CDA 4205 Course Project Report

Date: December 10, 2021
Group Members: Ricardo Frumento, Shawn Broyles, Ahmed El Maliki, Andre Roberts

Objectives

For this project, we were tasked with implementing a single-cycle RISC-V processor with hardware description language that supports the given RISC-V instructions. We have decided to use Logisim to create the circuit. The table below shows the testbench program that our circuit implements.

Table 1: RISC-V Testbench

PC	Machine Code	Basic Code	Original Code
0x00	0x00500113	addi x2 x0 5	main: addi x2, x0, 5
0x04	0x00C00193	addi x3 x0 12	addi x3, x0, 12
0x08	0xFF718393	addi x7 x3 -9	addi x7, x3, -9
0x0C	0x0023E233	or x4 x7 x2	or x4, x7, x2
0x10	00x0041F2B3	and x5 x3 x4	and x5, x3, x4
0x14	0x004282B3	add x5 x5 x4	add x5, x5, x4
0x18	0x02728863	beq x5 x7 48	beq x5, x7, end
0x1C	0x0041A233	slt x4 x3 x4	slt x4, x3, x4
0x20	0x00020463	beq x4 x0 8	beq x4, x0, around
0x24	0x00000293	addi x5 x0 0	addi x5, x0, 0
0x28	0x0023A233	slt x4 x7 x2	around: slt x4, x7, x2
0x2C	0x005203B3	add x7 x4 x5	add x7, x4, x5
0x30	0x402383B3	sub x7 x7 x2	sub x7, x7, x2
0x34	0x0471AA23	sw x7 84(x3)	sw x7, 84(x3)
0x38	0x06002103	lw x2 96(x0)	lw x2, 96(x0)
0x3C	0x005104B3	add x9 x2 x5	add x9, x2, x5
0x40	0x008001EF	jal x3 8	jal x3, end
0x44	0x00100113	addi x2 x0 1	addi x2, x0, 1
0x48	0x00910133	add x2 x2 x9	end: add x2, x2, x9
0x4C	0x0221A023	sw x2 32(x3)	sw x2, 32(x3)
0x50	0x00210063	beq x2 x2 0	done: beq x2, x2, done

Instructions Implemented

Our single-cycle RISC-V processor implements the following types of instructions:

- R-type instructions: add, sub, and, or, slt
- I-type instructions: addi
- Memory instructions: lw, sw
- Branch instruction: beq
- J-type instruction: jal

Processor Modules And Components

The single-cycle datapath circuit our team has created in Logisim includes a program counter, instruction memory, control unit, immediate generator, register memory, ALU, and data memory. Since it uses a clock cycle, it is a sequential circuit. There are basic logic components used throughout the circuit, such as multiplexers, splitters, adders, pins, registers, bit extenders, wires, AND gates, OR gates, NOT gates, and XOR gates.

