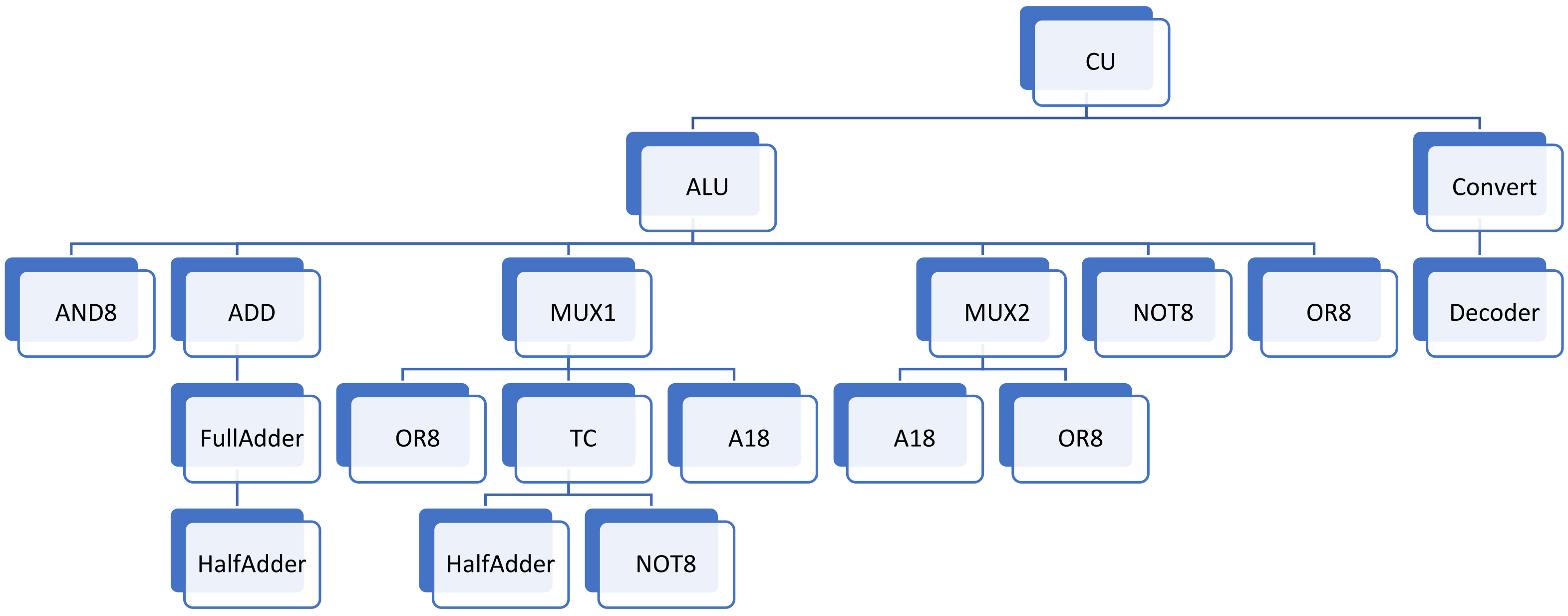
Report for the CPU design

My design for CPU contains few individual modules connected to get the final output using different logics.

Some of the modules use other modules. All those are represented in the following diagram.

Below is the tree diagram of all the modules followed by brief discussion about all the modules



Module HalfAdder:

Inputs: Two 1 bit inputs “A, B”.

Output: 1 bit output “sum”.

Function: Calculates Sum[Output] and Carry [Not an explicit output but will be used in TC].

Module FullAdder:

Inputs: Three 1 bit inputs “A, B, cin”.

Output: 1 bit output “sum”.

Function: Calculates Sum using two Halfadders.

Modules to be included: HalfAdder

Module AND8:

Inputs: Two 8 bit inputs “operand1, operand2”.

Output: 8 bit output “result”.

Function: AND for each bit in operand1 and operand2.

Module OR8:

Inputs: Two 8 bit inputs “operand1, operand2”.

Output: 8 bit output “result”.

Function: OR for each bit in operand1 and operand2.

Module NOT8:

Inputs: 8 bit input “operand1”.

Output: 8 bit output “result”.

Function: NOT for each bit in operand1.

Module ADD:

Inputs: Two 8 bit inputs “operand1, operand2” and One 1 bit input “cin”.

Output: 8 bit output “result”.

