Lecture 8:
Assembly Language
Today’s Topics

• Review the concept of memories and registers (accumulators)
• Generating machine code manually.
  ▪ You are expected to convert assembly code lines to machine codes.
• Files and processes associated with converting assembly source code to machine code
• To learn assembler directives
A Short Story

- World is converging again.
  - No pure EE, CE, and CS.

- Fundamentals are always important.

- A short story about hiring interviews.
  - Very simple but fundamental questions are asked even to Senior engineers.

- Grade or GPA is important but...

- Last but not least, you are expected to study outside classroom. Lectures cannot cover all topics.
Microcontrollers (or Microcomputers)

Basic ideas

- **Microcontroller**
  - CPU core + I/O ports + Memories (RAM and ROM) + ...

- **Memories**
  - We only use RAM area to learn assembly language and test programs.
  - No need to worry about burning your program into ROM.

- **Registers (accumulators) vs. memories**
  - Registers are small read/write memory cells inside CPU core.
  - Memories are located outside CPU.
  - To get a value from a location in a memory, the value should travel through data bus. (Remember memory modules are separated from CPU core)
    - This takes time (much longer than getting from/setting to Registers)
Before arithmetic operations including comparisons, the microcontroller requires a value on a register to do the operations.
As a programmer, you may think yourself as a master of a small world. You can do whatever you want as long as the environment supports for you.

Before arithmetic operations including comparisons, the microcontroller requires a value on a register to do the operations.

When it comes to the environment, you have two main tools for now (bunch of other things will come soon); a storage to save data; and a process unit to manipulate the data/to conduct arithmetic operations and logical decisions.

To control the processor and manipulate the memory, we write programs.
**Code Line and Program Counter**

- When we say **Code Line** in Lab assignments, quizzes, and exams, the instruction line is completed (executed). So registers are supposed to be affected by the execution.

- **Program Counter** always points the NEXT instruction!!
  - Caution on Branches. PC depends on whether the branch is taken or not.

- **Example:**

<table>
<thead>
<tr>
<th>Trace</th>
<th>Line</th>
<th>PC</th>
<th>A</th>
<th>B</th>
<th>X</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1503</td>
<td>-</td>
<td>-</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1508</td>
<td>-</td>
<td>-</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>150A</td>
<td>-</td>
<td>02</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>150C</td>
<td>-</td>
<td>02</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>150E</td>
<td>40</td>
<td>02</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1511</td>
<td>40</td>
<td>02</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>1513</td>
<td>40</td>
<td>02</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1516</td>
<td>40</td>
<td>02</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1517</td>
<td>40</td>
<td>02</td>
<td>2001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1518</td>
<td>40</td>
<td>01</td>
<td>2001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## Machine Code

Manually generate machine code*

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>CE 2000</td>
<td>LDX #$2000</td>
</tr>
<tr>
<td>1503</td>
<td>180B FF 1000</td>
<td>MOVB #$FF, $1000</td>
</tr>
<tr>
<td>1508</td>
<td>C6 02</td>
<td>LDAB #2</td>
</tr>
<tr>
<td>150A</td>
<td>27 0E</td>
<td>BEQ 14</td>
</tr>
<tr>
<td>150C</td>
<td>A6 00</td>
<td>LDAA 0,X</td>
</tr>
<tr>
<td>150E</td>
<td>B1 1000</td>
<td>CMPA $1000</td>
</tr>
<tr>
<td>1511</td>
<td>24 03</td>
<td>BHS 3</td>
</tr>
<tr>
<td>1513</td>
<td>7A 1000</td>
<td>STAA $1000</td>
</tr>
<tr>
<td>1516</td>
<td>08</td>
<td>INX</td>
</tr>
<tr>
<td>1517</td>
<td>53</td>
<td>DECB</td>
</tr>
<tr>
<td>1518</td>
<td>20 F0</td>
<td>BRA -16</td>
</tr>
<tr>
<td>151A</td>
<td>3F</td>
<td>SWI</td>
</tr>
</tbody>
</table>
Assembly Process

General case

Assembly source code: prog.asm

Assembler

Cross-assembler:
Translate assembly code into object code.

