MANO BASIC COMPUTER

ENG \ Ali Khamis
ENG \ Fares Hazem
ENG \ Ahmed Mohamed
ENG \ Mahmoud Abdelsamad
ENG \ Saleh Gamal
ENG \ Abdallah Alsayed
ENG \ Ahmed Shehata

MODERATOR :
ENG \ Kamel Mohamed

INTRODUCTION

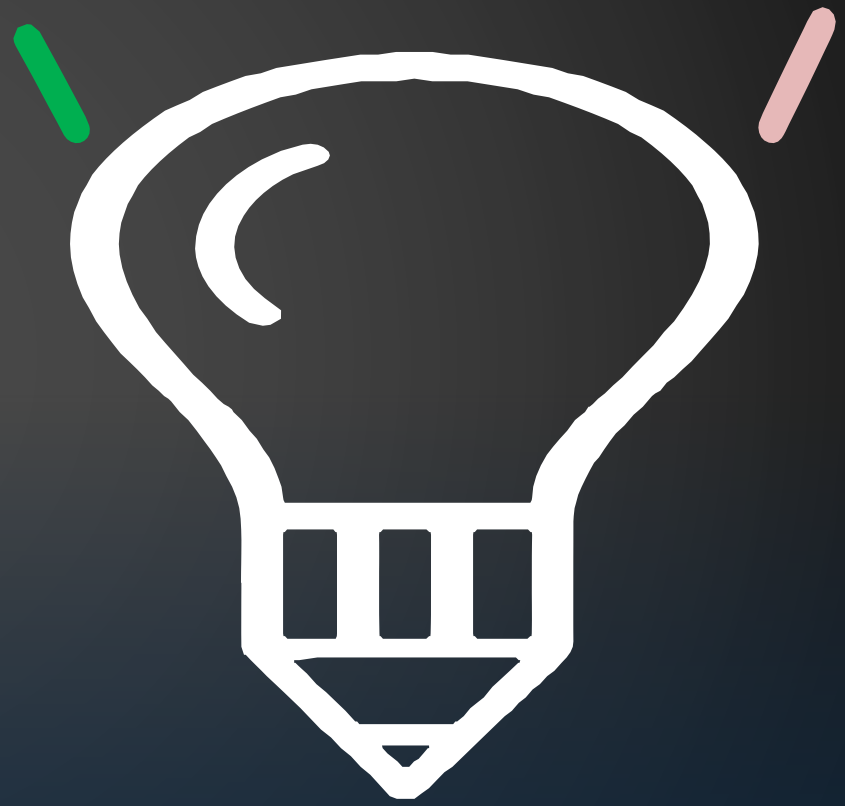
- *The Mano basic computer is a classic example of a simple computer architecture. It is a 16-bit computer.*
- *The Mano basic computer consists of several main components, including a central processing unit , memory, and a clock.*
- *We used verilog language to program this project*

VERILOG

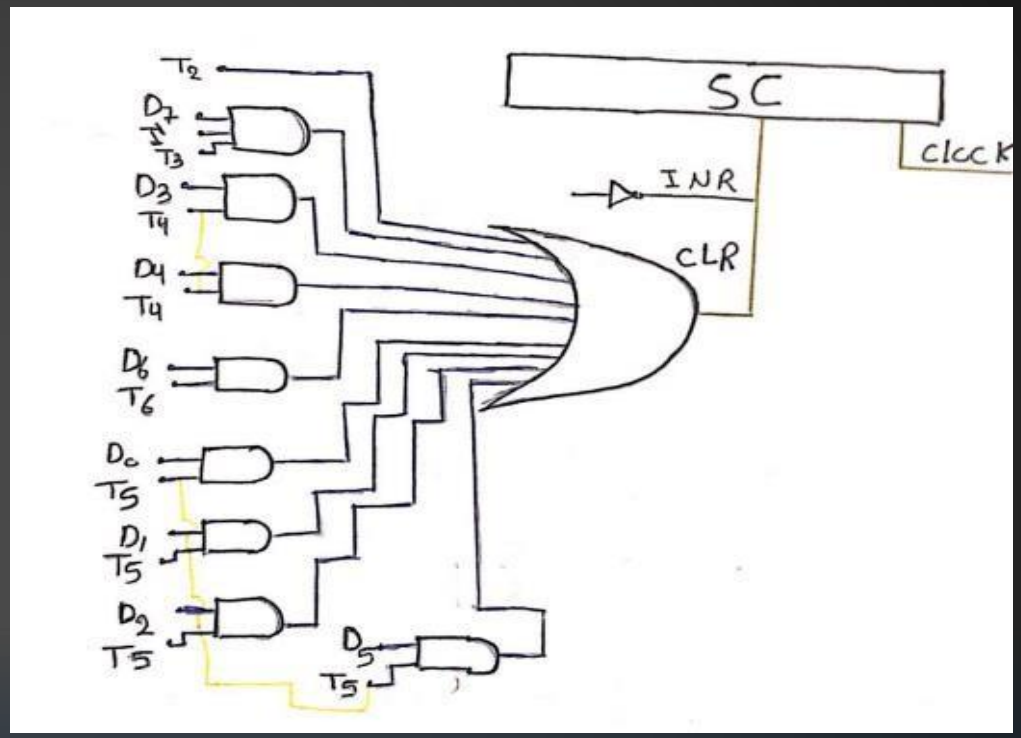
- *Once we have created the necessary modules, we can simulate the behavior of the Mano basic computer using a Verilog simulator. This allows us to test the functionality of the computer and verify that it operates as expected. We can also use Verilog to synthesize the design into actual hardware, which can be programmed onto a field programmable gate array (FPGA) or other hardware platform*

