CO224 Lab5 - Building a Simple Processor
Part 5 - Extended ISA

Additional Instructions supported

- **bne**
- **ror**
- **srl**
- **sll**
- **sra**

Branch Not Equal Instruction (bne)

INSTRUCTION ENCODING:

OPCODE	IMM	RT	RS
Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0

OPCODE - to identify the ror instruction from other instructions
IMM - branch target offset
RT - first register where the value to be read is found
RS - second register where the value to be read is found

OPCODE of the ror instruction : 0000_1000

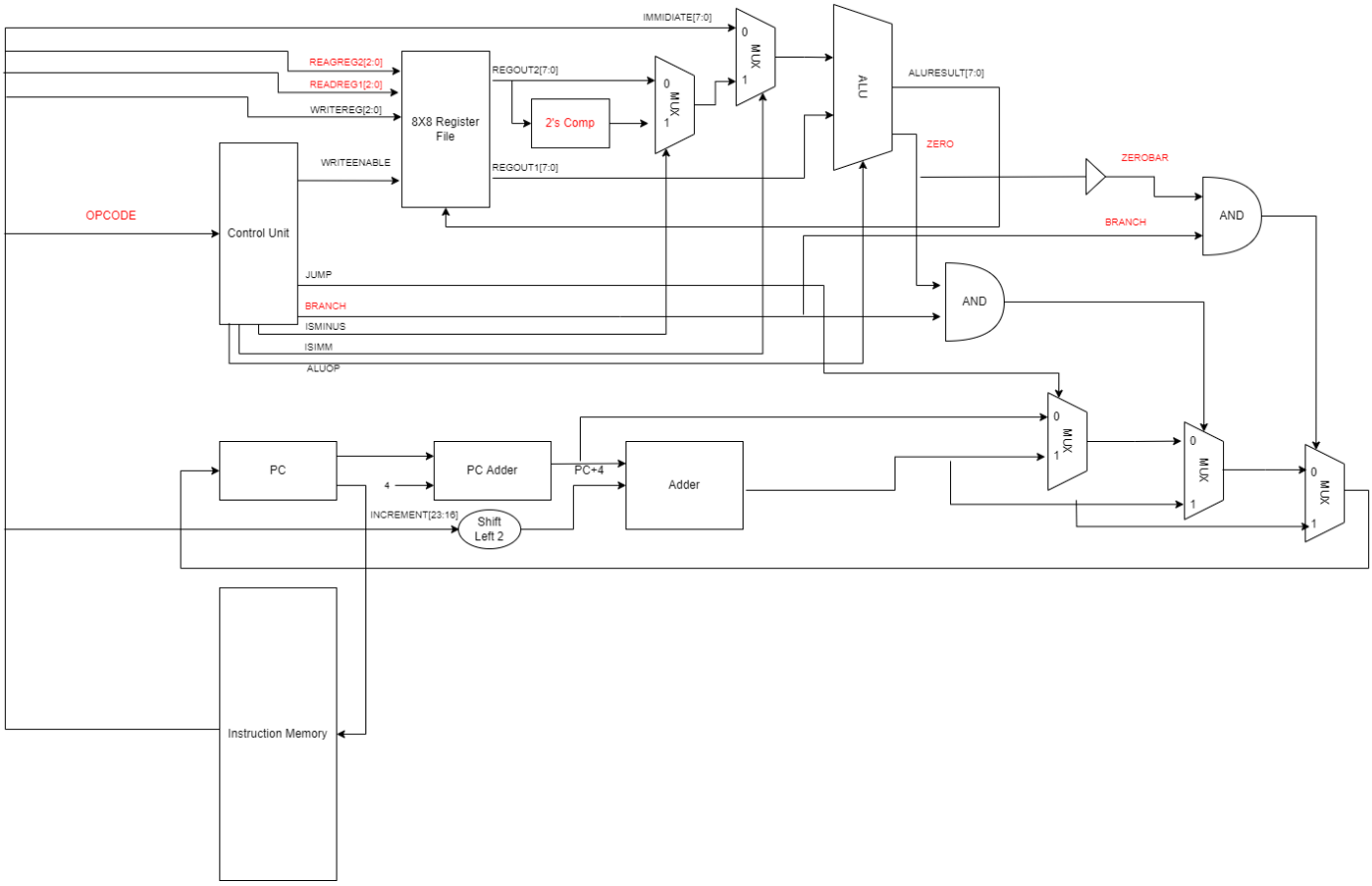
Example :
bne 0x02 1 2 (if values in reg 1 and reg 2 are not equal branch 2 instructions forward by manipulating the PC)

bne	0x02	1	2
00001000	00000010	00000001	00000010

Instruction :**00001000000000100000000100000010**
ALUOP signal given : **100**

PROCEDURE :

Here we have assigned another 3rd mux and inputs given to that are
IN 1 -> output from 2nd mux
IN 2-> PC +4
Selection will be done according to the value given by (BRANCH & ZEROBAR)



Rotate Right Instruction (ror)

INSTRUCTION ENCODING :

OPCODE	RD	RT	IMM
Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0

OPCODE - to identify the ror instruction from other instructions
RD - destination register where the result is stored after rotating right
RT - source where the value to be rotated is found
IMM - rotated amount (Immediate value)

OPCODE of the ror instruction : 0000_1001

Example :
ror 4 1 0x02 (apply rotate right 2 times on value in register 1, and place the result in register 4)

ror	4	1	0x02
00001001	00000100	00000001	00000010

Instruction : **00001001000001000000000100000010**
ALUOP signal given : **101**

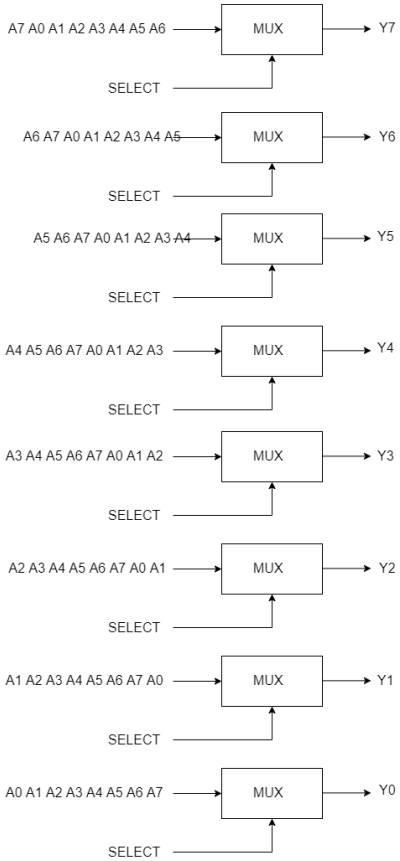
PROCEDURE :

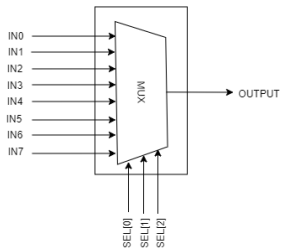
Here, 8 8x1 multiplexers are used.Each multiplexer is corresponding to one bit of the output signal.
We have restricted the shifting value to 7,so there's only 8 instances where shifting can happen. When there is a shift corresponding bit is selected from the each multiplexer resulting the output.

