EC327 - Spring 2025 - Homework 1

Background

This homework assignment will focus on Microarchitecture and Instruction Set levels of the six level computer architecture.

For the assignment, we will use the very simplified Intel assembly and machine language for a 16-bit computer. This is the same language that we went over at lecture and all of the supported instructions, as well as their binary codes are included below.

To help with your work on this assignment, you should use the Simplified Simulator. It will allow you to load your code and step through it instruction by instruction.

Some guidance for you as you use the simulator:

- The instructions for your Assembly program are input into the **Machine Code** input box in Hexidecimal.
- Our instruction codes are all 16-bit, which means that each one will be represented by four Hexidecimal digits (0-F).
- The **Load address** input box denotes, in Hexidecimal, the first memory location of where the code in **Machine Code** will be loaded. The **PC** register will be set to start at this location as well.
- The program in the **Machine Code** input box should be a sequence of instructions, each represented by four hexidecimal digits, with no spaces between the instructions. The instructions will get loaded into subsequent memory cells starting with the value of **Load address** and consuming as many memory cells as there are instructions.
- The simulator contains 16^3 = 4096 memory cells. The address of each memory cell is represented as a four digit Hexidecimal number where the first digit is 0. When you hover over a memory cell in the simulator, you'll see that it expresses the memory address as **Mem[0xZZZ]** where **Z** is a Hexidecimal digit (the leading 0 is omitted). The value is expressed as **0xZZZZ** and **0x****** for cells that have not been set.
- Each memory cell can store 16 bits

- Memory cells that have been modified are highlighted. The memory cell that **PC register** is currently pointing at is highlighted the brightest.

- Negative numbers are represented in two's complement.

- You can click on the **Monitor Registers** tab to see the value of
    - Six registers
    - PC register
    - EFLAGS register

- The buttons at the top allow you to step through your program one instruction at a time or all at once.

- The supported instruction set is on the page below.

instruction	binary code	description	arguments	example
halt	0000 0000 0000 0000	halt execution	none	0000 0000 0000 0000
inc R_n	1100 0	register R_n gets (the contents of R_n) + 1	11-bit argument representing **n**.	**Inc R_3** 1100 0000 0000 0011
jmp num	1100 1	jump to *relative* address num, by adding the integer num to the program counter.	11-bit argument representing **num**	**jmp 25** 1100 1000 0001 1001
jne num	1101 0	jump to relative memory address num, if the last cmp instruction compared unequal items.	11-bit argument representing **num**	**jne 13** 1101 0000 0000 1101
je [R_n]	1101 1	jump to absolute memory address given as the contents of R_n, if the last cmp instruction compared equal items.	11-bit argument representing **n**	**je [R_3]** 1101 1000 0000 0011
add R_n, R_m	0001	Rn gets the addition of the contents of Rn and Rm.	Two 6-bit arguments; first 6 bits representing **n** and next 6 **m**	**add R_3, R_5** 0001 0000 1100 0101
sub R_n, R_m	0010	Rn gets the result of contents of Rn minus the contents of Rm.	Two 6-bit arguments; first 6 bits representing **n** and next 6 **m**	**sub R_9, R_8** 0010 0010 0100 1000
xor R_n, R_m	0011	Rn gets the bit-wise exclusive or of Rn and Rm.	Two 6-bit arguments; first 6 bits representing **n** and next 6 **m**	**xor R_2, R_3** 0011 0000 1000 0011
cmp R_n, R_m	0100	Compares the contents of registers Rn and Rm and sets appropriate flag.	Two 6-bit arguments; first 6 bits representing **n** and next 6 **m**	**cmp R_5, R_1** 0100 0001 0100 0001
mov R_n, num	0101	R_n gets the value num.	Two 6-bit arguments; first 6 bits representing **n** and next 6 **num**	**mov R_5, 61** 0101 0001 0111 1101
mov R_n, R_m	0110	R_n gets a copy of the contents of R_m.	Two 6-bit arguments; first 6 bits representing **n** and next 6 **m**	**mov R_5, R_1** 0110 0001 0100 0001
mov [R_n], R_m	0111	copy data from register R_m into the absolute memory address stored in R_n	Two 6-bit arguments; first 6 bits representing **n** and next 6 **m**	**mov [R_5], R_1** 0111 0001 0100 0001
mov R_n, [R_m]	1000	copy data from absolute memory address stored in Rm to register Rn	Two 6-bit arguments; first 6 bits representing **n** and next 6 **m**	**mov R_5, [R_1]** 1000 0001 0100 0001

Submission Instructions

You will submit this assignment as single .zip file with the following name:

<first-name>_<last-name>-hw1.zip

So for example:

ed_solovey-hw1.zip.

