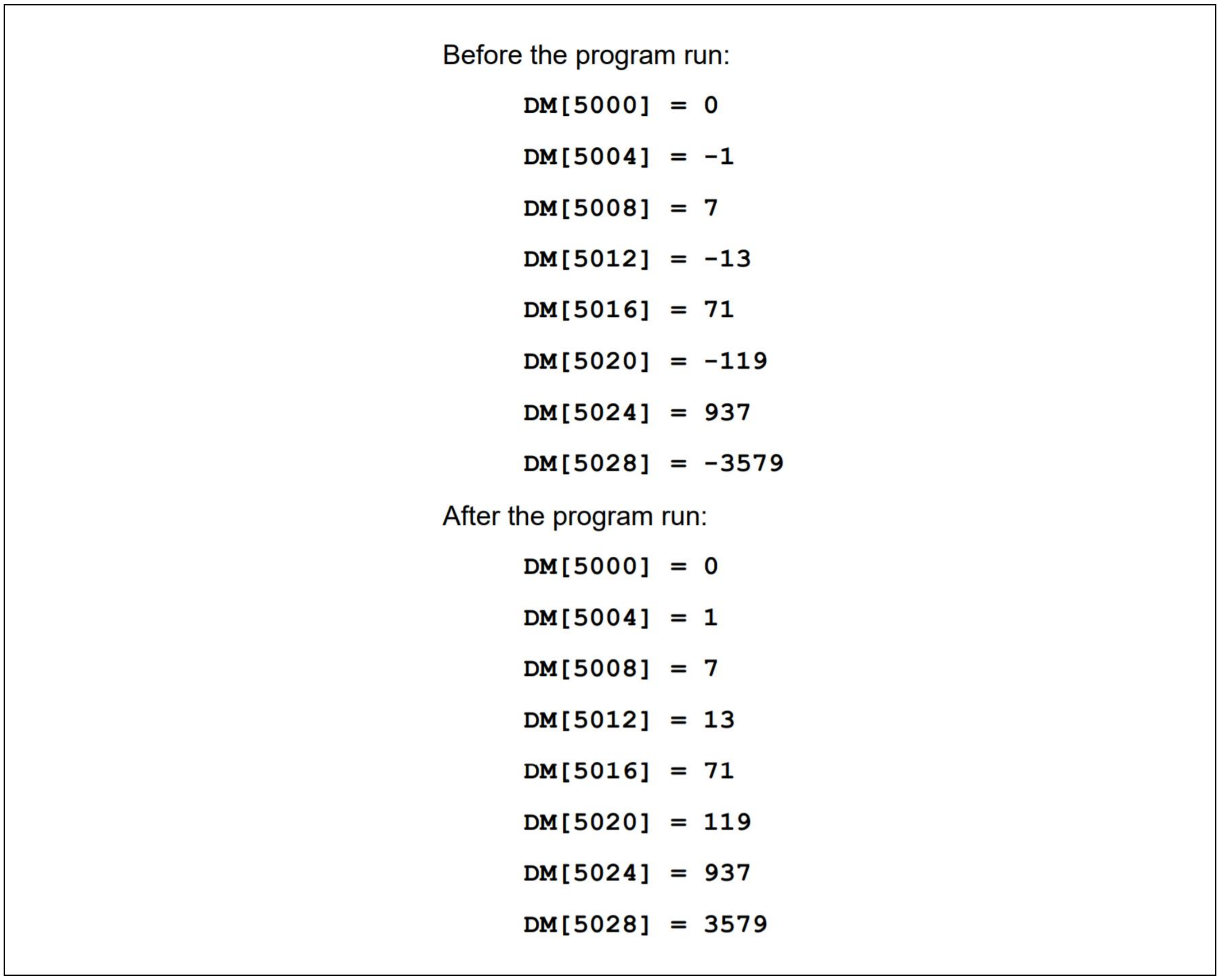
Joshua Pollock

EE 310 – Lab 9 Report

NAU, 3 May 2020

Problem Description

In this lab, we have been asked to add the SLT instruction to the supplemental files. This is detailed in section 6 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 SLT 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.



