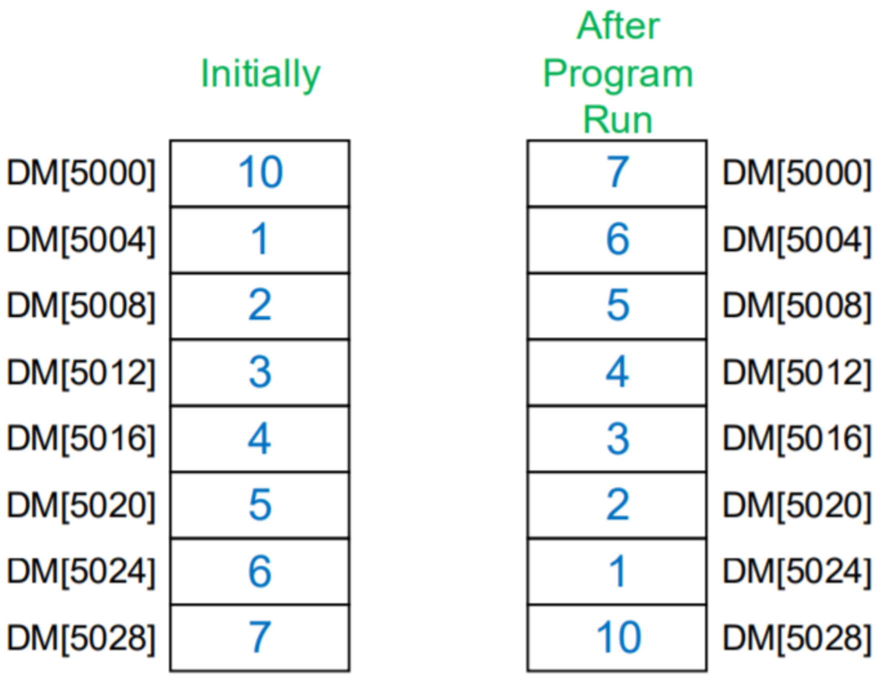
Joshua Pollock

EE 310 – Lab 6 Report

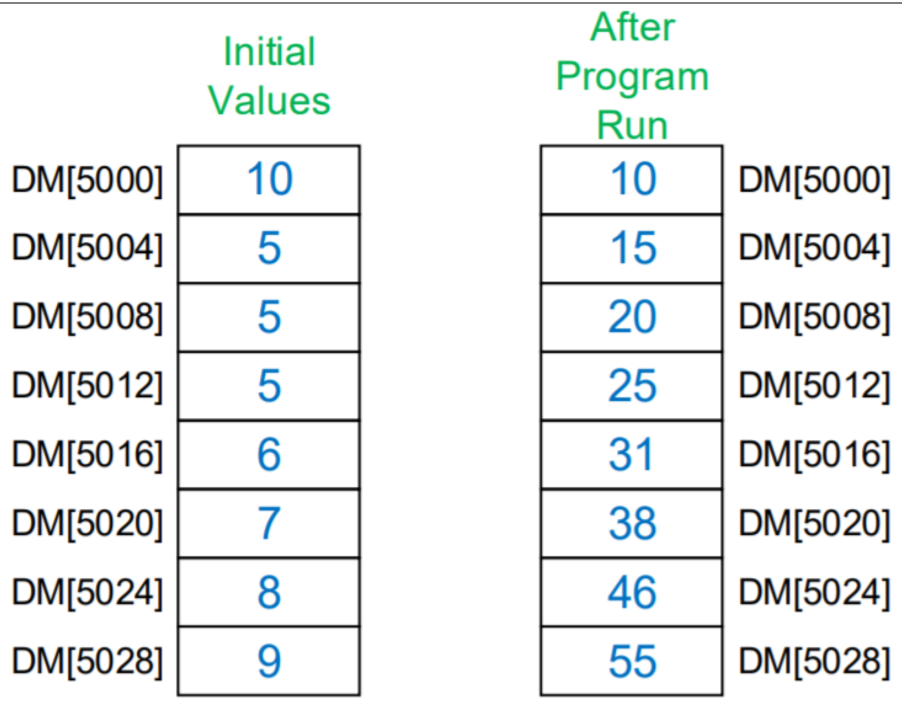
NAU, 12 April 2020

Problem Description

In this lab, we have been asked to create two programs for manipulating data using MIPS. The first program will read data stored in addresses 5000-5028 and write back to the same addresses in reverse order. The second program will read data stored in addresses 5000-5028, and write back to the same addresses the sum of data up to the given address. We can only use the lw, sw, addi, add instructions. We cannot implement any branches, jumps, comparisons. We are provided with a supplement zip file that contains the testbench for this lab. We will need to convert our code into bytecode using the tools on ZyBooks.