INSTRUCTION CODE & DESIGN

- *01 Sequence counter*
- *02 Memory*
- *03 Register*
- *04 Control Register*
- *05 ALU*



```
1  module SC_Control(CLR, T, D,I,B,S);
2
3      output CLR,S;
4      input [7:0] T, D;
5      input I;
6      input [15:0] B;
7
8      wire a1, a2, a3, a4, a5, o1, o2;
9      wire r,In;
10
11     assign I=~I ;
12     assign  r=  D[7] & T[3] &In;
13     assign o1 = D[4] | D[3];
14     assign a1 = o1 & T[4];
15     assign a2 = D[7] & T[3];
16     assign a3 = D[6] & T[6] ;
17     assign o2 = D[2] | D[1] | D[0] | D[5];
18     assign a4 = o2 & T[5];
19     assign CLR = a1 | a2 | a3 | a4 ;
20     assign S= B[0] & r;
21 endmodule
```



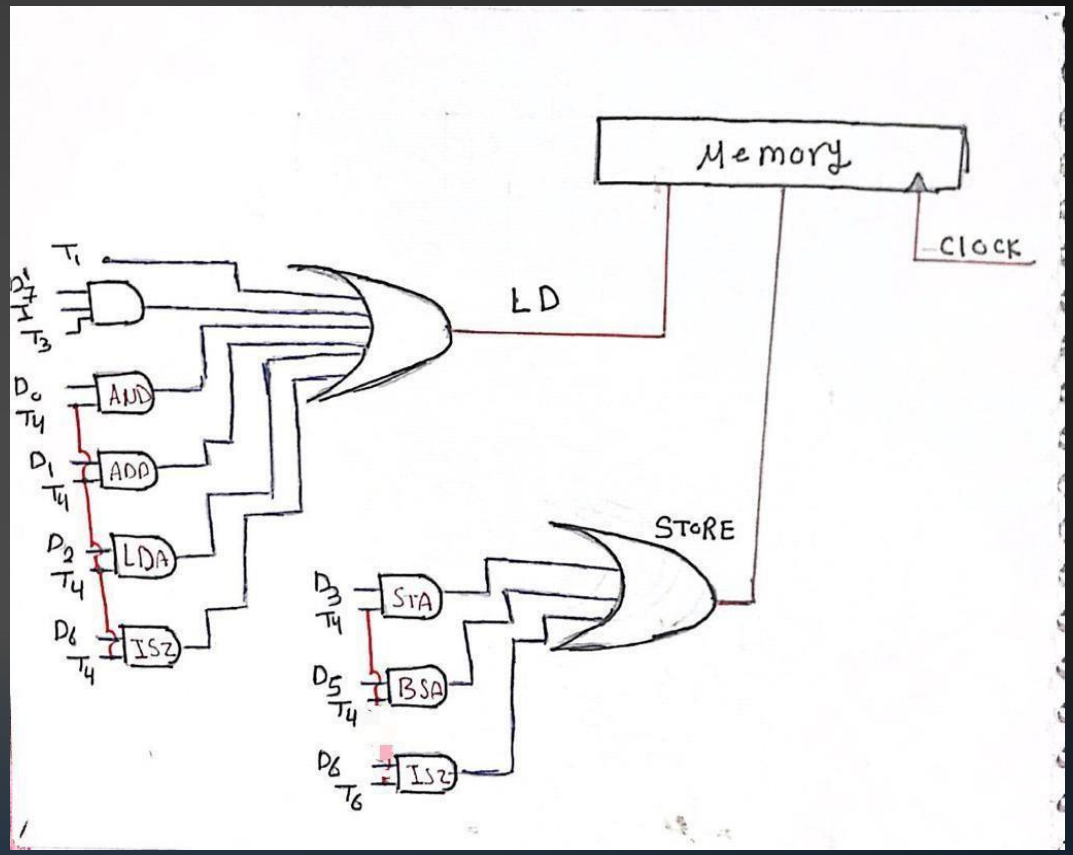
SEQUENCE COUNTER

```
module RAM (CLK,read,AR,write,INDATA,OUTDATA );
    input CLK,read,write;
    input [15:0]INDATA;
    input [11:0] AR;
    output reg [15:0] OUTDATA;

    reg [15:0] RAM_4K [4096:0];

    initial
    begin
    RAM_4K[0] = 16'h7020;       //clear AC
    RAM_4K[1] = 16'h7800;       //clear AC
    RAM_4K[2] = 16'h7200;       //CMA
    RAM_4K[3] = 16'h2004;       //LDA
    RAM_4K[4] = 16'h2002;       //ADD
    RAM_4K[5] = 16'h7001;       //HLT
    end

    always@* begin
        if (write)
            RAM_4K[AR] <= INDATA;
        else if(read)
            OUTDATA = RAM_4K[AR];
        end

endmodule
```

