|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 🔑 **Essential** 🔑 | | | |  | 🏆 **Enhanced** 🏆 | | | |
|  |  |  |  |  |  |  |  |  |
| \_\_\_\_\_ | \_\_\_\_\_ | \_\_\_\_\_ | \_\_\_\_\_ |  | \_\_\_\_\_ | \_\_\_\_\_ | \_\_\_\_\_ | \_\_\_\_\_ |

**Score: \_\_\_\_\_**

**15 – Assembly Language Programs**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2022

Prof. Grant Braught

**Name:**

Today’s class revisited the idea of program translation through the use of a compiler and an assembler. We also saw the machine simulator that we will be using to learn about assembly language programming. A first example of an assembly language program was presented and its execution was described at the assembly language machine level of abstraction (i.e. we did not discuss the underlying translation to machine language for execution. We will spend some time on that in a lab later.) Today’s activities will guide you through the process of creating, assembling and running assembly language programs on the machine simulator. You will become familiar with the simulator’s features and explore a few assembly language instructions that correspond to operations that the K&S computer did not support. Finally, you’ll write, assemble and run an assembly language program or two.

**Getting Setup for Assembly Language Programming:**

To write, assemble and run assembly language programs you will need to create a space to work, make sure you have a plain text editor and download the assembler and machine simulator programs. The exercises in this section walk you through that process.

*A Plain ASCII Text Editor:*

To write assembly language programs for our assembler you will need to use a plain (ASCII) text editor.

* On Windows you can use the Notepad program.
  + Start ® Programs ® Accessories ® Notepad
* On Mac you can use the TextEdit program.
  + You will need to configure it to work with plain text before starting:

1. Launch the TextEdit program from the Applications folder.
2. From the TextEdit menu choose Preferences.
3. Select “Plain text” under the Format heading in the dialog.
4. Close the dialog
5. Close the application
6. Launch the TextEdit program again.

* The window now should now just be plain, without any ruler or formatting (bold, italic, etc) buttons.
* If you have a different **plain text** editor that you prefer, you should feel free to use it. If you are unsure if your editor works with plain text, check with the instructor.

*Making a Space:*

🔑 1. Create a new folder on your computer named COMP256ASM that will hold the assembler program, the machine simulator program and all of the assembly programs that you write for this class.

No answer is required here, **just be sure to create the folder.**

I put this box here to get your attention.

People tend to miss questions that don’t have a box.

🔑 2. Open your text editor and enter the code for the sample program from the class slides. Save the file as Ex1.asm in your COMP256ASM folder. Give a screen shot of your text editor window with the sample program entered as your answer for this question.

*Windows Users:*

Our assembler and machine simulator are Java programs. Thus, you will need to have Java Runtime Environment (JRE) installed to use them. Unfortunately, Windows does not provide a JRE by default. Thus, if you are using your own Windows computer for these assignments, you will need to install a JRE before continuing. Alternatively, you can use one of the Mac computers in the Tome labs to complete these assignments. They all have a JRE installed. Using the Tome lab computers should also be your fallback if you encounter difficulties installing a JRE.

If you elect to work on your own Windows computer please follow the instructions in the Tech Decode Tutorials video *How to Install Java JDK JRE on Windows 10*:

* <https://www.youtube.com/watch?v=Ft0vza0omDQ> (3:23)

*The Assembler:*

To assemble programs, you’ll need the assembler program that translates assembly language into machine language. The following short screencast shows you how to download and use the assembler program:

* Downloading and using the Assembler:
  + <https://web.microsoftstream.com/video/bdb2a492-c33e-4fd6-b98e-d8c837399450> (2:39)
* Assembler Download Link:
  + <https://github.com/dickinson-comp256/AsmMachine/raw/main/Assembler/bin/Assembler.jar>

🔑 3. Use the process illustrated in the video to download the assembler program, move it to your COMP256ASM folder and then use it to assemble your Ex1.asm program. Don’t go on until you are able to assemble this program and obtain the Ex1.ml file containing the machine language translation of the program.

Open the Ex1.ml program in your text editor and paste the first three lines of that file here to show that you’ve been able to successfully assemble the program.

Note: If you are curious, the lines with \* and \*\* are comments that our machine simulator will use. The lines without a \* or \*\* contain only 0’s and 1’s and are the machine language instructions to our machine simulator. Conceptually, they are similar to those we saw earlier for the K&S, but because it is a different machine, the formats, opcodes and operands are different.

*The Machine:*

To run your assembled machine language programs, you’ll need the machine simulator program. Watch the following short screencast that shows you how to download and use the machine simulator program:

* Downloading and using the Machine Simulator:
  + <https://web.microsoftstream.com/video/ec3499a7-3dd0-4331-9710-aa395107ae26> (4:00)
* Machine Simulator Download link:
  + <https://github.com/dickinson-comp256/AsmMachine/raw/main/Machine/bin/Machine.jar>

🔑 4. Use the machine simulator to load and execute your Ex1.asm program from above. Paste a screen shot of the machine simulator window after running the program as your answer to this question.

🔑 5. This question will familiarize you with some of the features of the machine simulator. After running your Ex1.asm program, use the machine simulator to answer the following questions:

a. What is the **binary machine language instruction** stored starting at main memory address 12? (Hint: Use the “Memory Browser” pane).

b. What is the **hexadecimal representation** of the instruction from a? (Hint: Use the dropdown at the bottom of the machine simulator window.)

c. In what four representations can the value in a register be displayed? (Hint: Look at the drop down beside the registers.)

