- **AS12 Assembler Directives**
- **A Summary of 9S12 pnstructions**
- **Disassembly of 9S12 op codes**
- Huang Section 1.8, Chapter 2
- MC9S12 V1.5 Core User Guide Version 1.2, Section 12
    - A labels is a name assigned the address of the location counter where ithe label is defined
    - Use of {\tt dc} and {\tt ds} directives
    - A summary of 9S12 instruction
    - How to tell which branch instruction to use

HC12 Assembly Language Programming
Programming Model
Addressing Modes
Assembler Directives
HC12 Instructions
Flow Charts

1. Data Transfer and Manipulation Instructions — instructions which move and manipulate data (HCS12 Core Users Guide, Sections 4.3.1, 4.3.2, and 4.3.3).

• Load and Store—load copy of memory contents into a register; store copy of register contents into memory.

```
LDAA $2000          ; Copy contents of address $2000 into A
STD 0,X             ; Copy contents of D to address X and X+1
```

• Transfer — copy contents of one register to another.

```
TBA                 ; Copy B to A
TFR X,Y             ; Copy X to Y
```

• Exhange — exchange contents of two registers.

```
XGDX                ; Exchange contents of D and X
EXG A,B             ; Exchange contents of A and B
```

• Move — copy contents of one memory location to another.

```
MOVB $2000,$20A0 ; Copy byte at $2000 to $20A0
MOVW 2,X+,2,Y+   ; Copy two bytes from address held
                 ; in X to address held in Y
                 ; Add 2 to X and Y
```

2. Arithmetic Instructions — addition, subtraction, multiplication, divison (HCS12 Core Users Guide, Sections 4.3.4, 4.3.6 and 4.3.10).

```
ABA                 ; Add B to A; results in A
```

```
SUBD $20A1          ; Subtract contents of $20A1 from D
INX                 ; Increment X by 1
MUL                 ; Multiply A by B; results in D
```

3. Logic and Bit Instructions — perform logical operations (HCS12 Core Users Guide, Sections 4.3.8, 4.3.9, 4.3.11 and 4.3.12).

• Logic Instructions

```
ANDA $2000          ; Logical AND of A with contents of $2000
NEG -2,X            ; Negate (2's comp) contents of address (X-2)
LSLA                ; Logical shift left A by 1
```

• Bit manipulate and test instructions—work with one bit of a register or memory.

```
BITA #$08           ; Check to see if Bit 3 of A is set
BSET $0002,#$18     ; Set bits 3 and 4 of address $002
```

4. Data test instructions — test contents of a register or memory (to see if zero, negative, etc.), or compare contents of a register to memory (to see if bigger than, etc.) (HCS12 Core Users Guide, Section 4.3.7).

```
TSTA                ; (A)-0 -- set flags accordingly
CPX #$8000          ; (X) - $8000 -- set flags accordingly
```

5. Jump and Branch Instructions — Change flow of program (e.g., goto, it-then-else, switch-case) (HCS12 Core Users Guide, Sections 4.3.17 and 4.3.18).

```
JMP L1              ; Start executing code at address label L1
BEQ L2              ; If Z bit set, go to label L2
DBNE X,L3           ; Decrement X; if X not 0 then goto L3
BRCLR $1A,#$80,L4   ; If bit 7 of addr $1A clear, go to label L4
```

6. Function Call and Interrupt Instructions — initiate or terminate a subroutine; initiate or terminate and interrupt call (HCS12 Core Users Guide, Sections 4.3.18, 4.3.19).

• Subroutine instructions:

```
JSR sub1            ; Jump to subroutine sub1
RTS                 ; Return from subroutine
```

• Interrupt instructions

```
SWI                 ; Initiate software interrupt
RTI                 ; Return from interrupt
```

7. Load Effective Address Instructions — Put effective address into X, Y or SP (HCS12 Core Users Guide, Section 4.3.22).

```
LEAX 5,Y        ; Put address (Y) + 5 into X
```

8. Condition Code Instructions — change bits in Condition Code Register (HCS12 Core Users Guide, Section 4.3.23).

```
ANDCC #$f0      ; Clear N, Z, C and V bits of CCR
SEV             ; Set V bit of CCR
```

