# ECSE 324 G02 Lab 2 Report

Adrian Wang 260769387
Cheng Chen 260775674

## Part 1 The Implementation of Stack

The stack can be useful for situations when there are not enough registers for a program to use to store data. In this part, the stack is implemented by rewriting PUSH and POP using other ARM instructions.

For the PUSH instruction, we first set R0 to point to the address which stores the array (input) we want to push onto the stack and R1 holds the first number in the array. R2 holds the number of elements in the array which we will be using as a counter to traverse the array. In loop 1, we store the first number(i.e. R1) into the address of stack pointer and decrement the stack pointer address by four. After that, we move to the next number in the array by incrementing the pointer(R0) by four and decrement the counter(R2). Once the counter reaches zero, we exit the loop.

The POP is implemented in loop 2. The POP implementation is similar to the PUSH implementation. In PUSH, we decrement the stack pointer after we push a value onto the stack, while we increment the stack pointer after we pop a value from the stack in POP. Once the counter in loop 2 reaches zero, we end the program.

When we were implementing this program, we were confused about whether we should decrement or increment the stack pointer for PUSH. After some research, we were able to solve this issue. One of the improvements can be made is that the program can automatically find out the number of elements in the array instead of getting the information through user input.

## Part 2 The Simple Subroutine

In this part, the program is written using subroutines and returns the maximum number in a given array.

At first, R0 points to the array and R1 holds the number of elements in the array. By following callee-save convention, R0,R1 and link register are pushed onto the stack before calling the subroutine and R0 to R3 are also pushed onto the stack in the subroutine. In the subroutine, we load the number of elements, first number in the array into R1 and R2 respectively. Also, the first number in the array is moved into R0 as a temporary maximum number. After that, we move to the next number in the array and decrement the counter. If the number is greater than the number in R0, we

update the maximum number in R0. Once the counter reaches zero, we pop everything from the stack and exit the program. The max number is returned by R0.

The challenge faced in the part is that we did not know where the values such as the number of elements are stored on the stack since we pushed a lot of values onto the stack. However, we were able to figure out by drawing the stack and counting the positions of the values we want.

## Part 3 The Fibonacci Program Using Subroutine

In this part, the program computes the nth Fibonacci number recursively when given a number as input. This program is written by translating the C code given in the description into assembly language.

First, we load the number(input) into R1 and push R1 and LR onto the stack before calling the subroutine because of the callee-save convention. In the subroutine, we first check if the number is less than two. If yes, we move one into R0 and return. If not, we decrement R1 by one and call the Fibonacci subroutine again(i.e we are calculating F(n - 1)). After we finish calculating F(n-1) and storing the result into R2, we subtract one from the input number again so that the number becomes n-2 and call the Fibonacci subroutine. The result for F(n-2) is then stored into R3. After that, we add R2 and R3 together and return.

One of the challenges faced in this part is that we were trying to figure out how to add F(n-1) and F(n -2) together in assembly language. We were able to solve it by storing two results into two different registers and adding the values in the registers after we finish all the recursions. The program can be improved by implementing an iterative version as iteration can be more efficient sometimes depends on the input value.

## Part 4 The C program and Subroutine Calling

The Intel FPGA Monitor Program software not only provides the development environment in ARM assembly but also C language that could run on the development board. A pure C program is implemented as instructed in the manual. After the program is compiled and run, it could be seen that the C is translated into assembly in a very complex way: besides the direct translation of the code, there are many more lines of codes that do background works such as allocating memories.

Then another C program is implemented to call an assembly subroutine which looks for the maximum number within an array. The subroutine works the same way as described in Part 2. The C program could still be improved by setting the temporary

maximum number as the first number in the array instead of zero. In this way, there will be fewer steps and less computing power will be needed.