HCS12 Architecture
HCS12

• The HCS12 is a family of microcontrollers from Freescale, Inc.
  – Popular for automotive and industrial process control applications
  – Originally developed by Motorola
  – Cost ~$3 – 10

• Features
  – 16 bit CPU
  – On-chip flash memory to hold user programs
  – On-chip timers, I/O ports, A/D
  – On-chip communications interfaces
A and B are general purpose 8-bit accumulators.

A and B concatenated together is a 16-bit accumulator called D.

X,Y are primarily used for indexed addressing (discussed later).

SP is the stack pointer (discussed later).

PC is the program counter. It always holds the address of the next instruction to be fetched.

CCR is the condition code register – its bits are automatically set by the results of instruction execution.
HCS12 Instructions

• Consist of an *opcode* and an *operand*
  – *Opcode* specifies the operation to be performed by the CPU
    • Most opcodes are one byte; some are two bytes (for two byte opcodes, the first byte is always $18$)
    • The opcode also specifies the addressing mode to be used
  – *Operand* is usually a value or a memory address to be operated on
    • Can be zero to 5 bytes, depending on the opcode
    • Opcode and addressing modes determine the meaning of the operand

• Addressing Modes
  – Extended
  – Direct
  – Inherent
  – Immediate
  – Relative
  – Indexed
Instruction Execution Cycle

• One or more read cycles to fetch the opcode
• One or more read cycles to fetch the operand(s) (optional)
• Perform the operation specified by the opcode
• One or more write cycles to write back the result to either a register or a memory location (optional)

Side Note
• The HCS12 executes one instruction at a time ... some instructions take several clock cycles to complete
• When the CPU is performing the operation, it does not need to access memory
• The HCS12 prefetches instructions when the CPU is not accessing memory to speedup the instruction execution process
Addressing Modes

• Extended mode
  – The 16-bit address of the location of to be operated on is given in the instruction
  – Example
    \[
    \text{LDAA }\$1000 \quad ; \quad A \leftarrow [\$1000] \\
    \textit{load A with the contents of location }\$1000
    \]

• Direct mode
  – The location to be operated on must be in the range \$0000-\$00FF, so the address is only 8-bits
  – Example
    \[
    \text{LDAA }\$10 \quad ; \quad A \leftarrow [\$0010] \\
    \textit{load A with the contents of location }\$0010
    \]
Addressing Modes

• **Immediate mode**
  - The value to be operated on is included in the instruction itself
  - Examples
    - `LDAA #$10 ; A <- $10`
      
      load A with the value $10
    - `LDD #$1000 ; D <- $1000`
      
      load D with the value $1000

• **Inherent mode**
  - The instruction has only an opcode, and no operands
  - Example
    - `DECA ; A <- [A] - $01`
      
      decrement A by 1

Machine code is $86 $10

Machine code is $CC $10 $00

opcode

operand

opcode

(no operand)
Load and Store Instructions

• Each register has its own load and store instructions
  - LDAA, STAA  \textit{register A}
  - LDAB, STAB  \textit{register B}
  - LDD, STD    \textit{register D}
  - LDX, STX    \textit{register X}
  - LDY, STY    \textit{register Y}

  Note – since D,X,Y are 16-bit registers, these do a 16-bit load or store, instead of 8-bits

• In addition, each instruction can use different addressing modes
Hand Assembly

• To see how the assembler translates assembly instructions into machine code, look at the instruction set table in Appendix A

• This is also in the CPU12 Reference Manual (ReferenceManualS12CPUV2rev4.pdf) on the course website

• Example: LDAA $10

![Table A-1. Instruction Set Summary (Sheet 7 of 14)](image)

- We can use direct mode since the address is < 256
- The machine code is $96 $10
- Each letter represents one clock cycle – this takes 3 clock cycles
Examples

• Load A with *contents of memory at memory location $80*
  – Assembly instruction
  – Machine code
  – Result of execution

• Load A with *the number $80*
  – Assembly instruction
  – Machine code
  – Result of execution

Assume memory contains:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>01</td>
</tr>
<tr>
<td>0001</td>
<td>01</td>
</tr>
<tr>
<td>0002</td>
<td>01</td>
</tr>
<tr>
<td>0003</td>
<td>01</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>007F</td>
<td>01</td>
</tr>
<tr>
<td>0080</td>
<td>01</td>
</tr>
<tr>
<td>0081</td>
<td>01</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Examples

• Store register X to memory at locations $1000:$1001
  – Assembly instruction
  – Machine code
  – Result of execution
    • assume X initially contains $19AB

• Increment register B
  – Assembly instruction
  – Machine code
  – Result of execution
    • assume B initially contains $1F
## Load and Store Instructions

