

Lam Ngo  
Final Project  
Testing

*\*Note: Since there are a lot of screen shots, and this has gotten too long. You should go into my codes to see the implementation of my test. I will try to keep my screen shot labels as clear as possible. Also all these tests are done with a carry look ahead adder, so that makes sure my carry look ahead adder works (I also use it to increment PC).*

- I. Test Basic R Type Instructions (since these are straight forward, I am only showing the results):
  1. Professor Rieffel's Test: *this program should put the values 0..7 into registers 0..7*

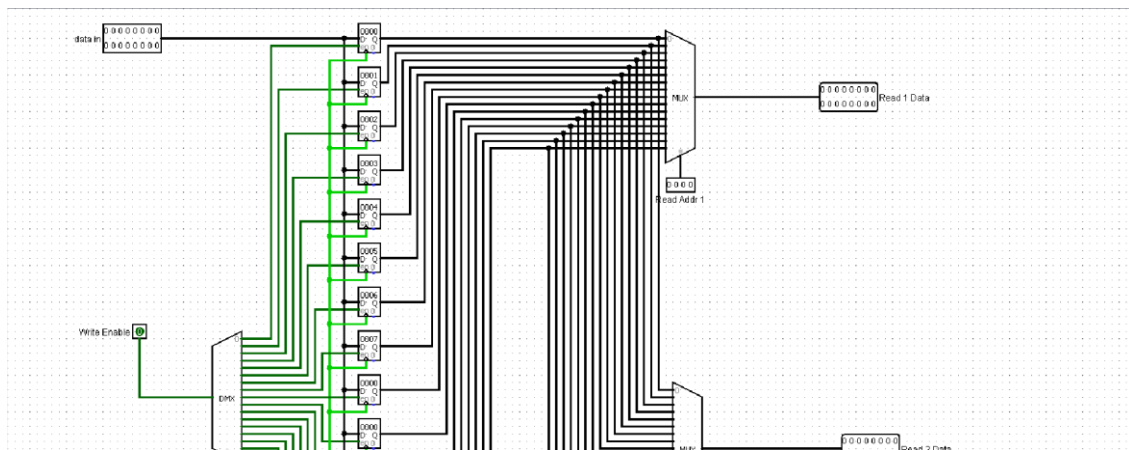


Figure 1: Basic R instructions test code: putting the values 0-7 into registers 0-7.

2. Lam Ngo Test:
  - # \$2 should contain: 5
  - # \$3 should contain: 3
  - # \$4 should contain: 7
  - # \$5 should contain: 2

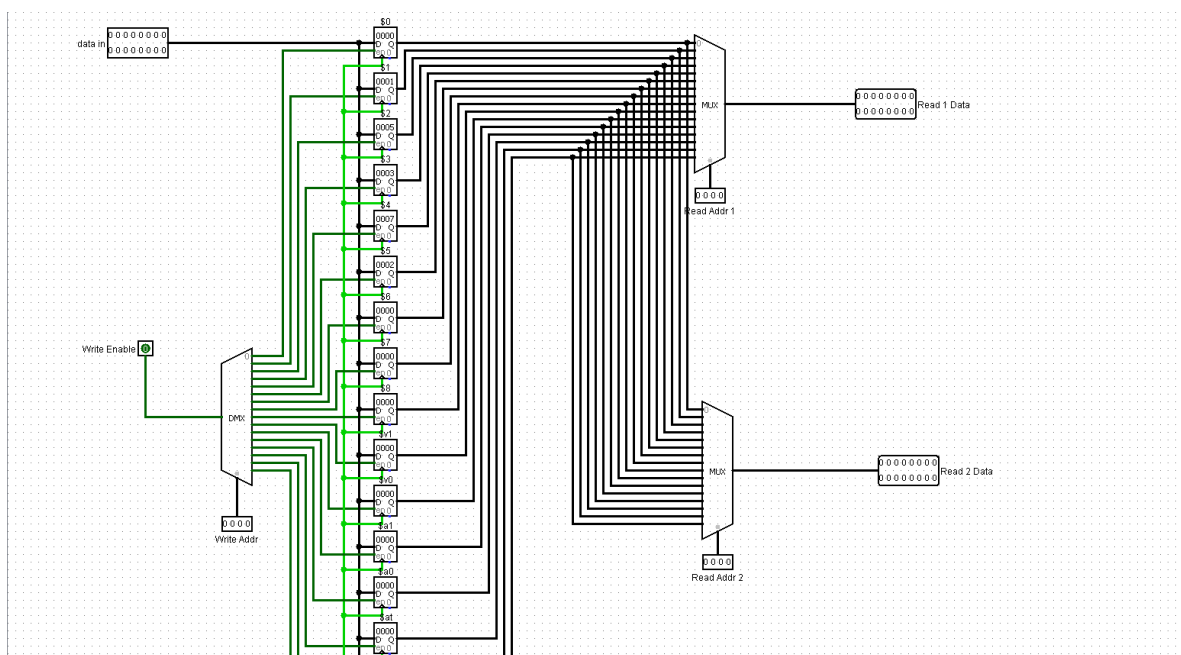


Figure 2: Lam Ngo's test: Adding and subbing values into registers.

- II. Test basic I-type Instructions (since these are straight forward, I am only showing the results):
1. Professor Rieffel's test: *values 0..7 should be in memory locations 0. Values 7..0 should be in registers 0 ... 7(switched)*

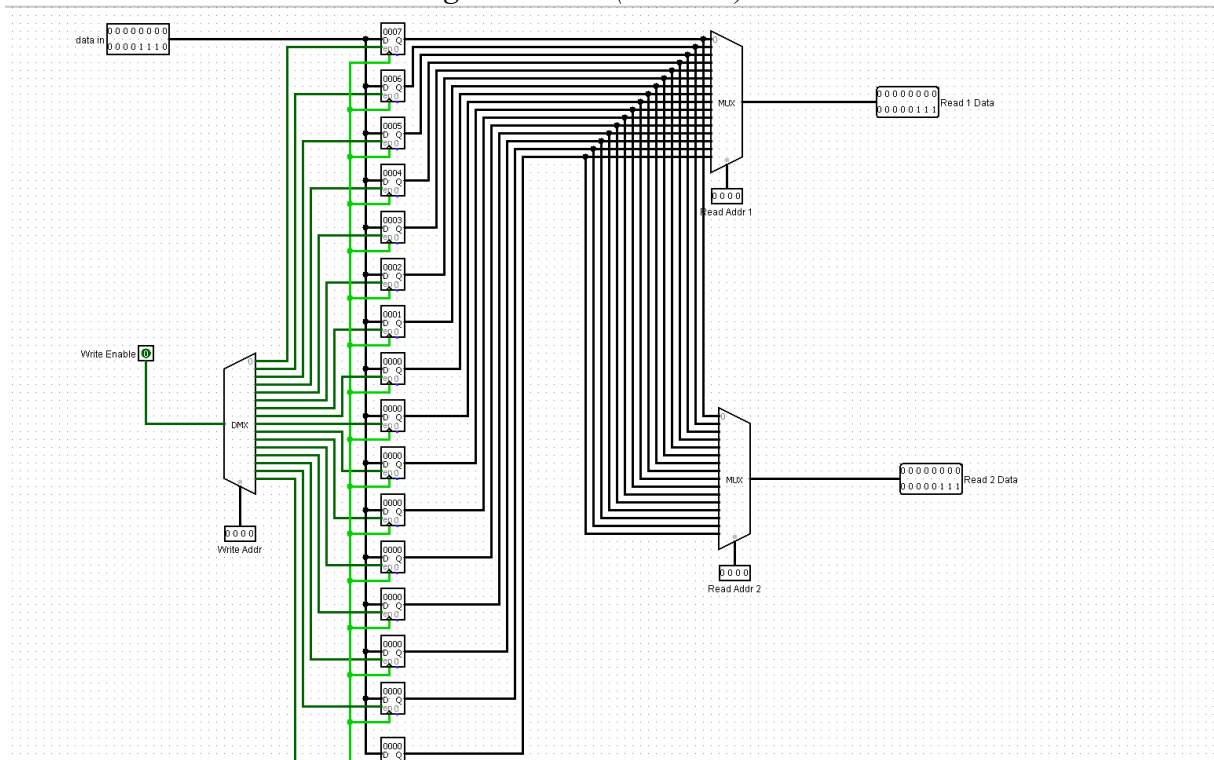


Figure 3: Values from 7 to 0 are in registers from 0 to 7.