Input - $A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

s1	s2	s3	Shift	Output							
				Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	1	1	A_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1
0	1	0	2	A_1	A_0	A_7	A_6	A_5	A_4	A_3	A_2
0	1	1	3	A_2	A_1	A_0	A_7	A_6	A_5	A_4	A_3
1	0	0	4	A_3	A_2	A_1	A_0	A_7	A_6	A_5	A_4
1	0	1	5	A_4	A_3	A_2	A_1	A_0	A_7	A_6	A_5
1	1	0	6	A_5	A_4	A_3	A_2	A_1	A_0	A_7	A_6
1	1	1	7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	A_7

Output - $Y_7\ Y_6\ Y_5\ Y_4\ Y_3\ Y_2\ Y_1\ Y_0$





Verilog module for mux:

```
module MUX8x1(OUT,IN0,IN1,IN2,IN3,IN4,IN5,IN6,IN7,SEL);
    input IN0,IN1,IN2,IN3,IN4,IN5,IN6,IN7;
    input [2:0] SEL;
    output reg OUT;

    always@(*) begin
        case(SEL)
            3'b000 : OUT = IN0;
            3'b001 : OUT = IN1;
            3'b010 : OUT = IN2;
            3'b011 : OUT = IN3;
            3'b100 : OUT = IN4;
            3'b101 : OUT = IN5;
            3'b110 : OUT = IN6;
            3'b111 : OUT = IN7;
        endcase
    end
endmodule
```

Verilog module for ror instruction:

```
module ROR(OUTPUT,IN,SHIFT); //module to rotate right by  given SHIFT value
    input[7:0] IN,SHIFT;
    output[7:0] OUTPUT;
    reg [2:0] SEL;
    wire[7:0] OUT;

    always@(SHIFT) begin
    case(SHIFT)
    //here depending on the shifting value it assigns values to select signals
        8'b000:
            SEL = 3'b000;
        8'b001 :
            SEL = 3'b001;
        8'b010 :
            SEL = 3'b010;
        8'b011 :
            SEL = 3'b011;
        8'b100 :
            SEL = 3'b100;
        8'b101 :
            SEL = 3'b101;
        8'd110 :
            SEL = 3'b110;
        8'd111 :
            SEL = 3'b111;
    endcase
    end

    //here mux 8 represents least significant bit of the value after right rotated and
    //mux 1 represents most significant bit of the value after right rotated
    //bits "shifted off" one end(right) are put at the other(left) end

    MUX8x1 mux1(OUT[7],IN[7],IN[0],IN[1],IN[2],IN[3],IN[4],IN[5],IN[6],SEL);
    MUX8x1 mux2(OUT[6],IN[6],IN[7],IN[0],IN[1],IN[2],IN[3],IN[4],IN[5],SEL);
    MUX8x1 mux3(OUT[5],IN[5],IN[6],IN[7],IN[0],IN[1],IN[2],IN[3],IN[4],SEL);
    MUX8x1 mux4(OUT[4],IN[4],IN[5],IN[6],IN[7],IN[0],IN[1],IN[2],IN[3],SEL);
    MUX8x1 mux5(OUT[3],IN[3],IN[4],IN[5],IN[6],IN[7],IN[0],IN[1],IN[2],SEL);
    MUX8x1 mux6(OUT[2],IN[2],IN[3],IN[4],IN[5],IN[6],IN[7],IN[0],IN[1],SEL);
    MUX8x1 mux7(OUT[1],IN[1],IN[2],IN[3],IN[4],IN[5],IN[6],IN[7],IN[0],SEL);
    MUX8x1 mux8(OUT[0],IN[0],IN[1],IN[2],IN[3],IN[4],IN[5],IN[6],IN[7],SEL);

    assign #2 OUTPUT = OUT;
endmodule
```

Since we have used the same ALUOP signal for 3 instructions (sll , slr, sra) , in order to differentiate them we have generated a 2-bit signal (SHIFT) from the control unit and they are as follows.

Instruction	Control signal
sll	01
srl	10
sra	11
other	00

In the ALU, another input is given in addition to previous inputs which is the SHIFT signal coming from the control unit.
According to the ALUOP and the SHIFT signal, results are assigned to the output.

Logical Shift Left Instruction (sll)

INSTRUCTION ENCODING :

OPCODE	RD	RT	IMM
Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0

OPCODE - to identify the sll instruction from other instructions
RD - destination register where the result is stored after logical shifting left
RT - source where the value to be left shifted is found
IMM - shifted amount (Immediate value)

OPCODE of the sll instruction : 0000_1010

Example :
sll 4 1 0x02 (apply logical shift left 2 times on value in register 1, and place the result in register 4)

sll	4	1	0x02
00001010	00000100	00000001	00000010

Instruction : **00001010000001000000000100000010**
ALUOP signal given : **110**

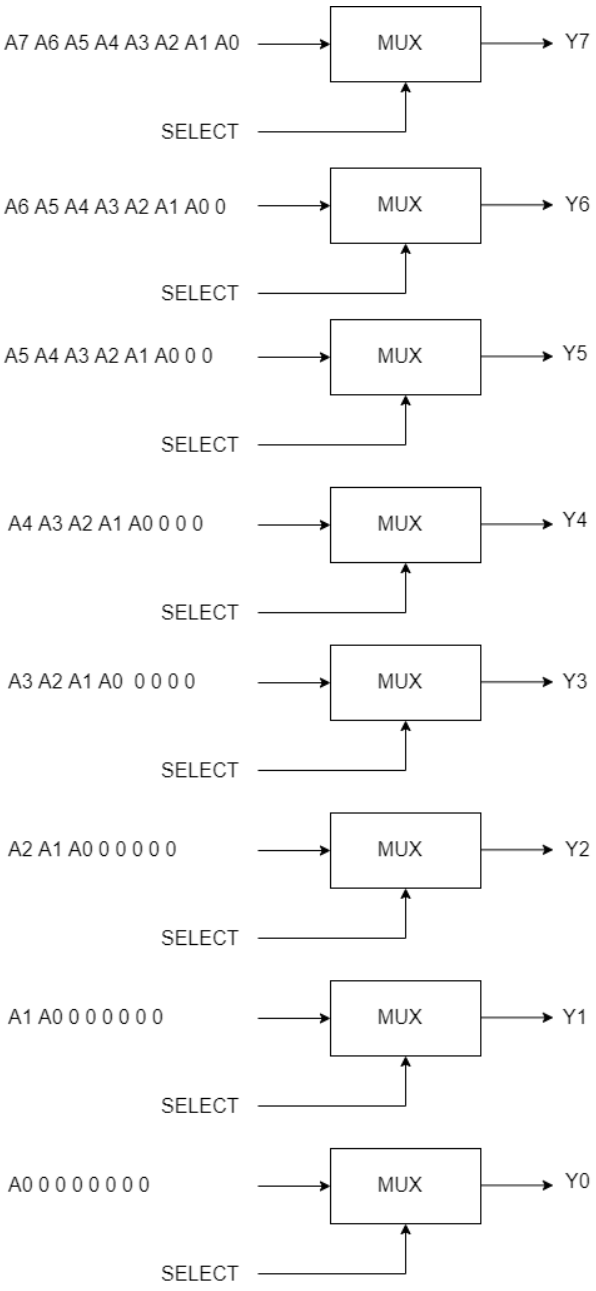
PROCEDURE :

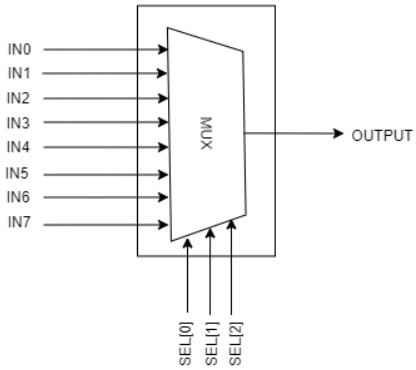
Here, 8 8x1 multiplexers are used.Each multiplexer is corresponding to one bit of the output signal.
We have restricted the shifting value to 7,so there's only 8 instances where shifting can happen. When there is a shift the corresponding bit is selected from each multiplexer resulting in the output.

