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

**MA3 – The Stored Program Architecture**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2023

Prof. Grant Braught

**Name:**

**Introduction:**

Today’s class extended the capabilities of the Knob and Switch computer by introducing branching instructions. These instructions allowed us to create machine language programs that perform operations like if/else, while loops and for loops that we see in the programming languages we are used to using. We then connected the operation of the Knob & Switch computer to larger concepts and structures that underlie the behavior of modern computers. These included the *stored program architecture*, the *central processing unit (CPU)* and *the instruction cycle* (fetch/decode/execute). In today’s activities you’ll gain experience with the branching instructions and learn more about the larger concepts of the stored program architecture.

You will need to use the Knob & Switch Machine language reference for many of the questions in today’s activities, so it is included here for convenience:



The version of the Knob & Switch computer that executes machine language programs can be found here:

* <https://dickinson-comp256.github.io/Knob-And-Switch-Computer/machine.html>

**Knob and Switch Branching Instructions:**

🔑 1. Give a binary machine language instruction for each of the following operations:

a. Branch to the instruction at memory address 12.

b. Branch to the instruction at memory address 8 if the result of the last ALU operation was zero.

c. Branch to the instruction at memory address 27 if the result of the last ALU operation was negative.

🔑 2. Each part of this question gives a sequence of machine language instructions. They are written using our shorthand instead of binary to make them easier to read. But they could easily be translated into binary using the reference at the top of this assignment.

For all parts of the question, assume that registers R0 and R1 have the following values:

R0: 10

R1: 20

For each of the following sequences, indicate if the branch will be taken or not and then give the value that would be in the program counter (PC) after the branching instruction has been executed.

Recall that the purpose of the operation performed just before the branching instruction is to set the ALU flags. The branching instructions then use those ALU flags to determine if the branch is taken or not. If the appropriate flag is set, then the branch is taken and the PC is changed to the given value. If the appropriate flag is not set, then the branch is not taken, and the PC is incremented (PC ← PC + 1)so that it indicates the next instruction in memory.

a. MM Operation

0: R2 ← R0 - R1

1: If Negative PC ← 7

2: …

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Branch Taken (Yes/No)? |  |  |
|  | Program Counter Value |  |  |
|  |  |  |  |

b. MM Operation

0: R2 ← R0 + R1

1: If Negative PC ← 10

2: …

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Branch Taken (Yes/No)? |  |  |
|  | Program Counter Value |  |  |
|  |  |  |  |

c. MM Operation

0: R2 ← R0 + R0

1: R2 ← R2 – R1

2: If Zero PC ← 15

3: …

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Branch Taken (Yes/No)? |  |  |
|  | Program Counter Value |  |  |
|  |  |  |  |

d. MM Operation

0: R2 ← R0 – R0

1: R2 ← R2 + R1

2: If Zero PC ← 9

3: …

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Branch Taken (Yes/No)? |  |  |
|  | Program Counter Value |  |  |
|  |  |  |  |

e. MM Operation

0: R2 ← R0 – R0

1: PC ← 12

2: …

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Branch Taken (Yes/No)? |  |  |
|  | Program Counter Value |  |  |
|  |  |  |  |

3. The Knob and Switch computer has only three branching instructions. While it may be surprising, it turns out that we can do all of the kinds of branching that we do in our high-level language programs (e.g. x > y, a <= b, etc…) with just these three basic instructions. However, sometimes you have to be pretty creative to get it done.

For each of the following questions, give a sequence of K&S machine language operations using our shorthand that branches as specified. It is not necessary to translate the operations into binary. However, if you want to you can always test your answers by converting the instructions to machine language and running them on the K&S simulator.

🔑 a. Branch to the instruction at memory address 22 if R0 is equal to R1, otherwise continue to the instruction at memory address 2.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **MM** | **Operation** |  |
|  | 0 |  |  |
|  | 1 |  |  |
|  | 2 | *Continue here if branch not taken.* |  |
|  |  |  |  |

🔑 b. Branch to the instruction at memory address 5 if R0 is less than R1, otherwise continue to the instruction at memory address 2.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **MM** | **Operation** |  |
|  | 0 |  |  |
|  | 1 |  |  |
|  | 2 | *Continue here if branch not taken.* |  |
|  |  |  |  |

🏆 c. Branch to the instruction at memory address 19 if R0 is negative, otherwise continue to the instruction at memory address 2. Hint: Use the & or | operations.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **MM** | **Operation** |  |
|  | 0 |  |  |
|  | 1 |  |  |
|  | 2 | *Continue here if branch not taken.* |  |
|  |  |  |  |

🏆🏆 d. Branch to the instruction at memory address 12 if R0 is not equal to R1, otherwise continue to the instruction at memory address 3.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **MM** | **Operation** |  |
|  | 0 |  |  |
|  | 1 |  |  |
|  | 2 |  |  |
|  | 3 | *Continue here if branch not taken.* |  |
|  |  |  |  |

**Branching in Programs:**

4. Consider the following computation expressed in pseudo code using our shorthand:

if MM[11] > MM[10]