MEMORY

```
1  module Memory_Control(Str, Load, T, D, I);
2
3      input  I;
4      input [7:0] T, D;
5      output Str, Load;
6
7      wire  Dn7;
8      wire  A2, A3, A4, A5, A6, A7, A8, A9;
9
10     assign Dn7 = ~D[7];
11     assign A3 = D[3] & T[4];
12     assign A4 = D[5] & T[4];
13     assign A5 = D[6] & T[6];
14     assign A6 = Dn7 & I & T[3];
15     assign A7 = D[6] & T[4];
16     assign A9 = D[2] | D[1] | D[0];
17     assign A8 = A9 & T[4];
18     assign Str =  A3 |A4 | A5;
19     assign Load = T[1] | A6 | A7 | A8;
20
21 endmodule
```

MEMORY CONTROL



DESIGN

REGISTERS

- ✓ *AR REGISTER*
- ✓ *PC REGISTER*
- ✓ *DC REGISTER*
- ✓ *AC REGISTER*

AR Register

```
module REG_11(Q, INR, Data, LD, CLK, CLR);

    output reg [11:0] Q;
    input [11:0] Data;
    input INR, LD, CLK, CLR;

    initial Q = 0;

    always @(posedge CLK)

    begin
        if(CLR)
            Q <= 16'b0;
        else if(LD)
            Q <= Data;
        else if(INR)
            Q <= Q + 1;
    end

endmodule
```

Pc Register

```
module REG_11(Q, INR, Data, LD, CLK, CLR);

    output reg [11:0] Q;
    input [11:0] Data;
    input INR, LD, CLK, CLR;

    initial Q = 0;

    always @(posedge CLK)

    begin
        if(CLR)
            Q <= 16'b0;
        else if(LD)
            Q <= Data;
        else if(INR)
            Q <= Q + 1;
    end

endmodule
```

DR Register

```
module REG_11(Q, INR, Data, LD, CLK, CLR);

    output reg [11:0] Q;
    input [11:0] Data;
    input INR, LD, CLK, CLR;

    initial Q = 0;

    always @(posedge CLK)

    begin
        if(CLR)
            Q <= 16'b0;
        else if(LD)
            Q <= Data;
        else if(INR)
            Q <= Q + 1;
    end

endmodule
```