The screenshot shows the Logisim Hex Editor window. The menu bar includes File, Edit, Project, Simulate, Window, and Help. The main display area shows a hex dump of memory. The first 16 rows (addresses 0000 to 000F) show the values 0000, 0001, 0002, 0003, 0004, 0005, 0006, 0007, followed by 0000 for the remaining addresses. The rest of the memory (addresses 0010 to 0100) is filled with 0000.

Address	Hex Value
0000	0000
0001	0001
0002	0002
0003	0003
0004	0004
0005	0005
0006	0006
0007	0007
0008	0000
0009	0000
000A	0000
000B	0000
000C	0000
000D	0000
000E	0000
000F	0000
0010	0000
0011	0000
0012	0000
0013	0000
0014	0000
0015	0000
0016	0000
0017	0000
0018	0000
0019	0000
001A	0000
001B	0000
001C	0000
001D	0000
001E	0000
001F	0000
0020	0000
0021	0000
0022	0000
0023	0000
0024	0000
0025	0000
0026	0000
0027	0000
0028	0000
0029	0000
002A	0000
002B	0000
002C	0000
002D	0000
002E	0000
002F	0000
0030	0000
0031	0000
0032	0000
0033	0000
0034	0000
0035	0000
0036	0000
0037	0000
0038	0000
0039	0000
003A	0000
003B	0000
003C	0000
003D	0000
003E	0000
003F	0000
0040	0000
0041	0000
0042	0000
0043	0000
0044	0000
0045	0000
0046	0000
0047	0000
0048	0000
0049	0000
004A	0000
004B	0000
004C	0000
004D	0000
004E	0000
004F	0000
0050	0000
0051	0000
0052	0000
0053	0000
0054	0000
0055	0000
0056	0000
0057	0000
0058	0000
0059	0000
005A	0000
005B	0000
005C	0000
005D	0000
005E	0000
005F	0000
0060	0000
0061	0000
0062	0000
0063	0000
0064	0000
0065	0000
0066	0000
0067	0000
0068	0000
0069	0000
006A	0000
006B	0000
006C	0000
006D	0000
006E	0000
006F	0000
0070	0000
0071	0000
0072	0000
0073	0000
0074	0000
0075	0000
0076	0000
0077	0000
0078	0000
0079	0000
007A	0000
007B	0000
007C	0000
007D	0000
007E	0000
007F	0000
0080	0000
0081	0000
0082	0000
0083	0000
0084	0000
0085	0000
0086	0000
0087	0000
0088	0000
0089	0000
008A	0000
008B	0000
008C	0000
008D	0000
008E	0000
008F	0000
0090	0000
0091	0000
0092	0000
0093	0000
0094	0000
0095	0000
0096	0000
0097	0000
0098	0000
0099	0000
009A	0000
009B	0000
009C	0000
009D	0000
009E	0000
009F	0000
00A0	0000
00A1	0000
00A2	0000
00A3	0000
00A4	0000
00A5	0000
00A6	0000
00A7	0000
00A8	0000
00A9	0000
00AA	0000
00AB	0000
00AC	0000
00AD	0000
00AE	0000
00AF	0000
00B0	0000
00B1	0000
00B2	0000
00B3	0000
00B4	0000
00B5	0000
00B6	0000
00B7	0000
00B8	0000
00B9	0000
00BA	0000
00BB	0000
00BC	0000
00BD	0000
00BE	0000
00BF	0000
00C0	0000
00C1	0000
00C2	0000
00C3	0000
00C4	0000
00C5	0000
00C6	0000
00C7	0000
00C8	0000
00C9	0000
00CA	0000
00CB	0000
00CC	0000
00CD	0000
00CE	0000
00CF	0000
00D0	0000
00D1	0000
00D2	0000
00D3	0000
00D4	0000
00D5	0000
00D6	0000
00D7	0000
00D8	0000
00D9	0000
00DA	0000
00DB	0000
00DC	0000
00DD	0000
00DE	0000
00DF	0000
00E0	0000
00E1	0000
00E2	0000
00E3	0000
00E4	0000
00E5	0000
00E6	0000
00E7	0000
00E8	0000
00E9	0000
00EA	0000
00EB	0000
00EC	0000
00ED	0000
00EE	0000
00EF	0000
00F0	0000
00F1	0000
00F2	0000
00F3	0000
00F4	0000
00F5	0000
00F6	0000
00F7	0000
00F8	0000
00F9	0000
00FA	0000
00FB	0000
00FC	0000
00FD	0000
00FE	0000
00FF	0000
0100	0000

Figure 4: Values from 0 to 7 are stored in memory location from 0 to 7.

## 2. Lam Ngo's Test:

- a. First test: test subtract a negative number. Results should be the same as Professor Rieffel' test. (Instead of addi \$1 \$0 1, it is subi \$1 \$0 -1, I'm testing subtract at the same time):

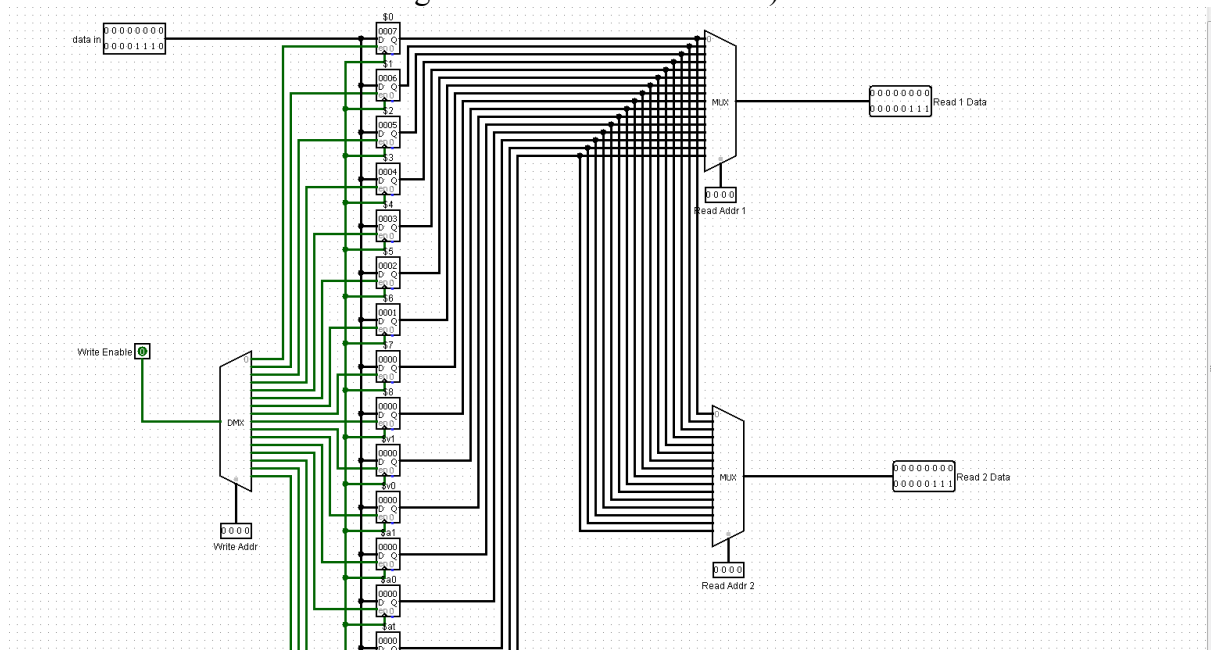


Figure 5: Values from 7 to 0 are in registers from 0 to 7.

Logisim: Hex Editor															
File Edit Project Simulate Window Help															
0000	0000	0001	0002	0003	0004	0005	0006	0007	0000	0000	0000	0000	0000	0000	0000
0010	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0040	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0050	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0070	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0080	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0090	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00a0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00b0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00c0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00d0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00e0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00f0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0110	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0120	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0130	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0140	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0150	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0160	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0170	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0180	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 6: Values from 0 to 7 are stored in memory location from 0 to 7.

