

EECS2021


LAB D

Lab Objectives

In this lab you will learn how to use the stack and implement procedures. This lab has two parts, in part 1, we will walk you through implementing a simple procedure that includes the memory layout of your code. In part 2, you will implement from-scratch a procedure that calls itself recursively.

Part 1

Write an assembly program that reads a doubleword from input, it then swaps the two least-significant bytes of the doubleword, and display the swapped and original values to the output. The swap should be done by a separate procedure. 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 procedure 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

$n = i \gg 8$ // n has i shifted to right by 8 bits, now the second least significant byte is in the least significant byte position

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

$i = i \gg 16$

$i = i \ll 8$

$i = i | m$ // $|$ is the logical OR

$i = i \ll 8$

$i = i | n$

return i

In Assembly, the code is (assume that the number in $x10$ is i)

```
myswap:
addi x5, x0, 255      // x5 is a 0000...011111111
and x6, x10, x5        // x6 is m=i&a, m has the least significant byte
srai x7, x10, 8        // x7 is i shifted to right by 8 bits
and x7, x7, x5        // x7 is the second least significant Byte of i
srai x8, x10, 16       // shift i to right by 16 bits; 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          //append the first least significant byte
```

In this code, we used registers $x5$, $x6$, $x7$, $x8$. According to the rules, $x5$, $x6$, $x7$ are not supposed to be preserved, but $x8$ is. **We need to save $x8$ when we implement the procedure**

The procedure requires 6 more instructions (4 to push and pop the stack, 1 for writing the return value in $x10$, and 1 for returning to the main program). That will be a total of 15 instructions, or 60 bytes.

Now, we implement the main program.

The main program reads a doubleword 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 procedure
ecall x0, x5, 1       //print the original value
```

There is **one error** here, the original value was in x5, but x5 is not guaranteed to have the same value after the procedure returns. X5 must be saved before we call the procedure, and restored after returning from the procedure. The program requires 5 more instructions (1 for initializing the stack, 4 for pushing and popping the stack). Just to be in the safe side, we will allocate 60 bytes for it.

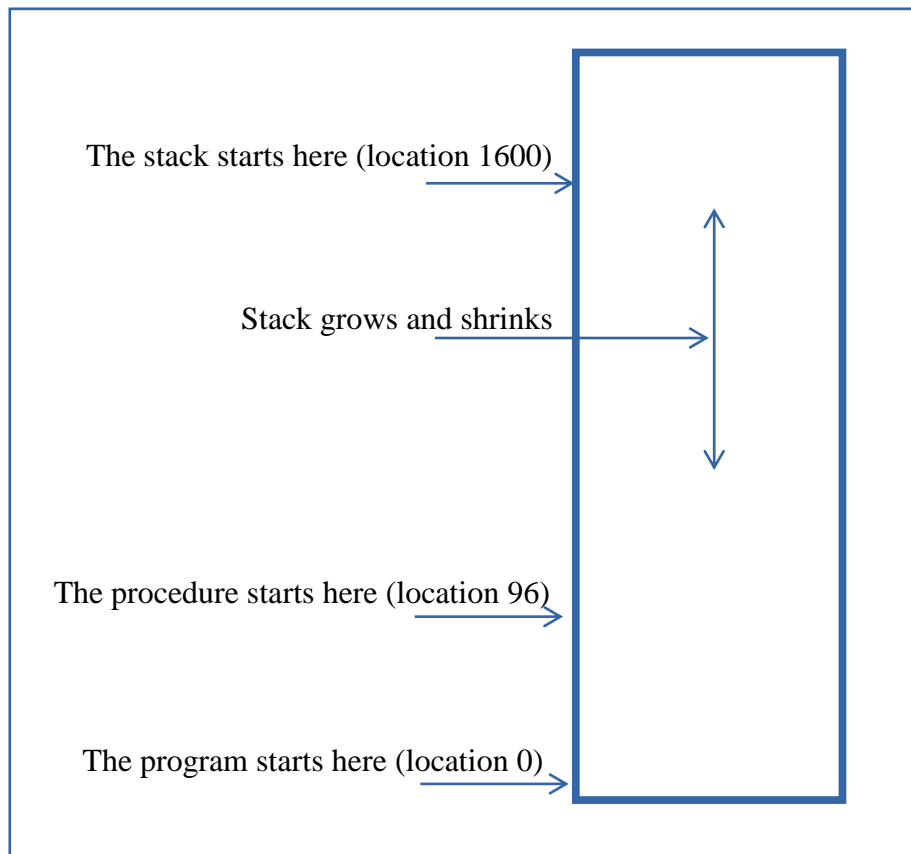
Memory Layout

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

Locations 0 – 59 in the memory is used for the main programs.

Locations 96 – up in the memory is used for the procedures.

The stack starts at location 1600 in the memory, in this case the stack can grow from 1600 all the way to 160 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 x5, x0, 5    //read the input to x5
add x10, x5, x0    //put the parameter in x10
addi x2, x2, -8    //make room to store x5
sd x5, 0(x2)
jal x1, myswap
ld x5, 0(x2)      //restore x5 rom the stack
addi x2, x2, 8    //pop the stack
ecall x0, x10, 2   //print the value returned from the procedure
ecall x0, x5, 2    //print the original value
ORG 96
myswap:
addi x2, x2, -8
sd x8, 0(x2)
addi x5, x0, 255   // x5 is a 0000 ...0111111111
and x6, x10, x5    // x6 is m=i&a, m has the least significant byte
srai x7, x10, 8    //x7 is i shifted to right by 8 bits
and x7, x7, x5     //x7 is the second least significant Byte of i
srai x8, x10, 16   // shift i to right by 16 bits, 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      //append the first least significant byte
add x10, x0, x8    // write the procedure result to x10
ld x8, 0(x2)      // pop x8 from the stack
addi x2, x2, 8
jalr x0, 0(x1)
```

Part 2

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

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

The procedure is called `rec_func`

if $i \leq 3$ `rec_func(i) = 1`

else `rec_func(i) = 2 * rec_func(i-2) + 1`

Note: After completing the lab, you need to show the TA that your code is working (This should be done for all labs).

Lab Report:

Your lab report should be submitted in pdf format (You can use LibreOffice; type `libreoffice` in the terminal). Name your report file **LabDReport.pdf**

Your report should include:

1. cover page with your name and student ID
2. The assembly code for part 2 program.
3. For the program, put a screen shot showing the simulator (all windows, including memory, registers, etc.) after the compile (before the run) and after the run.
4. Any further explanation you might want to add

Submit it as: **`submit 2021 labD LabDReport.pdf`**