模型机设计报告

班级 计科 1801 姓名 李泳昊 学号 201808010105

一、设计目的

完整、连贯地运用《数字逻辑》所学到的知识，熟练掌握 EDA 工具基本使用方法，为学习好后续《计算机原理》课程做铺垫。

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 要求灵活运用各方面知识，使得所设计的计算机具有较佳的性能；

③ 对所设计计算机的性能指标进行分析，整理出设计报告。

三、详细设计

3.1 设计的整体架构

1．指令计数器 PC

2．选择器

3．RAM 存储器

4．指令寄存器 IR

5．指令译码器

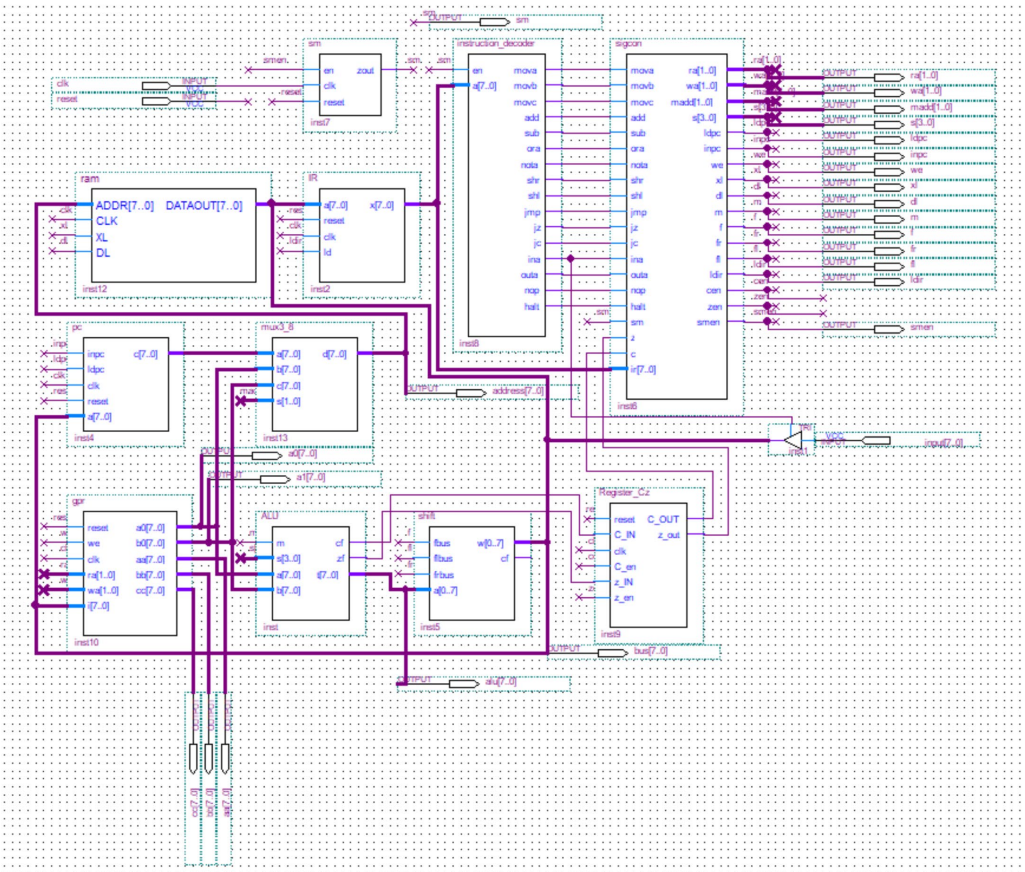
6．控制台，用于发出控制信号

7．通用寄存器组

8．函数发生器（ALU 运算器）

9．移位逻辑

顶层视图：



3.2 各模块的具体实现

（此部分必须有模块的接口设计，功能实现，功能的仿真验证等内容。）

1、指令计数器 PC

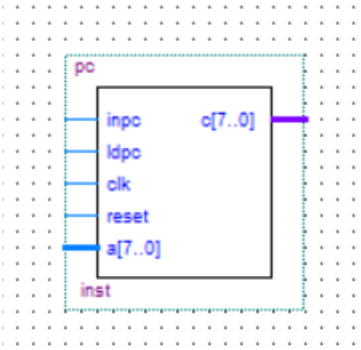
接口设计：

（1）inc,ld 两个信号输入，代表控制计数器自加一和数据输出以及读入的信号。

（2）clk 时钟信号，接受外部时序控制执行操作。

（3）a[7..0]和 c[8..0]，前者代表数据读入，后者代表输出。

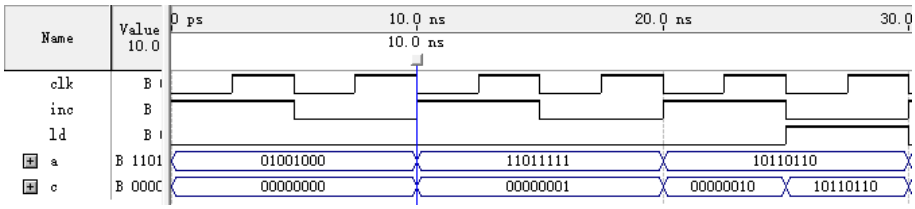
原理图



VHDL 代码：

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   entity pc is
5   port(
6        inpc,ldpc,clk,reset: in std_logic;
7        a: in std_logic_vector(7 downto 0);
8        c: out std_logic_vector(7 downto 0));
9   end pc;
10
11  architecture SSS of pc is
12  signal t,s: std_logic_vector(7 downto 0);
13  begin
14      process(clk,reset)
15      begin
16          if(reset='0')then
17          s<="00000000";
18          elsif(clk' event and clk='0')and(inpc='1')and(ldpc='0') then
19          s<=s+"00000001";
20          elsif(clk' event and clk='0')and(inpc='0')and(ldpc='1') then