MM[12] ← MM[11]

else

MM[12] ← MM[10]

The statements in the table below are a partial implementation of a program that performs the above computation. Do not complete the yellow highlighted lines yet. The questions below will ask about this program and help guide you toward the instructions that belong in the yellow lines.

|  |  |  |
| --- | --- | --- |
| Memory Address | Operation or  Value | Machine Language Instruction |
| 00000 (0) | R0 ← MM[10] | 1000 0001 0 00 01010 |
| 00001 (1) | R1 ← MM[11] | 1000 0001 0 01 01011 |
| 00010 (2) | R2 ← R0 - R1 | 1010 0010 00 10 00 01 |
| 00011 (3) |  |  |
| 00100 (4) | MM[12] ← R0 | 1000 0010 0 00 01100 |
| 00101 (5) |  |  |
| 00110 (6) | MM[12] ← R1 | 1000 0010 0 01 01100 |
| 00111 (7) | HALT | 1111 1111 1111 1111 |
| 01000 (8) |  |  |
| 01001 (9) |  |  |
| 01010 (10) | 100 |  |
| 01011 (11) | 200 |  |
| 01100 (12) |  |  |

a. Given what the program is intended to do and the sample values in MM[10] and MM[11] what value should appear in MM[12] if this program is executed?

b. The values given for MM[10] and MM[11] are just two possible values, and they could be changed for any given execution of the program. So without assuming anything about those values, if the result of the operation at address 2 (R2 ← R0 - R1) is negative then which value, MM[10] or MM[11], is larger?

c. Which memory address contains the instruction that does MM[12] ← MM[11]?

d. Which memory address contains the instruction that does MM[12] ← MM[10]?

e. Now, complete the program in the table above by filling in the yellow highlighted line with branching instructions using our operation shorthand and the binary machine language for the operations. Note if you want to you can check your work by entering the machine language program into the K&S and running it!

*Nothing is required here, but you must complete the highlighted lines in table at the top of this question.*

🏆 5. The operations shown in the program below partially implement a program that is intended to perform the following computation:

if MM[10] != 0

MM[13] = MM[11]

else

MM[13] = MM[12]

|  |  |  |
| --- | --- | --- |
| Memory Address | Operation or  Value | Machine Language Instruction |
| 00000 (0) | R0 ← MM[10] | 1000 0001 0 00 01010 |
| 00001 (1) | R1 ← MM[11] | 1000 0001 0 01 01011 |
| 00010 (2) | R2 ← MM[12] | 1000 0001 0 10 01100 |
| 00011 (3) | R0 ← R0 & R0 | 1010 0011 00 00 00 00 |
| 00100 (4) |  |  |
| 00101 (5) | MM[13] ← R1 | 1000 0010 0 01 01101 |
| 00110 (6) |  |  |
| 00111 (7) | MM[13] ← R2 | 1000 0010 0 10 01101 |
| 01000 (8) | HALT | 1111 1111 1111 1111 |
| 01001 (9) |  |  |
| 01010 (10) | 0 |  |
| 01011 (11) | 200 |  |
| 01100 (12) | 400 |  |
| 01101 (13) |  |  |

a. What is the & operation indicated at address 3? Look back at the homework for MA1 and MA2 if necessary.

b. If R0 is zero will the result of R0 & R0 be zero? Briefly explain your answer.

c. If R0 is not zero will the result of R0 & R0 be zero? Briefly explain your answer.

d. Complete the program above by filling in the operations using our shorthand and the machine language instructions for the operations at address 4 and 6 (highlighted in yellow). Note if you want to you can check your work by entering the machine language program into the K&S and running it!

*Nothing is required here, but you must complete the highlighted lines in table at the top of this question.*

**Writing ML Programs:**

6. In the table below, write a machine language program that places the absolute value of MM[13] into MM[14]. For each step of the program, give the operation using the shorthand notation. It is not required that you translate the operation to machine language instructions for this question. Your program may assume that MM[15] contains the value 0. While it is not required, if you want to check your work or just to have the satisfaction of seeing your program run you can translate it into ML and run it using the K&S!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **MM**  **Address** | **Operation or Value** | **Optional**  **Machine Language Instructions** |  |
|  | 00000 (0) |  |  |  |
|  | 00001 (1) |  |  |  |
|  | 00010 (2) |  |  |  |
|  | 00011 (3) |  |  |  |
|  | 00100 (4) |  |  |  |
|  | 00101 (5) |  |  |  |
|  | 00110 (6) |  |  |  |
|  | 00111 (7) |  |  |  |
|  | 01000 (8) |  |  |  |
|  | 01001 (9) |  |  |  |
|  | 01010 (10) |  |  |  |
|  | 01011 (11) |  |  |  |
|  | 01100 (12) |  |  |  |
|  | 01101 (13) | *<a + or – number here>* |  |  |
|  | 01110 (14) |  |  |  |
|  | 01111 (15) | 0 |  |  |
|  |  |  |  |  |

🏆 🏆 7. **Optional Extra Challenge:** In the table below, write a machine language program that places into MM[31] the sum of all of the values between 1 and the value in MM[30]. For example, if MM[30] = 5 then MM[31] will be set to 15 (5 + 4 + 3 + 2 + 1). You assume the following:

* MM[28] holds the value 0
* MM[29] holds the value 1
* MM[30] is positive.

