Lab 5 – MIPS Datapath for R-type and I-Type Instructions

CECS 341 – Computer Architecture & Organization

Kenry Yu, 028210726

Olenka Bilinska, 028897191

Garret Towner, 028303091

Professor: Mandy He

California State University, Long Beach

College of Engineering

1250 Bellflower Blvd, Long Beach, CA 90840

April 13, 2022

Goal/Objective:

The goal of this lab is to modify the existing MIPS datapath for R-type instruction from lab 4 to support and implement I-type instructions.

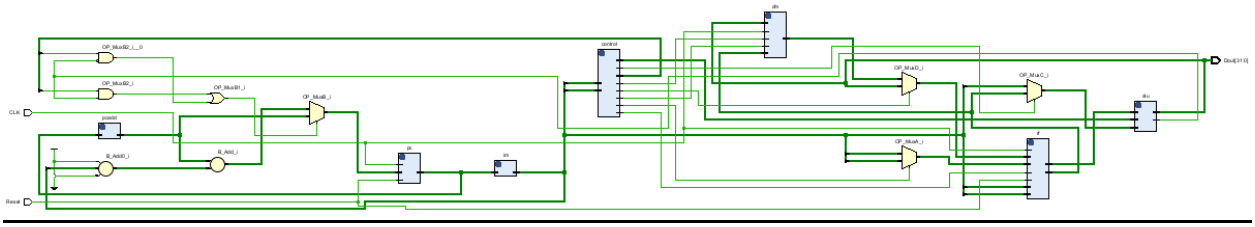
Technical Description/Steps:

The first step of the project is to modify the control module of the MIPS datapath, we added several extra variables to determine the behavior and add a switch statement for the I-type instructions.

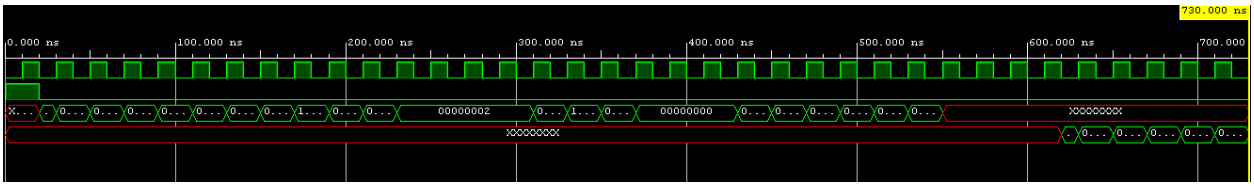
The second step of the project is to modify the datapath module of the MIPS datapath, we added several variables and wires in the module to connect the new variables added in the control module. We also added a few multiplexers to determine the data that we want to input to the ALU.

The third step of the project is to modify the testbench and import the new data files provided by the instructor. After that, we ran the testbench and compared it to our handwritten calculation.

Results:



The image above is the schematic of the datapath module, it shows that our modules are connected together with the new multiplexers we added. In addition, the “DataMem.v” file provided by the instructor is also connected in the schematic.



