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

**ADDITION:**

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></th>
<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>
<td></td>
</tr>
<tr>
<td>OC</td>
<td>7C</td>
<td>36</td>
<td>1E</td>
<td></td>
</tr>
</tbody>
</table>

C: 0 C: 0 C: 1 C: 1
V: 1 V: 0 V: 1 V: 0
N: 1 N: 0 N: 0 N: 1
Z: 0 Z: 0 Z: 0 Z: 0
Subtraction of Hexadecimal Numbers

SUBTRACTION:

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

\[
\begin{array}{cccc}
7A & 8A & 5C & 2C \\
-5C & -5C & -BA & -72 \\
1E & 2E & D2 & BA \\
\end{array}
\]

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       $1001
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

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>X</td>
<td>H</td>
<td>I</td>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
</tr>
</tbody>
</table>

The diagram above shows the registers of the HCS12 CPU, including:

- **B**: Instruction register
- **D**: Data register
- **X**: Accumulator
- **Y**: Index register
- **SP**: Stack pointer
- **PC**: Program counter
- **CCR**: Condition code register

Each register is 16 bits wide, with the least significant 7 bits being the operand and the most significant 9 bits being for addressing.
**How the HCS12 executes a simple program**

**EXECUTION OF SIMPLE HC12 PROGRAM**

<table>
<thead>
<tr>
<th>IDAA $2013</th>
<th>NEGA</th>
<th>STAA $2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>B6</td>
<td></td>
</tr>
<tr>
<td>0x1001</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>0x1002</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>0x1003</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>0x1004</td>
<td>7A</td>
<td></td>
</tr>
<tr>
<td>0x1005</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>0x1006</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>0x2013</td>
<td>6C</td>
<td></td>
</tr>
<tr>
<td>0x2014</td>
<td>5A</td>
<td></td>
</tr>
</tbody>
</table>

| PC = 0x1000 | Control unit reads B6 |
| PC = 0x1001 | Control unit reads address MSB 20 |
| PC = 0x1002 | Control unit reads address LSB 13 |
| PC = 0x1003 | Control units tells memory to fetch contents of address 0x2013 |
| PC = 0x1004 | Control units tells ALU to latch value |
| PC = 0x1005 | Control unit reads 7A |
| PC = 0x1006 | Control unit decodes 7A |
| PC = 0x1007 | Control unit tells ALU to negate ACCA |
| PC = 0x1008 | Control unit reads 40 |
| PC = 0x1009 | Control unit decodes 40 |
| PC = 0x100A | Control unit tells ALU to negate ACCA |
| PC = 0x100B | Control unit reads address MSB 20 |
| PC = 0x100C | Control unit reads address LSB 14 |
| PC = 0x100D | Control units fetches value of ACCA from ALU |
| PC = 0x100E | Control units tells memory to store value at address 0x2014 |

**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 instruction 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
## LDAA

**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>Δ</td>
<td>Δ</td>
<td>0</td>
<td>–</td>
</tr>
</tbody>
</table>

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

**Code and CPU Cycles**

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Machine Code (Hex)</th>
<th>CPU Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAA #opr8i</td>
<td>IMM</td>
<td>86 ii</td>
<td>P</td>
</tr>
<tr>
<td>LDAA opr8a</td>
<td>DIR</td>
<td>96 dd</td>
<td>rPf</td>
</tr>
<tr>
<td>LDAA opr16a</td>
<td>EXT</td>
<td>B6 hh ll</td>
<td>rPO</td>
</tr>
<tr>
<td>LDAA oprx0,xysppc</td>
<td>IDX</td>
<td>A6 xb</td>
<td>rPf</td>
</tr>
<tr>
<td>LDAA oprx9,xysppc</td>
<td>IDX1</td>
<td>A6 xb ff</td>
<td>rPO</td>
</tr>
<tr>
<td>LDAA oprx16,xysppc</td>
<td>IDX2</td>
<td>A6 xb ee ff</td>
<td>frPP</td>
</tr>
<tr>
<td>LDAA [D,xysppc]</td>
<td>[IDX]</td>
<td>A6 xb</td>
<td>fIPrPf</td>
</tr>
<tr>
<td>LDAA [oprx16,xysppc]</td>
<td>[IDX2]</td>
<td>A6 xb ee ff</td>
<td>fIPrPf</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

Inherent (INH) Addressing Mode

Instructions which work only with registers inside ALU

ABA ; Add B to A \((A) + (B) \rightarrow A\)
\[\begin{array}{c}
18 \\
06 \\
\end{array}\]

CLRA ; Clear A \(0 \rightarrow A\)
\[\begin{array}{c}
87 \\
\end{array}\]

ASRA ; Arithmetic Shift Right A
\[\begin{array}{c}
47 \\
\end{array}\]

TSTA ; Test A \((A) - 0x00\) Set CCR
\[\begin{array}{c}
97 \\
\end{array}\]

