EE 3170 Microcontroller Applications

Midterm 1 Review: Miller Chapter 1-3
-The Stuff That Might Be On the Exam

Q. 3.9 of HW3

1 ** Assembler Exercise
2 *
3 16 (4) COAST EQU 22
4 CO15 ORG $0015 (5)
5 CO15 none
6 CO15 TEMP RAB 1
7 CO17 33 VALUE FCH 00110011 (4)
8 CI23 ORG $0123
9 CI23 03 here FCH 8
10 C124 TEMP 1
11 *
12 CO00 ORG $0000
13 CO00 DENT (4) START 14ab VALUE
14 CO00 17 TBC (5)
15 CO00 DB16 (4) ADDA #COAST
16 CO00 D16 (4) STAX SAVE
17 CO07 DEX16 (4) LDX BPG
18 COA ETG1 STAB 1.X
19 COCC 2766 again BNEQ NEXT (6)
20 COOE 5C INC
21 COOP 6A10 (5) DEC 1.X
22 CO11 20FB (6) BAX again
23 CO13 D16 NEXT STAB TEMP

Q. 3.21 of HW3

- START CMPB #6
- BLT NEXT
- CMPB #4
- BRT NEXT
- CLR B
- NEXT SWI/STOP
- END
Q. 3.22 of HW3

- One version might look like this:
  - TAB
  - LDAA #$FF
  - TSTB
  - BMI NEXT
  - CLRA
  - NEXT ....

Coverage & Format

- Chapters 1 - 3
- Some multiple choice
- Some short answer
- Some quantitative questions

Various and Sundry

- Close book/notes
- The relevant pages (instruction table) of HC Reference Manual can be printed and brought to the exam
- You can bring your calculators
  - Not really needed

Chapter 1 / Lectures 2 - 4

- Number representation (lecture 3)
  - Conversion among: decimal, binary, octal, hexadecimal
  - Signed binary numbers, two’s complement
  - Binary Coded Decimal (BCD)
  - Ranges of signed and unsigned numbers
- Number storage
  - Endian-ness: Little endian vs. Big endian (lecture 5)
- Memory Systems (lecture 4)
  - What is the address vs. the data?
  - What are the other signals?
    - Read/Write, Timing
- Fetch/Decode/Execute Cycle (lecture 4)
- Computer vs. Microprocessor vs. microcontroller (lecture 4)
  - Building blocks of a computer: The four major components of a microprocessor
Number representation: example

- Convert 95 into 8-bit binary
  - 0101 1111
- Convert 1101 0011 into a decimal number if it is
  - Unsigned binary: 211
  - 2’s complement: -45
- The range of 8 bits signed number
  - (-128, 127)
- Extend 1101₂ into a 8-bit signed number
  - 1111 1101₂
- The value of 1101 in 2’s complement, S&M
  - -3, -5

Memory Organization

- Memory Organization Questions
- 68HC11
  - 8-bits (1 byte)/word
  - Byte-addressable
  - 16-bit address ⇒ \(2^{16}\) (64k) bytes maximum (hexadecimal expression)
  - Big-endian byte-ordering
  - Little-endian bit-ordering

Memory Interfacing

- \(N = \text{num. of bits in address} = \log_2(\text{num. of words})\)
- \(w = \text{word width} = \text{num. of bits in one word}\)

Number storage: Byte Ordering Illustration

- Example-Store 4-byte BCD number 12345678 starting in memory location $0400$.
- Big endian —Least significant byte has highest address.
  - \(0400\) 12 78
  - \(0401\) 34 56
  - \(0402\) 56 34
  - \(0403\) 78 12
- Little endian —Least significant byte has lowest address.
Memory Organization—Bit Ordering

- Which end of the word is labeled bit 0?
- **Little endian**—Bit 0 is the least significant bit (rightmost bit).
  - Almost all modern processors, including the 6811, use little endian.
- **Big endian**—Bit 0 is the most significant bit (leftmost bit).