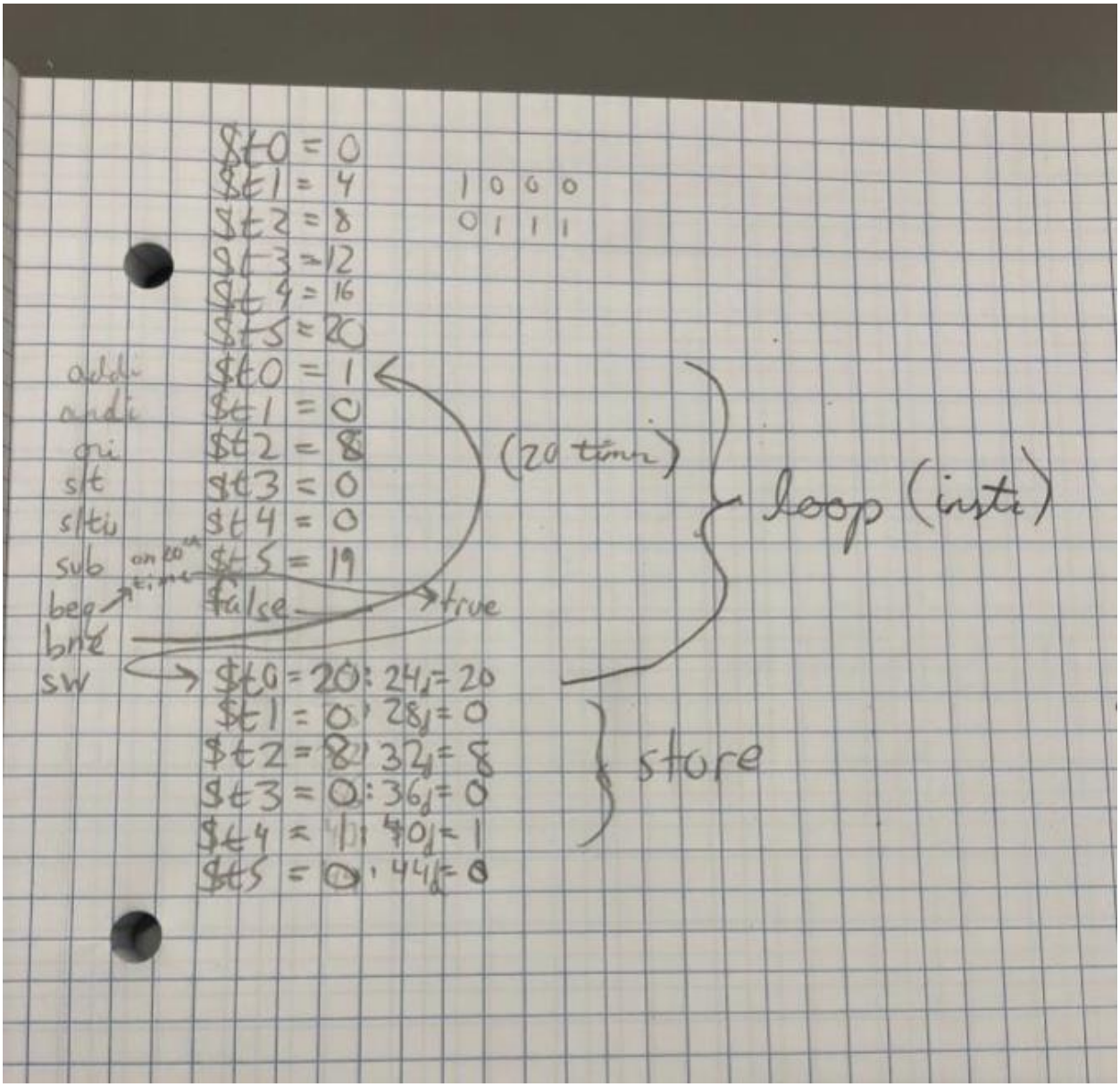
The image above is the waveform of the simulation of the MIPS datapath. The waveform shows the program began running after the reset turns to 0. After that, the program is fetched and decoded the

