

模型机设计报告

班级 计科 2002 班 姓名 杨鹏宇 学号 202004061409

一、设计目的

完整，连贯地运用《电路与电子学》所学到的数电知识，熟练掌握现代 EDA 工具基本使用方法，为后续课程学习和今后从事相关工作打下良好基础。

二、设计内容

1.按照给定的数据通路，数据格式和指令系统，使用 EDA 工具设计一台用硬连线逻辑控制的简易计算机。

2.对所设计的计算机的性能指标进行分析，整理设计报告。

三、详细设计

3.1 设计的整体架构

数据通路：

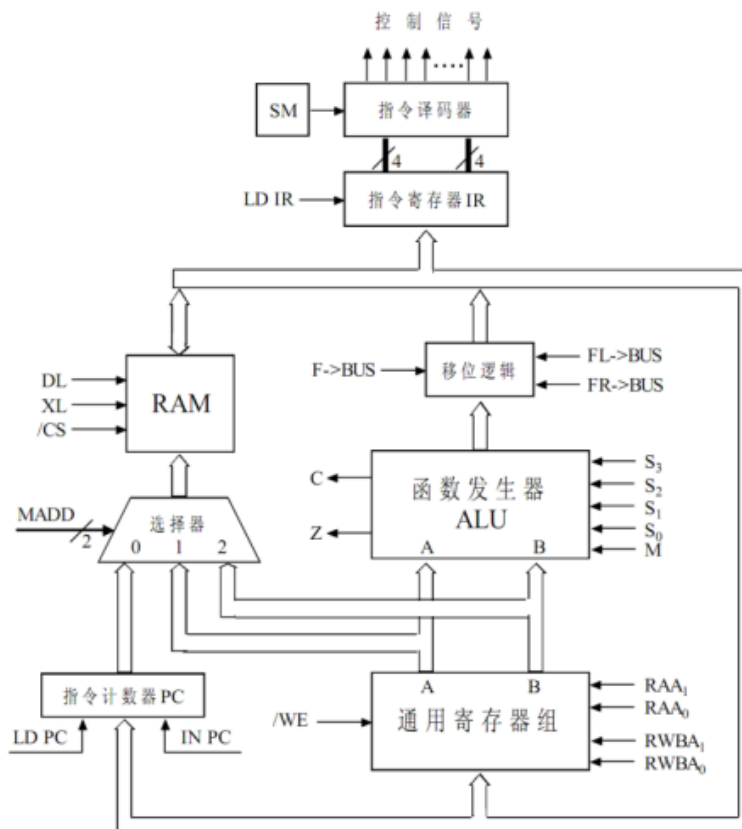


图 1 模型机数据通路

指令编码：

表 1 指令系统表

汇编符号	功能	编码
MOV R1, R2	$(R2) \rightarrow R1$	1100 R1 R2
MOV M, R2	$(R2) \rightarrow (C)$	1100 11 R2
MOV R1, M	$((C)) \rightarrow R1$	1100 R1 11
ADD R1, R2	$(R1) + (R2) \rightarrow R1$	1001 R1 R2
SUB R1, R2	$(R1) - (R2) \rightarrow R1$	0110 R1 R2
AND R1, R2	$(R1) \& (R2) \rightarrow R1$	1011 R1 R2
NOT R1	$\neg (R1) \rightarrow R1$	0101 R1 XX
RSR R1	$(R1)$ 循环右移一位 $\rightarrow R1$	1010 R1 00
RSL R1	$(R1)$ 循环左移一位 $\rightarrow R1$	1010 R1 11
JMP add	add \rightarrow PC	0011 00 00, address
JZ add	结果为 0 时 add \rightarrow PC	0011 00 01, address
JC add	结果有进位时 add \rightarrow PC	0011 00 10, address
IN R1	(开关 7-0) $\rightarrow R1$	0010 R1 XX
OUT R1	$(R1) \rightarrow$ 发光二极管 7-0	0100 R1 XX
NOP	$(PC) + 1 \rightarrow PC$	0111 00 00
HALT	停机	1000 00 00

控制信号：

表 2 基本控制信号及功能表

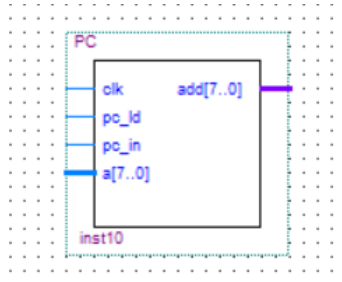
序号	信号	功能
1	IN PC	与 LD PC 配合使用，为 1 时 PC 加 1 计数，为 0 时加载 BUS 上的数据。
2	LD PC	当 IN PC=1 允许对 PC 加 1 计数，否则允许把 BUS 上的数据打入 PC。
3	LD IR	允许把 BUS 上的数据打入指令寄存器 IR。
4	/WE	允许把 BUS 上的数据打入通用寄存器组，低电平有效。
5	F \rightarrow BUS	ALU 的运算结果通过移位逻辑直接送到总线 BUS 的对应位。
6	FRL \rightarrow BUS	ALU 的运算结果通过移位逻辑循环左移一位送到总线 BUS，且 F7 送 Cr。
7	FRR \rightarrow BUS	ALU 的运算结果通过移位逻辑循环右移一位送到总线 BUS，且 F0 送 Cr。
8	/CS	允许访问存储器，低电平有效。
9~10	MADD	存储器 RAM 地址来源。0：指令计数器，1：通用寄存器 A 口，2：B 口。
11	DL	读存储器 RAM。
12	XL	写存储器 RAM。
13	M	M=1，表示 ALU 进行逻辑运算操作，否则进行算术操作。
14~17	S ₃ ~S ₀	使 ALU 执行各种运算的控制位。
18	HALT	此位为“1”时停机，下次输入人工操作。

3.2 各模块的具体实现

(1) 指令计数器 PC

端口设计：



clk 为时钟信号，pc_ld，pc_in 控制 PC 计数器+1 或载入地址，由控制信号发生器给出信号，a 连接总线，add 为输出端口，将指令地址经选择器后传输给存储器。



功能实现：

指令计数器存储当前指令在 RAM 中存放的地址。

模型机读取指令时，根据 PC 中的指令地址，从存储器中读出指令并写入指令寄存器，指令读取后，PC 中的地址+1，指向下一条指令。JMP, JZ, JC 指令执行时，写入需要下一条指令在存储器中的地址。

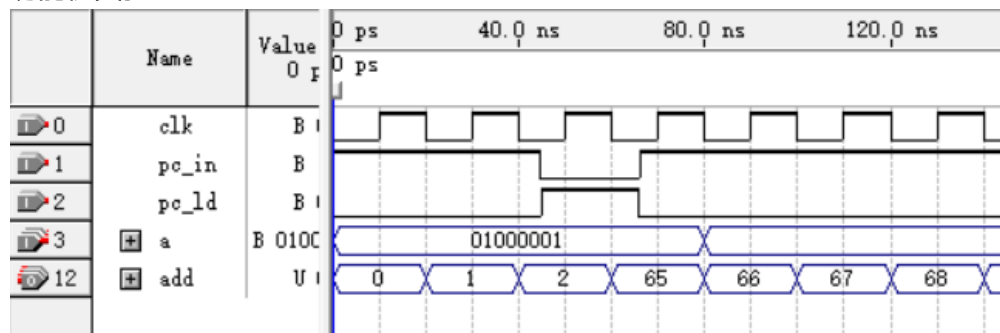
CLK	pc_inc	pc_ld	功能
	1	0	add[7..0] 中数据自加1
	0	1	a[7..0] 写入 add[7..0]

```

module PC(clk,pc_ld,pc_in,a,add);
    input clk,pc_ld,pc_in;
    input [7:0] a;
    output [7:0] add;
    reg [7:0] add;
    always@(negedge clk)
    begin
        if(pc_in&&!pc_ld) add[7:0]<=add[7:0]+8'b00000001;
        else if(!pc_in&&pc_ld) add[7:0]<=a[7:0];
        else;
    end
end
endmodule