Object code:
Mix of machine code and additional information

Listing file:
Human readable log file containing the original assembly and the machine code.

Linker:
Combine multiple object files into a single piece of machine code

Machine code file

Loader

Microcontroller
Assembly Process

In the lab

Assembly source code: prog.asm

Assembler

Cross-assembler:
Translate assembly code into object code.

Listing file:
Human readable log file containing the original assembly and the machine code.

S19 file:
Motorola S-record format
An ASCII encoding for binary data.

Loader:
D-Bug12 is used to load .s19 file to the memory of the microcontroller. Load instruction is used.
Proper Assembly Code

• 1. Separate the source code into constant section, data and variable sections, and code section.

• 2. Do not use numbers within the code
  ▪ Except for possibly 0 or 1 in obvious situation

• Always begin with a comment block stating
  ▪ Purpose of the program
  ▪ Inputs
  ▪ Outputs
  ▪ Programmer
  ▪ Anything else useful

• Comment within the code

• Assume that a reader understands the processor’s assembly code, so do not use comments to simply rephrase the assembly code.
Assembly Language Syntax

- The assembly language consists of lines of text in the form:
  - [label:] [command] [operand(s)] [;comment]
  or
  - ; comment

- where ‘:’ indicates the end of label and ‘;’ defines the start of a comment. The *command* field may be an instruction, a directive or a macro call.
Assembler Directives

Only small fractions...

- **Memory allocation**
  - **org**: puts an address into the assembler location counter
    - org $1000
  - **ds[size]** (Define Storage)
    ; allocate specified number of storage spaces
    ds.b (Define Storage Bytes)
    ds.w (Define Storage Words)
    - buffer ds.b 100
    - dbuffer ds.w 100

- **Data formation**
  - **dc[size]** (Define Constant)
    ; allocate and initialize storage for variables
    dc.b (Define Constant Byte)
    dc.w (Define Constant Word)
    - array dc.b $11, $12, $22
    - Initialize a 3-byte constant with the data

- **Note: need to know a difference between**
  - buffer ds.b $10
    - allocate 16 byte memory space for ‘buffer’
  - buffer dc.b $10
    - allocate 1 byte for ‘buffer’ and the value of ‘buffer’ will be initialized to $10
Assembler Directives

• Symbol Definition
  • equ
    • TRUE equ $FF

• Numbers
  • $xx or xxh: hexadecimal
  • %xxxx..: binary
  • Otherwise: decimal
Some More Instructions

Load Effective Address Instructions

• Load effective address instructions
  ▪ LEAX: Load effective address into X
    • LEAX 10, X
  ▪ LEAY: Load effective address into Y
    • LEAY B, Y
  ▪ LEAS: Load effective address into SP
    • LEAS 0, PC

• Can you tell a difference between "LEAX 10, X" and "LDX 10, X"?
  ▪ Assuming (X) = 1200, the content at 120A is 34h, and at 120B is 56h
    • “LEAX 10, X” makes X be 120A
    • “LDX 10, X” makes X be the content at memory (120A+120B) which is 3456.
Some More Instructions

Addition and Subtraction

- **8 bit addition**
  - ABA: \((A) + (B) \rightarrow A\); Note that there is no AAB instruction!
  - ADDA: \((A) + (M) \rightarrow A\)
    - ADDA $1000$
  - ADDB: \((B) + (M) \rightarrow B\)
    - ADDB #10
  - ADCA: \((A) + (M) + C \rightarrow A\)
  - ADCB: \((B) + (M) + C \rightarrow B\)

- **8 bit subtraction**
  - SBA: \((A) - (B) \rightarrow A\); Subtract B from A (Note: not SAB instruction!)
  - SUBA: \((A) - (M) \rightarrow A\); Subtract M from A
  - SUBB: \((B) - (M) \rightarrow B\)
  - SBCA: \((A) - (M) - C \rightarrow A\)
  - SBCB: \((B) - (M) - C \rightarrow B\)

- **16 bit addition and subtraction**
  - ADDD: \((A:B) + (M:M+1) \rightarrow A:B\)
  - SUBD: \((A:B) - (M:M+1) \rightarrow A:B\)
  - ABX: \((B) + (X) \rightarrow X\)
  - ABY: \((B) + (Y) \rightarrow Y\)

There is a pattern that make you easy to remember the instructions!!!