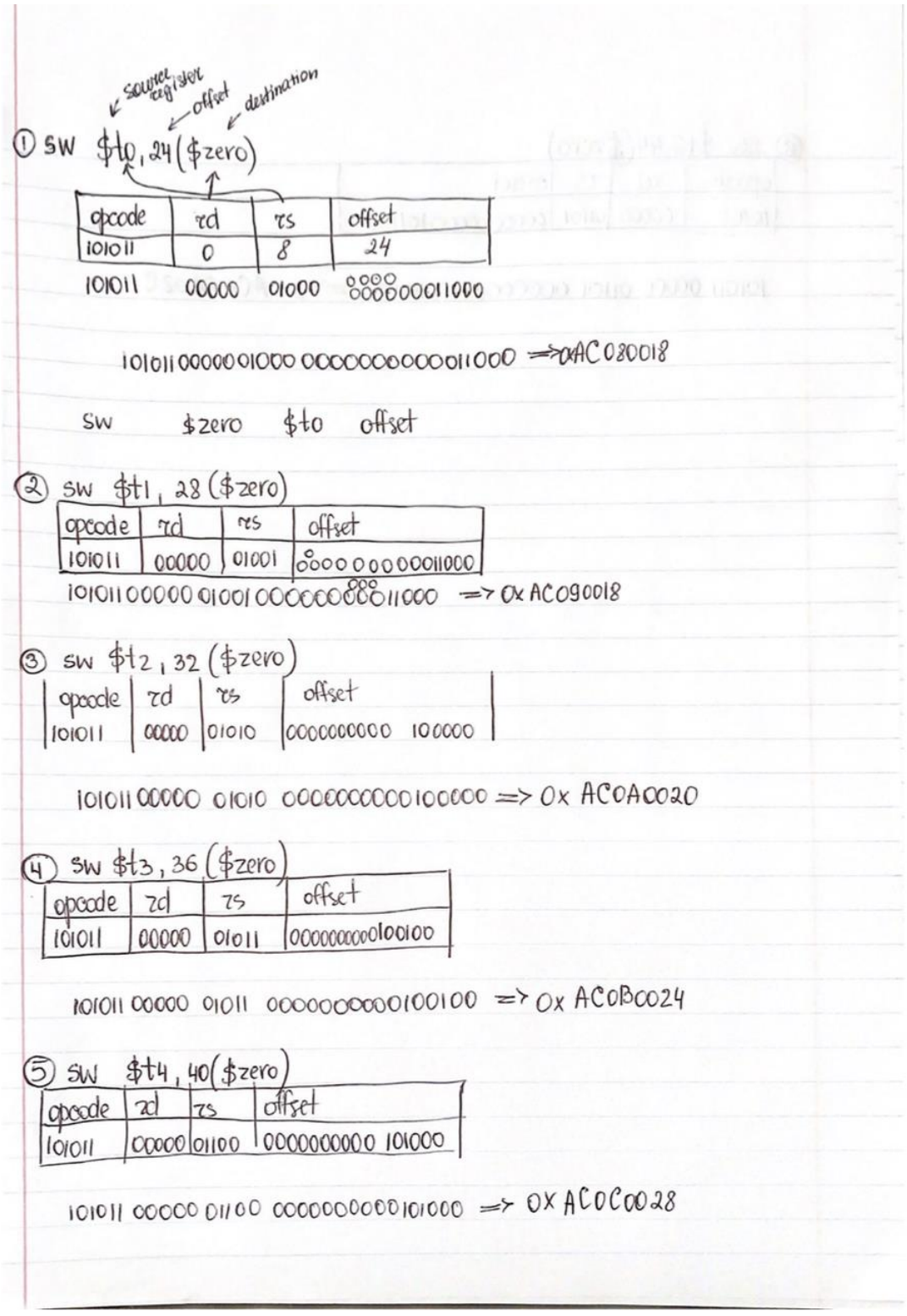
instruction from the “DataMem.dat” and the “imem - Copy.dat” files. The program fetched the data from the register files, performed calculations, then stored the results back to the register files. The output of the ALU became “XXXXXX” in the waveform because there’s no output for store word instructions.