- b. Lam Ngo Test 2: Same as first two test, but reverse the order of the array in memory, values 7..0 are stored from memory location 0 to 7(Adapted from Lab 4):

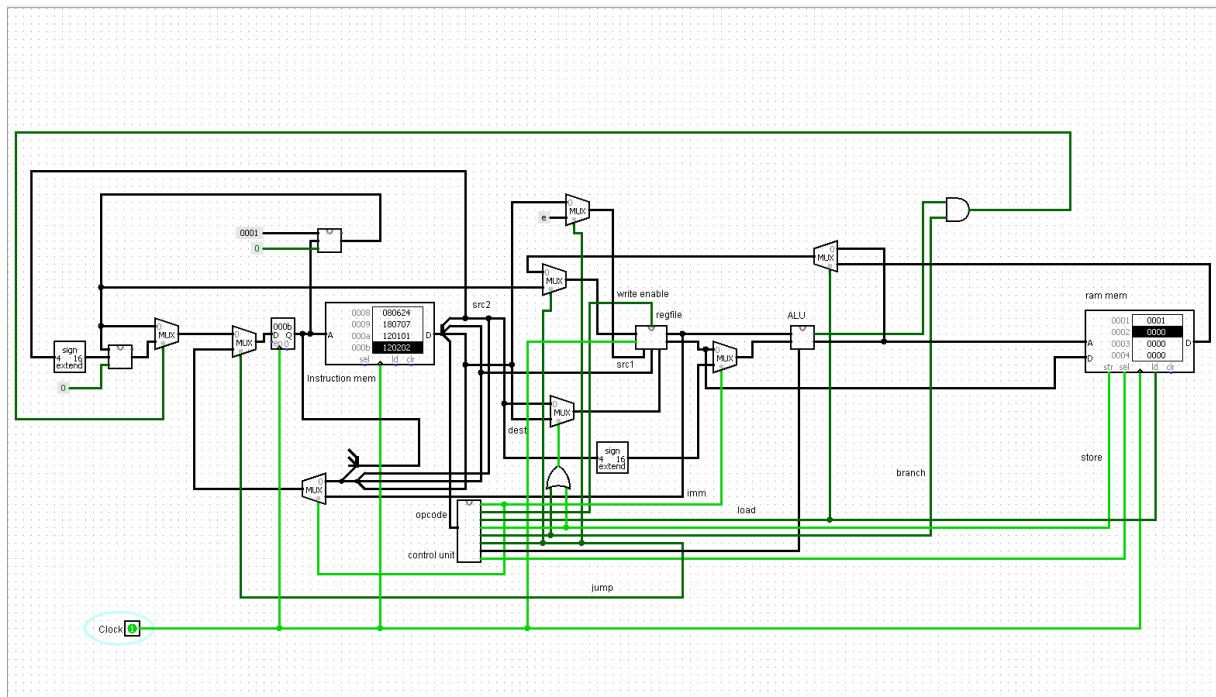


Figure 7: Load Word Control Signal

File	Edit	Project	Simulate	Window	Help
0000	0000	0007	0006	0005	0004 0003 0002 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0010	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0020	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0030	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0040	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0050	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0060	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0070	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0080	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0090	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00a0	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00b0	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00c0	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00d0	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00e0	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00f0	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0100	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0110	0000	0000	0000	0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Figure 8: Values 7...0 are reversed and stored from 1 to 7.

### III. Test Branch If Equal:

#### 1. Professor Rieffel Test:

- Test Positive branch offset: results: \$4 should contain 4 and \$5 should contain 5.

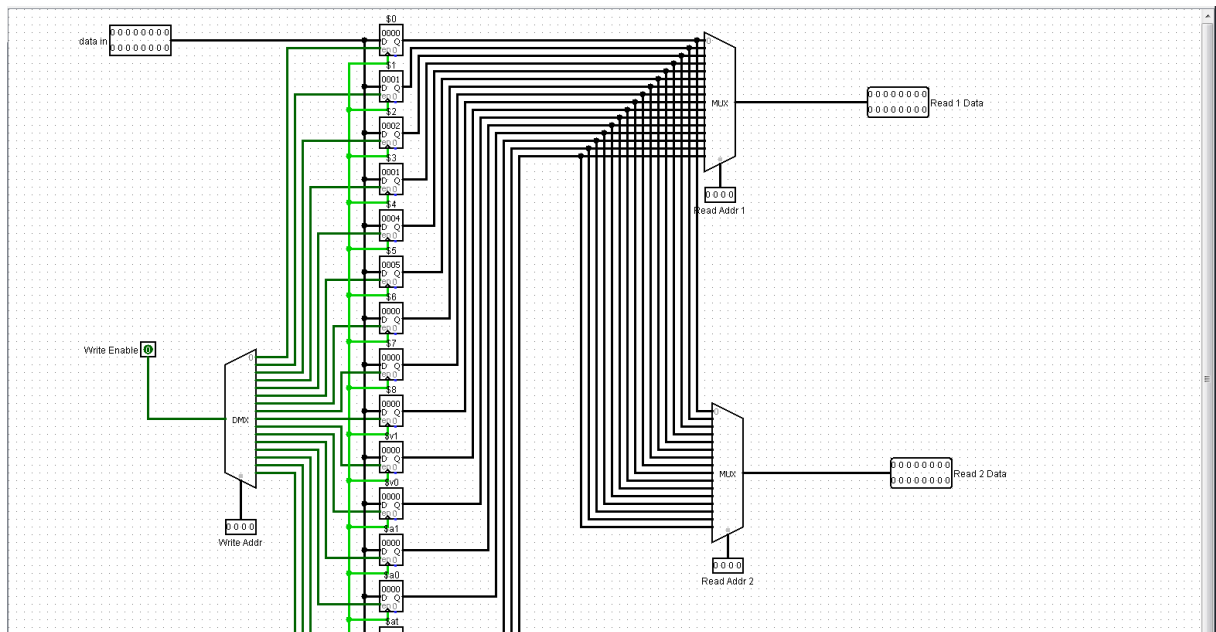


Figure 9: Register 4 contains value 4 and Register 5 contains value 5.

- Test Back Branch: registers from 1 to 5 should all contain 5.

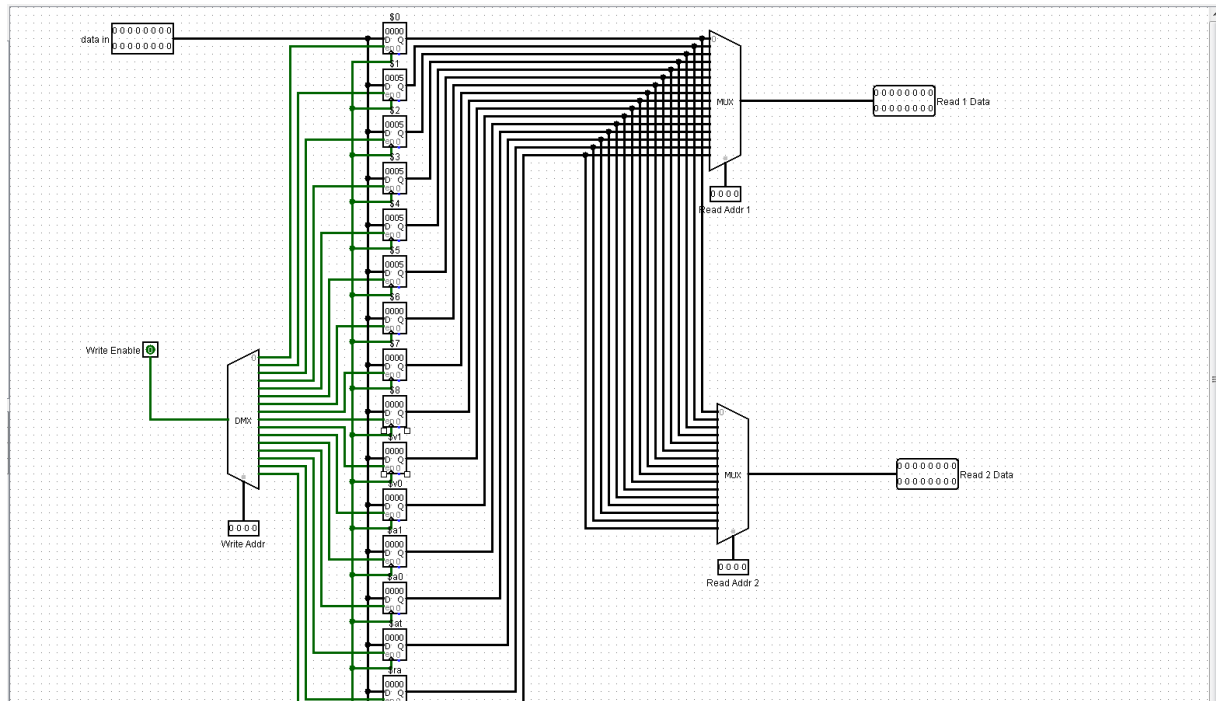


Figure 10: Registers from 1 to 5 all contains 5.

