UF Herbert Wertheim College of Engineering
Department of Electrical & Computer Engineering
UNIVERSITY of FLORIDA

Dr. Christophe Bobda
Omar Elbasri
EEL3701C - Fall 2025

Lab 7: Assembly Language

Overview

In the last lab, you created a simple CPU capable of running a basic instruction set architecture. Now, abstracting away the individual components of the CPU, you will use a new provided instruction set architecture (ISA) to write programs. In the first part of the lab, you will download the ISA simulator and its documentation to answer questions about the ISA. In the second part, you will create a program to modify registers using loops and branches. In the final part, you will create a larger program to perform operations on a list of values in memory.

Pre-Lab Procedure: Part 1 - Download and Instruction Set

First, ensure that you have the latest version of Python installed onto your computer. If you took or are taking COP3502C or COP3504C, it should already be installed. Python is a programming language used very frequently in many fields, including computer and electrical engineering. However, no Python knowledge is required for this lab.

After accepting the assignment in GitHub Classroom, you should see 4 main files in your repository (besides the lab document template): README.md, info.md, codes.py, and runner.py. Familiarize yourself with the architecture of the CPU and the instructions by reading info.md, then answer the following questions in your lab document.

1. What are the main registers that arithmetic and logic operations will be performed with?
2. What line of code would load the immediate value 0x37 into register B?
3. What line of code would take the bitwise AND of registers A and B and place it in register C?
4. What 2 lines of code would load memory address 0x3701 into register X, then increment it?
5. What 2 lines of code would compare register A to the immediate 5, then branch to the label LOOP if they are equal?
6. What 3 lines of code would subtract register A from register B and place the result in B, add register C to register B and place the result in B, then branch to LOOP if the result is negative?
7. What directive would place 0xFF at memory address 0x371?
8. In hexadecimal, what are the bounds of immediates that can be placed into the general purpose registers? What about the memory access registers?
9. For the following values of register A and B, what flags does CMP A, B set? 1: A = 3, B = 4. 2: A = 50, B = 32. 3: A = 9, B = 9.

Once you've answered the questions, read the README.md file to understand how to operate the CPU. Follow the directions in README.md to run the ex1.lab7, ex2.lab7, and ex3.lab7 files to make sure you know how they work and how to run them. Afterwards, you're ready to start on the next portion of the lab.

Part 1 Summary:

1. After accepting the assignment in GitHub Classroom, read the relevant files
2. Answer the questions and place the answers in your lab document.
3. Understand and run the ex1.lab7, ex2.lab7, and ex3.lab7 files to learn how to operate the CPU.

Pre-Lab Procedure: Part 2 - Register Operations and Branching

In this section of the lab, you will write a program to modify registers using **branches**, which set the program counter to different parts of the program based on two CPU flags, the zero flag and the negative flag. The zero flag is set when the result of an operation is zero, and the negative flag is set when the MSB of an operation is 1. The arithmetic operations typically set both flags, while logic operations typically only set the zero flag.

Once a flag is set, you can use one of the conditional branching instructions: JNZ, JEZ, JNE, and JPZ. This instruction lets you conditionally jump (also called a branch) to a label somewhere else in a program. With this, we can form a basic loop. This one will exit once Register A is equal to 5, running the code inside of the loop 5 times.

```
-- initialize register
LDI A, 0 -- value we're starting at
LDI B, 5 -- value we want to compare against

LOOP: -- address label for loop to jump back to
ADDI A, A, 1 -- adds 1 to the counter
-- ANY CODE TO BE EXECUTED REPEATEDLY GOES HERE
CMP A, B -- compare A and B
JNZ LOOP -- jump to LOOP if the zero flag is false

LDI C, 10 -- do things outside the loop
```

With a basic loop structure, it can be adapted to do whatever we need it to. Changing the value to be compared against, the beginning of the counter, and the loop condition allows for usefulness in a variety of situations. It is also possible to use CMPI instead of CMP for setting

flags. You will now write a program to loop a large number of times while performing logical and arithmetic operations.

This program will loop **200** times exactly. Use register A for holding the current iteration of the loop and start it at 0. Initialize the other registers as follows: B = 0x41, C = 0xF7, D = 0x59. Inside the loop, a couple operations will be performed. C = C AND D, then C = B + C. Afterwards, B = B XOR D, and finally, D = NOT D.

Put both your program under the filename “part2.lab7” and the final values of the B, C, and D registers into your lab document. It is recommended to use the skip flag (-s) when running your program.

Part 2 Summary:

1. Write a program in a file named “part2.lab7” to understand how looping and branching function in assembly.
2. Put the program and the final values of the registers into your lab document and make sure the “part2.lab7” file is in your GitHub repository when you submit.

Pre-Lab Procedure: Part 3 - Memory Access and Lists

In this last section of the lab, you will use looping and logic/arithmetic operations to modify a list within memory. A list, for our purposes, is just multiple adjacent entries in memory. Using the .list directive, we can place multiple entries in memory following a starting address.

View the part3.lab7 file within the repository to see the starting address and the number of entries in the list. However, note that the last value of the list is 0x00. Rather than ending the loop at a certain number, end the loop when the value read from memory is 0x00.

Look at the WRM and RDM instructions to ensure you understand how to write and read from memory, and think about what instructions could be used to increment the memory address every loop iteration. After performing operations on the values stored in memory, you will write it to a different starting address to create another list.

The **input list starts at 0x200** and the **output list should start at 0x300**. For the entries of the list, you will perform the following operations. If bit 4 (where bit 0 is the rightmost bit) of the number is set (1), add 5, divide by 2, then OR with 0x3A, and then place it in the output list. If the bit 4 of the number is not set (0), subtract 60, multiply by 2, then AND with 0xF9 and place it in the output list. Do this for every entry in the list, and stop when you hit the terminator entry, 0x00.

Do **not** hardcode the length of the list within your code or else points will be deducted.

Here are some helpful tips for this part of the lab. After writing your program, perform the operations manually on some entries of the list to ensure that your program has correct outputs. To execute code conditionally, take advantage of the branching instructions after checking for a condition. To check if a specific bit is set, think about what operations you could do to result in a value that is either 0 or non-zero depending on if the bit is set.

Part 3 Summary:

1. Learn how to read/write memory in assembly, then write the program using the “part3.lab7” file as a base.
2. Test your program for correctness by doing the operations manually on some entries of the list.
3. Put the entirety of the program into your lab document, and make sure the modified “part3.lab7” file is in your GitHub repository when you submit.

In-Lab Procedure

1. Complete the in-lab quiz as specified by your PI.