9. Stacking Instructions—push data onto and pull data off of stack (HCS12 Core Users Guide, Section 4.3.21).

```
PSHA            ; Push contents of A onto stack
PULX            ; Pull two top bytes of stack, put into X
```

10. Stop and Wait Instructions — put HC12 into low power mode (HCS12 Core Users Guide, Section 4.3.24).

```
STOP            ; Put into lowest power mode
WAI             ; Put into low power mode until next interrupt
```

11. Instructions we won't discuss or use — BCD arithmetic, fuzzy logic, minimum and maximum, multiply-accumulate, table interpolation (HCS12 Core Users Guide, Sections 4.3.5, 4.3.13, 4.3.14, 4.3.15, 4.3.16).

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

Will the branch be taken?

```
LDAA #$FF    LDAA #$FF
CMPA #$0     CMPA #$0
BLO label1   BLT label2

LDX #$C000   LDX #$C000
CPX #$8000   CPX #$8000
BGT label3   BHI label4
```

Disassembly of an HC12 Program

• It is sometimes useful to be able to convert HC12 op codes into mnemonics.
• For example, consider the hex code:

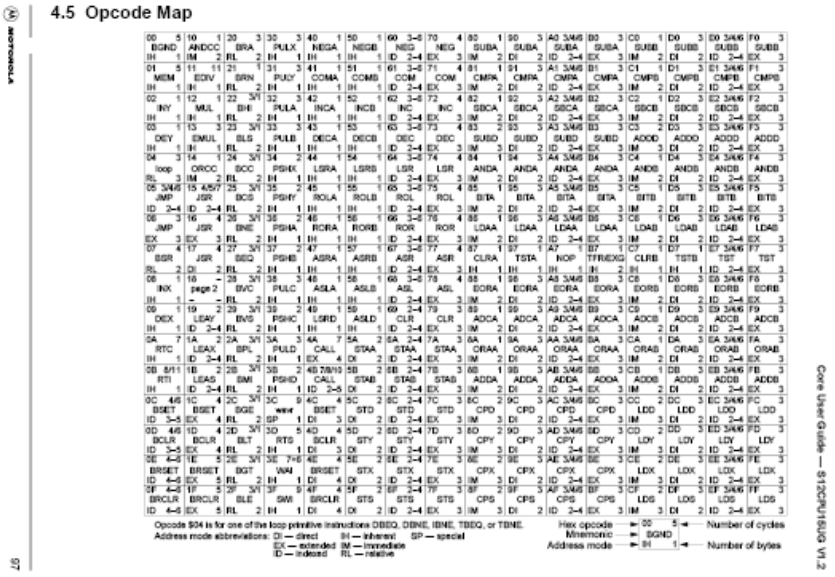
```
ADDR DATA
---- -----------------------------------------------
1000 C6 05 CE 20 00 E6 01 18 06 04 35 EE 3F
```

