ا

Faculty of Engineering and Technology

Electrical and Computer Engineering Department

COMPUTER STRUCTURE AND ORGANIZATION

Assignment 1

Prepared by: Karim Taha number:1211155

Instructor: Dr Jamal Seyam

Assistant:Eng Borhan Adain

Section: 1

Date:5/14/2024

Table of Contents

Table of figures

Question one

Micro_control

```
1   module micro_control(output  x8,x7,x6,x5,x4,x3,x2,x1 , input [3:0]q,input [7:0] t) ;
2
3   wire [8:0]a;
4   and g1(a[0],q[2],t[3]);
5   and g2(a[1],q[3],t[3]);
6   or o1(x1,t[0],a[0],a[1]);//x1=t0+q2.t3+q3t3
7   and g3(x2,q[3],t[5]);//x2=q3.t5
8
9   and g4(a[3],q[2],t[4]);
10  and g5(a[4],q[3],t[4]);
11  or xo2(x3,t[1],a[3],a[4]);//x3=t1+q2.t4+q3t4
12
13  and xa4(a[5],q[3],t[6]);
14  or xo4(x4,x3,a[5]); // x4 =x3+q3.t6
15
16  and xa5(a[6],q[2],t[5]),
17      xa51(a[7],q[3],t[7]);
18  or xo5(x5,a[6],a[7]); //x5 =q2.t5+q3.t7
19
20  and xa6(x6,q[1],t[3]);//x6=q1.t3
21
22  or xo7(x7,x5,x6);//x7=x5+x6
23
24  buf (x8,t[2]);//x8=t2
25  endmodule
```

Figure 1:micro control code

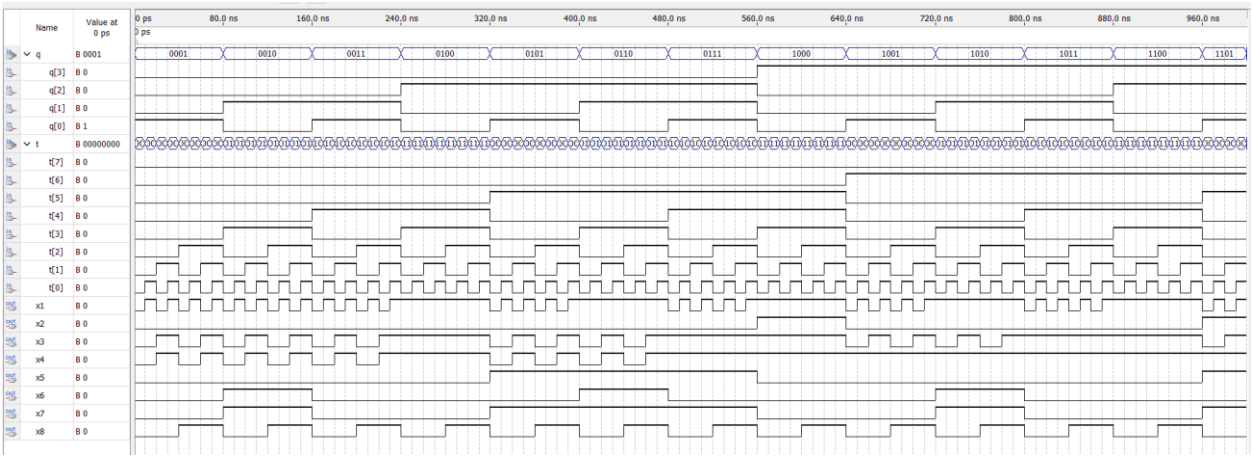


Figure 2:microcountrol waveform

T (time sequence)

```
1  module T_reg(output [2:0] Q ,input count,clk,reset);
2    //T_reg id 3-bit counter that count when count = 1 and remian the same if count = 0 , Reset when reset = 1 ;
3   wire q01 ;
4   and A0(q01,Q[0],Q[1]);
5   T_FF t0(Q[0],count,clk,reset),
6        t1(Q[1],Q[0],clk,reset),
7        t2(Q[2],q01,clk,reset);
8
9  endmodule
```

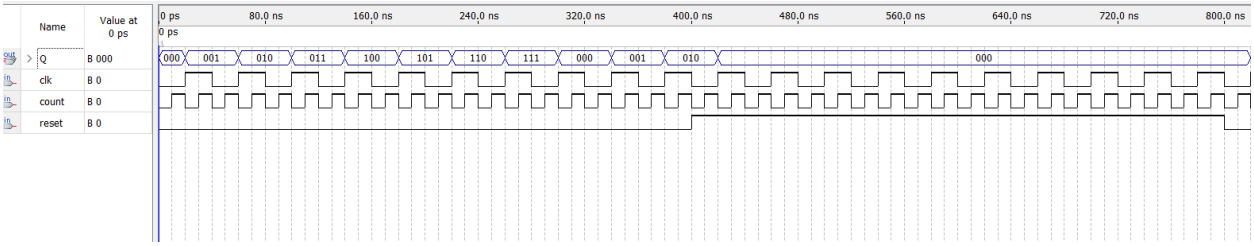


Figure 3:T wave form

The counter will count from (000-111) and reset when reset =1 the and the counter work at the positive edge of the clock T(output of counter) will enter the 3x8 decoder 3-bit input When T=000 the decoder will make t0 active

Register

```
1  module one_bit_reg(output Q , input data,select,clk,reset );
2
3  wire muxout;
4  mux2x1(muxout,Q,data,select);
5  D_FF(Q,muxout,clk,reset);
6
7  endmodule
```

Figure 4:one-bit register

To create the register, a mux 2x1 and a D flip-flop were used. The output Q from the D flip-flop and the load data were fed into the mux 2x1. The output of the mux was then connected to the input of the D flip-flop. In this manner, a one-bit register was constructed. To create a 10-bit register or an 8-bit register, 10 or 8 such circuits from this model would be needed.

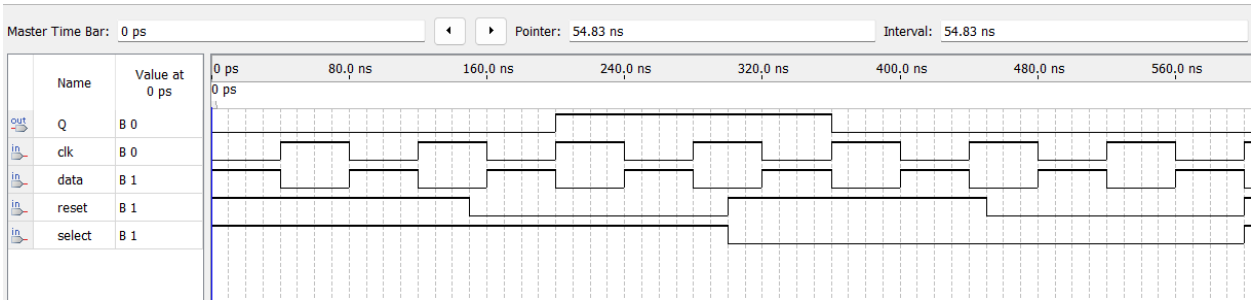


Figure 5:one bit regiesteg wave form

Reg_8bit

```
1  module reg_8_bit (output [7:0] Q ,input [7:0] data ,input select,clk,reset) ;
2
3
4  one_bit_reg r0(Q[0],data[0],select,clk,reset),
5              r1(Q[1],data[1],select,clk,reset),
6              r2(Q[2],data[2],select,clk,reset),
7              r3(Q[3],data[3],select,clk,reset),
8              r4(Q[4],data[4],select,clk,reset),
9              r5(Q[5],data[5],select,clk,reset),
10             r6(Q[6],data[6],select,clk,reset),
11             r7(Q[7],data[7],select,clk,reset);
12
13
14
15 endmodule
```

Figure 6: 8 bit register

reg_10bit

```
1  module reg_10_bit (output [9:0] Q ,input [9:0] data ,input select,clk,reset) ;
2
3
4
5
6
7
8  one_bit_reg r0(Q[0],data[0],select,clk,reset),
9              r1(Q[1],data[1],select,clk,reset),
10             r2(Q[2],data[2],select,clk,reset),
11             r3(Q[3],data[3],select,clk,reset),
12             r4(Q[4],data[4],select,clk,reset),
13             r5(Q[5],data[5],select,clk,reset),
14             r6(Q[6],data[6],select,clk,reset),
15             r7(Q[7],data[7],select,clk,reset),
16             r8(Q[8],data[8],select,clk,reset),
17             r9(Q[9],data[9],select,clk,reset);
18
19
20
21 endmodule
```

Figure 7: 10-bit register

PC

```
1  module PC (output reg [7:0] address,input count, reset);
2
3
4  always @(posedge count or posedge reset)
5     begin
6        if (reset)
7           address <= 4'h00; // Reset to 0x00
8        else if (count)
9           if(address < 8 )
10             address <= address + 1'b1; // Increment on count
11          else
12             address <= 0 ;
13    end
14
15
16 endmodule
```

Figure 8:PC counter code

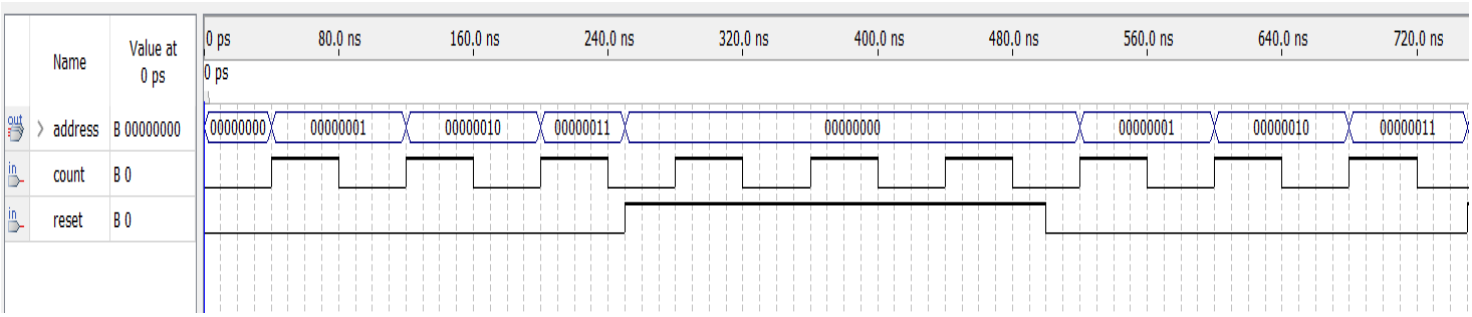


Figure 9:PC counter wave form

Pc counter will contain the address for next instruction to be fetched and decoded