**Assembly Instruction Reference:**

For reference the table below summarizes all of the assembly language instructions that we know about so far:



**Experimenting with Instructions:**

Many of the assembly language instructions for our machine simulator are similar to the machine language instructions supported by the K&S computer. However, this machine simulator has a more capable ALU and can perform some operations that we have not seen before. For the following questions you should enter the programs into a text editor, assemble them and run them on the machine so that you know how they work. Becoming proficient with these small less complicated programs will make it much easier to work with the longer more complex programs we will be seeing soon.

*Immediate Addressing Mode Instructions:*

One set of instructions that that is new to us are the *Immediate Addressing Mode* instructions. The Immediate Addressing Mode instructions are used when you need literal values in your programs. For example, if you want to put the specific value 108 into a register or add the value 51 to a register or subtract 13 from a register. The Immediate Addressing Mode instructions are what you need to perform each of those operations. These Immediate Addressing Mode Instructions appear in the table above and are indicated with a comment indicating that they are Immediate Mode.

🔑 6. Give the **Instruction Format** for each of the 5 Immediate Addressing Mode instructions shown in the table above.

🔑 7. Give an Immediate Addressing Mode instruction that will perform each of the following tasks. The instruction you give should use the specific registers and values indicated. Your instruction will look similar to what is shown in the “Example” column of the instruction reference.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Task** | **Assembly Instruction** |  |
|  | Put the value 100 into register 2. |  |  |
|  | Add 17 to the value in register 5 and put the result into register 9. |  |  |
|  | Set the value in register 3 to 22 less than its current value. |  |  |
|  | Increment (add 1 to) the value in register 7. |  |  |
|  |  |  |  |

🔑 8. Consider the following program that uses Immediate Addressing Mode instructions:

LOAD R0 #100

ADD R1 R0 #50

SUB R2 R1 #75

HALT

Complete the table below to show the base 10 values that will be contained in the indicated registers after this program is executed.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Register** | **Value** |  |
|  | R0 |  |  |
|  | R1 |  |  |
|  | R2 |  |  |
|  |  |  |  |

🔑 9. Write a small assembly language program that loads a value of your choice into R5 and another value of your choice into R7 and then uses assembly language instructions to compute:

R3 <- (R5+15)-(R7-30)

Hint: You’ll need to use both Immediate Addressing Mode and Register to Register instructions. Save your program in a text file, assemble it and run it in the machine simulator to ensure that it works correctly. Give your assembly language program as your answer to this question.

Note: Your program must work even if different values are entered into R5 and R7.

*Some New (to us) Instructions:*

Three of the instructions that exist in our assembly language did not exist in the Knob & Switch computer: NOT, SHL and SHR. These questions explore what those instructions do. You should enter the small programs into your text editor, assemble and run them in order to answer the questions.

🏆 10. The following program loads the value 92 into R0 and then uses the NOT instruction on it, placing the result in R1.

LOAD R0 #92

NOT R1 R0

HALT

a. Give the **32-bit binary values** that appear in registers R0 and R1 after you run the above program. Hint: You can use the dropdowns beside the registers in the machine simulator to switch between binary and base 10 representation.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Register** | **32 bit Binary Value** |  |
|  | R0 |  |  |
|  | R1 |  |  |
|  |  |  |  |

b. What does the NOT instruction do? Hint: Compare the binary representation of the values in R0 and R1 from part a.

🏆 11. Using a similar approach, experiment with the SHL and SHR instructions by writing small programs like the one in question #10, assembling and running them to learn about these instructions.

a. What does the SHL command do **to the bits** of the value it is operating on? Hint: Look at the binary values in the registers that you used.

b. What effect does SHL have on the base 10 value it is operating on?

c. What does the SHR command do to the bits of the value it is operating on?

d. What effect does SHR have on the base 10 value it is operating on? For a complete answer you’ll need to try it out on both even and odd base 10 values (i.e. a 0 in the LSbit (an even number) or a 1 in the LSbit (an odd number)).

🏆 12. What effect does the following program have on the base 10 value in R0? Briefly explain your answer.

LOAD R0 #92

NOT R0 R0

ADD R0 R0 #1

HALT

**Input and Output:**

Recall that input and output for our machine is accomplished using *Memory Mapped Input/Output*. This means that special memory addresses are set aside for performing input and output operations with devices. Reading (i.e. LOADing) from these addresses reads input from the device, and writing (i.e. STOREing) to these addresses writes output to the device.

Our assembler pre-defines two labels for us to use for input and output:

* STDIN – The standard input device. LOADing from STDIN will read input from the user.
* STDOUT – The standard output device. STOREing to STDOUT will display output for the user.

Watch the following short video to see how to use the STDIN and STDOUT labels in your assembly language programs to perform input and output using our machine:

* Using Standard Input and Standard Output:
  + <https://web.microsoftstream.com/video/6887c47a-f866-41ff-89d6-c35753b02885> (2:45)

🔑 13. Write and test a small assembly language program that reads two values from STDIN and displays their difference (i.e. subtract the second from the first) on STDOUT. Give the assembly language program that you wrote as your answer. Be sure to assemble and run your program in the machine simulator to be sure that it works.

🏆 14. Write a small assembly language program that loads some value of your choice into R0 and then stores 100 minus the value in R0 into R3. Your program should work for any value that is placed into R0. Be sure to write your program into a text file, save it, assemble it and run it in the machine simulator to ensure that it works. Give your assembly language program as your answer to this question. Hint:

**Assembly Programming:**

🏆 15. Write an assembly language program the performs the same computation as the High-Level Language (HLL) program shown below. Be sure to assemble and run your program in the machine simulator to check that it works.

Read X

Read Y

Read Z

Q = ((X + (Z – 5)) \* 2) + (4 \* (25 - Y))

Print Q

Give the assembly language program that you wrote as your answer.

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.