2. Lam Ngo's Test: True if registers 0-4 are 0, and first branch not taken and second branch

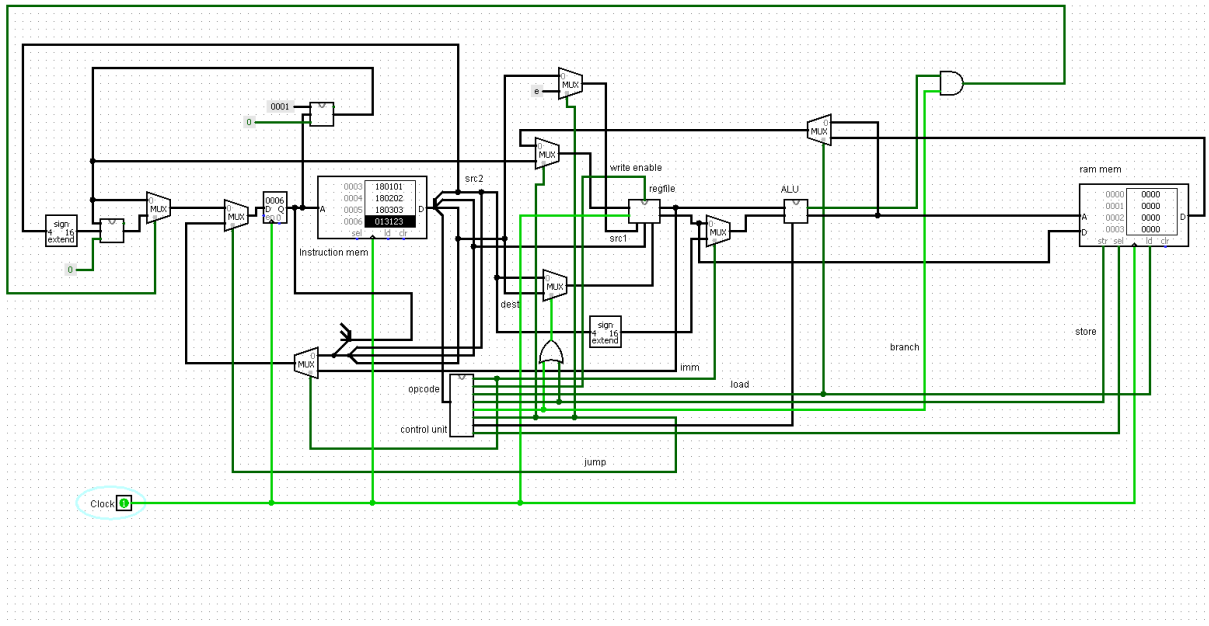


Figure 11: Branch Not Taken Control Signal

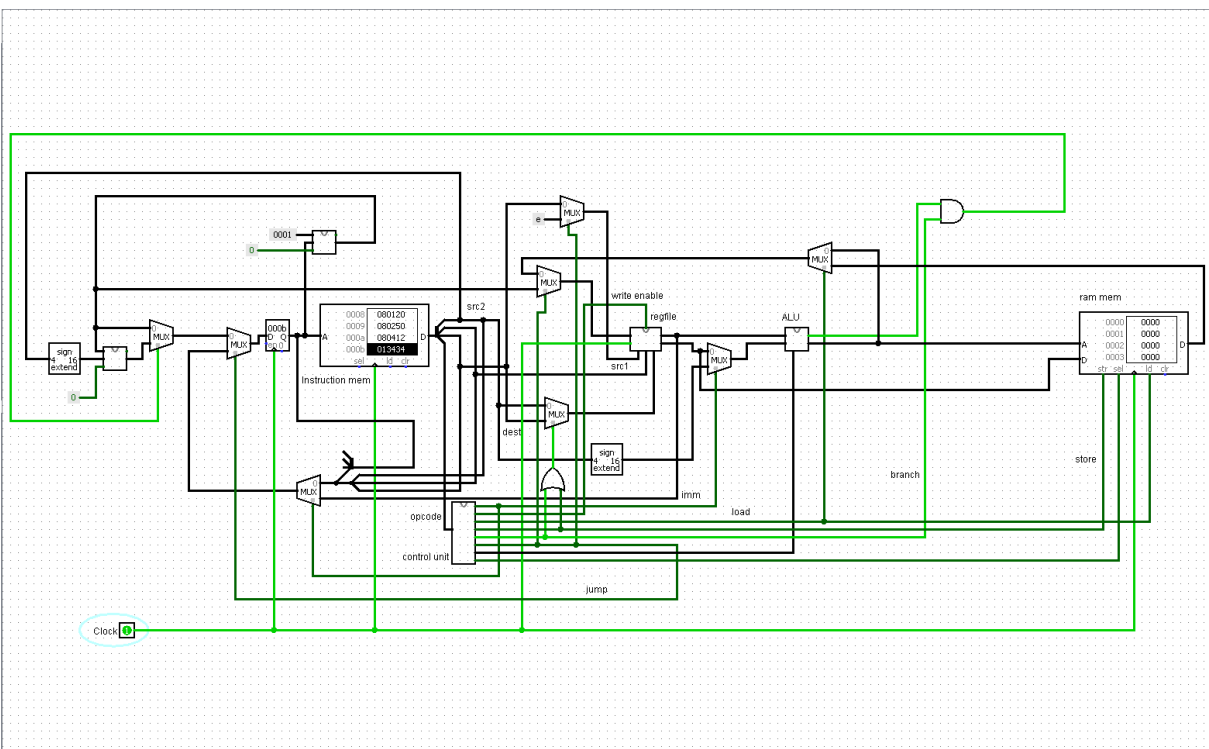


Figure 12: Branch Taken Control Signal.

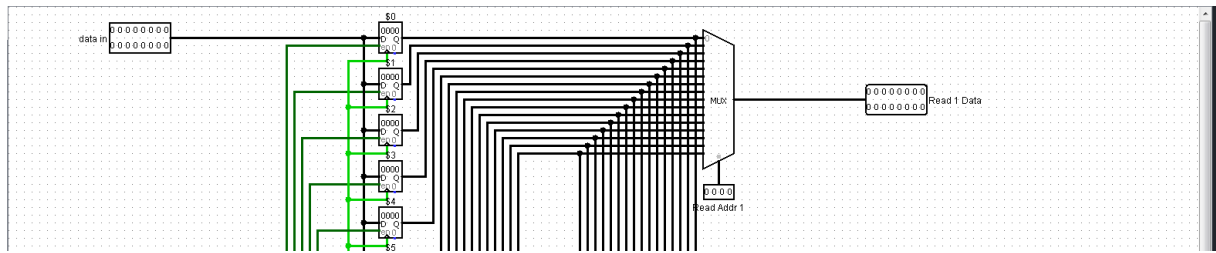


Figure 13: Registers from 0 to 4 all contain 0.

- IV. Test Jump Instructions (Jal and Jr will be tested in the next section):
1. Professor Rieffel's test: \$1 should have the value 3 if this passes test.

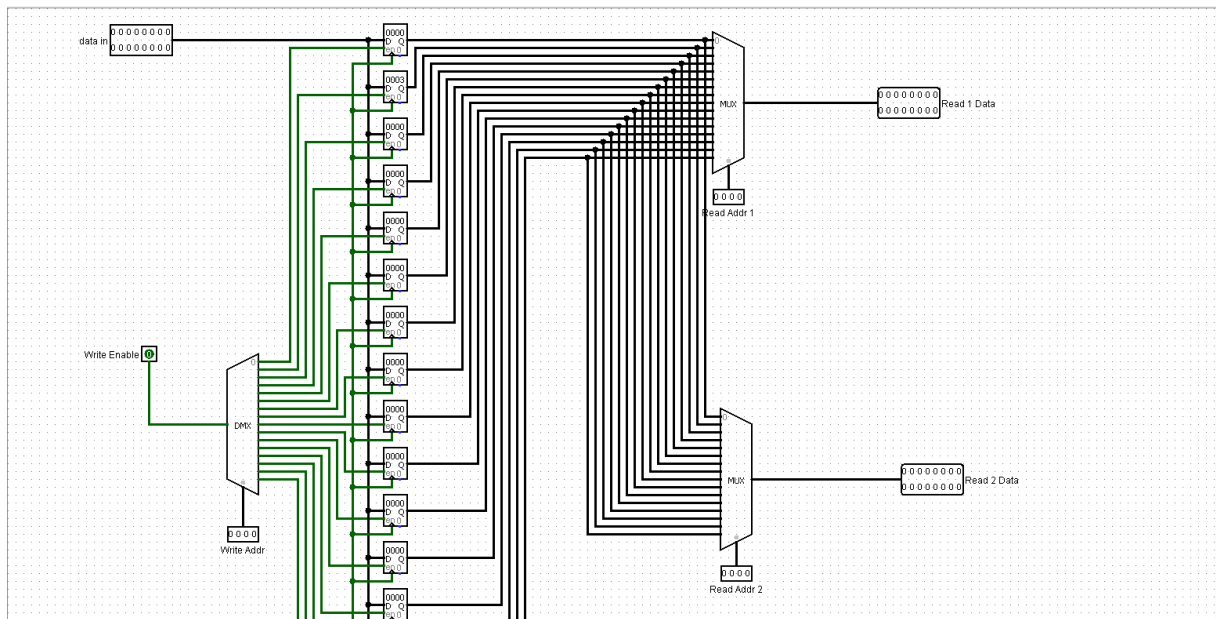


Figure 14: Register 1 has the value 3 after finishing the test.

