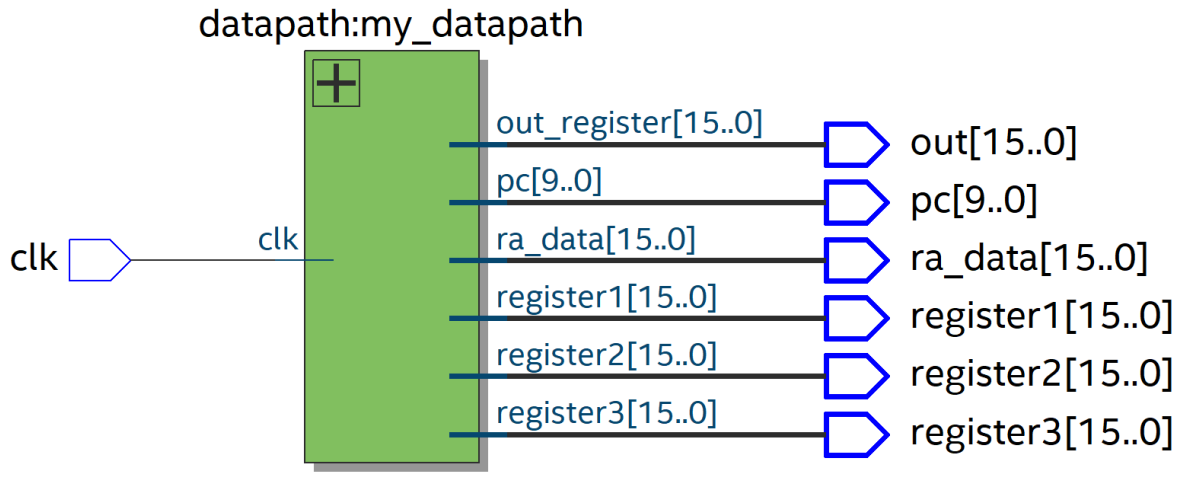
CSE-331 Final Project Report

Süleyman KORAMAZ 1901042615

Project:



Inputs: clk (clock)

Outputs: out[15:0] = data in output register(reg14 – 4'b1110),

Pc[9:0] = current instruction address

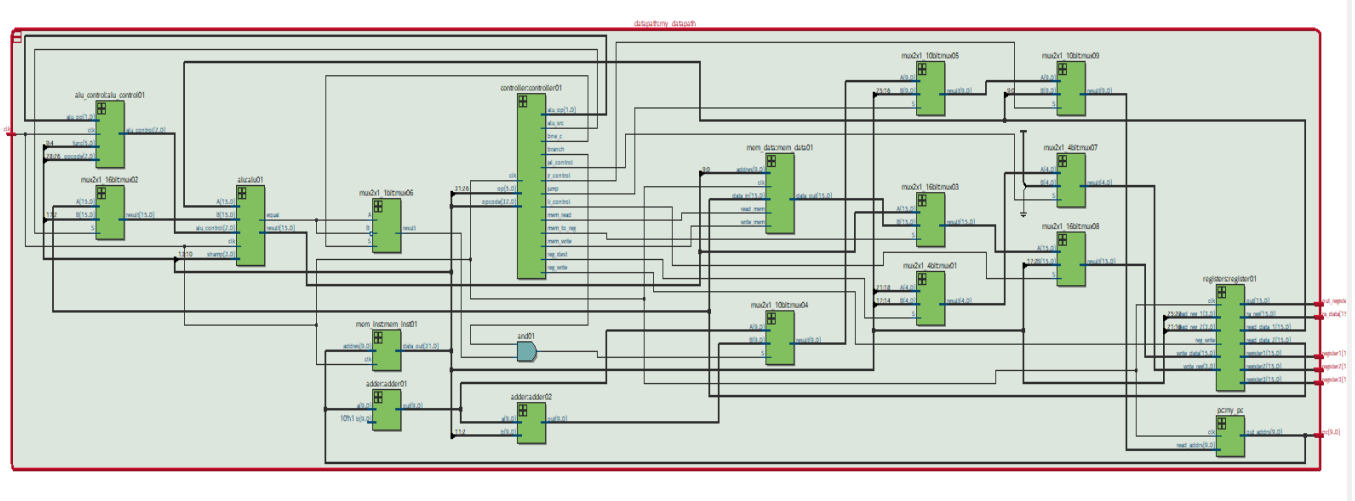
Ra_data[15:0] = ra register that holds adder for jal instruction

