ECSE – 324 Computer Organization

Lab 2 Report

Group 34

Aaron Heehyun Jang – 260745290

Kun Zheng – 260579245

1.1 The stacks

We first reserve a chunk of the memory starting from address “MYSTACK” to function as stacks, and then we have R4 points to the bottom of “MYSTACK”, which functions exactly as “SP” register. Each time we store data of R0 into “MYSTACK”, we use post-index to have the R4 point to the top of “MYSTACK” as the data is stored from bottom to top.

When we are ready to pop our data. We use pre-index to have the R4 point to the data we want to pop and load them into the register in a reverse order from which we stored them, that is, the first register, R0, stores the last data we pushed while R2 stores the first data we pushed.

The entire implementation of PUSH and POP is very straight forward, we did not run into difficulties.

1.2 The subroutine calling convention

1.3 Fibonacci Calculation using recursive subroutine calls

By inspecting the recursion example given to us, the main challenge to implement a BL to the same subroutine within the subroutine.

This is hard because when we call the subroutine using “BL”, we at the same time create an “ticket” (the address store in the LR register). We use this “ticket” to go back to where we were before entering the subroutine by calling BX. What makes this process very confusing is that we are entering the same subroutine within the subroutine. We need to keep track on how many “tickets” we created within the subroutine. Instead of using Pop and Push to keep track of the “tickets”, we use load and store to manually place them in the “ticketbooth”, which is the beginning address of a chunk of memory that functions same as stacks. First, in the starting phase, we load the number N, which stands for the Nth Fib, into R0. This register will help us keep track on how many times we have entered subroutines. We also need an register, R3, to store our current Fib number. Then we need another register, R7, as a pointer pointing to the memory address “ticketbooth” that is going to store all our tickets out of the recursion. Last but not least, we need two register R1 and R2 to store the numbers we need to calculate the next Fib number with.

With all these set up, we are ready to enter the subroutine.

The first time we enter the subroutine, we store the first ticket at the bottom of the “ticketbooth” using post-index mode, we proceed with calculating the next Fib number and decrement R0. If we have not reach the Nth Fib number that we desire, we enter Fib again. Eventually with “CMP R0 #1” and “BLE LETSGOHOME” condition, we enter “let’s go home”, in which is the instruction of returning to Fib by using “BX LR”. Now we load the next ticket into LR and use it to return to Fib. This process will repeat until we eventually load the first ticket into LR and get us out of the subroutine.

It is worth mentioning that, through observation, all the tickets except the first ticket stores the exact same content (which points to the exact same instruction in the memory). This initially got us confused because when we look at our “ticketbooth” in the memory. We saw multiple words storing the same “0x00004C” except the bottom of “ticketbooth” storing something else. We then realize this makes sense because though we created multiple tickets, the tickets themselves are actually the exact same instruction, that is, take us to this exact same step within the Fib. It is not until we reach the first ticket that take us elsewhere (outside the subroutine), hence being a different instruction stored in the memory. This means the size of “ticketbooth” can be independent of whatever Fib number we want to calculate as we can reuse the “ticket” within the subroutine, which frees a lot of memory resource and avoid loading the same “ticket” into LR over and over again. We can use R0 (which stores the number of N) to keep track of how many times we entered subroutine within subroutine, thus knowing how many times we should reuse that “ticket”. When we arrive at the first Fib that we entered (We know this when R0 decrement to 1), we load the first ticket and exit the subroutine.