- **Addition and Subtraction of Hexadecimal Numbers**
- **Simple assembly language programming**
- Huang, Section 2.2
- **HC12 Addressing Modes**
- Huang, Sections 1.6 and 1.7
    - The N, Z, C and V bits of the Condition Code Register (CCR)
    - Addition and Subtraction of Hex numbers
    - Simple 9S12 programs
    - Hex code generated from a simple 9S12 program
    - Things you need to know for 9S12 assembly language programming
    - HC12 Addressing Modes
    - Inherent, Extended, Direct, Immediate, Indexed, and Relative Modes
    - Summary of 9S12 Addressing Modes

Addition and Subtraction of Hexadecimal Numbers

Setting the C (Carry), V (Overflow), N (Negative) and Z (Zero) bits

How the C, V, N and Z bits of the CCR are changed

Condition Code Register Bits N, Z, V, C

N bit is set if result of operation in negative (MSB = 1)
Z bit is set if result of operation is zero (All bits = 0)
V bit is set if operation produced an overflow
C bit is set if operation produced a carry (borrow on subtraction)
Note: Not all instructions change these bits of the CCR

Addition of Hexadecimal Numbers

C bit set when result does not fit in word
V bit set when P + P = N, N + N = P
N bit set when MSB of result is 1
Z bit set when result is 0

7A	2A	AC	AC
$\underline{+52}$	$\underline{+52}$	$\underline{+8A}$	$\underline{+72}$
CC	7C	36	1E
C: 0	C: 0	C: 1	C: 1
V: 1	V: 0	V: 1	V: 0
N: 1	N: 0	N: 0	N: 0
Z: 0	Z: 0	Z: 0	Z: 0

Subtraction of Hexadecimal Numbers

C bit set on borrow (when the magnitude of the subtrahend is greater than the minuend)
V bit set when N − P = P, P − N = N
N bit set when MSB is 1
Z bit set when result is 0

7A	8A	5C	2C
$\underline{-5C}$	$\underline{-5C}$	$\underline{-8A}$	$\underline{-72}$
1E	2E	D2	BA
C: 0	C: 0	C: 1	C: 1
V: 0	V: 1	V: 1	V: 0
N: 0	N: 0	N: 1	N: 1
Z: 0	Z: 0	Z: 0	Z: 0

Simple Programs for the HCS12

A simple HCS12 program fragment

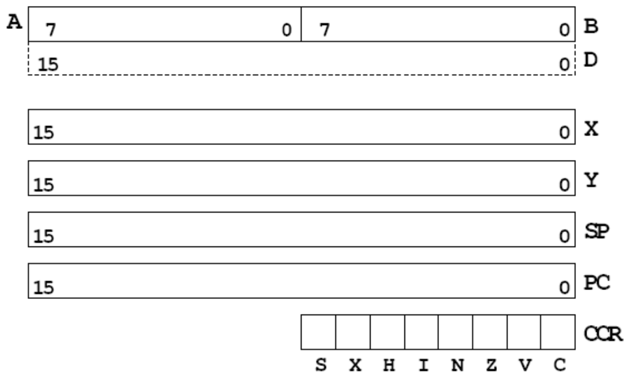
```
        org $1000
        ldaa $2000
        asra
        staa $2001
```

A simple HCS12 program with assembler directives

```
prog:   equ     $1000
data:   equ     $2000

        org     prog
        ldaa    input
        asra
        staa    result
        swi
```

```
         org     data
input:   dc.b    $07
result:  ds.b    1
```

HCS12 Programming Model — The registers inside the HCS12 CPU the programmer needs to know about



How the HCS12 executes a simple program

EXECUTION OF SIMPLE HC12 PROGRAM

0x1000 prog equ $1000
0x1000 org prog
0x1000 b6 20 13 ldaa $2013
0x1003 40 nega
0x1004 7a 20 14 staa $2014
0x1007 3f swi
 	