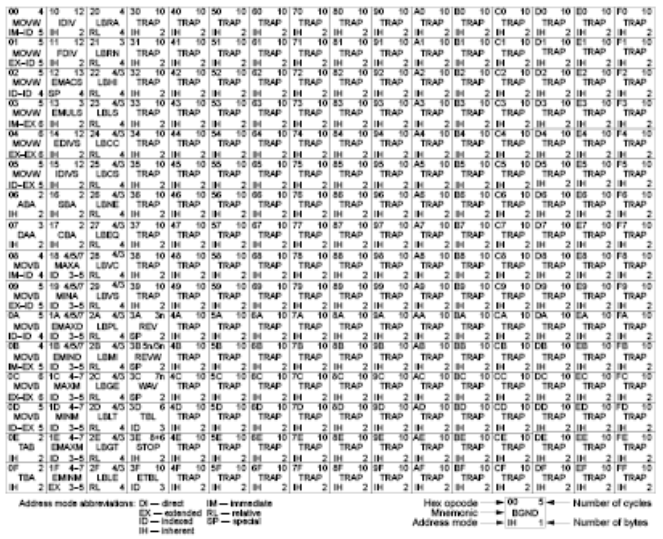
• To determine the instructions, use Table 4.5 of the HCS12 Core Users Guide.
– If the first byte of the instruction is anything other than $18, use Sheet 1 of 2 (Page 97). From this table, determine the number of bytes of the instruction and the addressing mode. For example, $C6 is a two-byte instruction, the mnemonic is LDAB, and it uses the IMM addressing mode. Thus, the two bytes C6 05 is the op code for the instruction LDAB #$05.
– If the first byte is $18, use Sheet 2 of 2 (Page 98), and do the same thing. For example, 18 06 is a two byte instruction, the mnemonic is ABA, and it uses the INH addressing mode, so there is no operand. Thus, the two bytes 18 06 is the op code for the instruction ABA.
– Indexed addressing mode is fairly complicated to disassemble. You need to use Table 4.8 to determine the operand. For example, the op code $E6 indicates LDAB indexed, and may use two to four bytes (one to three bytes in addition to the op code). The postbyte 01 indicates that the operand is 0,1, which is 5-bit constant offest, which takes only one additional byte. All 5-bit constant offset, pre and post increment and decrement, and register offset instructions use one additional byte. All 9-bit constant offset instructions use two additional bytes, with the second byte holding 8 bits of the 9 bit offset. (The 9th bit is a direction bit, which is held in the first postbyte.) All 16-bit constant offset instructions use three postbytes, withe the 2nd and 3rd holding the 16-bit unsigned offset.
– Transfer (TFR) and exchange (EXG) instructions all have the op code $B7. Use Table 4.6 to determine whether it is TFR or an EXG, and to determine which registers are being used. If the most significant bit of the postbyte is 0, the instruction is a transfer instruction.
– Loop instructions (Decrement and Branch, Increment and Branch, and Test and Branch) all have the op code $04. To determine which instruction the op code $04 implies, and whether the branch is positive (forward) or negative (backward), use Table

4.7. For example, in the sequence 04 35 EE, the 04 indicates a loop instruction. The 35 indicates it is a DBNE X instruction (decrement register X and branch if result is not equal to zero), and the direction is backward (negative). The EE indicates a branch of -18 bytes.

• Use up all the bytes for one instruction, then go on to the next instruction.

C6 05	=> LDAA #$05	two-byte LDAA, IMM addressing mode
CE 20 00	=> LDX #$2000	three-byte LDX, IMM addressing mode
E6 01	=> LDAB 1,X	two to four-byte LDAB, IDX addressing mode. Operand 01 => 1,X, a 5b constant offset which uses only one postbyte
18 06	=> ABA	two-byte ABA, INH addressing mode
04 35 EE	=> DBNE X,(-18)	three-byte loop instruction Postbyte 35 indicates DBNE X, negative
3F	=> SWI	one-byte SWI, INH addressing mode





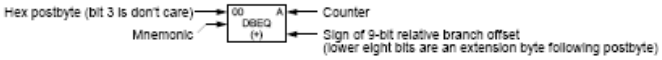
4.6 Transfer and Exchange Postbyte Encoding

