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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7A</td>
<td>2A</td>
<td>AC</td>
<td>AC</td>
</tr>
<tr>
<td>+52</td>
<td>+52</td>
<td>+8A</td>
<td>+72</td>
</tr>
<tr>
<td>CC</td>
<td>7C</td>
<td>36</td>
<td>1E</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

<table>
<thead>
<tr>
<th>7A</th>
<th>8A</th>
<th>5C</th>
<th>2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5C</td>
<td>-5C</td>
<td>-8A</td>
<td>-72</td>
</tr>
</tbody>
</table>

| 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 HC12

A simple HC12 program fragment

```
org       $0800
ldaa      $0900
asra      
staa      $0901
```

A simple HC12 program with assembler operatives

```
prog:     equ       $0800
data:     equ       $0900

CODE:     section    .text
          org        prog
          ldaa       input
          asra
          staa       result
          swi

DATA:     section    .data
          org        data
input:    dc.b       $07
result:   ds.b       1
```
## How the HC12 executes a simple program

### EXECUTION OF SIMPLE HC12 PROGRAM

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Memory Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAA $0913</td>
<td>0x0800</td>
</tr>
<tr>
<td>NEGA</td>
<td>0x0801</td>
</tr>
<tr>
<td>STAA $0914</td>
<td>0x0802</td>
</tr>
<tr>
<td></td>
<td>0x0803</td>
</tr>
<tr>
<td></td>
<td>0x0804</td>
</tr>
<tr>
<td></td>
<td>0x0805</td>
</tr>
<tr>
<td></td>
<td>0x0806</td>
</tr>
<tr>
<td></td>
<td>0x0913</td>
</tr>
<tr>
<td></td>
<td>0x0914</td>
</tr>
</tbody>
</table>

---

### Control Unit Actions

- **PC = 0x0800**: Control unit reads B6
- **PC = 0x0801**: Control unit reads address MSB 09
- **PC = 0x0802**: Control unit reads address LSB 13
- **PC = 0x0803**: Control unit reads 40
- **PC = 0x0804**: Control unit reads 7A
- **PC = 0x0805**: Control unit reads address MSB 09
- **PC = 0x0806**: Control unit reads address LSB 14
- **PC = 0x0807**: Control unit reads 5A

---

### Things you need to know to write an HC12 assembly language program

**HC12 Assembly Language Programming**

- **Programming Model**
- **HC12 Instructions**
- **Addressing Modes**
- **Assembler Directives**
HC12 Programming Model — The registers inside the HC12 CPU the programmer needs to know about

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

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

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SP</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
</tr>
</tbody>
</table>
Addressing Modes for the HC12

- Almost all HC12 instructions operate on data in 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 HC12 the address of the data in memory it operates on.
- The addressing mode of the instruction tells the HC12 how to figure out the effective address for the instruction.

The HC12 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>Inherent</td>
</tr>
<tr>
<td>IMM</td>
<td>Immediate</td>
</tr>
<tr>
<td>DIR</td>
<td>Direct</td>
</tr>
<tr>
<td>EXT</td>
<td>Extended</td>
</tr>
<tr>
<td>IDX</td>
<td>Indexed (won’t study indirect indexed mode)</td>
</tr>
<tr>
<td>REL</td>
<td>Relative (used only with branch instructions)</td>
</tr>
</tbody>
</table>
The *Inherent* (INH) addressing mode

**Inherent (INH) Addressing Mode**

Instructions which work only with registers inside ALU

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABA 18 06</td>
<td>Add B to A  ((A) + (B) \rightarrow A)</td>
<td></td>
</tr>
<tr>
<td>CLRA 87</td>
<td>Clear A 0 (\rightarrow A)</td>
<td></td>
</tr>
<tr>
<td>ASRA 47</td>
<td>Arithmetic Shift Right A</td>
<td></td>
</tr>
<tr>
<td>TSTA 97</td>
<td>Test A ((A) - 0x00 \text{ Set CCR})</td>
<td></td>
</tr>
</tbody>
</table>

