B,X
	LDAA D,X
		(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 4.2 of the **Core Users Guide** 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 4-2 Summary of Indexed Operations

5-bit constant offset indexed addressing (IDX)

	7	6	5	4	3	2	1	0
Postbyte:	rr[1]		0	5-bit signed offset				

Effective address = 5-bit signed offset + (X, Y, SP, or PC)

Accumulator offset addressing (IDX)

	7	6	5	4	3	2	1	0
Postbyte:	1	1	1	rr[1]		1	aa[2]	

Effective address = (X, Y, SP, or PC) + (A, B, or D)

Autodecrement/autoincrement) indexed addressing (IDX)

	7	6	5	4	3	2	1	0
Postbyte:	rr[1,3]		1	p[4]	4-bit inc/dec value[5]			

Effective address = (X, Y, or SP) ± 1 to 8

9-bit constant offset indexed addressing (IDX1)

	7	6	5	4	3	2	1	0
Postbyte:	1	1	1	rr[1]		0	0	s[6]

Effective address = s:(offset extension byte) + (X, Y, SP, or PC)

16-bit constant offset indexed addressing (IDX2)

	7	6	5	4	3	2	1	0
Postbyte:	1	1	1	rr[1]		0	1	0

Effective address = (two offset extension bytes) + (X, Y, SP, or PC)

16-bit constant offset indexed-indirect addressing ([IDX2])

	7	6	5	4	3	2	1	0
Postbyte:	1	1	1	rr[1]		0	1	1

(two offset extension bytes) + (X, Y, SP, or PC) is address of pointer to effective address

Accumulator D offset indexed-indirect addressing ([D,IDX])

	7	6	5	4	3	2	1	0
Postbyte:	1	1	1	rr[1]		1	1	1

(X, Y, SP, or PC) + (D) is address of pointer to effective address

NOTES:
1. rr selects X (00), Y (01), SP (10), or PC (11).
2. aa selects A (00), B (01), or D (10).
3. In autoincrement/decrement indexed addressing, PC is not a valid selection.
4. p selects pre- (0) or post- (1) increment/decrement.
5. Increment values range from 0000 (+1) to 0111 (+8). Decrement values range from 1111 (–1) to 1000 (–8).
6. s is the sign bit of the offset extension byte.

All indexed addressing modes use a 16-bit CPU register and additional information to create an indexed address. In most cases the indexed address is the effective address of the instruction, that is, the address of the memory location that the instruction acts on. In indexed-indirect addressing, the indexed address is the location of a value that points to the effective address.

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 -> PC

BRA    20 C7          PC + 2 + FFC7 -> PC
                      PC + 2 -  0039  -> PC
```

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

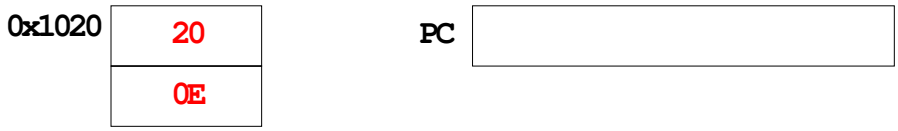
Treat the offset as an usigned number; add the offset to the address following the current instruction to get the address of the instruction to branch to

```
LBEQ   18 27 02 1A    If Z == 1 then PC + 4 + 021A -> PC
                      If Z == 0 then PC + 4 -> 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
```



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 $2000,$3000` Move byte from address `$2000` to `$3000`
- `MOVW 0,X,0,Y` Move word from address pointed to by `X` to address pointed to by `Y`

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` (`$2000`) 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`:

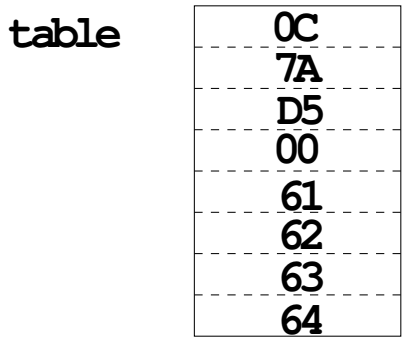
```
ldx     table     ;  First two bytes of table => X
```

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

```
ldaa    0,x
inx
```

or

```
ldaa    1,x+
```




```
        org   $2000
table:  dc.b  12,122,-43,0
        dc.b  'a','b','c','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 CCR)

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

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 Core Users Guide**, find the appropriate Addressing Mode, and the Object Code for that addressing mode.
   (E.g., `ldab IMM` has object code `C6 ii`.)
   - Table 5.1 of the **Core Users Guide** 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 addrss of the next instruction.)