Transfers								
↓LS MS→	0	1	2	3	4	5	6	7
0	A⇨A	B⇨A	CCR⇨A	$TMP3_L$⇨A	B⇨A	X_L⇨A	Y_L⇨A	SP_L⇨A
1	A⇨B	B⇨B	CCR⇨B	$TMP3_L$⇨B	B⇨B	X_L⇨B	Y_L⇨B	SP_L⇨B
2	A⇨CCR	B⇨CCR	CCR⇨CCR	$TMP3_L$⇨CCR	B⇨CCR	X_L⇨CCR	Y_L⇨CCR	SP_L⇨CCR
3	sex:A⇨TMP2	sex:B⇨TMP2	sex:CCR⇨TMP2	TMP3⇨TMP2	D⇨TMP2	X⇨TMP2	Y⇨TMP2	SP⇨TMP2
4	sex:A⇨D SEX A,D	sex:B⇨D SEX B,D	sex:CCR⇨D SEX CCR,D	TMP3⇨D	D⇨D	X⇨D	Y⇨D	SP⇨D
5	sex:A⇨X SEX A,X	sex:B⇨X SEX B,X	sex:CCR⇨X SEX CCR,X	TMP3⇨X	D⇨X	X⇨X	Y⇨X	SP⇨X
6	sex:A⇨Y SEX A,Y	sex:B⇨Y SEX B,Y	sex:CCR⇨Y SEX CCR,Y	TMP3⇨Y	D⇨Y	X⇨Y	Y⇨Y	SP⇨Y
7	sex:A⇨SP SEX A,SP	sex:B⇨SP SEX B,SP	sex:CCR⇨SP SEX CCR,SP	TMP3⇨SP	D⇨SP	X⇨SP	Y⇨SP	SP⇨SP
Exchanges								
↓LS MS→	8	9	A	B	C	D	E	F
0	A⇔A	B⇔A	CCR⇔A	$TMP3_L$⇔A $00:A⇨TMP3	B⇔A A⇔B	X_L⇔A $00:A⇔X	Y_L⇔A $00:A⇔Y	SP_L⇔A $00:A⇔SP
1	A⇔B	B⇔B	CCR⇔B	$TMP3_L$⇔B $FF:B⇨TMP3	B⇔B $FF⇨A	X_L⇔B $FF:B⇔X	Y_L⇔B $FF:B⇔Y	SP_L⇔B $FF:B⇔SP
2	A⇔CCR	B⇔CCR	CCR⇔CCR	$TMP3_L$⇔CCR $FF:CCR⇨TMP3	B⇔CCR $FF:CCR⇨D	X_L⇔CCR $FF:CCR⇔X	Y_L⇔CCR $FF:CCR⇔Y	SP_L⇔CCR $FF:CCR⇔SP
3	$00:A⇔TMP2 $TMP2_L$⇔A	$00:B⇔TMP2 $TMP2_L$⇔B	$00:CCR⇔TMP2 $TMP2_L$⇔CCR	TMP3⇔TMP2	D⇔TMP2	X⇔TMP2	Y⇔TMP2	SP⇔TMP2
4	$00:A⇔D	$00:B⇔D	$00:CCR⇔D B⇔CCR	TMP3⇔D	D⇔D	X⇔D	Y⇔D	SP⇔D
5	$00:A⇔X X_L⇔A	$00:B⇔X X_L⇔B	$00:CCR⇔X X_L⇔CCR	TMP3⇔X	D⇔X	X⇔X	Y⇔X	SP⇔X
6	$00:A⇔Y Y_L⇔A	$00:B⇔Y Y_L⇔B	$00:CCR⇔Y Y_L⇔CCR	TMP3⇔Y	D⇔Y	X⇔Y	Y⇔Y	SP⇔Y
7	$00:A⇔SP SP_L⇔A	$00:B⇔SP SP_L⇔B	$00:CCR⇔SP SP_L⇔CCR	TMP3⇔SP	D⇔SP	X⇔SP	Y⇔SP	SP⇔SP

TMP2 and TMP3 registers are for factory use only.

4.7 Loop Primitive Postbyte (lb) Encoding

00 A DBEQ (+)	10 A DBEQ (–)	20 A DBNE (+)	30 A DBNE (–)	40 A TBEQ (+)	50 A TBEQ (–)	60 A TBNE (+)	70 A TBNE (–)	80 A IBEQ (+)	90 A IBEQ (–)	A0 A IBNE (+)	B0 A IBNE (–)
01 B DBEQ (+)	11 B DBEQ (–)	21 B DBNE (+)	31 B DBNE (–)	41 B TBEQ (+)	51 B TBEQ (–)	61 B TBNE (+)	71 B TBNE (–)	81 B IBEQ (+)	91 B IBEQ (–)	A1 B IBNE (+)	B1 B IBNE (–)
02 —	12 —	22 —	32 —	42 —	52 —	62 —	72 —	82 —	92 —	A2 —	B2 —
03 —	13 —	23 —	33 —	43 —	53 —	63 —	73 —	83 —	93 —	A3 —	B3 —
04 D DBEQ (+)	14 D DBEQ (–)	24 D DBNE (+)	34 D DBNE (–)	44 D TBEQ (+)	54 D TBEQ (–)	64 D TBNE (+)	74 D TBNE (–)	84 D IBEQ (+)	94 D IBEQ (–)	A4 D IBNE (+)	B4 D IBNE (–)
05 X DBEQ (+)	15 X DBEQ (–)	25 X DBNE (+)	35 X DBNE (–)	45 X TBEQ (+)	55 X TBEQ (–)	65 X TBNE (+)	75 X TBNE (–)	85 X IBEQ (+)	95 X IBEQ (–)	A5 X IBNE (+)	B5 X IBNE (–)
06 Y DBEQ (+)	16 Y DBEQ (–)	26 Y DBNE (+)	36 Y DBNE (–)	46 Y TBEQ (+)	56 Y TBEQ (–)	66 Y TBNE (+)	76 Y TBNE (–)	86 Y IBEQ (+)	96 Y IBEQ (–)	A6 Y IBNE (+)	B6 Y IBNE (–)
07 SP DBEQ (+)	17 SP DBEQ (–)	27 SP DBNE (+)	37 SP DBNE (–)	47 SP TBEQ (+)	57 SP TBEQ (–)	67 SP TBNE (+)	77 SP TBNE (–)	87 SP IBEQ (+)	97 SP IBEQ (–)	A7 SP IBNE (+)	B7 SP IBNE (–)