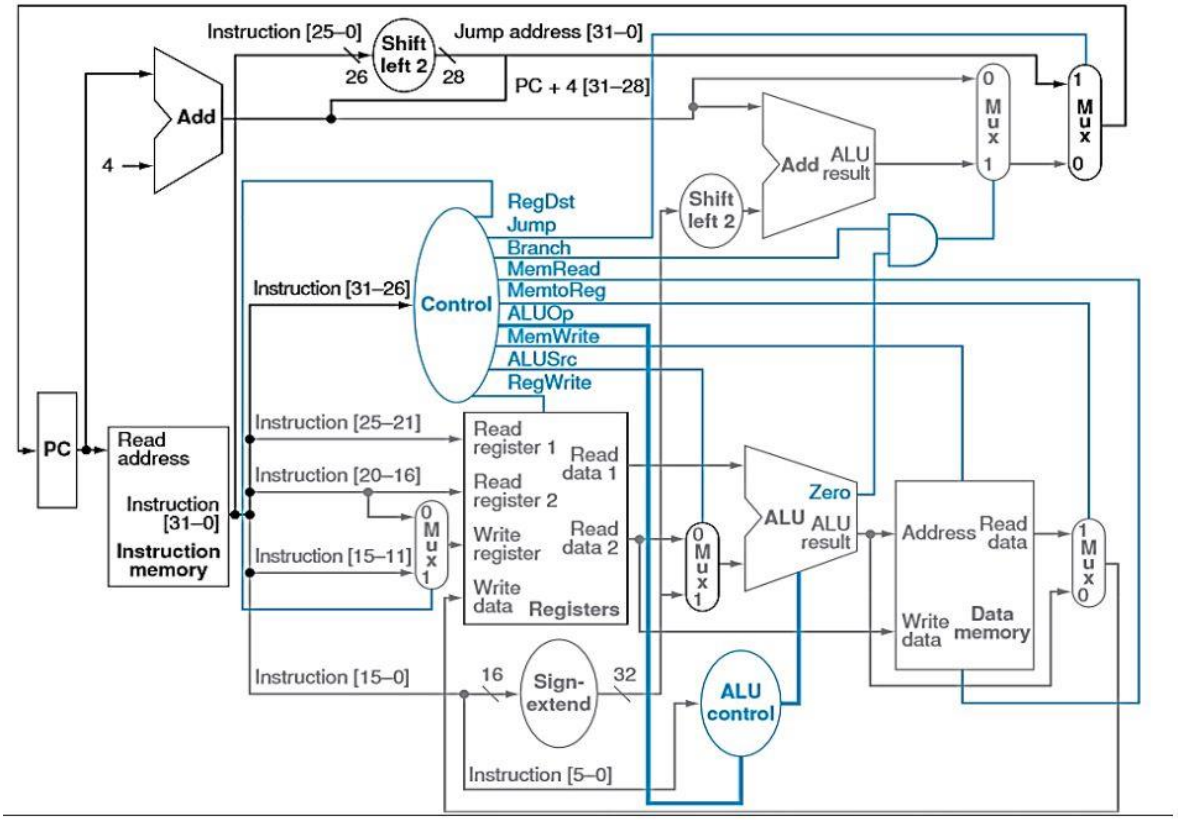
Register1 = data in reg1

Register2 = data in reg2

Register3= data in reg3

Datapath:





This is the basic of my data path.

The components added to it are:

-5 mux for branch, jump, jr, jal, li instructions.

-1 not for bne instruction.

Components not used in this data path:

-Sign-extend , shift left2. (Because of bit case rules)

All project components are behavioural.

However, **ALU** is structural.

Note: Adder add 1 to pc each clock rise.

Special registers:

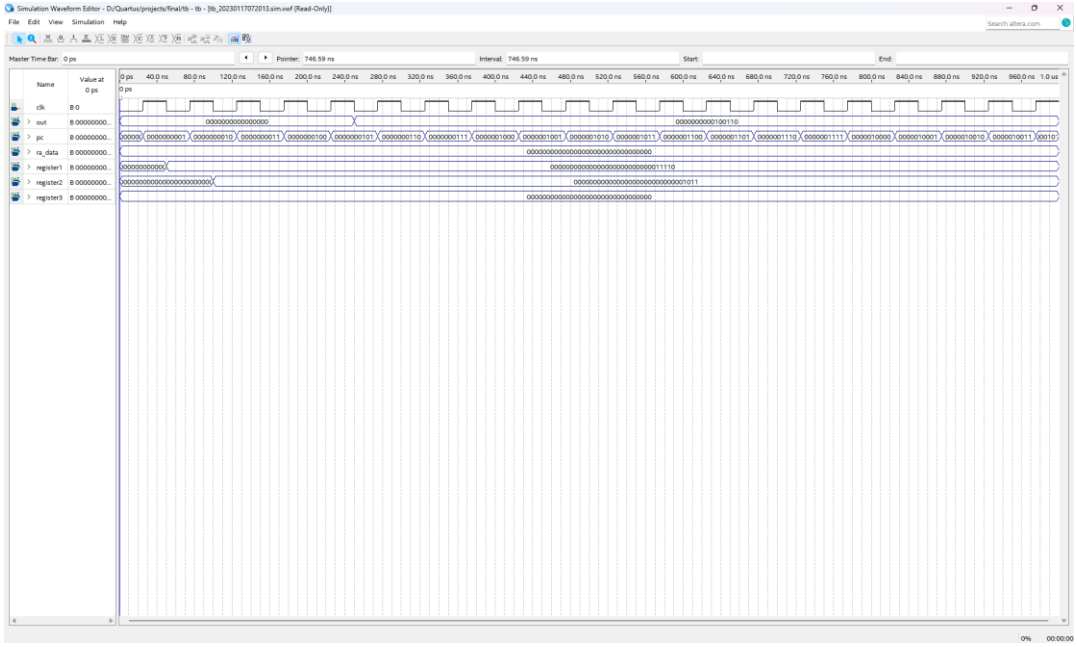
1111(reg15) is output register

1110(reg14) is ra register for jal instruction

And also, there is r1, r2, r3 outputs for more readable simulations.

In simulations registers are 32-bit but this is a mistake in my code they are 16-bit. It just tb output widths.

Note2: Data Memory has 2^10 words, because Quartus did not allow using 2'16 registers.