2. Lam Ngo's test: \$1 should have the value 4 if this passes test.

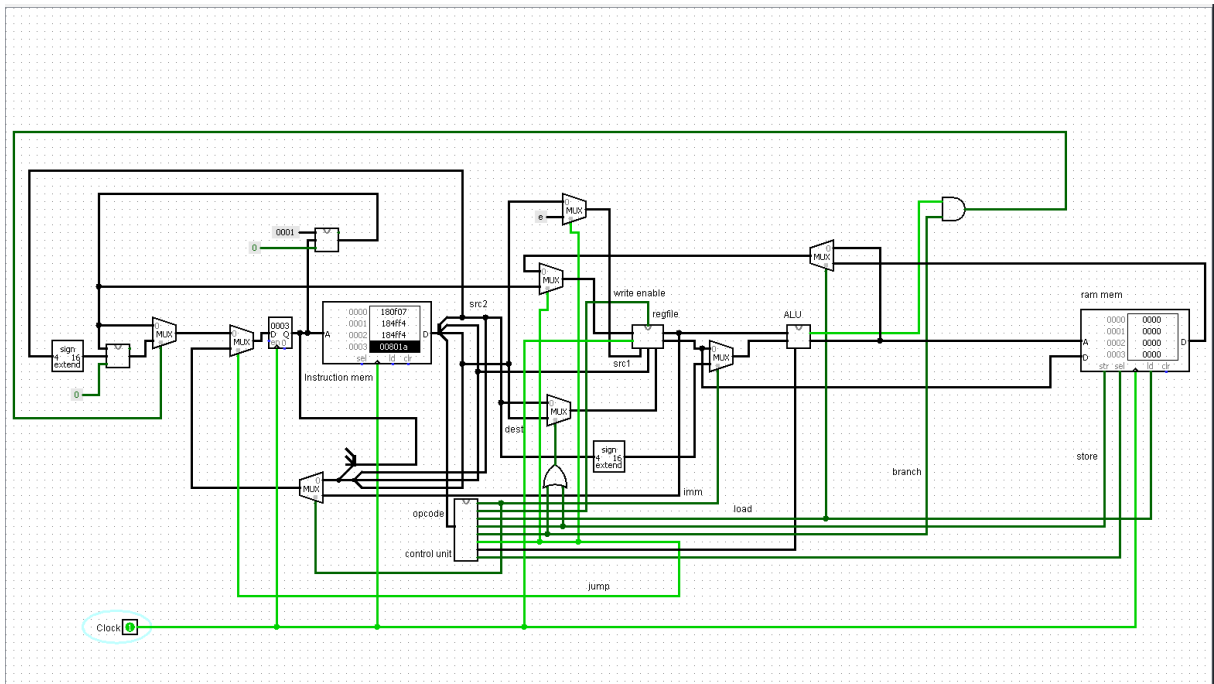


Figure 15: Jump Instruction control signal.

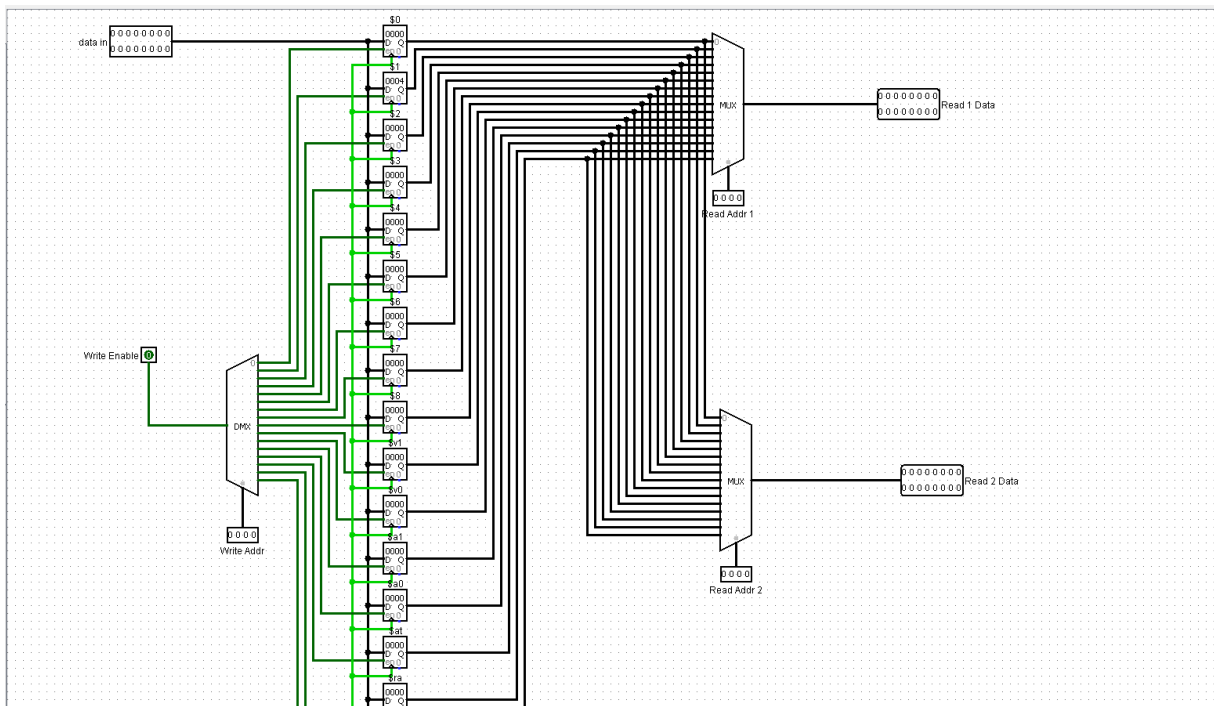


Figure 16: \$1 contains the value 4.



## V. Test Function calls and Jump And Link and Jump Return:

1. First test: Call 1 function min, return 1 if the first argument is smaller than the second argument. 1 should be stored at address 1.

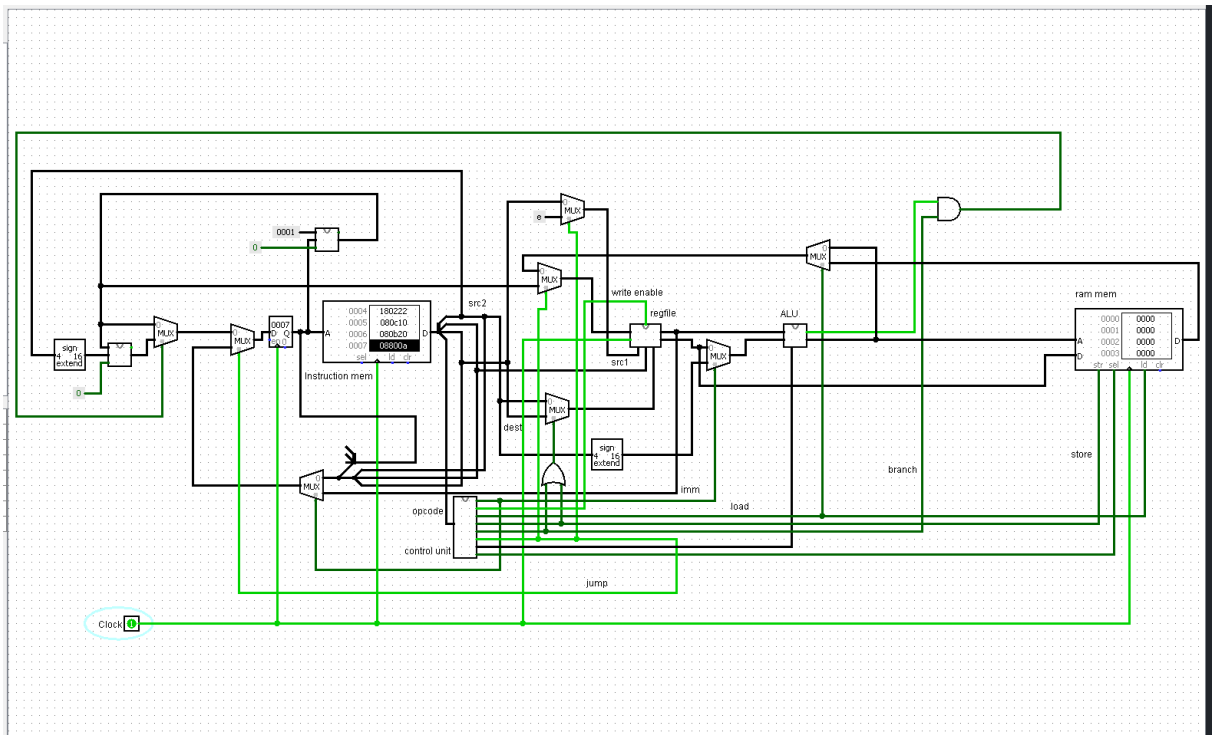


Figure 17: Jump And Link Control Signal (should return to line 8).