Function: Calculates each bit in result using full adders by using output carry of one fulladder as input carry for the next one. This is a Ripple Carry Adder.

Modules to be included: FullAdder

Module TC:

Inputs: One 8 bit input “operand”.

Output: 8 bit output “result”.

Function: Calculates two’s compliment of operand using NOT8 to invert and HalfAdder to add 1.

Modules to be included: NOT8, HalfAdder.

Module A18:

Inputs: One 8 bit input “inp” and One 1 bit input “c”.

Output: 8 bit output “result”.

Function: AND for each bit in “inp” with “c” to get each bit in “result”.

Module MUX1:

Inputs: Two 8 bit inputs “operand2, select”.

Output: 8 bit output “op2”.

Function: Uses last 4 bits of select and chooses one of the four possibilities for op2.

Possibilities for op2:

a) Op2 = operand2 [For Additon]

b) Op2 = two’s compliment of operand2 [For Subtraction]
c) Op2 = 00000001 [For Increment]
d) Op2 = 11111111 [For decrement]

Modules to be included: TC, A18, OR8

Module MUX2:

Inputs: Five 8 bit inputs “result1, result2, result3, result4, result5, select”.

Output: 8 bit output “result”.

Function: Last four bits of select are made as 1 bit using “or” gate for all four and with the next three bits we get a combined 4 bit select. With these 4 bits, we select one of the 4 results possible[result1, result2, result3, result4] and assign it to “result”.

Modules to be included: A18, OR8

Module Decoder:

Inputs: One 3 bit input “operation”.

Output: 8 bit output “select”.

Function: 3×8 decoder. Select has only one bit high and all 7 remaining as 0’s

Module Convert:

Inputs: 19 bit input “instruction”.

Output: Three 8 bit outputs “operand1, operand2, select”.

Function: Divides 19 bit instruction into two 8 bits “operand1 and operand2” and 1 3 bit “operation” which is then passed into decoder to calculate 8 bit “select”.

Modules to be included: Decoder

Module ALU:

Inputs: Three 8 bit outputs “operand1, operand2, select”.

Output: 8 bit output “result”.

Function: Using MUX1 finds op2 and perform ADD of operand1 and op2 with cin as 0 and performs AND8, OR8 for operand1 and operand2 and also performs NOT8 of operand1.

Now, using MUX2, final result is computed by assigning one among the four possible results.

Modules to be included: AND8, NOT8, OR8, ADD, MUX1, MUX2

Module CU:

Inputs: 19 bit input “instruction”.

Output: 8 bit output “result”.

Function: Uses Convert to get inputs for ALU and then computed final “result” in ALU.

