Laboratory 2: Stacks, Subroutines, and C

ECSE 324- Winter 2020

Lab Section 002 - Group 43
Dafne Culha (260785524)
Majd Antaki (260731626)

Culha Dafne (26078524)
Majd Antaki (260731626)

Introduction:

For this lab, we've learned how to use subroutines and the stack, program in C, and call code written in assembly from code written in C.

Part 1: Subroutines

The first part of the lab helped us better understand the stack structure in ARM and taught us to call subroutines by writing a code to practice the subroutine calling convention and implementing a recursive subroutine which calls itself.

1.1 - The Stack

The stack is a data structure which is helpful for situations when there are not enough registers for a program to use only registers to store data. It is also useful when calling subroutines to save the state of the code outside of the subroutine.

Stack is a last in first out data structure. To build it, we simply implement push and pop instructions. The pushing of R0 was implemented by using the STR R0, [R4, #-4] instruction and the popping of R0-R2 was implemented by using the instruction LDR R0, [R4, #4] from R0 to R2. This program was fairly simple to implement.

1.2 - The subroutine calling convention

In this part, we practice calling a subroutine. The code consists of a caller and callee. The caller moves arguments from R0 through R3 and calls the subroutine using the instruction BL FIND_MAX. FIND_MAX subroutine was implemented by using the same logic we followed in lab 1. The callee moves the return value into R0. Registers were restored using the instruction POP {R4-R12, LR}. Lastly, the instruction BX LR was used to return to the calling code.

1.3 - Fibonacci calculation using recursive subroutine calls

For this part of the lab, we try to implement a recursive subroutine. The algorithm we tried to implement was Fibonacci and we do this by following the pseudo code which was provided to us. There were 2 other subroutines called (nested) in the implementation of fibonacci subroutine.

The first one was the subroutine NON_RECURSIVE which the subroutine FIBONACCI called if n was less than 2. This subroutine simply returns 1 since FIB(0) and FIB(1) was already known to be 1. And the other one was the subroutine RECURSIVE which executed if n was not less than 2. The subroutine RECURSIVE calls back the subroutine FIBONACCI. It computes FIB(N-1) and FIB(N-2) and adds these up. R5 is used to return the value of FIB(N).

To implement this program,we followed a very simple algorithm. However, our implementation uses too many registers and this could be improved by using less registers in fibonacci which might be better use of memory. Because of the recursive calls, fib(3) for example, is calculated multiple times. For improvements, we could improve the time complexity of the program.

Part 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. For this part, we remined coding in C by implementing

2.1 - Pure C

For this part, we go through an example of programming in straight C. We were asked to write a code in pure C to find the max value in an array. We use the starter code provided to us and fill in the blanks to find the max value in an array. Our implementation initiates the int max_value to the first element instead of leaving it be the default value 0 so that if all elements in the array are negative values, the integer max_value equal to one of the elements in the array rather than 0. We use a for loop, it iterates through all values and updates the value of integer max_value if any of the other elements are greater than the first element of the array.

We compiled and ran the C program the same way we compiled and ran our previous assembly programs. We noticed that the disassembly viewer shows how the compiler has translated C into assembly.

The code implemented for this part was fairly simple, however, one challenge we faced was that we weren't able to implement a for loop at first. This was due to the fact that we were so used to using the Java syntax and we didn't remember to declare the variable i (which is incremented for each iteration) before starting the loop, which is what you're supposed to do when programming in C. We were able to figure this out and implement a for loop in the end by the help of our TAs.

2.2 - Calling an assembly subroutine from C

This part of the lab made us practice mixing C and assembly. We write a C program which calls an Assembly subroutine. We are simply given a subroutine MAX_2, written in Assembly which computes the maximum of 2 values by comparing the current max to the current value in the array and return the max of the two. We see that we are able to call this subroutine in our C program by including extern int MAX_2 (int x, int y); in our main class and then using the usual routine for calling functions in C.

This part of the lab was interesting. It was great to see how you are able to call code written in ARM from code written in C. It was fairly simple, since we were already given the code and explanations in the lab guide. Though simple, this will be very useful for our future labs.