The HC12 does not access memory

There is no effective address

![Memory Address Example]

```
0x0800  | 0x0900  | 17  | 35  | 02  | 4A  | C7  |
0x0800  | 0x0900  | 18  | 06  | 87  | 47  | 97  |
```

A | B
---|---
X |
The *Extended* (EXT) addressing mode

Extended (EXT) Addressing Mode

Instructions which give the 16–bit address to be accessed

```
LDAA $0900 ; ($0900) -> A
  B6 09 00   Effective Address: $0900

LDX $0901 ; ($0901:$0902) -> X
  FE 09 01   Effective Address: $0901

STAB $0903 ; (B) -> $0903
  7B 09 03   Effective Address: $0903
```

Effective address is specified by the two bytes following op code

```
0x0800
B6
09
00
FE
09
01
7B
09
03

0x0900
   17
   35
   02
   4A
   C7
```

```
A

X
```
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

Effective address is specified by byte following op code

<table>
<thead>
<tr>
<th>0x0800</th>
<th>96</th>
<th>20</th>
<th>5E</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0020</td>
<td>17</td>
<td>35</td>
<td>02</td>
<td>4A</td>
</tr>
<tr>
<td></td>
<td>C7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The *Immediate* (IMM) addressing mode

**Immediate (IMM) Addressing Mode**

Value to be used is part of instruction

\[
\begin{align*}
\text{LDAA} & \ #17 \quad ; \quad \$17 \rightarrow A \\
& \begin{array}{c}
\text{86}
\end{array}
\begin{array}{c}
17
\end{array} \\
& \text{Effective Address: } PC + 1
\end{align*}
\]

\[
\begin{align*}
\text{ADDA} & \ #10 \quad ; \quad (A) + \$0A \rightarrow A \\
& \begin{array}{c}
\text{8B}
\end{array}
\begin{array}{c}
0A
\end{array} \\
& \text{Effective Address: } PC + 1
\end{align*}
\]

Effective address is the address following the op code

```
0x0800          0x0900
  8B 17 8B 0A
  0x0900 0x0800
```

![Diagram of effective address calculation](image-url)
The **Indexed (IDX) addressing mode**

**Indexed (IDX) Addressing Mode**

Effective address is obtained from X or Y register (or SP or PC)

**Simple Forms**

- **LDA A 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 1
  - **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
(Note: There are several other types we will not discuss)

INDEXED ADDRESSING MODES
(Does not include indirect modes)

<table>
<thead>
<tr>
<th>Type</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>(X)</td>
<td>X, Y, SP, PC</td>
</tr>
<tr>
<td></td>
<td>LDAA D,X</td>
<td>(X)+(D)</td>
<td>0 to FFFF</td>
<td></td>
<td>X, Y, SP, PC</td>
</tr>
</tbody>
</table>

The *Relative (REL)* addressing mode

**Relative (REL) Addressing Mode**

Add offset + 2 (branch) or offset + 4 (long branch) to PC

BRA 20 35  \( \text{PC} + 2 + 0035 \rightarrow \text{PC} \)

BRA 20 C7  \( \text{PC} + 2 + FFC7 \rightarrow \text{PC} \)

PC + 2 − 0039 \( \rightarrow \text{PC} \)

LBEQ 18 27 02 1A  If \( Z = 1 \) then \( \text{PC} + 4 + 021A \rightarrow \text{PC} \)

If \( Z = 0 \) then \( \text{PC} + 4 \rightarrow \text{PC} \)

When writing assembly language program, assembler figures out offset

$0820$  BRA $0830$  ; Branch to instruction at address $0830$

0x0820 20  \text{PC}  

0x0820 0E
## Summary of HC12 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 Inherent</td>
<td>ABA</td>
<td>87</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 $0935</td>
<td>B6 09 35</td>
<td>0x0935</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 Preincrement</td>
<td>LDAA 3,X+</td>
<td>A6 32</td>
<td>X (X+3 -&gt; X)</td>
</tr>
<tr>
<td>IDX Indexed Postincrement</td>
<td>LDAA 3,+X</td>
<td>A6 22</td>
<td>X+3 (X+3 -&gt; X)</td>
</tr>
</tbody>
</table>