IR

```
1  module reg_10_bit (output [1:0]opcode ,output [7:0] Q ,input [9:0] data ,input select,clk,reset) ;
2
3  one_bit_reg r0(Q[0],data[0],select,clk,reset),
4              r1(Q[1],data[1],select,clk,reset),
5              r2(Q[2],data[2],select,clk,reset),
6              r3(Q[3],data[3],select,clk,reset),
7              r4(Q[4],data[4],select,clk,reset),
8              r5(Q[5],data[5],select,clk,reset),
9              r6(Q[6],data[6],select,clk,reset),
10             r7(Q[7],data[7],select,clk,reset),
11             r8(opcode[0],data[8],select,clk,reset),
12             r9(opcode[1],data[9],select,clk,reset);
13
14
15 endmodule
```

Figure 10:IR code

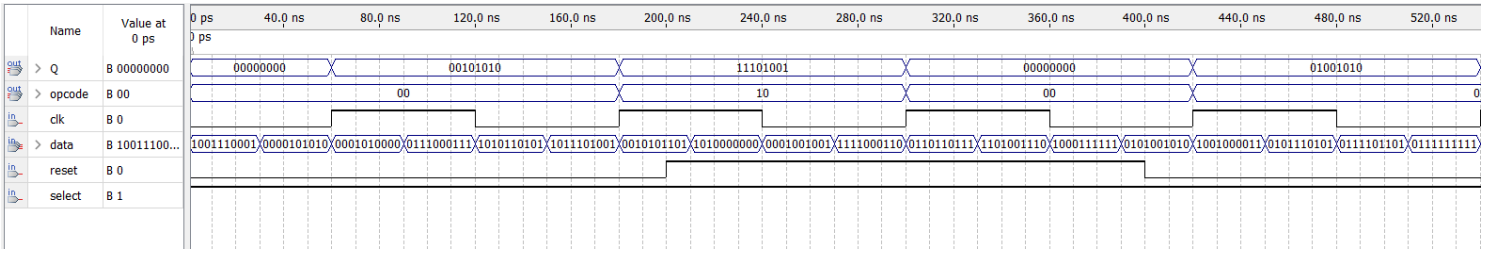


Figure 11:IR wave form

The IR will divide the instruction to opcode that will go to the combinational circuit and the rest of the instruction will go to the MAR

Memory

```
module RW_Memory16X14_final (output  reg [9:0]data_out,input  [9:0] data_in , input [2:0] address , input read,write,clk);
reg  [9:0] Mem [15:0];

generate
    integer i;
    initial begin
        for (i = 0; i < 5; i = i + 1) begin
            case (i)
                0: Mem[i]  = 10'b1100001011;
                1: Mem[i]  = 10'b1000001011;
                2: Mem[i]  = 10'b0110011111;
                2: Mem[i]  = 10'b1110011111;
                3: Mem[i]  = 10'b0100010001;
                4: Mem[i]  = 10'b1000001101; // LDI 10 , Mem[13] - > MBR -> A  = 0x6E
                5: Mem[i]  = 10'b1100001110; // LDA 11 , Mem[Mem[14]] -> MBR -> A = 0x88
                6: Mem[i]  = 10'b0100001011;
                7: Mem[i]  = 10'b0100001010;
                8: Mem[i]  = 10'b0100001010;
                9: Mem[i]  = 10'b0100001010;
                10:Mem[i]  = 10'b0100001011;
                11:Mem[i]  = 10'b0000001100;
                12:Mem[i]  = 10'b0001110111;
                13:Mem[i]  = 10'b0001101110;
                14:Mem[i]  = 10'b0000001111;
                15:Mem[i]  = 10'b0010001000;
```

Figure 12:memory code part1

```
                12:Mem[i]  = 10'b0001110111;
                13:Mem[i]  = 10'b0001101110;
                14:Mem[i]  = 10'b0000001111;
                15:Mem[i]  = 10'b0010001000;

                // Add more initializations for other memory locations if needed
                default: Mem[i] = 10'b0000000000; // Default initialization to 0
            endcase
        end
    end
endgenerate
always @(posedge clk)begin
if (write)
Mem[address]<= data_in;
else if (read)
data_out <=Mem[address];
end
endmodule
```

Figure 13:Memory code part 2

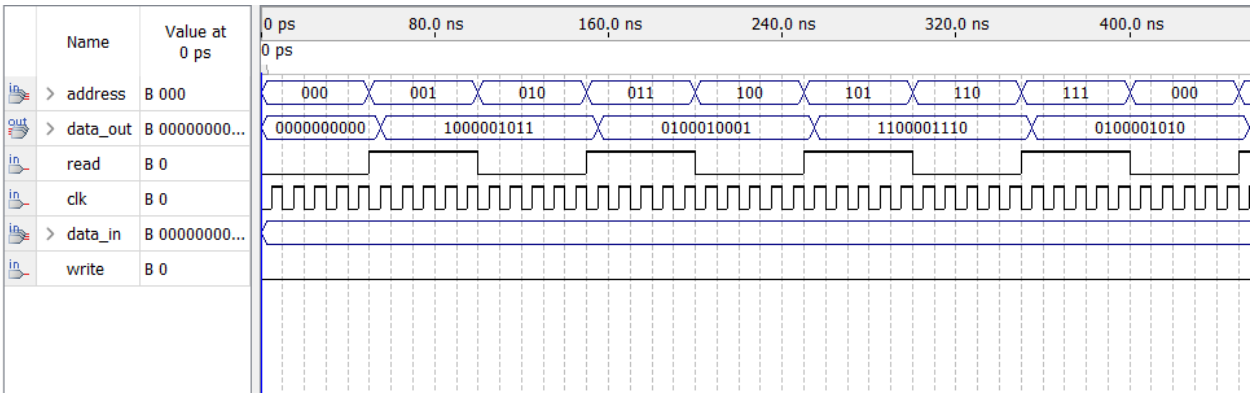


Figure 14:Read wave

At address 011 that means when Memory [3] the data will be 0100010001 so the memory reads the data from the memory with address 3 and the address will be getin from the mar

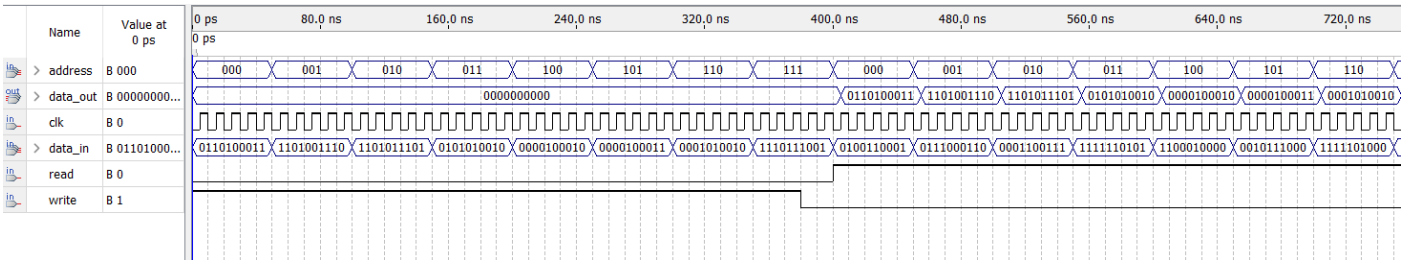


Figure 15:write memory

At address 001, the initial data in memory was 100001011. The new data to be written (data in) is 1101001110. After performing the write operation at address 001, the data at this address will be 1101001110. A read operation at the same address will now show this new data. The difference

between the initial and the new data can be deduced by comparing figures 14 and 15 for the same address."

Decoder3x8

```
1  module decoder3x8(input[2:0] in,output reg[7:0] out);
2
3  initial out = 8'b00000000;
4  always @(in )
5  begin
6    case (in)
7    3'b000: out=8'b00000001;
8    3'b001: out=8'b00000010;
9    3'b010: out=8'b00000100;
10   3'b011: out=8'b00001000;
11   3'b100: out=8'b00010000;
12   3'b101: out=8'b00100000;
13   3'b110: out=8'b01000000;
14   3'b111: out=8'b10000000;
15 endcase
16
17 end
18
19 endmodule
```

Figure 17:decoder3x8

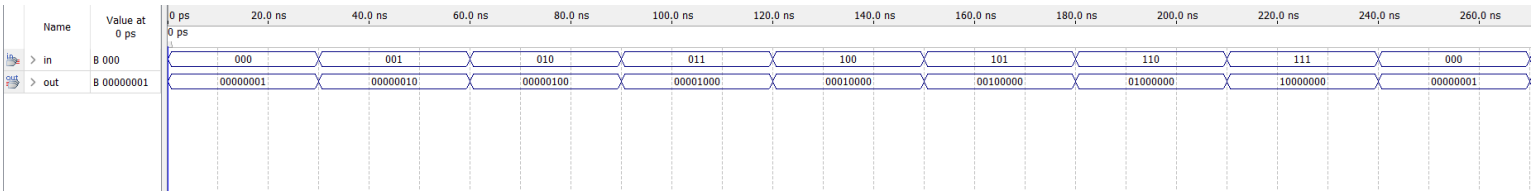


Figure 16:decoder3x8 waveform

decoder2x4

```
1  module decoder2x4(input[1:0] in,output reg[3:0] out);
2
3  initial out = 4'b0000;
4  always @(in )
5  begin
6
7    case (in)
8    2'b00: out=4'b0001;
9    2'b01: out=4'b0010;
10   2'b10: out=4'b0100;
11   2'b11: out=4'b1000;
12 endcase
13 end
14
15 endmodule
```