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

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
LBMI *rel16*	Long branch if minus If N=1, then (PC)+4+rel⇒PC	REL	18 2B qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBNE *rel16*	Long branch if not equal to 0 If Z=0, then (PC)+4+rel⇒PC	REL	18 26 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBPL *rel16*	Long branch if plus If N=0, then (PC)+4+rel⇒PC	REL	18 2A qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBRA *rel16*	Long branch always	REL	18 20 qq rr	OPPP	– – – – – – – –
LBRN *rel16*	Long branch never	REL	18 21 qq rr	OPO	– – – – – – – –
LBVC *rel16*	Long branch if V clear If V=0,then (PC)+4+rel⇒PC	REL	18 28 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –
LBVS *rel16*	Long branch if V set If V=1,then (PC)+4+rel⇒PC	REL	18 29 qq rr	OPPP (branch) OPO (no branch)	– – – – – – – –

LDAA #*opr8i*
LDAA *opr8a*
LDAA *opr16a*
LDAA *oprx0_xysppc*
LDAA *oprx9,xysppc*
LDAA *oprx16,xysppc*
LDAA [D,*xysppc*]
LDAA [*oprx16,xysppc*]
	Load A
	(M)⇒A
	or imm⇒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_xysppc*
LDAB *oprx9,xysppc*
LDAB *oprx16,xysppc*
LDAB [D,*xysppc*]
LDAB [*oprx16,xysppc*]
	Load B
	(M)⇒B
	or imm⇒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_xysppc*
LDD *oprx9,xysppc*
LDD *oprx16,xysppc*
LDD [D,*xysppc*]
LDD [*oprx16,xysppc*]
	Load D
	(M:M+1)⇒A:B
	or imm⇒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 –

LDS #*opr16i*
LDS *opr8a*
LDS *opr16a*
LDS *oprx0_xysppc*
LDS *oprx9,xysppc*
LDS *oprx16,xysppc*
LDS [D,*xysppc*]
LDS [*oprx16,xysppc*]
	Load SP
	(M:M+1)⇒SP
	or imm⇒SP
		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_xysppc*
LDX *oprx9,xysppc*
LDX *oprx16,xysppc*
LDX [D,*xysppc*]
LDX [*oprx16,xysppc*]
	Load X
	(M:M+1)⇒X
	or imm⇒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_xysppc*
LDY *oprx9,xysppc*
LDY *oprx16,xysppc*
LDY [D,*xysppc*]
LDY [*oprx16,xysppc*]
	Load Y
	(M:M+1)⇒Y
	or imm⇒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 –