For each step of the program, give the operation using the shorthand notation. It is not required that you translate the operation to machine language instructions for this question. While it is not required, if you want to check your work or just to have the satisfaction of seeing your program run you can translate it into ML and run it using the K&S!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | Memory Address | Operation or  Value | Machine Language Instruction |  |
|  | 00000 (0) |  |  |  |
|  | 00001 (1) |  |  |  |
|  | 00010 (2) |  |  |  |
|  | 00011 (3) |  |  |  |
|  | 00100 (4) |  |  |  |
|  | 00101 (5) |  |  |  |
|  | 00110 (6) |  |  |  |
|  | 00111 (7) |  |  |  |
|  | 01000 (8) |  |  |  |
|  | 01001 (9) |  |  |  |
|  | 01010 (10) |  |  |  |
|  | 01011 (11) |  |  |  |
|  | 01100 (12) |  |  |  |
|  | 01101 (13) |  |  |  |
|  | 01110 (14) |  |  |  |
|  | 01111 (15) |  |  |  |
|  | 10000 (16) |  |  |  |
|  | 10001 (17) |  |  |  |
|  | 10010 (18) |  |  |  |
|  | 10011 (19) |  |  |  |
|  | 10100 (20) |  |  |  |
|  | 10101 (21) |  |  |  |
|  | 10110 (22) |  |  |  |
|  | 10111 (23) |  |  |  |
|  | 11000 (24) |  |  |  |
|  | 11001 (25) |  |  |  |
|  | 11010 (26) |  |  |  |
|  | 11011 (27) |  |  |  |
|  | 11100 (28) | 0 |  |  |
|  | 11101 (29) | 1 |  |  |
|  | 11110 (30) | *<positive number here>* |  |  |
|  | 11111 (31) | *<result here>* |  |  |
|  |  |  |  |  |

**The Stored Program Architecture:**

Today’s class introduced the stored program architecture and the instruction cycle. It is not required viewing, but if you’d like another explanation of the material from a slightly different perspective check out *The CPU and Von Neumann Architecture* by MrBrownCS:

* <https://www.youtube.com/watch?v=SbqXqQ-2ixs> (9:23)

8. What is the defining characteristic of the stored program architecture? Briefly explain why it was a big advancement over earlier computers.

9. What are the three primary components that make up the stored program architecture? Give a sentence or two describing the main function of each.

10. What are the three phases of the instruction cycle and what happens during each phase? For credit your answer must correctly describe the roles played in each phase by the program counter (PC), instruction register (IR) and the instruction interpretation unit.

11. What are the three primary components that make up a central processing unit and what is the main function of each?

It is not required viewing, but if you would like to see how some of the components of the stored program architecture relate to what you see when you open up a real computer, check out the video *How To Identify The Components Inside Your Computer* from Gadgets and Gears:

* <https://www.youtube.com/watch?v=yRmPTbGBqVI> (5:28)

**Another Model Computer:**

We have been working extensively with the K&S computer. The K&S reflects one set of design choices that the computer designer would have had to make to build the computer. But they are far from the only possible choices. To help reinforce today’s material, to emphasize the generality of the ideas, as well as to also help you see some of what might be the same and different between different computer designs watch the video below:

* *The Central Processing Unit (CPU)* with Carrie Ann from Crash Course Computer Science.
  + <https://www.youtube.com/watch?v=FZGugFqdr60> (11:37)

In this video Carrie Ann introduces a different computer design and explains how it uses the fetch/decode/execute cycle to run programs. The following questions will ask you to compare her computer to our K&S computer.

12. The following questions ask you to compare Carrie Ann’s model computer in the video to our K&S computer:

a. The *word size* of a computer is defined by the number of bits that can be processed by a single instruction. It typically is the same as the number of bits that are stored in a single register. What is the word size of Carrie Ann’s computer? What is the word size of our K&S computer?

b. The K&S computer has a Program Counter (PC). What is the register in Carrie Ann’s computer that performs the same function as the K&S program counter?

c. Different computer designers will make different decisions about what operations to include and what binary OpCode to use for each of those operations. This is what prevents ML programs for one type of computer from running directly on another (e.g. iPhone vs Andriod). What opcode does Carrie Ann’s computer use for loading a value from memory into her machine’s A register? What is the corresponding opcode that we would use to load a value from memory into R0 on the K&S computer?

d. Which computer, the K&S or Carrie Ann’s can have more main memory? Why? Hint: Consider the number of bits in the memory addresses for each computer.

e. To do an addition, the computer needs to know what two values to add and where to put the result. The ADD instruction for the K&S has three operands that tell the computer which registers to add and where to put the result.

How many operands does the ADD instruction for Carrie Ann’s computer have? How do they communicate which registers to add and where to put the result?

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.