Your zip file should contain one file per problem in this assignment. Those files should be named like this:

<first-name>_<last-name>-hw1-<problem-number>.txt

So for example, my problem one file would be:

ed_solovey-hw1-1.txt

Each of the problems below will have specific instructions around what the text file for that problem should look like.

Note on Collaboration

You are welcome to talk to your classmates about the assignment and discuss high level ideas. However, all code that you submit must be your own. We will run code similarity tools against your submissions and will reach out with questions if anything is flagged as suspicious.

The closed book exams for this class will mostly cover material very similar to what you do on the homework assignments. The best way to prepare for the exams is to do the assignments on your own and internalize the learnings from that process. Cheating on the assignments will very likely result in you not doing well on the exams.

Problem 1 - Assemble (10 points)

Submission Instructions

Your solution to this problem should contribute a single **txt** file to your overall **hw1** zip. The **txt** file should follow the following format:

<first-name>_<last-name>-hw1-1.txt

So for example, my file would be:

ed_solovey-hw1-1.txt

The format of your **txt** file for this problem should look like this:

1. 0xZZZZ
2. 0xZZZZ
3. 0xZZZZ

where each line contains the number of the subproblem and then a single hexidecimal number. (**Z** is a hexidecimal digit here)

Actual Problem

Assemble each of the following instructions into machine code, represented by four Hexidecimal digits. Your answers should look like 0xZZZZ where **Z** is a Hexidecimal digit.

If our architecture does not support a given instruction, write the string “NONE”. The numeric constants below are base-10.

1. MOV R_3 37
2. INC R_2
3. CMP R_3 R_1
4. JE [R_3]
5. ADD R_4 R_0

Problem 2 - Dis-assemble (15 points)

Submission Instructions

Your solution to this problem should contribute a single **txt** file to your overall **hw1** zip. The **txt** file should follow the following format:

<first-name>_<last-name>-hw1-2.txt

So for example, my file would be:

ed_solovey-hw1-1.txt

The format of your **txt** file for this problem should look like this:

1.
    a. **MOV R_3 37**
    b. Set the value of register-3 to 37.
2.
    a. **INC R_2 , CMP R_1 R_2**
    b. Increment register-2 to be 1, and compare contents of register-1 and register-2, resulting in 0 because the contents are different

where there is a header line for each subproblem, then bullet a) contains a comma separated list of assembly instructions and bullet b) contains a one sentence English explanation of what the program is doing.

Actual Problem

Dis-assemble each of the following machine code strings into our simplified assembly instructions. The strings below are in hexadecimal where each group of four hexadecimal digits represents a single instruction.

1. C807
2. 7045
3. 5021C0014083D800
4. C000506C7040
5. 5028504A5083C00311024043D800CFFC0000

Submission Instructions For Problems 3 - 6

Your solution to each of the problems 3 through 6, should contribute a single **txt** file to your overall **hw1** zip. The **txt** file should follow the following format:

<first-name>_<last-name>-hw1-<problem-number>.txt

So for example, my file for problem 4 would be:

ed_solovey-hw1-4.txt

The format of your **txt** file for these problems should look like this:

1. Assembly code for solving the problem
2. Machine code in Hexidecimal just as you would input it into the Simplified Simulator
3. An English explanation in a sentence or two of what your program is doing.

For example:

1. INC R_0
2. C000
3. Program increments register-0 from 0 to 1

Problem 3 - Memory Manipulation (10 points)

Set the memory cell at absolute address 51 to the value 72.

Problem 4 - Looping (15 points)

Set memory cells 100 through 118 to the value 0 through 18.

Problem 5 - Odds (20 points)

Set the odd numbered memory cells between 100 and 200 to the same value as the address of that memory cell.

Problem 6 - Multiplication (30 points)

Write code to multiply the contents of memory cells 9 and 18, placing the result in memory cell 72. Your code should initialize memory cells 9 and 18 to some non-zero values and should work for any initial values of those cells (we will test it with different values).