Omar Yamak 260662616
Thomas Hillyer 260680811

ECSE324 Report- Lab 2
Group#47

1 Subroutines

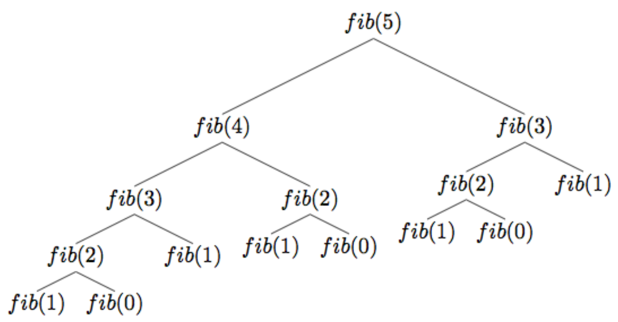
1.1 The Stack

This part of the lab simply required the use of stacks, it was expected to PUSH and POP values off the stack without using those instructions explicitly. The **STR R0, [SP, #-4]** instruction was used to push values on to the stack and the **LDR R0, [SP], #4** was used to pop values of the stack. This part of the lab was fairly simple and had no challenges. An improvement on this code would be to simply use the PUSH and POP instructions for simplicity.

1.2 The subroutine calling convention

Here, the aim was to learn how to call a subroutine in ARM assembly, it consisted of converting code written in lab 1 which found the max of an array into a program which uses a subroutine and returns the value of max in R0. The code simply consisted of a caller and callee, the caller would move arguments from R0 through R3 and call the subroutine using BL. The callee would move the return value into R0, ensure the state of the processor was restored, and finally, BX(branch and exchange instruction set) in addition to LR (Link Register) were used to return to the calling code. The logic of the Max subroutine was the same of that in lab 1. The only challenge faced here was getting used to calling subroutines in ARM using the BL instruction, however, this was also fairly simple. An improvement would have been to have an instruction that could restore the state of the processor to what it was before the subroutine was called.

1.3 Fibonacci calculation using recursive subroutine calls



The approach used for this part of the lab was to try and implement the pseudo code provided. A subroutine Fib was written, which is a recursive algorithm that calls itself on Fib(n-2) and fib(n-1). The algorithm first computes the left then the right side of the tree. For example, in the case where we want to find FIB(5) we would find FIB(4) and FIB(3) which correspond to FIB(n-1) and FIB(n-2), this would keep happening until the algorithm reaches FIB(1) and FIB(0) which are values already known. The algorithm will then move back up the tree adding all values pushed onto the stack previously. The challenging part faced here was to figure out how to use the link register throughout the code with the aid of the stack in order to move down and up the tree. An improvement to this algorithm would be to have a tree object within ARM, allowing for a much simpler recursive algorithm.

2 C Programming

2.1 Pure C

This part of the lab simply required us to write a code in pure C to find the max value in an array. This was very straight forward as it was just expected of us to fill in a short part of the code. The approach taken was to iterate through every content in the array whilst 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 current max would be the max value in the array. A loop was used for convenience. This part of the lab was fairly simple, it just took a couple of minutes to get used to writing algorithms in C as we were only used to writing algorithms in Java. Noticeably, having a subroutine implemented in ARM that found the max value in an array would have made this part of the lab redundant and even simpler.

2.2 Calling as assembly subroutine from C

This part of the lab was intriguing, it was interesting to see how you are able to call code written in ARM from code written in C, given some code in ARM which found the max of 2 numbers, we were to implement some code in C which called this code and found the max value in an array. This was fairly simple to do, a loop was used to iterate through contents of the array whilst calling the MAX_2 subroutine written in ARM which would compare the current max to the current value in the array and return the max of the two. No challenged were faced here as we were given a thorough explanation of how to implement such algorithm. No improvement could be made to this code as it seemed very straight forward and compact, it will be interesting to see in future labs a more in-depth example of how calling code in ARM from C works.