Ac Register

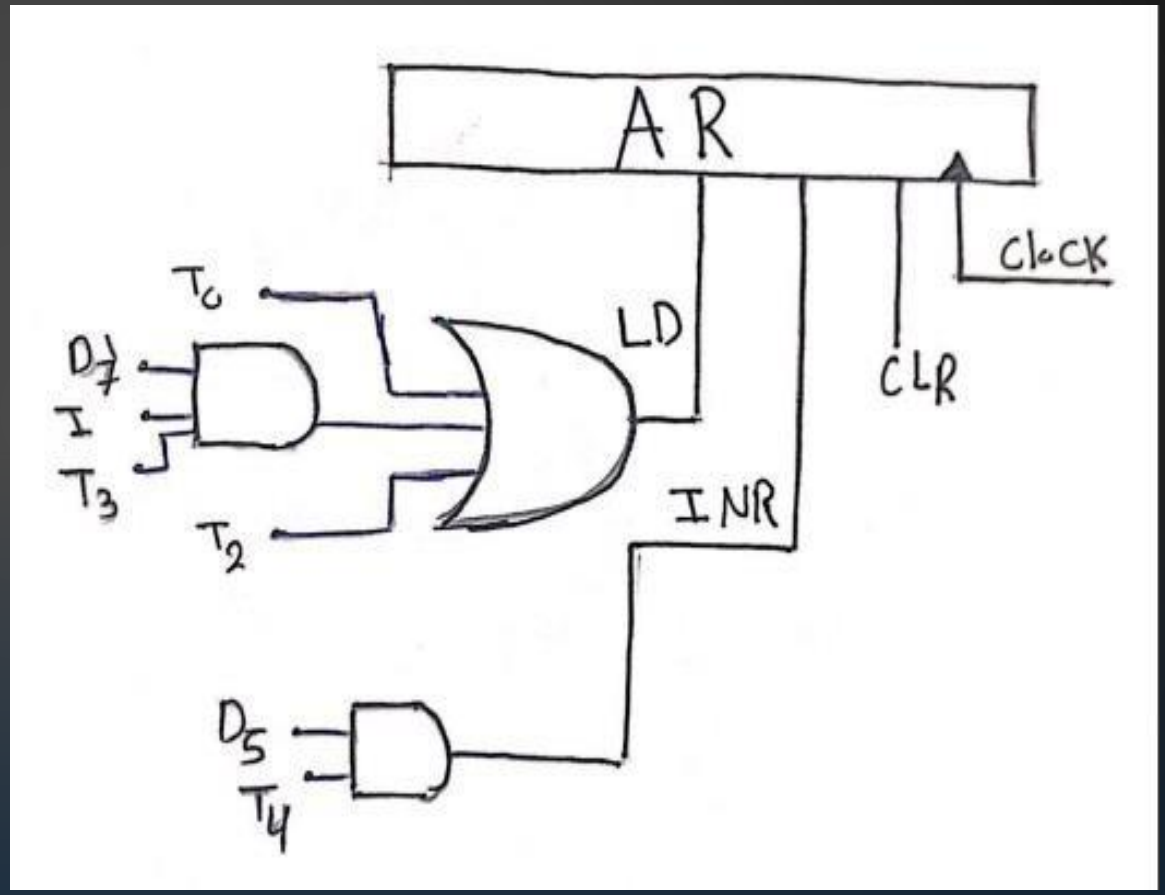
```
module REG_16(Q, INR, Data, LD, CLK, CLR);

    initial Q = 0;

    output reg [15:0] Q;
    input [15:0] Data;
    input INR, LD, CLK, CLR;

    always @(posedge CLK)
    begin
        if(CLR)
            Q <= 16'b0;
        else if(LD)
            Q <= Data;
        else if(INR)
            Q <= Q + 1;
    end
endmodule
```

CONTROL REGISTERS

AR CONTROL

```
1  module AR_Control(LD, CLR, INR, T, D, I);
2
3      input  I;
4      input [7:0] T, D;
5      output LD, CLR, INR;
6
7      wire D7n, Rn;
8      wire a1, a2, a3;
9
10     assign D7n = ~D[7];
11
12     assign a1 = D7n & I & T[3];
13     assign a2 = T[2] & Rn;
14     assign a3 = Rn & T[0];
15     assign CLR = T[0];
16     assign INR = D[5] & T[4];
17     assign LD = a1 | a2 | a3;
18
19 endmodule
```