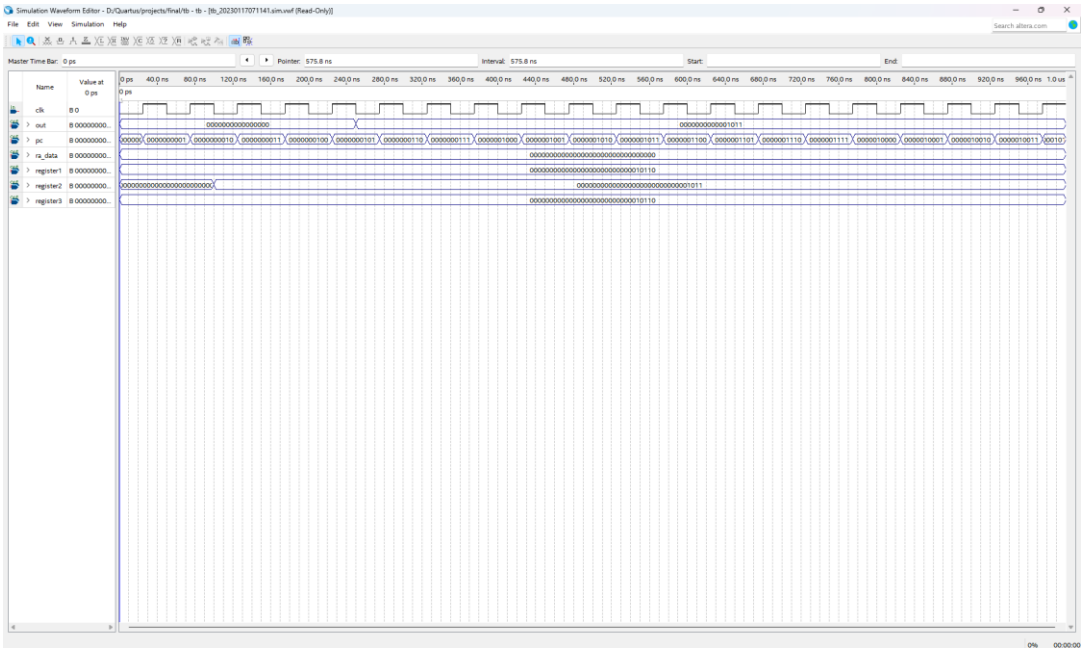
Addi

32'b0000_0100_0000_0100_0000_0000_0111_1000;//li r1

32'b0000_0100_0000_1000_0000_0000_0010_1100;//li r2

32'b0010_0000_0111_1100_0000_0010_0010_0000;//addi out r1 136

Assign reg1 and reg2 with li instruction and after that add reg1 data with 136 and put result output register.



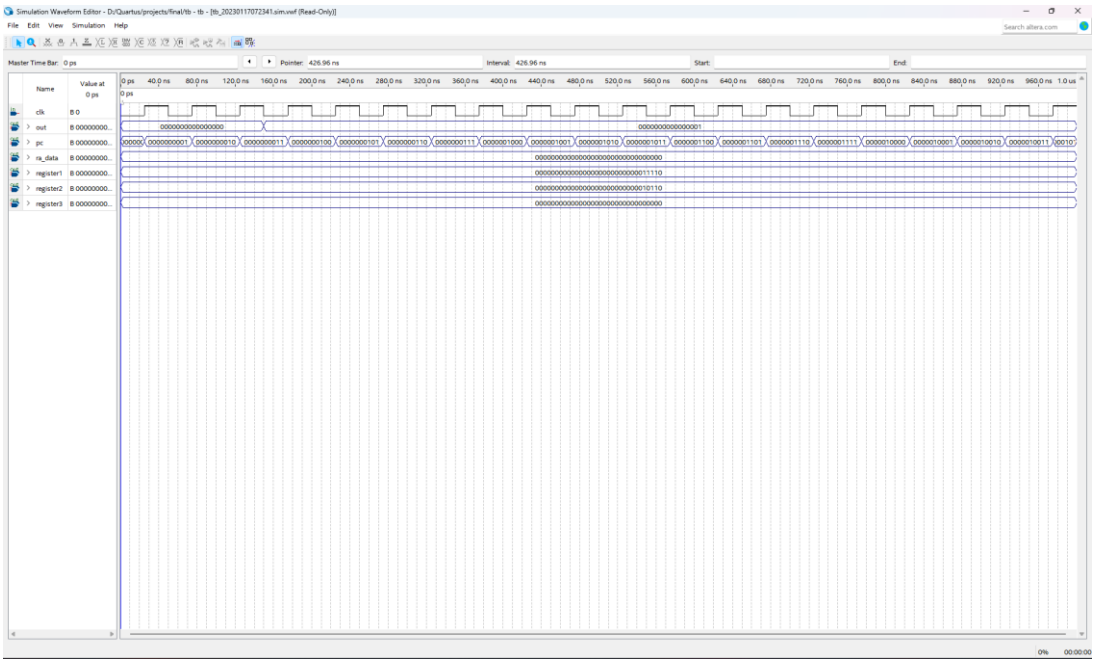
sub

32'b0000_0100_0000_0100_0000_0000_0111_1000;//li r1 30

32'b0000_0100_0000_1000_0000_0000_0010_1100;//li r2 31

32'b0000_0000_0100_1011_1100_0010_0000_0000;//sub out r1 r2

Assign reg1 and reg2 with li instruction and after that subtract reg2 from reg1 and put result output register.