```

功能仿真验证：

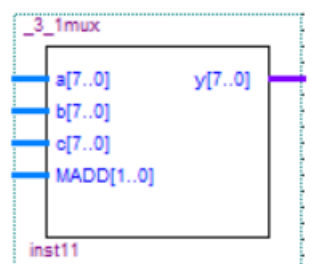


当 pc_in=0, pc_ld=1 时载入地址数据，pc_in=1, pc_ld=0 时，计数器+1。PC 计数器+1 和载入地址均在在时钟下降沿进行。PC 计数器可以正常工作。

(2) 3 选 1 多路复用器

端口设计：

madd 为选择信号，由控制信号发生器给出信号，从 a, b, c 三个端口选择数据从 y 输出。a, b, c 分别连接 PC 计数器，寄存器组的 s 口和 d 口，y 将数据(指令地址)输出给存储器。

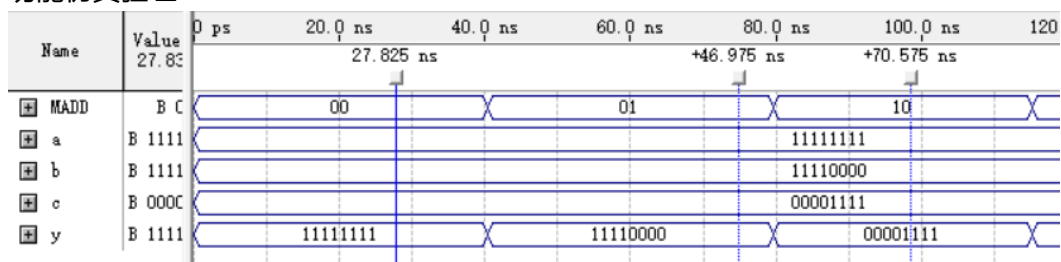


功能实现：

多路复用器根据选择信号输出三个输入信号中的一个。

```
module _3_lmux(a,b,c,y,MADD
);
    input [7:0]a;
    input [7:0]b;
    input [7:0]c;
    input [1:0]MADD;
    output [7:0]y;
    reg [7:0]y;
    always@ (a,b,c,MADD)
    begin
        case (MADD)
            2'b00: y[7:0]=a[7:0];
            2'b01: y[7:0]=b[7:0];
            2'b10: y[7:0]=c[7:0];
            default: y="XXXXXXXX";
        endcase
    end
endmodule
```

功能仿真验证：

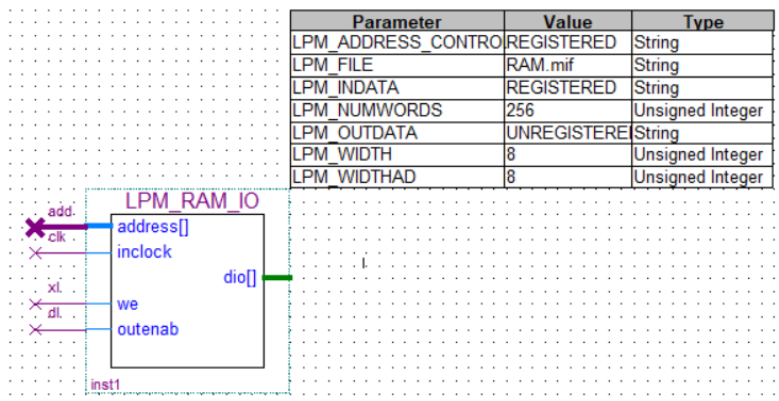


分别使用 madd 选择 a, b, c 输入的数据, y 能够正确输出对应的数据, 3 选 1 多路复用器可以正常工作。

(3) 存储器 RAM

端口设计：

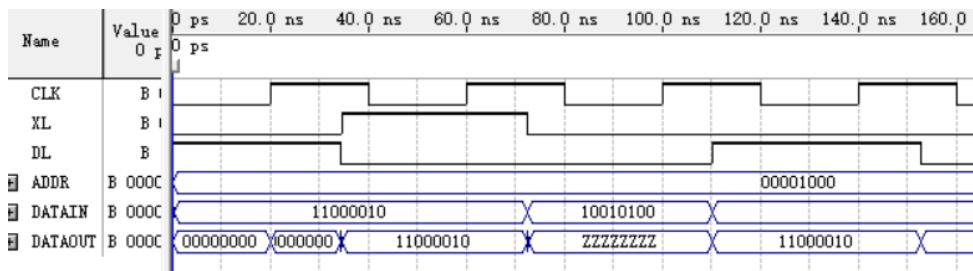
add 为需要操作的存储器地址, clk 为时钟信号, DL, XL 控制读写, 由控制信号发生器给出。dio 端口读或写数据。



功能实现：

存储器存储指令或其他数据。可以读出或写入数据。执行读取时, 输入一个地址, DL=1, XL=0, 输出存储器对应地址的内容。写入时 DL=0, XL=1, 将数据写入存储器对应的地址, DL=XL=0 时为高阻态。

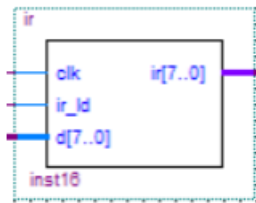
功能仿真验证：



向地址单元 00001000 写入一个数据后读出，输出正确，存储器可以正常工作。

(4) 指令寄存器 IR

端口设计：



clk 为时钟信号，ir_ld 为控制写入的信号，由控制信号发生器给出，d 接总线，ir 将寄存的指令输出给指令译码器。

功能实现：

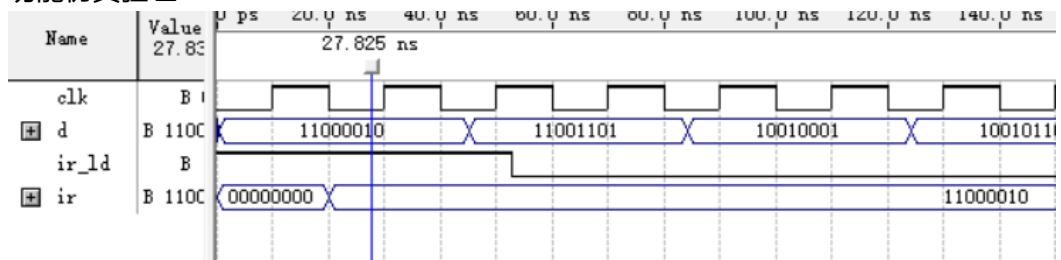
指令寄存器总线上的由存储器读出的指令在时钟下降沿写入寄存器，供指令译码执行使用。

```

module IR(clk,ir_ld,d,ir);
    input clk,ir_ld;
    input [7:0] d;
    output [7:0] ir;
    reg [7:0] ir;
    always@(negedge clk)
        if(ir_ld) ir<=d;
        else ;
endmodule