4.8 Indexed Addressing Postbyte (xb) Encoding

00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0,X 5b const	−16,X 5b const	1,+X pre-inc	1,X+ post-inc	0,Y 5b const	−16,Y 5b const	1,+Y pre-inc	1,Y+ post-inc	0,SP 5b const	−16,SP 5b const	1,+SP pre-inc	1,SP+ post-inc	0,PC 5b const	−16,PC 5b const	n,X 16b const	n,SP 16b const
01	11	21	31	41	51	61	71	81	91	A1	B1	C1	D1	E1	F1
1,X 5b const	−15,X 5b const	2,+X pre-inc	2,X+ post-inc	1,Y 5b const	−15,Y 5b const	2,+Y pre-inc	2,Y+ post-inc	1,SP 5b const	−15,SP 5b const	2,+SP pre-inc	2,SP+ post-inc	1,PC 5b const	−15,PC 5b const	−n,X 16b const	−n,SP 16b const
02	12	22	32	42	52	62	72	82	92	A2	B2	C2	D2	E2	F2
2,X 5b const	−14,X 5b const	3,+X pre-inc	3,X+ post-inc	2,Y 5b const	−14,Y 5b const	3,+Y pre-inc	3,Y+ post-inc	2,SP 5b const	−14,SP 5b const	3,+SP pre-inc	3,SP+ post-inc	2,PC 5b const	−14,PC 5b const	n,X 16b const	n,SP 16b const
03	13	23	33	43	53	63	73	83	93	A3	B3	C3	D3	E3	F3
3,X 5b const	−13,X 5b const	4,+X pre-inc	4,X+ post-inc	3,Y 5b const	−13,Y 5b const	4,+Y pre-inc	4,Y+ post-inc	3,SP 5b const	−13,SP 5b const	4,+SP pre-inc	4,SP+ post-inc	3,PC 5b const	−13,PC 5b const	[n,X] 16b indir	[n,SP] 16b indir
04	14	24	34	44	54	64	74	84	94	A4	B4	C4	D4	E4	F4
4,X 5b const	−12,X 5b const	5,+X pre-inc	5,X+ post-inc	4,Y 5b const	−12,Y 5b const	5,+Y pre-inc	5,Y+ post-inc	4,SP 5b const	−12,SP 5b const	5,+SP pre-inc	5,SP+ post-inc	4,PC 5b const	−12,PC 5b const	A,X A offset	A,SP A offset
05	15	25	35	45	55	65	75	85	95	A5	B5	C5	D5	E5	F5
5,X 5b const	−11,X 5b const	6,+X pre-inc	6,X+ post-inc	5,Y 5b const	−11,Y 5b const	6,+Y pre-inc	6,Y+ post-inc	5,SP 5b const	−11,SP 5b const	6,+SP pre-inc	6,SP+ post-inc	5,PC 5b const	−11,PC 5b const	B,X B offset	B,SP B offset
06	16	26	36	46	56	66	76	86	96	A6	B6	C6	D6	E6	F6
6,X 5b const	−10,X 5b const	7,+X pre-inc	7,X+ post-inc	6,Y 5b const	−10,Y 5b const	7,+Y pre-inc	7,Y+ post-inc	6,SP 5b const	−10,SP 5b const	7,+SP pre-inc	7,SP+ post-inc	6,PC 5b const	−10,PC 5b const	D,X D offset	D,SP D offset
07	17	27	37	47	57	67	77	87	97	A7	B7	C7	D7	E7	F7
7,X 5b const	−9,X 5b const	8,+X pre-inc	8,X+ post-inc	7,Y 5b const	−9,Y 5b const	8,+Y pre-inc	8,Y+ post-inc	7,SP 5b const	−9,SP 5b const	8,+SP pre-inc	8,SP+ post-inc	7,PC 5b const	−9,PC 5b const	[D,X] D indirect	[D,SP] D indirect
08	18	28	38	48	58	68	78	88	98	A8	B8	C8	D8	E8	F8