Input - $A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

s1	s2	s3	Shift	Output							
				Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	1	1	A_6	A_5	A_4	A_3	A_2	A_1	A_0	0
0	1	0	2	A_5	A_4	A_3	A_2	A_1	A_0	0	0
0	1	1	3	A_4	A_3	A_2	A_1	A_0	0	0	0
1	0	0	4	A_3	A_2	A_1	A_0	0	0	0	0
1	0	1	5	A_2	A_1	A_0	0	0	0	0	0
1	1	0	6	A_1	A_0	0	0	0	0	0	0
1	1	1	7	A_0	0	0	0	0	0	0	0

Output - $Y_7\ Y_6\ Y_5\ Y_4\ Y_3\ Y_2\ Y_1\ Y_0$





Verilog module for sll instruction:

Part5.v | ror.v | sll.v | sra.v | srl.v | alu.v | instr_mem.mem | Expected_output.txt

```
module SLL(OUTPUT,IN,SHIFT); //module to logical shift left by  given SHIFT value
    input[7:0] IN,SHIFT;    //shifting value will be maximum 7 for 8-bit value
    output[7:0] OUTPUT;
    reg [2:0]SEL;
    wire [7:0] OUT;

    always@(SHIFT) begin
    case(SHIFT)
    //here depending on the shifting value it assigns values to select signals
        8'b000:
            SEL = 3'b000;
        8'b001 :
            SEL = 3'b001;
        8'b010 :
            SEL = 3'b010;
        8'b011 :
            SEL = 3'b011;
        8'b100 :
            SEL = 3'b100;
        8'b101 :
            SEL = 3'b101;
        8'd110 :
            SEL = 3'b110;
        8'd111 :
            SEL = 3'b111;
    endcase
    end
    //here mux 8 represents least significant bit of the value after left shifted and
    //mux 1 represents most significant bit of the value after left shifted
    //after shifting to the left it's padding out with 0's(least significant bits)


    MUX8x1 mux1(OUT[7],IN[7],IN[6],IN[5],IN[4],IN[3],IN[2],IN[1],IN[0],SEL);
    MUX8x1 mux2(OUT[6],IN[6],IN[5],IN[4],IN[3],IN[2],IN[1],IN[0],1'b0,SEL);
    MUX8x1 mux3(OUT[5],IN[5],IN[4],IN[3],IN[2],IN[1],IN[0],1'b0,1'b0,SEL);
    MUX8x1 mux4(OUT[4],IN[4],IN[3],IN[2],IN[1],IN[0],1'b0,1'b0,1'b0,SEL);
    MUX8x1 mux5(OUT[3],IN[3],IN[2],IN[1],IN[0],1'b0,1'b0,1'b0,1'b0,SEL);
    MUX8x1 mux6(OUT[2],IN[2],IN[1],IN[0],1'b0,1'b0,1'b0,1'b0,1'b0,SEL);
    MUX8x1 mux7(OUT[1],IN[1],IN[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,SEL);
    MUX8x1 mux8(OUT[0],IN[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,SEL);

    assign #2 OUTPUT = OUT;
endmodule
```

Logical Shift Right Instruction (srl)

INSTRUCTION ENCODING :

OPCODE	RD	RT	IMM
Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0

OPCODE - to identify the srl instruction from other instructions
RD - destination register where the result is stored after logical shifting right
RT - source where the value to be right shifted is found
IMM - shifted amount (Immediate value)

OPCODE of the srl instruction : 0000_1011

Example :
srl 4 1 0x02 (apply logical shift right 2 times on value in register 1, and place the result in register 4)

srl	4	1	0x02
00001011	00000100	00000001	00000010

Instruction : **00001011000001000000000100000010**
ALUOP signal given : **110**

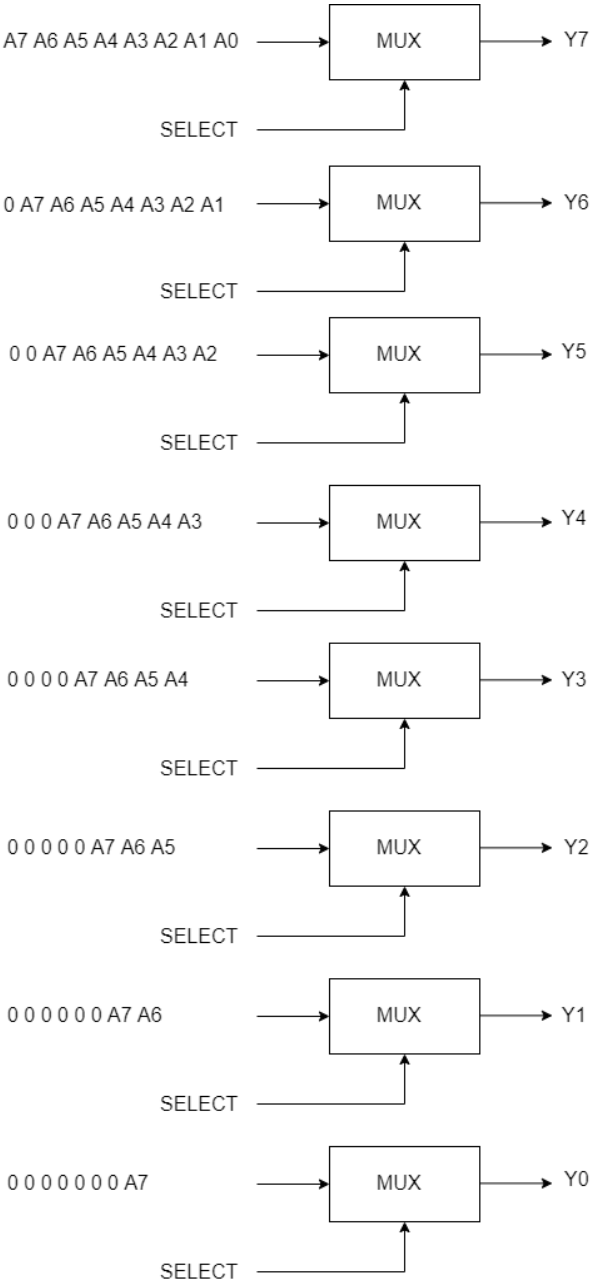
PROCEDURE :

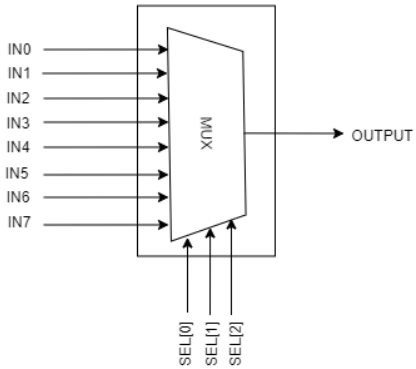
Here, 8 8x1 multiplexers are used.Each multiplexer is corresponding to one bit of the output signal.
We have restricted the shifting value to 7,so there's only 8 instances where shifting can happen. When there is a shift the corresponding bit is selected from each multiplexer resulting in the output.