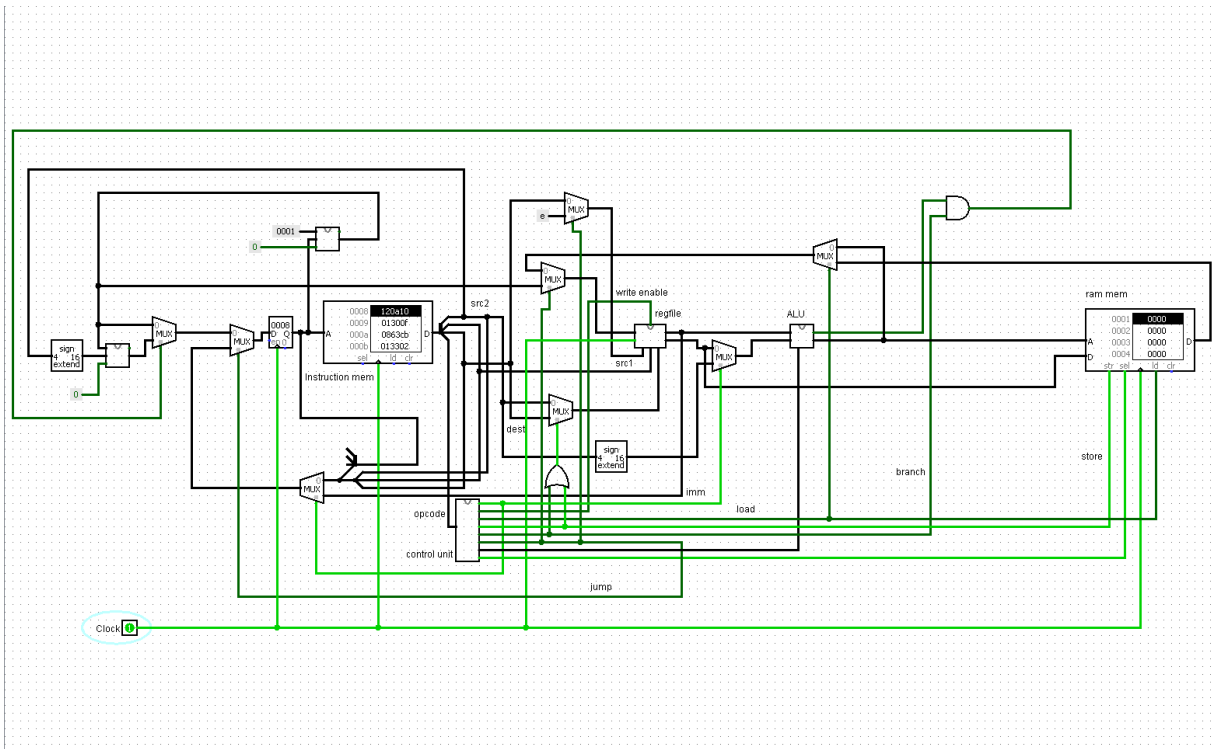


Figure 18: After doing jr \$ra, PC now has returned to line 8.

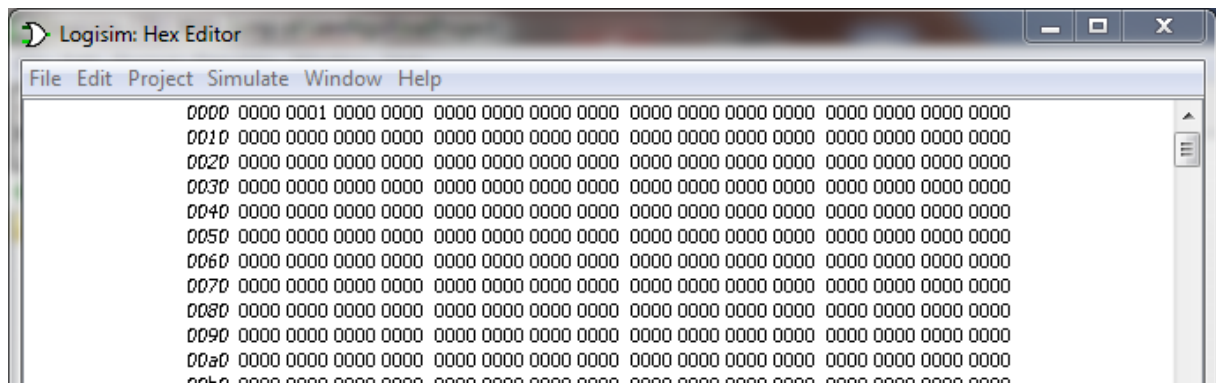


Figure 19: Value 1 is stored at address 1.

2. Second Test: Calls the function Add to add two arguments, then calls function Double to double the result. Should store the value 6 at memory location 1.

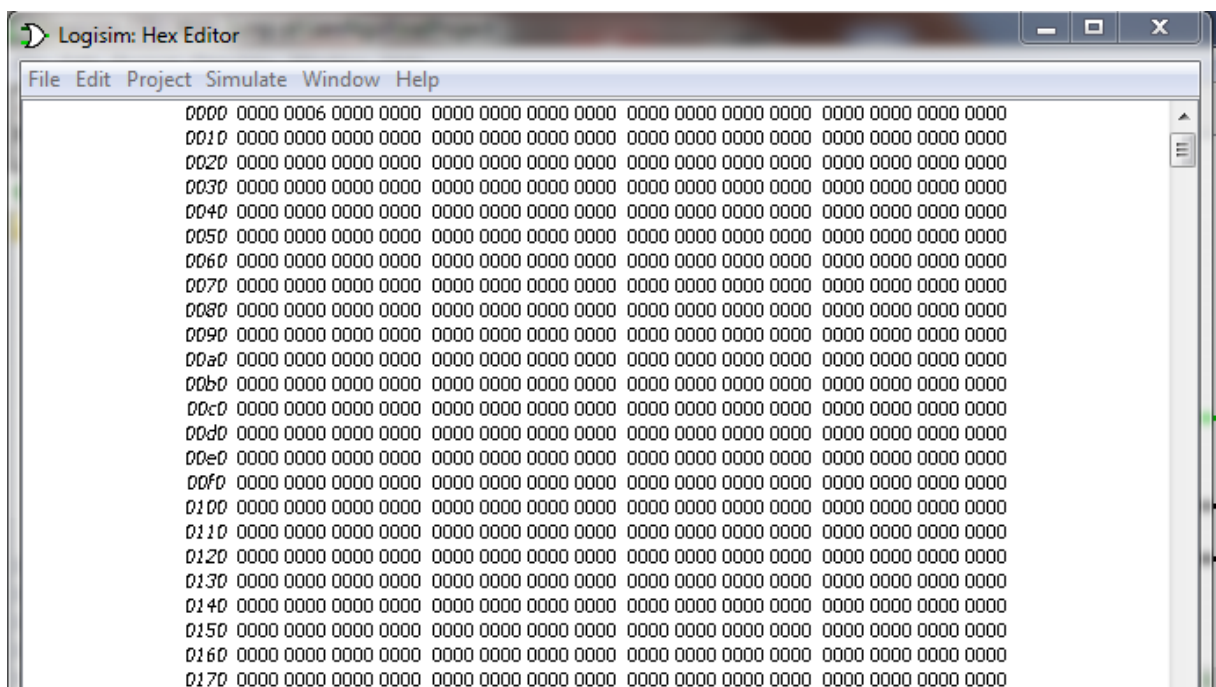


Figure 20: After calling the two functions, value 6 is stored at address 1.

