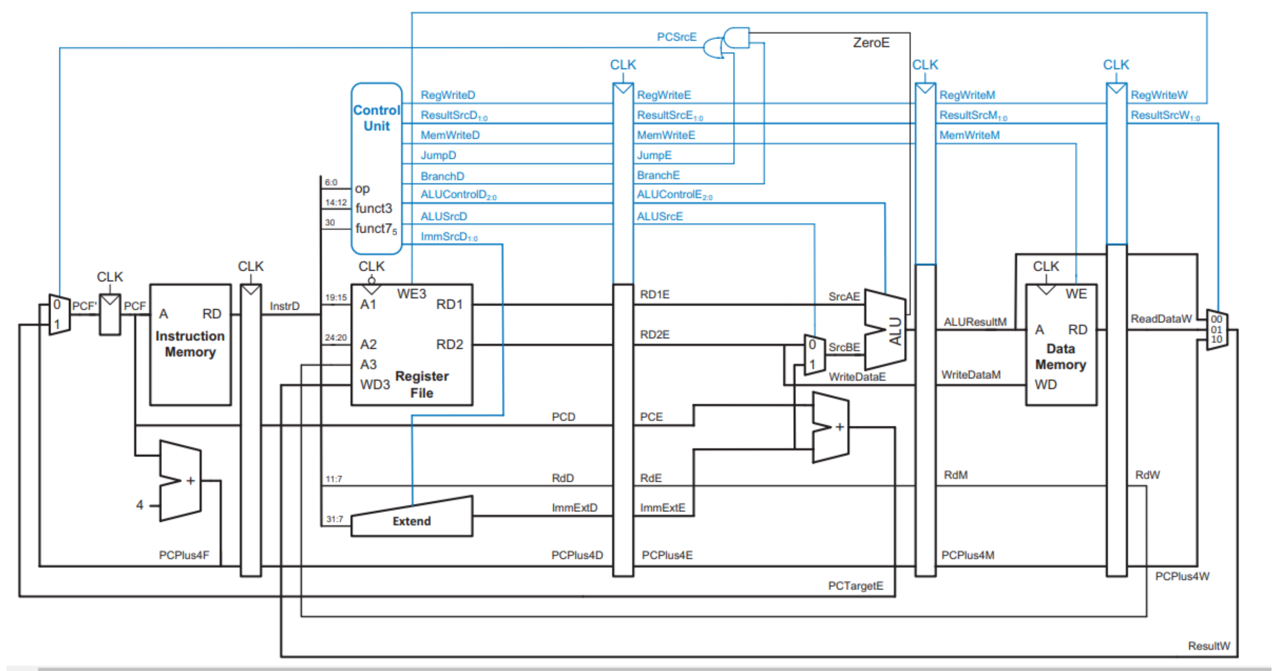
Pipeline Architecture

Datapath:



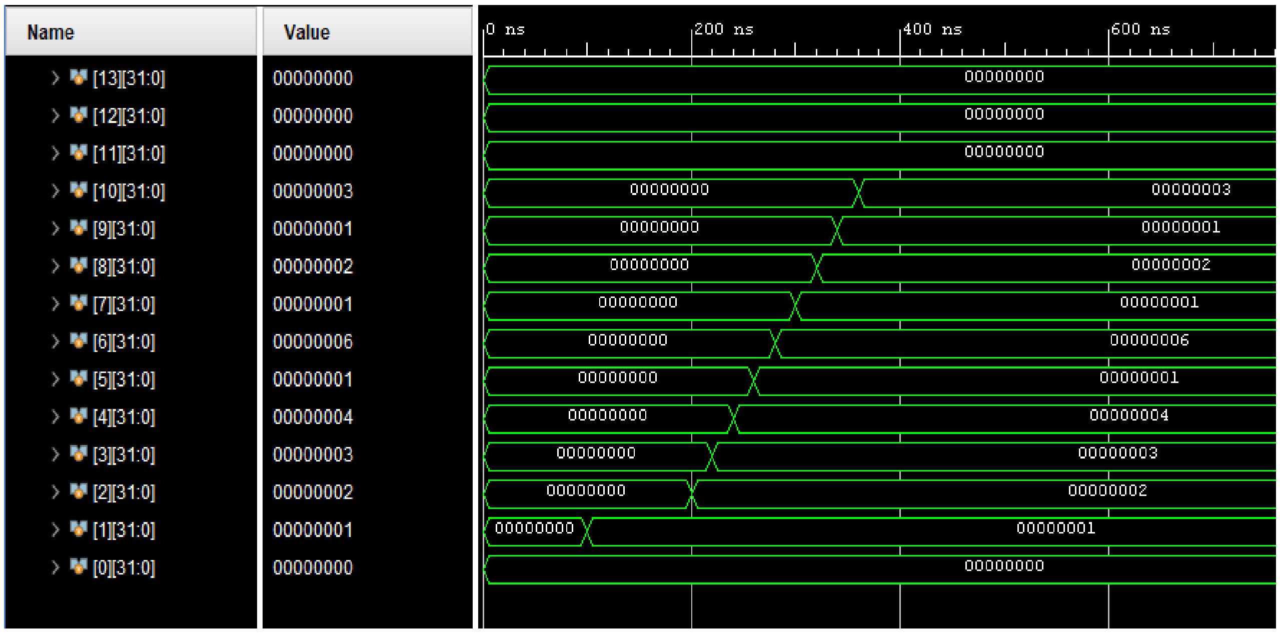
I followed this Datapath and convert my single cycle processor into pipeline architecture by adding four clock driven registers. Then I test it on a instruction memory which contains no hazards.

Testing Instruction Memory

```
32'h00: instruction = 32'h00100093;//addi x1 , x0 , 1
32'h04: instruction = 32'h00200113;//addi x2 , x0 , 2
32'h08: instruction = 32'h00300193;//addi x3 , x0 , 3
32'h0c: instruction = 32'h00400213;//addi x4 , x0 , 4
32'h10: instruction = 32'h401102b3;//sub  x5 , x2 , x1
32'h14: instruction = 32'h00508313;//add  x6 , x1 , 5
32'h18: instruction = 32'h402183b3;//sub  x7, x3 , x2
32'h1c: instruction = 32'h00138413;//addi x8,x7,1
32'h20: instruction = 32'h407404b3;//sub x9,x8,x7
32'h24: instruction = 32'h00940533;//sub x10,x8,x9

default: instruction = 32'h00000000;
        endcase
```