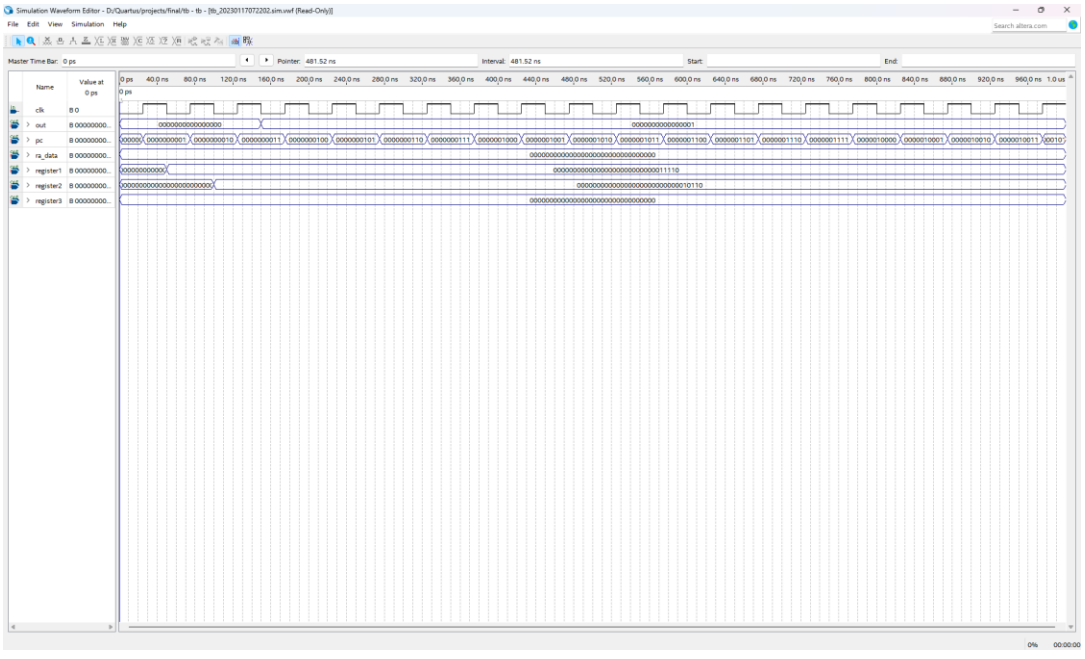
slt

32'b0000_0100_0000_0100_0000_0000_0111_1000;//li r1 30

32'b0000_0100_0000_1000_0000_0000_0101_1000;//li r2 22

32'b0000_0000_1000_0111_1100_0010_1010_0000;//slt out s2 s1

Assign reg1 and reg2 with li instruction and after that find if reg2 is less than reg1 data and put result output register.



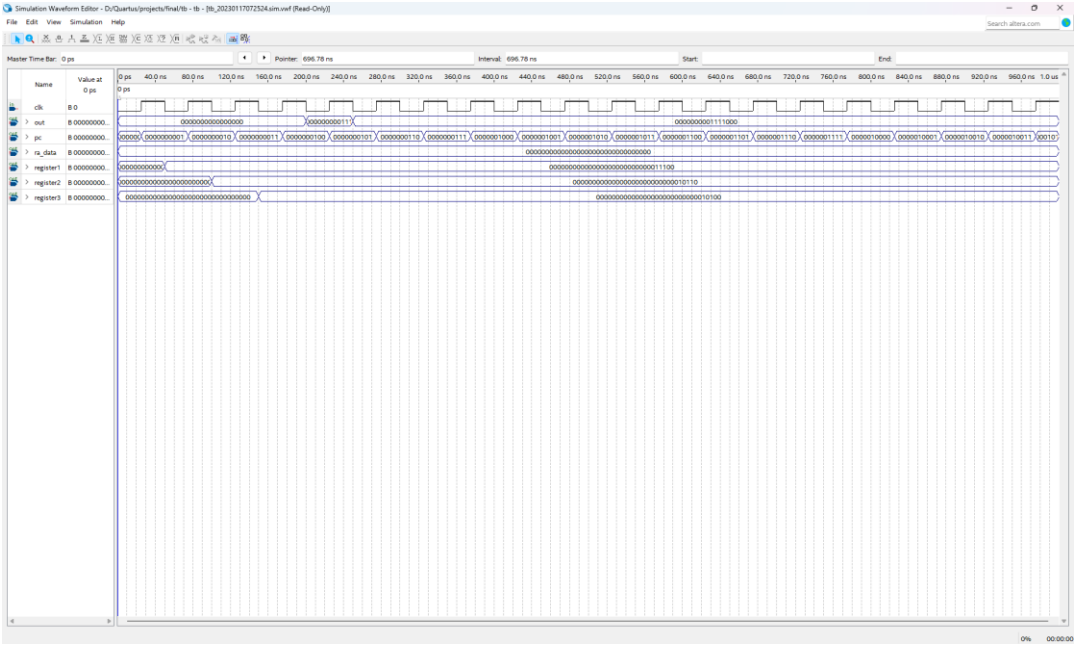
slti

32'b0000_0100_0000_0100_0000_0000_0111_1000;//li r1 30

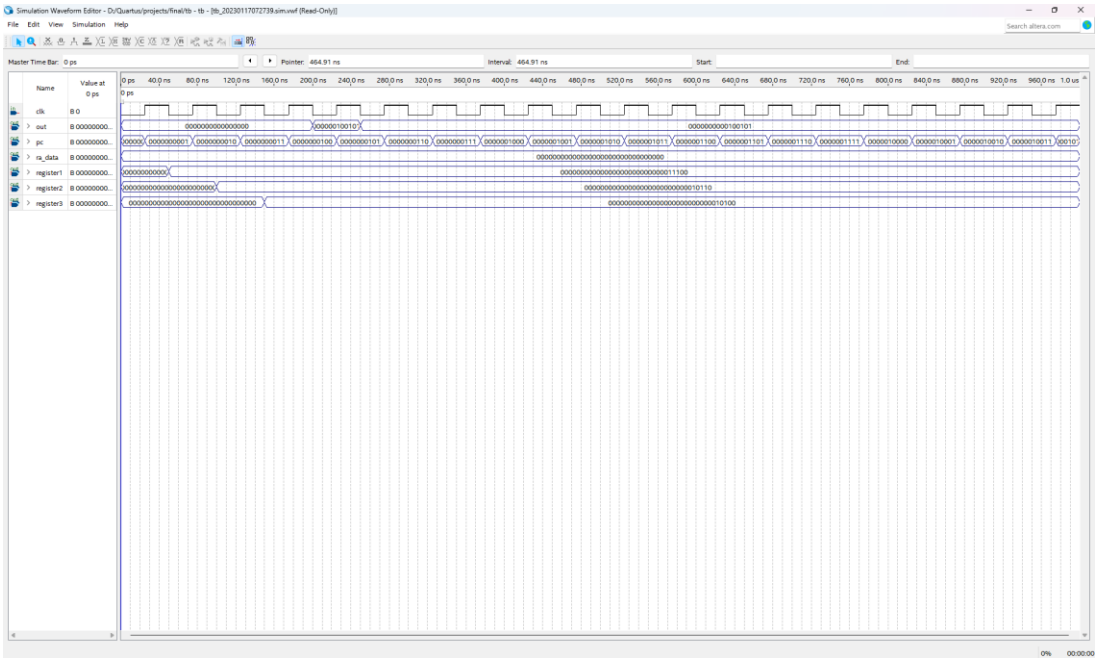
32'b0000_0100_0000_1000_0000_0000_0101_1000;//li r2 22

