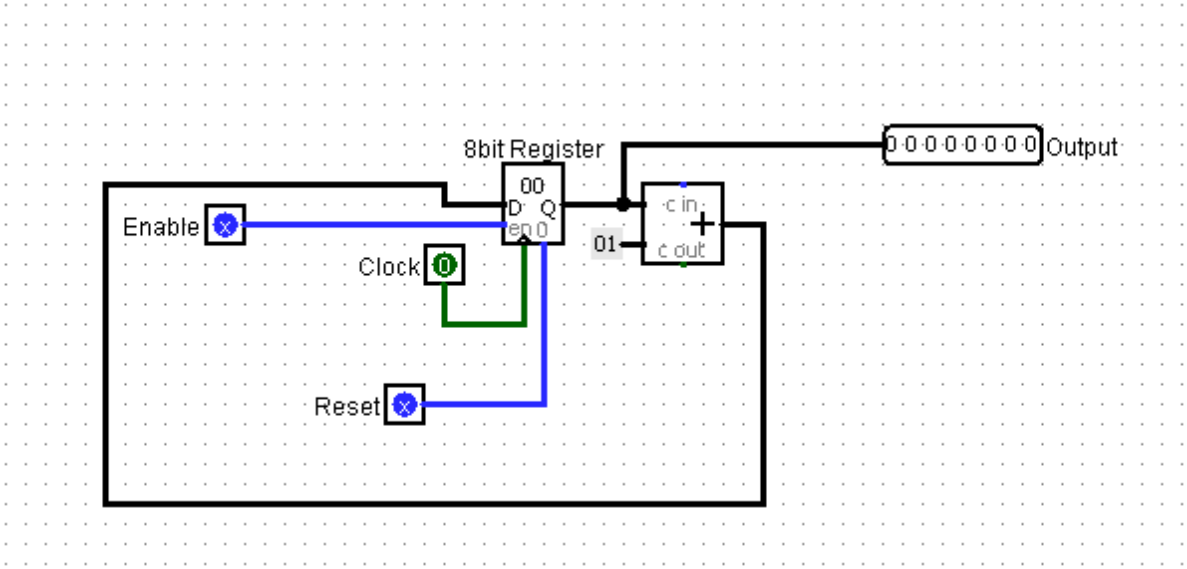
COMP30080 Processor Design

Assignment 5 Fergal Lonergan

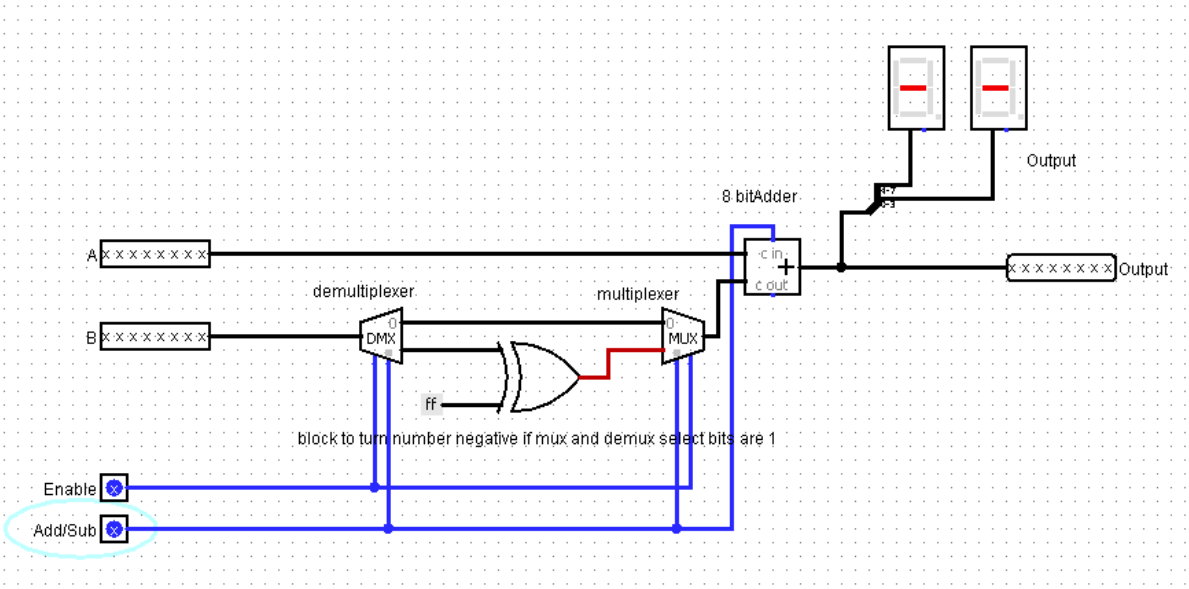
Part 1

PC&ROM



My PC&ROM is the same as last week. I increment through my Program counter and this cycles through the instructions in our instruction memory one by one.