<u>Testbench result:</u>



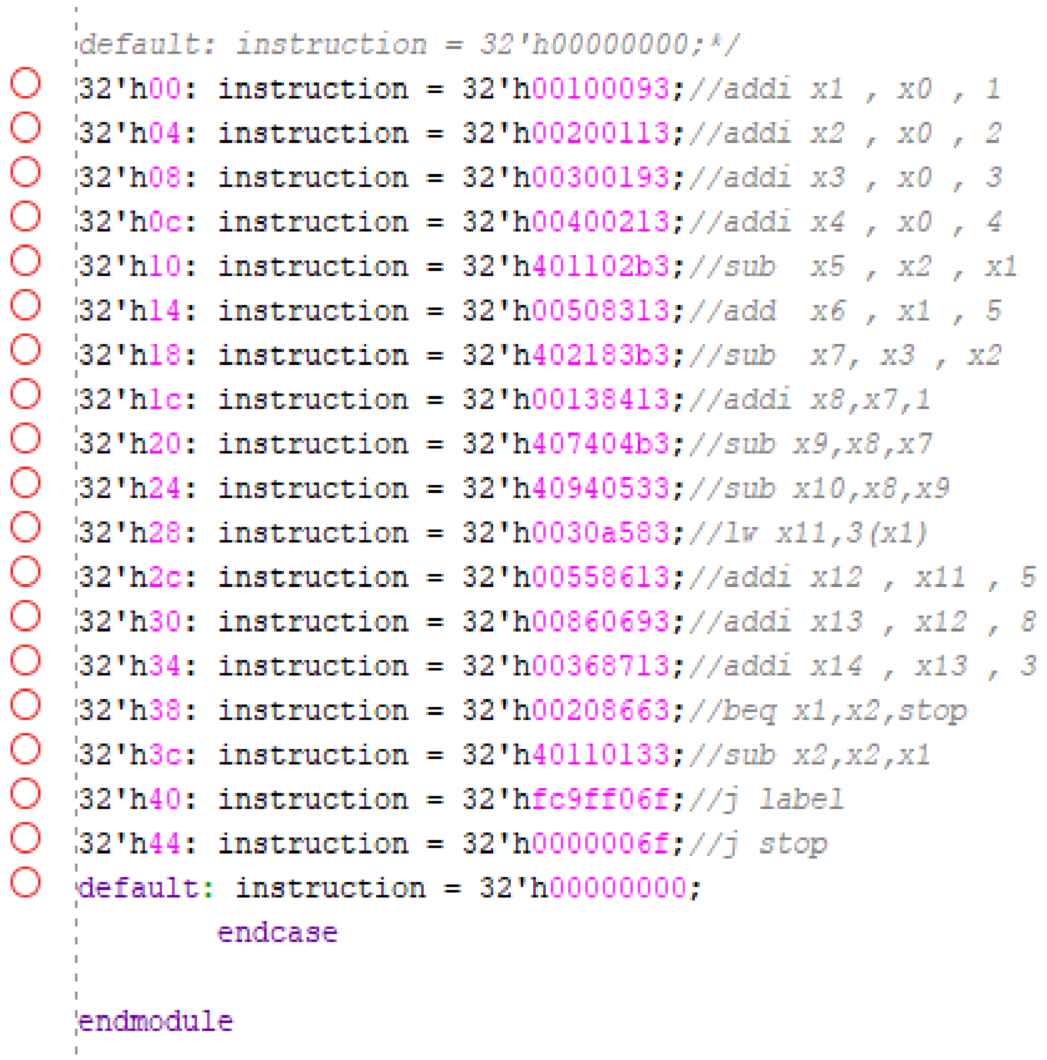
Name	Value
[13][31:0]	00000000
[12][31:0]	00000000
[11][31:0]	00000000
[10][31:0]	00000003
[9][31:0]	00000001
[8][31:0]	00000002
[7][31:0]	00000001
[6][31:0]	00000006
[5][31:0]	00000001
[4][31:0]	00000004
[3][31:0]	00000003
[2][31:0]	00000002
[1][31:0]	00000001
[0][31:0]	00000000

You can check from the test bench waveform after reset become zero first result is written back after five clock cycles. After that the result is written after every clock cycle.

Pipeline Architecture with Hazard Controller

In this part, I added hazard controller unit to control the hazard and test it on a instruction memory that contain hazards.

Testing Instruction Memory:

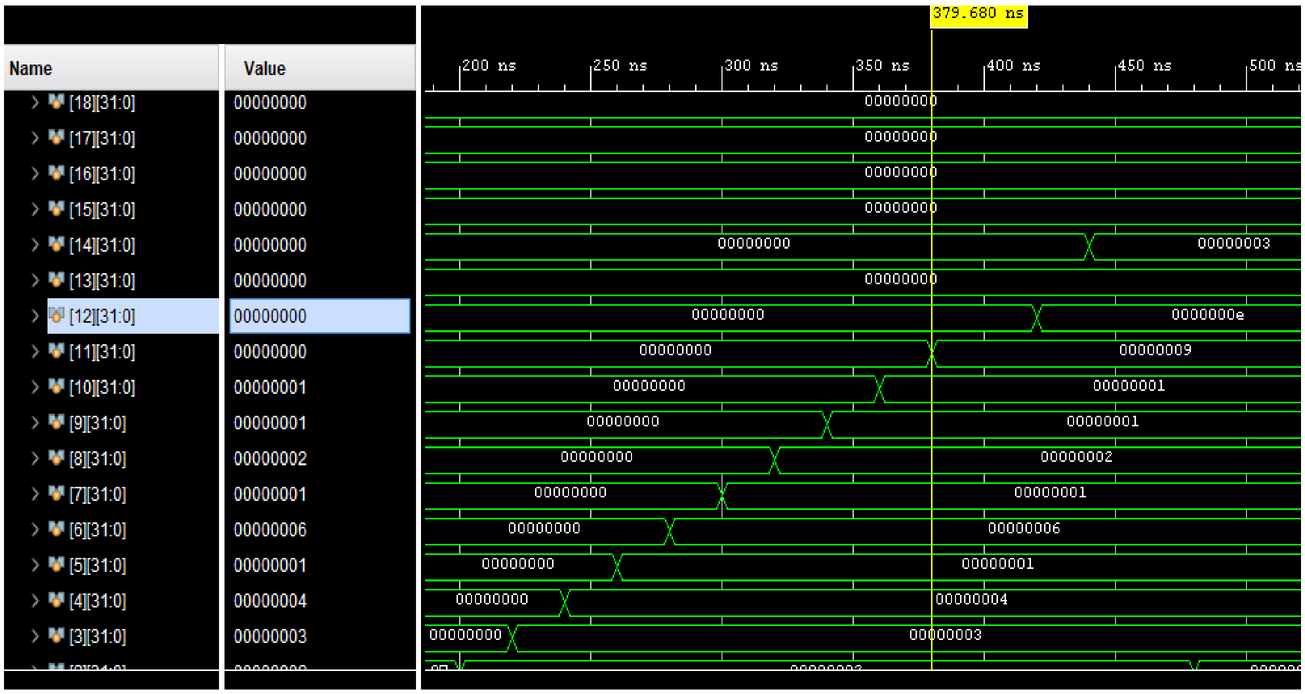


```
default: instruction = 32'h00000000;*/
32'h00: instruction = 32'h00100093;//addi x1 , x0 , 1
32'h04: instruction = 32'h00200113;//addi x2 , x0 , 2
32'h08: instruction = 32'h00300193;//addi x3 , x0 , 3
32'h0c: instruction = 32'h00400213;//addi x4 , x0 , 4
32'h10: instruction = 32'h401102b3;//sub  x5 , x2 , x1
32'h14: instruction = 32'h00508313;//add  x6 , x1 , 5
32'h18: instruction = 32'h402183b3;//sub  x7, x3 , x2
32'h1c: instruction = 32'h00138413;//addi x8,x7,1
32'h20: instruction = 32'h407404b3;//sub x9,x8,x7
32'h24: instruction = 32'h40940533;//sub x10,x8,x9
32'h28: instruction = 32'h0030a583;//lw x11,3(x1)
32'h2c: instruction = 32'h00558613;//addi x12 , x11 , 5
32'h30: instruction = 32'h00860693;//addi x13 , x12 , 8
32'h34: instruction = 32'h00368713;//addi x14 , x13 , 3
32'h38: instruction = 32'h00208663;//beq x1,x2,stop
32'h3c: instruction = 32'h40110133;//sub x2,x2,x1
32'h40: instruction = 32'hfc9ff06f;//j label
32'h44: instruction = 32'h0000006f;//j stop
default: instruction = 32'h00000000;
        endcase

endmodule
```

TestBench result:

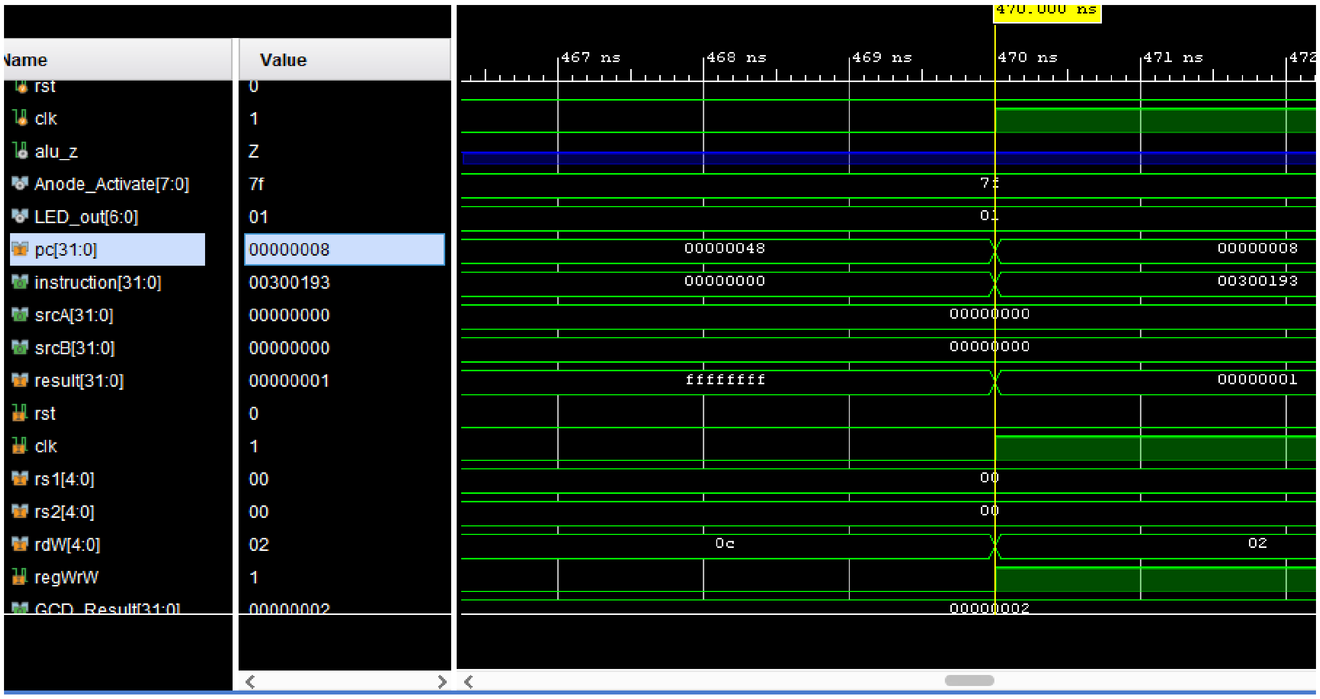
RAW hazard and stalling and flushing:



Name	Value
[18][31:0]	00000000
[17][31:0]	00000000
[16][31:0]	00000000
[15][31:0]	00000000
[14][31:0]	00000000
[13][31:0]	00000000
[12][31:0]	00000000
[11][31:0]	00000000
[10][31:0]	00000001
[9][31:0]	00000001
[8][31:0]	00000002
[7][31:0]	00000001
[6][31:0]	00000006
[5][31:0]	00000001
[4][31:0]	00000004
[3][31:0]	00000003

