**CSC 325 Computer Architecture**

**Class 22 Lab 10**

**March 30, 2015**

*You may work with your lab partner on the exercises of this lab assignment.*

**1**. **Programming the Basic Computer**

Recall the PowerPoint material from Class 17, slides 11-13 and 29, which provides an introduction to assembly language programming for the basic computer. In that first example, you saw how to add two numbers. We can apply the same approach for another simple program that uses subtraction. However, the basic computer does not have a subtract operation. How can we accomplish subtraction? Simple algebra supports the operation:

difference = 92 – (-23)

difference = minuend – subtrahend

difference = minuend + (- subtrahend)

difference = 92 + (- - 23) = 92 + 23 = 115

Recall that to negate a number in two’s complement, complement the number by toggling the bits and adding 1. Here is a simple example:

4 = 0100

-4 = 1011 + 1 = 1100 = (-8 + 4) = -4

*Program to subtract minuend – subtrahend*

ORG 100 /locate program at addr. 100 *ORG is an assembler directive*

LDA SUB /load subtrahend in AC *SUB is a label of a word in mem*

CMA /complement subtrahend in AC

INC /form 2’s complement of subtrahend

ADD MIN /add minuend to AC *MIN is a label of a word in mem*

STA DIF /store the difference *DIF is a label of a word in mem*

HLT /stop execution of program

MIN, DEC 92 /minuend *DEC is a directive, indicates data*

SUB, DEC -23 /subtrahend *is given in decimal format*

DIF, HEX 0 /location for difference result *HEX indicates a hexadecimal format*

END /end of symbolic program

Notice the syntax of the label field: three characters followed by a comma.

When programming in a high level language, such as C++ or Java, the conversion from decimal numbers to binary equivalents occurs behind the scenes. However, for your system, you must hand assemble the code and data.

Though the directives for the representation of the data are given above, in order to assemble this code, you need to make the conversion into binary.

An easy method to convert from decimal to binary is to use repeated division by 2. The remainders at each step are noted. When the division results in 0, then the remainders are collected from the last of the steps to the top (first) step. This gives the binary representation of the starting number.

Recall that dividend = quotient / divisor. At the first step, the quotient is divided by the divisor of 2. The dividend is noted in the next line, and the remainder is recorded. The process repeats with the dividend.

2 | 92

46 0

and repeat the process. Here is the entire conversion:

2 | 92

2 | 46 0

2 | 23 0

2 | 11 1

2 | 5 1

2 | 2 1

2 | 1 0

2 | 0 1

0

By collecting the remainders (from the bottom to the top), we get

1011100

This is just 7 bits. However, an 8 bit representation would pad the more significant bits with zero. Next, you can check your conversion:

What is the binary representation of 9210 ?

92 = 64 + 16 + 8 + 4 = 26 + 24 + 23+ 22 = 01011100

As you know, this number is a positive number, as the most significant bit is a 0. Next, you need to consider how to handle a negative number.

What is the binary representation of -2310 ?

Convert the absolute value of 23, then complement and add 1

2 | 23

2 | 11 1

2 | 5 1

2 | 2 1

2 | 1 0

* 1

Collect the remainders to get 10111. Pad this to 8 bits (or whatever width word you are working with) to get 00010111. This represents the absolute value of the number, which is then converted to a negative number by complementing it and adding 1.

00010111

11101000 toggle the bits (complement)

 + 00000001

11101001

Check the work:

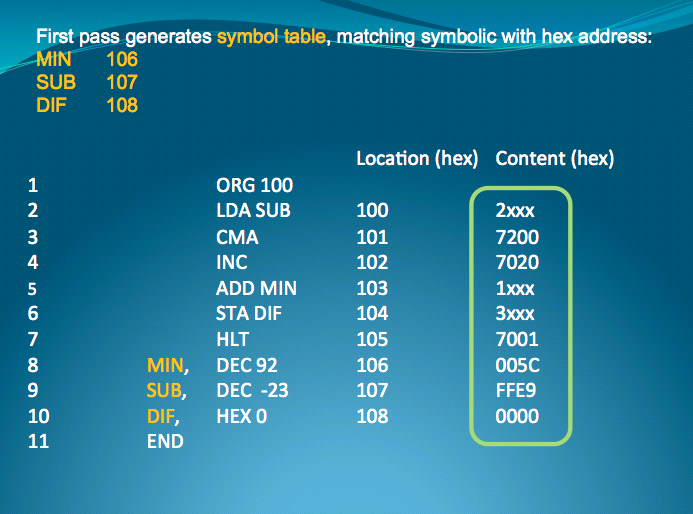
11101001 = -27 + 26 + 25 + 23 + 20 = -128 + 64 + 32 + 8 + 1 = -23