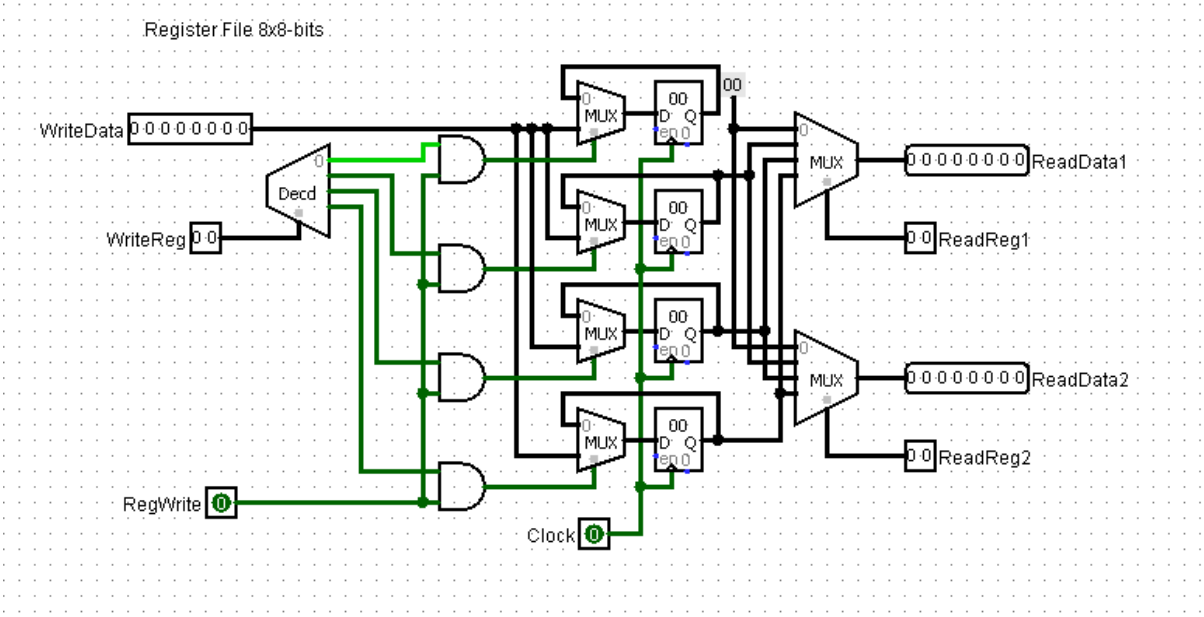
ALU



My ALU is also the same as last week. It adds or subtracts numbers depending on whether the Add/Sub input is 1 or 0. My control unit reads the 3rd MSB from our opcode to see if the function should be to add or subtract. As the first MSB is always 1 my control unit doesn't use it. My 2nd MSB is also only used for subtract, however if it is subtract we want, the conditions for lw or sw will fail

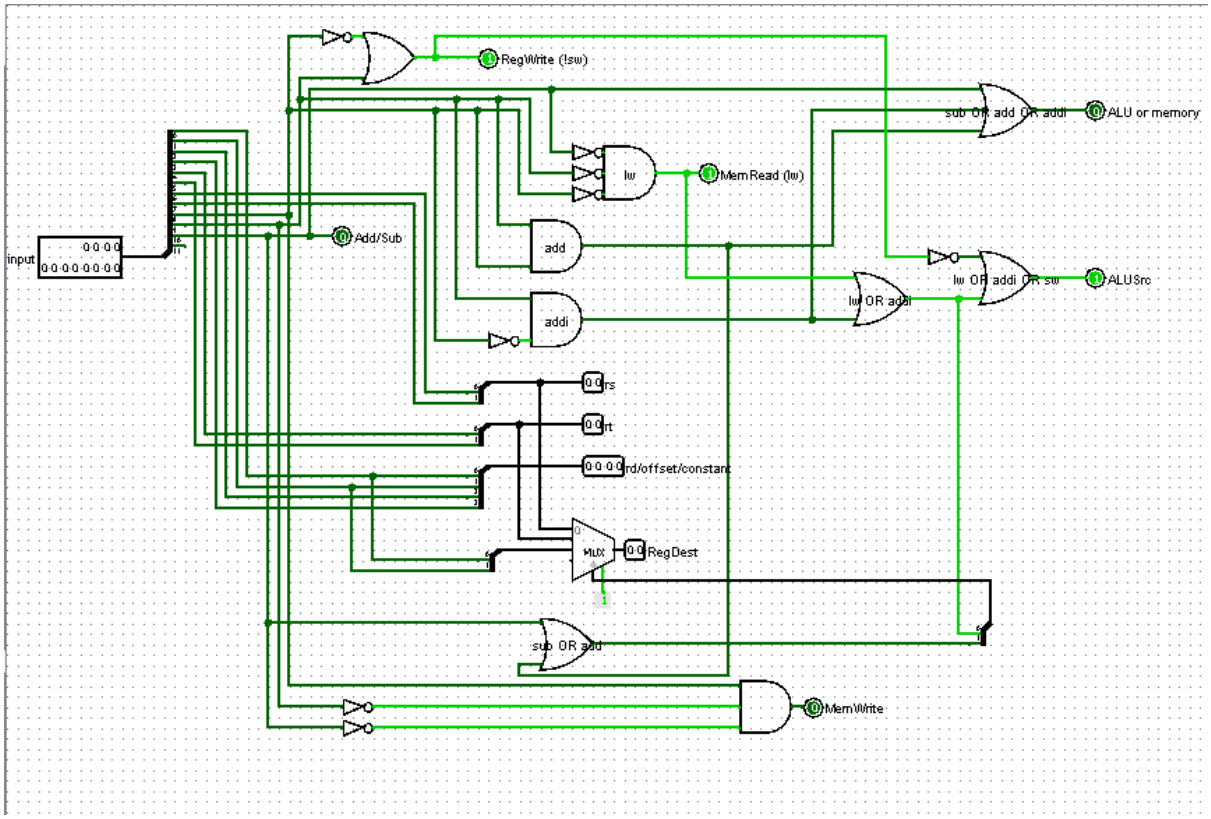
and my control unit will then just read the output from our ALU which will be a subtraction. The mux going into our ALU decides if it will be a (lw,sw) operation or a subtract.

Registers



The register unit I used was the one provided to us on moodle. It reads in an 8bit number and a register number and uses a decoder to see which register it then must write to. You can also read from two registers once you give it their reg codes. ReadReg1 is rs, ReadReg2 is rt and write reg is rd/offset/constant. Whether it read or writes and what constant, offset, rd it uses and which registers it reads to / writes from is all controlled by our control unit.

Control unit



This was the main bulk of our assignment. We had to design a control unit that would read in the instruction code from our instruction memory, decode it, and then perform the desired operations.

My control unit first read in the 12 bit binary number from the instruction memory and divides it up so I can read each bit individually.

The MSB doesn't change so it isn't used in my control unit.

The 4 LSBs (0-3) make up the value for rd/offset/constant. The next 2 LSBs (4-5) are rt. Rs is bits 6-7 and you can calculate the operation using bits 8 and 9.

Rt, rs and rd go into a multiplexer as their positions can change from formula to formula, its select bits are the answer to whether the operation is sub or and. We then know from this which position each should be in from our codes given below.

We see that if all three significant bits are 0 then we use load word, lw.

If only the LSB of the opcode is 1 then we know its sw, so we set MemWrite to 1.

If the 2nd MSB is 1 then it is subtraction.

We know than when the 2nd LSB of the opcode is 1 then it is one of the add operations and using simple gate logic to see if the LSB is also 1 we can decipher whether it is addi (10) or add (11).