- VI. Test Function calls with Stacks (nested function calls and recursion):
  1. Professor Riffel's test: Recursion: this program should store values 0...5 into registers 0...5 then recursively add them together, storing the result in address 1.

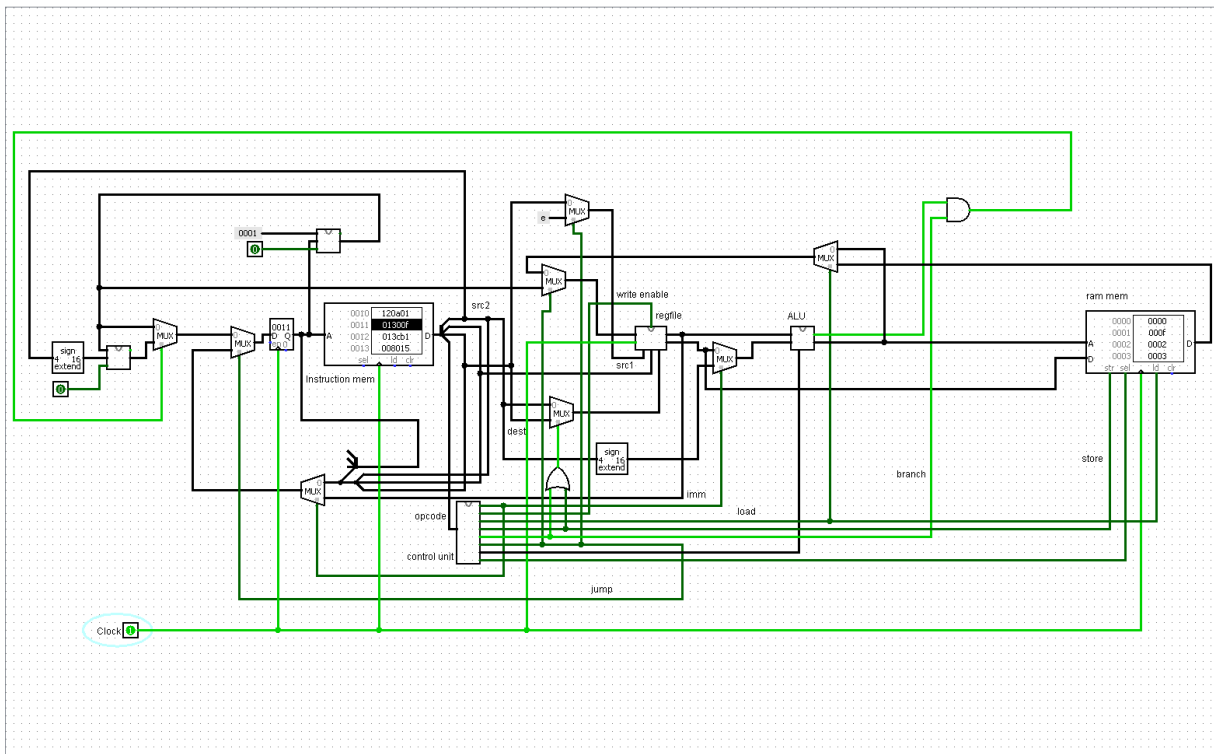


Figure 21: The CPU in an infinite loop at the end of the program (was stated in the program).  
Can see the correct results loaded in address 1.

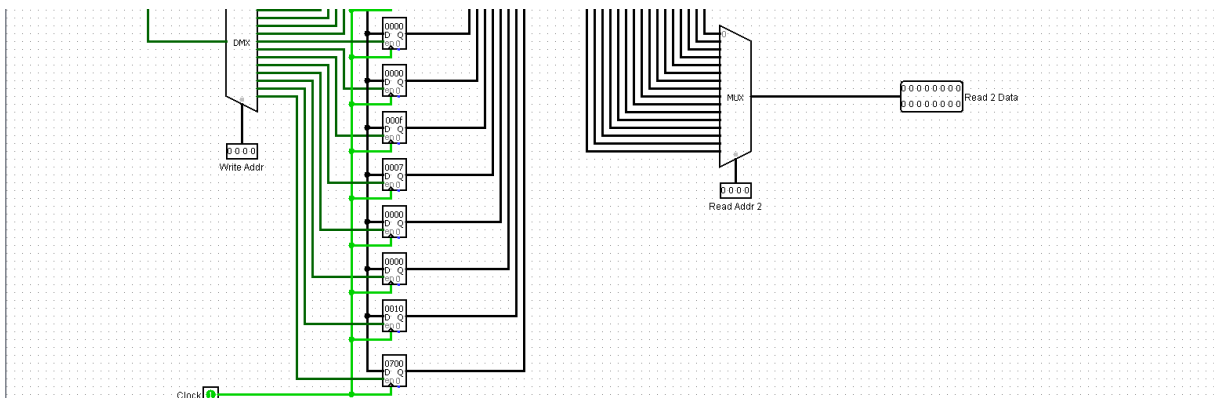


Figure 22: Stack Pointer was initialized at the beginning of the program and reset at the end of the program.

```

06d0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
06e0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
06f0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0700 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

```

Figure 23: How the stack was used during the program.

2. Lam Ngo's test: Calling a function Add, and then calls a function Double (which calls another function Triple). The program should store the value 18 at address 1 at the end.

```

06a0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
06b0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
06c0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
06d0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
06e0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
06f0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0700 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0710 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0720 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0730 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0740 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

```

Figure 24: The stack used to save the return address.

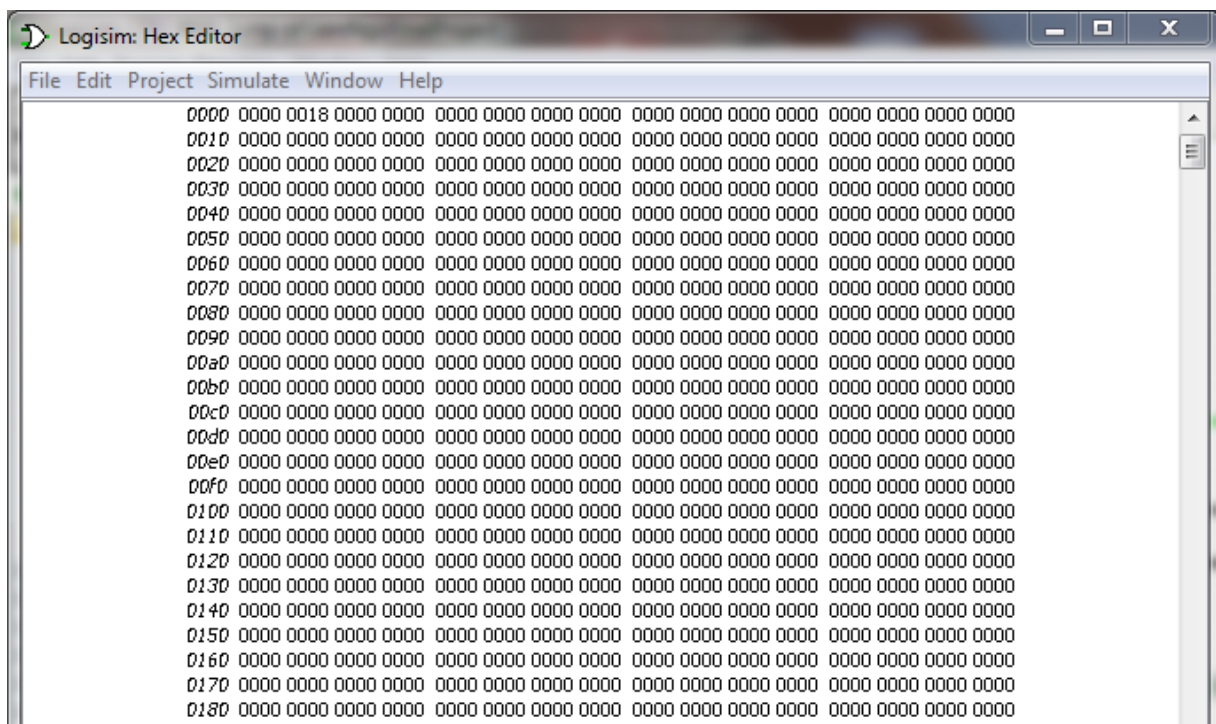


Figure 25: The result 18 at address 1.

## VII. Test Pseudoinstructions:

Test BLT and BGT: BLT should be taken and BGT not taken. \$4 and \$5 should contain 4 and 5 at the end of the program.