```

功能仿真验证：

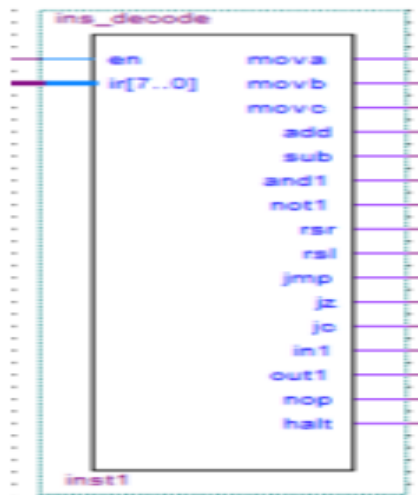


当 ir_ld=1 时可写入数据，ir_ld=0 时不写入，指令寄存器可以正常工作。

(5) 指令译码器

端口设计：

en 为使能信号，由 sm 给出，ir 接收指令寄存器的数据，16 个输出对应 16 个指令信号，传输给控制信号发生器执行指令。



功能实现：

指令译码器将指令编码译码为指令信号并传给控制信号发生器，进行指令的执行。

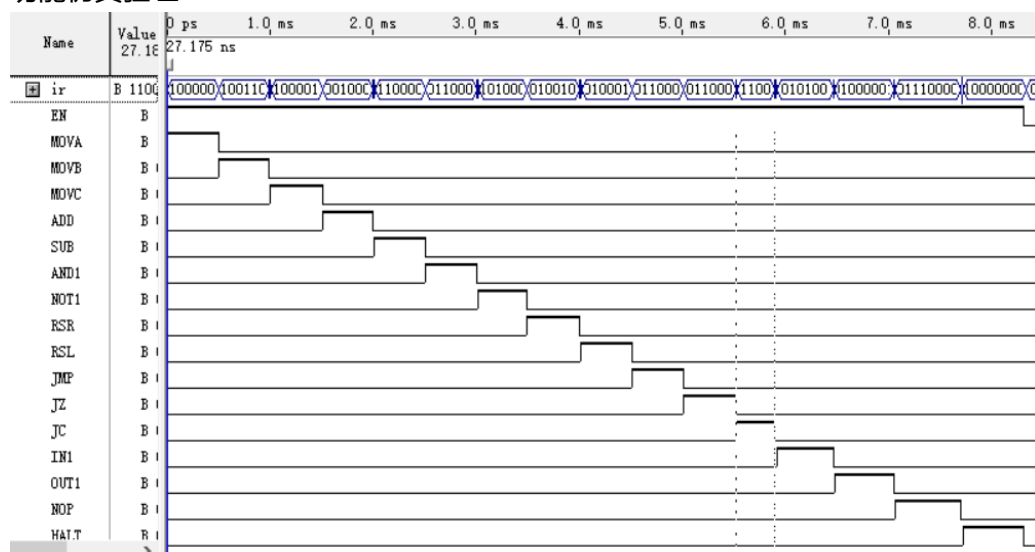
MOVA=0;MOVB=0;MOVC=0;ADD=0;SUB=0;AND1=0;NOT1=0;RSR=0;RSL=0;JMP=0;JZ=0;JC=0;IN1=0;OUT1=0;NOP=0;HALT=0;

```
if (EN)
begin
  if(ir[7:4]==4'b1100)
  begin
    if(ir[3]&ir[2]) MOVB=1;
    else if(ir[1]&ir[0]) MOVC=1;
    else MOVA=1'b1;
  end
  else if(ir[7:4]==4'b1001) ADD=1;
  else if(ir[7:4]==4'b0110) SUB=1;
  else if(ir[7:4]==4'b0111) AND1=1;
  else if(ir[7:4]==4'b0101) NOT1=1;
  else if(ir[7:4]==4'b0100)
  begin
    if(~ir[1]&~ir[0]) RSR=1;
    else RSL=1;
  end
  else if(ir[7:4]==4'b0011)
  begin
    JC=ir[1];
    JZ=ir[0];
    JMP=!ir[1]&&!ir[0];
  end
  else if(ir[7:4]==4'b0010) IN1=1;
  else if(ir[7:4]==4'b0100) OUT1=1;
  else if(ir[7:4]==4'b0111) NOP=1;
  else if(ir[7:4]==4'b1000) HALT=1;
  else ;
end
else ;
```

表 1 指令系统表

汇编符号	功能	编码
MOV R1, R2	(R2) → R1	1100 R1 R2
MOV M, R2	(R2) → (C)	1100 11 R2
MOV R1, M	((C)) → R1	1100 R1 11
ADD R1, R2	(R1) + (R2) → R1	1001 R1 R2
SUB R1, R2	(R1) - (R2) → R1	0110 R1 R2
AND R1, R2	(R1) & (R2) → R1	1011 R1 R2
NOT R1	/ (R1) → R1	0101 R1 XX
RSR R1	(R1) 循环右移一位 → R1	1010 R1 00
RSL R1	(R1) 循环左移一位 → R1	1010 R1 11
JMP add	add → PC	0011 00 00, address
JZ add	结果为 0 时 add → PC	0011 00 01, address
JC add	结果有进位时 add → PC	0011 00 10, address
IN R1	(开关 7-0) → R1	0010 R1 XX
OUT R1	(R1) → 发光二极管 7-0	0100 R1 XX
NOP	(PC) + 1 → PC	0111 00 00
HALT	停机	1000 00 00

功能仿真验证:

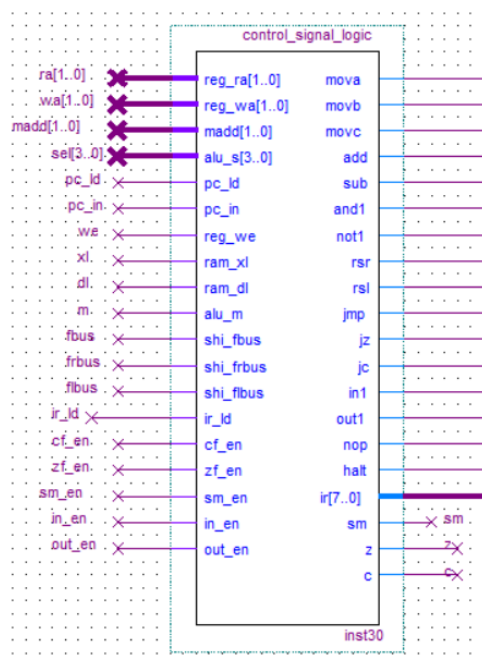


依次输入 16 个信号的指令编码，指令信号能够正确输出，指令译码器可以正常工作。

(6) 控制信号发生器

端口设计:

16 个输入端口接收 16 个指令信号, sm 控制指令执行时间, 接入状态寄存器的 z, c 用于决定 JZ, JC 指令是否执行跳转, reg_ra, reg_wa, reg_we 控制寄存器工作; pc_ld, pc_in 控制指令计数器工作; madd 为多路复用器的选择信号, 选择指令地址来源; ir_ld 控制指令寄存器写入指令, alu_m, sel 控制运算单元 ALU 的工作; cf_en, zf_en 控制状态寄存器写入 cf, zf; shi_fbus, shi_frbus, shi_flbus 控制移位逻辑工作; ram_dl, ram_xl 控制存储器的读写; in_en, out_en 为输入输出三态门的控制信号。