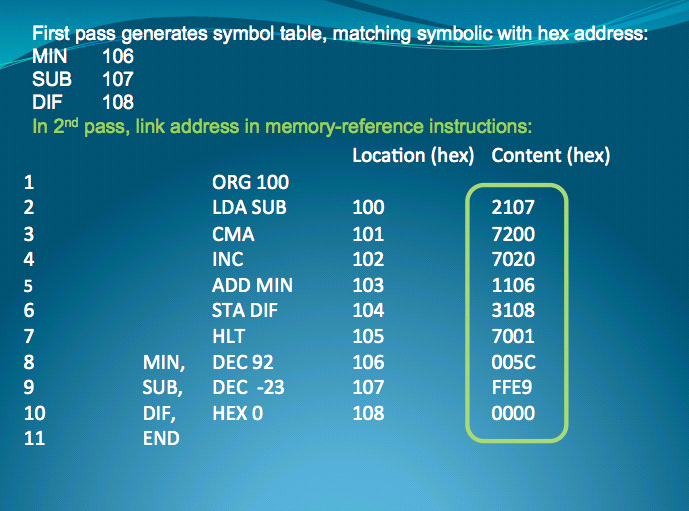
\_\_\_\_\_\_\_\_\_\_\_\_\_\_

An assembler is used to convert the assembly code into a binary translation of the program. We want to convert the **source** code into **object** code

* Symbolic assembly language binary program
* Assemblers often scan symbolic code twice
* Scan 1 (**first pass**): Assign memory location to each machine instruction and operand. This defines the address value of labels. Forms a symbol table associating address symbol with real memory address
* Scan 2 (**second pass**): Uses the symbol table to determine address value of memory-reference instruction

There is a listing of Mano’s 25 instructions and the corresponding hex codes on the next to last page of this document.





**Exercise 1**

Write an assembly language program and hex code translation for the following arithmetic expression:

Val = a + b –c + (- d)

Where the variables should be initialized

a = 1010 = 1010

b = 1310 = 1101

c = -4510 = 1101 0010

d = -6310 = 1100 0001

What is the final result in Val? 131: 10000011

Use the following table for your code.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LOCATION | LABEL | INSTR | OPERAND | COMMENT | HEX CODE |
|  |  | ORG | 100 | Originate program at 100 |  |
| 100 |  | LDA | A | Load AC with A | 2114 |
| 101 |  | ADD | B | Added A +B | 1115 |
| 102 |  | STA | SUM | Stored AC in SUM | 3116 |
| 103 |  | LDA | C | Load AC with C | 2117 |
| 104 |  | CMA |  | Take Comp. of C | 7200 |
| 105 |  | INC |  | Add 1 to C (2’s Comp) | 7020 |
| 106 |  | ADD | SUM | Added SUM + C | 1116 |
| 107 |  | STA | SUN | Stored AC in SUN | 3119 |
| 108 |  | LDA | D | Loaded AC with D | 2118 |
| 109 |  | CMA |  | Take Comp. of D | 7200 |
| 110 |  | INC |  | Add 1 to D | 7020 |
| 111 |  | ADD | SUN | Added SUN + D | 1119 |
| 112 |  | STA | VAL | Stored final AC into VAL | 3120 |
| 113 |  | HLT |  |  | 7001 |
| 114 | A | DEC | 10 |  | 000A |
| 115 | B | DEC | 13 |  | 000D |
| 116 | SUM | DEC | 0 |  | 0000 |
| 117 | C | DEC | -45 |  | 00D3 |
| 118 | D | DEC | -63 |  | 00C1 |
| 119 | SUN | DEC | 0 |  | 0000 |
| 120 | Val | DEC | 0 | Will be 131 | 0000 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