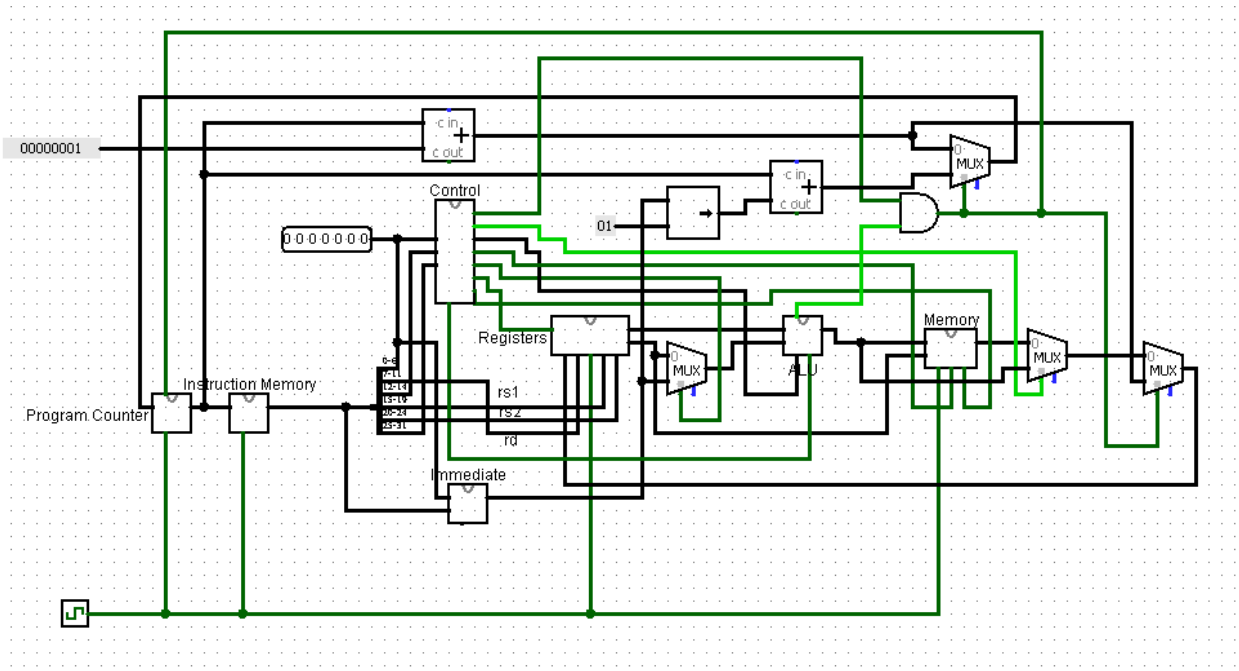

Figure 1: Single-cycle datapath circuit

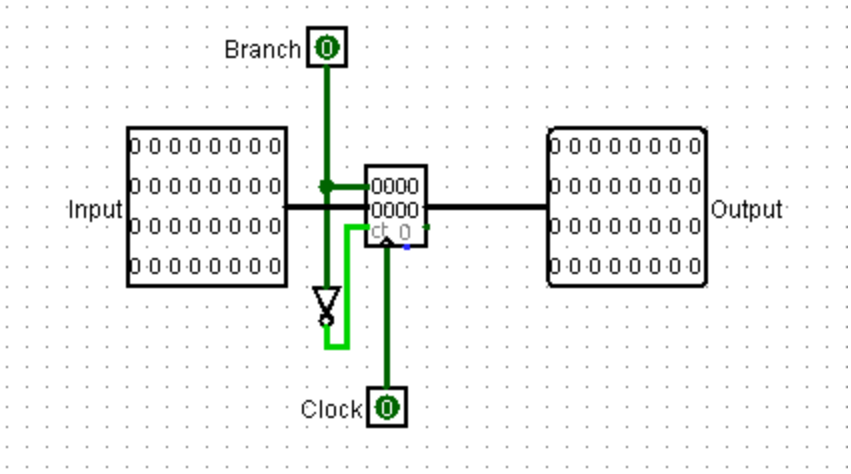

Figure 2: Program counter module

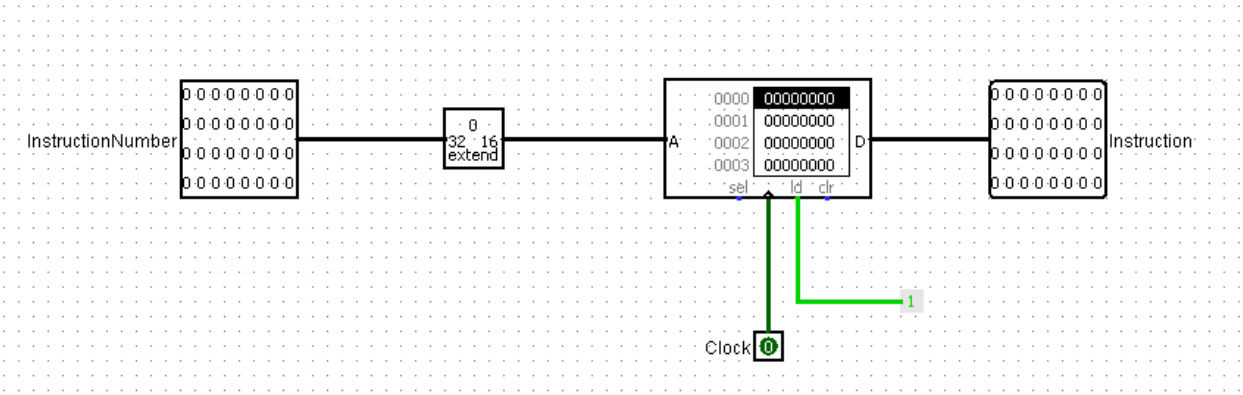

Figure 3: Instruction memory module

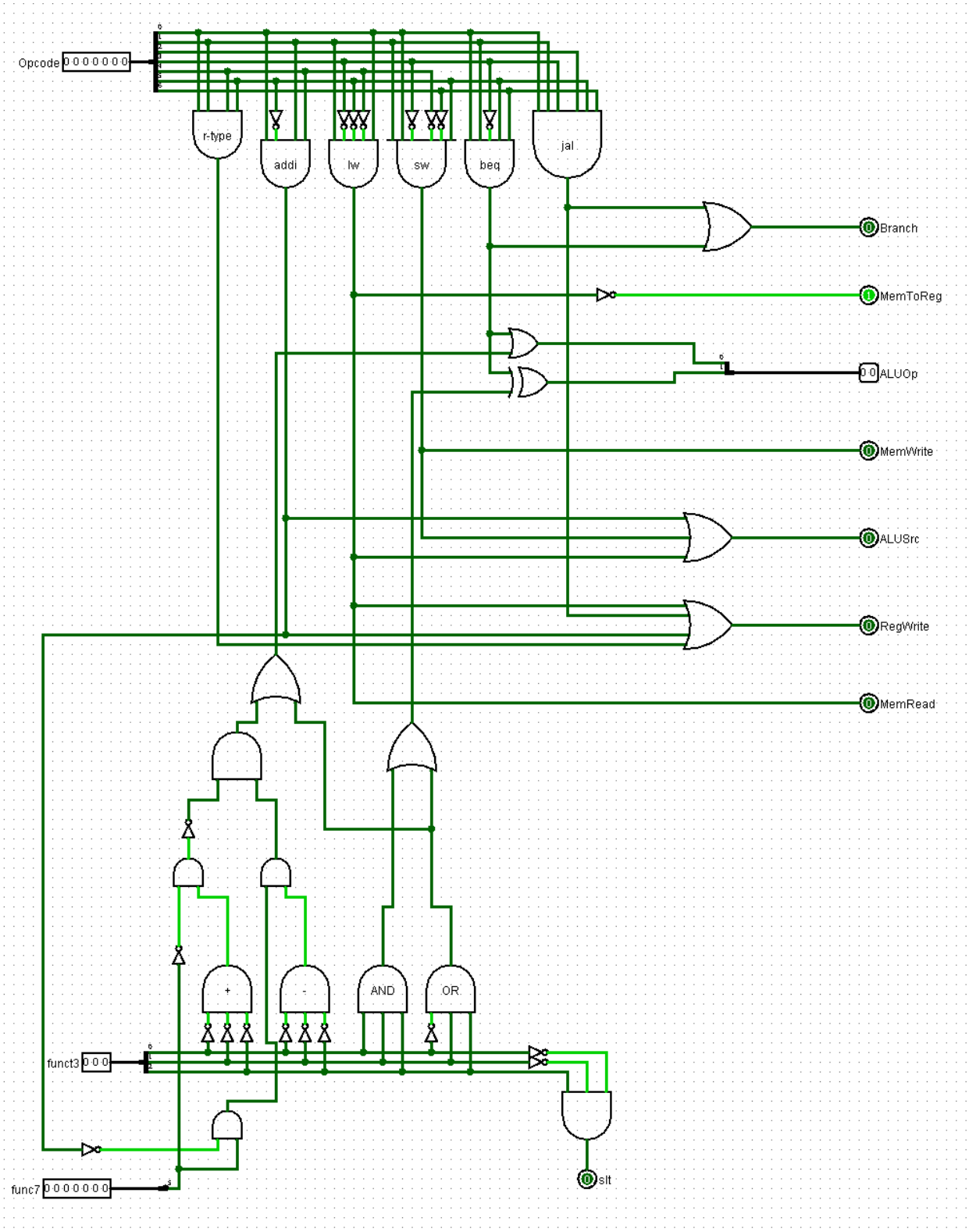

Figure 4: Control unit module

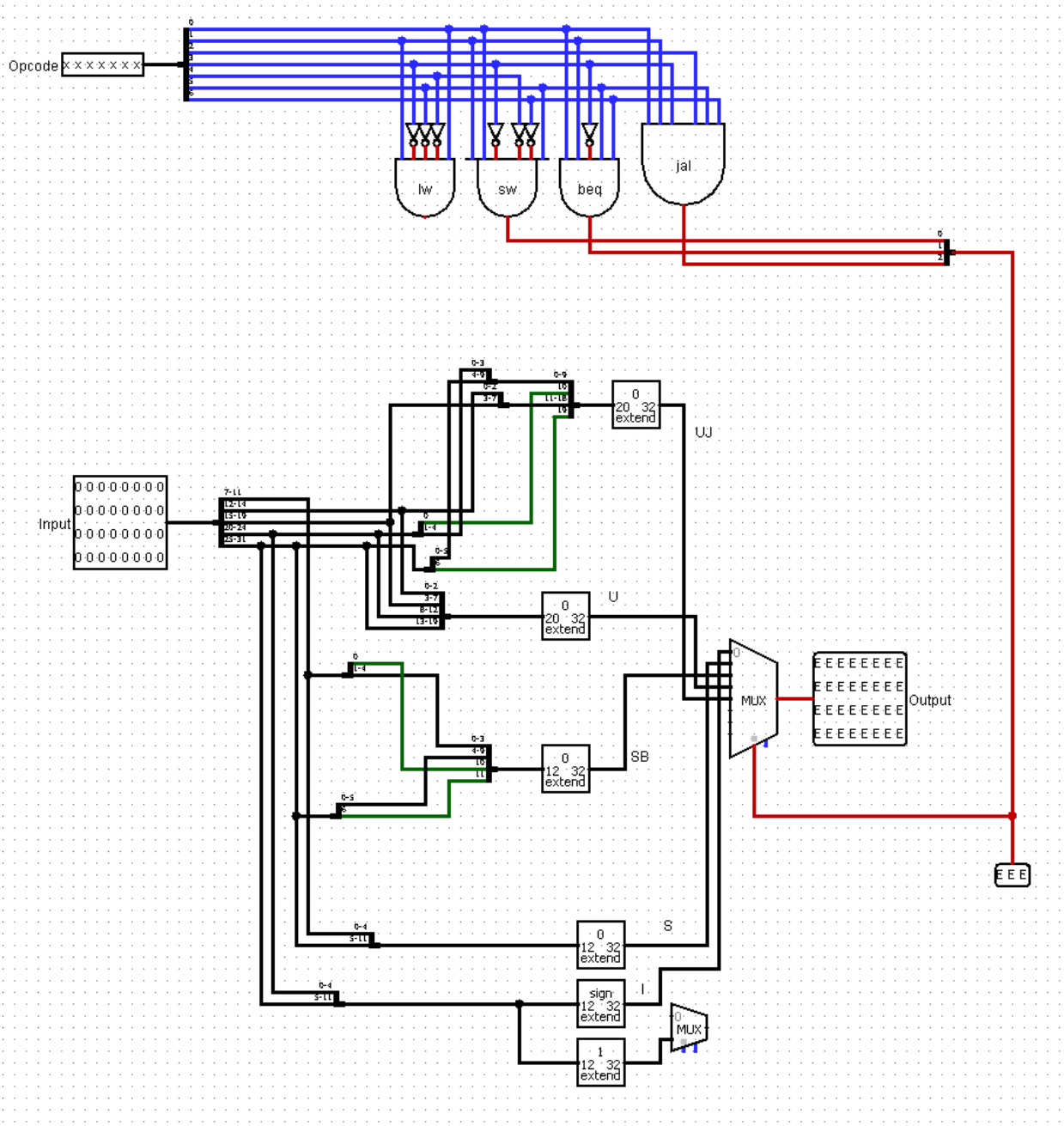

Figure 5: Immediate generator module

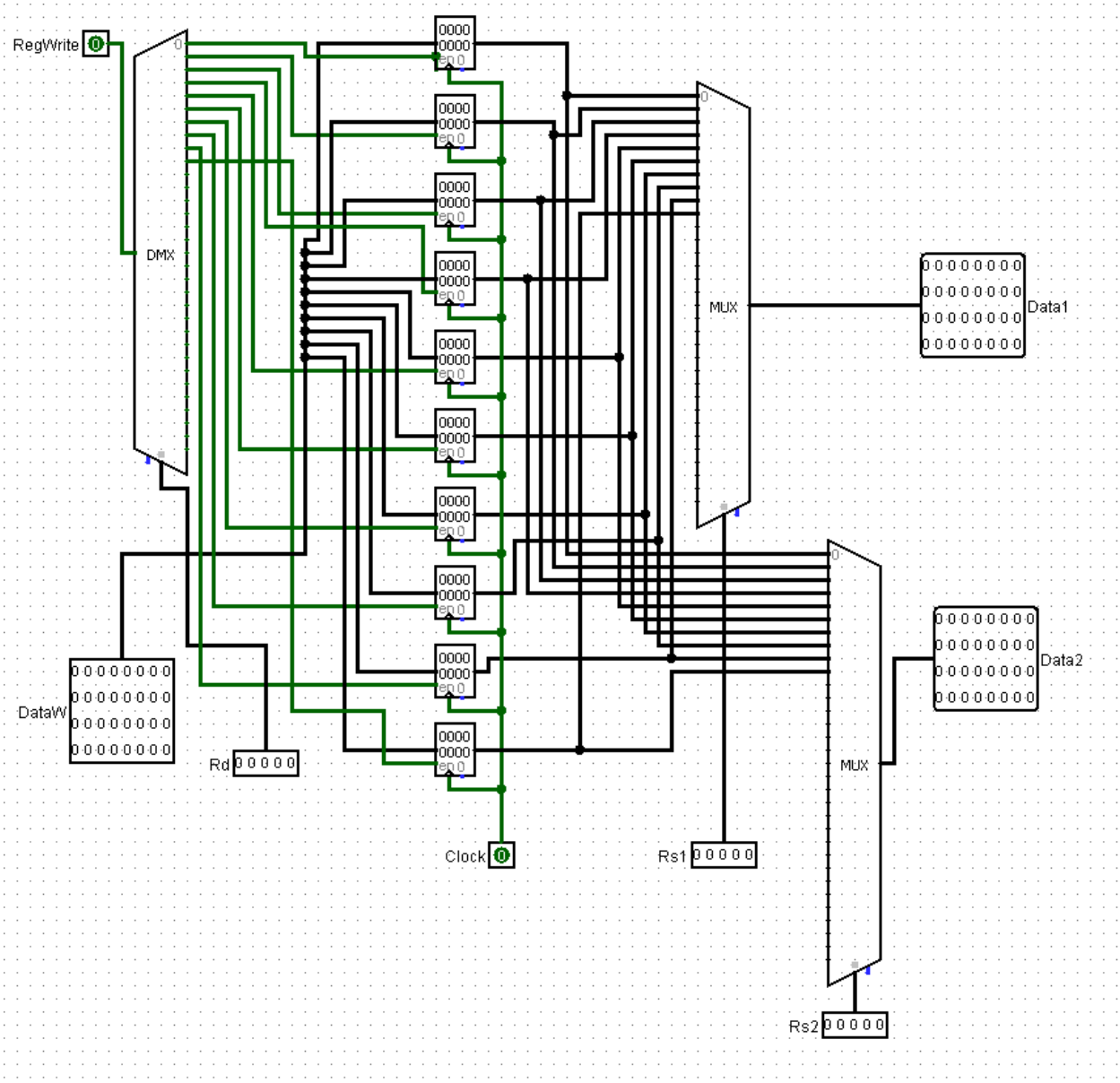

Figure 6: Register memory module

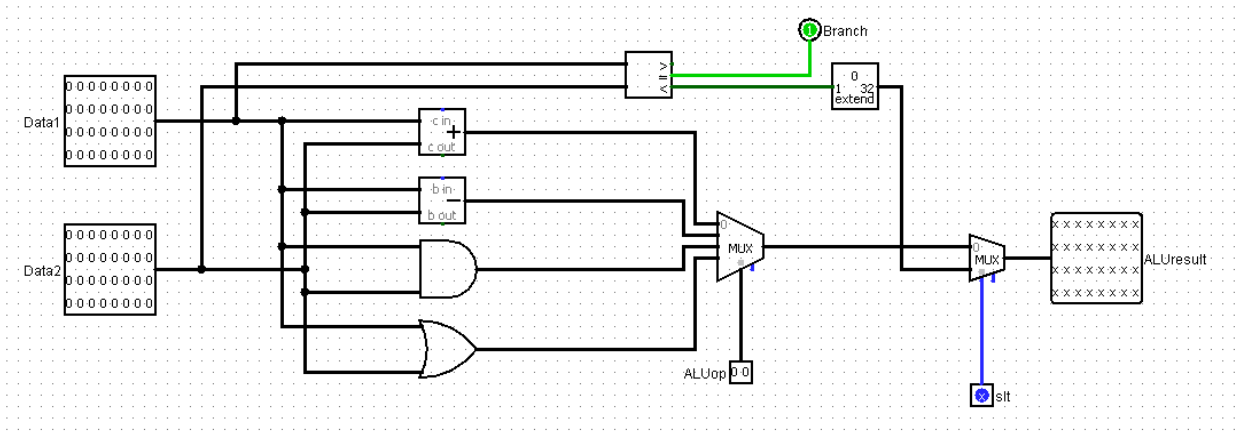

Figure 7: ALU module

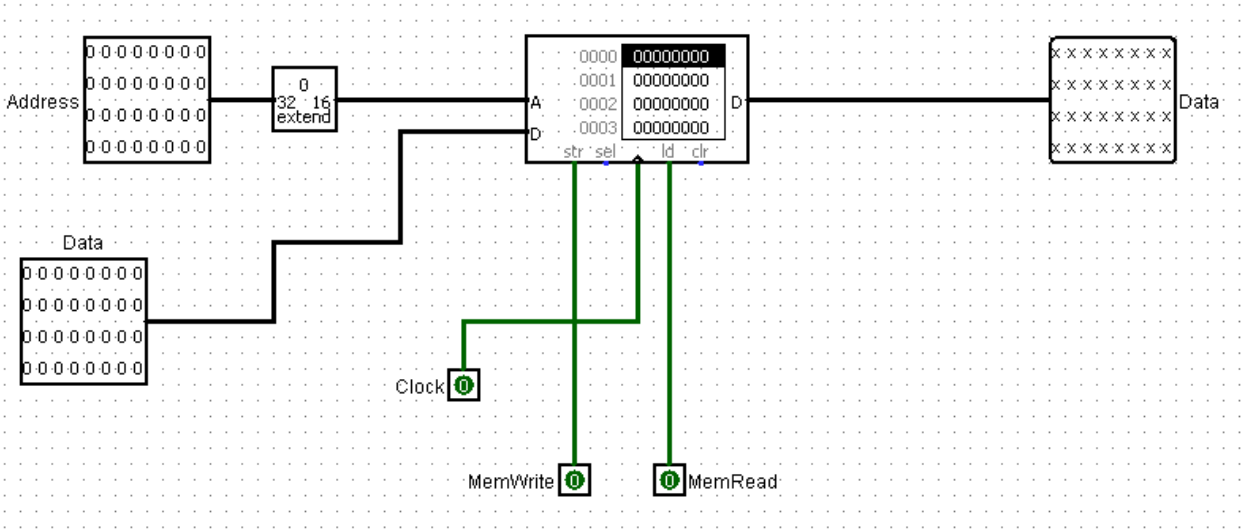

Figure 8: Data memory module

Testbench Files