功能实现：

根据指令译码器传入的指令信号发出控制信号，完成指令的执行过程。

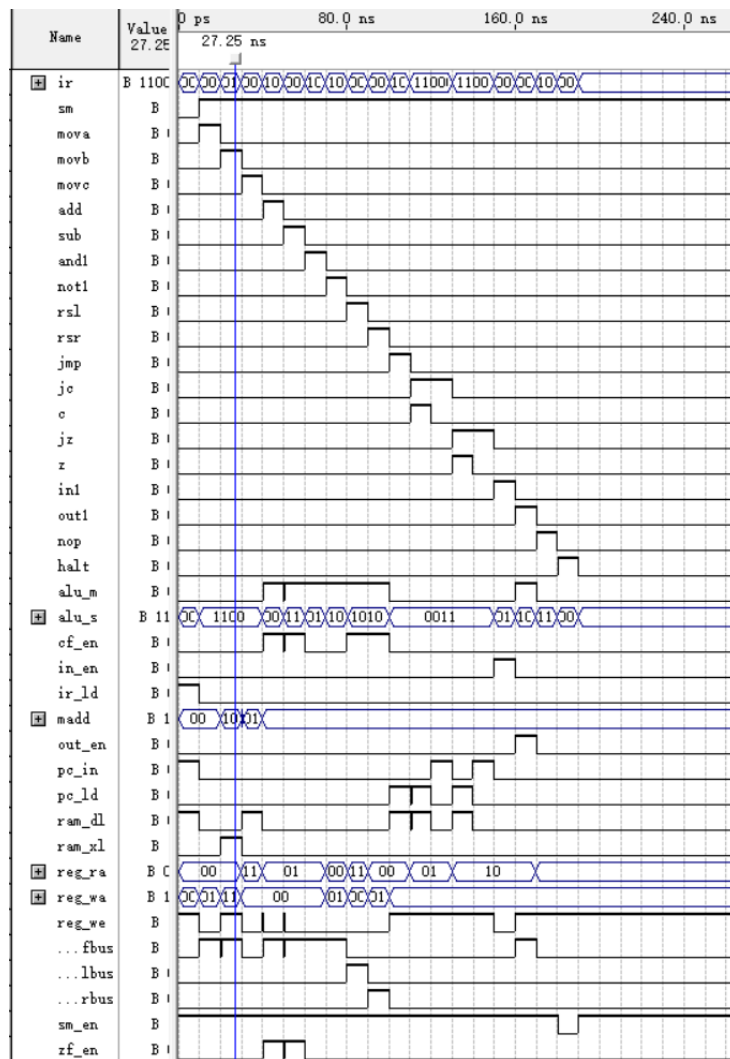
```
begin
    sm_en<=!halt;
    //
    alu_m<=add||sub||andl||notl||rsr||rsl||outl;
    cf_en<=add||sub||rsr||rsl;
    zf_en<=add||sub;
    alu_s[3:0]<=ir[7:4];
    //
    shi_fbus<=movb||movb||add||sub||andl||notl||outl;
    shi_frbus<=rsr;
    shi_flbus<=rsl;
    //
    ram_dl<=movc||jmp||(jz&&z)|| (jc&&c)||!sm;
    ram_xl<=movb;
    //
    ir_ld<=!sm;
    //
    reg_we<=! (movb||movc||add||sub||andl||notl||rsl||rsr||inl)||!sm;
    reg_wa<=ir[3:2];
    reg_ra[1:0]<=ir[1:0];
    //
    pc_ld<=jmp||(jz&&z)|| (jc&&c);
    pc_in<=(jz&&!z)|| (jc&&!c)||!sm;
    //
    if(movb&&sm) madd<=2'b10;
    else if(movc&&sm) madd<=2'b01;
    else if(!sm) madd<=2'b00;
    else madd<=2'b00;
    //in,out
    in_en<=inl;
    out_en<=outl;
end
```

各控制信号如下表：

表 2 基本控制信号及功能表

序号	信号	功能
1	IN PC	与 LD PC 配合使用，为 1 时 PC 加 1 计数，为 0 时加载 BUS 上的数据。
2	LD PC	当 IN PC=1 允许对 PC 加 1 计数，否则允许把 BUS 上的数据打入 PC。
3	LD IR	允许把 BUS 上的数据打入指令寄存器 IR。
4	/WE	允许把 BUS 上的数据打入通用寄存器组，低电平有效。
5	F→BUS	ALU 的运算结果通过移位逻辑直接送到总线 BUS 的对应位。
6	FRL→BUS	ALU 的运算结果通过移位逻辑循环左移一位送到总线 BUS，且 F7 送 C _f 。
7	FRR→BUS	ALU 的运算结果通过移位逻辑循环右移一位送到总线 BUS，且 F0 送 C _f 。
8	/CS	允许访问存储器，低电平有效。
9~10	MADD	存储器 RAM 地址来源。0：指令计数器，1：通用寄存器 A 口，2：B 口。
11	DL	读存储器 RAM。
12	XL	写存储器 RAM。
13	M	M=1，表示 ALU 进行逻辑运算操作，否则进行算术操作。
14~17	S ₃ ~S ₀	使 ALU 执行各种运算的控制位。
18	HALT	此位为“1”时停机，下次输入人工操作。

功能仿真验证：



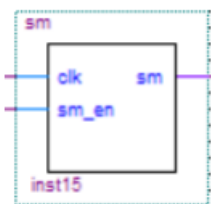
测试 16 条指令，并逐一确认对应的控制信号是否正确。可以观察到 in, out 指令对应的使能信号正确；mov 类指令 reg_ra, reg_wa 等信号正确；运算类指令执行时 s, m, fbus, flbus, flbus, we, reg_ra, reg_wa 控制信号正确；转移类指令执行时，指令计数器的 pc_in, pc_ld 等信号正确；存储器的 xl, dl 等信号在 movb, movc 指令下正确，状态寄存器的 zf_en, cf_en 指令在运算类指令执行时正确；sm 为 0 时指令寄存器 ir_ld 正确；halt 执行时 sm_en 信号正确。

根据功能仿真验证，控制信号发生器可以正常工作。

(7) sm

端口设计：

clk 为时钟信号，下降沿 sm 翻转，sm_en 为使能信号，由控制信号发生器提供。



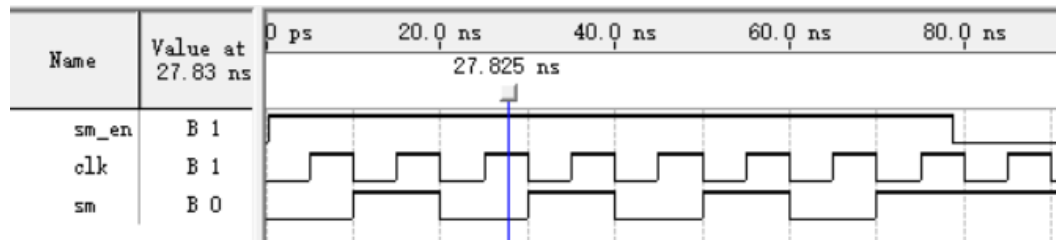
功能实现：

模型机指令的执行在两个时钟周期完成。使用 sm 标识指令读取周期和指令执行周

期。sm 在时钟下降沿翻转，并受使能信号控制。

```
module SM(clk, sm, sm_en);
    input clk, sm_en;
    output reg sm;
    always@(negedge clk)
    begin
        if(sm_en) sm<=!sm;
        else ;
    end
endmodule
```