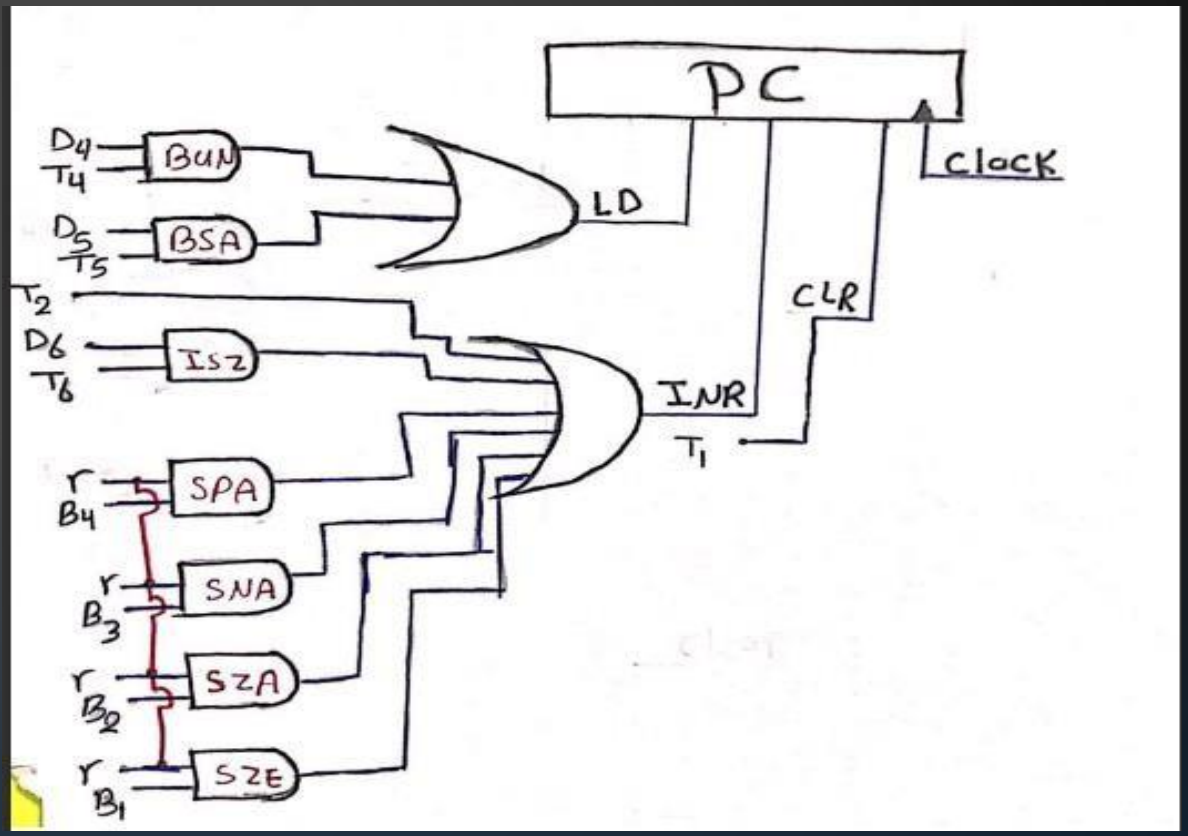
DESIGN



PC CONTROL

```
module PC_control(CLR, INR, Load, I, E, T, D, B, DR, AC);
    output CLR, INR, Load;
    input I, E;
    input [7:0] T, D;
    input [15:0] B;
    input [15:0] DR, AC;

    wire In,AC15n,En,a1,a3,r,p,a6 ,a7 ,a8 ,a9 ,a10 ,o1 , a11 ,o2;
    wire a15,a16;

    assign In= ~I;
    assign AC15n=~AC[15];
    assign En=~E;
    assign a1=  T[1];
    assign CLR=T[1];
    assign a3= T[2];
    assign p= I & T[3] & D[7];
    assign r= In & D[7] & T[3];
    //
    //
    //
    assign a8 = B[3] & AC[15];      //SNA
    assign a9 = AC15n & B[4];       //SPA
    assign a10 = En & B[1];         //SZE
    assign o1 = a8 | a9 | a10;
    assign a11 = r & o1;            // all SNA , SPA , SZE


    assign Load = a15 | a16;        // if one of these conditions is true then pc = loa
    assign INR = a1 | a3 | a11 | p; // if one of these conditions is true then pc <= pc

endmodule
```