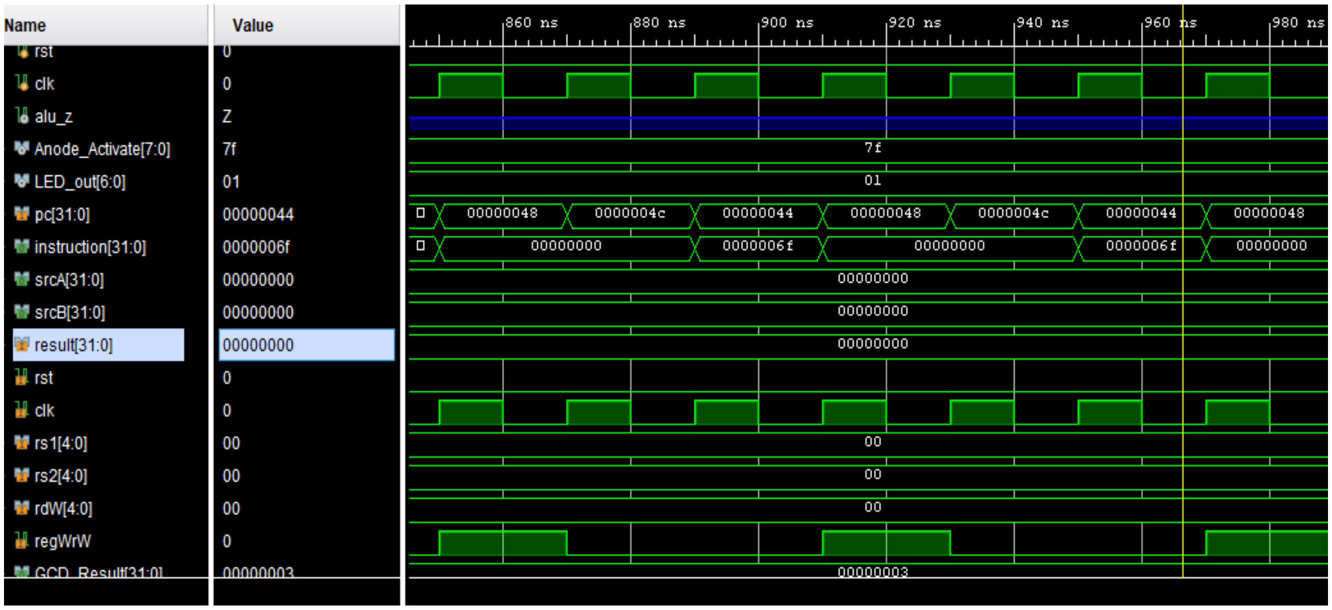
Raw hazards are solved by forwarding and there is a one cycle stall when load instruction came. When result is written back to x11 after it there is a one cycle stall due to load instruction.

Controll Hazards:



Name	Value
rst	0
clk	1
alu_z	Z
Anode_Activate[7:0]	7f
LED_out[6:0]	01
pc[31:0]	00000008
instruction[31:0]	00300193
srcA[31:0]	00000000
srcB[31:0]	00000000
result[31:0]	00000001
rst	0
clk	1
rs1[4:0]	00
rs2[4:0]	00
rdW[4:0]	02
regWrW	1
GCD_Result[31:0]	00000002

