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

**LA2 – Assembly Language Programs**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2023

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 a bit 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.

**Assembly Instruction Reference:**

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



Note that in the “Instruction Format” column an R indicates that a register must appear in that position, an L indicates that a label must appear in that location and # indicates that an immediate value (e.g. #231) must appear in that location.

**Assembly Language Instructions:**

When writing assembly language instructions, you must use only the specific formats that are allowed by the particular assembly language you are using. Some of the allowed formats for our assembly language are shown in the “Instruction Format” column in the table above. The “Example” column then gives an example of that format using specific registers, values and/or labels. Attempting to use a format that is not allowed will result in a program that will not assemble.

🔑 1. Consider each of the assembly instructions in the table below. For each instruction, indicate that it is *valid* if it matches one of the formats in the table above or *invalid* if it does not.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Instruction** | **Valid or Invalid?** |  |
|  | LOAD R3 X |  |  |
|  | STORE Y R2 |  |  |
|  | OR R1 R2 R7 |  |  |
|  | ADD R3 #9 R5 |  |  |
|  | SUB R3 A B |  |  |
|  | NOT R9 R10 |  |  |
|  |  |  |  |

🔑 2. Consider each of the assembly instructions in the table below. For each instruction, indicate its addressing mode as Register to Register, Direct or Immediate.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Instruction** | **Addressing Mode** |  |
|  | LOAD R3 X |  |  |
|  | AND R7 R5 R3 |  |  |
|  | SUB R2 R9 #3 |  |  |
|  | LOAD R7 #12 |  |  |
|  | STORE R8 B |  |  |
|  | NOT R9 R10 |  |  |
|  |  |  |  |

**Getting Setup for Assembly Language Programming:**

To write, assemble and run assembly language programs we will be using the Docker container that you created at the end of the LA1 homework.

3. Follow the instructions at the end of LA1 to be sure that you have Docker Desktop running, the Comp256Assembly container started and that you are connected to it with Tiger VNC. You should see the desktop in the image below when everything is working:

A screenshot of a computer

Description automatically generated

4. Open the Mousepad text editor by clicking on its icon in the “Launcher” at the bottom of the Desktop. Enter the following assembly language program into Mousepad and save it as

LA2-1.asm. Note that when we create files containing assembly language we will name them with a .asm extension. Using this extension help us know what is in the file. It is sort of like naming text files with the .txt extension or Java source code files with the .java extension.

ONCE: .word 15

TWICE: .word 0

LOAD R0 ONCE

ADD R1 R0 R0

STORE R1 TWICE

HALT

Give a screen shot of your program in the Mousepad editor as your answer for this question.

5. The command to assemble a program is:

assembler <asm file> <ml file>

where <asm file> is the name of the assembly language source code file that is to be converted to machine language and <ml file> is the name of the machine language file to be generated. Like we use .asm as the extension on files containing assembly language, we will use .ml as the extension on our machine language files.

Assemble the program in your LA2-1.asm file into the file LA2-1.ml. Give a screenshot of the command you used and its output here.

6. Open your program in the machine and run it. Give a screenshot showing the machine simulator window after the program was run as your answer.

**Experimenting with some New Instructions:**

The operations performed by many of the assembly language instructions for our machine simulator will seem very similar to the machine language instructions supported by the K&S computer. For example, adding two registers or moving a value from memory to a register. However, this machine simulator has a more capable ALU that can perform some operations that we have not seen before. The following sections will explore some of these instructions.

*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 both by the use of the # character and by a comment indicating that they are Immediate Mode.

🔑 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. You can put your instructions into a program, assemble and run it if you would like to check your work.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **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. You can put these instructions into a program, assemble and run it if you would like to check your work.

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

*Some New (to us) Operations:*

Three of the instructions that exist in our assembly language did not exist in the Knob & Switch computer are: NOT, SHL and SHR. The questions in this section have you 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.

9. 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. Assemble and run this small program. Give the **32-bit binary values** that appear in registers R0 and R1 after you run the 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.

🏆 10. What effect would the following program have on a signed (two’s complement) base 10 value in R0? If you are unsure you can assemble and run the program, observe the result and then try to understand why this program does what it does.

LOAD R0 #92

NOT R0 R0

ADD R0 R0 #1

HALT

11. Question #9 used a small program including the NOT statement to help you understand what it does. Experiment with the SHL and SHR instructions by writing, assembling and running small programs, like the one in question #9, 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)).

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

The following short video illustrates 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)
    - Note: The commands used to run the assembler and machine in this video are slightly different than what we have been using. You should continue to use the assembler and machine commands we have learned.

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

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

**Assembly Programming:**

🔑 13. Translate the following HLL program into assembly language:

READ NUM1

READ NUM2

RESULT = (NUM1-10)+(20-NUM2)+5

Save your program in a text file, assemble it and run it in the machine simulator with different values for NUM1 and NUM2 to ensure that it works correctly.

Give your assembly language program as your answer to this question.

🏆 14. **Optional Extra Challenge:** Translate the High-Level Language program shown below into assembly language. 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.