0x2013 6c
0x2014 94
	PC=0x1000 Control unit (CU) reads B6
	Control decodes B6
	PC=0x1001 CU reads address MSB 20
	PC=0x1002 CU reads address LSB 13
	CU tells memory to fetch contents at address 0x2013
	CU tells ALU to latch value
	
	PC=0x1003 CU reads 40
	CU decodes 40
	CU tells ALU to negate ACCA
	
	PC=0x1004 CU reads 7A
	Control decodes 7A
	PC=0x1005 CU reads address MSB 20
	PC=0x1006 CU reads address LSB 14
	CU fetches value of ACCA from ALU
	CU tells memory to store value at address 0x2014
	
	PC=0x1007

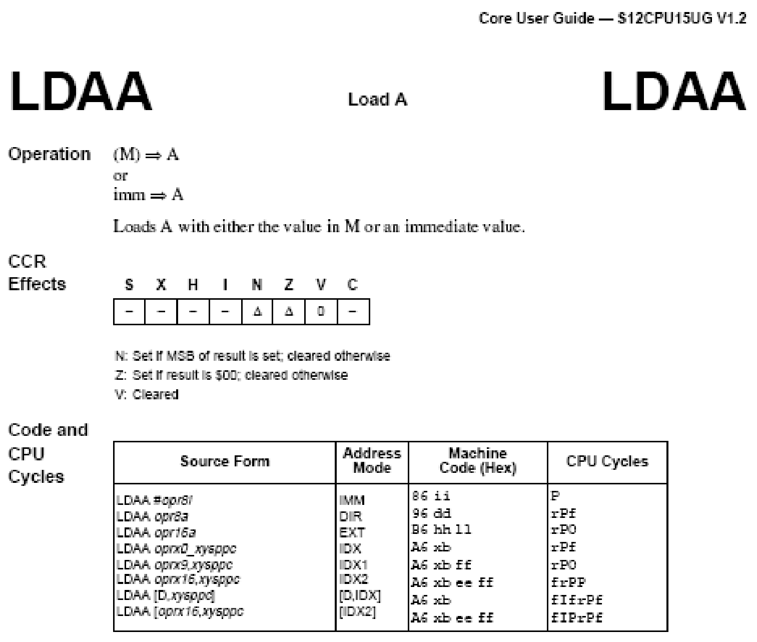
A ______________________________

Things you need to know to write HCS12 assembly language programs

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

Addressing Modes for the HCS12

• Almost all HCS12 instructions operate on memory
• The address of the data an instruction operates on is called the effective address of that instruction.
• Each instruction has information which tells the HCS12 the address of the data in memory it operates on.
• The addressing mode of the instruction tells the HCS12 how to figure out the effective address for the instruction.
• Each HCS12 instructions consists of a one or two byte op code which tells the HCS12 what to do and what addressing mode to use, followed, when necessary by one or more bytes which tell the HCS12 how to determine the effective address.
– All two-byte op codes begin with an $18.
• For example, the LDAA instruction has 4 different op codes, one for each of the 4 different addressing modes.



Core User Guide — S12CPU15UG V1.2

LDAA Load A **LDAA**

Operation (M) ⇒ A
or
imm ⇒ A

Loads A 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
LDAA #*opr8i*	IMM	86 ii	P
LDAA *opr8a*	DIR	96 dd	rPf
LDAA *opr16a*	EXT	B6 hh ll	rPO
LDAA *oprx0_xysppc*	IDX	A6 xb	rPf
LDAA *oprx9,xysppc*	IDX1	A6 xb ff	rPO
LDAA *oprx16,xysppc*	IDX2	A6 xb ee ff	frPP
LDAA [D,*xysppc*]	[D,IDX]	A6 xb	fIfrPf
LDAA [*oprx16,xysppc*	[IDX2]	A6 xb ee ff	fIPrPf

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:

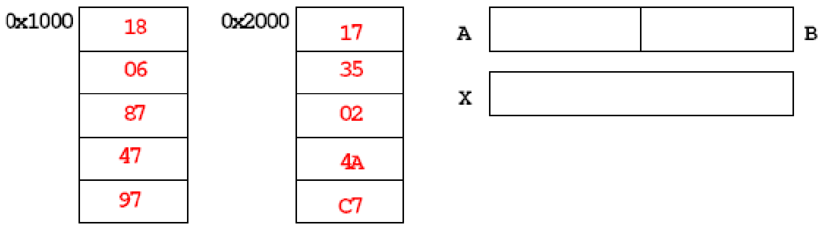
INH Inherent
IMM Immediate
DIR Direct
EXT Extended
REL Relative (used only with branch instructions)
IDX Indexed (won't study indirect indexed mode)

The Inherent (INH) addressing mode

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

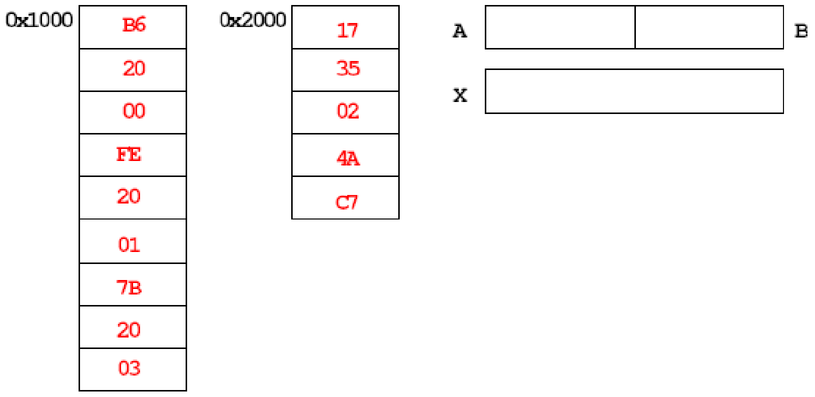


The Extended (EXT) addressing mode

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

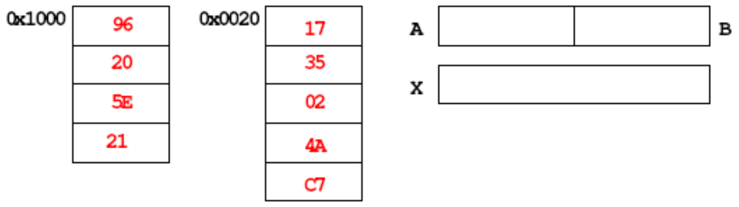


The Direct (DIR) addressing mode

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

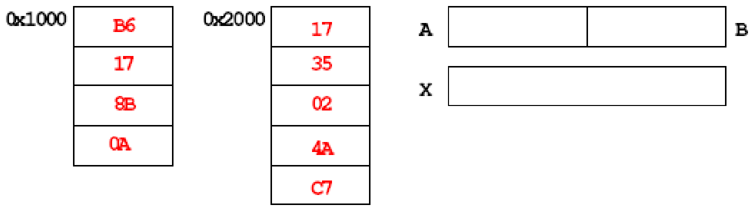


The Immediate (IMM) addressing mode

Value to be used is part of instruction

LDAA #$17	; $17 –> A
B6 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



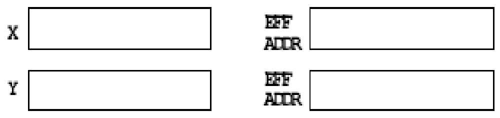
The Indexed (IDX, IDX1, IDX2) addressing mode

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



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

INDEXED ADDRESSING MODES
(Does not include indirect modes)

	Example	Effective Addr	Offset	Value in X	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
Postincrement	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.

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

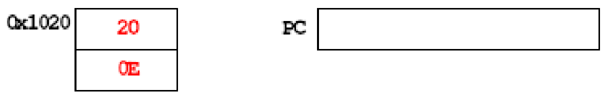
```
BRA 20 35     PC + 2 + 0035 –> PC

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

```
0x1020          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	0x2035
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 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