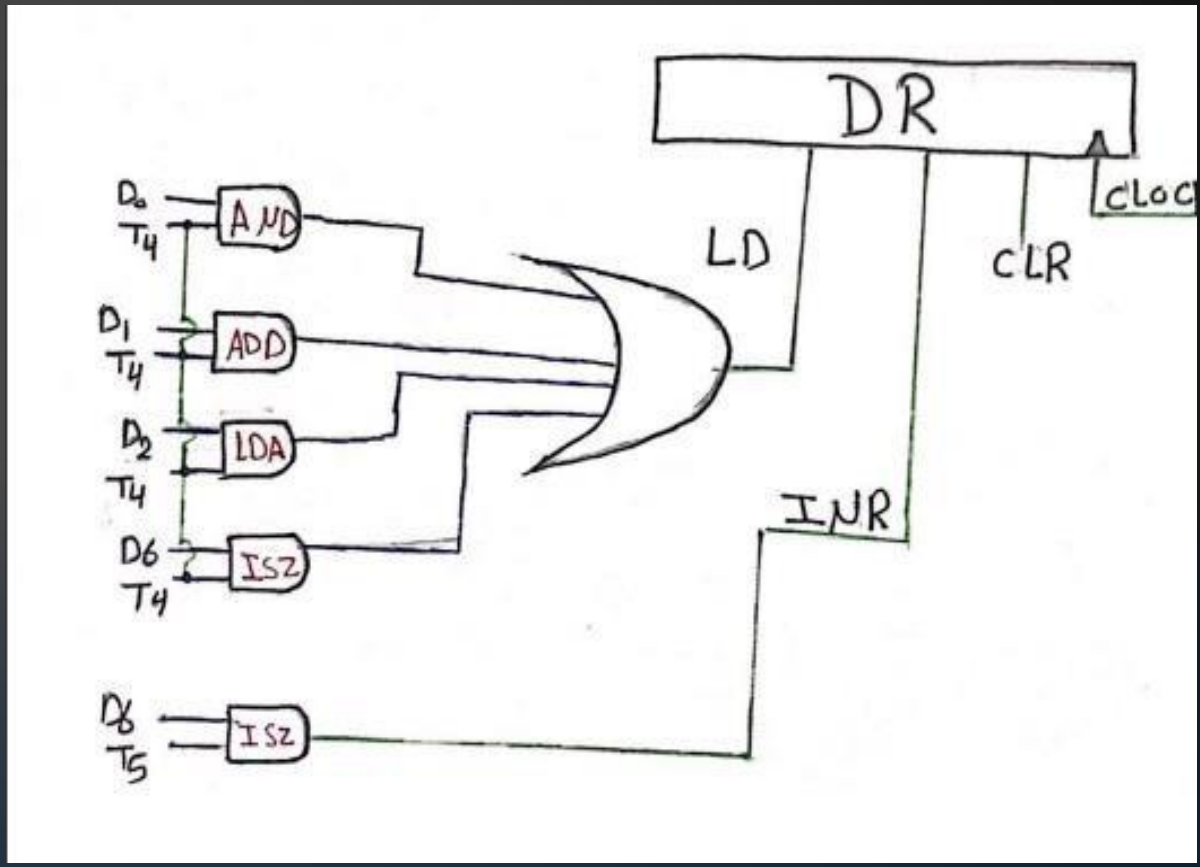
DESIGN



DR CONTROL

```
1  module DR_Control(Load, INR, T, D);
2
3      input [7:0] T,D;
4      output Load, INR;
5
6      wire ToLoad;
7
8      assign INR = D[6] & T[5];
9      assign ToLoad = (D[0] | D[1] | D[2] | D[6]);
10     assign Load = T[4] & ToLoad;
11
12 endmodule
```