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
BRCLR *opr8a*, *msk8*, *rel8* BRCLR *opr16a*, *msk8*, *rel8* BRCLR *oprx0_xysppc*, *msk8*, *rel8* BRCLR *oprx9,xysppc*, *msk8*, *rel8* BRCLR *oprx16,xysppc*, *msk8*, *rel8*	Branch if bit(s) clear; if (M)•(mask byte)=0, then (PC)+2+rel⇒PC	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	REL	21 rr	P	– – – – – – – –
BRSET *opr8, msk8, rel8* BRSET *opr16a*, *msk8*, *rel8* BRSET *oprx0_xysppc*, *msk8*, *rel8* BRSET *oprx9,xysppc*, *msk8*, *rel8* BRSET *oprx16,xysppc*, *msk8*, *rel8*	Branch if bit(s) set; if ($\overline{M}$)•(mask byte)=0, then (PC)+2+rel⇒PC	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_xysppc*, *msk8* BSET *oprx9,xysppc*, *msk8* BSET *oprx16,xysppc*, *msk8*	Set bit(s) in M (M) | mask byte⇒M	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*	Branch to subroutine; (SP)–2⇒SP RTN_H:RTN_L⇒M_{SP}:M_{SP+1} (PC)+2+rel⇒PC	REL	07 rr	SPPP	– – – – – – – –
BVC *rel8*	Branch if V clear; if V=0, then (PC)+2+rel⇒PC	REL	28 rr	PPP (branch) P (no branch)	– – – – – – – –
BVS *rel8*	Branch if V set; if V=1, then (PC)+2+rel⇒PC	REL	29 rr	PPP (branch) P (no branch)	– – – – – – – –
CALL *opr16a*, *page* CALL *oprx0_xysppc*, *page* CALL *oprx9,xysppc*, *page* CALL *oprx16,xysppc*, *page* CALL [D,*xysppc*] CALL [*oprx16*, *xysppc*]	Call subroutine in expanded memory (SP)–2⇒SP RTN_H:RTN_L⇒M_{SP}:M_{SP+1} (SP)–1⇒SP; (PPG)⇒M_{SP} pg⇒PPAGE register subroutine address⇒PC	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	Compare A to B; (A)–(B)	INH	18 17	OO	– – – – Δ Δ Δ Δ
CLCSame as ANDCC #$FE	Clear C bit	IMM	10 FE	P	– – – – – – – 0
CLISame as ANDCC #$EF	Clear I bit	IMM	10 EF	P	– – – 0 – – – –
CLR *opr16a* CLR *oprx0_xysppc* CLR *oprx9,xysppc* CLR *oprx16,xysppc* CLR [D,*xysppc*] CLR [*oprx16,xysppc*] CLRA CLRB	Clear M; $00⇒M Clear A; $00⇒A Clear B; $00⇒B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	79 hh ll 69 xb 69 xb ff 69 xb ee ff 69 xb 69 xb ee ff 87 C7	PwO Pw PwO PwP PIfw PIPw O O	– – – – 0 1 0 0
CLVSame as ANDCC #$FD	Clear V	IMM	10 FD	P	– – – – – – 0 –

CMPA #*opr8i*
CMPA *opr8a*
CMPA *opr16a*
CMPA *oprx0_xysppc*
CMPA *oprx9,xysppc*
CMPA *oprx16,xysppc*
CMPA [D,*xysppc*]
CMPA [*oprx16,xysppc*]
	Compare A
	(A)–(M) or (A)–imm
		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_xysppc*
CMPB *oprx9,xysppc*
CMPB *oprx16,xysppc*
CMPB [D,*xysppc*]
CMPB [*oprx16,xysppc*]
	Compare B
	(B)–(M) or (B)–imm
		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