To test our single-cycle RISC-V processor and ensure that the entire circuit works properly, we have created a couple of small testbench files to test the functionality. Each file contains a set of machine code that the circuit can run to check if everything is working as intended. The names of the machine code files and their contents are shown below.

TB_add
v2.0 raw
528293 530313 628233

TB_sub
v2.0 raw
528293 630313 40530233

TB_and
v2.0 raw
528293 630313 62f233

TB_or
v2.0 raw
528293 630313 62e233

TB_slt
v2.0 raw
310113 518193 312233

TB_sw
v2.0 raw
528293 532023

TB_lw
v2.0 raw
120213 22283

TB_jal
v2.0 raw
8000ef 110113 101b3

TB_beq
v2.0 raw
463 108093 110113

Below is the machine code file that we created from the given testbench file in Table 1. Our single-cycle RISC-V processor circuit is able to load the file into memory and then it executes the instructions. After the program finishes, the value 25 is written to address 100.

TB_logisimcode
v2.0 raw
500113 c00193 ff718393 23e233 41f2b3 4282b3 2728863 41a233
20463 293 23a233 5203b3 402383b3 471aa23 6002103 5104b3
8001ef 100113 910133 221a023 210063

Results For Each Module:

Testbench for TB_add file:
Hex instructions:
00528293
00530313
0062f233

Machine code:

000000000101 00101 000 00101 0010011
000000000101 00110 000 00110 0010011
0000000 00110 00101 000 00100 0110011

RISC-V assembly:
addi x5, x5, 5
addi x6, x6, 5
add x4, x5, x6

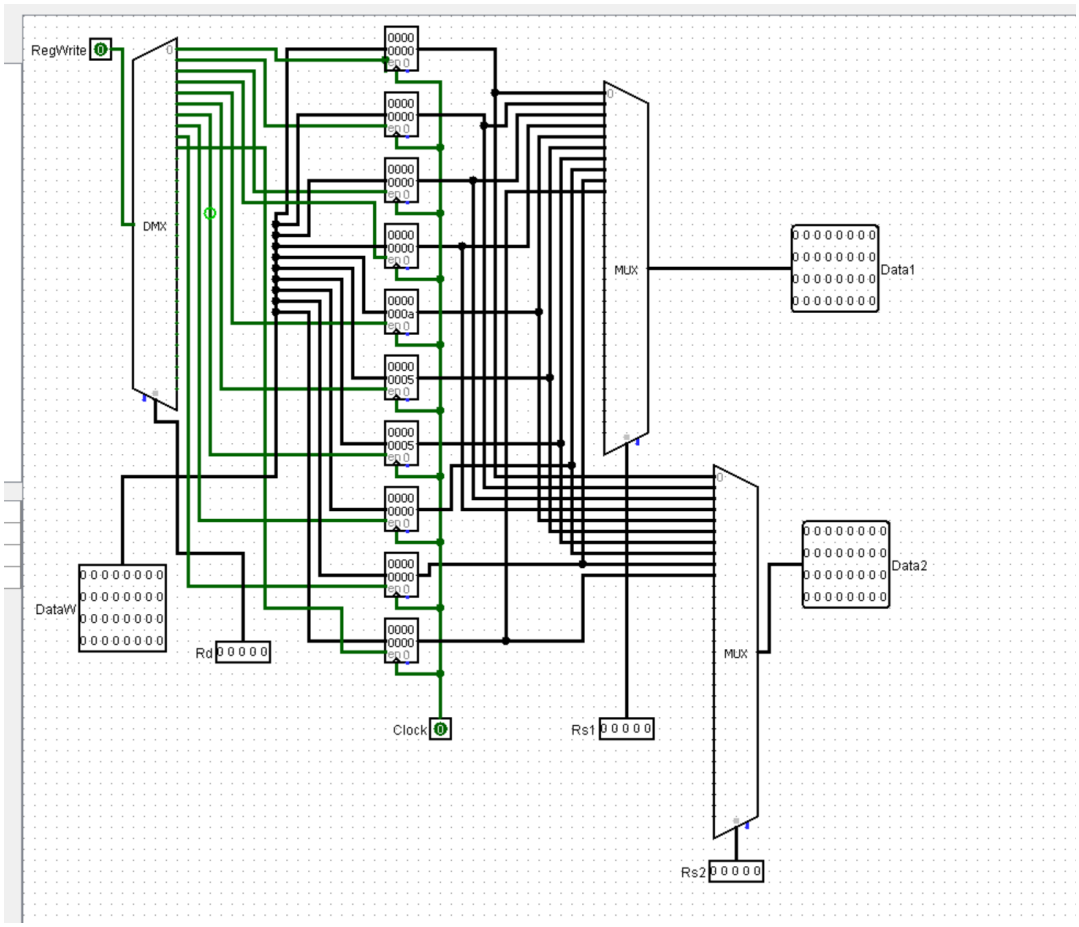

Figure 9: After running the TB_add testbench, register x4 is 10, register x5 is 5, and register x6 is 5.