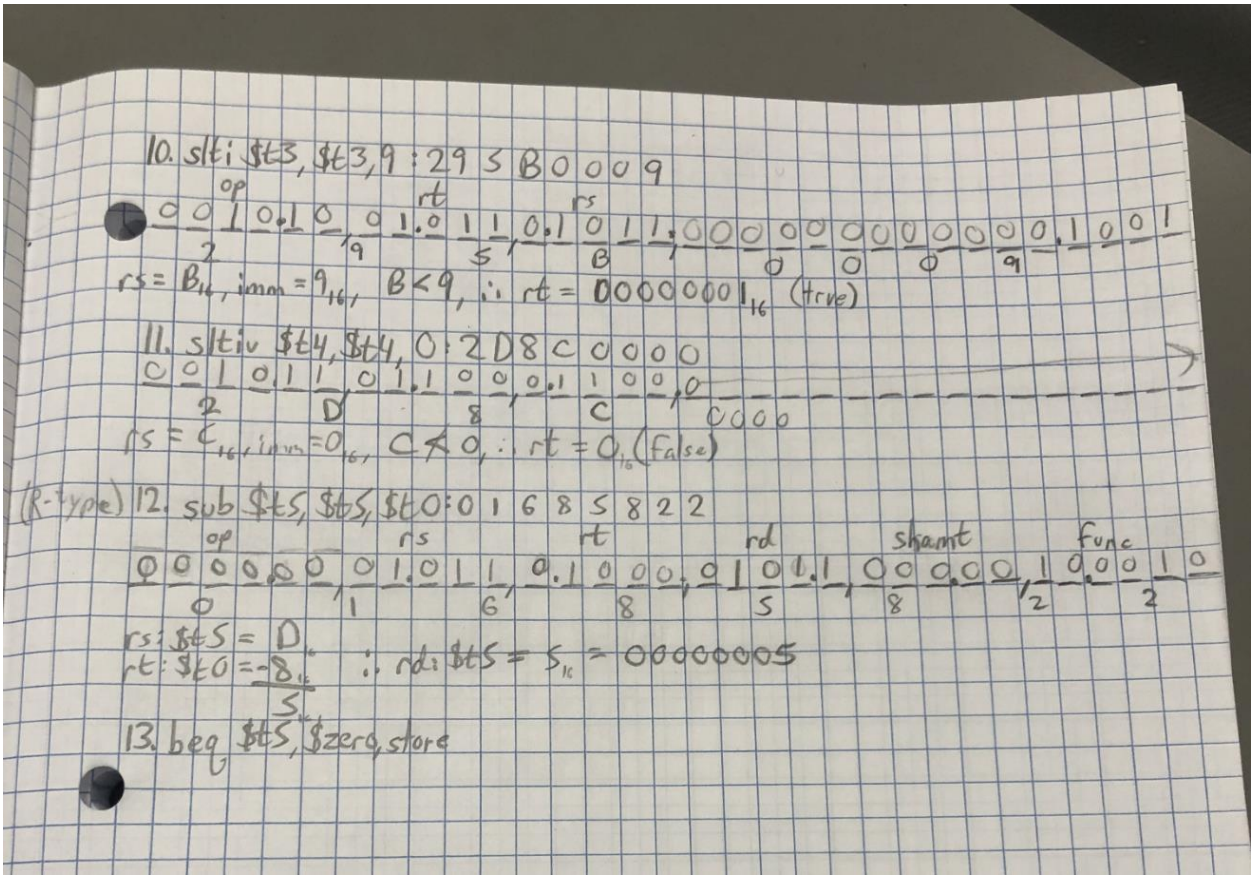
```
# run 1000ns
t=  630.0ns dm[24] 00000002
t=  650.0ns dm[28] 00000000
t=  670.0ns dm[32] 1234567f
t=  690.0ns dm[36] 00000001
t=  710.0ns dm[40] 00000000
t=  730.0ns dm[44] 00000000
```