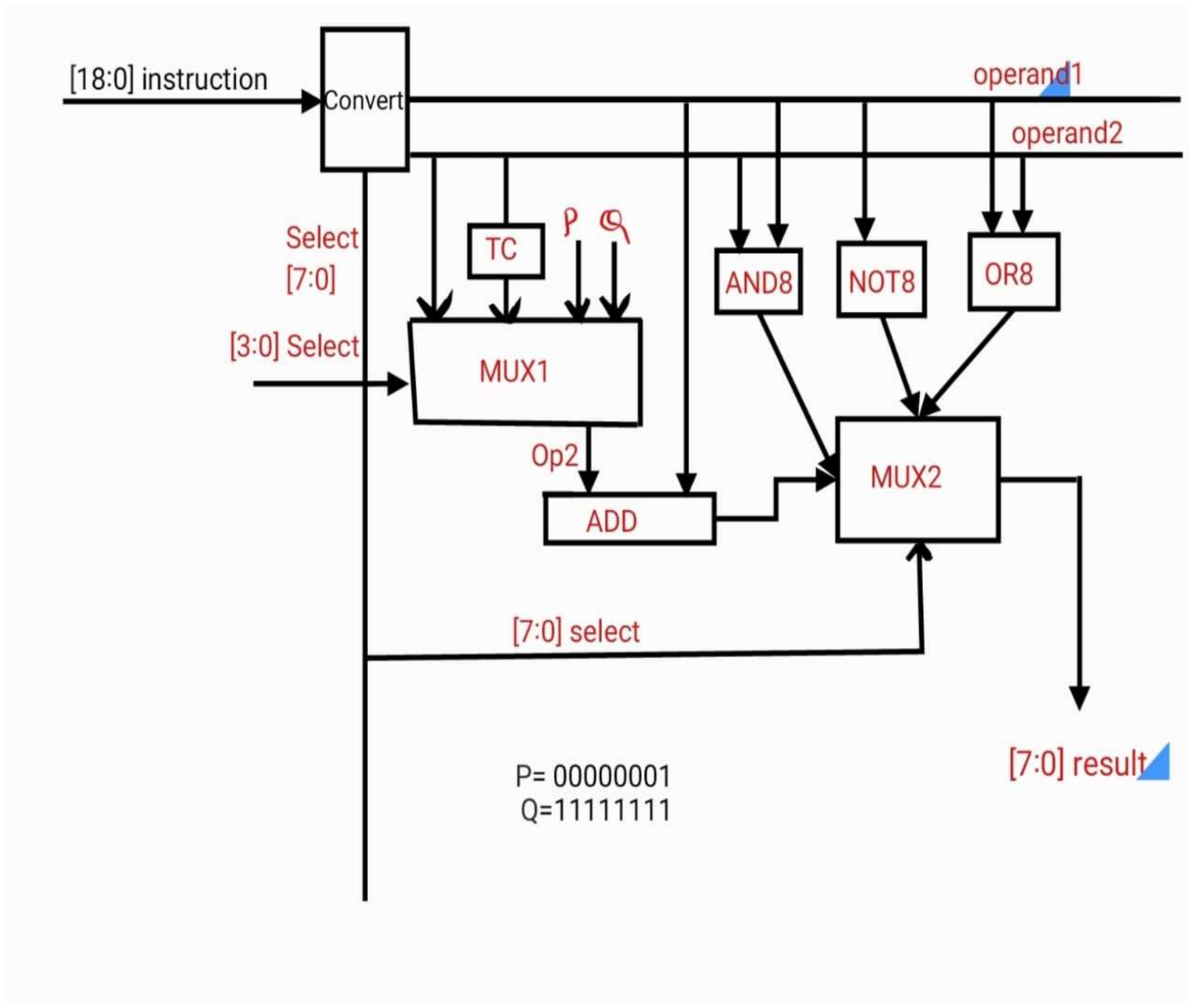
Modules to be included: Convert, ALU

Circuit Diagram and logic

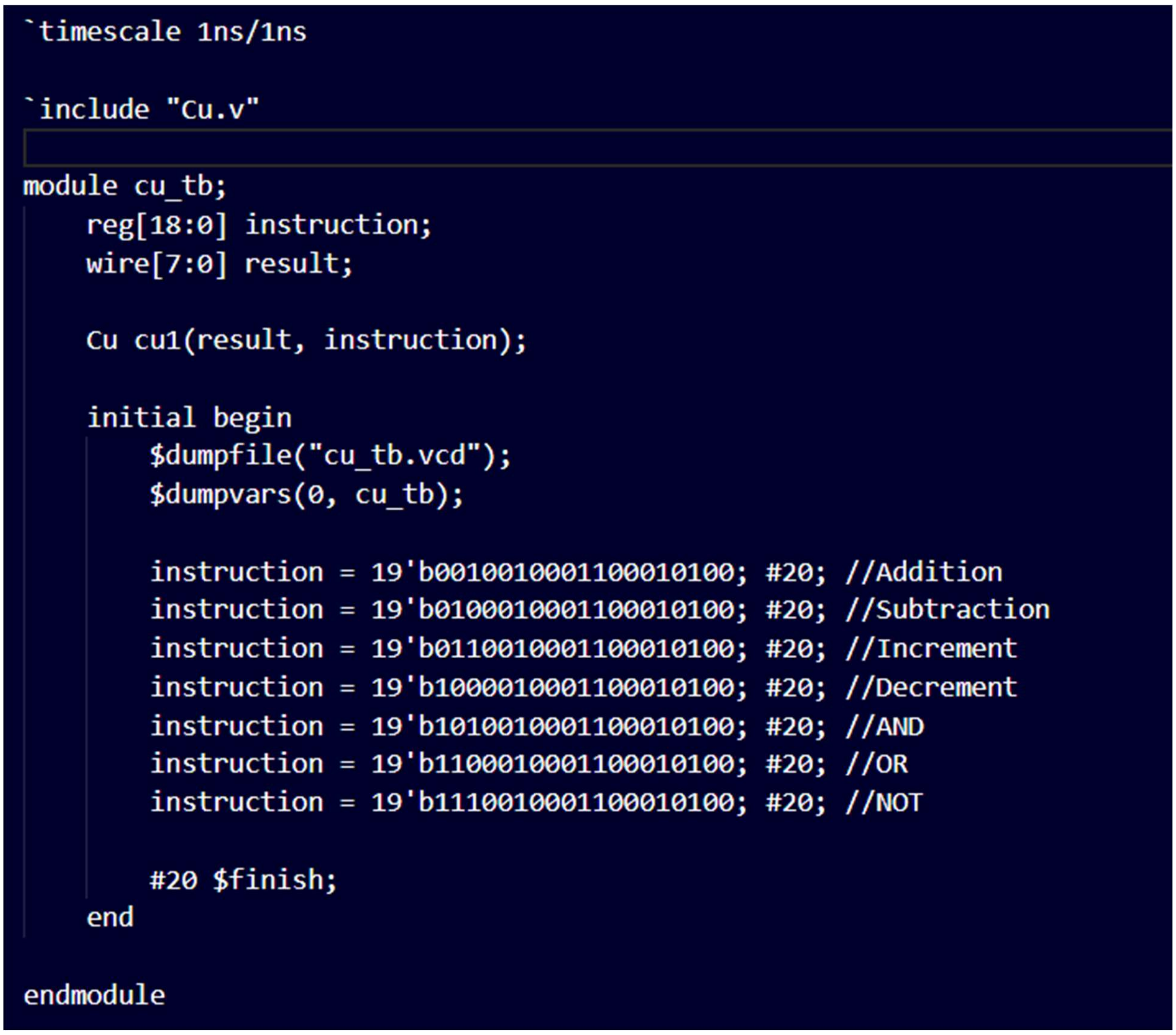
The logic for the cpu using all these modules is shown pictorially below.