32'b0010_1000_1011_1100_0000_0000_0101_1100;//slti out r2 23

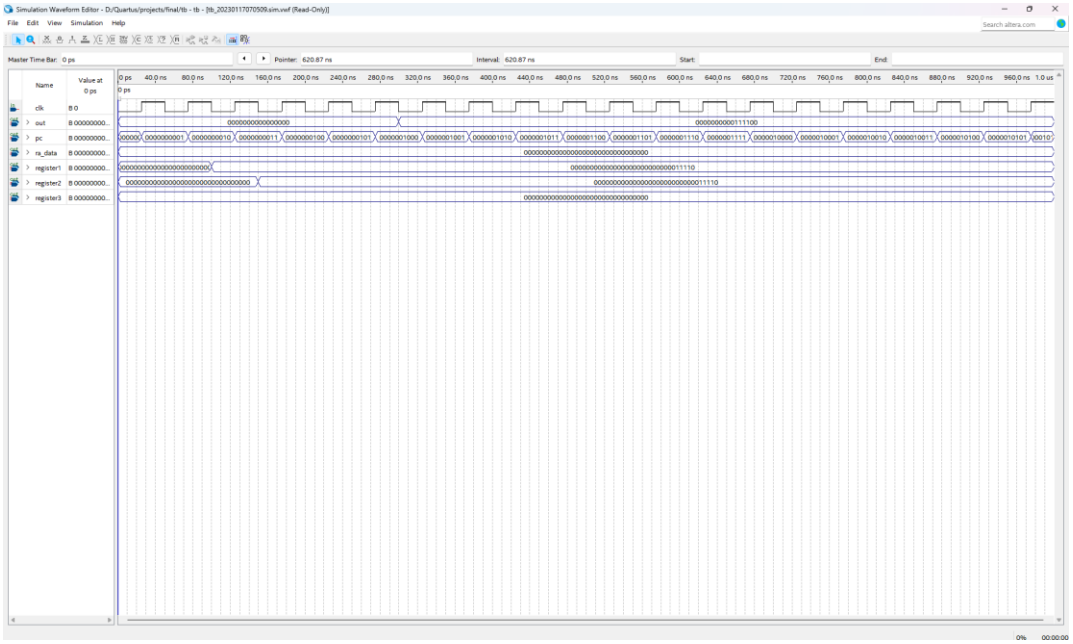
Assign reg1 and reg2 with li instruction and after that add reg1 data with 136 and put result output register.



and, or, sll
32'b0000_0100_0000_0100_0000_0000_0111_0000;//li r1
32'b0000_0100_0000_1000_0000_0000_0101_1000;//li r2
32'b0000_0000_1000_0100_1100_0010_0100_0000;//and r3 r1 r2
32'b0000_0000_1000_0111_1100_0010_0101_0000;//or out r1 r2
32'b0000_0000_0011_1111_1100_1000_0000_0000;//sll out out 2
Assign reg1 and reg2 with li instruction -> ‘and’ them, put result r3 -> ‘or’ them, put result out, shift left out data 2 times.



andi, ori, srl
32'b0000_0100_0000_0100_0000_0000_0111_0000;//li r1
32'b0000_0100_0000_1000_0000_0000_0101_1000;//li r2
32'b0011_0000_0100_1100_0000_0000_0101_1000;//andi r3 r1 22
32'b0011_0111_1111_1100_0000_0010_0101_0000;//ori out r1 148
32'b0000_0000_0011_1111_1100_1000_0010_0000;//srl out out 2