Figure 18:deocder2x4 code

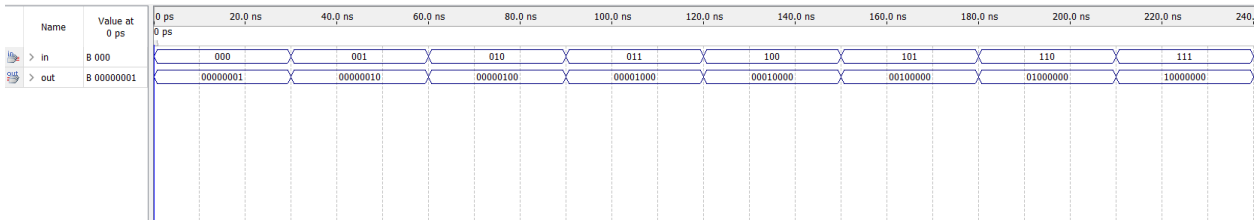


Figure 19:decoder 2x4 waveform

Block Diagrams

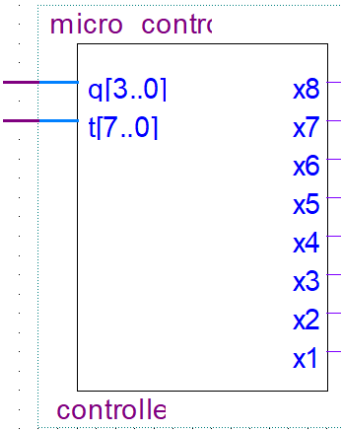


Figure 20:combntional circuit block

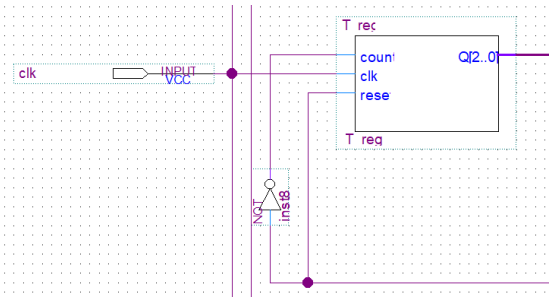


Figure 22: T block

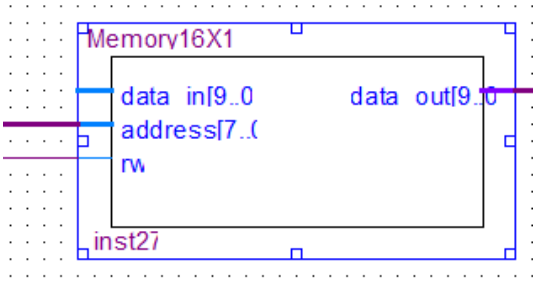


Figure 21:memory block

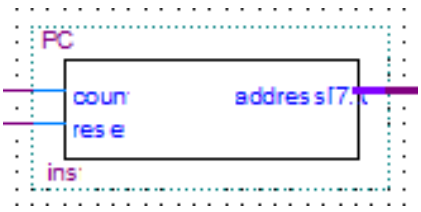


Figure 23:PC block

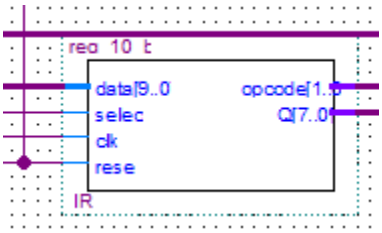


Figure 25:IR block

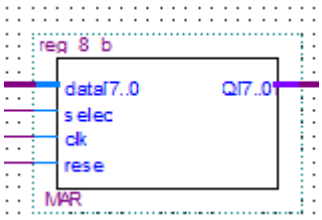


Figure 24:MAR block

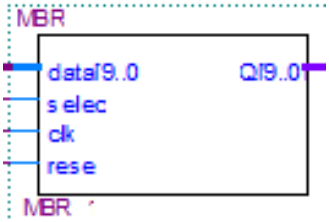


Figure 27:MBR block

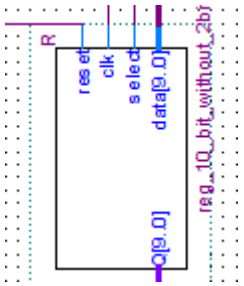


Figure 26: R block

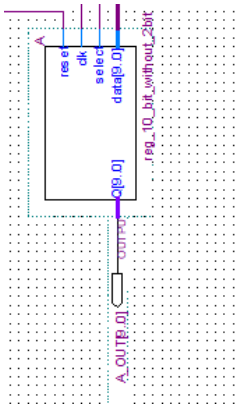


Figure 28: A block

T with combntional circuit

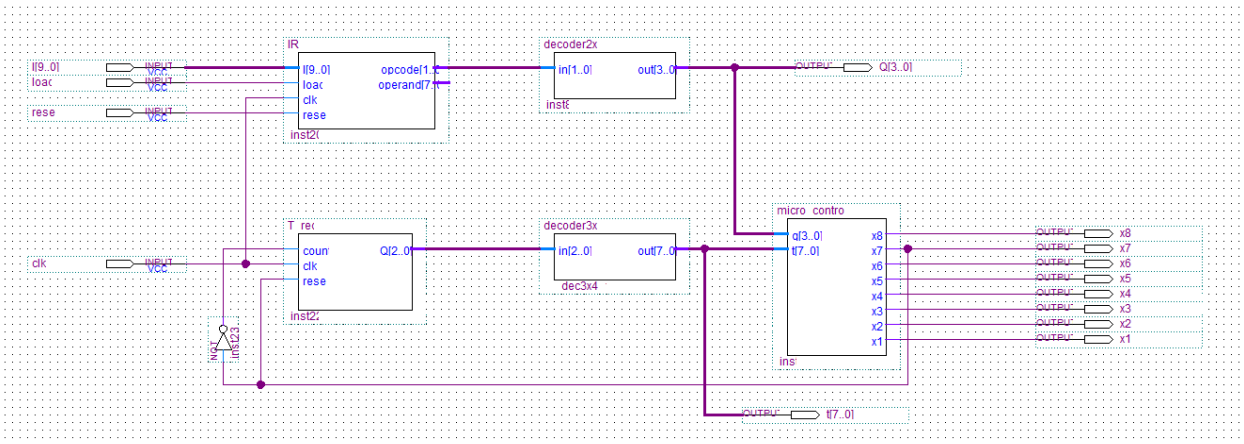


Figure 29:control unit with IR

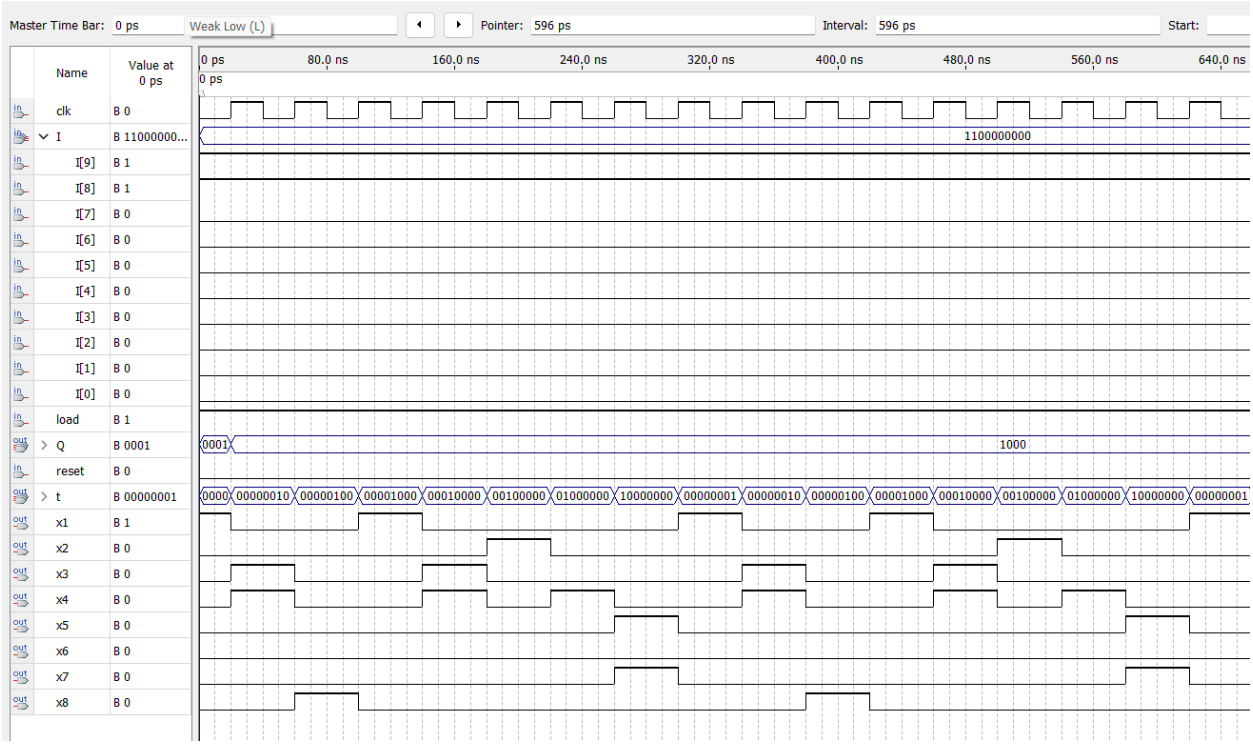


Figure 30: control unit with IR wave form

[I] is the instruction it is a 10-bit instruction when the instruction enters the IR the IR will divide it to the opcode and address the opcode will go 2x4 decoder to determine the operation of the combinational circuit while the address will go to the MAR

If opcode equals 11 (LDA) the T will work for 7 state sequence(t0-t7)

, if opcode equals 10 (LDI) the T will work five state sequence (t0-t5)