- A 19 bit instruction is given as input.
- Convert makes the 19 bits into 3 parts: operand1(8 bits), operand2(8 bits), select(8 bit). Select is the output of the 3×8 decoder in which instruction[18:16] is passed as input.
- The instructions ADD, SUBTRACT, INCREMENT, DECREMENT only differ in the second operand. So, we use MUX1 for selecting the second operand.
- The bits select[3:0] are used for choosing op2 among operand2, two’s compliment of operand2, P (00000001), Q (11111111).
- ADD is performed with operand1, op2 and carry input as 0. Let this be result1.

- AND8, OR8, NOT8 are performed accordingly and let these results be result2, result3, result4.
- To differentiate between the first four operations and last three operations, we can use or gate for the bits select[3:0]. Let it be s.
- MUX2 is used for choosing the correct result among result1, result2, result3, result4.
- The bits select[6:4] and s are used to get the correct result among the four results possible.

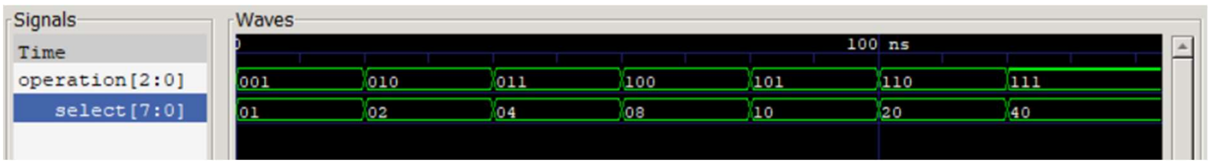


Below are the waveform snapshots for each module for the following test bench.



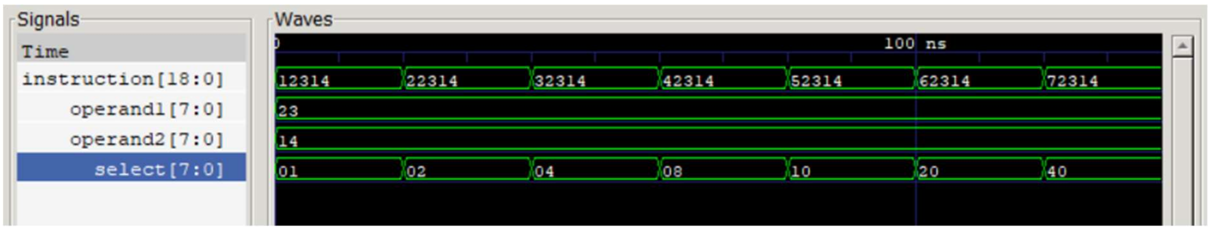
```
`timescale 1ns/1ns

`include "Cu.v"

module cu_tb;
    reg[18:0] instruction;
    wire[7:0] result;

    Cu cu1(result, instruction);

    initial begin
        $dumpfile("cu_tb.vcd");
        $dumpvars(0, cu_tb);

        instruction = 19'b0010010001100010100; #20; //Addition
        instruction = 19'b0100010001100010100; #20; //Subtraction
        instruction = 19'b0110010001100010100; #20; //Increment
        instruction = 19'b1000010001100010100; #20; //Decrement
        instruction = 19'b1010010001100010100; #20; //AND
        instruction = 19'b1100010001100010100; #20; //OR
        instruction = 19'b1110010001100010100; #20; //NOT

        #20 $finish;
    end

endmodule
```

