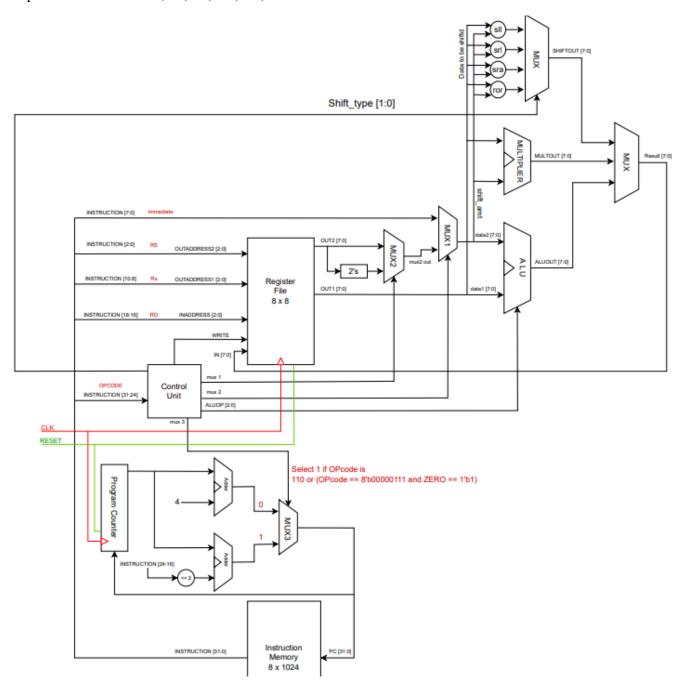
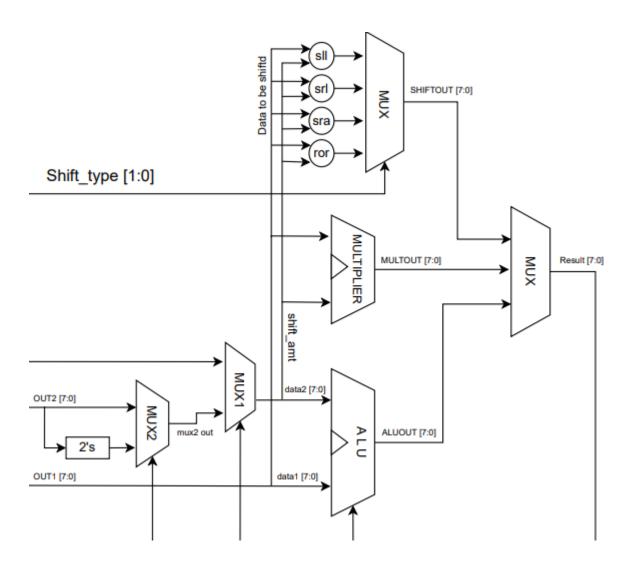
The Big Picture of the Simple 8-Bit Computer

Here is the complete block diagram of a simple 8-bit computer which performs the **add**, **sub**, **and**, **or**, **beq**, **j**, **mult**, **sll**, **srl**, **sra**, **ror**, **bne** instructions. So this can be used to perform different algorithms to find factorial of a number, nth fibonacci number, et cetra. So lets dive into the implementation of mult, sll, srl, sra, ror, bne instructions...



Implementation of mult, ssl, srl, sra, ror, bne instructions...

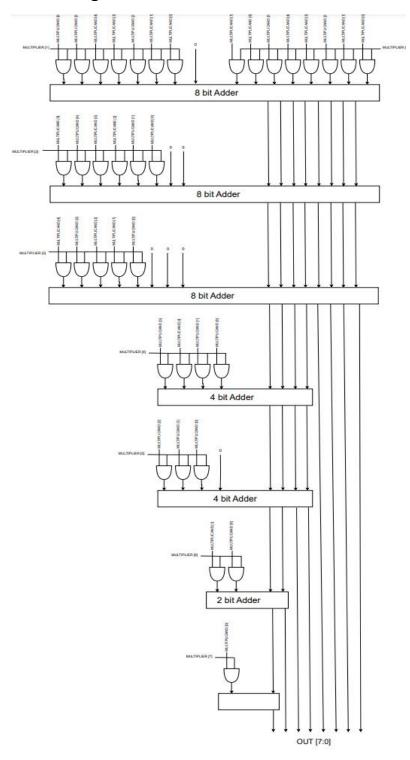
Additional two units have been introduced to perform shift and multiplication operations. Shift module has 4 sub units and a multiplexer to select shift_type. And there is another multiplexer to select the result from shift, ALU or Multiplier based on the instruction. Selection signals come from the control unit.