Before the program run:

```
DM[5000] = 0
DM[5004] = -1
DM[5008] = 7
DM[5012] = -13
DM[5016] = 71
DM[5020] = -119
DM[5024] = 937
DM[5028] = -3579
```

After the program run:

```
DM[5000] = 0
DM[5004] = 1
DM[5008] = 7
DM[5012] = 13
DM[5016] = 71
DM[5020] = 119
DM[5024] = 937
DM[5028] = 3579
```

Figure 1. Expected behavior of the circuit

Solution Plan

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 SLT instruction. This was straight forward and just required following along with the picture/guide provided.

Next, we were tasked with creating a program that would perform take the absolute value of data in memory from 5000 to 5028. This can be done with a loop that first tests to see if we have reached the end memory address. If not, we fetch the data at the memory address and use SLT to compare it to zero. If the data is less than 0, this will be set to 1. If the register is set to 1 the program will subtract the number from zero (0-$t2 = -($t2)). This will inverse the number. If the register is not set, this step can be ignored. Next iterate the memory pointer and loop.

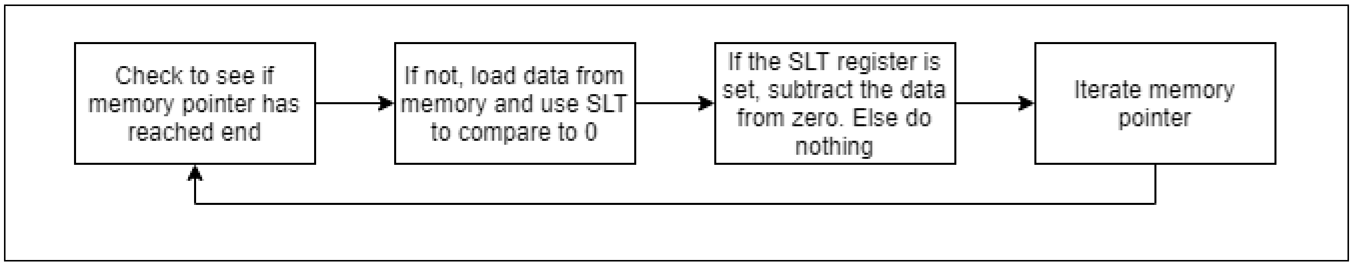

Check to see if memory pointer has reached end
If not, load data from memory and use SLT to compare to 0
If the SLT register is set, subtract the data from zero. Else do nothing
Iterate memory pointer

Figure 2. State diagram for the proposed solution

Implementation and Test Plan

I have implemented the solution plan explained above, by first adding the SLT instruction to the MIPSzy instruction set. This was easy and straightforward as we were given picture guides and told how to do it like the last 2 labs. Next, I simply created a basic MIPS file that did the operations requested in Figure 1. This program simply followed Figure 2. First the program would store the memory address 5000 and 5032 into $t0 and $t1. These would act as the beginning and end. Next, we would test to see if $t0 was equal to $t1, if so, the program would end. Else the program iterated the memory pointer. This was to prevent the program from not iterating if the SLT operation did not set a 1. Next the program loaded the data at the current memory address in $t0 and stored it at $t2. It used the SLT operation to compare this to 0 and store the result in $t3. If this set a 1 in $t3, the program would subtract $t2 from 0, taking the inverse of $t2. Next the program stored the value of $t2 into -4($t0).

Lab9.txt

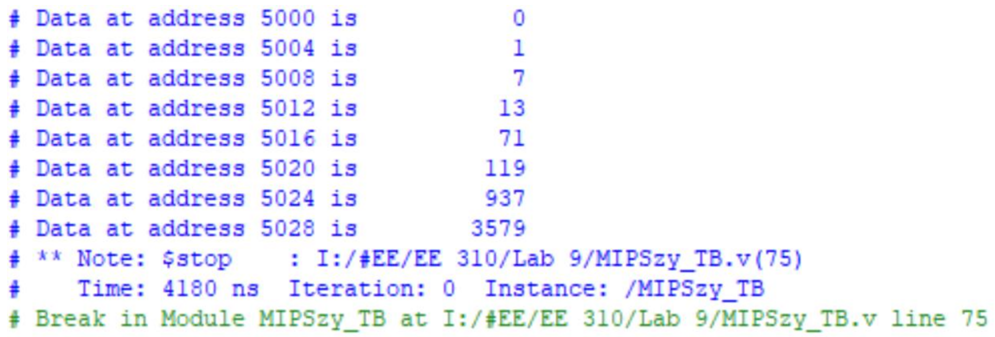
```
addi $t0,$zero,5000
addi $t1,$t0,32

loop:
	beq $t0,$t1,end
	lw  $t2,0($t0)
	addi $t0,$t0,4
	slt $t3,$t2,$zero
	beq $t3,$zero,loop
	sub $t2,$zero,$t2
	sw	$t2,-4($t0)
	beq $zero,$zero,loop
end:
```

