EECE 218
Microcontrollers

The CPU12 Programmer’s Model
Simple code
CPU12 Programmer’s Model

The registers:

A/B = D: accumulators (data)
X,Y = index (address)
SP = special address (‘stack’)
PC = program counter
CCR = ‘machine state’ reg.
Observations

‘Reading’ into CPU (from memory):
non-destructive for source, data is copied into
CPU register.

‘Writing’ from CPU (into memory):
destructive for destination, data is copied into
memory cell(s).

Instructions:
» Codes that directly control what the CPU does.
» Hex (byte) codes: 1-2 instruction code, 0..4
  operands
» For readability: mnemonics (short, cryptic notation)
First program...

Spec: Add five, 1-byte binary numbers located $1000-1004$ and place the sum into $1005$.

1. Picture

```
$1000:  
   ...  
   ...  
   $1004:  
$1005:  
```

2. Design:

   Input: list of numbers; Output: sum
   Operation: take first, add next, add next, ..., store result
First program…

3. Implementation:
   - Have to keep a running sum: use accumulator (e.g. A)

Pseudo-code: (English like, structured)

```plaintext
clear acc
acc <- acc + [$1000]
acc <- acc + [$1001]
....
acc <- acc + [$1004]
[$1005] <- acc
```
First program...

4. Coding:

Explaination: Mnemonic: Code:
“clear acc” CLRA $87
- Clears all bits in A (puts a 0 into A)

Note:
- Self-contained instruction, no operand
- Other instructions have operands, e.g.
  LDAA $1000 A <- [$1000]
  Code: $B6 10 00
  1st byte: instruction code, 2nd/3rd bytes: operand
  In this case: operand is an address (of data)
First program...

4. Coding: (cont.)

<table>
<thead>
<tr>
<th>Explanation</th>
<th>Mnemonic</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>“acc &lt;- acc + [$1000]”</td>
<td>ADDA</td>
<td>$1000 $BB 10 00</td>
</tr>
<tr>
<td>- Adds the contents of byte at $1000 to A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- We have to repeat this for $1001,$1002,$1003,and $1004.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>“[$1005] &lt;- acc”</td>
<td>STAA</td>
<td>$1005 $7A 10 05</td>
</tr>
<tr>
<td>- Stores the content of A into the byte at $1005.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
First program...

<table>
<thead>
<tr>
<th>Code</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$87</td>
<td>CLRA</td>
</tr>
<tr>
<td>$BB 10 00</td>
<td>ADDA $1000</td>
</tr>
<tr>
<td>$BB 10 01</td>
<td>ADDA $1001</td>
</tr>
<tr>
<td>$BB 10 02</td>
<td>ADDA $1002</td>
</tr>
<tr>
<td>$BB 10 03</td>
<td>ADDA $1003</td>
</tr>
<tr>
<td>$BB 10 04</td>
<td>ADDA $1004</td>
</tr>
<tr>
<td>$7A 10 05</td>
<td>STAA $1005</td>
</tr>
</tbody>
</table>

Note: Code bytes should be stored in memory (e.g. from $2000).

- Left side shows the content of cells $2000..2012.
- Somehow we have to ‘stop’ the computer!
- For now (and in lab): add a ‘SWI’ ($3F) instruction at the end.
Improvements to the code

1. CLRA followed by ADDA $1000:
   It is really:
   LDAA $1000 (load acc A with content of $1000)
   $B6 10 00

2. Iterate over the list! (Organize a loop?)
   Change data in ADD, but use the same instruction!
   ADDA $100_ with _ = 1..4 : must change from cycle to cycle?
   Answer: Addressing modes