beq

32'b0000_0100_0000_0100_0000_0000_0111_1000;//li r1 30

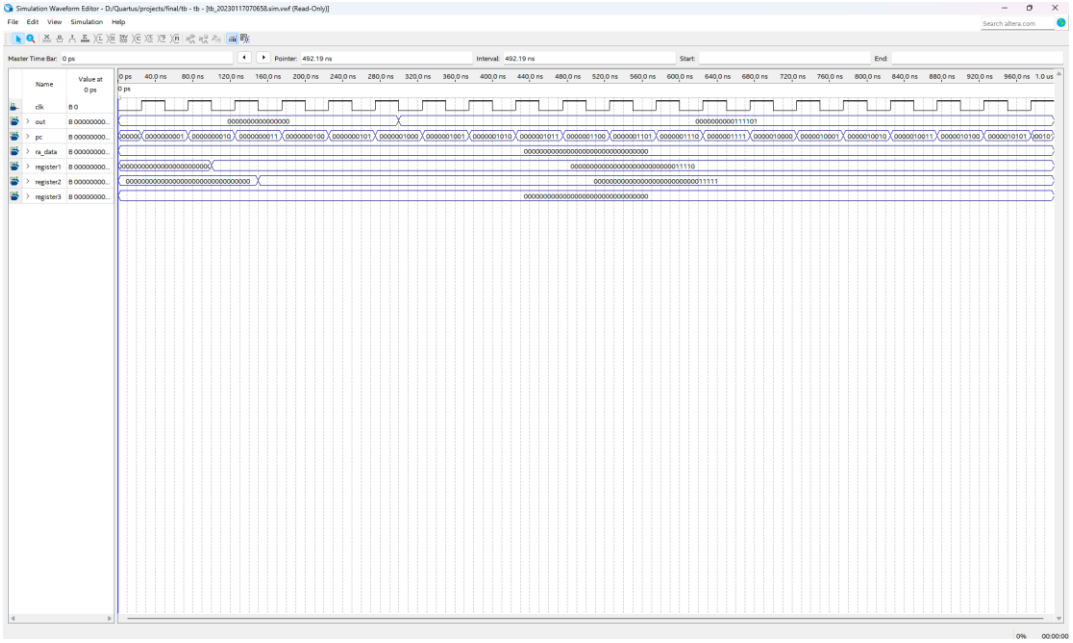
32'b0000_0100_0000_1000_0000_0000_0111_1000;//li r2 30

32'b0001_0000_0100_1000_0000_0000_0000_1000;//beq r1 r2 2

32'b0000_0100_0000_0100_0000_0001_0000_0000;//li r1 64

32'b0000_0000_0100_1011_1100_0010_0000_0000;//add out r1 r2

Put reg1 30, put reg2 30, if they are equal it will jump target register and output will be 60. If they are not equal, put reg1 64, after add.



bne

32'b0000_0100_0000_0100_0000_0000_0111_1000;//li r1 30

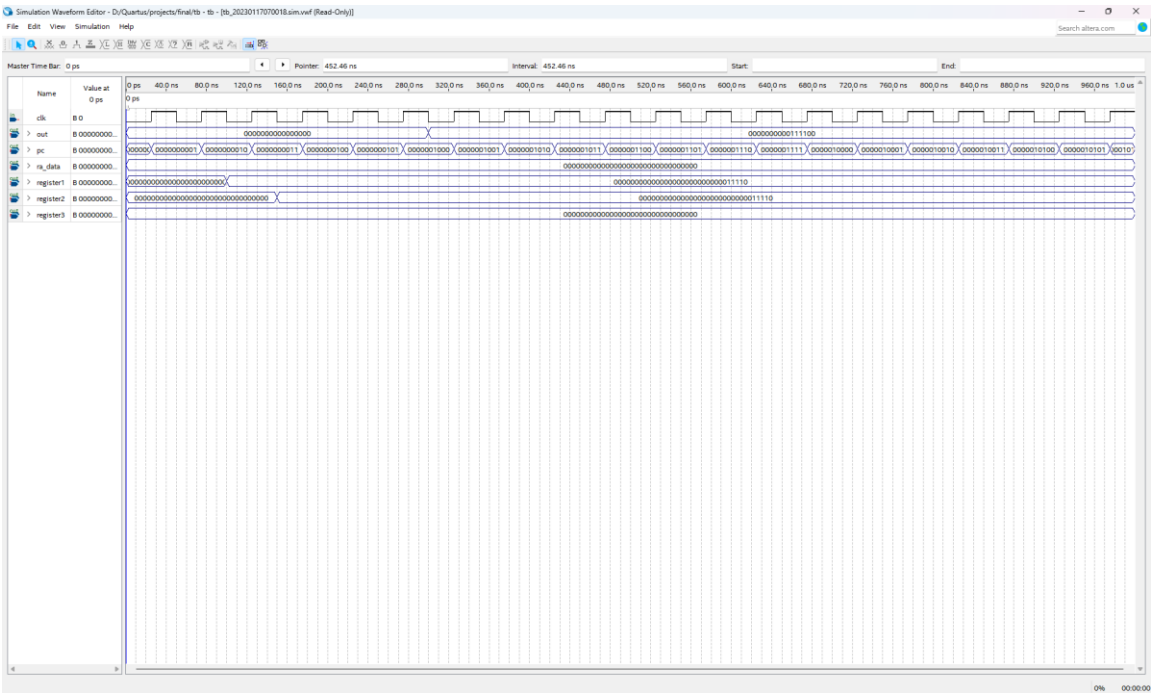
32'b0000_0100_0000_1000_0000_0000_0111_1100;//li r2 31

32'b0001_0100_0100_1000_0000_0000_0000_1000;//bne r1 r2 2

32'b0000_0100_0000_0100_0000_0001_0000_0000;//li r1 64

32'b0000_0000_0100_1011_1100_0010_0000_0000;//add out r1 r2

Put reg1 30, put reg2 31, if they are not equal it will jump target register and output will be 61. If they are not equal, put reg1 64, after add.