Test for the system

X3

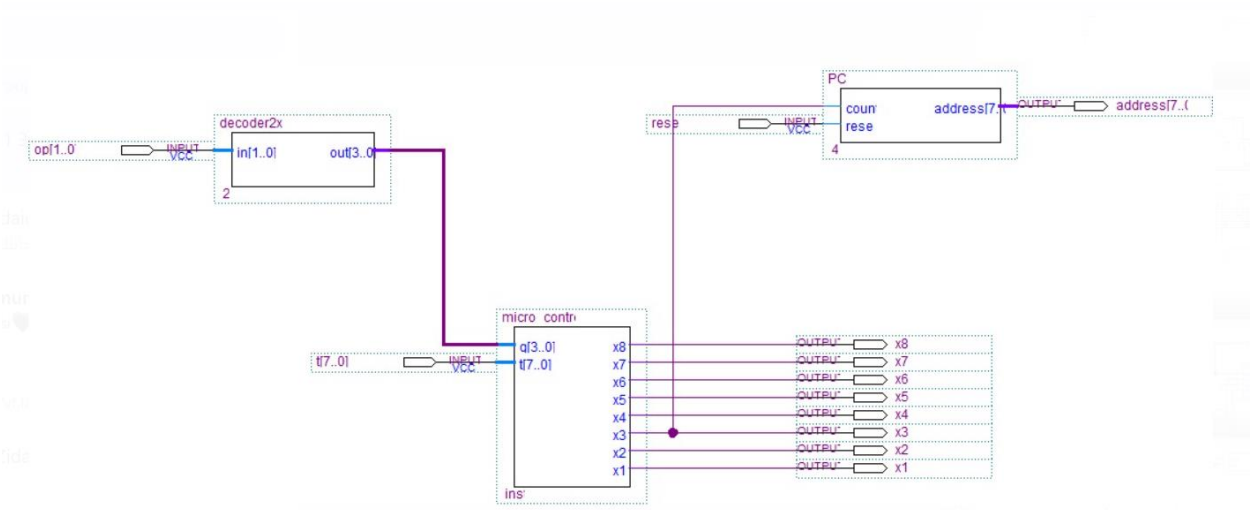


Figure 32:X3 test

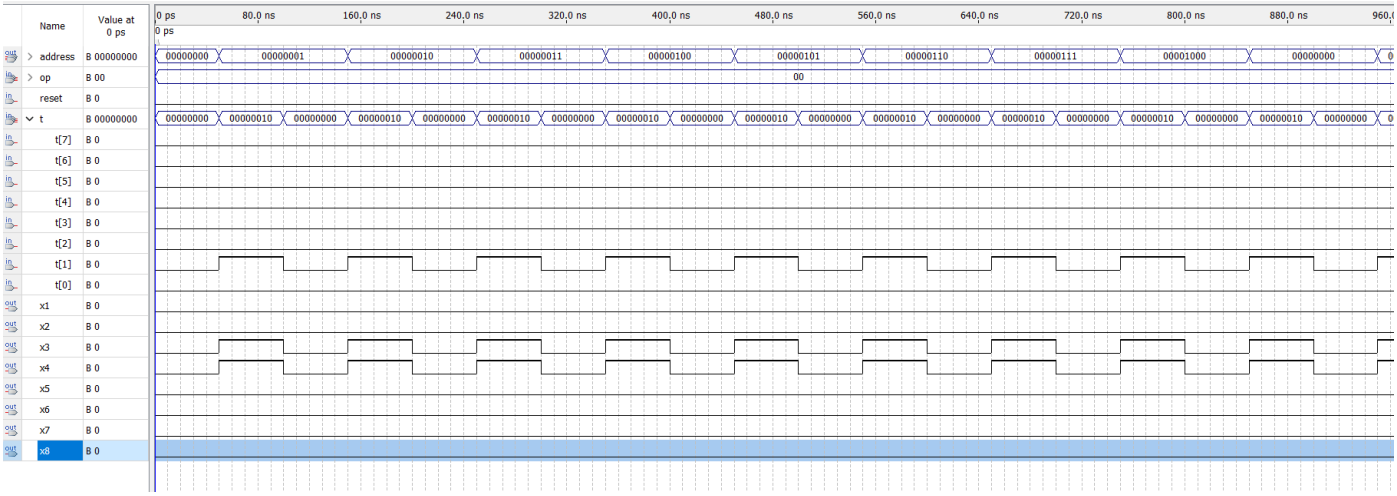


Figure 31:X3 waveform

X4

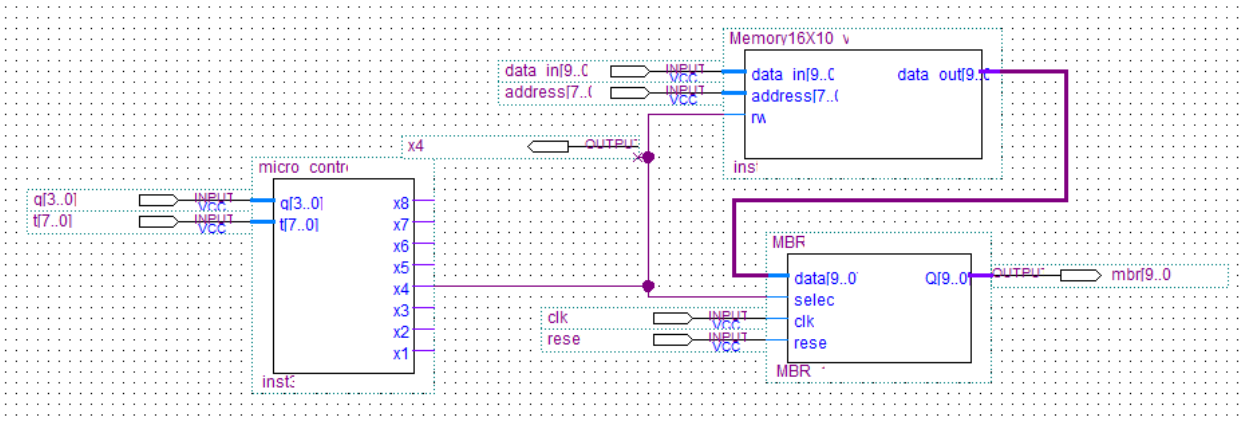


Figure 33:X4 test

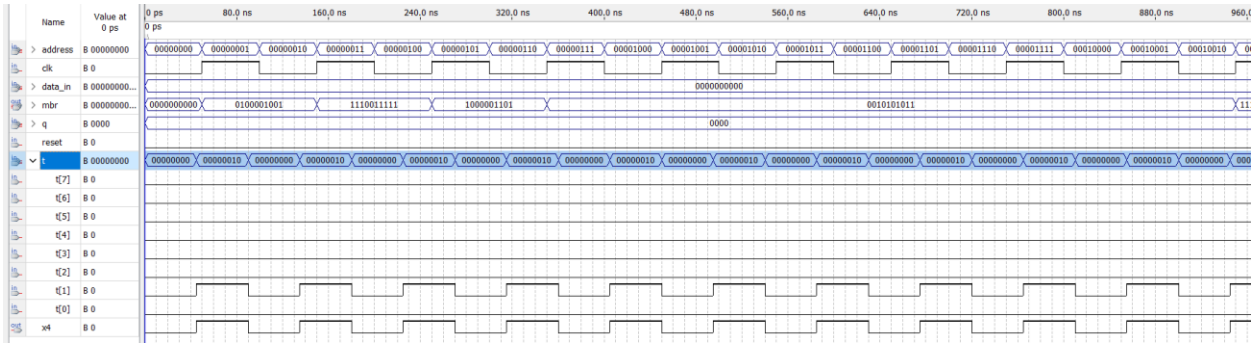


Figure 34:x4 wave

When x4 is one the data in the memory will be transfer from memory to the mbr for example at address 4 the data will be 1000001101 and that what the mbr takes and x4 will be on when the address is 4

