Group 30
Zhuonan Wang 260714137
Suhui Shen 260705934

ECSE 324 Lab2 Report

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

1.1 The Stack

This part of the lab simply required the use of stacks. There is no big challenge in this problem and the only thing we need to understand is how stack actually works. The theory of the stack is ‘First in, Last out’, meaning that the PUSH function would add a new value to the stack while the POP function would take out the latest value that was pushed in the stack. In this program, we need to implement PUSH and POP instructions by using other ARM instructions.

To realize the instruction PUSH, we first move the stack pointer SP using the instruction SUB SP, SP, #4. Then the instruction STR R0, [SP] stores the number in R0 to SP. Now the SP points to the new top element in the stack.

In POP part, we add #4 to the SP with instruction ADD SP, SP, #4. Now the stack pointer points to the second top element in the stack and the previous top memory slot is out of the stack. We also store the value that was popped out, which is now at memory location SP - #4.

1.2 The subroutine calling convention

The aim in this part is to learn how to call a subroutine in ARM assembly. A caller and a callee are used in this program. With the convention specified, the caller would move arguments into R0 through R3 and call the subroutine using BL, while the callee would move the return value into R0 and ensure the state of the processor was restored. BX(branch and exchange instruction set) and LR (Link Register) was used to return to the calling code. In general, this program is used to find the max of an array list, which consists of converting the code written in lab 1. However, in this lab we will make use of the subroutine and eventually return the max value in R0.

We started with pushing the initial information including the memory location of ARRAY and the total number N to the stack. Then the caller would call the callee subroutine using BL after pushing the corresponding LR to the stack. In subroutine, we first push R0 - R3 to save all the registers before overwriting them. After that we load N and the memory location of ARRAY that was pushed before to R1 and R2, and load the first number of ARRAY to R0. So far, we finished all the preparations before entering the loop to find the maximum value.

The following MAX_LOOP was adapted from what we had in lab 1 to find the max of the number list and put it in R0. After looping through all the entries of the array (when the counter goes to zero) we branched to DONE, where the maximum value is saved in the stack, and we popped out R0-R3 to restore the registers before getting out of the subroutine. Finally, the subroutine was finished and we used BX LR to go back to the caller and continue from the instruction after BL subroutine. Here we loaded the max value to R0 and store it in the memory of MAX, and then we initialize everything by restoring the link register and resetting the stack pointer.

The challenge of this problem is to understand the working theory of link register. Every time the BL instruction is used, the LR would be updated and contains the address of the next instruction. This feature would help us to mark out the corresponding instruction so that at the end of each subroutine we could know where to go back.

1.3 Fibonacci calculation using recursive subroutine calls

The goal of this part of the lab was to try and implement the pseudo code provided, which describes a method to calculate the number in Fibonacci. A subroutine Fib was written. The LR is pushed to stack at first when entering the subroutine. We compare N with number 2, if N is smaller than 2, simply move number 1 to R0 to return 1, as described in the pseudo code. Otherwise, save current N to the stack and subtract 1 from N to get N-1 and branch back to Fib until N-1 becomes 1. By doing this we save all the values from N to 2 in the stack. It then keeps calculating the value of fib(N-1) and fib(N-2) and sums them together to get fib(N). The difficulty in this part is to store and restore the LR while recursive calls are made. Also, we need to only use R0 in passing the parameter and to get the returned value.

2.1 Pure C

In this part, we need to write a code in pure C to find the max value in an array. This was relatively straightforward. The approach taken was to iterate through every content in the array and at the same time comparing the current Max value (set to 0 in the beginning) to the current value in the array, if the current value is larger we would update the current max; After going through every value in the array, the current max would be the max value in the array. A loop was used for convenience.

2.2 Calling as assembly subroutine from C

This part of the lab is to see how we are able to call code written in ARM from code written in C. We are given a sample code in ARM which found the max of 2 numbers. We need to implement C code which called the subroutine and found the max value in an array. We first initiate an array that contains several integers. Then we call the subroutine MAX_2 that finds the max of two numbers for the list of integers in a loop. The returned value max_val is the max of the array.

3 Possible Improvement

In the last part, we used the easiest and most straightforward algorithm to find the maximum number of the list. It is possible that we use a better algorithm that could reduce the running time and optimize the program.