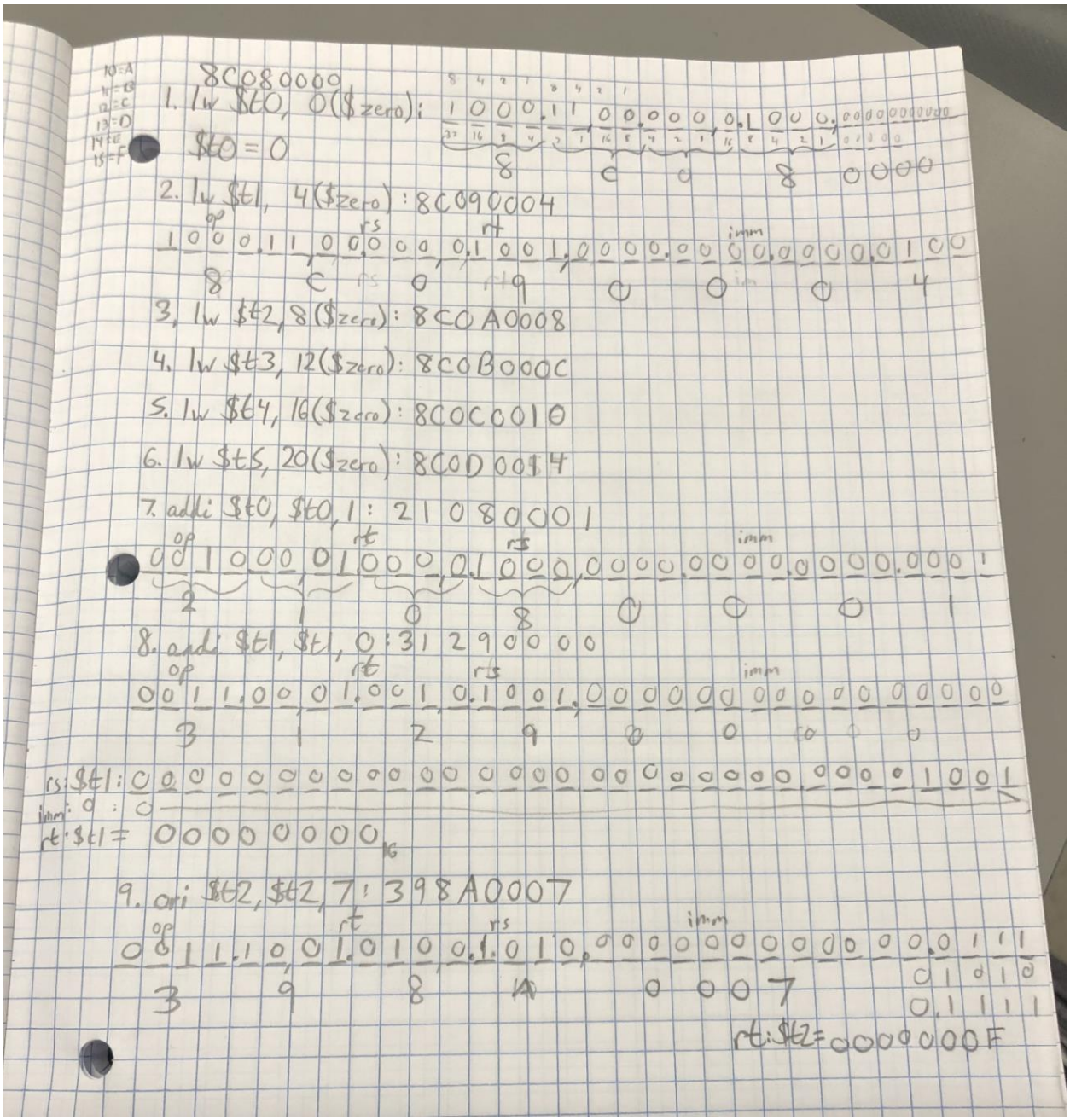
The image above is the console output of the program, it displays the result of the instructions that were stored in the specified registers. Our handwritten calculation matches the operation performed

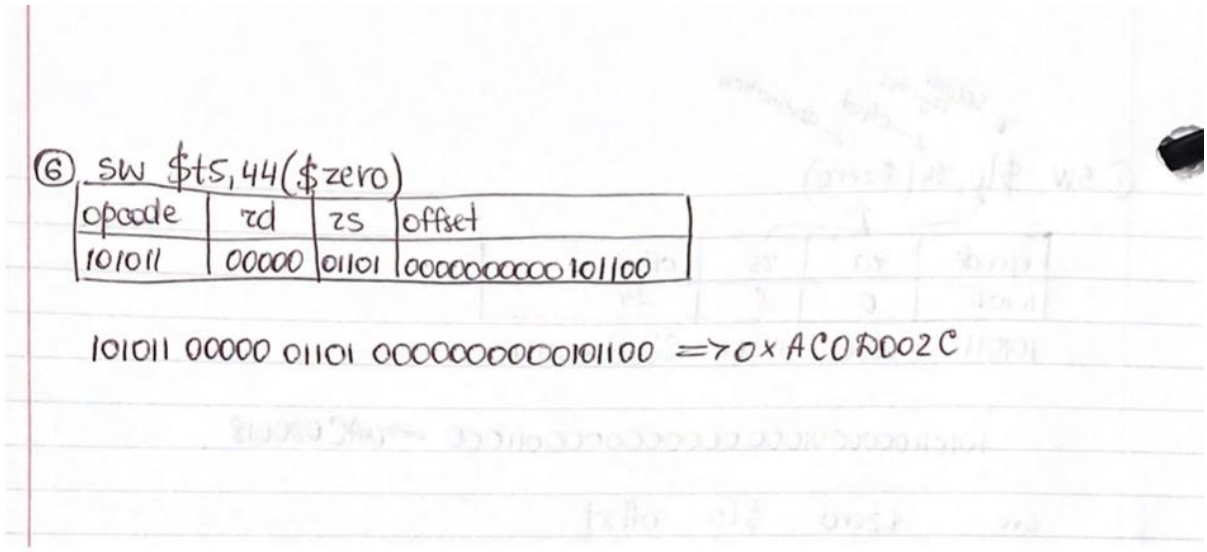
by the project, meaning that our module codes are working as intended.




