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

<table>
<thead>
<tr>
<th>7A</th>
<th>2A</th>
<th>AC</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>+52</td>
<td>+52</td>
<td>+8A</td>
<td>+72</td>
</tr>
</tbody>
</table>

CC: 7C
36  1E

C: 0  C: 0  C: 1  C: 1
V: 0  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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7A</td>
<td>8A</td>
<td>5C</td>
<td>2C</td>
<td></td>
</tr>
<tr>
<td>−5C</td>
<td>−5C</td>
<td>−8A</td>
<td>−72</td>
<td></td>
</tr>
<tr>
<td>1E</td>
<td>2E</td>
<td>D2</td>
<td>BA</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C: 0</td>
<td>C: 0</td>
<td>C: 1</td>
<td>C: 1</td>
</tr>
<tr>
<td>V: 0</td>
<td>V: 1</td>
<td>V: 1</td>
<td>V: 0</td>
</tr>
<tr>
<td>N: 0</td>
<td>N: 0</td>
<td>N: 1</td>
<td>N: 1</td>
</tr>
<tr>
<td>Z: 0</td>
<td>Z: 0</td>
<td>Z: 0</td>
<td>Z: 0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

How the HCS12 executes a simple 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

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.

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>CPU Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>3</td>
</tr>
<tr>
<td>Indirect</td>
<td>4</td>
</tr>
<tr>
<td>Indexed</td>
<td>5</td>
</tr>
<tr>
<td>Indexed Indirect</td>
<td>6</td>
</tr>
</tbody>
</table>

LDAA
Load A

Operation: (M) → A
or
 imm → A

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

CCR Effects:

<table>
<thead>
<tr>
<th>S</th>
<th>X</th>
<th>H</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

N: Set if MSB of result is set, cleared otherwise
Z: Set if result is 000, cleared otherwise
V: Cleared

<table>
<thead>
<tr>
<th>Code and CPU Cycles</th>
<th>Source Form</th>
<th>Address Mode</th>
<th>Machine Code (Hex)</th>
<th>CPU Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAA</td>
<td>MN</td>
<td>68 11</td>
<td>00</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>CR</td>
<td>68 88</td>
<td>00</td>
<td>#P9</td>
</tr>
<tr>
<td></td>
<td>EXT</td>
<td>68 11</td>
<td>00</td>
<td>#P10</td>
</tr>
<tr>
<td></td>
<td>EX</td>
<td>68 00</td>
<td>00</td>
<td>#P11</td>
</tr>
<tr>
<td></td>
<td>QU</td>
<td>68 00</td>
<td>00</td>
<td>#P12</td>
</tr>
<tr>
<td></td>
<td>DX</td>
<td>68 00</td>
<td>00</td>
<td>#P13</td>
</tr>
<tr>
<td></td>
<td>QX</td>
<td>68 00</td>
<td>00</td>
<td>#P14</td>
</tr>
<tr>
<td></td>
<td>QY</td>
<td>68 00</td>
<td>00</td>
<td>#P15</td>
</tr>
<tr>
<td></td>
<td>DQ</td>
<td>68 00</td>
<td>00</td>
<td>#P16</td>
</tr>
<tr>
<td></td>
<td>QA</td>
<td>68 00</td>
<td>00</td>
<td>#P17</td>
</tr>
<tr>
<td></td>
<td>EX</td>
<td>68 00</td>
<td>00</td>
<td>#P18</td>
</tr>
<tr>
<td></td>
<td>DX</td>
<td>68 00</td>
<td>00</td>
<td>#P19</td>
</tr>
<tr>
<td></td>
<td>QX</td>
<td>68 00</td>
<td>00</td>
<td>#P20</td>
</tr>
<tr>
<td></td>
<td>QY</td>
<td>68 00</td>
<td>00</td>
<td>#P21</td>
</tr>
<tr>
<td></td>
<td>DQ</td>
<td>68 00</td>
<td>00</td>
<td>#P22</td>
</tr>
<tr>
<td></td>
<td>QA</td>
<td>68 00</td>
<td>00</td>
<td>#P23</td>
</tr>
</tbody>
</table>
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