Testbench for TB_sub file:

Hex instructions:
00528293
00630313
40530233

Machine code:
000000000101 00101 000 00101 0010011
000000000110 00110 000 00110 0010011
0100000 00101 00110 000 00100 0110011

Assembly:
addi x5, x5, 5
addi x6, x6, 6
sub x4, x6, x5

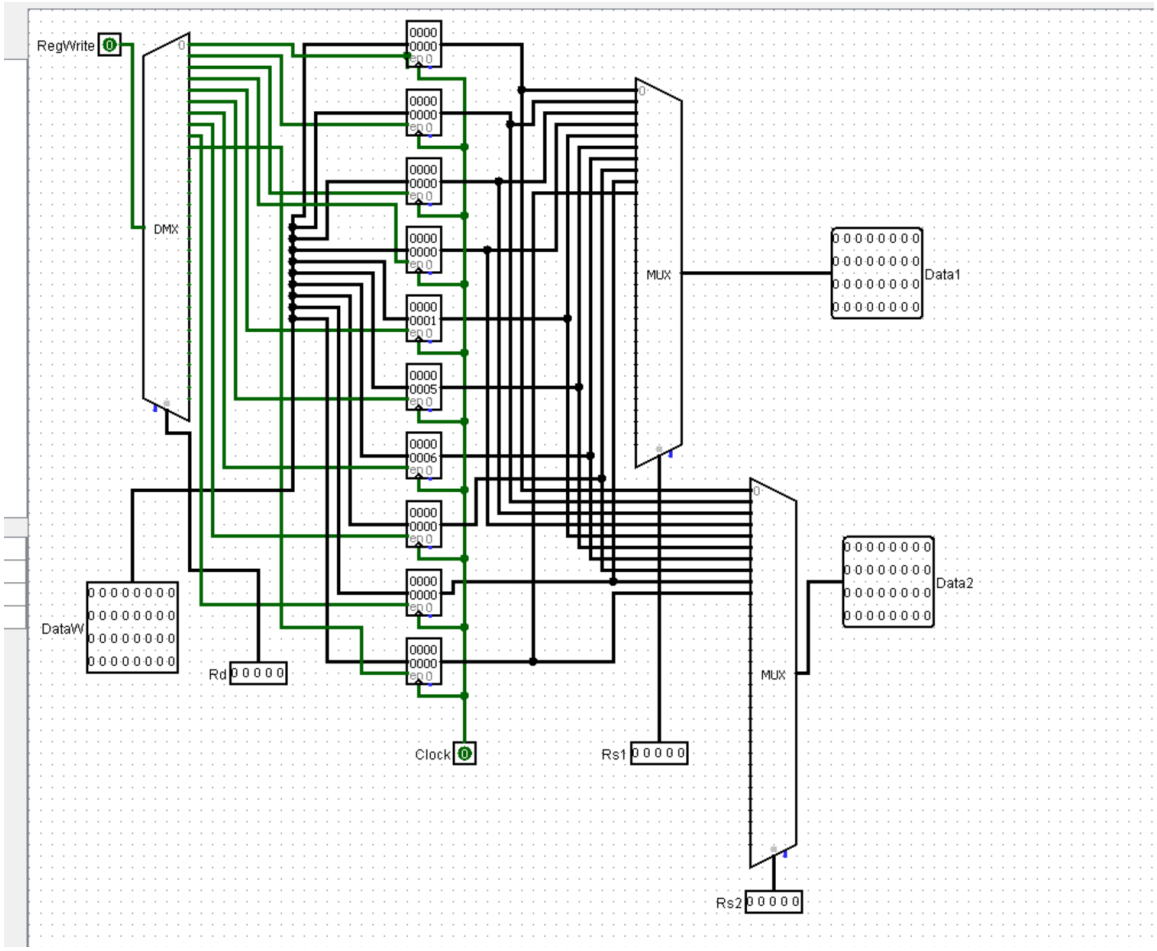

Figure 10: After running the TB_sub testbench, register x4 is 1, register x5 is 5, and register x6 is 6.

Testbench for TB_and file:
Hex instructions:
00528293
00630313
0062f233

Machine code:
000000000101 00101 000 00101 0010011
000000000110 00110 000 00110 0010011
0000000 00110 00101 111 00100 0110011

RISC-V assembly:

addi x5, x5, 5
addi x6, x6, 6
and x4, x5, x6

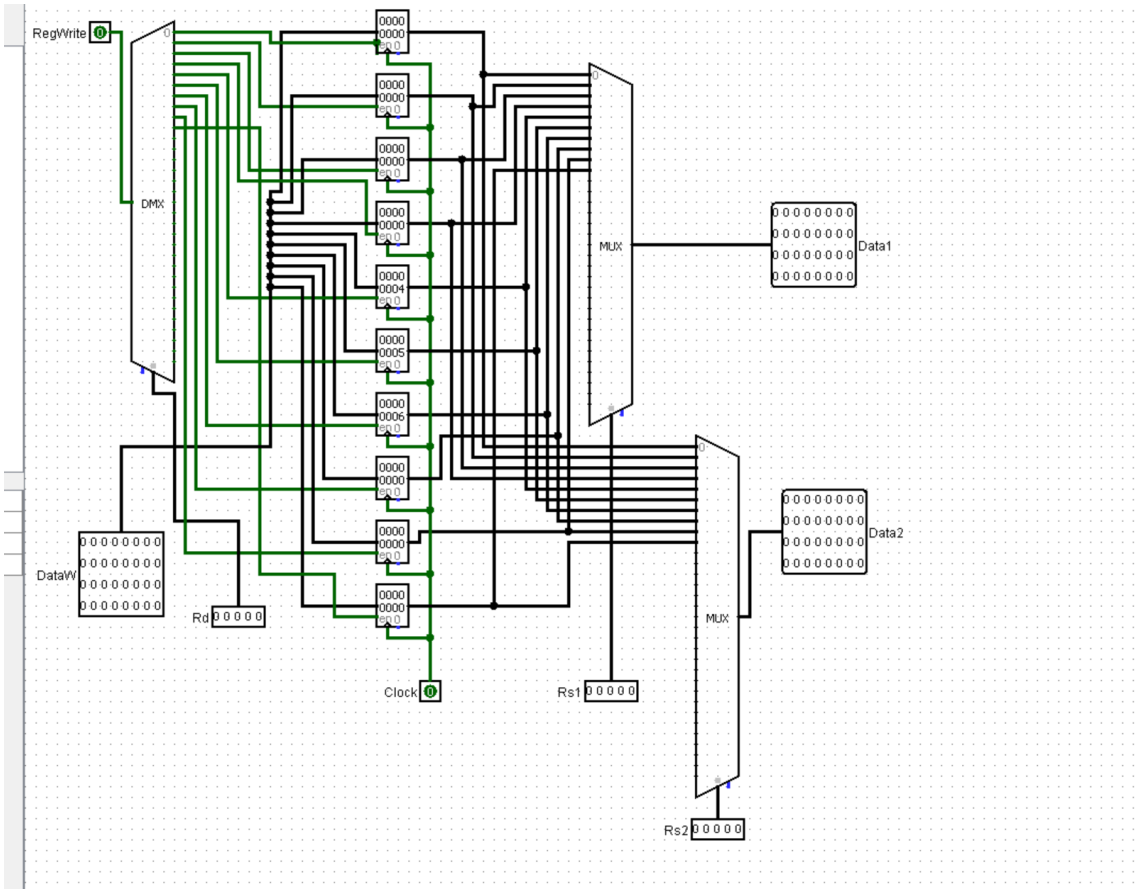

Figure 11: After running the TB_and testbench, register x4 is 4, register x5 is 5, and register x6 is 6.