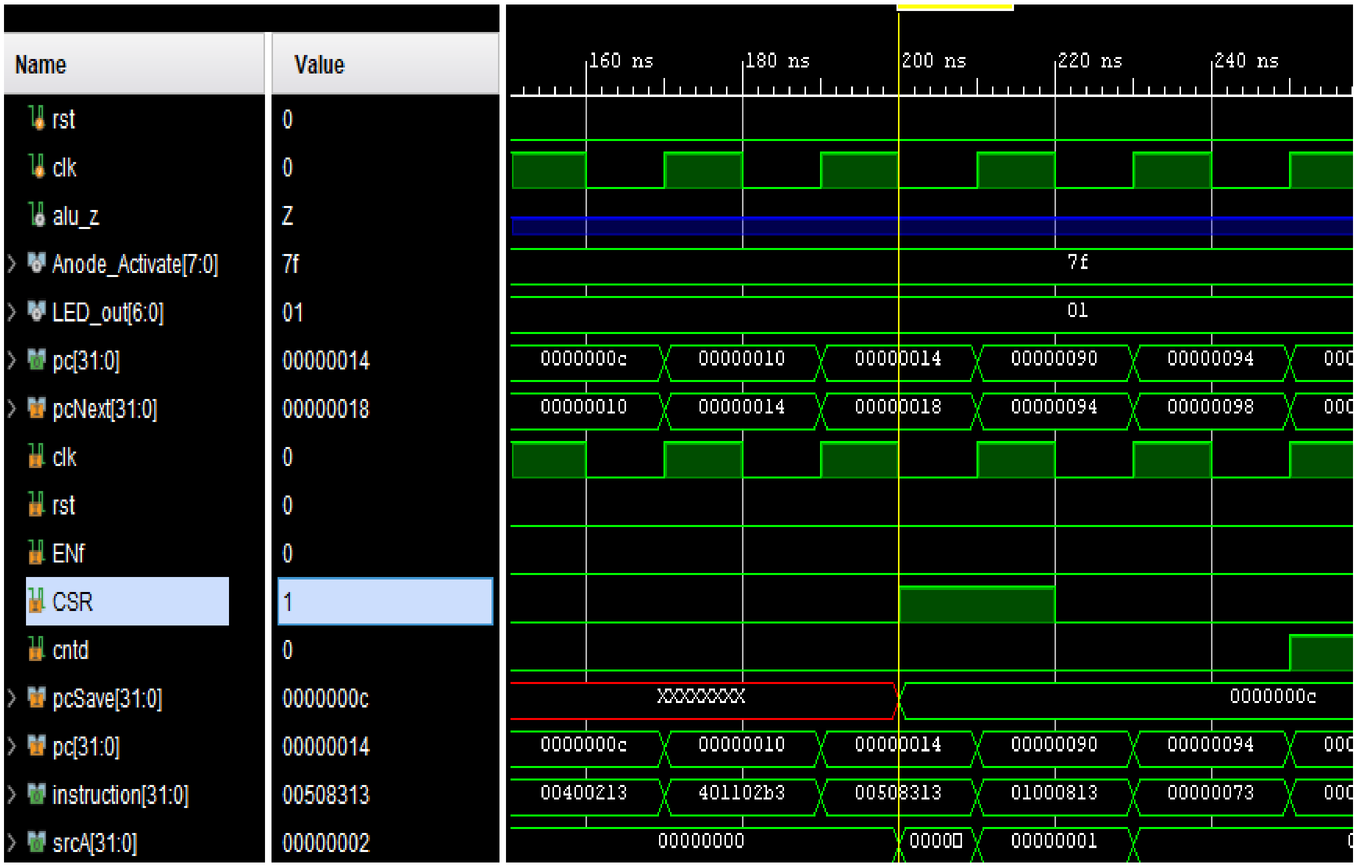
It also supports jump instruction, pc jump from 48 to 08 hen jump instruction came.



Name	Value
rst	0
clk	0
alu_z	Z
Anode_Activate[7:0]	7f
LED_out[6:0]	01
pc[31:0]	00000044
instruction[31:0]	0000006f
srcA[31:0]	00000000
srcB[31:0]	00000000
result[31:0]	00000000
rst	0
clk	0
rs1[4:0]	00
rs2[4:0]	00
rdW[4:0]	00
regWrW	0
GCD_Result[31:0]	00000003

It also supports branch instruction when beq instruction came it stops as shown in instruction memory.