8,X 5b const	−8,X 5b const	8,−X pre-dec	8,X− post-dec	8,Y 5b const	−8,Y 5b const	8,−Y pre-dec	8,Y− post-dec	8,SP 5b const	−8,SP 5b const	8,−SP pre-dec	8,SP− post-dec	8,PC 5b const	−8,PC 5b const	n,Y 9b const	n,PC 9b const
09	19	29	39	49	59	69	79	89	99	A9	B9	C9	D9	E9	F9
9,X 5b const	−7,X 5b const	7,−X pre-dec	7,X− post-dec	9,Y 5b const	−7,Y 5b const	7,−Y pre-dec	7,Y− post-dec	9,SP 5b const	−7,SP 5b const	7,−SP pre-dec	7,SP− post-dec	9,PC 5b const	−7,PC 5b const	−n,Y 9b const	−n,PC 9b const
0A	1A	2A	3A	4A	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA
10,X 5b const	−6,X 5b const	6,−X pre-dec	6,X− post-dec	10,Y 5b const	−6,Y 5b const	6,−Y pre-dec	6,Y− post-dec	10,SP 5b const	−6,SP 5b const	6,−SP pre-dec	6,SP− post-dec	10,PC 5b const	−6,PC 5b const	n,Y 16b const	n,PC 16b const
0B	1B	2B	3B	4B	5B	6B	7B	8B	9B	AB	BB	CB	DB	EB	FB
11,X 5b const	−5,X 5b const	5,−X pre-dec	5,X− post-dec	11,Y 5b const	−5,Y 5b const	5,−Y pre-dec	5,Y− post-dec	11,SP 5b const	−5,SP 5b const	5,−SP pre-dec	5,SP− post-dec	11,PC 5b const	−5,PC 5b const	[n,Y] 16b indir	[n,PC] 16b indir
0C	1C	2C	3C	4C	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC
12,X 5b const	−4,X 5b const	4,−X pre-dec	4,X− post-dec	12,Y 5b const	−4,Y 5b const	4,−Y pre-dec	4,Y− post-dec	12,SP 5b const	−4,SP 5b const	4,−SP pre-dec	4,SP− post-dec	12,PC 5b const	−4,PC 5b const	A,Y A offset	A,PC A offset
0D	1D	2D	3D	4D	5D	6D	7D	8D	9D	AD	BD	CD	DD	ED	FD
13,X 5b const	−3,X 5b const	3,−X pre-dec	3,X− post-dec	13,Y 5b const	−3,Y 5b const	3,−Y pre-dec	3,Y− post-dec	13,SP 5b const	−3,SP 5b const	3,−SP pre-dec	3,SP− post-dec	13,PC 5b const	−3,PC 5b const	B,Y B offset	B,PC B offset
0E	1E	2E	3E	4E	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE
14,X 5b const	−2,X 5b const	2,−X pre-dec	2,X− post-dec	14,Y 5b const	−2,Y 5b const	2,−Y pre-dec	2,Y− post-dec	14,SP 5b const	−2,SP 5b const	2,−SP pre-dec	2,SP− post-dec	14,PC 5b const	−2,PC 5b const	D,Y D offset	D,PC D offset
0F	1F	2F	3F	4F	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF
15,X 5b const	−1,X 5b const	1,−X pre-dec	1,X− post-dec	15,Y 5b const	−1,Y 5b const	1,−Y pre-dec	1,Y− post-dec	15,SP 5b const	−1,SP 5b const	1,−SP pre-dec	1,SP− post-dec	15,PC 5b const	−1,PC 5b const	[D,Y] D indirect	[D,PC] D indirect

Hex postbyte → 00
Type of offset → 0,X
5b const

← Source code syntax