Lab 2: Stacks, Subroutines, and C

ECSE 324 - Computer Organization

Winter 2020

Introduction

In this lab, you will learn how to use subroutines and the 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 which can be helpful for situations when there are not enough registers for a program to use 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 of the subroutine.

Review how the PUSH and POP instructions work. Note that pushing and popping can be implemented without using the PUSH and POP instructions by using other ARM instructions. Rewrite the following PUSH and POP instructions using only other instructions:

- `PUSH {R0}`
- `POP {R0 - R2}`

Write a test program in assembly to show that your rewritten versions correctly implement PUSH and POP. You should be able to show the TA the contents of main memory changing as registers are pushed onto the stack.

1.2 The subroutine calling convention

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

The call*er* must:

- Move arguments 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 call*ee* 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 LR` to return to the calling code

(The state can be saved and restored by pushing 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 max of an array into a program which uses a subroutine. The subroutine should return the max in R0.

1.3 Fibonacci calculation using recursive subroutine calls

A recursive subroutine is a subroutine which calls itself. You can calculate the nth Fibonacci number, F_n (where $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, ...), using a recursive subroutine as follows:

```
Fib(n):
    if n >= 2:
        return Fib(n-1) + Fib(n-2)
    if n < 2:
        return 1
```

For example, F_4 is computed as follows: $\text{Fib}(4) = \text{Fib}(3) + \text{Fib}(2) = (\text{Fib}(2) + \text{Fib}(1)) + (\text{Fib}(1) + \text{Fib}(0)) = ((\text{Fib}(1) + \text{Fib}(0)) + 1) + (1 + 1) = (1 + 1 + 1) + (1 + 1) = 5$

Write an assembly program which computes the nth Fibonacci number in this way. Your program should have a main section which calls the Fibonacci subroutine recursively for the above pseudocode.

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

Figure 1: C code for computing the max.

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 Pure C

We will first go through an example of programming in straight C.

- 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 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 Figure 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. Notice that the disassembly viewer shows how the compiler has translated C into assembly.

2.2 Calling an assembly subroutine from C

It is also possible to mix C and assembly. You will need to do this from Lab 3 onward. 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 does not matter, but it should have the .s extension.)
- Copy the code from Figure 2 into "subroutine.s". This code computes the maximum of two numbers and returns the result. Notice that the subroutine does not bother to save and restore the caller state. This is sometimes OK to do, in subroutines which do not change the state.
- Next, edit your C program so that it contains the code in Figure 3. This code uses the assembly subroutine to compute the max of two numbers.
- Compile and run the program. Find the main section and the MAX_2 section, and put breakpoints to see the processor run those sections.
- Finally, rewrite your C program to find the max of a list using the MAX_2 subroutine.

```
        .text
        .global MAX_2
MAX_2:
        CMP R0, R1
        BXGE LR
        MOV R0, R1
        BX LR
        .end
```

Figure 2: Assembly code with MAX_2 subroutine.

```
extern int MAX_2(int x, int y);

int main() {
        int a,b,c;
        a = 1;
        b = 2;
        c = MAX_2(a,b);
        return c;
}
```

Figure 3: C code which calls MAX_2 subroutine.

3 Grading

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

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

Full marks are awarded for a deliverable only if the program functions correctly and the TA's questions are answered satisfactorily.

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 about. Full marks are awarded for a deliverable only if the program functions correctly and the TA's questions are answered satisfactorily.

Finally, the remaining 20% of the grade for this Lab will go towards a report. Write up a short (3-4) page report that gives a brief description of each part completed, the approach taken, and the challenges faced, if any. Please don't include the entire code in the body of the report. Save the space for elaborating on possible improvements you made or could have made to the program.

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 February 3rd to February 14th. You should demo your code during those dates, **within your assigned lab period.** The report for Lab 2 is due by 11:59 pm, February 21st.