Register Set
$zero, $t0, $t1, $t2

Instruction Set Architecture

```
lw   rt, offset(rs)        # load word from M[rs+offset] to R[rt]
sw   rt, offset(rs)        # store word from R[rt] to M[rs+offset]
addi rt, rs, const         # add immediate R[rt] = R[rs] + const
add  rd, rs, rt            # add R[rd]=R[rs]+R[rt]
sub  rd, rs, rt            # add R[rd]=R[rs]-R[rt]
```

Instruction Formats
{opcode; rs; rt; rd/offset/const}
opcode = 4 bits; rs = 2 bits; rt = 2 bits; rd/offset/const = 4 bits

Opcodes

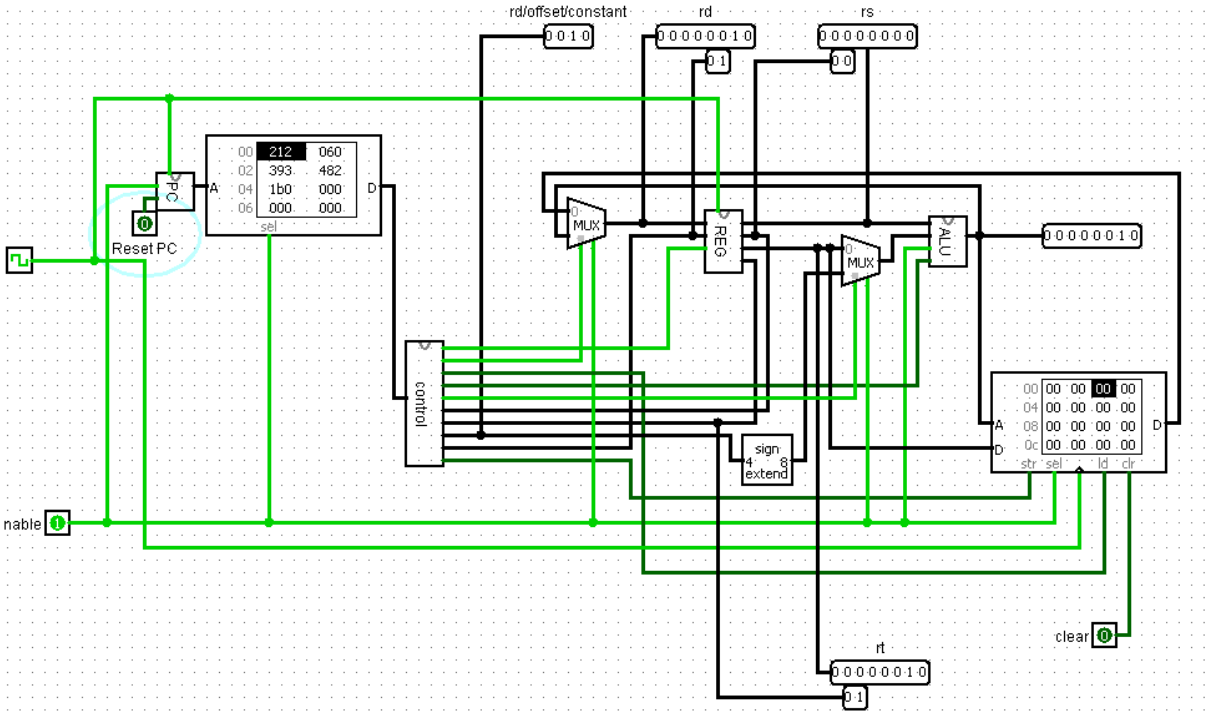
```
lw   : opcode = 0000
sw   : opcode = 0001
addi : opcode = 0010
add  : opcode = 0011
sub  : opcode = 0100
```

To test my design I came up with the following instructions that were to be saved in my ROM.

I made 5 instructions using all five operations given to us in the assignment and saved them into my ROM in the first 5 data slots. They are saved in Hexadecimal format which is why I have converted my values below to hex.

- addi $t0, $zero, 2 — $0010\ 00\ 01\ 0010_2$ — $0x212_{16}$
- lw $t1, 0($t0) , — $0000\ 01\ 10\ 0000_2$ — $0x060_{16}$
- add $t2, $t1, $t0 — $0011\ 10\ 01\ 0011_2$ — $0x393_{16}$
- sub $t1, $t2, $zer0 — $0100\ 11\ 00\ 0010_2$ — $0x4C2_{16}$
- sw $t1,0($t0), — $0001\ 01\ 10\ 0000_2$ — $0x160_{16}$

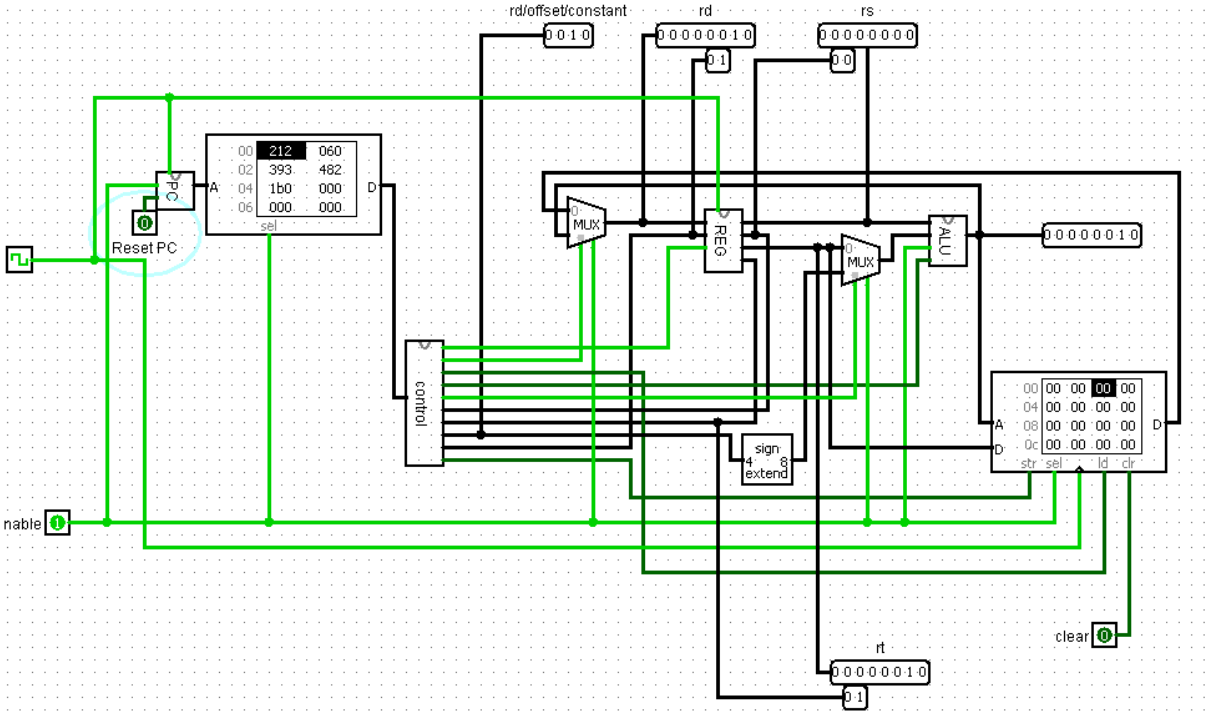
Final design



The following is my final design.