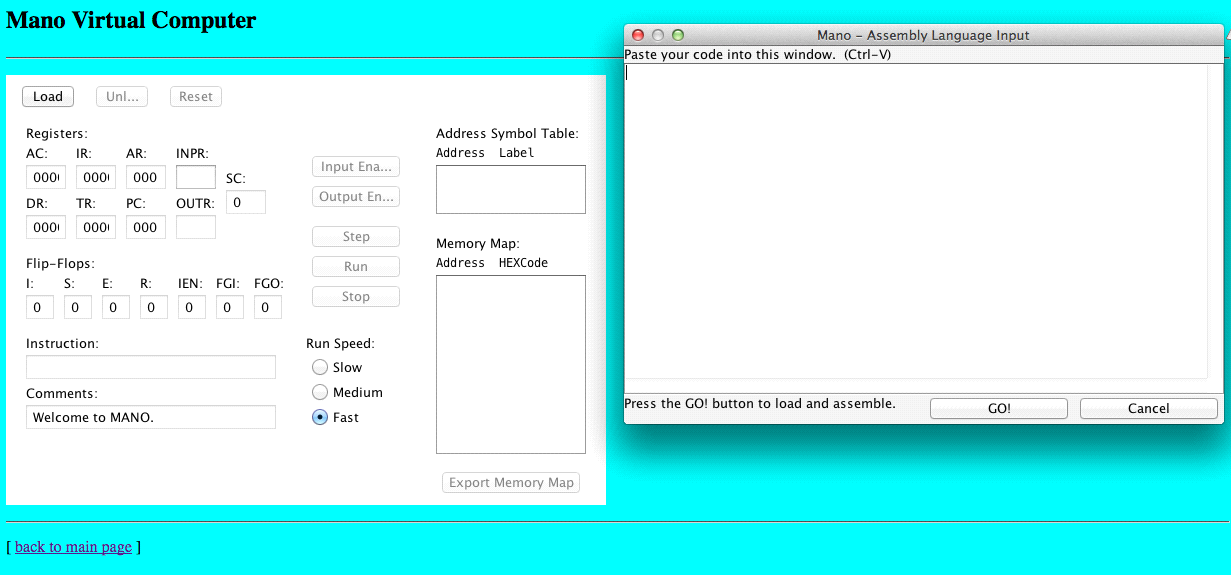
**Exercise 2**

There is a Java Applet tool that you are to become familiar with. It is a virtual machine for the basic computer. It will permit you to test programs and gain a better understanding of executing the microinstructions in order to accomplish program execution. Visit <http://www.unco.edu/nhs/mathsci/ClassSites/hoppercourse/CS222/MANO/mano.htm>

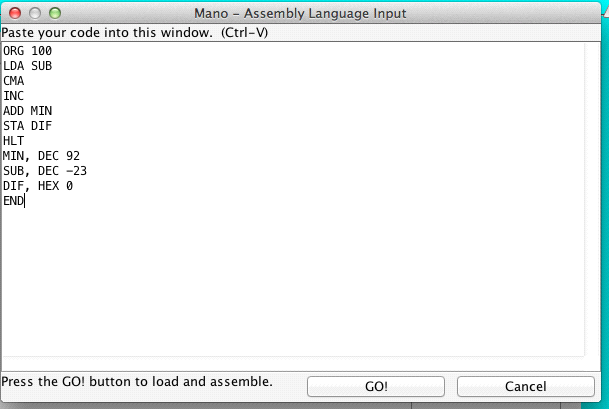
and read the Quick Start Guide.

Enter the code from the solved example for subtraction (page 4) into the Load window.

(You don’t need to enter the comments.)

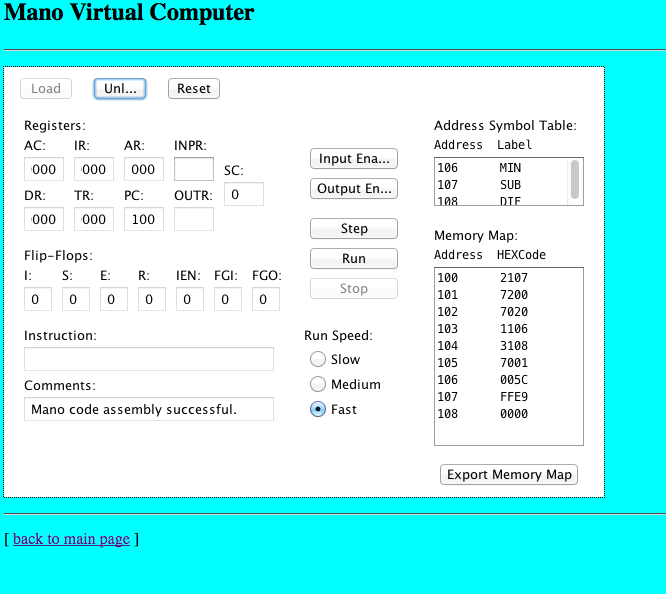


Note that pasting isn’t working on my Mac, but it does on a PC.



Press the GO! button. (You will get an error window pop up if you had a typo. Fix it.)

