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

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

**18 – Calling Functions**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2022

Prof. Grant Braught

**Name:**

Today’s class extended the collection of high-level language features that we now know can be implemented in assembly language, and thus in machine language via an assembler. To the basic operations, standard control structures and arrays, we have added function calling. The ideas and techniques introduced today showed how we can call a function in assembly language using the CALL instruction. We also saw how a function can return a value in R14 and how it jumps back to where it was called from using the RET instruction. We learned about stacks and how they will be used to pass arguments to a function using the PUSH instruction and how we use the POP instructions to clean up afterwards. In today’s activities you will gain experience with function calls, while also practicing some of the earlier assembly language topics (e.g. control structures). In the next class, we will finish off assembly language by seeing how we actually implement the functions themselves.

**Assembly Instruction Reference:**

A full reference to all of the assembly language instructions that we have seen so far is provided here for your convenience as you work though the activities.

*Arithmetic and Logic Instructions:*



*Data Movement Instructions:*



*Branching Instructions:*



*Function Calling Instructions:*



**Reserved Registers:**

Up until now we had been working exclusively with R0-R11 as the general-purpose registers. It was mentioned that R12-R15 are reserved registers that are used for specific purposes in the machine. Today we saw the use of three of those reserved special purpose registers R12, R13 and R14.

🔑 1. For each of these registers, give a sentence or two of explanation of their purpose based on the examples from today’s class:

a. R12

b. R13

c. R14

**Calling a Function:**

🔑 2. Consider the following high-level language program:

main() {

n = nine()

print n;

}

int nine() {

p = 16;

r = 7;

s = p – r;

return s;

}

a. Give a translation of the nine function into assembly language. Use the label NINE for the first instruction of your function.

b. Give an assembly language translation of the main program that calls your nine function from part a. Hint: Follow the similar example in the class slides.

c. Your solution to part b should contain a CALL instruction. If that CALL instruction were stored at memory address 132, what value will be placed into R12 when the CALL instruction is executed?

d. When will the value that is placed into R12 by the CALL instruction be used?

**The Stack:**

The *stack* is a section of main memory that is set aside to be used to pass arguments to functions (and some other things we will see in the next class).

🔑 3. What directive allocates the main memory space for the stack?

🔑 4. Write a directive that allocates enough stack space for 512 bytes.

🔑 5. Write a directive that allocates enough stack space for 100 **integer values**.

🔑 6. Consider the assembly program that use a stack:

.stacksize 20

LOAD R0 #5

LOAD R1 #10

LOAD R2 #20

PUSH R0

PUSH R1 \* LINE A

PUSH R2

POP R4 \* LINE B

POP R3

POP R5 \* LINE C

HALT

a. What values will be in R3, R4 and R5 when this program completes?

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Register** | **Value** |  |
|  | R3 |  |  |
|  | R4 |  |  |
|  | R5 |  |  |
|  |  |  |  |

b. Assume that R13, (i.e. the stack pointer), is initialized to the address 200 when this program begins. What value (i.e. address) will be in R13 at each of the points indicated in the table below? Hint: Trace the program like the example we did in class.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Point** | **R13 Value** |  |
|  | Just before LINE A executes |  |  |
|  | Just after LINE A executes |  |  |
|  | Just before LINE B executes |  |  |
|  | Just after LINE B executes |  |  |
|  | Just after LINE C executes |  |  |
|  |  |  |  |

**Calling Functions with Parameters (Passing Arguments):**

The high-level language function calling mechanism is an abstraction. For example, if we have the sum function from the class slides with the signature:

int sum(int a, int b)

Then we know we can call this function by providing values for the parameters and saving the return value as shown here:

Read a;

Read b;

c = sum(a,b);

It is not necessary for us to know the internal details of the how the sum function works.

The same is true in assembly language. We only need to know what parameters the function requires so that we can push them onto the stack and clean them up and that the result will be returned in R14 so that we can save that value for future use. For example, the sum function can be called as we did in class:

LOAD R0 STDIN \* x

LOAD R1 STDIN \* y

PUSH R0 \* Pass x as a

PUSH R1 \* Pass y as b

CALL SUM

POP R15

POP R15

MOV R3 R14 \* get result.

