

CSSE 132 – Introduction to Systems Programming
 Rose-Hulman Institute of Technology

Quiz 10 Stack/Procedure Call

Name (Print): _____ RHIT Username: _____

1. Answer the following questions about your findings in Lab 3 regarding function input arguments, return values, and procedure calls. (Note: the line numbers below correspond to the provided Assembly code.)

```

1 int my_func(int x){
2     return x+1;
3 }
4 int main() {
5     int x = 56;
6     x = my_func(x);
7     int y = 78;
8 }

```

Instruction	Equivalents
str x30, [sp, -32]!	sub sp, sp, 32 str x30, [sp]
ldr x30, [sp], 32	ldr x30, [sp] add sp, sp, 32

```

1 my_func:
2     sub    sp, sp, 16
3     str    w0, [sp, 12]
4     ldr    w0, [sp, 12]
5     add    w0, w0, 1
6     add    sp, sp, 16
7     ret
8 main:
9     str    x30, [sp, -32]!
10    mov    w0, 56
11    str    w0, [sp, 28]
12    ldr    w0, [sp, 28]
13    bl     my_func
14    str    w0, [sp, 28]
15    mov    w0, 78
16    str    w0, [sp, 24]
17    mov    w0, 0
18    ldr    x30, [sp], 32
19    ret

```

- 1.1 How much space is **allocated** by the compiler to the functions? (Hint: Refer to the Assembly code.)

main: Bytes my_func: Bytes

- 1.2 How much space is actually **used** by the functions? (Hint: Refer to the C and Assembly code to see how much space is used for local variables, input arguments, return address, etc.)

main: Bytes my_func: Bytes

- 1.3 How is the input argument x passed from main to my_func?

Before calling my_func (line 13), the value of the input argument is first loaded into .

After entering my_func (at line 3), this value is read from and assigned to the C input variable .

1.4 How is the return value of `my_func` passed to `main`?

Before returning to `main` (line 7), the return value is first put into register .

After returning to `main` (at line 14), this value is read from .

1.5 Branch and link: What does the `bl my_func` instruction do?

With a function call, we need to 1) branch to the start of that function and 2) remember where to return to after that function completes. `bl` stands for “branch and link”, and

it will 1) and 2) store the return address (i.e.,

the address of the next instruction after `bl`) to .

2. Now consider the following code where a function calls another function.

```

1 void do_nothing(){
2 }
3 int my_func(int x){
4     do_nothing();
5     return x+1;
6 }
7 int main() {
8     int x = 56;
9     x = my_func(x);
10    int y = 78;
11 }

```

```

1 do_nothing:
2     ret
3 my_func:
4     str    x30, [sp, -32]!
5     str    w0, [sp, 28]
6     bl     do_nothing
7     ldr    w0, [sp, 28]
8     add    w0, w0, 1
9     ldr    x30, [sp], 32
10    ret
11 main:
12     str    x30, [sp, -32]!
13     mov    w0, 56
14     str    w0, [sp, 28]
15     ldr    w0, [sp, 28]
16     bl     my_func
17     str    w0, [sp, 28]
18     mov    w0, 78
19     str    w0, [sp, 24]
20     mov    w0, 0
21     ldr    x30, [sp], 32

```

2.1 What is the value of `lr (x30)` right after running each line? (Answer with a line number in Assembly code.)

Line 16:

Line 6:

Line 9:

2.2 What is the purpose of the `str` and `ldr` in lines 4 and 9?

2.3 Does every function need to store the return address on the stack? For what kind of functions is it necessary?

No. It is only necessary for functions that

2.4 What things are stored on the *stack*?

, , .

2.5 In a procedure call, the following steps are done by the **caller** (e.g., **main** when it calls **my_func**). Write down the line numbers in Assembly code to indicate which line completes each of the steps below.

• Line : Prepare the function input arguments by setting registers **w0**, **w1**, ...

• Line : Branch to the callee by calling **bl**, which saves the return address to **lr**

• Line : After the function returns, retrieve the return value from register **w0**

2.6 In a procedure call, the following steps are done by the **callee** (e.g., **my_func** when it is called by **main**). Write down the line numbers in Assembly code to indicate which line completes each of the steps below.

• Line : Store the return address in **lr** on the stack

• Line : Assign the values of input arguments from registers **w0**, **w1**, ...

• Line : Move the function return value to register **w0**

• Line : If the return address is stored on the stack, read the return address from the stack

• Line : Branch back to the caller

3. **Reflection:** What is one thing from this topic that you found interesting, OR what is one question you are now curious to learn more about?