功能仿真验证：

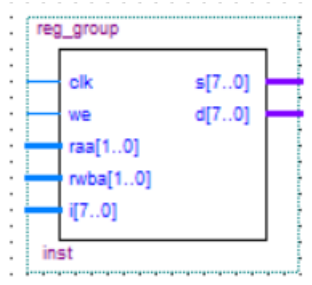


sm 在 sm_en=1 时，在时钟下降沿翻转，可以正常工作。

(8) 通用寄存器组

端口设计：

clk 为时钟信号；we 控制是否写入，raa, rwba 分别提供一个寄存器组编号，读出或写入该寄存器组数据，这三个信号由控制信号发生器给出；i 连接总线，接收数据；s, d 连接运算器 ALU 和多路复用器，输出数据。



功能实现：

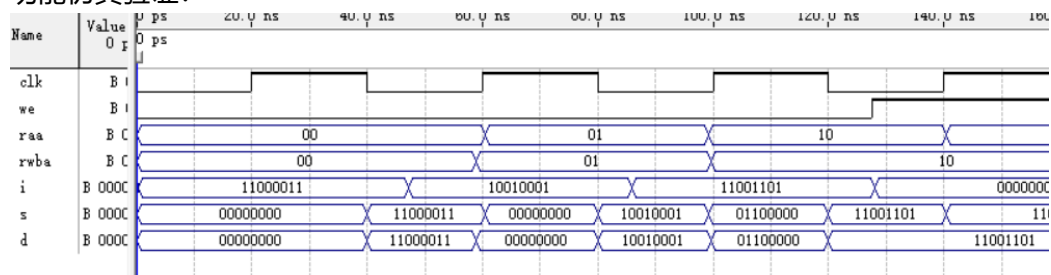
通用寄存器组有三个寄存器，通过 ra, wa 的控制从 s 口，d 口读出对应寄存器编号的数据，在 we 和 ra, wa 的控制下向寄存器写入数据。

```

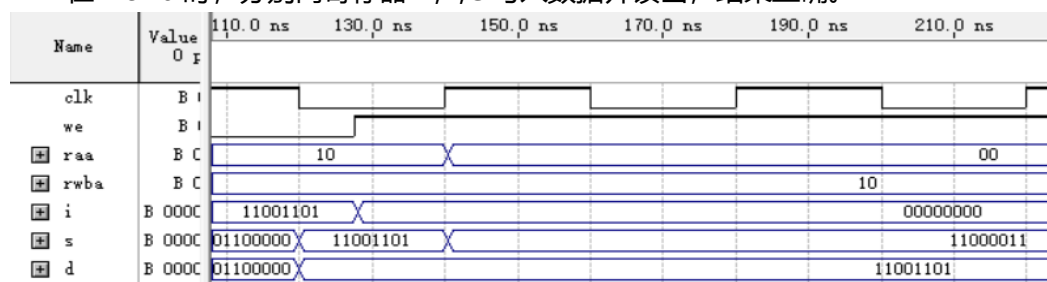
module reg_group(clk,we,raa,rwba,i,s,d);
    input clk,we;
    input [1:0]raa;
    input [1:0]rwba;
    input [7:0]i;
    output reg [7:0]s;
    output reg [7:0]d;
    reg [7:0]a;
    reg [7:0]b;
    reg [7:0]c;
    initial c[7:0]=8'b01100000;
    //
    always@(raa or rwba)
    begin
        if(raa==2'b00) s<=a;
        else if(raa==2'b01) s<=b;
        else s<=c;
        if(rwba==2'b00) d<=a;
        else if(rwba==2'b01) d<=b;
        else d<=c;
    end
    //
    always@(negedge clk)
    begin
        if(!we)
        begin
            if(rwba==2'b00) a<=i;
            else if(rwba==2'b01) b<=i;
            else c<=i;
        end
    end
endmodule

```

功能仿真验证:



在 we=0 时, 分别向寄存器 A,B,C 写入数据并读出, 结果正确。

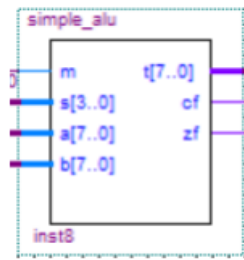


we=1 时, 数据不能写入。修改 raa 与 rwba, d 口读出的是 rwba 提供的寄存器编码对应的寄存器的数据, s 口读出的是 raa 提供的寄存器编码对应寄存器的数据。寄存器组可以正常工作。

(9) 运算单元 ALU

端口设计:

a, b 接收寄存器组从 s, d 口输出的数据; s, m 为运算控制信号, 由控制信号发生器发出; 输出 t 为运算结果, cf, zf 分别为溢出借位状态位和记录 0 的状态位。



功能实现：

ALU 接收寄存器组的数据，并根据控制信号 s 与 m 进行加减，求与，求反的操作，将结果输出。具体功能如下表：

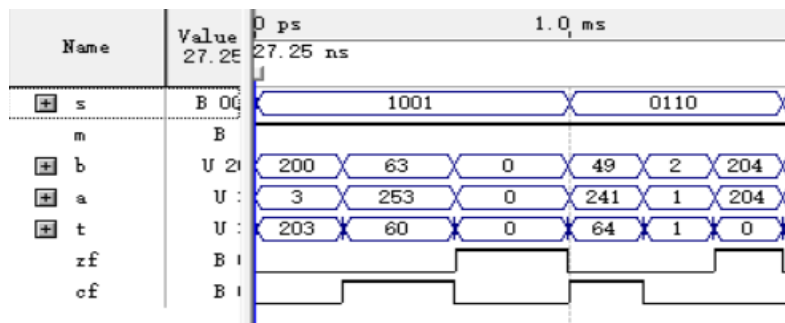
m	s[3..0]	t[7..0]	cf	zf
1	1001	$t=a+b$	有进位, cf=1 无进位, cf=0	和为零, zf=1 和不为零, zf=0
1	0110	$t=b-a$	有借位, cf=1 无借位, cf=0	差为零, zf=1 差不为零, zf=0
1	1011	$t=a\&b$	不影响	不影响
1	0101	$t=\sim b$ (注: b 相反)	不影响	不影响
1	1010 或 0100	$t=b$	不影响	不影响
0	1100	$t=a$	不影响	不影响

```

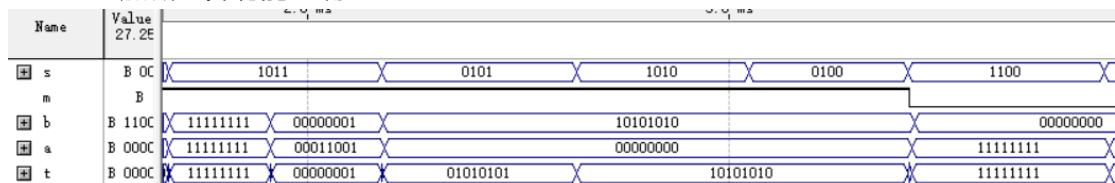
module ALU(a,b,s,m,t,cf,zf);
    input [7:0] a;
    input [7:0] b;
    input m;
    input [3:0] s;
    output cf,zf;
    output [7:0] t;
    reg cf,zf;
    reg [7:0] t;
    always@(m,s,a,b)
    begin
        t=8'b0;
        cf=1'b0;
        zf=1'b0;
        if(m==1)
        begin
            if(s==4'b1001)
            begin
                {cf,t}=a+b;
                if(t==0) zf=1;
                else zf=1'b0;
            end
            else if(s==4'b0110)
            begin
                {cf,t}=b-a;
                if(t==0) zf=1'b1;
                else zf=0;
            end
            else if(s==4'b1011) t=a&b;
            else if(s==4'b0101) t=~b;
            else if(s==4'b1010 || s==4'b0100) t=b;
            else t=a;
        end
        else t=a;
    end
endmodule