🔑 7. Assume that an assembly language program begins as follows:

.stacksize 100

LOAD R3 STDIN \* x

LOAD R4 STDIN \* y

LOAD R5 STDIN \* z

\* Your code for part a, b or c would go here…

a. Consider a function diff that computes the difference between its two parameters. This function might have the high-level language signature shown below:

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

Assume that an assembly language implementation of diff has been written and the label DIFF is the address of the first instruction in the function.

Give some assembly language statements that that could be added to the above program to call the DIFF function passing x and y to the function as the parameters p and q and saving the return value in R8.

**Note: This question is asking only for statements to call diff, clean up, and use the result. It is not asking for the implementation of diff.**

b. Consider a function flipSign that changes the sign of its parameter. For example flipSign(-7) would return 7 and flipSign(7) would return -7. This function might have the high-level language signature shown below:

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

Assume that an assembly language implementation of flipSign exists and the label FLIP is the address of the first instruction in the function.

Give some assembly language statements that that could be added to the above program to call the flipSign function passing z to the function as the parameter n and saving the return value in R7.

**Note: This question is asking only for statements to call flipSign, clean up, and use the result. It is not asking for the implementation of flipSign.**

c. Consider a function add3 that computes the sum of its three parameters. This function might have the high-level language signature shown below:

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

Assume that an assembly language implementation of add3 exists and the label ADD3 is defined and is the address of the first instruction in the function.

Give some assembly language statements that that could be added to the above program to call the ADD3 function passing x, y and z to the function as the parameters a, b and c and saving the return value in R9.

**Note: This question is asking only for statements to call add3, clean up, and use the result. It is not asking for the implementation of add3.**

🏆 8. One of the advantages of using a stack to pass arguments to a function is that it does not matter which registers contain the values that should be passed to the function. Rewrite your answer to question #7a assuming that x is in R9 and y is in R2.

**Some Complications:**

While we have the basics of function calling worked out there are some complications that will arise that we will need to deal with. We will see how to deal with these complications in the next class. However, for now the following questions will give you a little insight into these complications.

**Function Side Effects:**

When we call a function in a high-level language, we expect that it will not have any unexpected side effects (e.g. changing the value of a variable in the calling code).

🏆 9. Consider the high-level language program shown below:

main() {

a = 3;

n = two();

print a; // Line 1

}

int two() {

a = 1;

a = a + a;

return a;

}

What output would you expect Line 1 to generate? Hint: Remember variables in functions are local variables.

🏆 10. Consider the following assembly language translation of the high-level language program from question #9:

.stacksize 100

LOAD R1 #3

CALL TWO

STORE R1 STDOUT \* Line 1

HALT

TWO: LOAD R1 #1

ADD R1 R1 R1

MOV R1 R14

RET

a. What output will be generated by Line 1 in this program?

b. Why does this translation give a different result than the program in question #9?

c. One unsatisfactory solution to this complication would be to ensure that the main program and the function use different registers for their computations. Why might that not be a satisfactory solution? Hint: See question #8.

**Nested Function Calls:**

It is quite common for one function to make a call to another function, and for this to even go on further for several more calls to additional functions. This situation introduces a subtle issue that must be dealt with in order for these calls to work correctly.

🏆 11. Consider the assembly language program below.

.stacksize 500

LOAD R0 #0

STORE R0 STDOUT \* Line 1

CALL ONE

STORE R0 STDOUT \* Line 5

HALT

ONE: LOAD R1 #1

STORE R1 STDOUT \* Line 2

CALL TWO

STORE R1 STDOUT \* Line 4

RET

TWO: LOAD R2 #2

STORE R2 STDOUT \* Line 3

RET

a. Without assembling and running this program, what output do you think would be generated by this program?

b. Assemble and run the program. What output does it actually generate?

c. Which of the labeled lines (e.g. Line 1) is generating the repeated output that results?

🏆 🏆 d. Briefly explain why this program behaves in the somewhat unexpected way that it does. Hint: It has to do with the CALL instruction and the Return Address (R12).

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.