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

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

**16 – Branching and Looping**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2022

Prof. Grant Braught

**Name:**

Today’s class addressed the need to include branching instructions in the instruction set for our machine language machine. We saw several examples showing that these allow us to implement high-level language control structures such as if and for. In fact, the branching instructions that we have are sufficient to implement all of the standard HLL control structures that we are used to using if, if/else, while, for, do, switch/case and compound logical conditions with logical AND (&&) and OR (||) operations in them. Today’s activities will give you some hands-on practice with these branching instructions and with implementing HLL control structures in assembly language.

A reference is included here for all of the assembly language instructions we know so far.





**The Branching Instructions:**

While we saw examples of some of the branching instructions in action, we certainly did not cover them all in detail. Use the reference table on the previous page to answer the following questions.

🔑 1. Give the “Instruction Format” of the instruction that you would use to implement each of the following conditions.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Condition** | **Opcode** |  |
|  | a. X < 7 |  |  |
|  | b. Y >= Z |  |  |
|  | c. 5 > Y |  |  |
|  | d. Q > 0 |  |  |
|  | e. N != M |  |  |
|  |  |  |  |

🔑 2. Give a complete assembly language instruction including specific registers (i.e. like those in the “Example” column in the tables above) that will branch to the label TARGET if the specified condition is true:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Operation** | **Instruction** |  |
|  | a. R1 is odd |  |  |
|  | b. R0 < R1 |  |  |
|  | c. R7 == R8 |  |  |
|  | d. R9 is nonzero |  |  |
|  | e. R12 >= R3 |  |  |
|  |  |  |  |

**A Branching Program:**

🔑 3. One of the examples from class showed how to express a high-level language program for computing the absolute value of a number in assembly language. Give a modified version of that program that computes the negative of the absolute value ( -|x| ). That is, the result is always a negative value. The high-level language code for this would be:

int x

Read x

if (x > 0) {

x = -x;

}

Print x

Be sure to assemble your program to ensure that it is syntactically correct. Also be sure to run your program on the machine simulator to and check that it works correctly with both positive and negative inputs. Give your assembly language program as you answer.

**The if / else Statement:**

In the class we saw the generalized structures for translating if statements into assembly language. There are similar structures for converting all of the high-level language constructs that we know. One generalized structure for an if/else statement is

if (condition) {

… if body …

}

else {

… else body …

}

… program continues…

Branch to IFBODY if condition is true

JUMP ELSE

IFBODY: … if body …

JUMP END

ELSE: … else body …

END: … program continues…

🔑 4. The high-level language program reads in two numbers and prints out the larger value.

int a;

int b;

Read a;

Read b;

if (a > b) {

Print a;

}

else {

Print b;

}

Use the generalized if/else structure to translate this high-level language program into assembly language. Be sure to assemble your program to ensure that it is syntactically correct. Also be sure to run your program on the machine simulator to ensure that it works correctly. Give your assembly language program as your answer.

**The while Loop:**

Like the if, if/else and for loops we can specify a generalized structure for a while loop also. One generalized structure for a while loop is:

while (condition) {

… while body …

}

… program continues…

JUMP to COND

TOP: … while body …

COND: JUMP to TOP if condition is TRUE

… program continues…

Notice that like the for loop example from class, we place the test for the loop condition at the end of the while loop.

🔑 5. The high-level language program below reads in a value n and then uses a while loop to compute and display the value of n\*7.

int n;

int total;

Read n;

total = 0;

while (n > 0) {

total = total + 7

n = n - 1

}

Print total

Use the generalized while structure to translate this high-level language program into assembly language. Be sure to assemble your program to ensure that it is syntactically correct. Also be sure to run your program on the machine simulator to ensure that it works correctly. Give your assembly language program as your answer.

**Compound Logic Conditions:**

Thus far all of the conditions we have looked at in our if and while loops have been simple conditions (e.g. x > 0, y <= z). But we know that it is also possible to have compound logical conditions using the logical AND (&&) and logical OR (||) operators. For example:

if (x >= 0 || x < 10) {…}

while (y > 7 && z <= 5) {…}

The questions in this section explore how to implement these types of compound conditions in assembly language.

🔑 6. The following assembly language program is intended to read a value from standard input and print a 1 if the value is less than 10 OR greater than 20, otherwise the program prints a 0. For example, if the input is 5 the program should print 1, if the input is 15 then it should print 0, and if the input is 25, it should print 1.

LOAD R0 STDIN

LOAD R1 #10

LOAD R2 #20

\* Add branches here.

YES: LOAD R3 #1

JUMP PRINT

NO: LOAD R3 #0

PRINT: STORE R3 STDOUT

HALT

Give the branching instructions that would need to be added to make this program function as desired. Be sure to assemble and run your program in the machine simulator to check that it works correctly.

🏆 7. Read a value from standard input and if it is between 50 and 75 (inclusive) then output 1, otherwise output 0. Hint: You can use one of DeMorgan’s Identities here and then use the same approach you used in #6!

LOAD R5 STDIN

LOAD R6 #50

LOAD R7 #75

\* Add branches here.

YES: LOAD R8 #1

JUMP PRINT

NO: LOAD R8 #0

PRINT: STORE R8 STDOUT

HALT

**Assembly Language Challenge:**

🏆 🏆 8. Write an assembly language program that reads a specified number of positive integer inputs from standard input and outputs the largest value that was read. For example, if standard input contains:

5 3 2 7 9 4

The 5 indicates that there will be 5 additional values. Then program should read those 5 additional values and print the largest of them (e.g 9 in this case). One strategy here would be to outline the program in high-level language code and then translate it into assembly code. Be sure to assemble, run and test your program to ensure that it is syntactically correct and that it works correctly. The program should work for any valid input not just the one shown above.

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.