Lab2: Stacks, Subroutines and C
ECSE 324 – Computer Organization
Fall 2019

Introduction
In this lab you will learn how to use subroutines and stack, program in C and call code written in assembly from code written in C.

1. Subroutines

1.1 The stack

The stack is a data structure that can be helpful for programming cases when there are not enough registers for a program, which uses only registers to store data. You will also need to make use of the stack when calling subroutines to save the state of the code outside the subroutine.

The stack is a memory region within the program. Memory is allocated for stack operations if program contains subroutines. Stacks are used for temporal storage of data such as local variables used by functions. Data is loaded onto stack and taken from stack using PUSH and POP instructions. PUSH and POP can be used as stand-alone instructions or can be implemented by other assembly instructions such as Load and Store.

Stack grows when data is pushed onto stack. The stack can grow UP (stack implemented in descending order) or DOWN (stack implemented in ascending order). The memory location where the next word is to be stored in the stack is indicated by the Stack Pointer (SP holds that memory location). The address could be pointing to the current (last) item in the stack or the next available memory slot for the item. If SP is currently pointing to the last item in the stack, SP will be decreased (in case of descending stack) or increased (ascending stack) and only then the item will be placed in the stack. If SP is currently pointing to the next empty memory location, the data will be first placed and only then SP will be decreased (descending stack) or increased (ascending stack).

Write a short program in assembly to:

a) PUSH values: 2, 3, 4 onto stack using R0 register
b) POP the above values from the stack into R1, R2 and R3 registers.

Show to TA contents of memory after pushing onto stack values 2, 3 and 4, and content registers R1, R2 and R3 after popping the values from the stack.

1.2 The subroutine calling convention

The convention which we use for calling subroutine in ARM assembly is as follows.

The caller must:

- Move arguments are placed into R0 through R3. If more than four arguments are required, the caller should push the arguments onto the stack.
- Call the subroutine using BL

The callee must

- Move the return value into R0
- Ensure that the state of the processor is restored to what it was before the subroutine call
- Use BX and LR return to the calling code.

The state can be saved and restored by pushing registers R4 through LR onto the stack at the beginning of the subroutine and popping R4 through LR off the stack at the end of the subroutine.

Convert your program from Lab 1 for finding the **min** of an array into a subroutine calculating the min value of 3 numbers. The subroutine should return the min value to R0.

1.3 Recursive subroutine to calculate factorial of an integer number

The recursive factorial function for integer numbers is defined as follow:

n! = n * (n-1)!
0! = 1

The recursive algorithm for calculating the factorial is:

```
Factor (n):
  if (n == 0)
    return 1
  else
    return n*factor(n-1)
```

Write an assembly program that computer the n! Your program should have a main section which calls the factorial subroutine recursively. The subroutine should implement the recursive algorithm above.

2. C Programming

Assembly language is useful for writing fast, low-level code, but it can be tedious to work with. Often, high-level languages like C are used instead.

2.1 **C Program**

We will start with a short C programming exercise.

- Create a new project, performing the same steps as you performed for an assembly project. However, when the New Project Wizard asks what program type you would like, select “C Program”. Click on the box next to “Include a sample program with the project”, and select the “Getting Started” program.
- Delete all the code in “getting_started.c” and replace it with the incomplete C program shown in Fig. 1.
- Fill in the code with a for-loop which iterates through the array to find the maximum.
- Compile and run the C program the same way you compile and run assembly programs.

```
int mail() {
        int a[5] = {1, 20, 3, 4, 5};
        int min_val;
        //TO DO – FILL THIS IN
        return min_val;
}
```

Figure 1: C code for computing min value

2.2 **Calling an assembly subroutine from C**

It is also possible to mix C and assembly. Perform the following steps to write a C program which calls an assembly subroutine.

- Create a new C project, as you did in Section 2.1.
- Add a file to your project called “subroutine.s”. (The filename can be anything, but it must have .s extension).
- Copy the code from Figure 2 into “subroutine.s”. This code computes the minimum of two numbers and returns the result. Notice that the subroutine does not save and restore the caller state. This is fine for the subroutines which do not change the state.
- Next, edit C program so that it contains the code in Figure 3. This code uses the assembly subroutine to compute the min of two numbers.
- Compile and run the program. Find the main section and the MIN_2 section, and put breakpoins to see the processor run those sections.
- Finally, rewrite your C program to find the max of a list using the MIN_2 subroutine.

```
        .text
        .global MIN_2
MIN_2:
        CMP R0, R1
        BXLE LR
        MOV RO, R1
        BXLR
        .end
```

Figure. 2: Assembly code with MIN_2 subroutine

```
extern int MIN_2(int x, int y);

int  main() {
        int a, b, c;
        a = 1;
        b = 2;
        c = MIN_2(a,b);
        return c;
}
```

Figure 3: C code which calls MIN_2 subroutine

3 Grading

The TA will ask you to demo the following deliverables during the demo (the corresponding portion of your grade for each element is indicated in brackets):

- Test program with PUSH and POP instructions (15%)
- Assembly code which computes the min of array using an assembly subroutine (15%)
- Factorial program with recursive subroutine (20%)
- C code which computes the min of an array using C (15%)
- C code which computes the min of an array using an assembly subroutine (15%).

A portion of the grade is reserved for answering questions about the code, which is awarded individually to group members. All members of your group should be able to answer any questions the TA has about any part of the deliverables, whether or not you wrote the particular part of the code the TA asks.

The remaining 20% of the grade for this lab will go towards a report. Write a short (3-4) page report that gives a brief description of each part, the approach taken and the challenges faced. Do not include the entire code in the body of the report.

Your final submission should be a single compressed folder that contains your report and all the code files (.c and .s).

This lab will run for two weeks, from October 7th to October 18th . You should demo your code during those dates, within your assigned lab period. The report for Lab 2 is due on Oct 27, 5 PM. Due to Thanksgiving holiday, groups scheduled for Mondays, will use for demo the first session of Lab 3 on October 21st (this will be done parallel with the execution of Lab 3).