X8

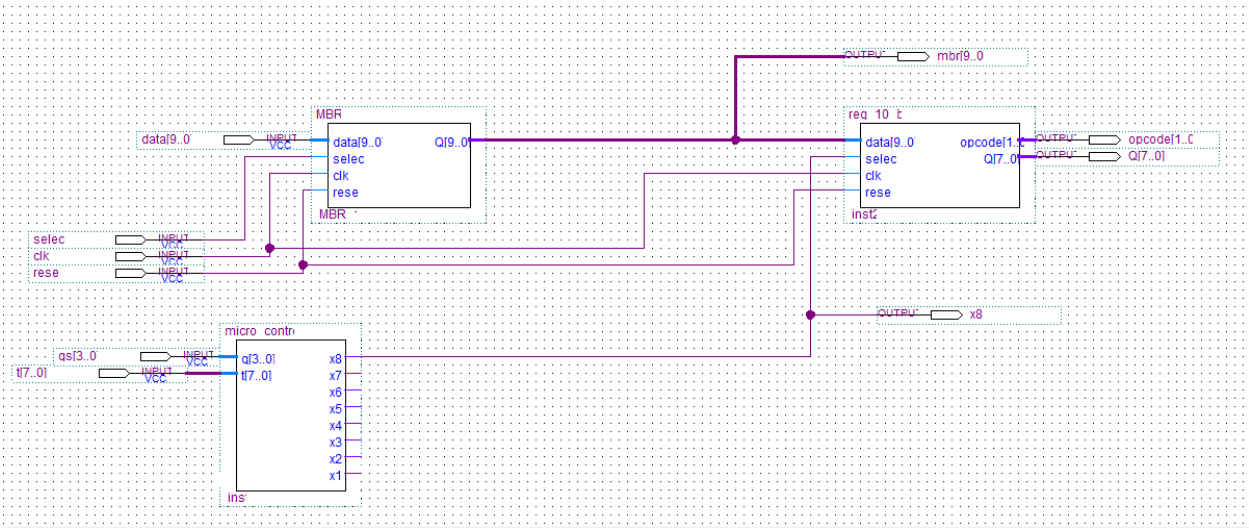


Figure 35:X8 tets

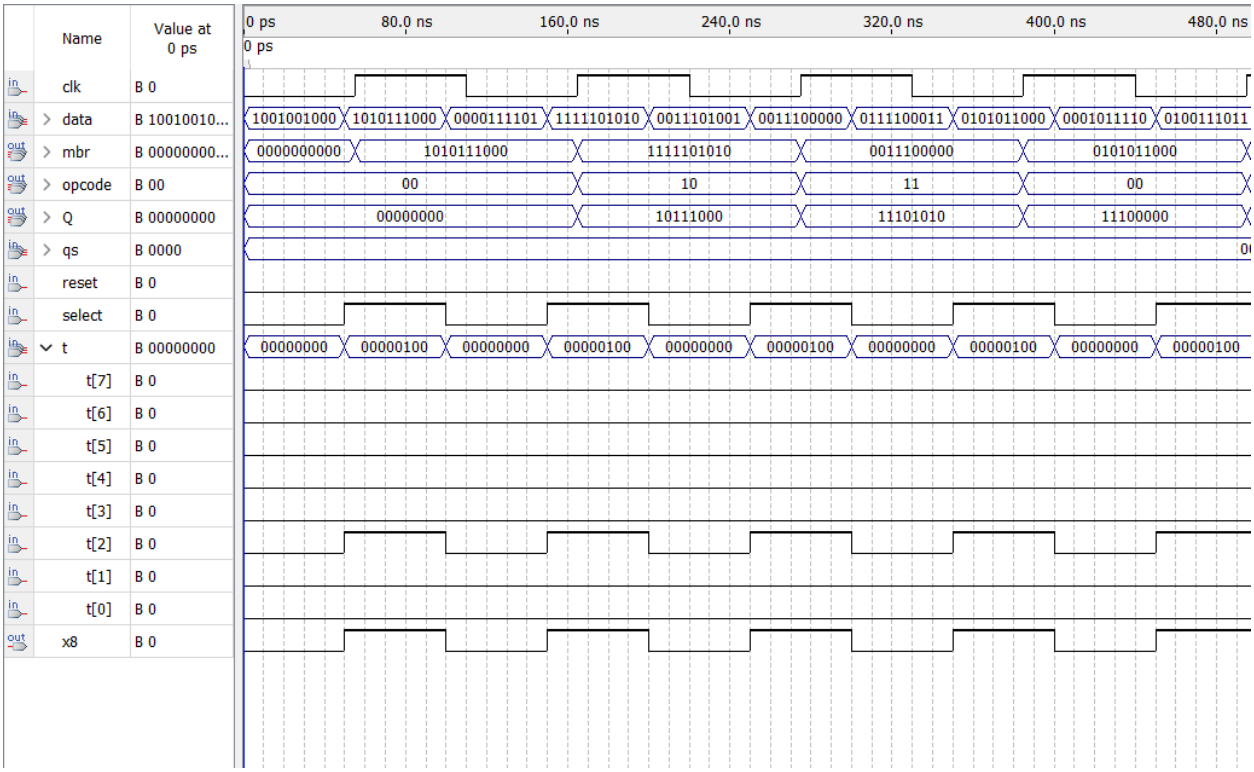


Figure 36:x8 wav

microcomputer

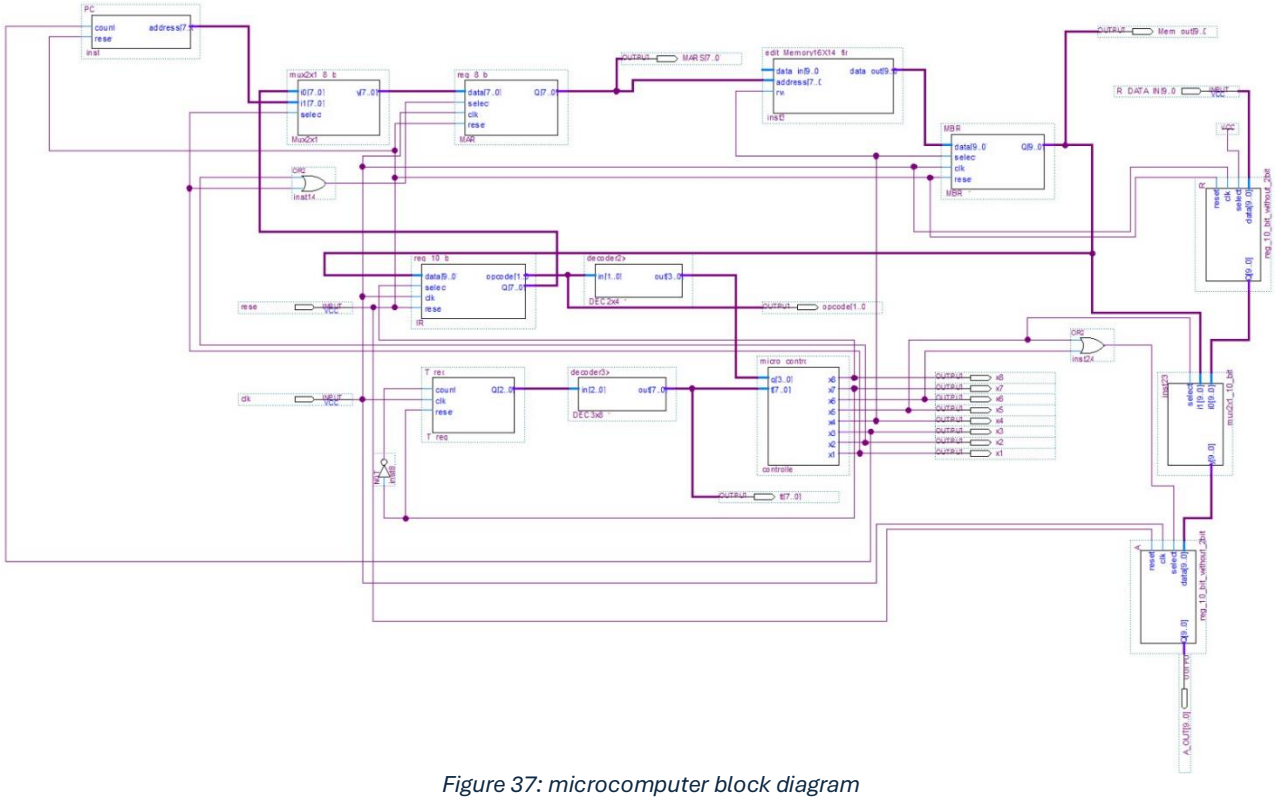


Figure 37: microcomputer block diagram

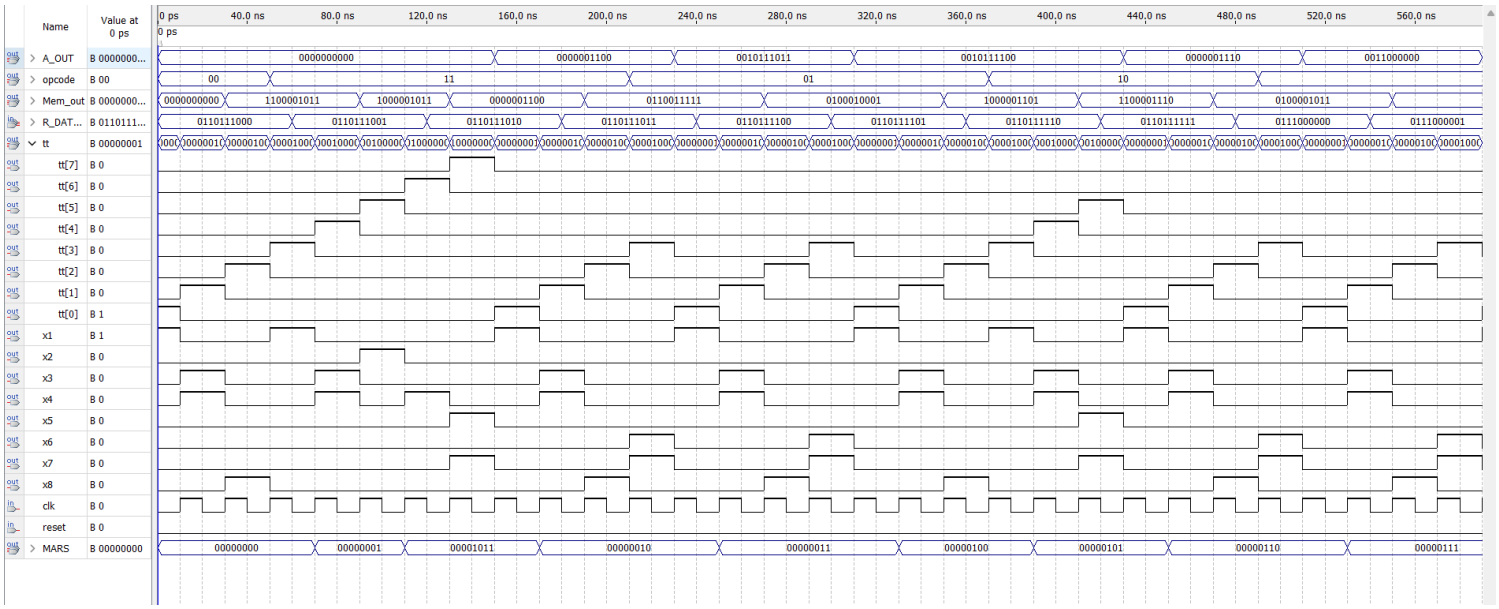


Figure 38:microcomputer block diagram waveform

When the opcode be (00) there is no sequnce to do when it is (01) transfer the data from R to A

When it is (10) transfer the data in the MBR to A when it is (11) transfer the data from memory to A

01: after T finishes 3 clocks the instruction will be in R(0110111100) but A will take the firs 8 bit from R because in 10opreation it required the operand so A should be 8 bit register but I made it 10 bit register and it will take but the last two bit 00 so A (0010111100)

10: after T finishes 5 clocks the instrcion will be 11000011100 the operand of this instruction 0111100 so A 000111100

11 after T finish 7clock the instction will be 0000001100 will be in A

Question two

arithmetic circuit

```
1  module arthimtic_circuit (output [7:0]D,output cout , input [7:0]A,B ,input cin, input [1:0]S);
2  wire c1,c2,c3 ,c4,c5,c6,c7;
3  wire y0,y1,y2,y3 ,y4,y5,y6,y7;
4
5  mux4x1 mx0(y0,B[0],~B[0],1'b0,1'b1,S[0],S[1]),
6         mx1(y1,B[1],~B[1],1'b0,1'b1,S[0],S[1]),
7         mx2(y2,B[2],~B[2],1'b0,1'b1,S[0],S[1]),
8         mx3(y3,B[3],~B[3],1'b0,1'b1,S[0],S[1]),
9         mx4(y4,B[4],~B[4],1'b0,1'b1,S[0],S[1]),
10        mx5(y5,B[5],~B[5],1'b0,1'b1,S[0],S[1]),
11        mx6(y6,B[6],~B[6],1'b0,1'b1,S[0],S[1]),
12        mx7(y7,B[7],~B[7],1'b0,1'b1,S[0],S[1]);
13
14 full_adder FA0(D[0],c1,A[0],y0,cin),
15            FA1(D[1],c2,A[1],y1,c1),
16            FA2(D[2],c3,A[2],y2,c2),
17            FA3(D[3],c4,A[3],y3,c3),
18            FA4(D[4],c5,A[4],y4,c4),
19            FA5(D[5],c6,A[5],y5,c5),
20            FA6(D[6],c7,A[6],y6,c6),
21            FA7(D[7],cout,A[7],y7,c7);
22
23 endmodule
```

Figure 39:arithmetiv circuit cod

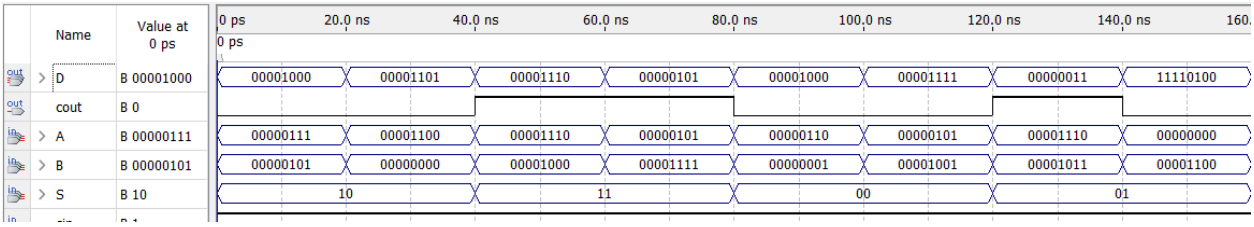


Figure 40:arithmetic circuit waveform

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Figure 41;arithmetic circuit Truth table

Logic circuit

```
1   module logic_circuit (output [7:0]E , input[7:0]A,B ,input [1:0]S);
2
3   wire [7:0] and_op,or_op,xor_op,not_op ;
4   // 00 -> A and B , 01 -> A or B , 10 -> A xor B , 11 -> not A
5   assign and_op = A & B ;
6   assign or_op = A | B ;
7   assign xor_op = A ^ B ;
8   assign not_op = ~A ;
9
10  mux4x1 mx0(E[0],and_op[0],or_op[0],xor_op[0],not_op[0],S[0],S[1]),
11         mx1(E[1],and_op[1],or_op[1],xor_op[1],not_op[1],S[0],S[1]),
12         mx2(E[2],and_op[2],or_op[2],xor_op[2],not_op[2],S[0],S[1]),
13         mx3(E[3],and_op[3],or_op[3],xor_op[3],not_op[3],S[0],S[1]),
14         mx4(E[4],and_op[4],or_op[4],xor_op[4],not_op[4],S[0],S[1]),
15         mx5(E[5],and_op[5],or_op[5],xor_op[5],not_op[5],S[0],S[1]),
16         mx6(E[6],and_op[6],or_op[6],xor_op[6],not_op[6],S[0],S[1]),
17         mx7(E[7],and_op[7],or_op[7],xor_op[7],not_op[7],S[0],S[1]);
18
19
20  endmodule
```

Figure 42:Logic circuit cod

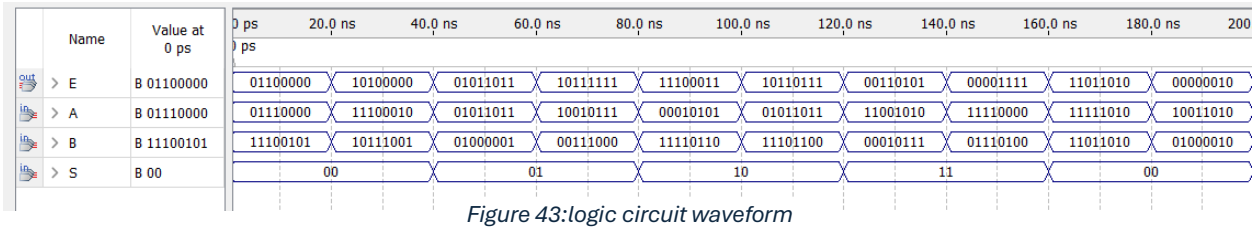


Figure 43:logic circuit waveform

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Figure 44:logic circuit Truth table