We can see that during the first instruction:

addi $t0, $zero, 2 0010 00 01 0010_2 $0x212_{16}$



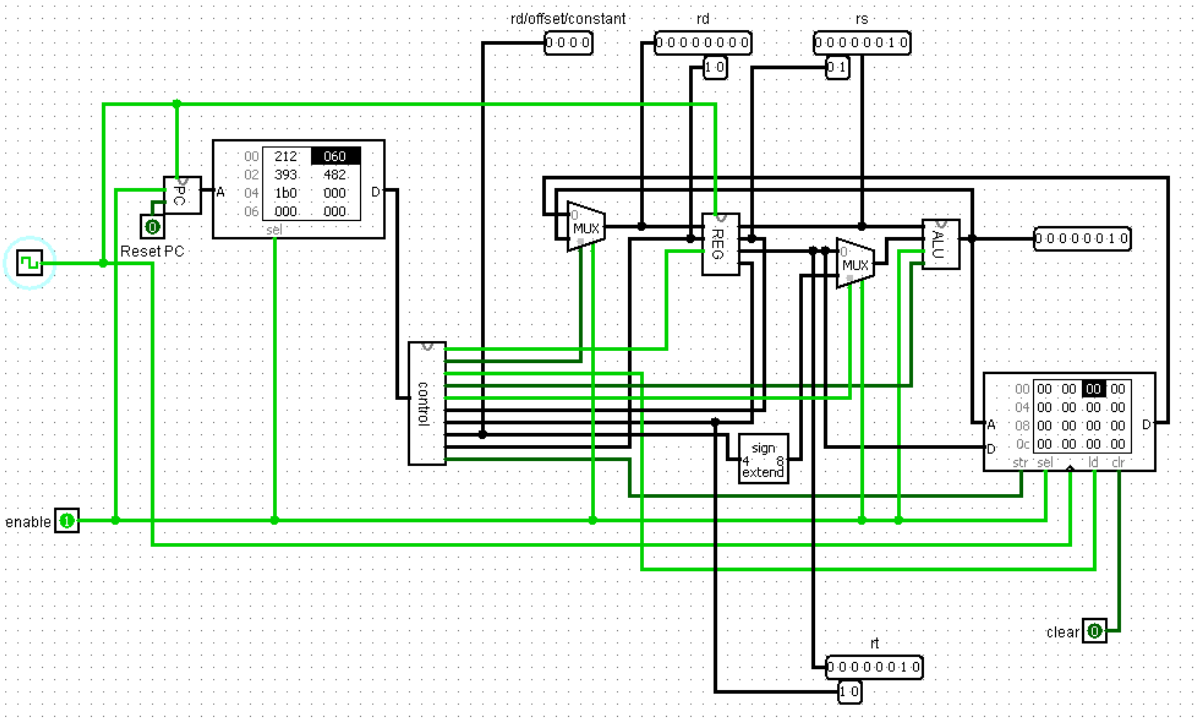
Our rt is set to the $t0 register and it has been loaded with a value of 2. Our rs, $zero in this case, is also 0 and the addition of the two is 0 that's why the output of our ALU is 0.

$zero will always have a value of zero.

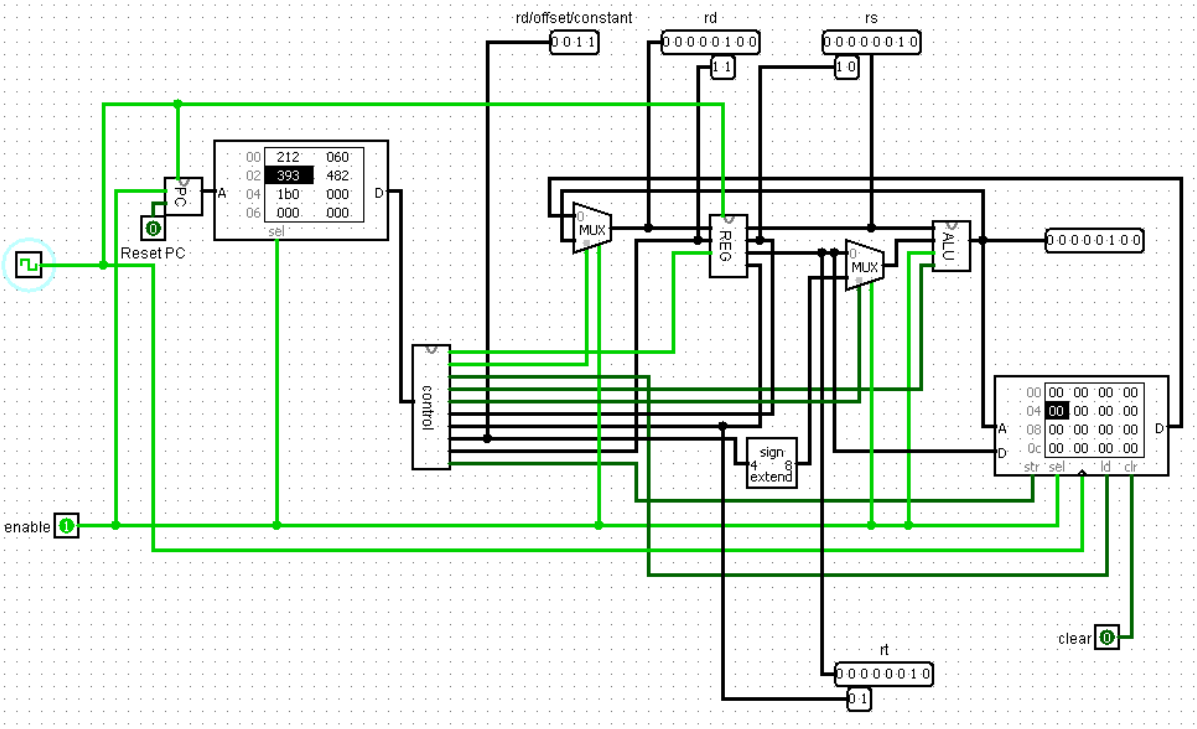
It also works for the other 4 instructions.

lw $t1, 0($t0) , 0000 01 10 0000_2 $0x060_{16}$

rt $t1 is loaded with the value at $t0, 2.