Decoder (select, operation)

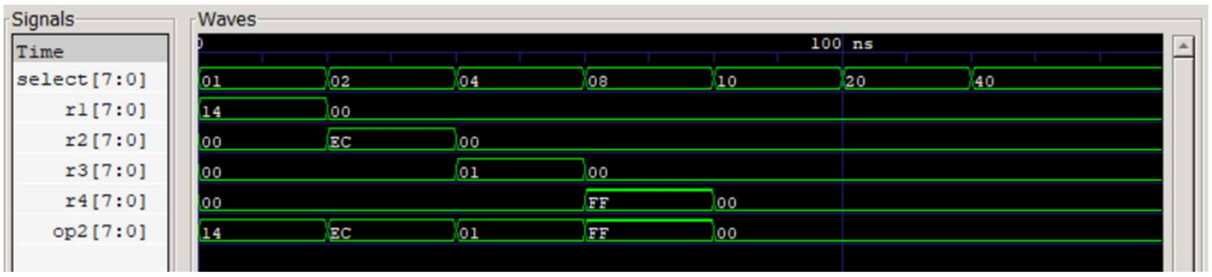


```
Signals            Waves
Time               0                                                100 ns
operation[2:0]     001    010    011    100    101    110    111
select[7:0]        01     02     04     08     10     20     40
```

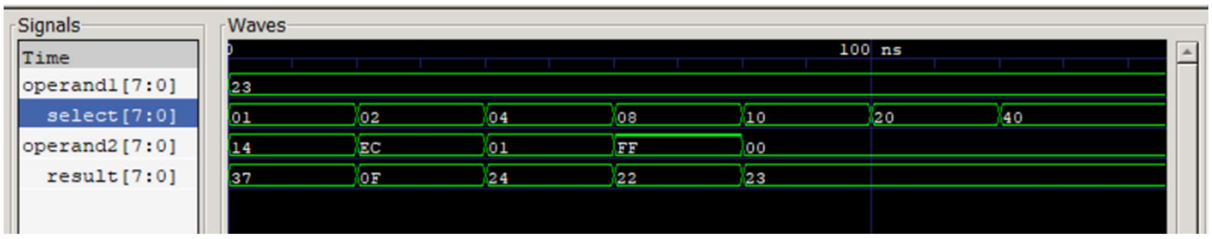
Convert (select, operand1, operand2, instruction)



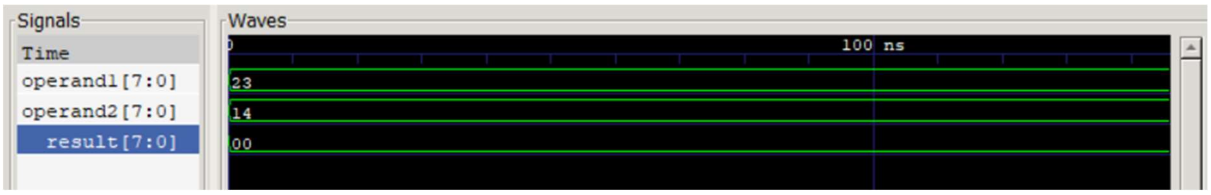
MUX1 (op2, select, operand2)



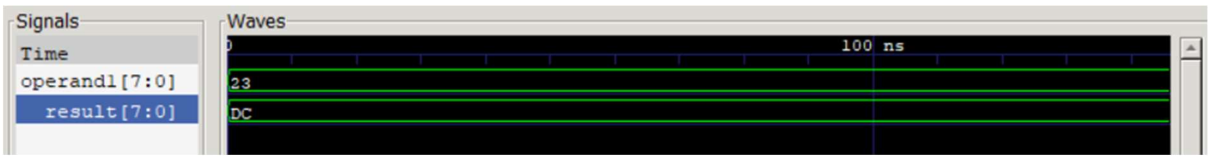
ADD (result, operand1, op2)