<table>
<thead>
<tr>
<th>Mem. Loc (Byte)</th>
<th>Big Endian (Bit)</th>
<th>Little Endian (Bit)</th>
<th>Big Endian (Bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0400</td>
<td>12</td>
<td>0001,0010</td>
<td>0100,1000</td>
</tr>
<tr>
<td>0401</td>
<td>34</td>
<td>0011,0100</td>
<td>0010,1100</td>
</tr>
<tr>
<td>0402</td>
<td>56</td>
<td>0101,0110</td>
<td>0110,1010</td>
</tr>
<tr>
<td>0403</td>
<td>78</td>
<td>0111,1000</td>
<td>0001,1110</td>
</tr>
</tbody>
</table>

Chapter 2 / Lectures 5 & 6s & 7 & 8

- The HC11 programming model
  - Registers, memory, CCR, PC, etc.
- Instructions
  - Logical (AND, OR, EOR, NOT)
  - Shift (Logical, Arithmetic, Roll)
  - Arithmetic
  - Branch
  - Comparison
- Addressing Modes

Four basic components of a computer

Register

- Register organization issues
  - number of registers
  - register uniformity

<table>
<thead>
<tr>
<th>7</th>
<th>A</th>
<th>0</th>
<th>7</th>
<th>B</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>U</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>X</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SP</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>PC</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8-bit accumulators
16-bit accumulator
Index register
Index register
Stack pointer
Program counter
Condition code register
## Condition code questions

### Example 1:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101 0000</td>
<td>1101 0000</td>
</tr>
<tr>
<td>+1001 0001</td>
<td>+1011 0001</td>
</tr>
<tr>
<td>10110 0001</td>
<td>11000 0001</td>
</tr>
</tbody>
</table>

- **C**: 1
- **V**: 1
- **N**: 0
- **Z**: 0

valid for unsigned? | valid for signed?
______________ | ______________ |
| yes | no |

### Example 2: A=5, B=-5

- **R** = A+B = ?
- **C**: 0000
- **V**: 0101
- **N**: 1
- **Z**: 1

valid for unsigned? | valid for signed?
______________ | ______________ |
| yes | no |

### Example 3: A=41, B=-50

- **R** = A+B = ?
- **C**: 1111
- **V**: 1011
- **N**: 1
- **Z**: 1

valid for unsigned? | valid for signed?
______________ | ______________ |
| yes | no |

### Example 3: A=120, B=136

- **R** = A+B = ?
- **C**: ?
- **V**: ?
- **N**: ?
- **Z**: ?

valid for unsigned? | valid for signed?
______________ | ______________ |
| yes | no |

## Instructions and Addressing

### Load and store instructions

- direct addressing (8-bit address)
- extended addressing (16-bit address)
- immediate data

## Exercise

- Originally A holds 7D and B holds 6E.
- Find the contents of A, B and memory after the sequence:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldna #16</td>
<td>A=$10</td>
<td></td>
</tr>
<tr>
<td>ldab $1102</td>
<td>B=$71</td>
<td></td>
</tr>
<tr>
<td>staa $1100</td>
<td>(1100)=$10</td>
<td></td>
</tr>
<tr>
<td>ldd $1100</td>
<td>D=$1054</td>
<td></td>
</tr>
</tbody>
</table>
Logical operations

- and, or, eor
  \[ \text{regA} = \begin{array}{ccccccc} 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ \text{regB} = \begin{array}{ccccccc} 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{array} \end{array} \]

- and may be used to isolate a bit, usually for testing
  - Example: \( \text{anda} \) #$04 isolates bit 2
    - bits are numbered from 0 at the right
  - and may be used to clear a bit
    - Example: \( \text{anda} \) #$F7 clears bit 3
    - the other bits are not affected
- or may be used to set a bit
  - Example: \( \text{oraa} \) #$40 sets bit 6
  - the other bits are not affected

Logic operations: example

- Write assembly software that toggles PORTA bits 3,4
  \[
  \begin{align*}
  \text{ldaa PORTA} \\
  \text{eora} \ #0C \quad *\% 0000 \ 1100 \\
  \text{staa PORTA}
  \end{align*}
  \]

6811 Shift Instructions

- For register A
  - asla arithmetic shift left
  - asra arithmetic shift right
  - lsia logical shift left
  - lsra logical shift right
  - rola rotate left
  - rora rotate right
- Registers B and D and memory have similar instructions.
- Shifting into carry allows us to check carry bit.