Input - $A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

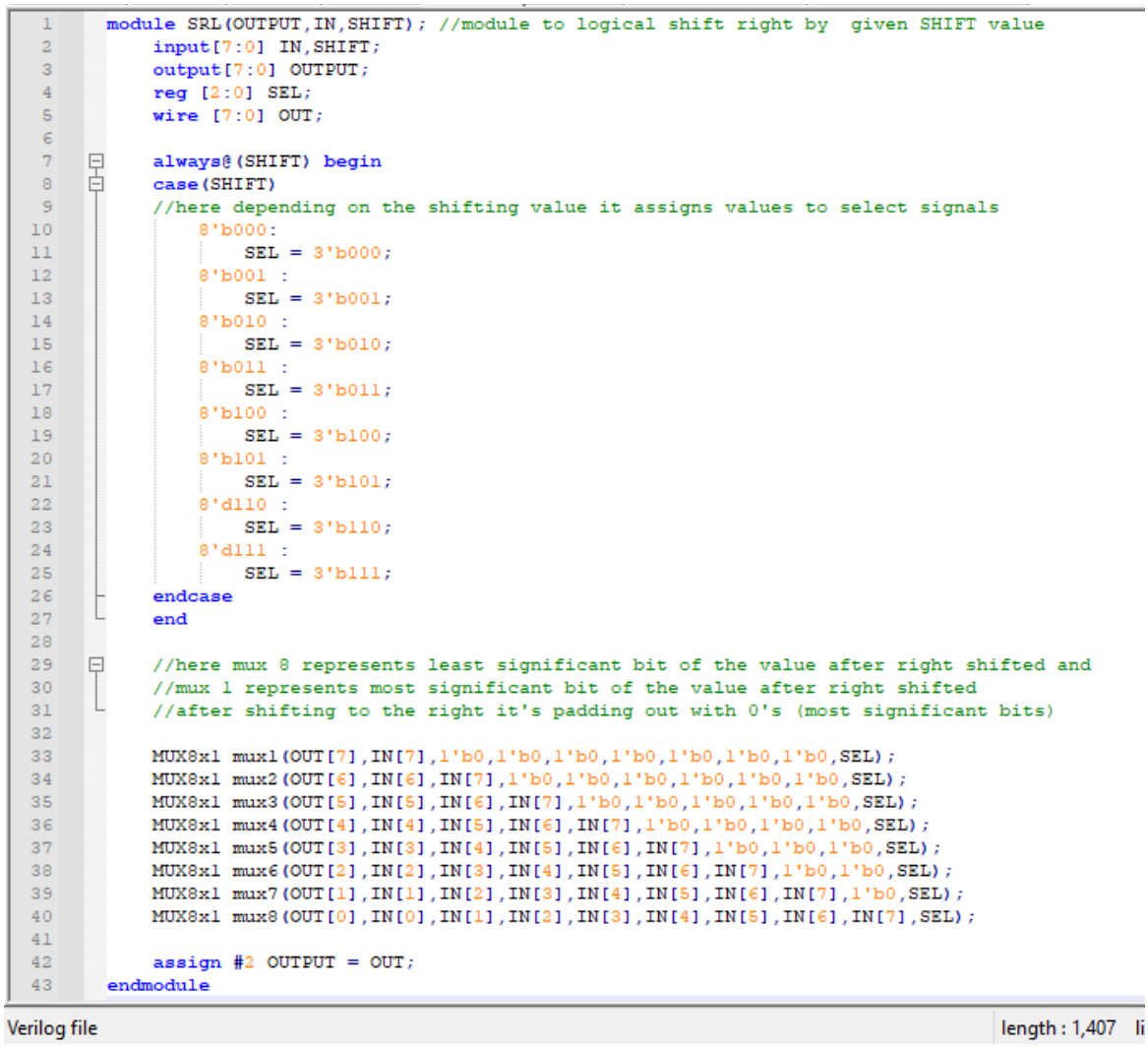
s1	s2	s3	Shift	Output							
				Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	1	1	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1
0	1	0	2	0	0	A_7	A_6	A_5	A_4	A_3	A_2
0	1	1	3	0	0	0	A_7	A_6	A_5	A_4	A_3
1	0	0	4	0	0	0	0	A_7	A_6	A_5	A_4
1	0	1	5	0	0	0	0	0	A_7	A_6	A_5
1	1	0	6	0	0	0	0	0	0	A_7	A_6
1	1	1	7	0	0	0	0	0	0	0	A_7

Output - $Y_7\ Y_6\ Y_5\ Y_4\ Y_3\ Y_2\ Y_1\ Y_0$





Verilog module for srl instruction:



```
module SRL(OUTPUT,IN,SHIFT); //module to logical shift right by  given SHIFT value
    input[7:0] IN,SHIFT;
    output[7:0] OUTPUT;
    reg [2:0] SEL;
    wire [7:0] OUT;

    always@(SHIFT) begin
    case(SHIFT)
    //here depending on the shifting value it assigns values to select signals
        8'b000:
            SEL = 3'b000;
        8'b001 :
            SEL = 3'b001;
        8'b010 :
            SEL = 3'b010;
        8'b011 :
            SEL = 3'b011;
        8'b100 :
            SEL = 3'b100;
        8'b101 :
            SEL = 3'b101;
        8'd110 :
            SEL = 3'b110;
        8'd111 :
            SEL = 3'b111;
    endcase
    end

    //here mux 8 represents least significant bit of the value after right shifted and
    //mux 1 represents most significant bit of the value after right shifted
    //after shifting to the right it's padding out with 0's (most significant bits)

    MUX8x1 mux1(OUT[7],IN[7],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,SEL);
    MUX8x1 mux2(OUT[6],IN[6],IN[7],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,SEL);
    MUX8x1 mux3(OUT[5],IN[5],IN[6],IN[7],1'b0,1'b0,1'b0,1'b0,1'b0,SEL);
    MUX8x1 mux4(OUT[4],IN[4],IN[5],IN[6],IN[7],1'b0,1'b0,1'b0,1'b0,SEL);
    MUX8x1 mux5(OUT[3],IN[3],IN[4],IN[5],IN[6],IN[7],1'b0,1'b0,1'b0,SEL);
    MUX8x1 mux6(OUT[2],IN[2],IN[3],IN[4],IN[5],IN[6],IN[7],1'b0,1'b0,SEL);
    MUX8x1 mux7(OUT[1],IN[1],IN[2],IN[3],IN[4],IN[5],IN[6],IN[7],1'b0,SEL);
    MUX8x1 mux8(OUT[0],IN[0],IN[1],IN[2],IN[3],IN[4],IN[5],IN[6],IN[7],SEL);

    assign #2 OUTPUT = OUT;
endmodule
```