### Table 1.4 Load and store instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldaa &lt;opr&gt;</td>
<td>Load A</td>
<td>A ← [opr]</td>
</tr>
<tr>
<td>ldab &lt;opr&gt;</td>
<td>Load B</td>
<td>B ← [opr]</td>
</tr>
<tr>
<td>ldd &lt;opr&gt;</td>
<td>Load D</td>
<td>A:B ← [opr]:[opr+1]</td>
</tr>
<tr>
<td>lds &lt;opr&gt;</td>
<td>Load SP</td>
<td>SP ← [opr]:[opr+1]</td>
</tr>
<tr>
<td>ldx &lt;opr&gt;</td>
<td>Load index register X</td>
<td>X ← [opr]:[opr+1]</td>
</tr>
<tr>
<td>ldy &lt;opr&gt;</td>
<td>Load index register Y</td>
<td>Y ← [opr]:[opr+1]</td>
</tr>
<tr>
<td>leas &lt;opr&gt;</td>
<td>Load effective address into SP</td>
<td>SP ← effective address</td>
</tr>
<tr>
<td>leax &lt;opr&gt;</td>
<td>Load effective address into X</td>
<td>X ← effective address</td>
</tr>
<tr>
<td>leay &lt;opr&gt;</td>
<td>Load effective address into Y</td>
<td>Y ← effective address</td>
</tr>
</tbody>
</table>

#### Load Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>staa &lt;opr&gt;</td>
<td>Store A in a memory location</td>
<td>m[opr] ← [A]</td>
</tr>
<tr>
<td>stab &lt;opr&gt;</td>
<td>Store B in a memory location</td>
<td>m[opr] ← [B]</td>
</tr>
<tr>
<td>std &lt;opr&gt;</td>
<td>Store D in a memory location</td>
<td>m[opr]:m[opr+1] ← [A]:[B]</td>
</tr>
<tr>
<td>sts &lt;opr&gt;</td>
<td>Store SP in a memory location</td>
<td>m[opr]:m[opr+1] ← [SP]</td>
</tr>
<tr>
<td>stx &lt;opr&gt;</td>
<td>Store X in a memory location</td>
<td>m[opr]:m[opr+1] ← [X]</td>
</tr>
<tr>
<td>sty &lt;opr&gt;</td>
<td>Store Y in a memory location</td>
<td>m[opr]:m[opr+1] ← [Y]</td>
</tr>
</tbody>
</table>
Add and Subtract instructions

• Add something to a register; the result goes into the register
  – Examples

```
ADDA $1000          ; A <- A + [$1000]
                       add the contents of location $1000 to A
ADDB #$0F           ; B <- B + $0F
                       add $0F to B
```

<table>
<thead>
<tr>
<th>Machine code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BB $10 $00</td>
</tr>
<tr>
<td>$CB $0F</td>
</tr>
<tr>
<td>$B3 $10 $00</td>
</tr>
</tbody>
</table>

• Subtract something from a register; the result goes into the register
  – Example

```
SUBD $1000          ; D <- D - [$1000:$1001]
                       subtract the 16-bit number located at $1000:$1001, from D
```

<table>
<thead>
<tr>
<th>Machine code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B3 $10 $00</td>
</tr>
</tbody>
</table>
## Add and Subtract Instructions

### Add Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>aba</td>
<td>Add B to A</td>
<td>$A \leftarrow [A] + [B]$</td>
</tr>
<tr>
<td>abx</td>
<td>Add B to X</td>
<td>$X \leftarrow [X] + [B]$</td>
</tr>
<tr>
<td>aby</td>
<td>Add B to Y</td>
<td>$Y \leftarrow [Y] + [B]$</td>
</tr>
<tr>
<td>addca &lt;opr&gt;</td>
<td>Add with carry to A</td>
<td>$A \leftarrow [A] + [opr] + C$</td>
</tr>
<tr>
<td>addcb &lt;opr&gt;</td>
<td>Add with carry to B</td>
<td>$B \leftarrow [B] + [opr] + C$</td>
</tr>
<tr>
<td>adda &lt;opr&gt;</td>
<td>Add without carry to A</td>
<td>$A \leftarrow [A] + [opr]$</td>
</tr>
<tr>
<td>addb &lt;opr&gt;</td>
<td>Add without carry to B</td>
<td>$B \leftarrow [B] + [opr]$</td>
</tr>
<tr>
<td>addd &lt;opr&gt;</td>
<td>Add without carry to D</td>
<td>$D \leftarrow [D] + [opr]$</td>
</tr>
</tbody>
</table>