Arithmetic operations

- Arithmetic Operations
  - instructions and results
  - including validity of results
- Example: Add $80 and $01
  \[
  \begin{align*}
  10000000 + 00000001 &= 10000001 \\
  \text{Sum is $81. } \ C = 0 \ N = 1 \ V = 0 \ Z = 0
  \end{align*}
  \]
  - Unsigned interpretation: 128 + 1 = 129 valid
  - Signed interpretation: -128 ++1 = -127 valid
**Architecture Example**

- Explain the execution of a 3-Byte Instruction (4 Ticks) using the processor model as shown (slides 22-27 of lecture 5): **LDAA C200**
  - **Fetch Phase**
    - Fetch Opcode, Decode Opcode (need 2 more bytes) and Increment PC-- Tick 1
    - Fetch 2nd byte and Increment PC-- Tick 2
    - Fetch 3rd byte and Increment PC-- Tick 3
  - **Execute Phase**
    - Execute the instruction-- Tick 4

**Excise: Loaded Object Program and Data**

- **ldaa C20B**
- **adda C20C**
- **staa C20D**
- **stop**

**Addressing Mode Summary**

- **68HC11 Operand Formats:**
  - *Inherent* = obvious from the opcode (there is no operand)
  - *Immediate* = precede value with a "#" (e.g. #35 #23)
  - *Extended* = just write the address (e.g. 266 $010A)
  - *Direct* = just write the address (like extended), but:
    - If the address is <256
    - Then the assembler chooses direct instead of extended
  - *Indexed* = write "offset, Index-Register" (e.g. $AF, X;)
  - *Relative* = just write the symbolic address
    - instructions using relative addressing have no options
    - obvious from the opcode

**Addressing example**

- **X=$2000**
- **Y=$2080**
- **A=$45**
- **B=$67**
- For each of the instruction, specify EA, result, memory location and value:
  - **staa 40, x** (2040)=$45
  - **std 66, y** (20E6)=$45, (20E7)=$67
Review of instructions

- **Load/store**
  - `lda`, `ldab`, `lde`, `ldx`, `ldy`
  - `staa`, `stab`, `std`, `sts`, `stx`, `sty`

- **Arithmetic**
  - `adda`, `addb`, `addd`
  - `suba`, `subb`, `subd`

- **Logical**
  - `anda`, `andb`, `oraa`, `orab`, `eora`, `eorb`

- **Shifts of three types**
  - `asla`, `asra`, `lsla`, `lsra`, `rola`, `ror`a`

- **Comparison and test instructions**
  - `cmpa cmpb`, `cpd`, `cpx`, `cpy`
  - `tst (memory)`, `tsta`, `tstb`

- **Branch instructions**

- **Increment/decrement**

Condition Code and Branch

- **C = Carry-out bit**
  - Meaning:
    - `C=1`: Previous instruction had a carry-out
    - `C=0`: Previous instruction didn’t have a carry-out
  - Usage:
    - `C=1 ⇒ Overflow for unsigned arithmetic only`
    - `Ignore C bit for signed arithmetic`
  - Relevant Branch opcodes:
    - `BCC`: Branch if Carry Clear (C=0)
    - `BCS`: Branch if Carry Set (C=1)

- **V = Overflow bit (2’s complement)**
  - Meaning:
    - `V=1`: Previous instruction had an overflow
    - `V=0`: Previous instruction didn’t have an overflow
  - Usage:
    - `V=1 ⇒ Overflow for signed arithmetic only`
    - `Ignore V bit for unsigned arithmetic`
  - Relevant Branch opcodes:
    - `BVC`: Branch if Overflow Clear (V=0)
    - `BVS`: Branch if Overflow Set (V=1)

- **Z = Zero bit**
  - Meaning:
    - `Z=1`: Previous instruction result = 0
    - `Z=0`: Previous instruction result ≠ 0
      - watch these; they are counter-intuitive
  - Usage:
    - `Z=1 ⇒ the zero condition is true`
    - `all bits of the result are zeros`
  - Relevant Branch opcodes:
    - `BNE`: Branch if result not equal to zero (Z=0)
    - `BEQ`: Branch if result is equal to zero (Z=1)
Condition Code and Branch

- **N = Negative bit**
  - Meaning:
    - N=1: Previous instruction result < 0
    - N=0: Previous instruction result ≥ 0
  - Usage:
    - N=1 ⇒ Left-hand bit = 1 (sign-bit)
    - Evaluates 2’s complement negative
  - Relevant Branch opcodes:
    - BPL: Branch if result sign is plus (N=0) actually branches if result is not-negative
    - BMI: Branch if result sign is minus (N=1)

---

Chapter 3 / Lectures 9 &10

- Process of Assembly Language Program Development
- Labels
  - Rules, what are they used for
- Assembler Directives (Pseudo-ops)
  - What do they do?
  - How are they different?
- The Assembler
  - How many passes?
  - Location Counter
  - Symbol Table
- Be able to hand-assemble individual instructions (probably not whole programs).