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
COM *opr16a* COM *oprx0_xysppc* COM *oprx9,xysppc* COM *oprx16,xysppc* COM [D,*xysppc*] COM [*oprx16,xysppc*] COMA COMB	Complement M; ($\overline{M}$)=$FF–(M)⇒M Complement A; ($\overline{A}$)=$FF–(A)⇒A Complement B; ($\overline{B}$)=$FF–(B)⇒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	rPw0 rPw rPw0 frPwP fIfrPw fIPrPw 0 0	– – – – Δ Δ 0 1
CPD #*opr16i* CPD *opr8a* CPD *opr16a* CPD *oprx0_xysppc* CPD *oprx9,xysppc* CPD *oprx16,xysppc* CPD [D,*xysppc*] CPD [*oprx16,xysppc*]	Compare D (A:B)–(M:M+1) or (A:B)–imm	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	P0 RPf RP0 RPf RP0 fRPP fIfRPf fIPRPf	– – – – Δ Δ Δ Δ
CPS #*opr16i* CPS *opr8a* CPS *opr16a* CPS *oprx0_xysppc* CPS *oprx9,xysppc* CPS *oprx16,xysppc* CPS [D,*xysppc*] CPS [*oprx16,xysppc*]	Compare SP (SP)–(M:M+1) or (SP)–imm	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	P0 RPf RP0 RPf RP0 fRPP fIfRPf fIPRPf	– – – – Δ Δ Δ Δ
CPX #*opr16i* CPX *opr8a* CPX *opr16a* CPX *oprx0_xysppc* CPX *oprx9,xysppc* CPX *oprx16,xysppc* CPX [D,*xysppc*] CPX [*oprx16,xysppc*]	Compare X (X)–(M:M+1) or (X)–imm	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	P0 RPf RP0 RPf RP0 fRPP fIfRPf fIPRPf	– – – – Δ Δ Δ Δ
CPY #*opr16i* CPY *opr8a* CPY *opr16a* CPY *oprx0_xysppc* CPY *oprx9,xysppc* CPY *oprx16,xysppc* CPY [D,*xysppc*] CPY [*oprx16,xysppc*]	Compare Y (Y)–(M:M+1) or (Y)–imm	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	P0 RPf RP0 RPf RP0 fRPP fIfRPf fIPRPf	– – – – Δ Δ Δ Δ
DAA	Decimal adjust A for BCD	INH	18 07	0f0	– – – – Δ Δ ? Δ
DBEQ *abdxysp*, *rel9*	Decrement and branch if equal to 0 (counter)–1⇒counter if (counter)=0, then branch	REL (9-bit)	04 lb rr	PPP (branch) PP0 (no branch)	– – – – – – – –
DBNE *abdxysp*, *rel9*	Decrement and branch if not equal to 0; (counter)–1⇒counter; if (counter)≠0, then branch	REL (9-bit)	04 lb rr	PPP (branch) PP0 (no branch)	– – – – – – – –
DEC *opr16a* DEC *oprx0_xysppc* DEC *oprx9,xysppc* DEC *oprx16,xysppc* DEC [D,*xysppc*] DEC [*oprx16,xysppc*] DECA DECB	Decrement M; (M)–1⇒M Decrement A; (A)–1⇒A Decrement B; (B)–1⇒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	rPw0 rPw rPw0 frPwP fIfrPw fIPrPw 0 0	– – – – Δ Δ Δ –
DESSame as LEAS –1,SP	Decrement SP; (SP)–1⇒SP	IDX	1B 9F	Pf	– – – – – – – –
DEX	Decrement X; (X)–1⇒X	INH	09	0	– – – – – Δ – –
DEY	Decrement Y; (Y)–1⇒Y	INH	03	0	– – – – – Δ – –
EDIV	Extended divide, unsigned; 32 by 16 to 16-bit; (Y:D)÷(X)⇒Y; remainder⇒D	INH	11	fffffffff0	– – – – Δ Δ Δ Δ

DBNE Decrement and Branch if Not Equal to Zero DBNE

Operation

(counter) – 1 ⇒ counter
If (counter) not = 0, then (PC) + $0003 + rel ⇒ 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 *abdxysp*, *rel9*	REL (9-bit)	`04 lb rr`	PPP (branch) PPO (no branch)

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

NOTES:
1. 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.

Source Form	Operation	Address Mode	Machine Coding (Hex)	Access Detail	S X H I N Z V C
STY *opr8a* STY *opr16a* STY *oprx0_xysppc* STY *oprx9,xysppc* STY *oprx16,xysppc* STY [D,*xysppc*] STY [*oprx16,xysppc*]	Store Y $(Y_H:Y_L) \Rightarrow M:M+1$	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 PWO PW PWO PWP PIfW PIPW	– – – – Δ Δ 0 –

SUBA #*opr8i*
SUBA *opr8a*
SUBA *opr16a*
SUBA *oprx0_xysppc*
SUBA *oprx9,xysppc*
SUBA *oprx16,xysppc*
SUBA [D,*xysppc*]
SUBA [*oprx16,xysppc*]
	Subtract from A
	$(A) - (M) \Rightarrow A$
	or $(A) - imm \Rightarrow A$
		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
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					– – – – Δ Δ Δ Δ