ALU

```
1  module ALU (output [7:0] F ,output cout , input [7:0] A,B, input cin , S2,S1,s0);
2
3  wire [7:0] logic ,arthimtic ;
4
5
6  arthimtic_circuit(arthimtic,cout,A,B,cin,{S1,S0});
7  logic_circuit(logic,A,B,{S1,S0});
8  mux2x1_8_bit main_mx(F,arthimtic,logic,S2);
9
10 endmodule
```

Figure 45:ALU code

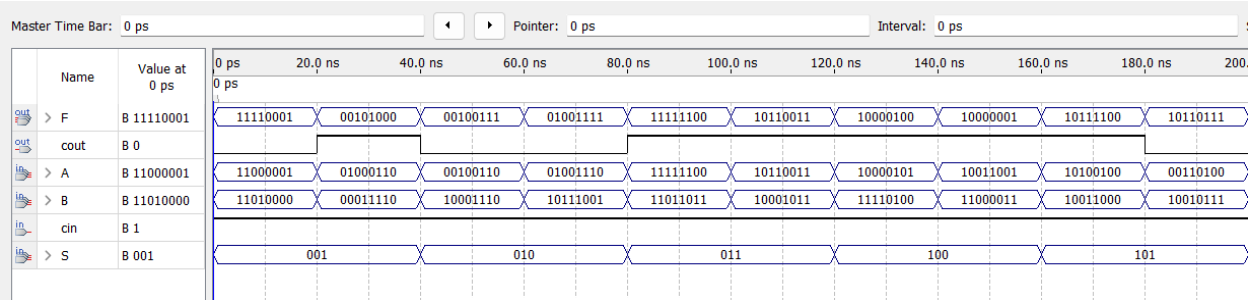


Figure 46:ALU wave

state controller

```
module state_controller(output  [7:0]y,input S,E,qa,qs,clk,reset) ;
reg [7:0] state ;
parameter T0=8'b00000001,
          T1=8'b00000010,
          T2=8'b00000100,
          T3=8'b00001000,
          T4=8'b00010000,
          T5=8'b00100000,
          T6=8'b01000000,
          T7=8'b10000000;
always@(posedge clk )
begin
      if (reset==1) state <= T0 ;
      else case(state)

         T0:   if(qs==1) state <= T1 ;else if (qa==1) state<= T2 ; else state <= T0 ; // if qs ==1 state will become T1 else remain the same
         T1:   state <= T2 ;
         T2:   if(S==1) state <= T4 ; else state <= T3 ;
         T3:   state <= T0 ;
         T4:   state<= T5 ;
         T5:   if (E==0) state<= T6 ; else state <= T0 ;
         T6:   state <= T7 ;
         T7:   state <= T0;
         endcase
end
assign y = state ;
```

Figure 48:state_ controller

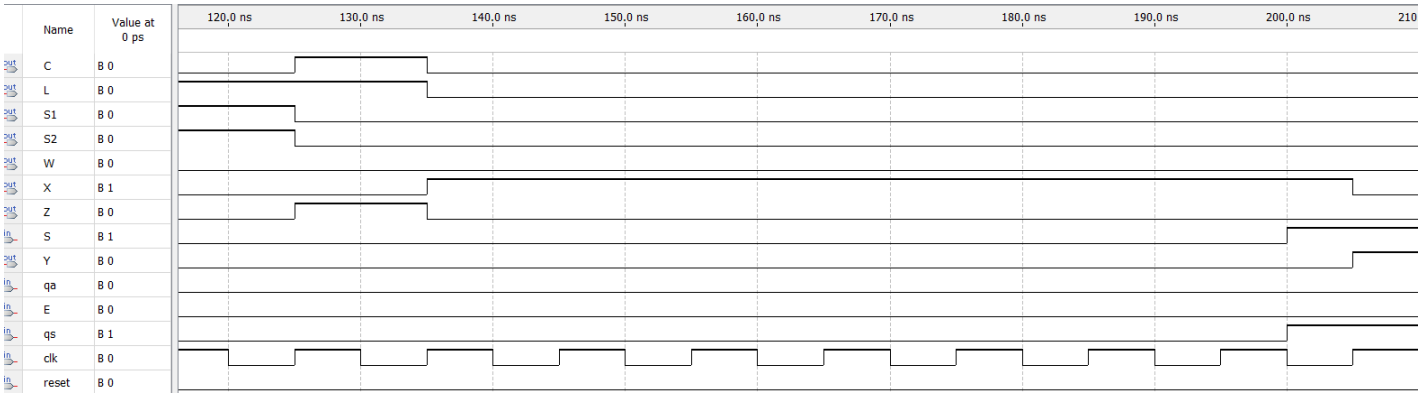
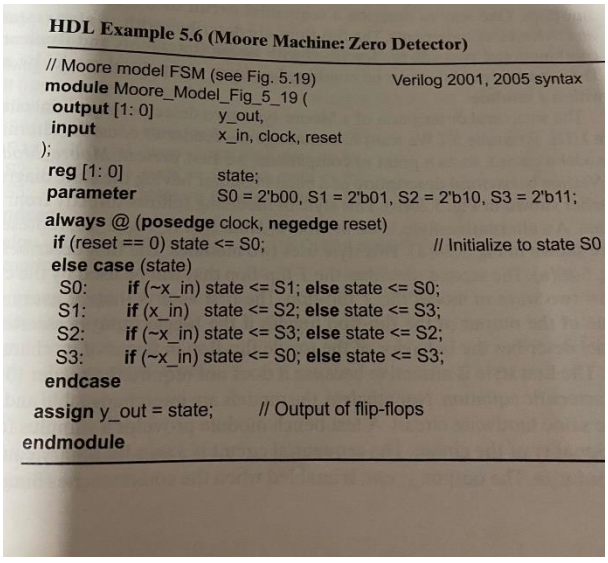


Figure 47:state control waveform



HDL Example 5.6 (Moore Machine: Zero Detector)

```
// Moore model FSM (see Fig. 5.19)          Verilog 2001, 2005 syntax
module Moore_Model_Fig_5_19 (
 output [1: 0]          y_out,
 input                  x_in, clock, reset
);
 reg [1: 0]             state;
 parameter              S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
 always @ (posedge clock, negedge reset)
  if (reset == 0) state <= S0;                          // Initialize to state S0
  else case (state)
   S0:      if (~x_in) state <= S1; else state <= S0;
   S1:      if (x_in)  state <= S2; else state <= S3;
   S2:      if (~x_in) state <= S3; else state <= S2;
   S3:      if (~x_in) state <= S0; else state <= S3;
  endcase
 assign y_out = state;          // Output of flip-flops
endmodule
```