The Load window will close, and the control panel will display the translated program in hex. It is “loaded” into the RAM as you specified by the ORG statement. In the comments field, you can see that a successful assembly completed.



Note that the PC is initialized to 100, since the code had a directive for ORG 100.

At this point, you are able to **single step through the program to subtract** (as specified on page 1). You are to **trace program execution one step at a time**. You will see the fetch, decode, and execute steps for each of the instructions. Study this in detail, running the program through to completion. Watch the values of the registers at each step. While you are studying this step-by-step execution, you should refer to Table 5-6, which is replicated on the last page of this document. Ask questions if you do not understand the process or the intermediate results.

Did you get a result of 115 as described on page 1? Yes

What is the hex value of the result in DIF? Where can you find that on the virtual machine? 0073, Address 108

**Exercise 3**

When you are finished executing the subtraction code from above, you need to unload the program by clicking on Unl… button. Then, click on the Load button again and enter your assembly language program from Exercise 1. (Sometimes I find that I need to close the window and restart the virtual machine.)

Does the resulting hex code match what you derived when you hand assembled the program? If not, figure out what you did wrong. Yes

Next, trace the program execution. Make certain that you understand intermediate steps.

What is the hex value of the result in VAL? 0083

Checking your work (useful tool) <http://www.binaryhexconverter.com/decimal-to-hex-converter>

**Exercise 4**

Study the program on the next page. What does the following program accomplish? Counts from 0000 (0) to 0014 (20) by increment of 2

Explain why CNT is negated.

Step through the program. What is the final answer in VAL? 0014 (20)

Complete the Hex code in the table, using the tables at the end of this document.

Enter the program in the simulator.

Check your hex code translation. Correct any errors (and understand what you did wrong).

Execute the program in the simulator, stepping through the execution instruction by instruction. (Go through at least 2 iterations of the loop, then click on run. However, be certain you understand termination of the loop.)

Check the answer in VAL after execution. Does it match your original answer? If not, where did you make a mistake?

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LOCATION | LABEL | INSTR | OPERAND | COMMENT | HEX CODE |
|  |  | ORG | 300 |  |  |
| 300 |  | LDA | CNT | Count | 230A |
| 301 |  | CMA |  | Get 2’s comp | 7200 |
| 302 |  | INC |  | Of CNT | 7020 |
| 303 |  | STA | LPC | Loop Counter (-10) | 330B |
| 304 | LOP, | LDA | NUM | Get NUM | 230C |
| 305 |  | ADD | VAL | Tally | 130D |
| 306 |  | STA | VAL |  | 330D |
| 307 |  | ISZ | LPC | Done yet? Increment Op and skip if 0 | 630B |
| 308 |  | BUN | LOP | Back to top of loop | 4304 |
| 309 |  | HLT |  |  | 7001 |
| 30A | CNT, | DEC 10 |  |  | 000A |
| 30B | LPC, | DEC 0 |  |  | 0000 |
| 30C | NUM, | DEC 2 |  |  | 0002 |
| 30D | VAL, | DEC 0 |  |  | 0000 |

**Mano’s Basic Computer: 25 Instructions**

**Hex Codes**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Symbol** | **Hex code** | **Description** |  | **Symbol** | **Hex code** | **Description** |
| AND | 0 or 8 | AND M TO AC |  | INC | 7020 | Increment AC |
| ADD | 1 or 9 | Add M to AC, carry to E |  | SPA | 7010 | Skip if AC ≥ 0 |
| LDA | 2 or A | Load AC from M |  | SNA | 7008 | Skip if AC < 0 |
| STA | 3 or B | Store AC in M |  | SZA | 7004 | Skip if AC = 0 |
| BUN | 4 or C | Branch to m |  | SZE | 7002 | Skip if E = 0 |
| BSA | 5 or D | Save return addr in m and branch to m+1 |  | HLT | 7001 | Halt computer |
| ISZ | 6 or E | Increment m and skip if zero |  | INP | F800 | Input info and clear flag |
| CLA | 7800 | Clear AC |  | OUT | F400 | Output info and clear flag |
| CLE | 7400 | Clear E |  | SKI | F200 | Skip if input flag is on |
| CMA | 7200 | Complement AC |  | SKO | F100 | Skip if output flag is on |
| CME | 7100 | Complement E |  | ION | F080 | Turn interrupt on |
| CIR | 7080 | Circulate right E and AC |  | IOF | F040 | Turn interrupt off |
| CIL | 7040 | Circulate left E and AC |  |  |  |  |