```

功能仿真验证：



加减运算功能正确。

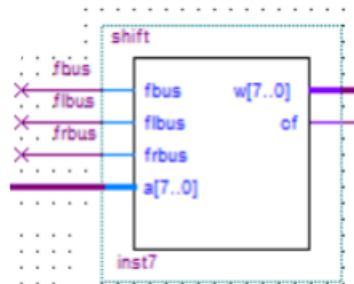


求与，求反及左右移位、输出、mov 指令直接通过的结果正确。运算单元 ALU 可以正常工作。

(10) 移位逻辑

端口设计：

a 接收运算单元 ALU 输出的数据，fbus, frbus, flbus 控制左移右移和不移位，由控制信号发生器给出信号，cf 为移位溢出，w 连接总线输出数据。



功能实现：

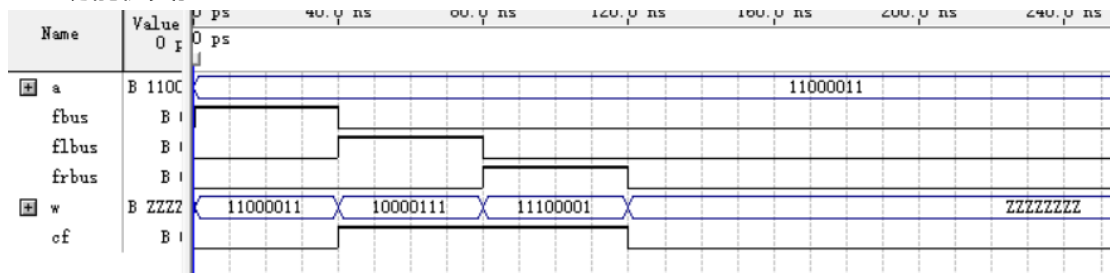
fbus=1, flbus=frbus=0 时，数据直接通过；fbus=flbus=0, frbus=1 时数据右移一位，cf 输出移出位；fbus=frbus=0, flbus=1 时数据左移一位，cf 输出移出位。fbus=flbus=frbus=0 时输出高阻态。

```

module shift_logic(a,fbus,flbus,frbus,w,cf)
);
    input [7:0]a;
    input fbus,flbus,frbus;
    output reg [7:0]w;
    output reg cf;
    always@(fbus,flbus,frbus,a)
    begin
        cf=1'b0;
        if(fbus==1'b1) w[7:0]=a[7:0];
        else if(frbus==1'b1) begin w[7:0]={a[0],a[7:1]};cf=a[0]; end
        else if(flbus==1'b1) begin w[7:0]={a[6:0],a[7]};cf=a[7]; end
        else w[7:0]=8'hZZ;
    end
endmodule

```

功能仿真验证：



在 fbus, flbus, frbus 控制下, 数据左移, 右移, 直接通过, 输出高阻态均正确。移位逻辑可以正常工作。

(11) 状态寄存器 PSW

端口设计：

cf_en, zf_en 为使能信号, 由控制信号发生器给出; clk 为时钟信号; zf 接入为运算单元 ALU 的 zf 输出, cf 接入为运算单元 ALU 的 cf 与移位逻辑的 cf 输出相或。c, z 两个寄存器保存 cf, zf 的状态并输出给控制信号发生器。

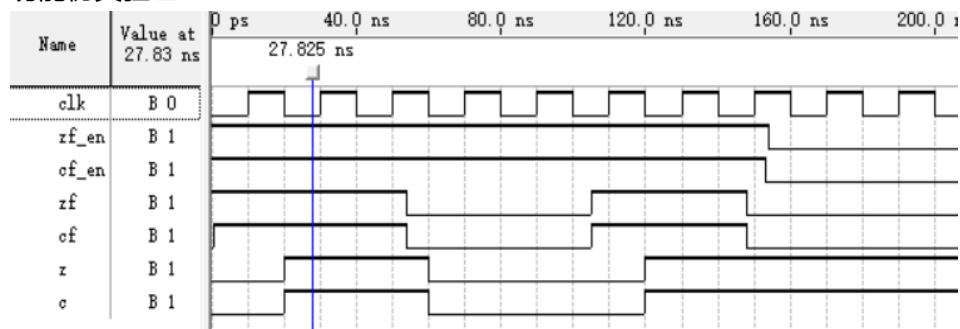


功能实现：

在 cf_en 和 zf_en 控制下, 在时钟下降沿将 cf, zf 写入 c, z 寄存器。

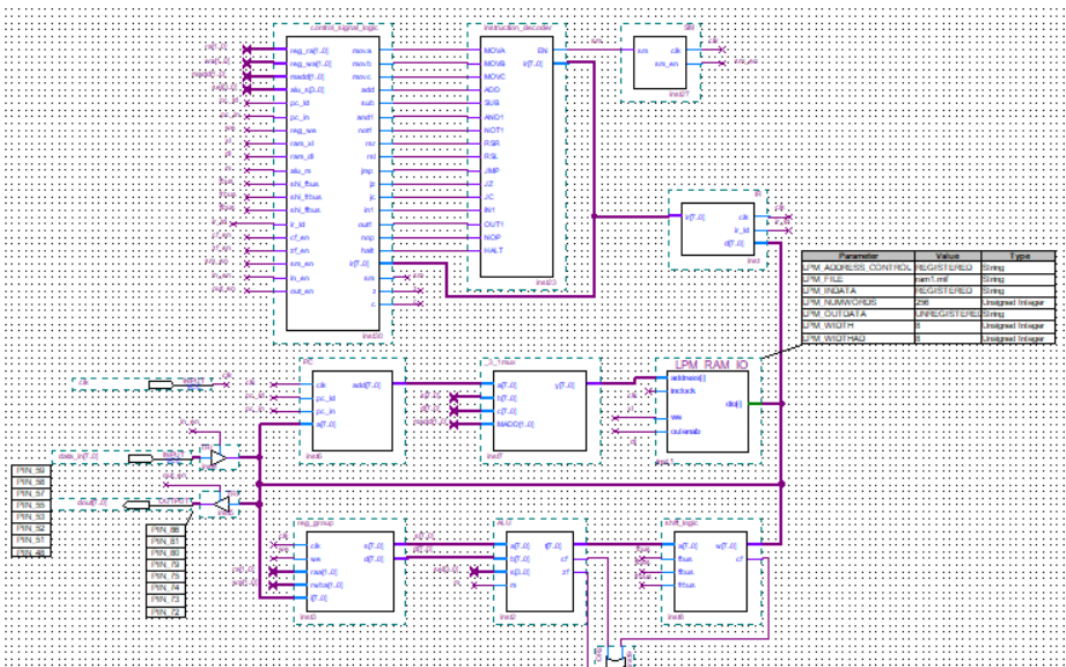
```
module PSW(clk,cf_en,zf_en,c,z,cf,zf);
    input clk,cf_en,zf_en,cf,zf;
    output c,z;
    reg c,z;
    always@(negedge clk)
    begin
        if(cf_en) c<=cf;
        if(zf_en) z<=zf;
    end
endmodule
```