Multiplier

A delay of #2 is added for the multiply operation.

Circuit Diagram

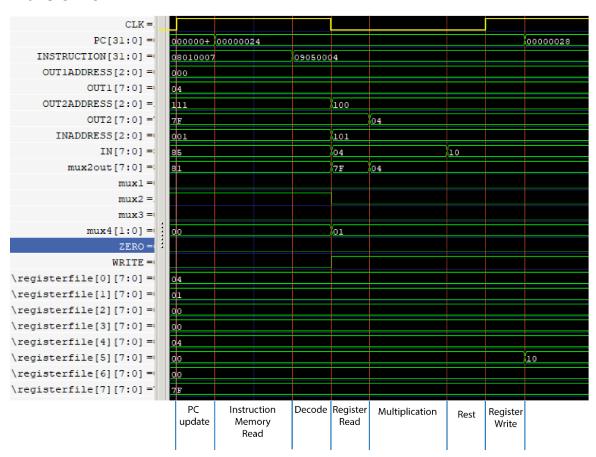


Verilog Code

```
module multiplier #(
    parameter W = 8
) (
    input [W-1:0] MULTIPLICAND,
    input [W-1:0] MULTIPLIER,
    output [W-1:0] OUT
);
    // normal multiplication as we do on decimal numbers
    assign #2 OUT = {MULTIPLICAND*MULTIPLIER[0]
                    {MULTIPLICAND*MULTIPLIER[1],1'b0} +
                    {MULTIPLICAND*MULTIPLIER[2],2'b00} +
                    {MULTIPLICAND*MULTIPLIER[3],3'b000} +
                    {MULTIPLICAND*MULTIPLIER[4],4'b0000} +
                    {MULTIPLICAND*MULTIPLIER[5],5'b00000} +
                    {MULTIPLICAND*MULTIPLIER[6],6'b000000} +
                    {MULTIPLICAND*MULTIPLIER[7],7'b0000000};
```

Endmodule

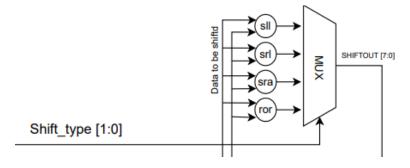
Waveforms



Shift

As shown below I have added new functional units for shifting and to select what type of shift is needed to perform base on the instruction. Shift type [1:0] signal comes from the control unit.

A delay of #3 is added for the shift operation.



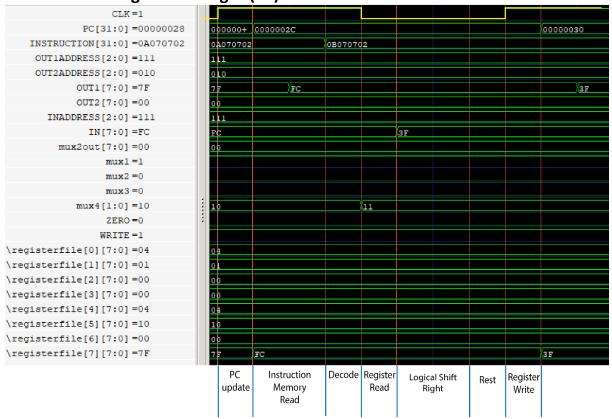
Verilog Code

```
module shift #(
    parameter W = 8
) (
                 [W-1:0] data,
    input
    input
                 [W-1:0] shift_amt,
    input
                 [1:0] shift_type,
    output reg [W-1:0] result
);
    integer i;
    always @(*) begin
        #3
        result = data;
        case(shift_type)
            2'b00: begin // LSL
                     for (i=0;i<shift_amt;i = i+1) result = {result[W-2:0], 1'b0};</pre>
             2'b01:
                     begin // LSR
                     for (i=0;i<shift_amt;i =i+1) result = {1'b0, result[W-1:1]};</pre>
             2'b10:
                     begin // SRA
                     for (i=0;i<shift_amt;i=i+1) result = {data[W-1], result[W-</pre>
1:1]};
                     end
            2'b11:
                     begin // ROR
                     for (i=0;i<shift_amt;i=i+1) result = {data[0], result[W-</pre>
1:1]};
                     end
        endcase
    end
endmodule
```