Figure 49:Moore machine code

Hardware state

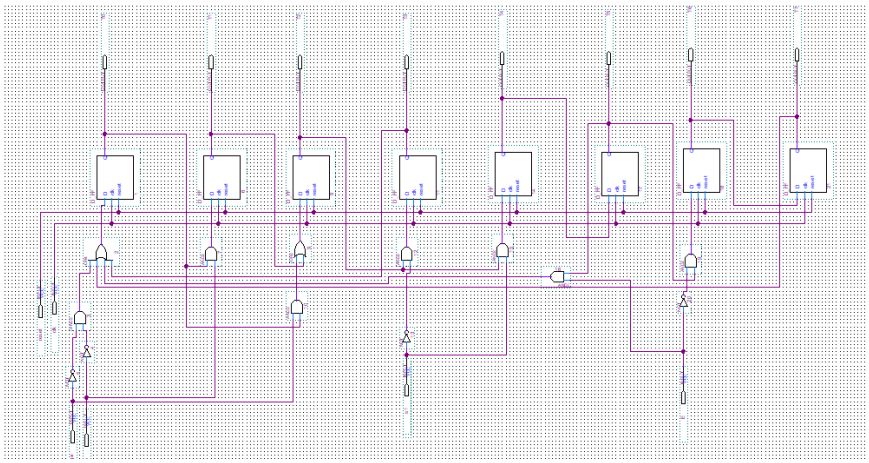


Figure 50:hardware state

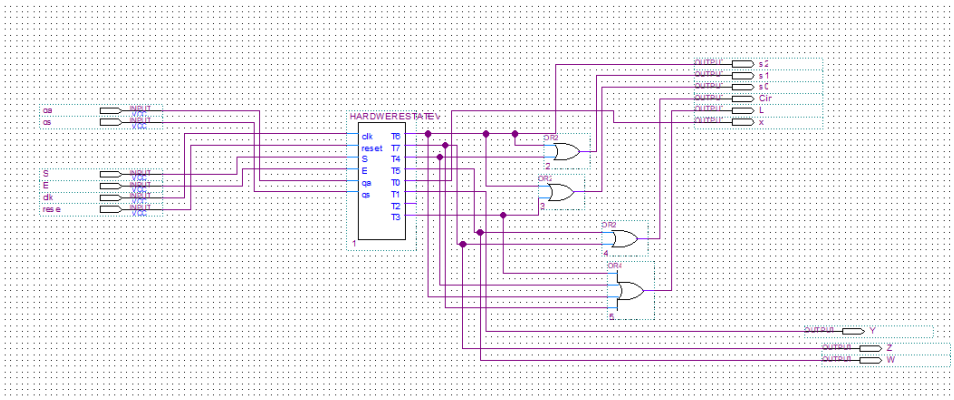


Figure 51 state with output

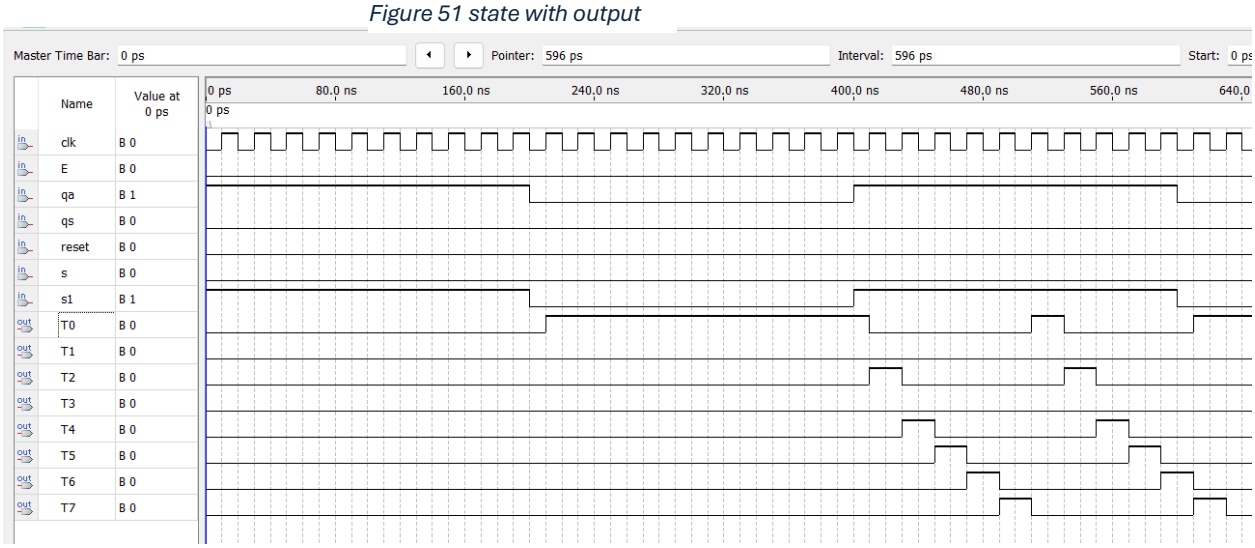


Figure 52hardware state wave

ALU with state controller

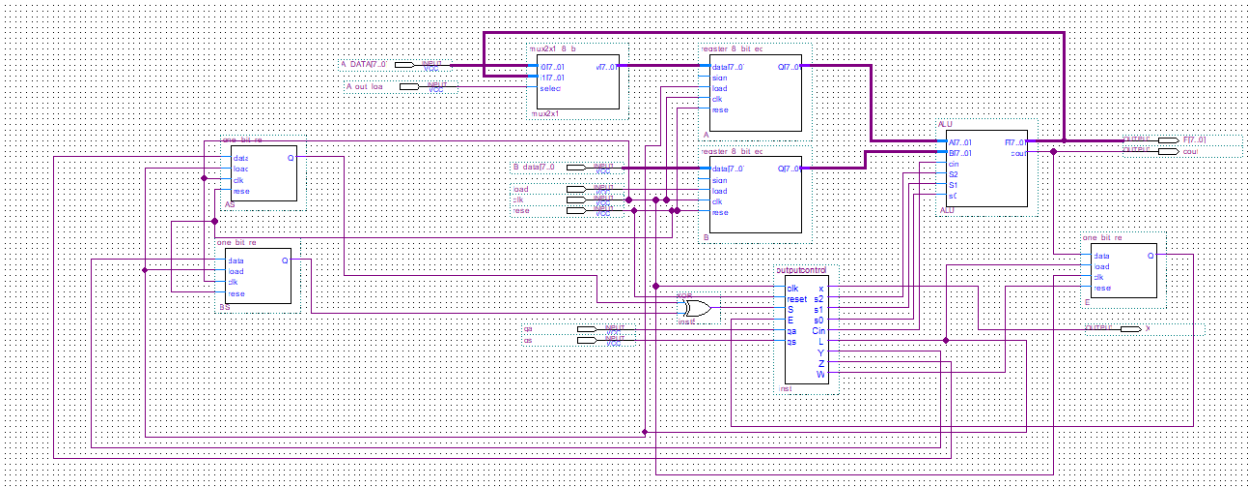


Figure 53:ALU with state control block

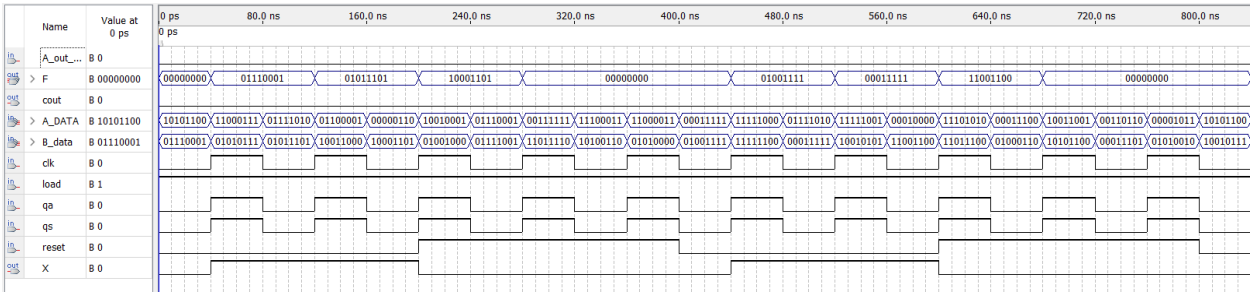


Figure 54 alu with state wave form

ROM8X14

```
module ROM8x14t(output reg X,S2,S1,S0,Cin,L,Y,Z,W,A2,A1,A0,MUX_s1,MUX_s0,input [2:0] data_in );
always @(data_in)

case(data_in)

 3'b000: {X,S2,S1,S0,Cin,L,Y,Z,W,A2,A1,A0,MUX_s1,MUX_s0} =14'b10000000000001;
 3'b001: {X,S2,S1,S0,Cin,L,Y,Z,W,A2,A1,A0,MUX_s1,MUX_s0} = 14'b00000010001001;
 3'b010: {X,S2,S1,S0,Cin,L,Y,Z,W,A2,A1,A0,MUX_s1,MUX_s0} = 14'b00000000010010;
 3'b011: {X,S2,S1,S0,Cin,L,Y,Z,W,A2,A1,A0,MUX_s1,MUX_s0} = 14'b00010100000001;
 3'b100: {X,S2,S1,S0,Cin,L,Y,Z,W,A2,A1,A0,MUX_s1,MUX_s0} = 14'b00101100010101;
 3'b101: {X,S2,S1,S0,Cin,L,Y,Z,W,A2,A1,A0,MUX_s1,MUX_s0} = 14'b10000000100011;
 3'b110: {X,S2,S1,S0,Cin,L,Y,Z,W,A2,A1,A0,MUX_s1,MUX_s0} = 14'b01110100011101;
 3'b111: {X,S2,S1,S0,Cin,L,Y,Z,W,A2,A1,A0,MUX_s1,MUX_s0} = 14'b00001101000000;

 endcase

endmodule
```

Figure 55:ROM8X14 code

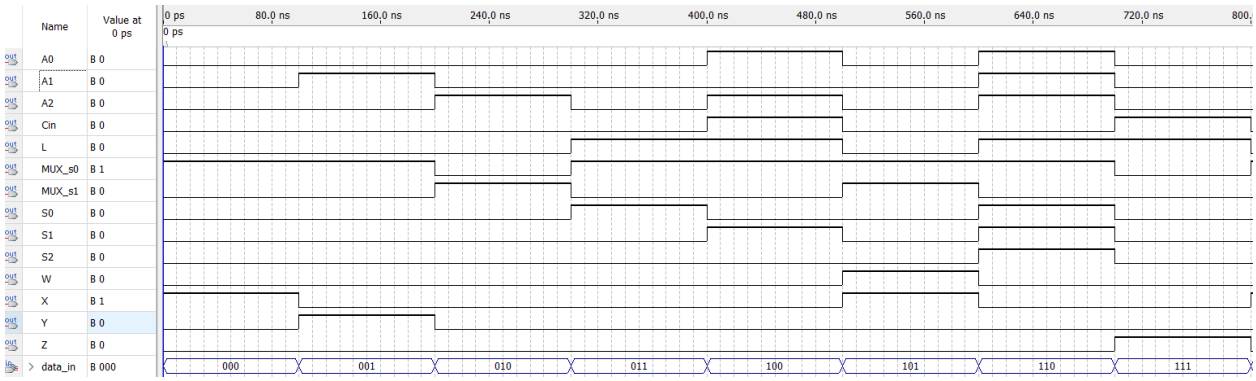


Figure 56:ROM8X14 waveform

CAR

```
1  module CARt(output reg [2:0] adres,input d1,d2,d3,load,increment);
2  always @(*)
3  begin
4  if(load)
5  begin
6  adres[0]=d1;
7  adres[1]=d2;
8  adres[2]=d3;
9  end
10 else
11 begin
12 adres[0]=d1+1'b0;
13 adres[1]=d2+1'b1;
14 adres[2]=d3+1'b1;
15 end
16 end
17 endmodule
18
```