Verilog file length : 1,407 li

Arithmetical Shift right Instruction (sra)

INSTRUCTION ENCODING :

OPCODE	RD	RT	IMM
Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0

OPCODE - to identify the sra instruction from other instructions
RD - destination register where the result is stored after arithmetical shifting right
RT - source where the value to be right shifted is found
IMM - shifted amount (Immediate value)

OPCODE of the srl instruction : 0000_1100

Example :
sra 4 1 0x02 (apply logical shift right 2 times on value in register 1, and place the result in register 4)

sra	4	1	0x02
00001100	00000100	00000001	00000010

Instruction : **00001100000001000000000100000010**
ALUOP signal given : **110**

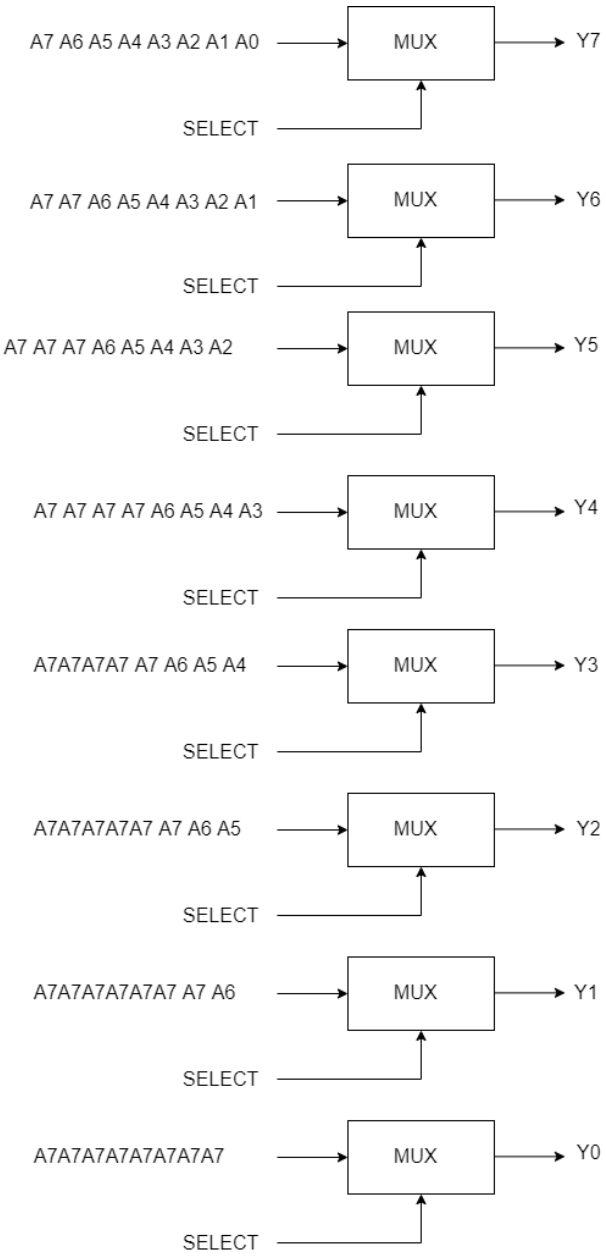
PROCEDURE :

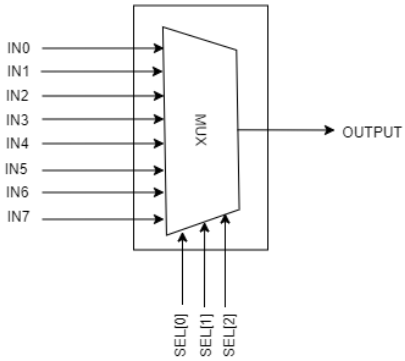
Here, 8 8x1 multiplexers are used.Each multiplexer is corresponding to one bit of the output signal.
We have restricted the shifting value to 7,so there's only 8 instances where shifting can happen. When there is a shift the corresponding bit is selected from each multiplexer resulting in the output.

Input - $A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

s1	s2	s3	Shift	Output							
				Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	1	1	A_7	A_7	A_6	A_5	A_4	A_3	A_2	A_1
0	1	0	2	A_7	A_7	A_7	A_6	A_5	A_4	A_3	A_2
0	1	1	3	A_7	A_7	A_7	A_7	A_6	A_5	A_4	A_3
1	0	0	4	A_7	A_7	A_7	A_7	A_7	A_6	A_5	A_4
1	0	1	5	A_7	A_7	A_7	A_7	A_7	A_7	A_6	A_5
1	1	0	6	A_7	A_7	A_7	A_7	A_7	A_7	A_7	A_6
1	1	1	7	A_7	A_7	A_7	A_7	A_7	A_7	A_7	A_7

Output - $Y_7\ Y_6\ Y_5\ Y_4\ Y_3\ Y_2\ Y_1\ Y_0$





Verilog module for sra instruction:

```
module SRA(OUTPUT,IN,SHIFT); //module to arithmatical shift right by  given SHIFT value
    input[7:0] IN,SHIFT;
    output[7:0] OUTPUT;
    reg [2:0] SEL;
    wire [7:0] OUT;

    always@(SHIFT) begin
    case(SHIFT)
    //here depending on the shifting value it assigns values to select signals
        8'b000:
            SEL = 3'b000;
        8'b001 :
            SEL = 3'b001;
        8'b010 :
            SEL = 3'b010;
        8'b011 :
            SEL = 3'b011;
        8'b100 :
            SEL = 3'b100;
        8'b101 :
            SEL = 3'b101;
        8'd110 :
            SEL = 3'b110;
        8'd111 :
            SEL = 3'b111;
    endcase
    end

    //here mux 8 represents least significant bit of the value after right shifted and
    //mux 1 represents most significant bit of the value after right shifted
    //after shifting to the right it's padding out with the value of its most significant bit

    MUX8x1 mux1(OUT[7],IN[7],IN[7],IN[7],IN[7],IN[7],IN[7],IN[7],IN[7],SEL);
    MUX8x1 mux2(OUT[6],IN[6],IN[7],IN[7],IN[7],IN[7],IN[7],IN[7],IN[7],SEL);
    MUX8x1 mux3(OUT[5],IN[5],IN[6],IN[7],IN[7],IN[7],IN[7],IN[7],IN[7],SEL);
    MUX8x1 mux4(OUT[4],IN[4],IN[5],IN[6],IN[7],IN[7],IN[7],IN[7],IN[7],SEL);
    MUX8x1 mux5(OUT[3],IN[3],IN[4],IN[5],IN[6],IN[7],IN[7],IN[7],IN[7],SEL);
    MUX8x1 mux6(OUT[2],IN[2],IN[3],IN[4],IN[5],IN[6],IN[7],IN[7],IN[7],SEL);
    MUX8x1 mux7(OUT[1],IN[1],IN[2],IN[3],IN[4],IN[5],IN[6],IN[7],IN[7],SEL);
    MUX8x1 mux8(OUT[0],IN[0],IN[1],IN[2],IN[3],IN[4],IN[5],IN[6],IN[7],SEL);

    assign #2 OUTPUT = OUT;
endmodule
```