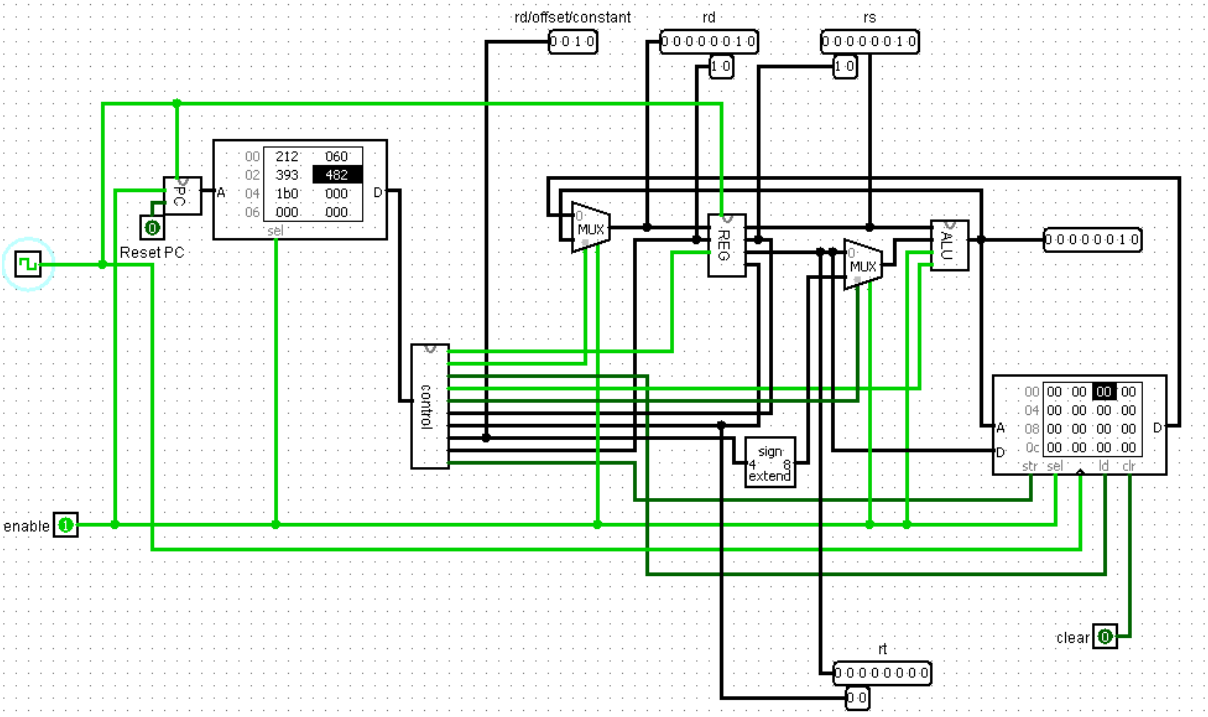
add $t2, $t1, $t0 0011 10 01 0011$_2$ 0x393$_{16}$



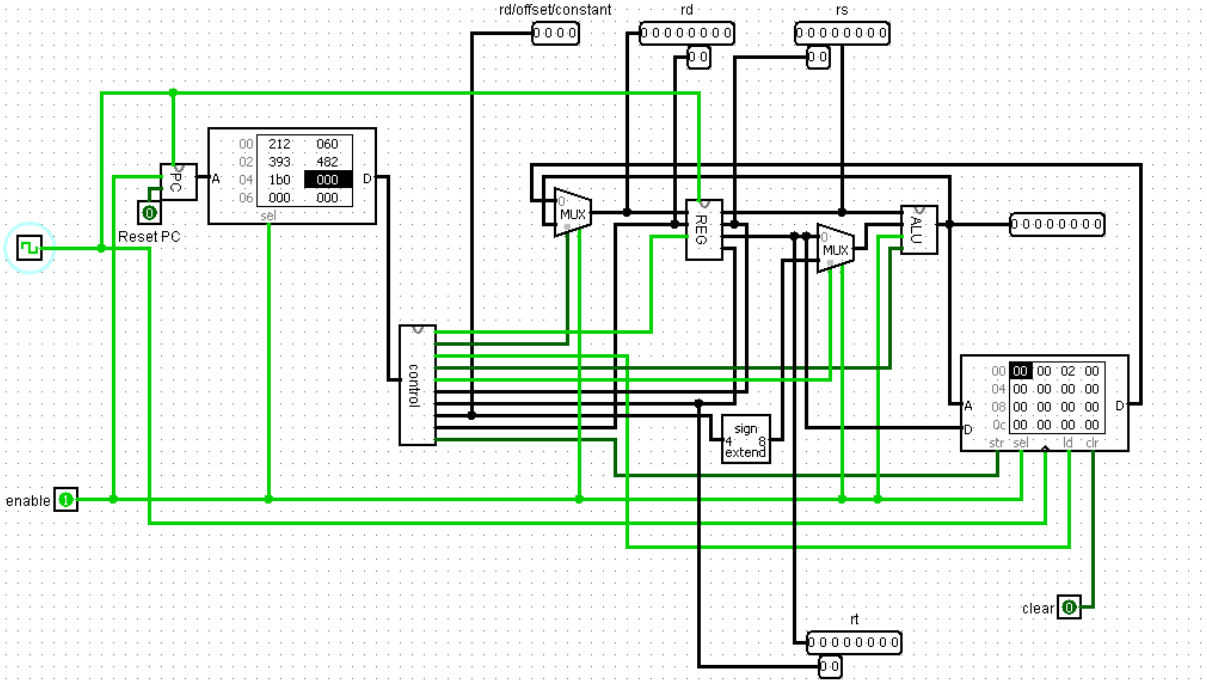
rd $t2 has been loaded with 4 the addition of $t1, 2 and $t0, 2.

sub $t1, $t2, $zer0 0100 10 00 0010$_2$ 0x4C2$_{16}$

$t1 (rd) is now equal to $t1, rs, minus $zero, rt. Which is 2.