Part 1



	Initial Values	After Program Run	
DM[5000]	10	10	DM[5000]
DM[5004]	5	15	DM[5004]
DM[5008]	5	20	DM[5008]
DM[5012]	5	25	DM[5012]
DM[5016]	6	31	DM[5016]
DM[5020]	7	38	DM[5020]
DM[5024]	8	46	DM[5024]
DM[5028]	9	55	DM[5028]

Part 2

Figure 1. Expected behavior of the circuit

Solution Plan

In order to solve the problem explained above, I used the code from provided on BBLearn to create the skeleton testbench I needed to execute the bytecode. From here I needed to implement and test my solution only using the 4 instructions that were allowed: sw, add, addi, and lw. No jumps or offsets were usable in this lab, adding a significant length to my code. Not being able to use loops, there was no way to easily complete this code and I needed to have a large amount of redundant code. During this lab I would utilize the tools in ZyBooks to double check my code quickly when needed. It was also the tool needed to translate my code into bytecode. For the first portion of this lab, I utilized the stack to pt numbers on and pop them off in reverse order. The stack proved to be very useful in this case as it is LIFO (Last in first out). This meant I wouldn't have to manipulate addresses at all and could just read the stack in its normal order. The second portion of this lab proved to be even easier than the first. A register was used to keep a running sum. As we iterated up the memory addresses, the value would be read, added to the running sum, and then the running sum would be put back into memory.

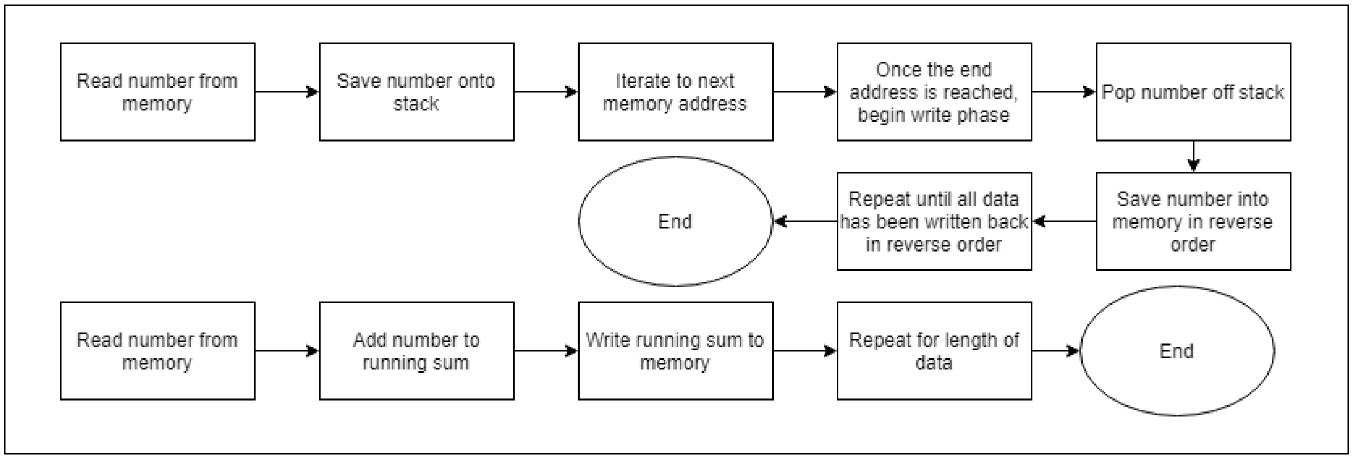

Read number from memory
Save number onto stack
Iterate to next memory address
Once the end address is reached, begin write phase
Pop number off stack
Save number into memory in reverse order
Repeat until all data has been written back in reverse order
End
Read number from memory
Add number to running sum
Write running sum to memory
Repeat for length of data
End

Figure 2. State diagram for the proposed solution

Implementation and Test Plan

I have implemented the solution plan explained above, by first writing and testing the code using the Zybooks' tools. Next, I used the given MIPSzy code for Quartus and set everything up within the testbench. From here I needed to convert the MIPS code into bytecode. Using the Zybooks' tool, I simply ran my code through and copied the bytecode output. After cleaning up the bytecode, I was able to tediously paste it into the testbench. After running the testbench simulation, I saw that my output was correct and showed no errors. I was very happy with my result even though it was pretty tedious working with only a small fraction of the full MIPS instruction set. I wish in future labs to see if there is a way to utilize the entire instruction set.