The HC12 does not access memory

There is no effective address

\[
\begin{array}{|c|c|}
\hline
0x1000 & 18 & 0x2000 & 17 \\
06 & 35 \\
87 & 02 \\
47 & 4A \\
97 & C7 \\
\hline
\end{array}
\]
The *Extended (EXT)* addressing mode

**Extended (EXT) Addressing Mode**

Instructions which give the 16-bit address to be accessed

**LDAA $2000** ; ($2000) → A

\[
\begin{array}{c}
B6 \\
20 \\
00 \\
\end{array}
\]

*Effective Address: $2000*

**LDX $2001** ; ($2001:$2002) → X

\[
\begin{array}{c}
FE \\
20 \\
01 \\
\end{array}
\]

*Effective Address: $2001*

**STAB $2003** ; (B) → $2003

\[
\begin{array}{c}
7B \\
20 \\
03 \\
\end{array}
\]

*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
  
  Effective Address: $0020

- **STX $21** ; (X) -> $0021:$0022
  
  Effective Address: $0021

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

<table>
<thead>
<tr>
<th>0x1000</th>
<th>0x2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>B6</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>35</td>
</tr>
<tr>
<td>8B</td>
<td>02</td>
</tr>
<tr>
<td>0A</td>
<td>4A</td>
</tr>
<tr>
<td></td>
<td>C7</td>
</tr>
</tbody>
</table>

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

INDEXED ADDRESSING MODES
(Does not include indirect modes)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Example</th>
<th>Effective Address</th>
<th>Offset</th>
<th>Value in X After Done</th>
<th>Registers To Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant Offset</td>
<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>Constant Offset</td>
<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>Postincrement</td>
<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>Preincrement</td>
<td>LDAA n, +X</td>
<td>(X)+n</td>
<td>1 to 8</td>
<td>(X)+n</td>
<td>X, Y, SP</td>
</tr>
<tr>
<td>Postdecrement</td>
<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>Predecrement</td>
<td>LDAA n, -X</td>
<td>(X)-n</td>
<td>1 to 8</td>
<td>(X)-n</td>
<td>X, Y, SP</td>
</tr>
<tr>
<td>ACC Offset</td>
<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></td>
<td>LDAA B, X</td>
<td>(X)+(B)</td>
<td>0 to FF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDAA D, X</td>
<td>(X)+(D)</td>
<td>0 to FFFF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
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.

<table>
<thead>
<tr>
<th>Branch instruction: One byte following op code specifies how far to branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>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</td>
</tr>
<tr>
<td><strong>BRA</strong> 20 35</td>
</tr>
<tr>
<td><strong>BRA</strong> 20 C7</td>
</tr>
<tr>
<td><strong>PC + 2 - 0039</strong> → <strong>PC</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Long branch instruction: Two bytes following op code specifies how far to branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>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</td>
</tr>
<tr>
<td><strong>LBEQ</strong> 18 27 02 1A</td>
</tr>
<tr>
<td>If ( Z = 0 ) then <strong>PC + 4</strong> → <strong>PC</strong></td>
</tr>
</tbody>
</table>

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

\[
\begin{align*}
$1020 & \quad \text{BRA} \quad $1030 \quad ; \text{Branch to instruction at address$1030$} \\\n0x1020 & \quad 20 \quad \text{PC} \\
& \quad 0E
\end{align*}
\]
Summary of HCS12 addressing modes

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</td>
<td>ABA</td>
<td>18 06</td>
<td>None</td>
</tr>
<tr>
<td>IMM</td>
<td>LDAA #$35</td>
<td>86 35</td>
<td>PC + 1</td>
</tr>
<tr>
<td>DIR</td>
<td>LDAA $35</td>
<td>96 35</td>
<td>0x0035</td>
</tr>
<tr>
<td>EXT</td>
<td>LDAA $2035</td>
<td>B6 20 35</td>
<td>0x0935</td>
</tr>
<tr>
<td>IDX</td>
<td>LDAA 3,X</td>
<td>A6 03</td>
<td>X + 3</td>
</tr>
<tr>
<td>IDX1</td>
<td>LDAA 30,X</td>
<td>A6 E0 13</td>
<td></td>
</tr>
<tr>
<td>IDX2</td>
<td>LDAA 300,X</td>
<td>A6 E2 01</td>
<td></td>
</tr>
<tr>
<td>IDX</td>
<td>LDAA 3,X+</td>
<td>A6 32</td>
<td>X (X+3 -&gt; X)</td>
</tr>
<tr>
<td>IDX</td>
<td>LDAA 3,+X</td>
<td>A6 22</td>
<td>X+3 (X+3 -&gt; X)</td>
</tr>
<tr>
<td>IDX</td>
<td>LDAA 3,X−</td>
<td>A6 3D</td>
<td>X (X−3 -&gt; X)</td>
</tr>
<tr>
<td>IDX</td>
<td>LDAA 3,−X</td>
<td>A6 2D</td>
<td>X−3 (X−3 -&gt; X)</td>
</tr>
<tr>
<td>REL</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:

- **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