source register
offset
destination
① sw $t0, 24($zero)
opcode | rd | rs | offset
101011 | 0 | 8 | 24
101011 00000 01000 0000000011000
101011000000 01000 0000000000011000 => 0xAC080018
sw $zero $t0 offset
② sw $t1, 28($zero)
opcode | rd | rs | offset
101011 | 00000 | 01001 | 0000000000011100
101011 00000 01001 0000000000011100 => 0x AC090018
③ sw $t2, 32($zero)
opcode | rd | rs | offset
101011 | 00000 | 01010 | 0000000000100000
101011 00000 01010 0000000000100000 => 0x AC0A0020
④ sw $t3, 36($zero)
opcode | rd | rs | offset
101011 | 00000 | 01011 | 0000000000100100
101011 00000 01011 0000000000100100 => 0x AC0B0024
⑤ sw $t4, 40($zero)
opcode | rd | rs | offset
101011 | 00000 | 01100 | 0000000000101000
101011 00000 01100 0000000000101000 => 0x AC0C0028


10. slti $t3, $t3, 9 : 29 5B 00 09
op rt rs
001010 01011 01011 0000 0000 0000 1001
2 9 5 B 0 0 0 9
rs = B16, imm = 916, B < 9, ∴ rt = 0000 0001 16 (true)
11. sltiu $t4, $t4, 0 : 2D 8C 00 00
001011 01100 01100 0000 0000 0000 0000
2 D 8 C 0 0 0 0
rs = C16, imm = 016, C ≠ 0, ∴ rt = 016 (false)
(R-type) 12. sub $t5, $t5, $t0 : 01 68 58 22
op rs rt rd shamt funct
000000 01011 01000 01011 00000 100010
0 1 6 8 5 8 2 2
rs: $t5 = D16
rt: $t0 = -816
5
∴ rd: $t5 = 516 = 0000 0005
13. beq $t5, $zero, store


8C080000
1. lw $t0, 0($zero):
$t0 = 0
2. lw $t1, 4($zero): 8C090004
3. lw $t2, 8($zero): 8C0A0008
4. lw $t3, 12($zero): 8C0B000C
5. lw $t4, 16($zero): 8C0C0010
6. lw $t5, 20($zero): 8C0D0014
7. addi $t0, $t0, 1: 21080001
8. addi $t1, $t1, 0: 31290000
rs: $t1:
imm: 0 :
rt: $t1 =
9. ori $t2, $t2, 7: 398A0007
rt: $t2 = 0000000F



Conclusion:

In this lab, we learned how to modify the project from the last lab to support I-type instructions in the datapath. We also learned how to use the syntax for multiplexers to determine the input data that we want. Over the project, we encountered two challenges that slowed our progress.

First, our console output wasn't displaying any numbers even when the waveform shows the program is running as intended. We realized that we were supposed to delay the display statement until when the store word instructions were completed. Moving the display statement under the time statement fixed the problem immediately.

Second, the result of the "slti" instructions does not match our handwritten calculation. After examining the behavior of "slti" instruction, we changed our codes in ALU and it began to output correct results.