功能仿真验证：



cf 和 zf 可以在使能信号有效的情况下正常写入寄存器组。状态寄存器可以正常工作。

3.3 模块整合及连接



四.系统测试

4.1 测试环境

在 Quarters 中，选用 Cyclone II EP2C5T144C8 进行测试。

4.2 测试代码

寄存器初值：

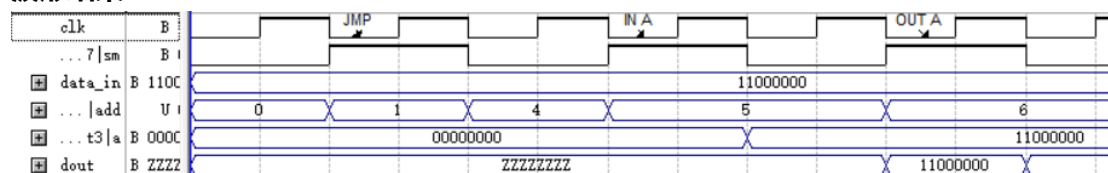
C:01100000

使用如下指令进行测试，共 16 个指令，加上 JZ，JC 执行不成功共 20 种情况，增加两次输出便于观察结果，共执行 20 个指令，停机后的 OUT 指令不执行。

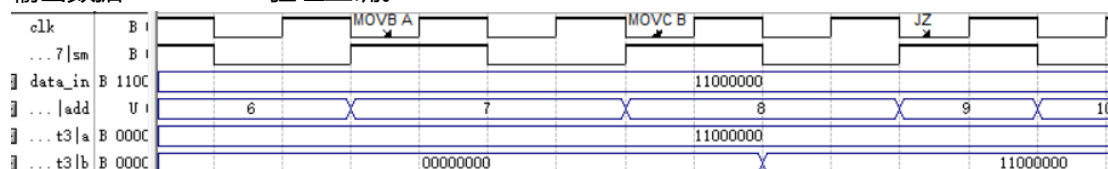
顺序	地址单元	指令	指令编码	执行情况	输出
1	0	JMP	00110000 00000100	跳转至地址单元4	
2	4	IN A	00100000	A=INPUT=11000000	
3	5	OUT A	01000000		OUTPUT=A=11000000
4	6	MOVB A	11001100	将A中数据移至地址C01100000	
5	7	MOVC B	11000111	地址单元C01100000移至B， B=11000000	
6	8	JZ	00110001 00010000	跳转不成功	
7	10	SUB A B	01100001	A=00000000， Z=1	
8	11	JZ	00110001 00010000	跳转至地址单元16	
9	16	NOT A	01010000	A=/A=11111111	
10	17	AND A C	10110010	A=A&C=01100000	
11	18	JC	00110010 01000000	跳转不成功	
12	20	ADD A B	10010001	A=A+B=001000000， C=1	
13	21	JC	00110010 01000000	跳转至地址单元64	
14	64	NOP	01110000	空操作	
15	65	RSR A	10100000	A右移， A=00010000	
16	66	OUT A	01000000		OUTPUT=A=00010000
17	67	RSL B	10100111	B左移， B=10000001	
18	68	MOVA A B	11000001	A=B=10000001	
19	69	OUT A	01000000		OUTPUT=A=10000001
20	70	HALT	10000000	停机	
21	71	OUT C	01001000	不执行	

4.3 测试结果

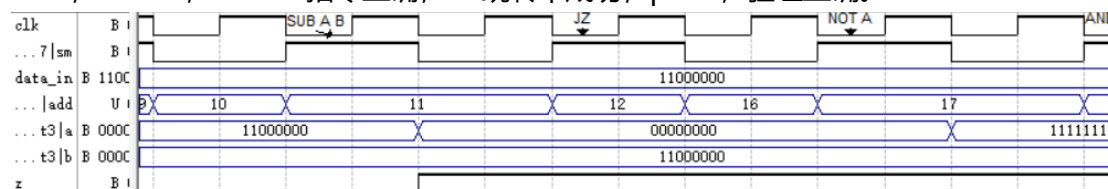
波形结果:



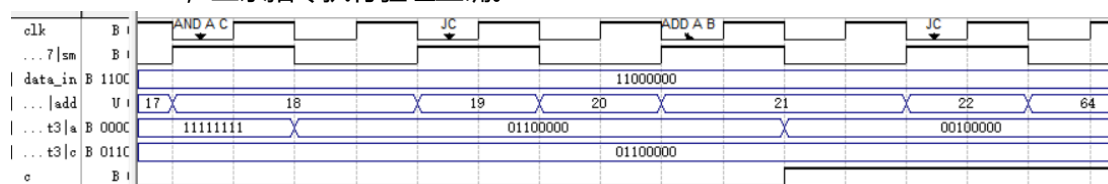
JMP 跳转后计数器中指令地址变为 4, 验证正确。向 A 寄存器 IN 指令写入 OUT 指令输出数据 11000000 验证正确。



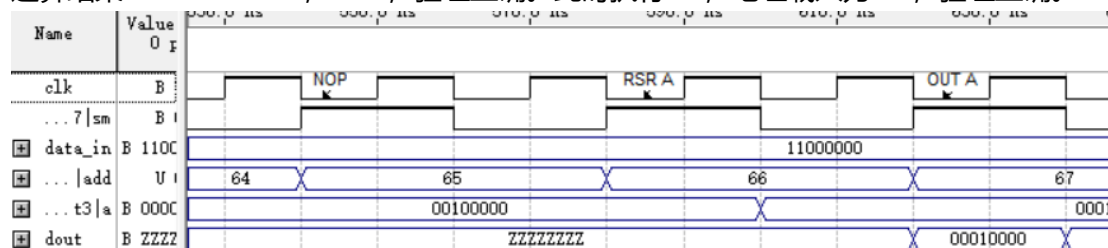
将寄存器 A 中数据写入存储器地址 01100000(寄存器 C 的初值), 再写给寄存器 B, A=B, MOV B, MOV C 指令正确, JZ 跳转不成功, pc+1, 验证正确。



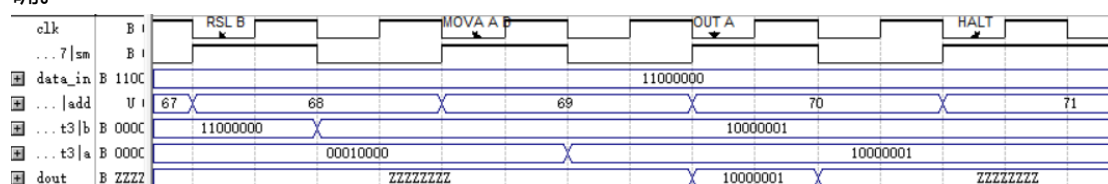
执行 SUB 指令, $A-B=0$, $Z=1$, JZ 跳转成功, 地址跳转至 16, 执行 NOT 指令, $A=11111111$, 三条指令执行验证正确。