1. The last letter in these instructions is the destination!
2. Also it comes to the first in the operation
Some More Instructions

Increments, Decrements, and Negate

- **Increments**
  - INC: (M) + 1 → M
  - INCA: (A) + 1 → A
  - INCB
  - INS
  - INX
  - INY

- **Decrements**
  - DEC
  - DECA
  - DECB
  - DES
  - DEX
  - DEY

- **Negate**
  - NEG: negate a memory byte
  - NEGA
  - NEGB

Note that we don’t have IND and DED!
Assembly Code Example

Requirements

- Write a program to copy a table of one-byte values.
- Our table will be defined by a starting address, supplied at $1000, and by a one-byte number of elements in the table, supplied at $1002.
- The table will be copied a given distance from the original table, and this two-byte offset will be supplied at address $1003.
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>=00001000</td>
<td>ORG $1000</td>
</tr>
<tr>
<td>2:</td>
<td>1000 +0002</td>
<td>table ds.w 1</td>
</tr>
<tr>
<td>3:</td>
<td>1002 +0001</td>
<td>length ds.b 1</td>
</tr>
<tr>
<td>4:</td>
<td>1003 +0002</td>
<td>offset ds.w 1</td>
</tr>
<tr>
<td>5:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td>=00001800</td>
<td>ORG $1800</td>
</tr>
<tr>
<td>7:</td>
<td>1800 FE 1000</td>
<td>LDX table</td>
</tr>
<tr>
<td>8:</td>
<td>1803 B7 54</td>
<td>TFR X,D</td>
</tr>
<tr>
<td>9:</td>
<td>1805 F3 1003</td>
<td>ADDD offset</td>
</tr>
<tr>
<td>10:</td>
<td>1808 B7 46</td>
<td>TFR D,Y</td>
</tr>
<tr>
<td>11:</td>
<td>180A F6 1002</td>
<td>LDAB length</td>
</tr>
<tr>
<td>12:</td>
<td>180D 27 09</td>
<td>loop BEQ done</td>
</tr>
<tr>
<td>13:</td>
<td>180F 180A 00 40</td>
<td>MOVB 0,X,0,Y</td>
</tr>
<tr>
<td>14:</td>
<td>1813 08</td>
<td>INX</td>
</tr>
<tr>
<td>15:</td>
<td>1814 02</td>
<td>INY</td>
</tr>
<tr>
<td>16:</td>
<td>1815 53</td>
<td>DECB</td>
</tr>
<tr>
<td>17:</td>
<td>1816 20 F5</td>
<td>BRA loop</td>
</tr>
<tr>
<td>18:</td>
<td>1818 3F</td>
<td>done SWI</td>
</tr>
</tbody>
</table>

Symbols:
- done *00001818
- length *00001002
- loop *0000180d
- offset *00001003
- table *00001000
Modification of the Example

• What changes are required to handle a table of two-byte numbers?
  ▪ Need to copy two bytes instead of one byte.

• What changes are required to handle a two-byte length?
  ▪ The length should represent two-byte numbers!
ORG $1000

table ds.w 1
length ds.b 1
offset ds.w 1

ORG $1800

LDX table
TFR X,D
ADDD offset
TFR D,Y
!

LDAB length
LDAB length
LDAB length
!

INX
INY
DECB
BRA loop
SWI
!

Symbols:
done *00001818
length *00001002
loop *0000180d
offset *00001003
table *00001000

SUBD #1
Questions?
Wrap-up

What we’ve learned

• Registers and memories
• Generating machine code manually.
• Concept of assembly language
• Assembler directives
What to Come

- Flowcharts
- Some assembly programming examples