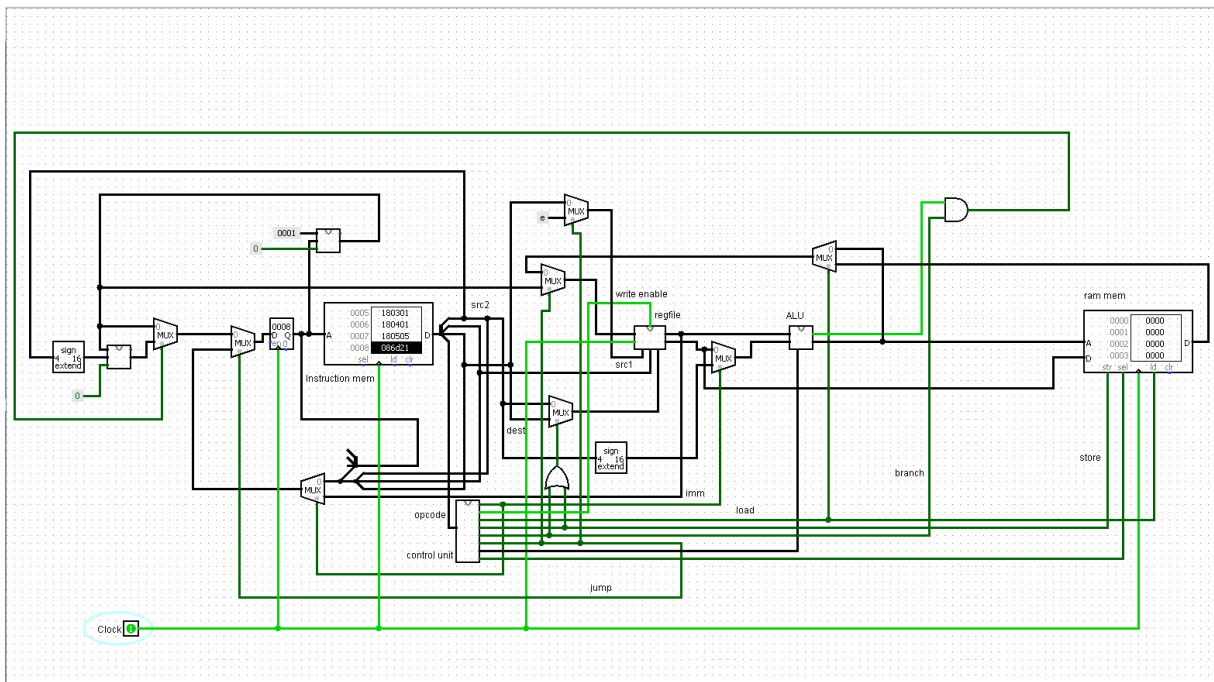


Figure 26: BGT not taken control signal.

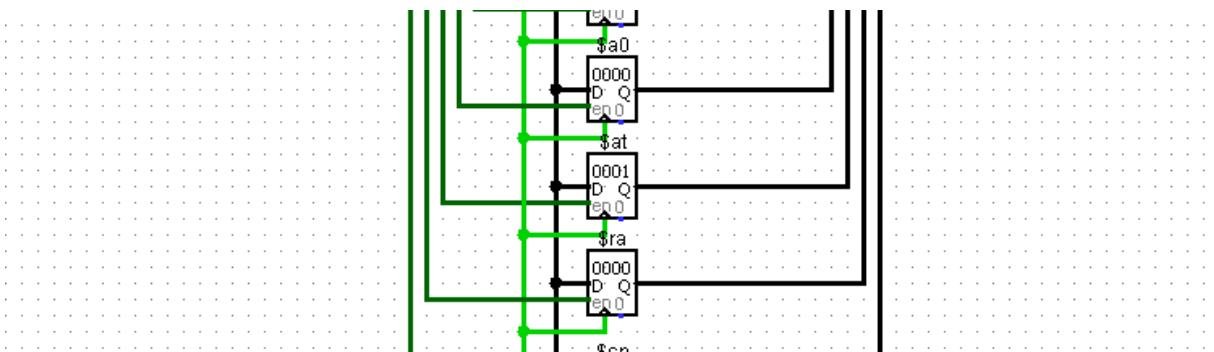


Figure 27: \$at changed to 1 for Blt, so BLT should be taken.

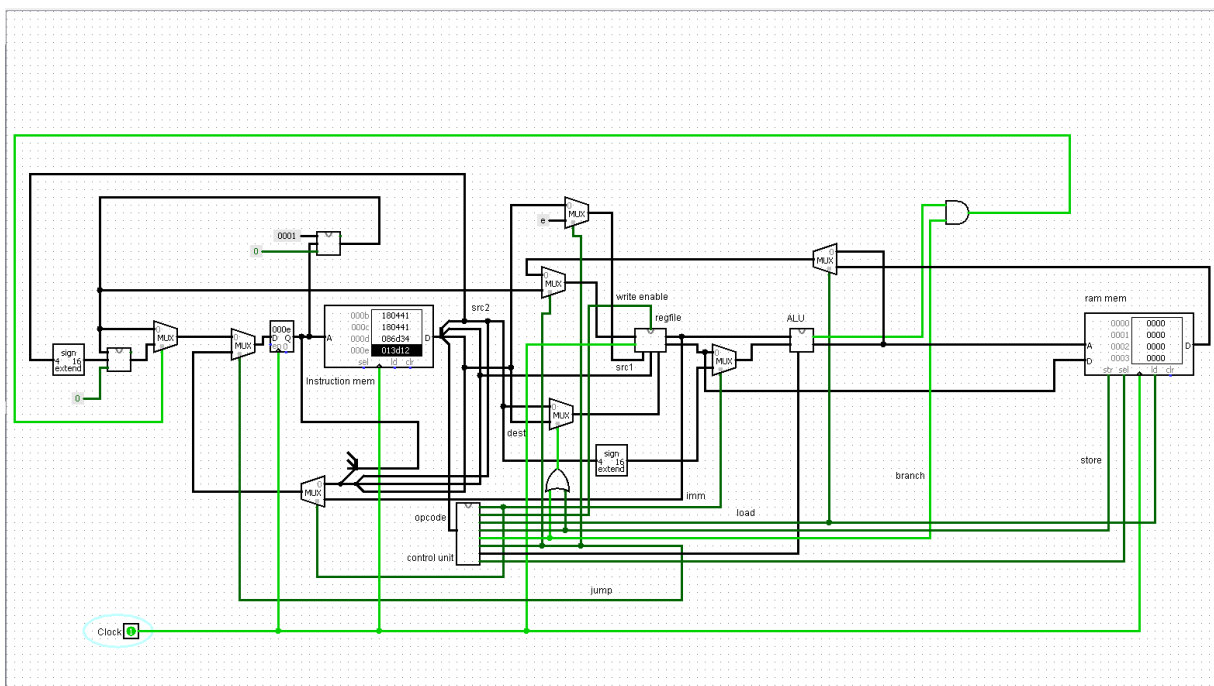


Figure 28: Branch is taken after \$at is changed to 1 for BLT.

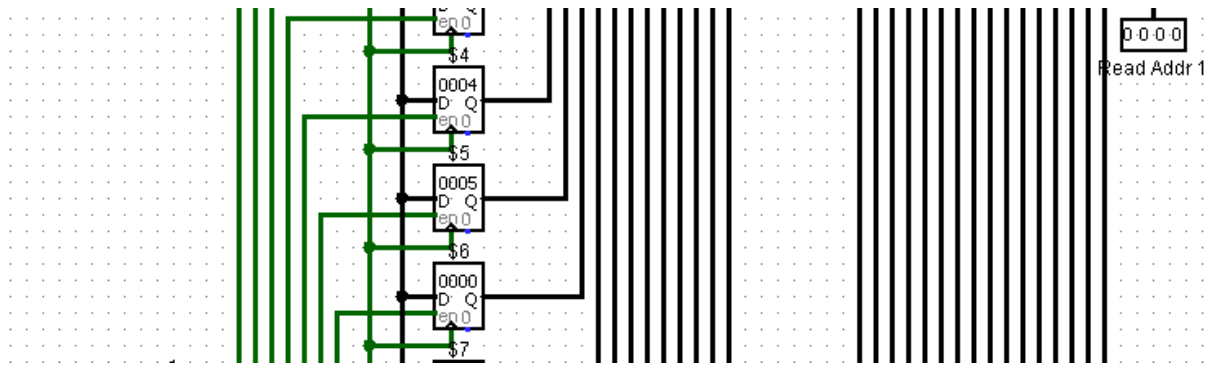


Figure 29: \$4 and \$5 contain 4 and 5.