Testbench for TB_or file:
Hex instructions:
00528293
00630313
0062e233

Machine code:
000000000101 00101 000 00101 0010011
000000000110 00110 000 00110 0010011
0000000 00110 00101 110 00100 0110011

RISC-V assembly:
addi x5, x5, 5
addi x6, x6, 6
or x4, x5, x6

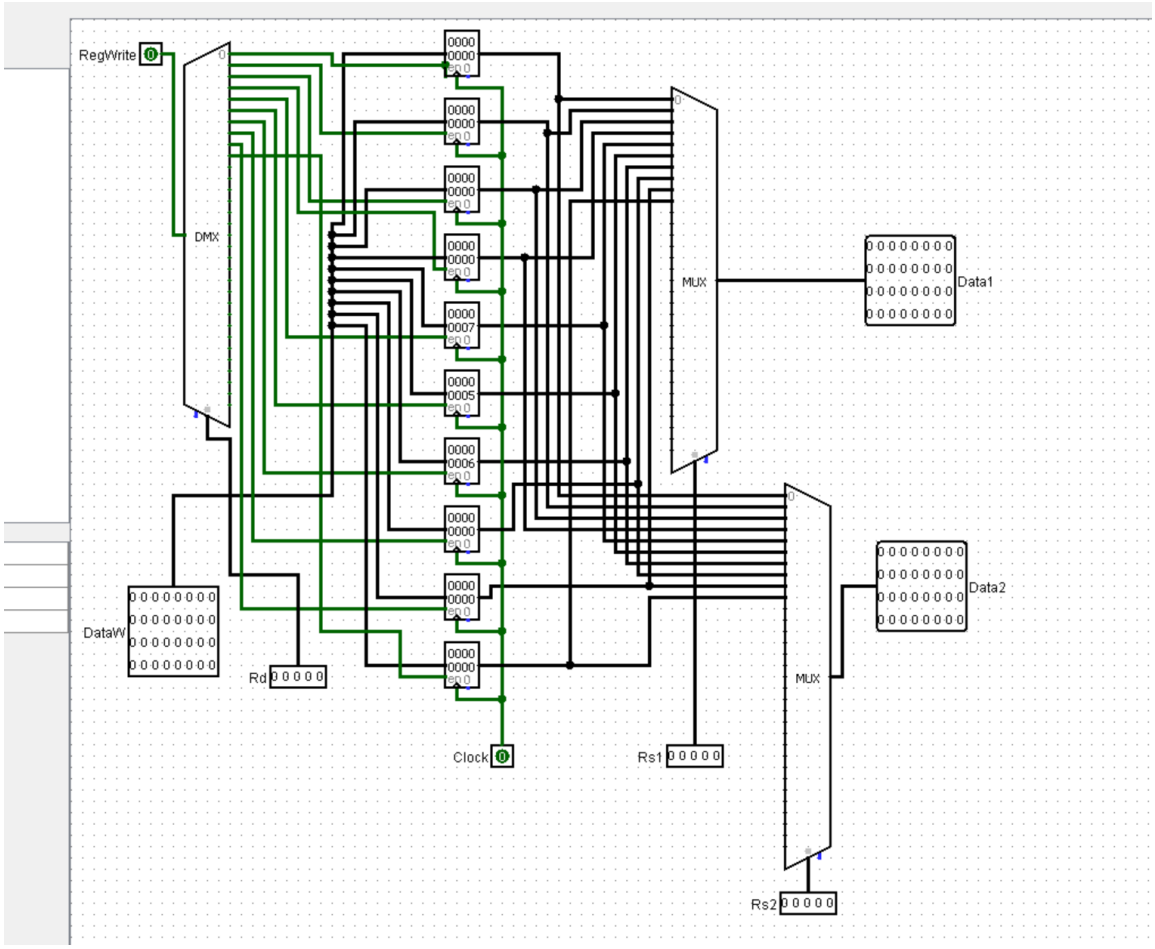

Figure 12: After running the TB_or testbench, register x4 is 7, register x5 is 5, and register x6 is 6.

Testbench for TB_slt file:
Hex instructions:
00310113
00518193
00312233

Machine code:
000000000011 00010 000 00010 0010011
000000000101 00011 000 00011 0010011
0000000 00011 00010 010 00100 0110011

RISC-V assembly:
addi x2, x2, 3
addi x3, x3, 5
slt x4, x2, x3

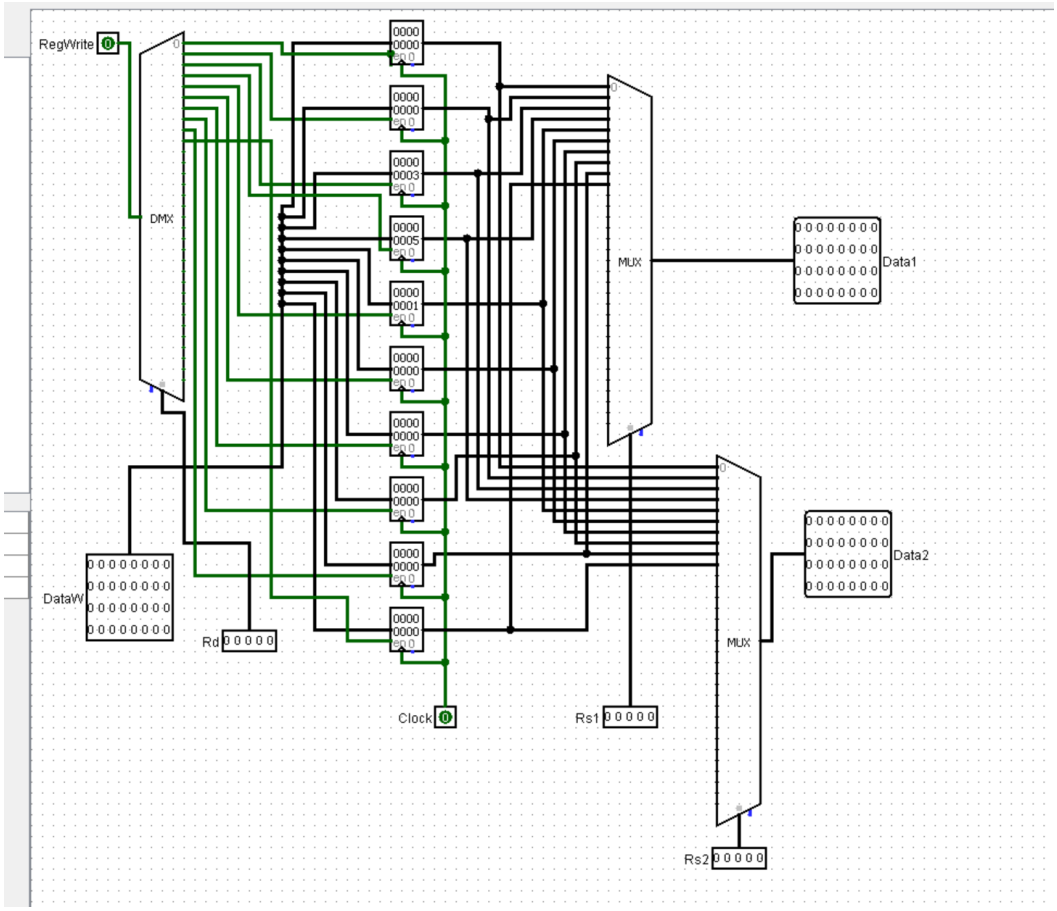

Figure 13: After running the TB_slt testbench, register x2 is 3, register x3 is 5, and register x4 is 1.