SUBB #*opr8i*
SUBB *opr8a*
SUBB *opr16a*
SUBB *oprx0_xysppc*
SUBB *oprx9,xysppc*
SUBB *oprx16,xysppc*
SUBB [D,*xysppc*]
SUBB [*oprx16,xysppc*]
	Subtract from B
	$(B) - (M) \Rightarrow B$
	or $(B) - imm \Rightarrow B$
		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_xysppc*
SUBD *oprx9,xysppc*
SUBD *oprx16,xysppc*
SUBD [D,*xysppc*]
SUBD [*oprx16,xysppc*]
	Subtract from D
	$(A:B) - (M:M+1) \Rightarrow A:B$
	or $(A:B) - imm \Rightarrow A:B$
		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	Software interrupt; $(SP) - 2 \Rightarrow SP$ $RTN_H:RTN_L \Rightarrow M_{SP}:M_{SP+1}$ $(SP) - 2 \Rightarrow SP$; $(Y_H:Y_L) \Rightarrow M_{SP}:M_{SP+1}$ $(SP) - 2 \Rightarrow SP$; $(X_H:X_L) \Rightarrow M_{SP}:M_{SP+1}$ $(SP) - 2 \Rightarrow SP$; $(B:A) \Rightarrow M_{SP}:M_{SP+1}$ $(SP) - 1 \Rightarrow SP$; $(CCR) \Rightarrow M_{SP}$; $1 \Rightarrow I$ (SWI vector) ⇒ PC	INH	3F	VSPSSPSsP*	– – – 1 – – – –
*The CPU also uses VSPSSPSsP for hardware interrupts and unimplemented opcode traps.					
TAB	Transfer A to B; (A) ⇒ B	INH	18 0E	OO	– – – – Δ Δ 0 –
TAP	Transfer A to CCR; (A) ⇒ CCR Assembled as TFR A, CCR	INH	B7 02	P	Δ ⇓ Δ Δ Δ Δ Δ Δ
TBA	Transfer B to A; (B) ⇒ A	INH	18 0F	OO	– – – – Δ Δ 0 –
TBEQ *abdxysp,rel9*	Test and branch if equal to 0 If (counter)=0, then (PC)+2+rel ⇒ PC	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	– – – – – – – –
TBL *oprx0_xysppc*	Table lookup and interpolate, 8-bit $(M) + [(B) \times ((M+1) - (M))] \Rightarrow A$	IDX	18 3D xb	ORfffP	– – – – Δ Δ – Δ
TBNE *abdxysp,rel9*	Test and branch if not equal to 0 If (counter) ≠ 0, then (PC)+2+rel ⇒ PC	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	– – – – – – – –
TFR *abcdxysp,abcdxysp*	Transfer from register to register (r1) ⇒ r2r1 and r2 same size $00:(r1) \Rightarrow r2r1$=8-bit; r2=16-bit $(r1_L) \Rightarrow r2r1$=16-bit; r2=8-bit	INH	B7 eb	P	– – – – – – – – or Δ ⇓ Δ Δ Δ Δ Δ Δ
TPASame as TFR CCR ,A	Transfer CCR to A; (CCR) ⇒ A	INH	B7 20	P	– – – – – – – –

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 μs) 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 Core Users Guide.
  - 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 rPf).
  - Section A.27 of the Core Users Guide explains what the HCS12 is doing during each of the different types of CPU cycles.

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

The program executes the `ldab #10` instruction once (which takes one cycle). It then goes through loop 10 times (which has two instructions, on 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 cycles = 50 × 41.7 ns/cycle = 2.08 μs

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	rPO
LDAB *oprx0_xysppc*	IDX	E6 xb	rPf
LDAB *oprx9,xysppc*	IDX1	E6 xb ff	rPO
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