Lab6_1.txt

```
addi $t1, $zero, 5000           # Set begin to 5000

lw $t3, 0($t1)
sw $t3,0($sp)
```

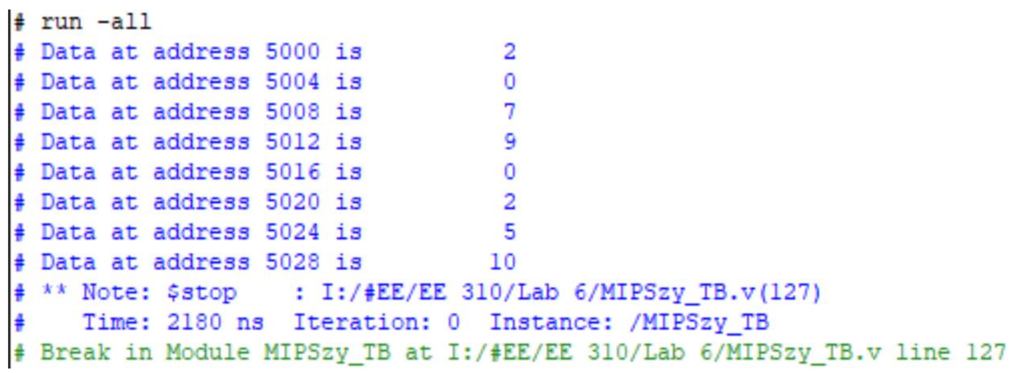
```
addi $t1, $t1, 4
addi $sp,$sp,-4

lw $t3, 0($t1)
sw $t3,0($sp)
addi $t1, $t1, 4
addi $sp,$sp,-4

lw $t3, 0($t1)
sw $t3,0($sp)
addi $t1, $t1, 4
addi $sp,$sp,-4

lw $t3, 0($t1)
sw $t3,0($sp)
addi $t1, $t1, 4
addi $sp,$sp,-4

lw $t3, 0($t1)
sw $t3,0($sp)
addi $t1, $t1, 4
addi $sp,$sp,-4

lw $t3, 0($t1)
sw $t3,0($sp)
addi $t1, $t1, 4
addi $sp,$sp,-4

lw $t3, 0($t1)
sw $t3,0($sp)
addi $t1, $t1, 4

lw $t3, 0($t1)
addi $t1,$zero,5000

sw $t3,0($t1)
addi $t1, $t1, 4

lw $t3,0($sp)
sw $t3,0($t1)
addi $t1, $t1, 4
addi $sp,$sp,4

lw $t3,0($sp)
sw $t3,0($t1)
addi $t1, $t1, 4
addi $sp,$sp,4

lw $t3,0($sp)
sw $t3,0($t1)
addi $t1, $t1, 4
addi $sp,$sp,4

lw $t3,0($sp)
sw $t3,0($t1)
addi $t1, $t1, 4
addi $sp,$sp,4

lw $t3,0($sp)
sw $t3,0($t1)
addi $t1, $t1, 4
addi $sp,$sp,4

lw $t3,0($sp)
sw $t3,0($t1)
addi $t1, $t1, 4
addi $sp,$sp,4

lw $t3,0($sp)
sw $t3,0($t1)
addi $t1, $t1, 4
addi $sp,$sp,4
```

Lab6_2.txt

```
addi $t1, $zero, 5000        # Set begin to 5000

lw $t3, 0($t1)
add $t2,$t2,$t3
sw $t2,0($t1)
addi $t1,$t1,4

lw $t3, 0($t1)
add $t2,$t2,$t3
sw $t2,0($t1)
addi $t1,$t1,4

lw $t3, 0($t1)
add $t2,$t2,$t3
sw $t2,0($t1)
addi $t1,$t1,4

lw $t3, 0($t1)
add $t2,$t2,$t3
```

```
sw $t2,0($t1)
addi $t1,$t1,4

lw $t3, 0($t1)
add $t2,$t2,$t3
sw $t2,0($t1)
addi $t1,$t1,4

lw $t3, 0($t1)
add $t2,$t2,$t3
sw $t2,0($t1)
addi $t1,$t1,4

lw $t3, 0($t1)
add $t2,$t2,$t3
sw $t2,0($t1)
addi $t1,$t1,4

lw $t3, 0($t1)
add $t2,$t2,$t3
sw $t2,0($t1)
```

Figure 3. Verilog code for the proposed solution



```
# run -all
# Data at address 5000 is          2
# Data at address 5004 is          0
# Data at address 5008 is          7
# Data at address 5012 is          9
# Data at address 5016 is          0
# Data at address 5020 is          2
# Data at address 5024 is          5
# Data at address 5028 is         10
# ** Note: $stop    : I:/#EE/EE 310/Lab 6/MIPSzy_TB.v(127)
#    Time: 2180 ns  Iteration: 0  Instance: /MIPSzy_TB
# Break in Module MIPSzy_TB at I:/#EE/EE 310/Lab 6/MIPSzy_TB.v line 127
```

Part 1

Figure 4. Lab pictures of the running solution