The data paths for sll,slr,ror and sra instructions are same as of the add instruction

Timing details for the datapath:

bne :

PC Update	Instruction Memory Read		Register Read	2's Comp	ALU
#1	#2		#2	#1	#2
	PC+4 Adder		Branch/Jump Target Adder		
	#1		#2		
			Decode		
			#1		

sll,slr,ror,sra :

PC Update	Instruction Memory Read		Register Read	ALU	
#1	#2		#2	#2	
	PC+4 Adder		Decode		
	#1		#1		
Register Write					
#1					

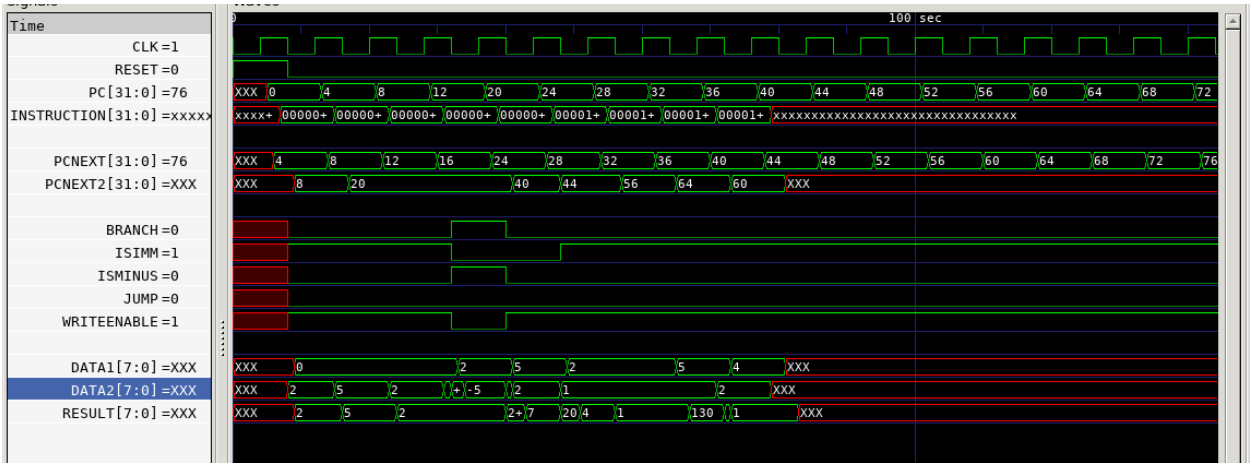
The set of instructions :

```
                    Register     Value
loadi 1 0x02          1      -->   2
loadi 3 0x05          3      -->   5
loadi 2 0x02          2      -->   2
bne 0x01 1 3
add 4 1 2
add 4 3 2             4      -->   7
sll 4 2 0x01          4      -->   4
srl 6 1 0x01          6      -->   1
ror 7 3 0x01          7      -->   130
sra 5 4 0x02          5      -->   1
```

Output of the register file :

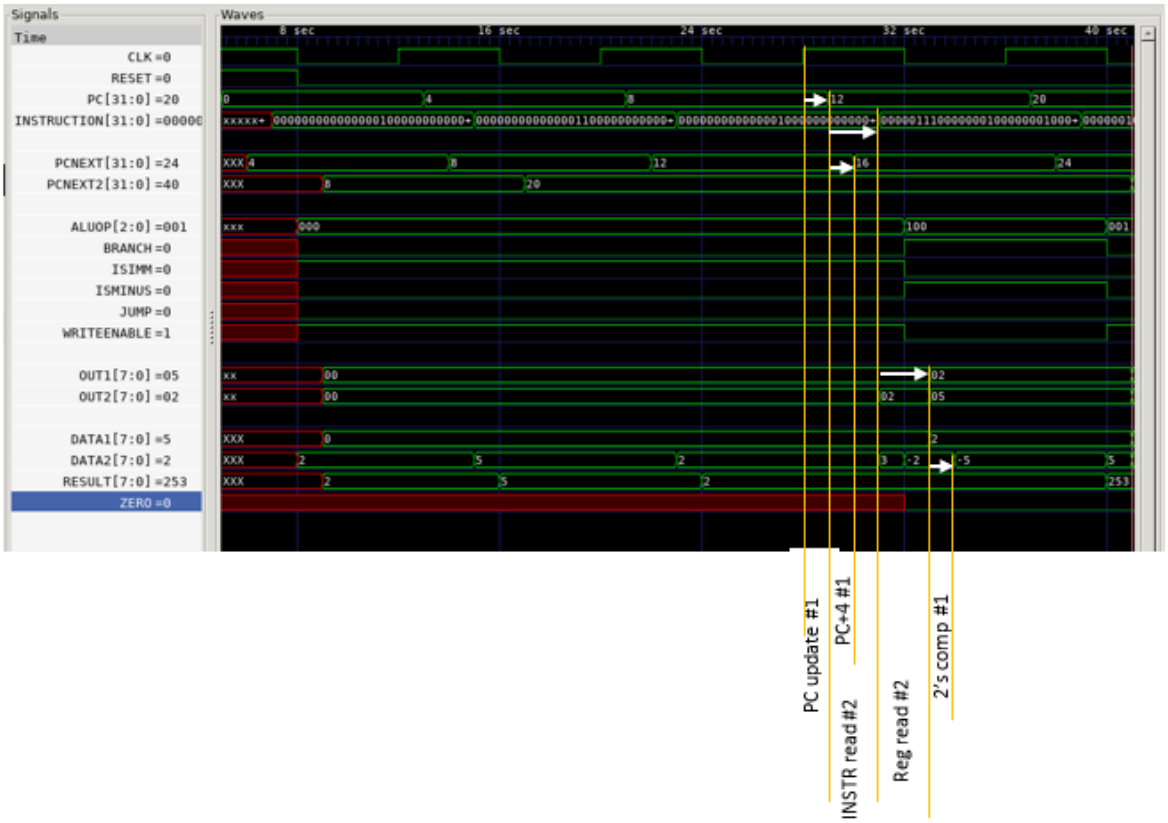
```
            ================================================
             Change of Register Content starting from Time #5
            ================================================

      time    reg0    reg1    reg2    reg3    reg4    reg5    reg6    reg7
      ------------------------------------------------------------------------
         5       0       0       0       0       0       0       0       0
        13       0       2       0       0       0       0       0       0
        21       0       2       0       5       0       0       0       0
        29       0       2       2       5       0       0       0       0
        45       0       2       2       5       7       0       0       0
        53       0       2       2       5       4       0       0       0
        61       0       2       2       5       4       0       1       0
        69       0       2       2       5       4       0       1     130
        77       0       2       2       5       4       1       1     130
APTOP-96FSIF5U:/mnt/c/Users/Chamudi/Desktop/Part5_Submit$
```