执行 AND A C, $A=A \& C=01100000$, AND 指令验证正确, JC 跳转不成功, PC+1。执行 ADD A B, $A=A+B=01100000+11000000$ (转移指令后, B 为 IN 指令时输入的 A) 运算结果 $A=00100000$, $C=1$, 验证正确。此时执行 JC, 地址载入为 64, 验证正确。



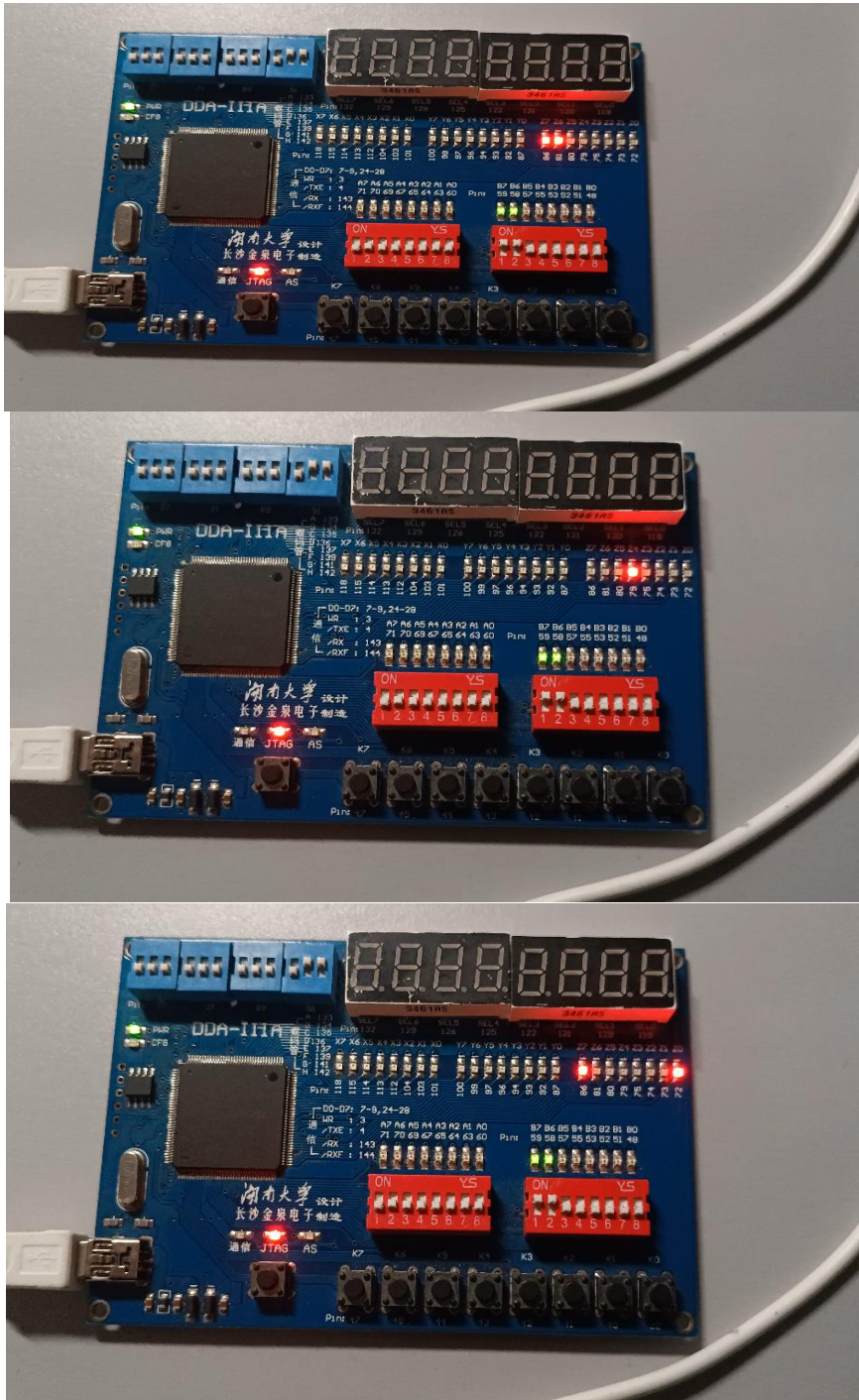
跳转后执行空指令后, RSR 指令将 A 中数据右移一位, $A=00010000$, 输出, 验证正确。



RSL 指令将 B 中数据左移一位, $B=10000001$, 验证正确。MOVA A B 将 B 中数据移给 A 并输出, 验证正确。HALT 停机指令后, $sm=1$, 不再翻转, 验证正确。

下载验证:

共有三次输出寄存器 A 的数据, 第一次输出为 11000000, 第二次输出为 00010000, 第三次输出为 10000001。将时钟频率调低, 下载进行验证, 验证正确。

**4.4 模型机性能分析**

时序分析后性能如下:

Timing Analyzer Summary						
	Type	Slack	Required Time	Actual Time	From	To
1	Worst-case tsu	N/A	None	9.695 ns	data_in[0]	lR:inst3[ir[0]
2	Worst-case tco	N/A	None	20.259 ns	reg_group:inst3[a[1]	dout[0]
3	Worst-case tpd	N/A	None	15.115 ns	data_in[0]	dout[0]
4	Worst-case th	N/A	None	-6.843 ns	data_in[3]	reg_group:inst3[c
5	Clock Setup: 'clk'	N/A	None	36.27 MHz (period = 27.572 ns)	reg_group:inst3[a[1]	lpm_ram_io:inst1[
6	Total number of failed paths					

最大频率为 36.27MHz，周期为 27.572ns。

资源消耗如下：

Family	Cyclone II
Device	EP2C5T144C8
Timing Models	Final
Met timing requirements	Yes
Total logic elements	201 / 4,608 (4 %)
Total combinational functions	201 / 4,608 (4 %)
Dedicated logic registers	43 / 4,608 (< 1 %)
Total registers	43
Total pins	19 / 89 (21 %)
Total virtual pins	0
Total memory bits	2,048 / 119,808 (2 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)

共使用逻辑单元 201 个。

五.实验总结、必得体会及建议

- 1.通过实验，了解与熟悉了模型机的功能，内部结构，数据通路及工作原理。
- 2.了解了模型机的指令计数器，选择器，存储器，指令寄存器等各个模块的工作原理和端口设计及代码实现的方式。通过 RTL 视图了解了模块的电路组成，通过时序分析进一步认识了电路的延迟问题。
- 3.了解了模块端口对接设计，总线数据传输，高阻态避免总线冲突，三态门的使用等电路中数据传输的方式方法。
- 4.通过各个模块设计，熟悉了 VERILOG 语言的使用和 BDF 电路原理图连接的方式对电路的设计过程以及后续的功能仿真验证，时序仿真验证的方法，资源和性能分析方法等。
- 5.对组合电路与时序电路的认识进一步加深，并通过设计过程深入认识了电子与电路学课程中所学习的编码，译码器，寄存器等组合逻辑电路和时序逻辑电路的知识和设计方法。
- 6.通过引脚分配和下载验证了解了开发板的使用方式。
- 7.通过在实验中不断学习，对控制信号的分析过程，最后的整合验证与测试，体验了模块设计，整体设计，功能验证的完整设计过程。