Lab9Binary.txt

```
MIPSzy_0.IM.memory[0] = 'b00100000000010000001001110001000;
MIPSzy_0.IM.memory[1] = 'b00100001000010010000000000100000;
MIPSzy_0.IM.memory[2] = 'b00010001001010000000000000001000;
MIPSzy_0.IM.memory[3] = 'b10001101000010100000000000000000;
MIPSzy_0.IM.memory[4] = 'b00100001000010000000000000000100;
MIPSzy_0.IM.memory[5] = 'b00000001010000000101100000100100;
MIPSzy_0.IM.memory[6] = 'b00010000000010111111111111111100;
MIPSzy_0.IM.memory[7] = 'b00000000000010100101000000100010;
MIPSzy_0.IM.memory[8] = 'b10101101000010101111111111111100;
MIPSzy_0.IM.memory[9] = 'b00010000000000001111111111111001;
```

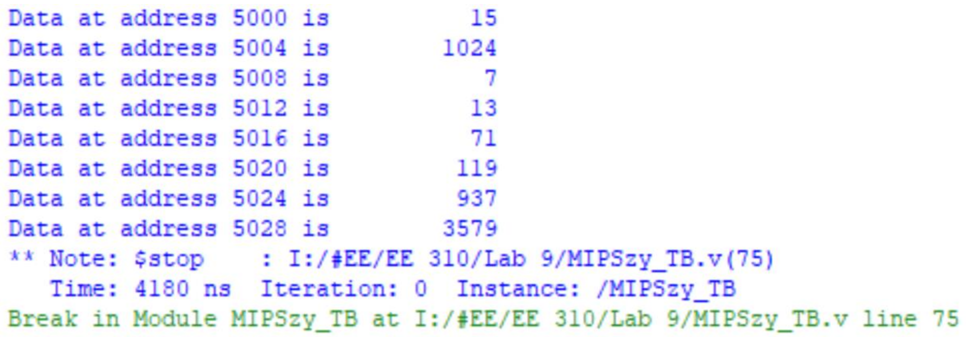
The other edited code files will be attached to the BBLearn submission.

Figure 3. Verilog code for the proposed solution



```
# Data at address 5000 is           0
# Data at address 5004 is           1
# Data at address 5008 is           7
# Data at address 5012 is          13
# Data at address 5016 is          71
# Data at address 5020 is         119
# Data at address 5024 is         937
# Data at address 5028 is        3579
# ** Note: $stop    : I:/#EE/EE 310/Lab 9/MIPSzy_TB.v(75)
#    Time: 4180 ns  Iteration: 0  Instance: /MIPSzy_TB
# Break in Module MIPSzy_TB at I:/#EE/EE 310/Lab 9/MIPSzy_TB.v line 75
```

First run with the default numbers set



```
Data at address 5000 is          15
Data at address 5004 is        1024
Data at address 5008 is           7
Data at address 5012 is          13
Data at address 5016 is          71
Data at address 5020 is         119
Data at address 5024 is         937
Data at address 5028 is        3579
** Note: $stop    : I:/#EE/EE 310/Lab 9/MIPSzy_TB.v(75)
   Time: 4180 ns  Iteration: 0  Instance: /MIPSzy_TB
Break in Module MIPSzy_TB at I:/#EE/EE 310/Lab 9/MIPSzy_TB.v line 75
```

Second run with random numbers

```
test_numbers[0] = -15     ;  // Number to be stored in DM[5000]
test_numbers[1] = -1024    ;  // Number to be stored in DM[5004]
test_numbers[2] = 7      ;  // Number to be stored in DM[5008]
test_numbers[3] = 13   ;  // Number to be stored in DM[5012]
test_numbers[4] = -71    ;  // Number to be stored in DM[5016]
test_numbers[5] = 119  ;  // Number to be stored in DM[5020]
test_numbers[6] = -937   ;  // Number to be stored in DM[5024]
test_numbers[7] = -3579 ;  // Number to be stored in DM[5028]
```

Figure 4. Lab pictures of the running solution