Testbench for TB_sw file:
Hex instructions:
00528293
00532023

Machine code:
0000000 00101 00101 000 00101 0010011
0000000 00101 00110 010 00000 0100011

RISC-V assembly:
addi x5, x5, 5
sw x5, 0(x6)

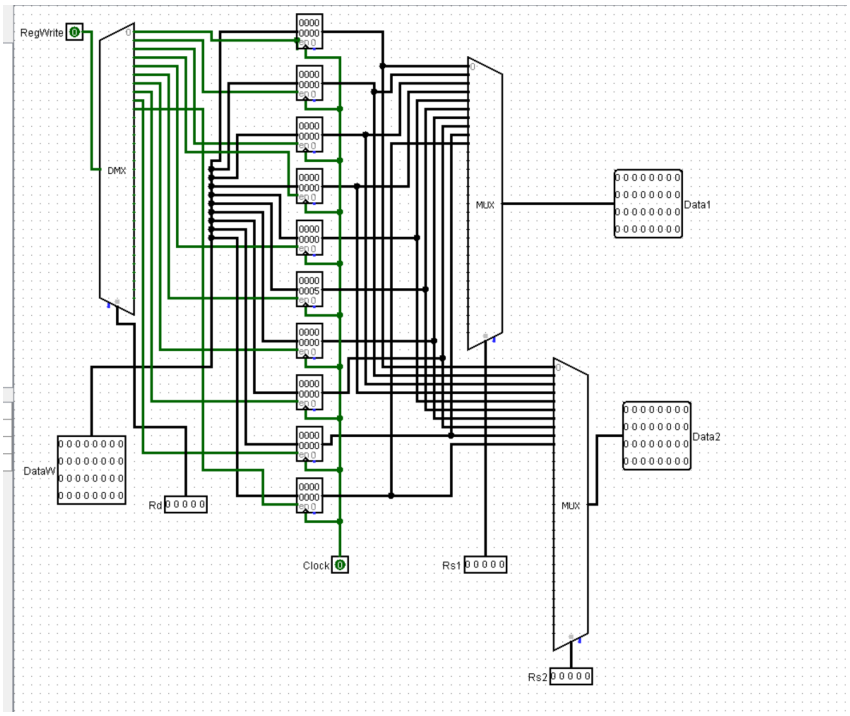

Figure 14: After running the TB_sw testbench, register x5 is 5.

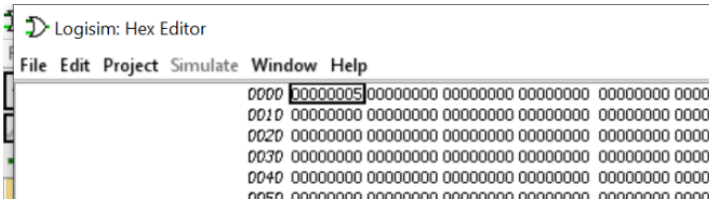

Figure 15: After running the TB_sw testbench, the data memory module contains the number 5 at memory location 0x00.

Testbench for TB_lw file:
Hex instructions:
00120213
00022283

Machine code:
0000000000001 00100 000 00100 0010011
0000000000000 00100 010 00101 0000011

RISC-V assembly:
addi x4, x4, 1
lw x5, 0(x4)

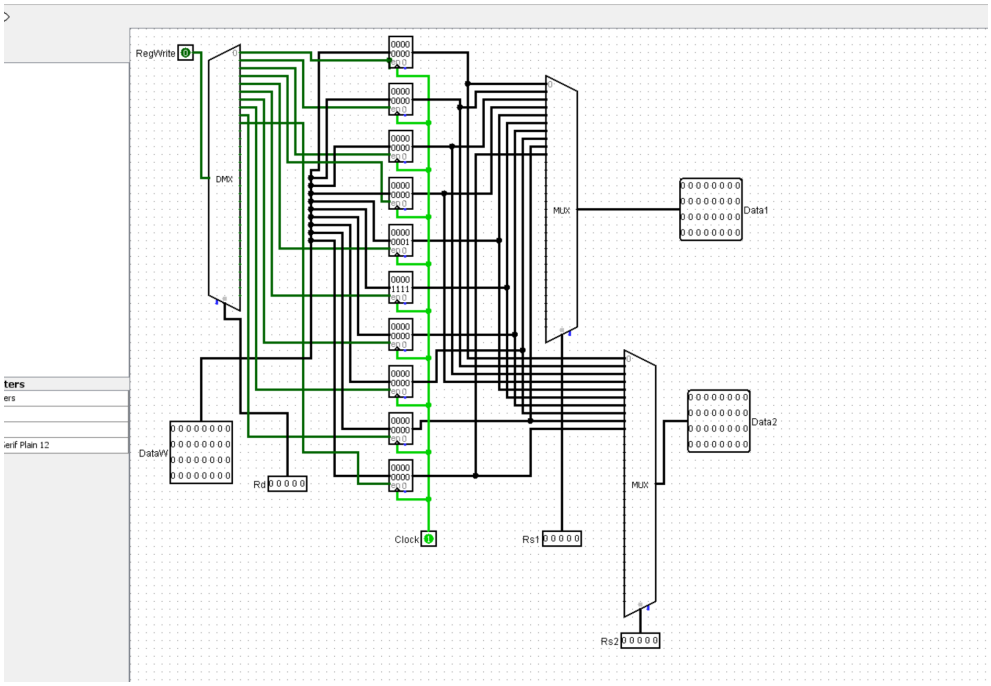

Figure 16: After running the TB_lw testbench, register x4 is 1 and register x5 is 15.

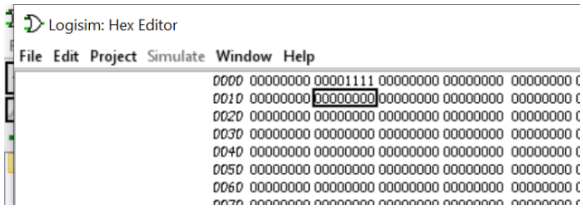

Figure 17: After running the TB_lw testbench, memory location 0x01 has the number 15.

Testbench for TB_jal file:
Hex instructions:
008000ef
00110113
000101b3

Machine code:
000000001000 00000000 00001 1101111
000000000001 00010 000 00010 0010011
0000000 00000 00010 000 00011 0110011