Waveform for logical shift left (sll)

			- 1	<u>, </u>						
CLK=1										
PC[31:0] =00000024		0000	00024	00000028						0000002C
INSTRUCTION[31:0] =09050004		0905	0004		0A07070	2				
OUT1ADDRESS[2:0] =000		000				111				
OUT2ADDRESS[2:0] =100		100				010				
OUT1[7:0] =04		04					7F			FC
OUT2[7:0] =04		04					00			
INADDRESS[2:0] =101		101				111				
IN[7:0] =10		10				04		FC		
mux2out[7:0] =04		04					00			
mux1=0										
mux2 =0										
mux3=0										
mux4[1:0] =01	1	01				10				
ZERO=0	1									
WRITE =1										
registerfile[0][7:0] =04		04								
registerfile[1][7:0] =01		01								
registerfile[2][7:0] =00		00								
registerfile[3][7:0] =00		00								
registerfile[4][7:0]=04		04								
registerfile[5][7:0]=00		00		10						
registerfile[6][7:0] =00		00								
registerfile[7][7:0] =7F		7F								FC
		-	PC update	Instruction Memory Read	Decode	Register Read	Logical Shift Left	Rest	Register Write	

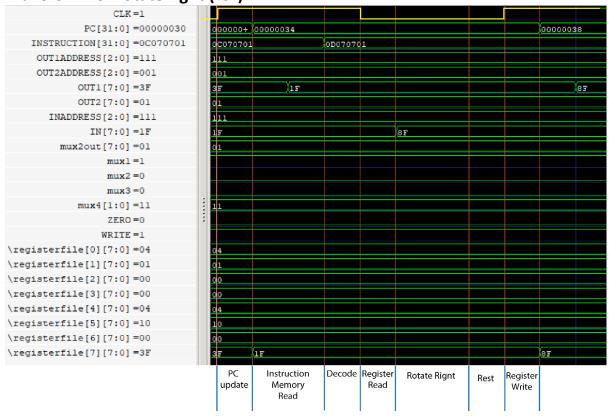
Waveform for logical shift right (srl)



Waveform for arithmetic shift right (sra)

CLK=1			(0.0.)						
PC[31:0] =0000002C			00000030						
INSTRUCTION[31:0] =08070702			100000030	V					00000034
• •		B070702		0C0707)1				
OUT1ADDRESS[2:0] =111		11							
OUT2ADDRESS[2:0] =010		010	V		001				V.
OUT1[7:0] =FC		rc .	ЗF						<u> </u>
OUT2[7:0] =00		00				01			
INADDRESS[2:0] =111		11							
IN[7:0] =3F		F				1F			
mux2out[7:0] =00	2	00				01			
mux1 =1									
mux2 =0									
mux3 =0									
mux4[1:0] =11		.1							
ZERO =0	1								
WRITE =1									
registerfile[0][7:0]=04)4							
registerfile[1][7:0]=01	0	1							
registerfile[2][7:0] =00	0	00							
registerfile[3][7:0]=00	0	00							
registerfile[4][7:0] =04	0)4							
registerfile[5][7:0]=10		.0							
registerfile[6][7:0]=00	0	00							
registerfile[7][7:0] =FC		rc	3F						1F
		PC update	Instruction Memory Read	Decode	Register Read	Arithmetic Shift Right	Rest	Register Write	

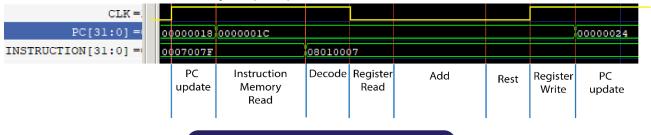
Waveform for rotate right (ror)



Bne instruction

For bne instruction we did a small modification for beq instruction. When we call beq control unit checks whether the given two registers equal or not and it indicates by a signal called ZERO. If ZERO = 1, given two registers are equal otherwise not equal. So if ZERO = 0 and instruction = 8'b00001000 then bne should execute.