j

32'b0000_0100_0000_0100_0000_0000_0111_1000;//li r1 30

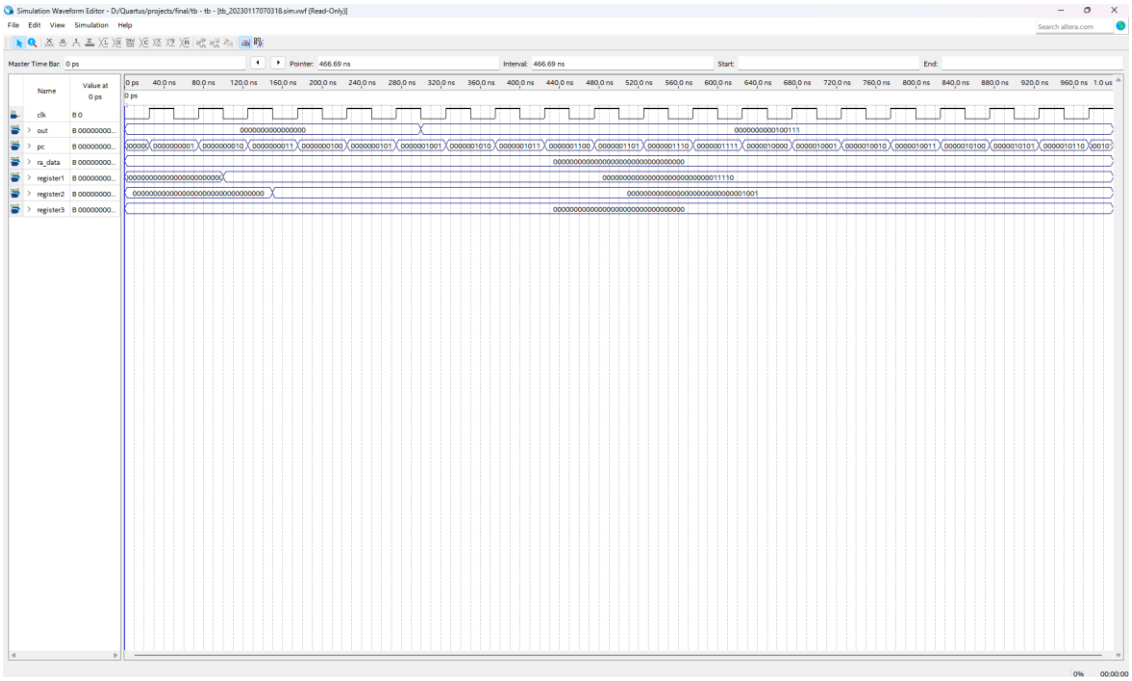
32'b0000_0100_0000_1000_0000_0000_0111_1000;//li r2 30

32'b0000_1000_0000_1000_0000_0000_0000_0000;//j 8

32'b0000_0100_0000_0100_0000_0001_0000_0000;//li r1 64

32'b0000_0000_0100_1011_1100_0010_0000_0000;//add out r1 r2

Put reg1 30, put reg2 31, jump target register and output will be 61. If jump won't work, program would put reg1 64, after add.



jr

32'b0000_0100_0000_0100_0000_0000_0111_1000;//li r1 30

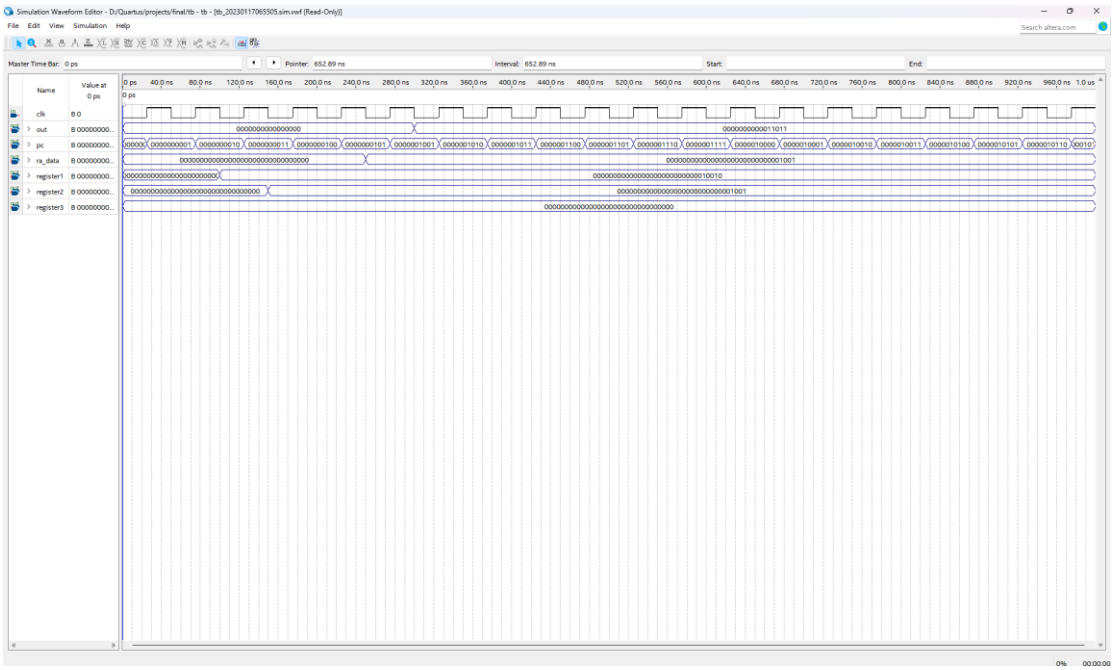
32'b0000_0100_0000_1000_0000_0000_0010_0100;//li r2 9

32'b0000_0000_1000_0000_0000_0000_1000_0000;//jr r2

32'b0000_0100_0000_0100_0000_0001_0000_0000;//li r1 64

32'b0000_0000_0100_1011_1100_0010_0000_0000;//add out r1 r2

Jump target address data in holds the reg2. Example logic same with j test.