RISC-V assembly:
jal x1, 8
addi x2, x2, 1
add x3, x2, x0

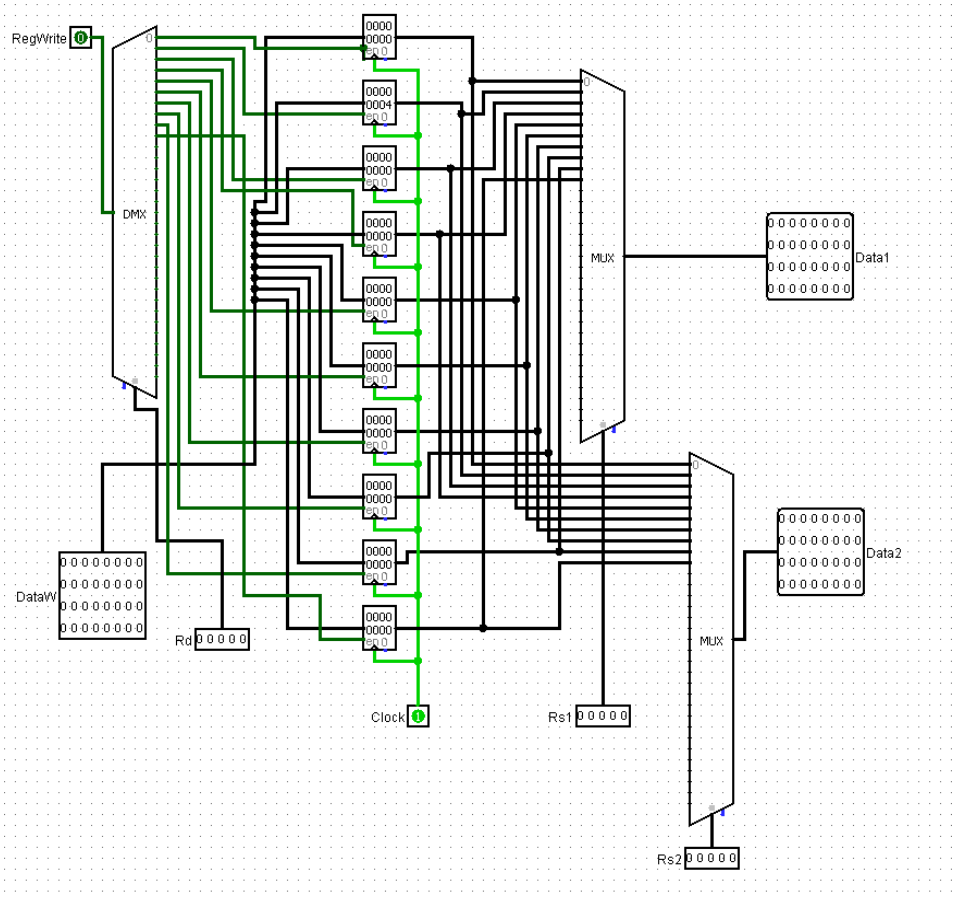

Figure 18: After running the testbench, register x1 has the return address. Register x2 is 0 because it was skipped by the jal instruction.

Testbench for TB_beq file:
Hex instructions:
00000463
00108093
00110113

Machine code:
0000000 00000 00000 000 01000 1100011
000000000001 00001 000 00001 0010011
000000000001 00010 000 00010 0010011

RISC-V assembly:
beq x0, x0, 8
addi x1, x1, 1
addi x2, x2, 1

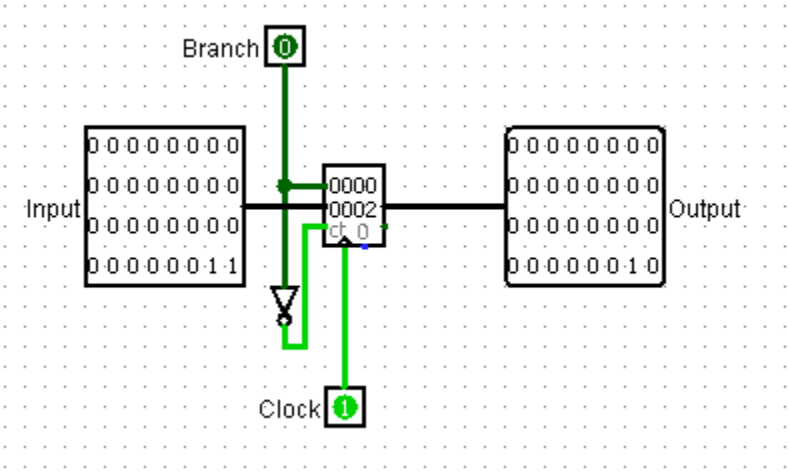

Figure 19: While running the TB_beq testbench, the program counter updates to 0x02 after the first instruction and jumps to the third instruction. Logisim uses RAM and registers that increment by 1 instead of by 4, so the testbench works as intended.

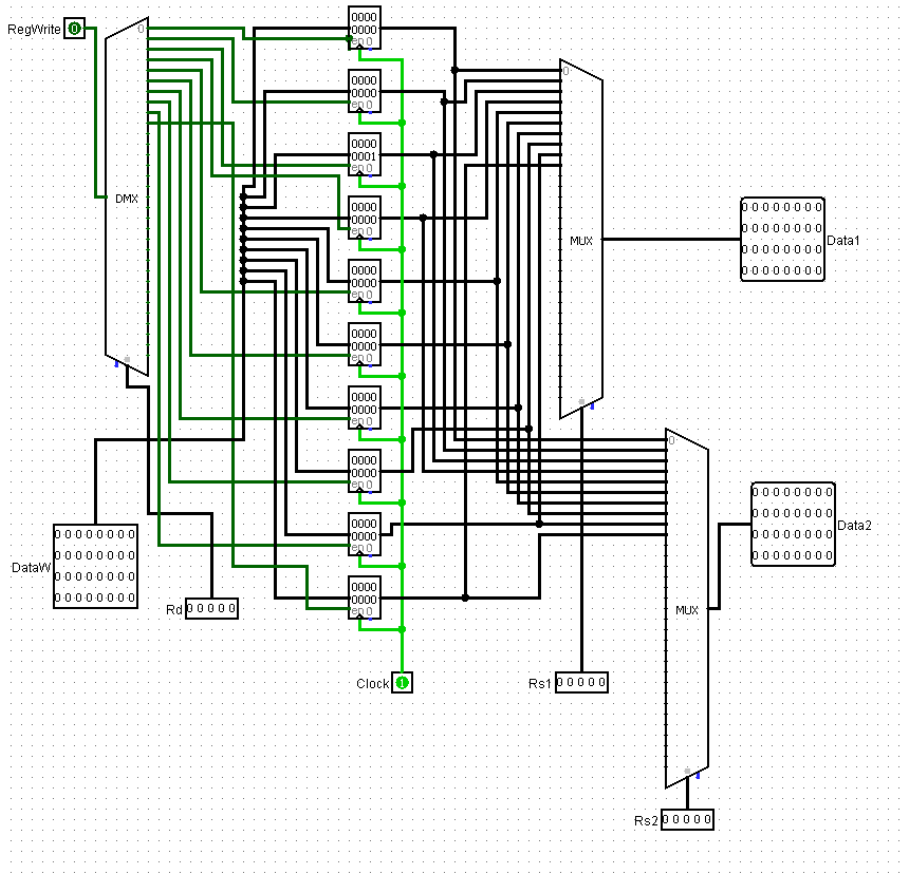

Figure 20: After running the TB_beq testbench, register x2 is 1. Register x1 is 0 because it was skipped by the beq instruction.

Testbench Program Results

After loading the machine code for the testbench program in Table 1, the circuit will run the program and give the expected results in the data memory module.

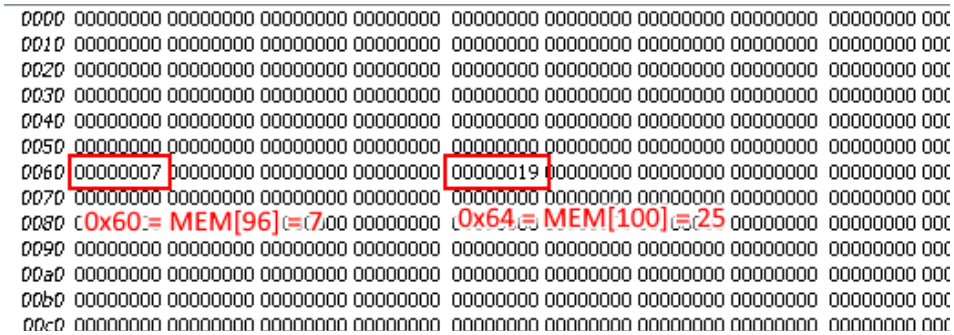

Figure 21: Hexadecimal contents of the data memory module after running the testbench program. Since logisim uses RAM addresses that increment by 1 instead of by 4, these two values are not directly next to each other in memory.