Waveform for branch not equal (bne)

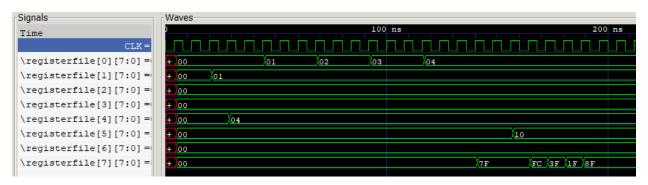


Initial Address : 28 (in decimal) After bne instruction : 36 (in decimal)

Here is the sample assembly code that we used for testing purpose,

```
// binary instructions
// Assembly program
loadi 0 0x00
                            0000000 00000000 00000000 00000000
                            0000001 00000000 00000001 00000000
loadi 1 0x01
loadi 4 0x04
                            00000100 00000000 00000100 00000000
                            00000100 00000000 00000010 00000111
beq 0x2 0 4
add 0 0 1
                            0000001 00000000 00000000 00000001
                            00000000 00000000 11111101 00000110
j 0xFD
loadi 7 0xFF
                            01111111 00000000 00000111 00000000
bne 0x01 0 7
                            00000111 00000000 00000001 00001000
                            1111111 0000000 00000110 00000000
loadi 6 0xFF
mult 5 0 4
                            00000100 00000000 00000101 00001001
sll 7 7 0x02
                            00000010 00000111 00000111 00001010
                            00000010 00000111 00000111 00001011
srl 7 7 0x02
                            0000001 00000111 00000111 00001100
sra 7 7 0x01
ror 7 7 0x01
                            0000001 00000111 00000111 00001101
```

Variation of Register Values for the above program,



Instruction Formats

add (bits 31-24)	Rd (bits 23-16)	Rt (bits 15-8)	Rs (bits 7-0)
OPCODE:0000_0000			
add (bits 31-24)	Rd (bits 23-16)	Rt (bits 15-8)	Rs (bits 7-0)
OPCODE:0000_0001			
sub (bits 31-24)	Rd (bits 23-16)	Rt (bits 15-8)	Rs (bits 7-0)
OPCODE:0000_0100			
and (bits 31-24)	Rd (bits 23-16)	Rt (bits 15-8)	Rs (bits 7-0)
OPCODE:0000_0010			
orr (bits 31-24)	Rd (bits 23-16)	Rt (bits 15-8)	Rs (bits 7-0)
OPCODE:0000_0011			
beq (bits 31-24)	Addr>>2 (bits	R1 (bits 15-8)	R2 (bits 7-0)
	23-16)		
OPCODE:0000_0111			
j (bits 31-24)	Addr>>2 (bits	- (bits 15-8)	- (bits 7-0)
	23-16)		
	/		
OPCODE:0000_0110			
mult (bits 31-	Rd (bits 23-16)	Rt (bits 15-8)	Rs (bits 7-0)
_	,	Rt (bits 15-8)	Rs (bits 7-0)
mult (bits 31-	,	Rt (bits 15-8)	Rs (bits 7-0)
mult (bits 31- 24)	,	Rt (bits 15-8) R2 (bits 15-8)	Shift_amt (bits
mult (bits 31- 24) OPCODE:0000_1001 sll (bits 31-24)	Rd (bits 23-16)	,	
mult (bits 31- 24) OPCODE:0000_1001	Rd (bits 23-16)	,	Shift_amt (bits
mult (bits 31- 24) OPCODE:0000_1001 sll (bits 31-24)	Rd (bits 23-16)	,	Shift_amt (bits 7-0) Shift_amt (bits
mult (bits 31- 24) OPCODE:0000_1001 sll (bits 31-24) OPCODE:0000_1010	Rd (bits 23-16) R1 (bits 23-16)	R2 (bits 15-8)	Shift_amt (bits 7-0)

sra	(bits	31-24)	R1 (bits 23-16)	R2 (bits 15-8)	Shift_amt (bits
					7-0)
OPCO	DE:0000	0_1100			_
ror	(bits	31-24)	R1 (bits 23-16)	R2 (bits 15-8)	Shift_amt (bits
	•	·	,	,	7-0)
OPCO	DE:0006	0_1101			
bne	(bits	31-24)	Addr>>2 (bits	R1 (bits 15-8)	R2 (bits 7-0)
			23-16)		

OPCODE:0000_1000

END OF THE DOCUMENT