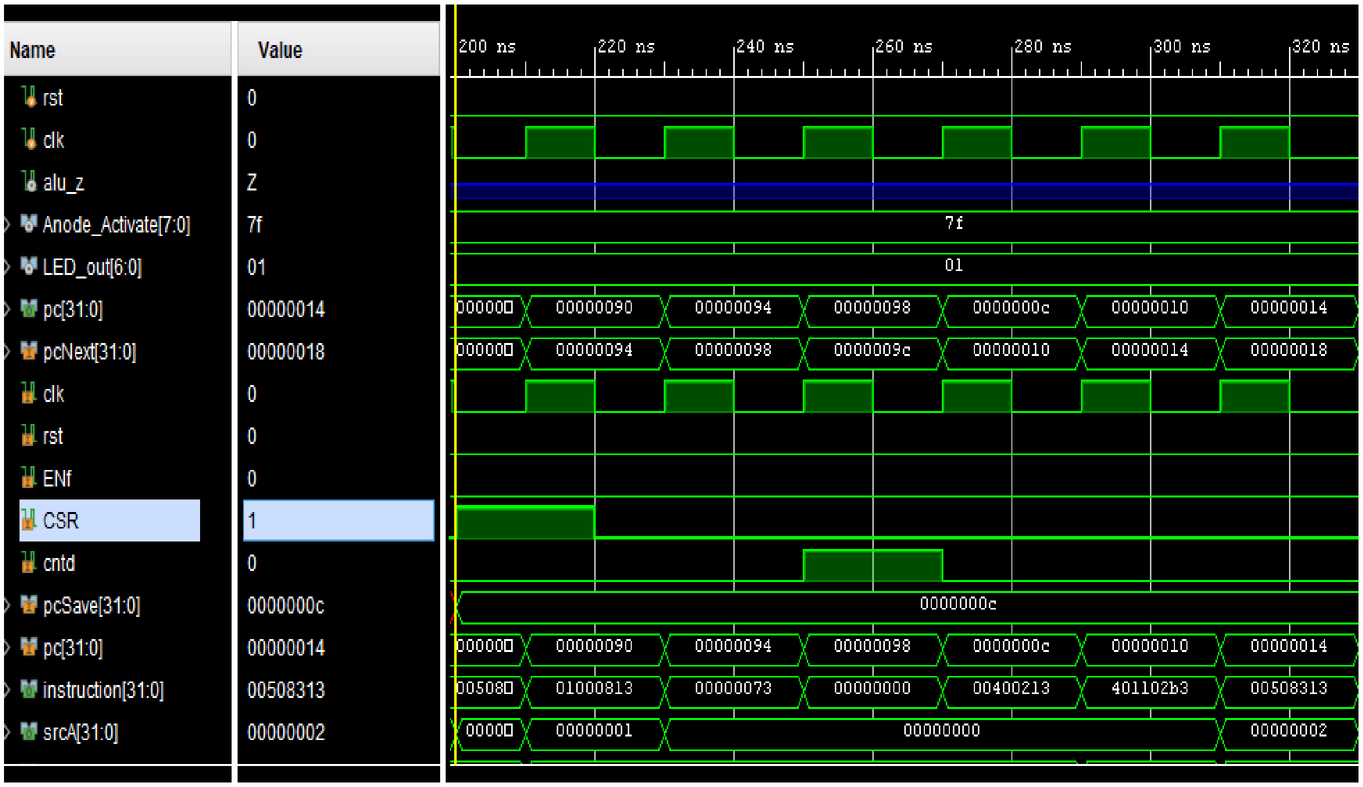
TestBench result:



Name	Value
rst	0
clk	0
alu_z	Z
Anode_Activate[7:0]	7f
LED_out[6:0]	01
pc[31:0]	00000014
pcNext[31:0]	00000018
clk	0
rst	0
ENf	0
CSR	1
cntd	0
pcSave[31:0]	0000000c
pc[31:0]	00000014
instruction[31:0]	00508313
srcA[31:0]	00000002

160 ns
180 ns
200 ns
220 ns
240 ns
7f
01
0000000c 00000010 00000014 00000090 00000094
00000010 00000014 00000018 00000094 00000098
XXXXXXXX 0000000c
0000000c 00000010 00000014 00000090 00000094
00400213 401102b3 00508313 01000813 00000073
00000000 00000 00000001

When CSR flag is turned on pc jumps to a handler hard coded in instruction memory and pc of execution is saved in pcSave.



Name	Value
rst	0
clk	0
alu_z	Z
Anode_Activate[7:0]	7f
LED_out[6:0]	01
pc[31:0]	00000014
pcNext[31:0]	00000018
clk	0
rst	0
ENf	0
CSR	1
cntd	0
pcSave[31:0]	0000000c
pc[31:0]	00000014
instruction[31:0]	00508313
srcA[31:0]	00000002

After the execution of intrupt handler instruction it return back to its normal execution.