Figure 57:CAR code

Hardware countrol

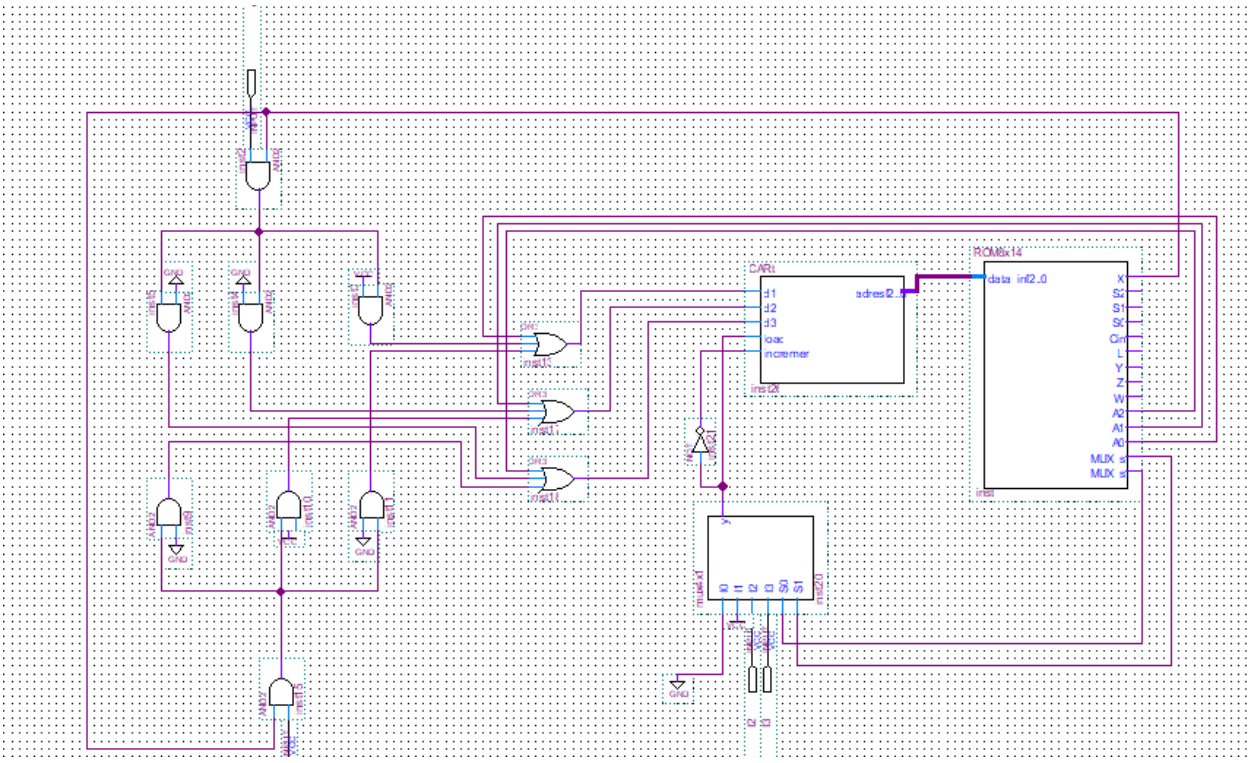


Figure 58:hardwire control implmen

Hardwar control with ALU

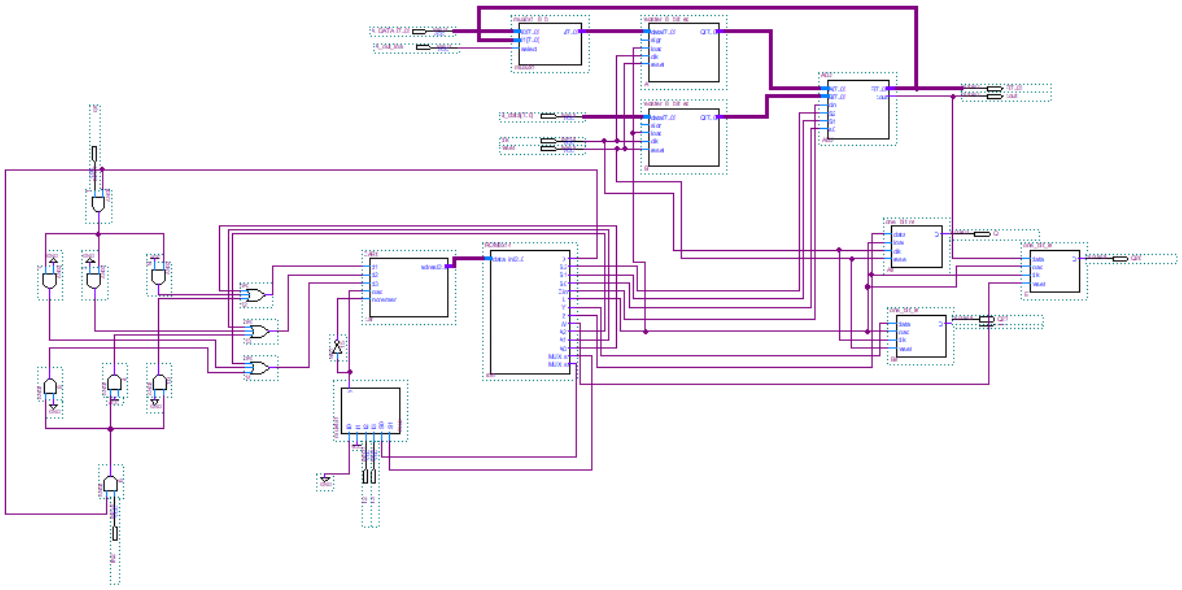


Figure 60:hardware control with alu

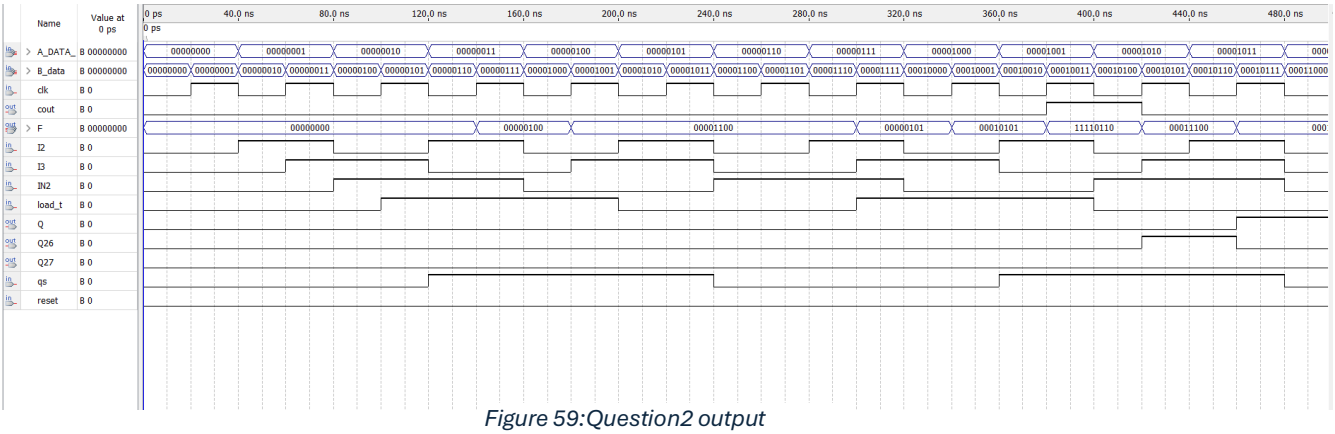


Figure 59:Question2 output

Blocks digrams

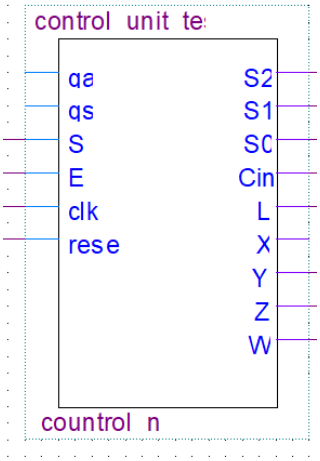


Figure 63:control unit

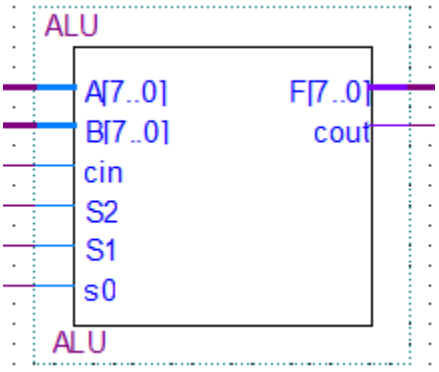


Figure 62:ALU

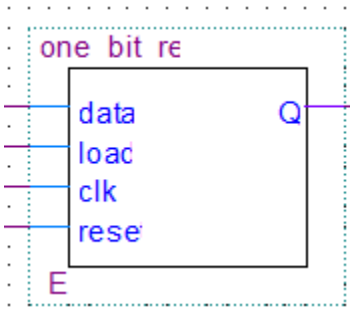


Figure 61:E

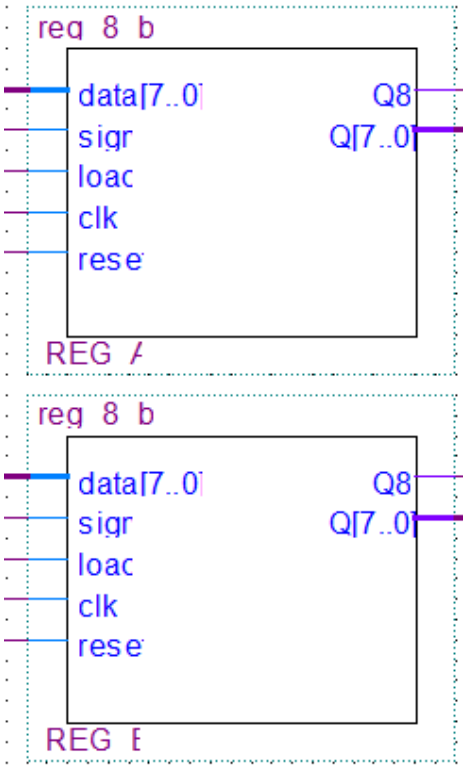


Figure 64:A&B

COMMON use circuit

```
1   module mux2x1(output   out ,
2                 input  i0,i1,
3                 input s);
4
5   assign out = (s) ? i1:i0 ;
6
7    endmodule
8
```

Figure 65:Mux2x1

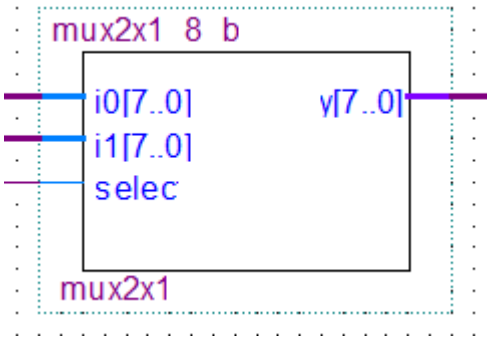


Figure 66MUX2X1 block

```
1   module decoder2x4(input[1:0] in,output reg[3:0] out);
2
3   initial out = 4'b0000;
4   always @(in )
5  begin
6
7     case (in)
8     2'b00: out=4'b0001;
9     2'b01: out=4'b0010;
10    2'b10: out=4'b0100;
11    2'b11: out=4'b1000;
12  endcase
13  end
14
15  endmodule
```

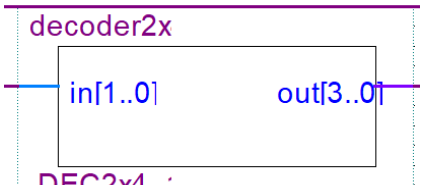


Figure 67:Decode 2X4 block

Figure 68:decoder 2x4

```
module mux4x1(output y ,input I0,I1,I2,I3, input S0,S1 );

wire y0,y1,y2,y3  ;
and a0(y0,I0,~S0,~S1),
    a1(y1,I1,S0,~S1),
    a2(y2,I2,~S0,S1),
    a3(y3,I3,S0,S1);
or o1(y,y0,y1,y2,y3);



endmodule
```

Figure 69:MUX4x1

```
 Smodule D_FF (output reg Q , input D,clk,reset ) ;

 always @(posedge clk )
begin
 if(reset == 1)
    Q <= 1'b0 ;
 else
    Q <=  D ;

 end


 endmodule
```

Figure 70:Dflipflop

```
module decoder3x8strcture (output [7:0]t_out,input[2:0] tin );
wire n1,n2,n3;
not a(n1,tin[0]);
not b(n2,tin[1]);
not c(n3,tin[2]);

    and (t_out[0], n1, n2, n3);  // 000
    and (t_out[1], tin[0], n2, n3);  // 001
    and (t_out[2], n1, tin[1], n3);  // 010
    and (t_out[3], tin[0], tin[1], n3);  // 011
    and (t_out[4], n1, n2, tin[2]);  // 100
    and (t_out[5], tin[0], n2, tin[2]);  // 101
    and (t_out[6], n1, tin[1], tin[2]);  // 110
    and (t_out[7], tin[0], tin[1], tin[2]);  // 111

endmodule
```

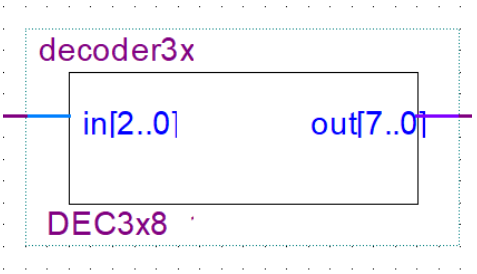


Figure 71:Decoder3x8 BLOCK

Figure 72:Decoder 3X8

```
1  module T_FF(output reg Q  ,input T,clk,reset);
2
3
4  always@(posedge clk )
5  begin
6     if(reset == 1)
7        begin
8           Q <= 1'b0 ;
9        end
10    else
11       begin
12          Q <= Q ^ T ;
13       end
14 end
15
16 //assign nQ = ~Q;
17
18 endmodule
```

Figure 73:Tflipflop