Timing diagram

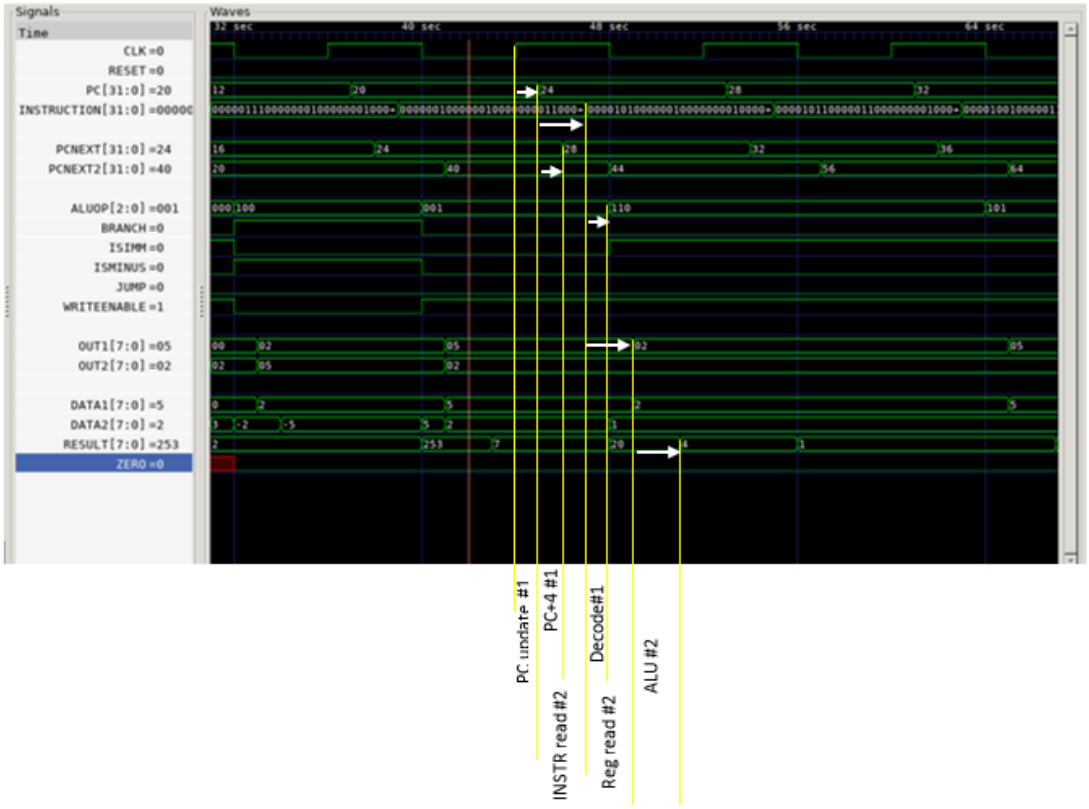


Timing details for each instruction

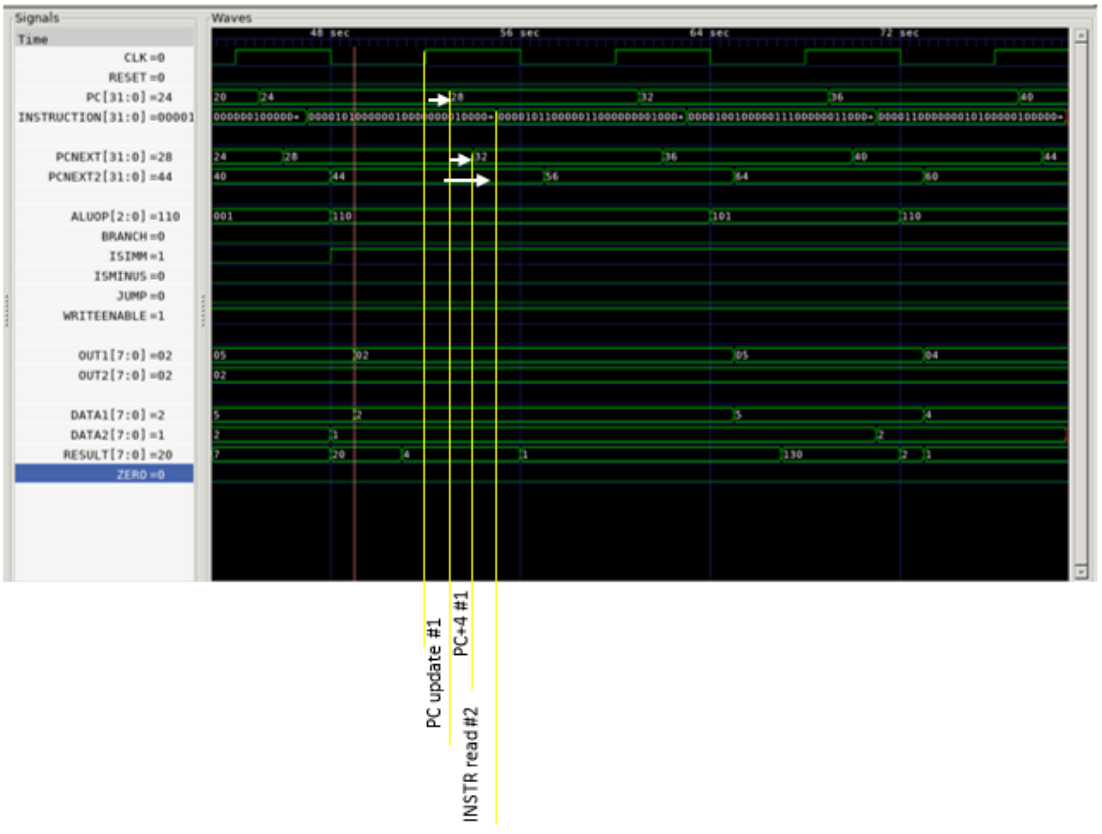
bne :



sll :