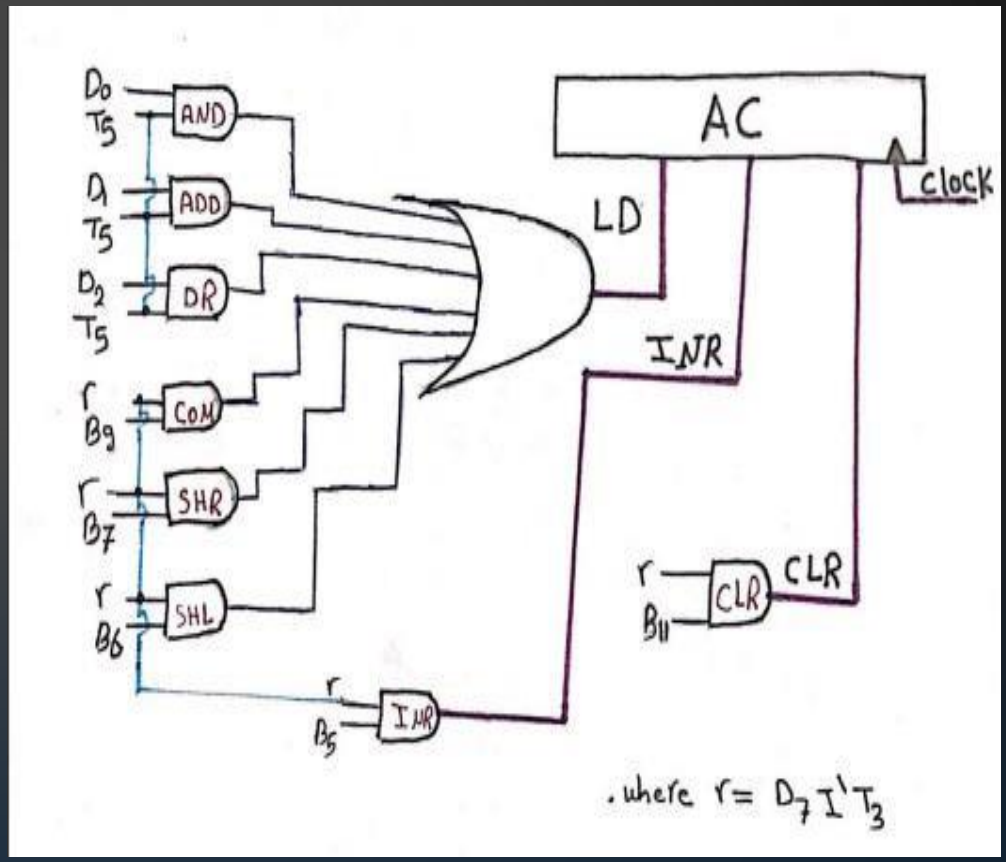
DESIGN



AC CONTROL

```
module AC_Control(AND, COM, SHR, SHL,INPT, DR, ADD, LD, INR, CLR, T, D, B, I);

    output AND, SHR, SHL , COM, INPT, DR, ADD, LD, INR, CLR;
    input[7:0] T, D;
    input[15:0] B;
    input I;

    wire In;
    wire r, p;
    assign In = ~I;

    assign p = D[7] & I & T[3];
    assign r = D[7] & In & T[3];

    assign AND = D[0] & T[5];
    assign ADD = T[5] & D[1];
    assign DR = T[5] & D[2];
    assign COM = B[9] & r;
    assign SHL = B[6] & r;
    assign SHR = B[7] & r;
    assign INR = B[5] & r;
    assign INPT = B[11] & p;

    assign LD =( AND | ADD | DR | SHL | SHR | COM | INPT );
    assign CLR = r & B[11];
endmodule
```