### Subtract Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sba</td>
<td>Subtract B from A</td>
<td>$A \leftarrow [A] - [B]$</td>
</tr>
<tr>
<td>sbca &lt;opr&gt;</td>
<td>Subtract with borrow from A</td>
<td>$A \leftarrow [A] - [opr] - C$</td>
</tr>
<tr>
<td>sbcb &lt;opr&gt;</td>
<td>Subtract with borrow from B</td>
<td>$B \leftarrow [B] - [opr] - C$</td>
</tr>
<tr>
<td>suba &lt;opr&gt;</td>
<td>Subtract memory from A</td>
<td>$A \leftarrow [A] - [opr]$</td>
</tr>
<tr>
<td>subb &lt;opr&gt;</td>
<td>Subtract memory from B</td>
<td>$B \leftarrow [B] - [opr]$</td>
</tr>
<tr>
<td>subd &lt;opr&gt;</td>
<td>Subtract memory from D</td>
<td>$D \leftarrow [D] - [opr]$</td>
</tr>
</tbody>
</table>
Example

• Write instructions to add 3 to the memory locations at $10 and $15

• Approach:
  – A memory location cannot be the destination of an ADD instruction
  – We need to copy the memory value to an accumulator (A or B), add 3 to it, and then store it back
## Exercise

- **Hand assemble**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldaa</td>
<td>$10</td>
<td></td>
</tr>
<tr>
<td>adda</td>
<td>#3</td>
<td></td>
</tr>
<tr>
<td>staa</td>
<td>$10</td>
<td></td>
</tr>
<tr>
<td>ldaa</td>
<td>$15</td>
<td></td>
</tr>
<tr>
<td>adda</td>
<td>#3</td>
<td></td>
</tr>
<tr>
<td>staa</td>
<td>$15</td>
<td></td>
</tr>
</tbody>
</table>
Addressing Modes

- Relative mode
  - Used only by branch instructions
  - The operand is the offset to be added to the program counter
  - Example
    
    ```
    BRA loop ; PC <- [PC] + branch offset
    
    • The opcode is $20
    • The branch offset is the difference between the address of “loop” and the address following the branch instruction
    ```

    Assume this is the address of “loop”

    | Address | Value |
    |---------|-------|
    | $1000   | $20   |
    | $1001   | ??    |
    | $1002   | :     |
    | $1003   | :     |

    The offset is $1010 - $1002 = $0E
    So the machine code is $20 $0E
Move Instructions

• “Move” instructions actually do a “copy” (ie, they don’t change the source location)
• Instead of copying through a register (e.g., LDAA, STAA), you can copy with one “move” instruction

<table>
<thead>
<tr>
<th>Table 1.6 Move instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mnemonic</strong></td>
</tr>
<tr>
<td>movb &lt;src&gt;, &lt;dest&gt;</td>
</tr>
<tr>
<td>movw &lt;src&gt;, &lt;dest&gt;</td>
</tr>
</tbody>
</table>

• Examples

movb $1000, $1001   Copy the byte at location $1000 to location $1001
movb #$05, $3000    Put the number $05 into location $3000
movw $1000, $1002   Copy the 16-bit word at $1000:$1001 to location $1002:$1003
Summary / Questions

• The HCS12 has a small number of internal registers, which are used in machine code instructions ... what are the registers?

• Each instruction (such as “load A”) can have multiple “addressing modes” ... what are the 6 types of addressing modes?

• How do you know how long each instruction will take to execute?
Examples

• Add the contents of memory location $4000 to that of $4010; store the result in $4020.

• Do the same, except do 16-bit addition. Namely locations $4000:$4001 and $4010:$4011 contain 16-bit numbers. The 16-bit result goes into $4020:$4021.
Examples

• Add the contents of memory location $4000 to that of $4010; store the result in $4020.

  ldaa $4000 ; extended addressing
  adda $4010 ; A <- [$4010]+[A]
  staa $4020 ; [A] -> [$4020]

• Do the same, except do 16-bit addition. Namely locations $4000:$4001 and $4010:$4011 contain 16-bit numbers. The 16-bit result goes into $4020:$4021.

  ldd $4000 ; D <- [$4000:4001]
  addd $4010 ; D <- [$4010:4011]+[D]
  std $4020 ; [D] -> [$4020:4021]
Examples

• Add 128 to memory location $4000

• Subtract 2 from memory location $100
Examples

• Add 128 to memory location $4000

  ldab #$80 ; Hex $80 is decimal 128
  adda $4000 ; A <- [$4000]+[A]
  staa $4000

• Subtract 2 from memory location $100

  ldaa $100
  suba #2 ; A <- [A]-2
  staa $100
Examples

• Add $1000 to the 16-bit number stored at memory locations $2000:$2001

• Add the 16-bit number stored at memory locations $1000:$1001, to that stored at $2000:$2001
Examples

• Add $1000 to the 16-bit number stored at memory locations $2000:$2001
  
  ldd #$1000 ; Use immediate mode (#)
  addd $2000
  std $2000

• Add the 16-bit number stored at memory locations $1000:$1001, to that stored at $2000:$2001
  
  ldd $1000 ; No immediate mode (#)
  addd $2000
  std $2000