Eldor Bekpulatov (eb654)
Kaushik Ravikumar (kr437)

Lab 1 Report

Design

In order to display the dashes and dots, we decided to turn the LED light on and use either a short pause to indicate a dot or a long pause to indicate a dash. We created helper functions to complete a short pause and a long pause. In order to pause, we ran a loop that essentially decremented a value and only exited the loop once the value was equal to 0. The original value for a long pause was 480,000 and 180,000 for a short pause.

In order to translate a digit into morse code we came up with the following algorithm. We noticed that there was a pattern for the digits, in which a dot could be represented as a 1 and a dash as a 0. In this case the digit 5 can be represented in binary as 11111. In order to get any other digit, all we had to do was perform a logical shift either left or right depending on by how much greater or less than a digit is from 5. If a digit is greater than 5 we will shift to the right and if it is less than 5, we will shift to the left. For example, consider the digit 7. Since it is greater than 5 by 2, we will shift to the right and by two bits. Using this algorithm, we obtain a binary representation for each digit.

Then we simply go through every bit and perform the and operation first with 16, in order to isolate the most significant bit, then with 8 in order to isolate the next most significant bit, etc. Depending on whether it is 1 or 0, we simply call the appropriate function on whether to display a dot or a dash.

In order to do part 2, we simply ensured that we were following the convention where the first four registers are caller saved and that the remaining registers were callee saved.

In order to do part 3, we had to simply compare the value of n we are provided with 1. If it's less than 1 (0) then this is the base case of 0 and if it's equal to 1 then this is also a base case of 1. We then had a function that took care of the recursive part if the value of n was greater than 1. In this function we used a stack to recursively backtrack and hold onto the old values for R1, R2, and R3. Then we simply kept adding the result of the fib function applied to R0-1 with that of the fib function applied to R0-2.

To do the extra credit part, we utilized an algorithm requiring 5 registers and a stack to keep track of all the individual digits in a multi-digit number. R5 kept track of how many digits there are, so it's initialized at 0. R0 is where the multi-digit number is and R4 is what we're dividing by, which is 10 so we can get each digit. Then we store the quotient into R2. We multiply R2 by 10 and then subtract this from R0 for the remainder, which is stored in R3. This value of R3 is pushed onto the stack and then the value in R2 moves into R0 for the next iteration of this loop. This loop continues until we have an R0 of 0. Essentially using this algorithm, we are able to iteratively call our function MorseDigit for each digit in a multi-digit number.

Testing

In order to test that our Morse Code program was working we used in-built debugging tools. We ran the program checking the values in registers, periodically using breakpoints. We also wrote out code in the main function that would simply move in 0 into R1 and then invoke our function MorseCode, then move 1 into R1 and invoke the function, until we got to 9. After running this code on the board, we saw that all the digits were properly displayed in Morse Code.

In order to test that our Fibonacci program was working, we moved 6 and other arbitrary numbers into R0. With this in mind, we calculated which fibonacci number should be displayed. We then ran the program and saw that the morse code displayed on the board matched that of the fibonacci number that should've been displayed. We also similarly used debugging tools and breakpoints to intensively ensure that the registers contained the right values.

Work Distribution

Eldor came up with the overall plan on how to utilize the pattern amongst the digits in morse code and wrote a major chunk of this code. We both then practiced peer programming where we had a driver and a navigator. This worked better than if we were coding on two separate laptops concurrently. We both reasoned together, which helper functions were important to create and how to go about creating them. We met up twice and communicated over Facebook Messenger when necessary. We worked together on reasoning about how to convert the C code of the fibonacci function into assembly code and how many registers we needed to accomplish this. Kaushik wrote the lab report.