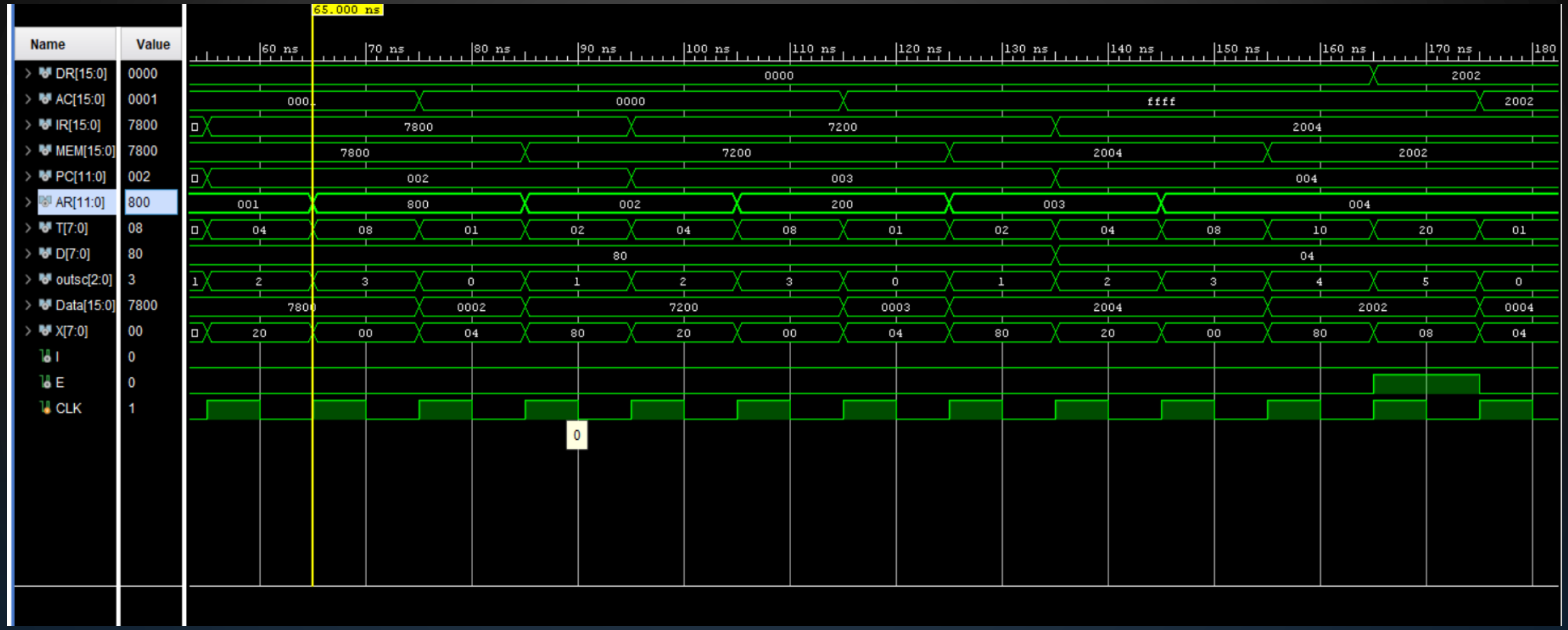
DESIGN



ALU

```
module ALU(AND,ADD, LDA, CMA,E,AC, DR,cout , ACDATA);
    input AND,ADD, LDA, CMA,E;     //instructions signals
    input [15:0] AC, DR;
    output cout ;
    output [15:0]  ACDATA;

    wire [15:0] and16,add16,lda16,cma16;      // turning the instructions signals into 16-bit signal to be anded
    wire [15:0] AND1,AND2,AND3,AND4 ,SUM;          //AND gate of each instruction
    wire CARRY;

    //AND OPERATION
    assign and16 = (AND ? 16'hffff :16'b0);
    assign AND1 = AC & and16 & DR;

    //ADD OPERTION , {CARRY,SUM} is the result of a full-adder
    assign {CARRY,SUM} = AC + DR ;
    assign cout = CARRY;

    assign add16 = (ADD ? 16'b1111111111111111 :16'b0);
    assign AND2 = SUM & add16 ;

    //LDA OPERATION (load AC with DR content )
    assign lda16 = (LDA ? 16'hffff :16'b0);
    assign AND3 = lda16 & DR ;

    //CMA OPERATION (Complement AC content)
    assign cma16 = (CMA ? 16'hffff : 16'b0);
    assign AND4 = (~AC) & cma16;
        wire o1 ,o2;
    assign ACDATA = AND1 | AND2| AND3| AND4;    // All AND gates signals to be into an OR gate that results in AC
endmodule
```

SIMULATION


65.000 ns
Name
Value
DR[15:0] 0000
AC[15:0] 0001
IR[15:0] 7800
MEM[15:0] 7800
PC[11:0] 002
AR[11:0] 800
T[7:0] 08
D[7:0] 80
outsc[2:0] 3
Data[15:0] 7800
X[7:0] 00
I 0
E 0
CLK 1
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns
130 ns
140 ns
150 ns
160 ns
170 ns
180
0
0000
2002
0001
0000
ffff
2002
7800
7200
2004
7800
7200
2004
2002
002
003
004
001
800
002
200
003
004
04 08 01 02 04 08 01 02 04 08 10 20 01
80
04
1 2 3 0 1 2 3 0 1 2 3 4 5 0
7800
0002
7200
0003
2004
2002
0004
20 00 04 80 20 00 04 80 20 00 80 08 04