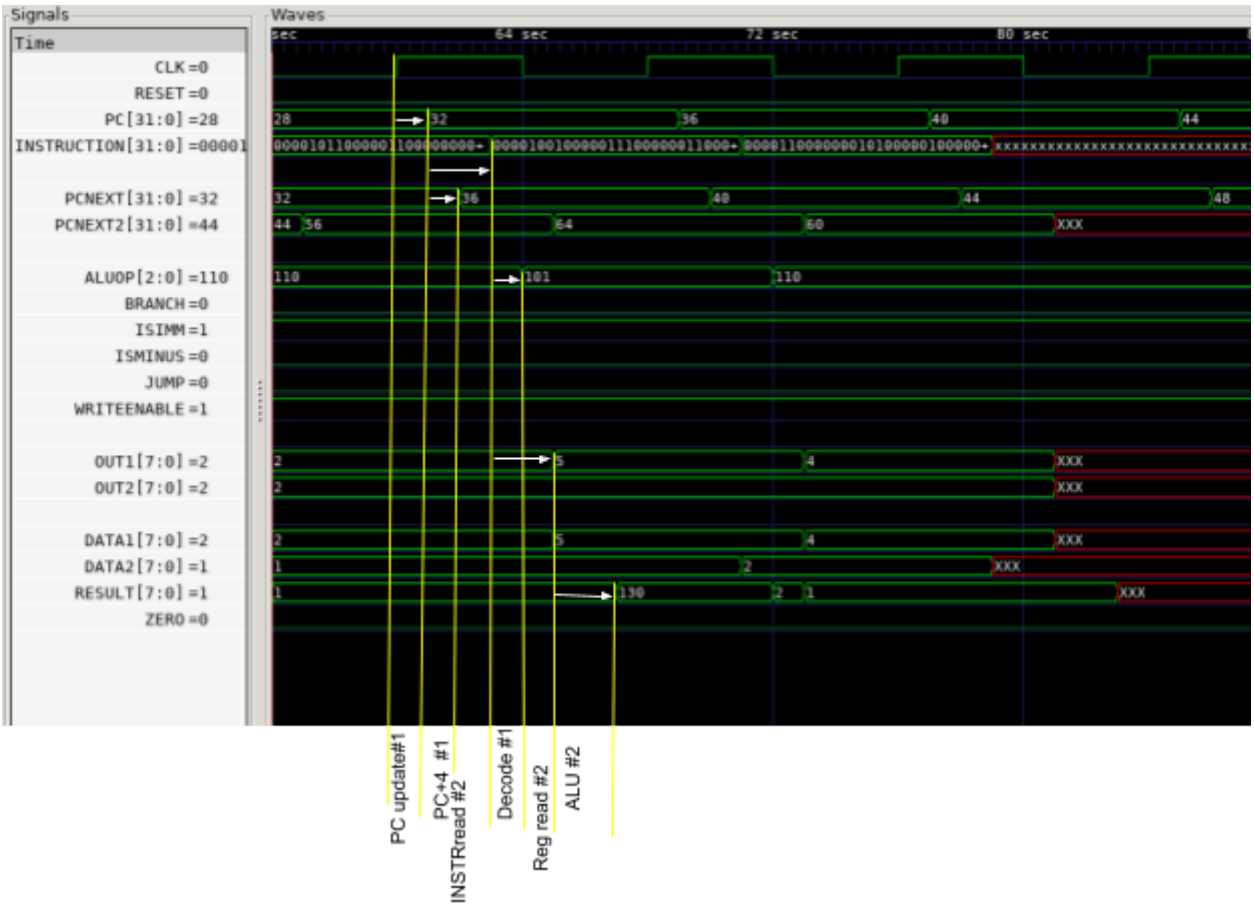

Signals
Waves
Time
CLK =0
RESET =0
PC[31:0] =20
INSTRUCTION[31:0] =00000
PCNEXT[31:0] =24
PCNEXT2[31:0] =40
ALUOP[2:0] =001
BRANCH =0
ISIMM =0
ISMINUS =0
JUMP =0
WRITEENABLE =1
OUT1[7:0] =05
OUT2[7:0] =02
DATA1[7:0] =5
DATA2[7:0] =2
RESULT[7:0] =253
ZERO =0
PC update #1
PC+4 #1
Decode#1
ALU #2
INSTR read #2
Reg read #2

srr:

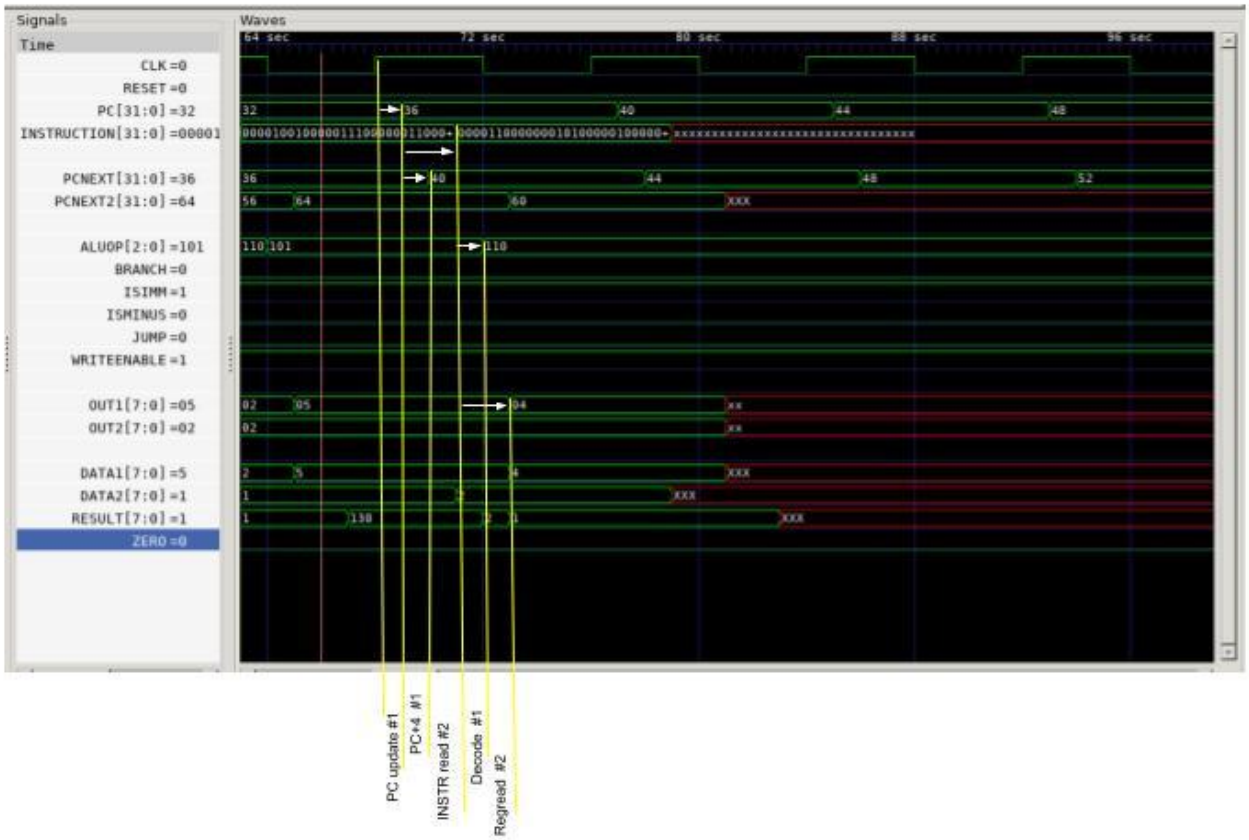

Signals
Waves
Time
CLK =0
RESET =0
PC[31:0] =24
INSTRUCTION[31:0] =00001
PCNEXT[31:0] =28
PCNEXT2[31:0] =44
ALUOP[2:0] =110
BRANCH =0
ISIMM =1
ISMINUS =0
JUMP =0
WRITEENABLE =1
OUT1[7:0] =02
OUT2[7:0] =02
DATA1[7:0] =2
DATA2[7:0] =1
RESULT[7:0] =20
ZERO =0
PC update #1
PC+4 #1
INSTR read #2

Here the delays for decoding,register read cannot be seen as the signals are same as the previous.Also as the data1,data2 are the same as the previous the delay of 2 time units in the ALU cannot be seen.

Ror :



Sra :



data1,data2 are the same as the previous the delay of 2 time units in the ALU cannot be seen