LAB D

Lab Objectives

In this lab you will learn how to use the stack and implement functions that call other functions. After completing this lab, you should be able to

1. Implement simple if then structure in assembly
2. Implement multiple if then elif
3. Implement a compound if then (if (A &&B)).

This lab is two parts, in part 1, we will walk you through implementing a simple function that calls another function, that includes the memory layout of your code. In part 2, you will implement from-scratch two a function that calls another function recursively XXXX

Part 1

Write an assembly program that reads an array of double words, for each word it swaps the low order two bytes and store it in another array, the swap should be done by a separate function. In C our program looks like

for example if the number is (in hex)

A0125087B1CD45D6

After the swap

A0125087B1CDD645

The rules are

- Parameters are passed to the procedure in x10-x17
- Return address is in x1
- Return value in x10
- x8-x9 and x18-x27 are preserved across function calls
- x5-x7 and x28-x31 are not preserved by the callee

Let us start with the procedure.

In pseudo code

a=255 (in binary, that is 0x00000000000000FF, that is 8 1's in the least significant position and is used to extract the bits in the least significant byte).

The number passed to the procedure is in I

m=i&a // m has only the least significant byte of I

n=i >> 8 // n has I shifted by 8, now the second least significany byte is in the least significant byte position

n=n&a // n has the least significant byte

i=i>>16

i=i<<8

i=i |m | is the logical OR

i=i<<8

i=i|n

return i

In Assembly, the code is, the number is in x10 refer to as (i)

```
addi  x5, x0, 255          // x5 is a 0000 ..011111111
and   x6, x10, x5          // x6 is m=i&a, m has lest significant byte
srai  x7, x10, 8      //x7 is I shifted to left by 8
and   x7, x7, x5      //x7 is the second least sig. Byte of I
srai  x8, x10, 16          // shift I by 16 put it in x8
slli  x8, x8, 8
or    x8, x8, x6           //append the second least significant byte
slli  x8, x8, 8
or    x8, x8, x7
```

In this code, we used registers x5, x6, x7, x8. According to the rules, x5, x6, x7 are not suppose to be preserved, but x8 is **we have to save x8 when we implement the function**

Also, the function requires 9 instruction, that is a total of 36 bytes. We will allocate 50 bytes

Now, we implement the main program

The main program reads a double word from the input, calls the processor swap, then display the result to the output

The programs

```
ecall     x5, x0, 5 //read the input to x5
add  x10, x5, x0    //put the parameter in x10
jal  x1, myswap
ecall     x0, x10, 1     //print the value returned from the functions
ecall     x0, x5, 1 //print the original value
```

There is **one error** here, the value was in x5, but x5 is not guaranteed to have the same value after the function returns. X5 must be saved before we call the function, and restored after returning from the function. The program requires 5 instructions (4 more for pushing and popping the stack) just to be in the safe side, we will allocate 60 bytes for it.

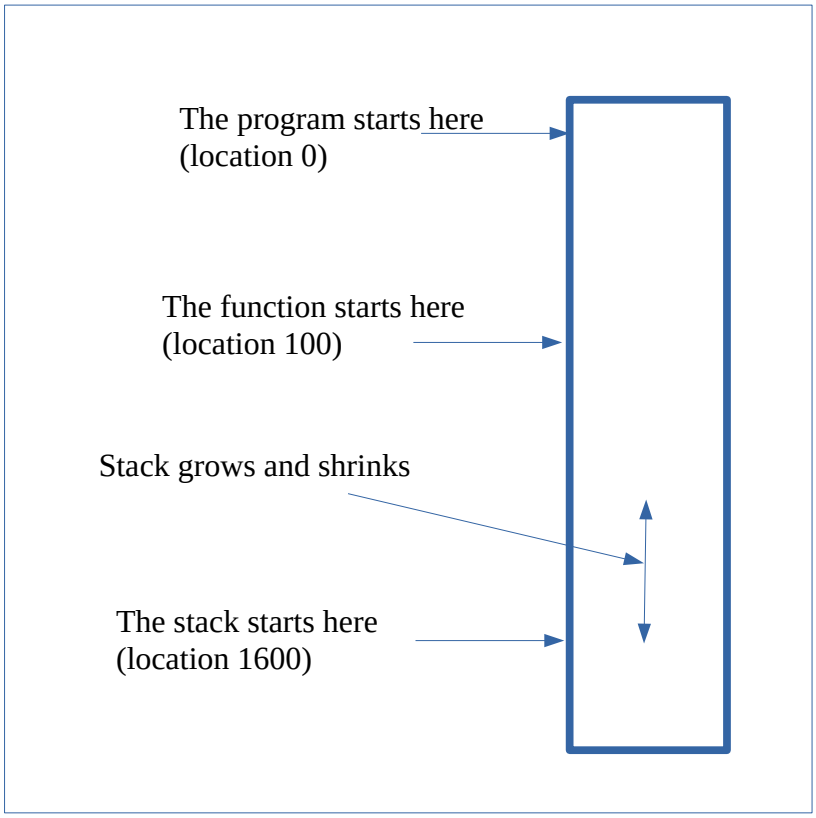
Memory Layout

We will use the same convention for the stack as in the lecture and the textbook. The stack grows towards low memory address

Locations 0 – 59 used for the main programs.

Locations 96- up used for the functions.

The stack starts at location 1600, in this case the stack can grow from 1600 all the way to 150 without affecting the code.



The Whole Program

Here we show the whole program, the code in bold red indicates the added instructions to manipulate the stack and save/restore registers

```
addi x2, x0, 1600   //initialize the stack to 1600, x2= stack pointer
ecall     x6, x0, 5 //read the input to x5
add  x10, x6, x0    //put the parameter in x10
addi x2, x2, -8          //make room to store x5
sd   x6, 0(x2)
jal  x1, myswap
ld   x6, 0(x2)      //restore x5 rom the stack
addi x2, x2, 8      //pop the stack
ecall     x0, x10, 2 //print the value returned from the functions
ecall     x0, x6, 2 //print the original value
ORG  96
myswap:
addi x2, x2, -8
sd   x8, 0(x2)
addi x5, x0, 255         // x5 is a 0000 ..011111111
and  x6, x10, x5         // x6 is m=i&a, m has lest significant byte
srai x7, x10, 8          //x7 is I shifted to left by 8
and  x7, x7, x5          //x7 is the second least sig. Byte of I
srai x8, x10, 16         // shift I by 16 put it in x8
slli x8, x8, 8
or   x8, x8, x6          //append the second least significant byte
slli x8, x8, 8
or   x8, x8, x7
add  x10, x0, x8
ld   x8, 0(x2)           // pop x8 frm the stack
addi x2, x2, 8
jalr x0, 0(x1)
```

Part 2

In this part, you are asked to write a function that recursively calls itself.

The calling program is a simple one that reads one integer from the input, call the function, and display the result on the output window.

The function is a modification of the factorial function (just to make things interesting and prevent you from copying it from the textbook.

The function is called not_really_fac

if i<= 3 not_really_fac(i) =1

else not_really_fac(i) = 2 * not_really_fac(i-2) +1