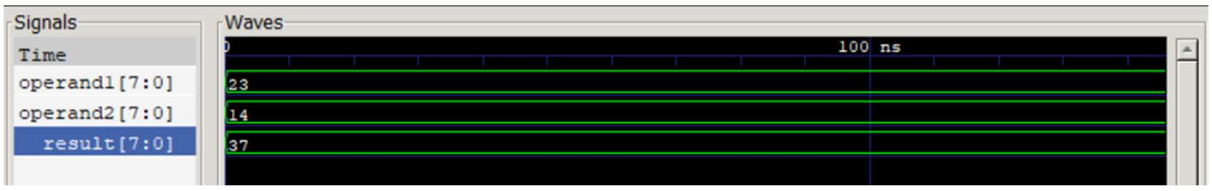
AND8 (result, operand1, operand2)



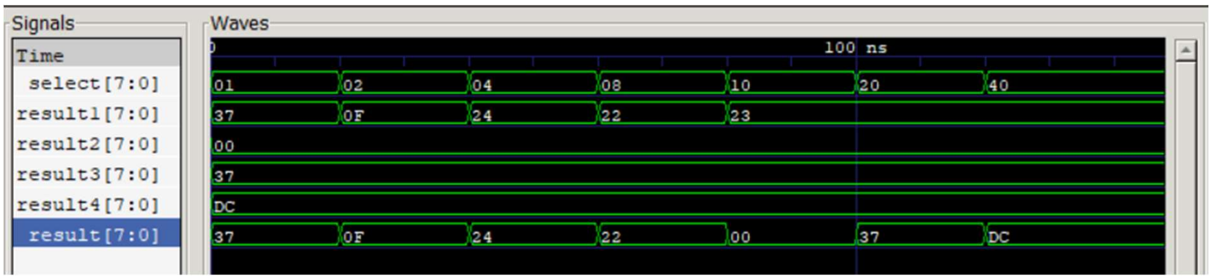
NOT8 (result, operand1)



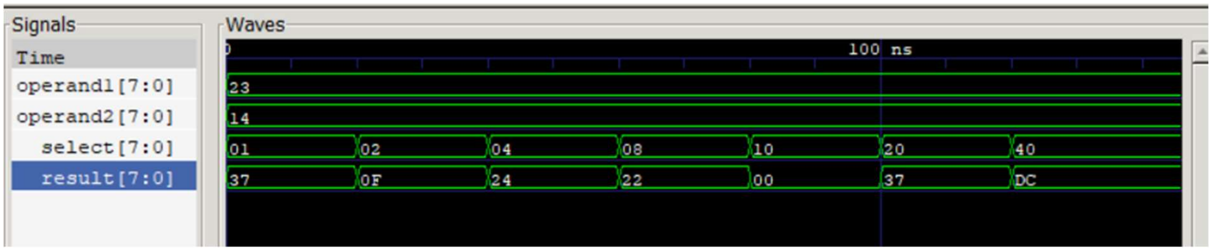
OR8 (result, operand1, operand2)



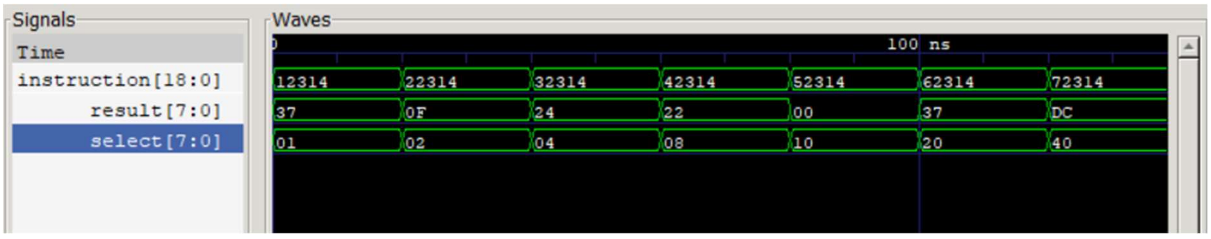
MUX2 (result, result1, result2, result3, result4)



ALU (result, select, operand1, operand2)



CU (result, instruction)



K.Puneeth Kumar,CS21B040