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

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

**19 – Implementing Functions**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2022

Prof. Grant Braught

**Name:**

In the prior class we saw how to call functions and how to pass arguments by pushing them onto the stack. Today’s class built on that one by showing how functions are implemented. We started by seeing how functions can use indirect addressing with the stack pointer (R13) to access the argument values. We then saw that because functions change registers, we have to be careful that those changes do not create side effects in the calling code. We learned how to use the stack within the function to prevent these side effects. Combined with the use of CALL and RET and placing the return value in R14, we now have a full implementation of the high-level language function calling abstraction. The following activities will give you some hands-on experience with both using and implementing this abstraction.

**Implementing Functions:**

🔑 1. In the prior homework you were asked to write code that called the diff function, which had the following signature:

int diff(int p, int q) // Compute p - q

Below is an implementation of the diff function with a few blanks highlighted in yellow. Fill in the blanks in the implementation of the diff function so that it properly accesses the argument values that are being passed to it. You can check your solution by placing this implementation of diff into a text file with your code with your code from the prior assignment that calls it, and then assembling and running it.

DIFF: LOAD R0 R13 \_\_\_\_ \* get p

LOAD R1 R13 \_\_\_\_ \* get q

SUB R2 R0 R1

MOV R14 R2

RET

🔑 2. The implementation of the diff function in question 1 does not save and restore the values of the registers that it changes. Thus, it would have side effects. We’ll fix that in this question.

a. Which registers are changed by the implementation of diff in question #1 above?

b. A new implementation of diff that will not have side effects is given below with a few blanks highlighted in yellow. Fill in these blanks to create an implementation of the diff function that does not have side effects. You can check your solution by replacing the implementation of diff in your text file with this one, and then assembling and running it.

DIFF: PUSH \_\_\_

PUSH \_\_\_

PUSH \_\_\_

LOAD R0 R13 \_\_\_\_ \* get p

LOAD R1 R13 \_\_\_\_ \* get q

SUB R2 R0 R1

MOV R14 R2

POP \_\_\_

POP \_\_\_

POP \_\_\_

RET

🔑 3. After studying the implementation of the diff function above a programmer realizes that it can be implemented more efficiently using only two registers instead of three. Fill in the blanks in this more efficient implementation of the diff function given below. You can put this version into your text file in place of the earlier version and then assemble and run it to check your answer.

DIFF: PUSH \_\_\_

PUSH \_\_\_

LOAD R3 R13 \_\_\_\_ \* get p

LOAD R4 R13 \_\_\_\_ \* get q

SUB R3 R3 R4

MOV R14 \_\_\_

POP \_\_\_

POP \_\_\_

RET

🔑 4. In the prior homework you were asked to write code that called the flipSign function, which had the following signature:

int flipSign(int n) // change the sign of n

The statements below give one possible implementation of the flipSign function with some of the statements omitted. Replace the yellow highlighted comments in the following code with the statements that are necessary to create a side effect free implementation of flipSign. Be sure to combine your answer in a text file with code that calls flipSign and then assemble and run it in the simulator to test your work.

FLIP: \* Save modified registers

\* Load argument n into R3

LOAD R5 #0

SUB R3 R5 R3 \* Assumes R3 holds n.

\* Set return value

\* Restore registers

RET

🔑 5. In the prior homework you were asked to write code that called the add3 function, which had the following signature:

int add3(int a, int b, int c) // Compute a + b + c

Write a side effect free implementation of the add3 function. A good process for writing functions like this is:

1. Write the statements to get the arguments.
2. Write the body of the function.
3. Set the return value.
4. Determine which registers your function modifies.
5. Add the PUSH and POP instructions to save and restore the modified registers.
6. Adjust the statements that get the arguments to account for the additional pushes and pops.

Be sure to combine your answer in a text file with code that calls add3, then assemble and run it in the simulator to test your work.

**Putting it All Together:**

In class we have seen the following example several times:



In this section you will play the role of the compiler and translate this high-level language code into an equivalent assembly language program.

🏆 6. Let’s start by implementing and testing the max function. Then in the next question you’ll implement and test the max3 function.

a. Give an implementation of the max function that is side effect free.

b. Give a main function that reads two values from standard input, passes them to the max function and then prints the result to standard output. Be sure to combine this code with your implementation of max, assemble it and test it using the machine simulator to be sure it works.

🏆 7. Give an implementation of the max3 function that is side effect free. Hints:

* Have your max3 function call your max function twice. Each call is the same, push the arguments you want to pass, call the function, pop the arguments, and then use the return value in R14.
* Remember that R12 is modified by the CALL instruction. So, because max3 uses CALL, you will need to be sure to save and restore R12, just like any other register that max3 modifies. (See question #11 in Homework 18).

🏆 8. Write the main function that reads 3 values from standard input, passes them to the max3 function and then prints the result to standard output. Combine this code with your implementation of max3 and max, assemble it and test it using the machine simulator to be sure it works.

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.