**Joshua Pollock**

**EE 310 – Lab 6 Report**

**NAU, 26 April 2020**

**Problem Description**

In this lab, we have been asked to add the beq instruction to the supplemental files. This is detailed in section 6.5 of Zybooks but a simplified run-down/guide was provided to us in the lab report. In this lab the use of offsets is allowed but loops and subroutines are still not (we can bypass this using the beq instruction and a label). Once adding the beq instruction to the instruction set, we were tasked with creating a MIPS assembly file that completed the operations seen in Figure 1. Once we verify our code is fully functional, we were tasked with editing lines of the test bench to test the code.

|  |
| --- |
|  |

Figure 1. Expected behavior of the circuit

**Solution Plan**

In order to solve the problem explained above, I used the supplemental code provided to set up the project. Once the project was set up, following the instructions within the lab I modified the mipzy\_control, adder\_32, and MIPSzy files to all include the beq instruction. This was straight forward and just required following along with the pictures provided.

Next, we were tasked with creating a program that would perform multiplication without using the mul instruction. This can be done with a loop adding the multiplicand to a running sum and then subtracting from the multiplier. This is looped until the multiplier reaches zero, thus completing the multiplication operation. A simple check should also be implemented to check if the multiplicand is zero. This will keep the code from running completely and will save run time in this special case. The multiplier does not need to be checked beforehand because it will be checked when the loop executes.

|  |
| --- |
|  |

Figure 2. State diagram for the proposed solution

**Implementation and Test Plan**

I have implemented the solution plan explained above, by first adding the beq instruction to the MIPSzy instruction set. This was easy and straightforward as we were given picture guides and told how to do it like last lab. Next, I simply created a basic MIPS file that did the operations requested in Figure 1. Register t4 acted as my running sum variable. Register t1 was set to 5000 for loading and saving registers. Next, I loaded the multiplier into t3 and the multiplicand into t2. A check using the beq instruction was used next to test if the multiplicand was 0. If it was, then it would immediately end. The code would next add the multiplicand to the running sum and then subtract 1 from the multiplier. This loop would repeat if the multiplier did not equal 0. Lastly it would save the running sum at memory address 5008.

|  |
| --- |
| **Lab8.txt**  addi $t1, $zero, 5000 # Pointer for the input  addi $t4, $zero, 0 # Set output as zero, for 0 case  lw $t2, 0($t1) # Fetch multiplicand  lw $t3, 4($t1) # Fetch the multiplier  beq $t2,$zero,End # Check if multiplicand is 0 to avoid needless runtime  mulLoop:  beq $t3,$zero, End # Check if multiplier is 0, if so end loop  add $t4, $t4, $t2 # Add multiplicand to running sum  addi $t3,$t3,-1 # Subtract 1 from multiplier  beq $zero,$zero, mulLoop # Repeat loop  End:  sw $t4,8($t1) # Save t4 into the output address 5008  **Lab8Binary.txt**  MIPSzy\_0.IM.memory[0] = 'b00100000000010010001001110001000;  MIPSzy\_0.IM.memory[1] = 'b00100000000011000000000000000000;  MIPSzy\_0.IM.memory[2] = 'b10001101001010100000000000000000;  MIPSzy\_0.IM.memory[3] = 'b10001101001010110000000000000100;  MIPSzy\_0.IM.memory[4] = 'b00010000000010100000000000000101;  MIPSzy\_0.IM.memory[5] = 'b00010000000010110000000000000100;  MIPSzy\_0.IM.memory[6] = 'b00000001100010100110000000100000;  MIPSzy\_0.IM.memory[7] = 'b00100001011010111111111111111111;  MIPSzy\_0.IM.memory[8] = 'b00010000000000001111111111111101;  MIPSzy\_0.IM.memory[9] = 'b10101101001011000000000000001000;  The other edited code files will be attached to the BBLearn submission. |

Figure 3. Verilog code for the proposed solution

|  |
| --- |
| First run with the default numbers set    Second run with zero case. X=65 Y=0    Last run with random numbers set. X=23 Y=82 |

Figure 4. Lab pictures of the running solution