sw $t1,0($t0), $0001\ 01\ 10\ 0000_2$ $0x160_{16}$



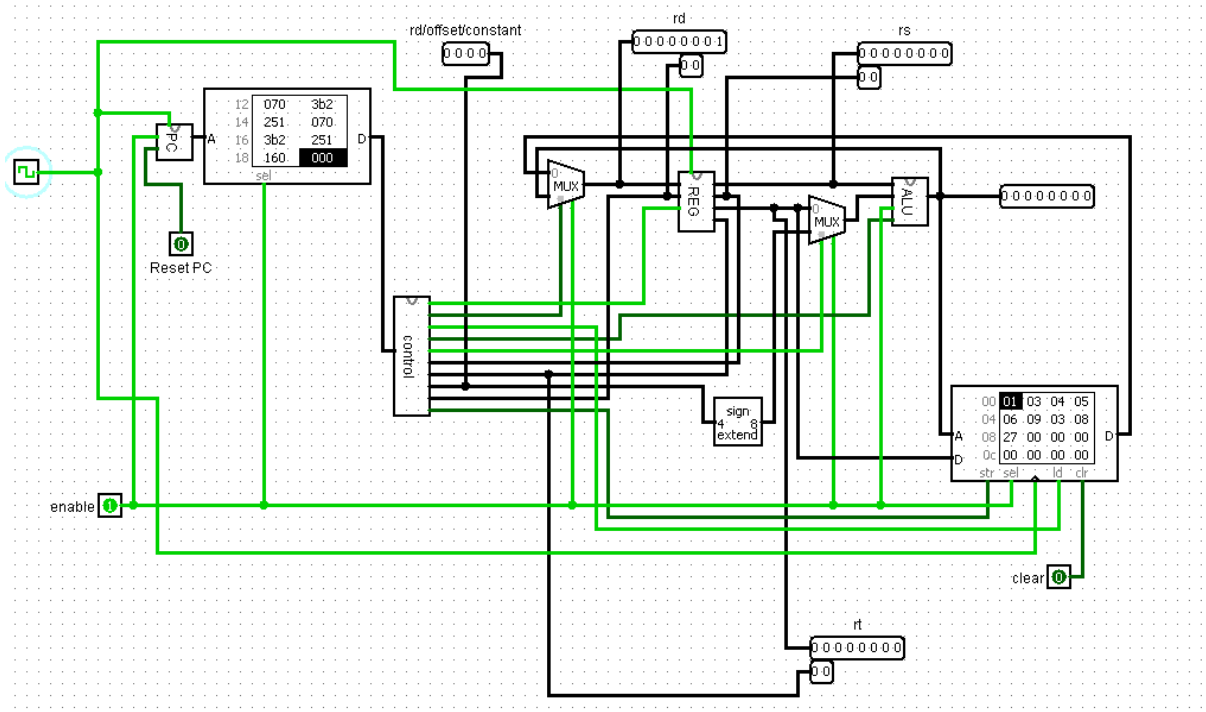
and finally we store the value of 2 from $t0 into the memory location for $t1.

So all operations work as designed.

Part 2

We then had to write a checksum program, convert it to hex, store it in our instruction memory and compile it manually.

I entered my values into the first 8 places in my ram and then ran my program and it calculated 27 in hex or 39 in decimal which is the value of my student number 13456938 added up.



Hex code of my program:

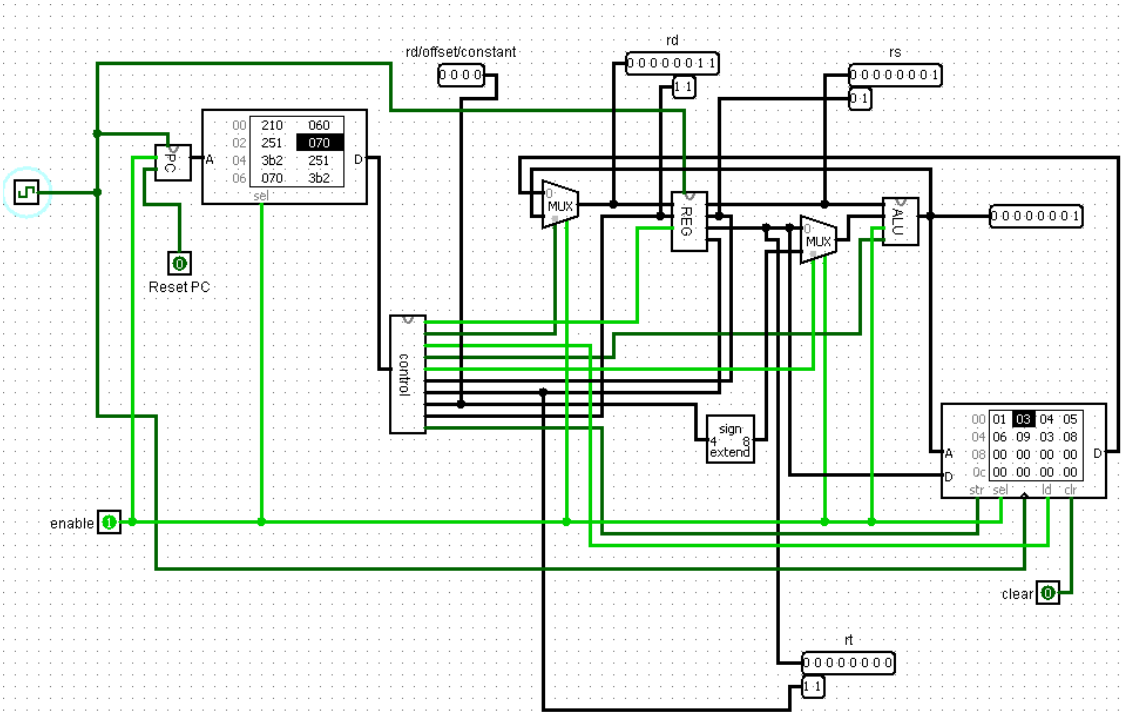
```
00 210 060 251 070  3b2 251 070 3b2  251 070 3b2 251  070 3b2 251 070
10 3b2 251 070 3b2  251 070 3b2 251  160 000 000 000  000 000 000 000
20 000 000 000 000  000 000 000 000  000 000 000 000  000 000 000 000
30 000 000 000 000  000 000 000 000  000 000 000 000  000 000 000 000
40 000 000 000 000  000 000 000 000  000 000 000 000  000 000 000 000
50 000 000 000 000  000 000 000 000  000 000 000 000  000 000 000 000
60 000 000 000 000  000 000 000 000  000 000 000 000  000 000 000 000
```

Assembly code:

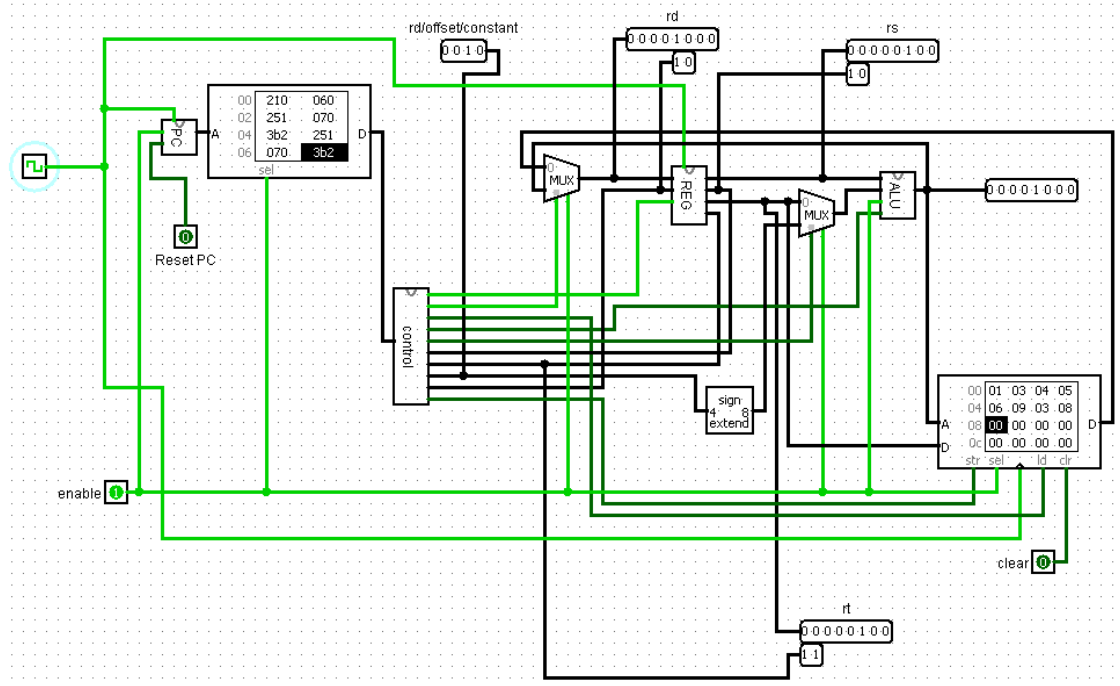
```
addi $t0, $zero, 0        # set $t0 offset variable to 0
lw $t1, 0($t0)            # load first number in memory into $t1
addi $t0, $zero, 1        # increment offset

lw $t2, 0($t0)            # load second number into $t2
add $t1, $t1, $t2         # add value of $t2 to our checksum $t1
addi $t0, $zero, 1        # increment offset

lw $t2, 0($t0)            #load third number into $t2
add $t1, $t1, $t2         # add value of $t2 to our checksum $t1
addi $t0, $zero, 1        # increment offset
```

```
lw $t2, 0($t0)          #load fourth number into $t2
add $t1, $t1, $t2       # add value of $t2 to our checksum $t1
addi $t0, $zero, 1      # increment offset

lw $t2, 0($t0)          #load fifth number into $t2
add $t1, $t1, $t2       # add value of $t2 to our checksum $t1
addi $t0, $zero, 1      # increment offset

lw $t2, 0($t0)          #load sixth number into $t2
add $t1, $t1, $t2       # add value of $t2 to our checksum $t1
addi $t0, $zero, 1      # increment offset

lw $t2, 0($t0)          #load seventh number into $t2
add $t1, $t1, $t2       # add value of $t2 to our checksum $t1
addi $t0, $zero, 1      # increment offset

lw $t2, 0($t0)          #load eighth number into $t2
add $t1, $t1, $t2       # add value of $t2 to our checksum $t1
addi $t0, $zero, 1      # increment offset

sw $t1, 0($t0)          #store value of checksum into $t0 directly after
                        #student number
```

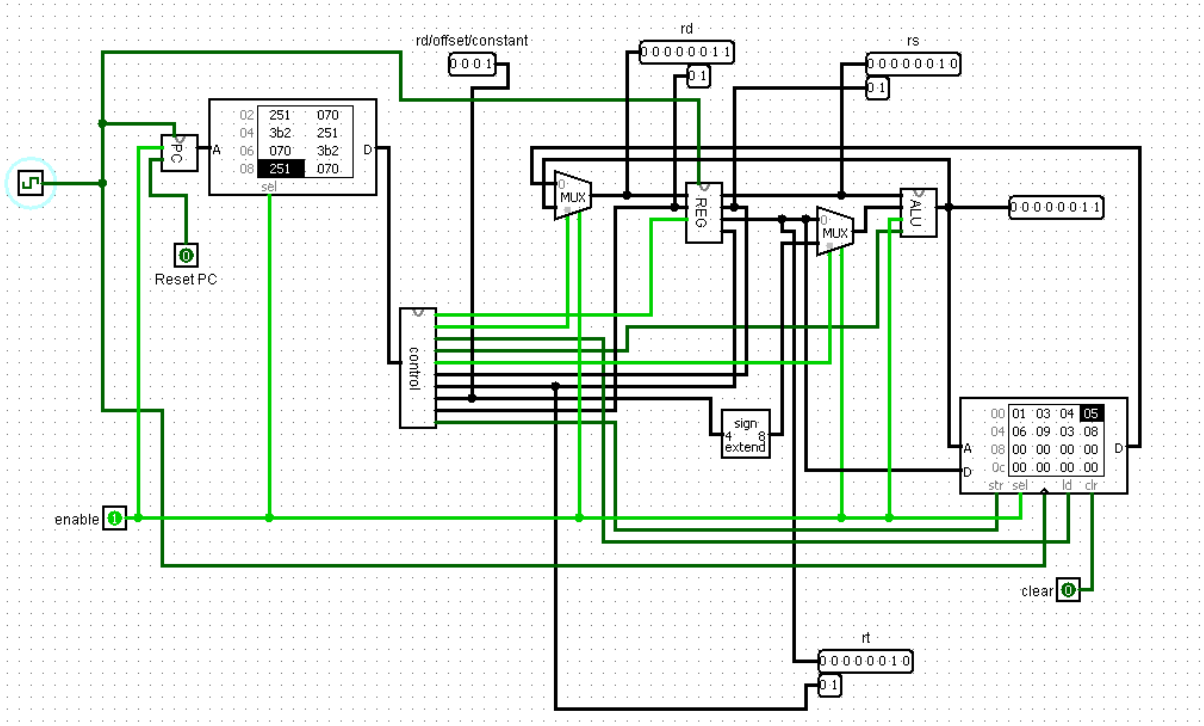
Loading the value 3 from memory into $t2.