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 the 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 the effective address is specified by the byte following the 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) 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

----

INDEXED ADDRESSING MODES
(Does not include indirect modes)

<table>
<thead>
<tr>
<th>Example</th>
<th>Effective Addr</th>
<th>Offset</th>
<th>Value in X</th>
<th>Registers to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAA n,x</td>
<td>(X)+n</td>
<td>0 to FFFF</td>
<td>(X)</td>
<td>X,Y,SP,PC</td>
</tr>
<tr>
<td>LDAA –n,x</td>
<td>(X)-n</td>
<td>0 to FFFF</td>
<td>(X)</td>
<td>X,Y,SP,PC</td>
</tr>
<tr>
<td>LDAA n,X+</td>
<td>(X)</td>
<td>1 to 8</td>
<td>(X)+n</td>
<td>X,Y,SP</td>
</tr>
<tr>
<td>LDAA n,X-</td>
<td>(X)</td>
<td>1 to 8</td>
<td>(X)-n</td>
<td>X,Y,SP</td>
</tr>
<tr>
<td>LDAA A,X</td>
<td>(X)+(A)</td>
<td>0 to FF</td>
<td>(X)</td>
<td>X,Y,SP,PC</td>
</tr>
<tr>
<td>LDAA B,X</td>
<td>(X)+(B)</td>
<td>0 to FF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDAA D,X</td>
<td>(X)+(D)</td>
<td>0 to FFFF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 35  PC + 2 + 0035 ⇒ PC

20 35

BRA C7  PC + 2 + C7 ⇒ PC

20 C7  PC + 2 − 39 ⇒ 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

LBEQ 21A  If Z == 1 then PC + 4 + 021A ⇒ PC

18 27 02 1A  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

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Op Code</th>
<th>Effective Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH Inherent</td>
<td>ABA</td>
<td>18 06</td>
<td>None</td>
</tr>
<tr>
<td>IMM Immediate</td>
<td>LDAA #$35</td>
<td>86 35</td>
<td>PC+1</td>
</tr>
<tr>
<td>DIR Direct</td>
<td>LDAA $35</td>
<td>96 35</td>
<td>0x0035</td>
</tr>
<tr>
<td>EXT Extended</td>
<td>LDAA $2035</td>
<td>B6 20 35</td>
<td>0x2035</td>
</tr>
<tr>
<td>IDX Indexed</td>
<td>LDAA 3,X</td>
<td>A6 03</td>
<td>X+3</td>
</tr>
<tr>
<td>IDX Indexed Postincrement</td>
<td>LDAA 3,X+</td>
<td>A6 32</td>
<td>X (X+3 ⇒ X)</td>
</tr>
<tr>
<td>IDX Indexed Preincrement</td>
<td>LDAA 3,+X</td>
<td>A6 22</td>
<td>X+3 (X+3 ⇒ X)</td>
</tr>
<tr>
<td>IDX Indexed Postdecrement</td>
<td>LDAA 3,X-</td>
<td>A6 3D</td>
<td>X (X-3 ⇒ X)</td>
</tr>
<tr>
<td>IDX Indexed Predecrement</td>
<td>LDAA 3,-X</td>
<td>A6 2D</td>
<td>X-3 (X-3 ⇒ X)</td>
</tr>
<tr>
<td>REL Relative</td>
<td>BRA $1050</td>
<td>20 23</td>
<td>PC+2+Offset</td>
</tr>
<tr>
<td></td>
<td>LBRA $1F00</td>
<td>18 20 0E CF</td>
<td>PC+4+Offset</td>
</tr>
</tbody>
</table>

A few instructions have two effective addresses:

- **MOV B $2000,$3000** moves byte from address $2000 to $3000
- **MOV W 0,X,0,Y** moves word from address pointed to by X to address pointed to by Y