jal

32'b0000_0100_0000_0100_0000_0000_0100_1000;//li r1 18

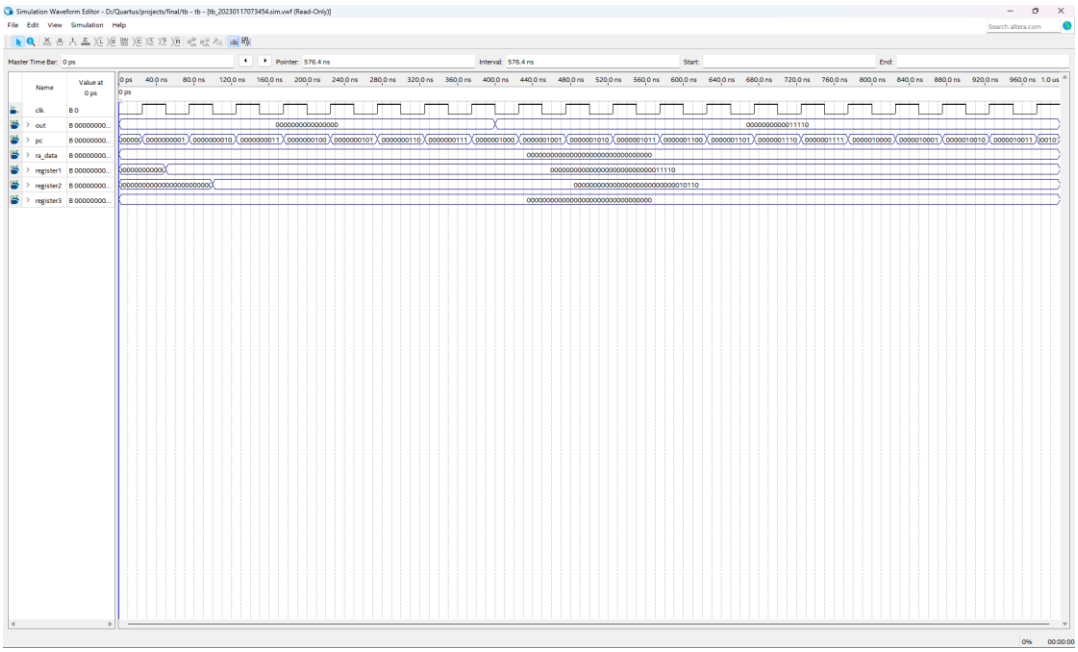
32'b0000_0100_0000_1000_0000_0000_0010_0100;//li r2 9

32'b0000_1100_0000_1001_0000_0000_0000_0000;//jal 9

32'b0000_0100_0000_0100_0000_0001_0000_0000;//li r1 64

32'b0000_0000_0100_1011_1100_0010_0000_0000;//add out r1 r2

Jump target address and put PC+1 to ra register. Example logic same with j test.



special1

32'b0000_0100_0000_0100_0000_0000_0111_1000;//li r1 30

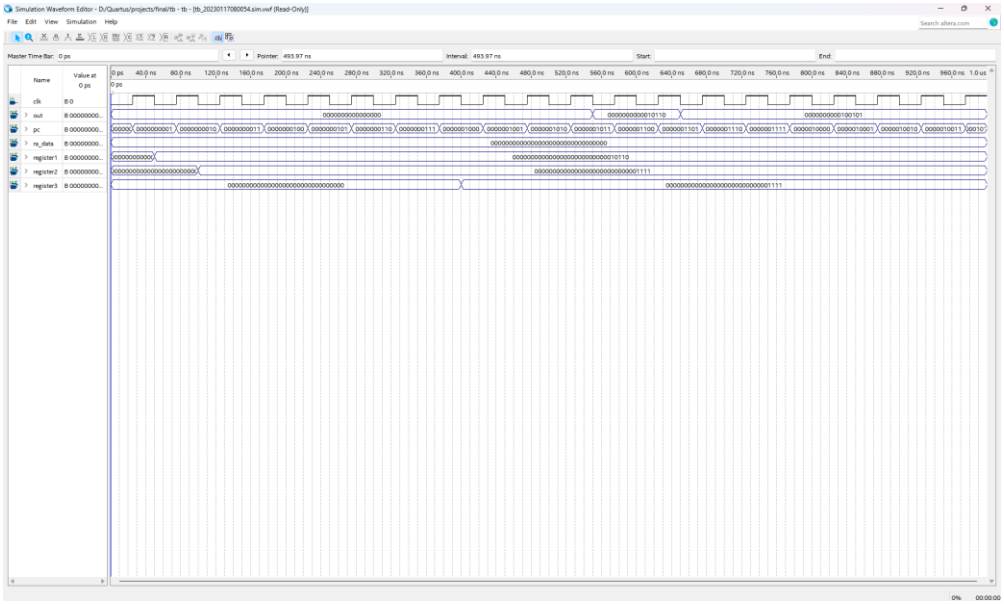
32'b0000_0100_0000_1000_0000_0000_0101_1000;//li r2 22

32'b1000_1100_0011_1100_0000_0000_0000_1000;//lw out m0(2)

32'b1010_1100_0000_0100_0000_0000_0000_1000;//sw r1 m0(2)

32'b1000_1100_0011_1100_0000_0000_0000_1000;//lw out m0(2)

Put reg1 -> 30 , reg2 -> 22. After that instructions, program try load word from memory but it is 0 so output didn't change. After that store word that register and after store, program will load that data.(22) (Address is m0 + 2 = m2)



special2

32'b0000_0100_0000_0100_0000_0000_0101_1000;//li r1 22

32'b0000_0100_0000_1000_0000_0000_0011_1100;//li r2 15

32'b1000_1100_0000_1000_0000_0000_0001_0000;//sw r2 m0(4)

32'b1010_1100_0000_0100_0000_0000_0000_1000;//sw r1 m0(2)

32'b1000_1100_0000_1100_0000_0000_0001_0000;//lw r3 m0(4)

32'b1000_1100_0011_1100_0000_0000_0000_1000;//lw out m0(2)

32'b0000_0000_1111_1111_1100_0010_0000_0000;//add out out r3

Put reg1 -> 22 , reg2 -> 15. After that store that values to memory contents. (m4 and m2).

Subsequently, program load that values to r3 and out reg. The last instruction add them and put result out reg.