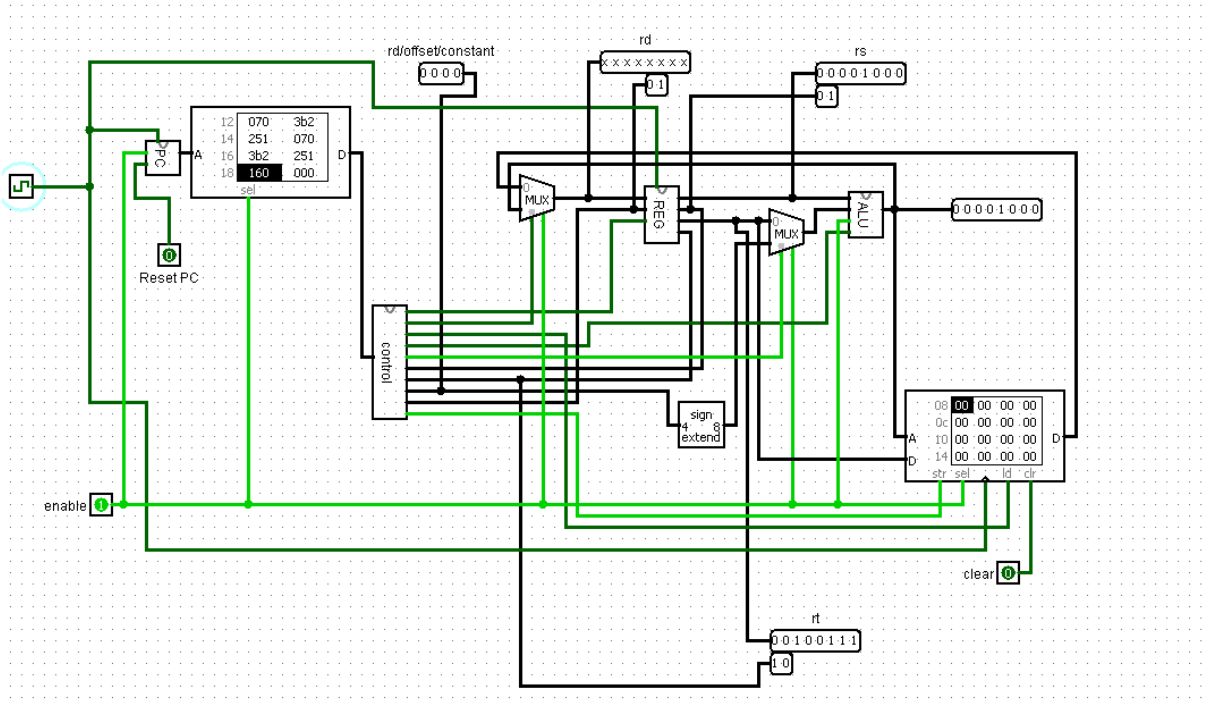
Adding $t2 to our checksum $t1. I 3 and 4 have been added so the value of $t1 is 8 or 0x00001000.(rd)



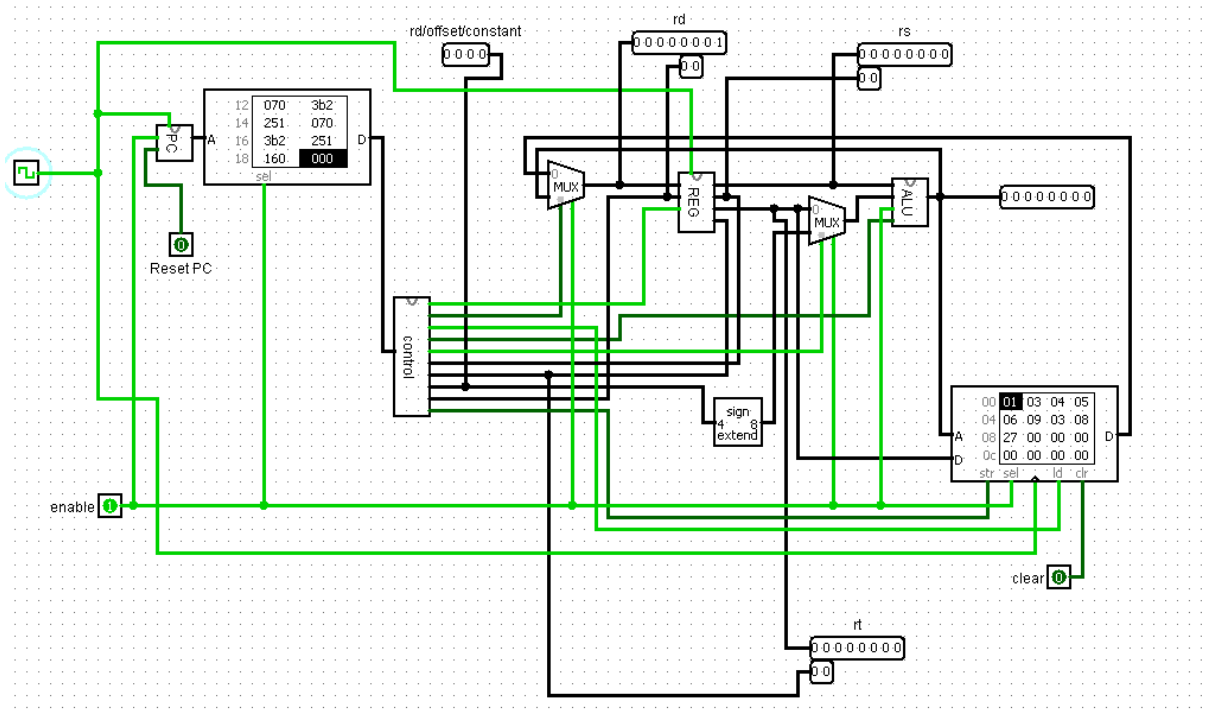
Incrementing offset $t0 by 1 to cycle to next memory address



About to store checksum



Checksum stored on positive clock edge. As entire system triggers on positive clock.


rd/offset/constant
0 0 0 0
rd
0 0 0 0 0 0 0 1
0 0
rs
0 0 0 0 0 0 0 0
0 0
PC
A
D
sel
12 070 3b2
14 251 070
16 3b2 251
18 160 000
Reset PC
MUX
REG
MUX
ALU
0 0 0 0 0 0 0 0
control
sign
4
extend
8
00 01 03 04 05
04 06 09 03 08
08 27 00 00 00
0c 00 00 00 00
A
D
D
str sel ld clr
enable
clear
rt
0 0 0 0 0 0 0 0
0 0