21          s(7 downto 0)<=a(7 downto 0);
22          end if;
23          c<=s;
24      end process;
25  end SSS;
```

仿真波形：

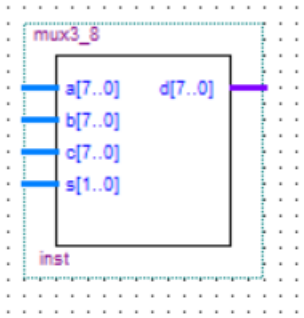


功能分析：

CLK	inc	ld	功能
⎍	1	0	c<=c+1（不加载输入，输出自加一）
⎍	0	1	c<=a（加载输入）

2、选择器

原理图



接口设计：

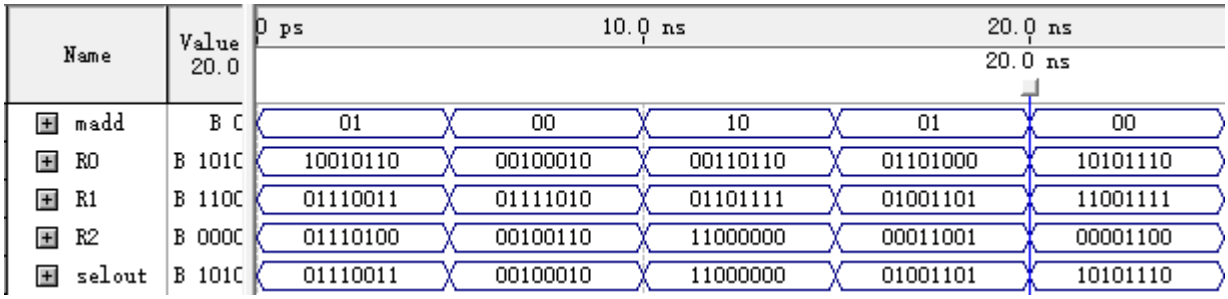
（1）S 是 2 位控制信号，用于选择地址来源。

（2）a，b，c 是三个地址来源，对应 PC、寄存器 A 口和寄存器 B 口。

VHDL 代码：

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity mux3_8 is
5   port (a,b,c:in std_logic_vector(7 downto 0);
6   d:out std_logic_vector(7 downto 0);
7   s:in std_logic_vector(1 downto 0));
8   end mux3_8;
9
10  architecture SSS of mux3_8 is
11  begin
12  with s(1 downto 0) select
13      d<= a(7 downto 0) when "00",
14          b(7 downto 0) when "01",
15          c(7 downto 0) when "10",
16          "XXXXXXXX" when others;
17  end SSS;
```

仿真波形：



功能分析：

madd	功能
00	selout<=R0(选择R0输出)
01	selout<=R1(选择R1输出)
10	selout<=R2(选择R2输出)

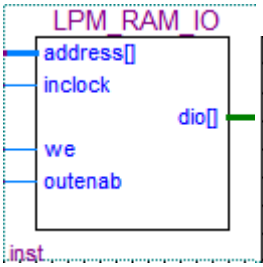
3、RAM 存储器

接口设计：

（1）使用 address 输入和 dio 输出，都是 8 位，输入为地址，输出为指令，指令存储在 RAM

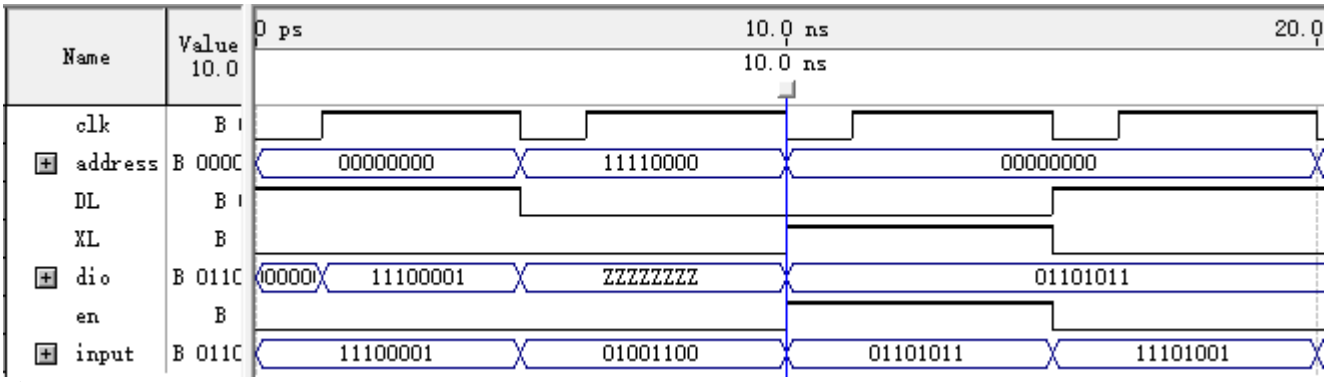
中。
（2）有 we,outenab 控制信号，代表读写有效。
（3）有时钟控制信号，对 RAM 的读取写入进行流程控制。
功能分析
（1）RAM 是存储各种指令的地方，可以进行指令的读写。
（2）构造 RAM 中的指令，需要在新建文件里面建立一个 mif 文件，设定好参数（256 个，8 位）根据指令表构造指令存入即可。
（3）RAM 也有时钟信号，配合其他元件，用于控制 RAM 何时进行读写操作。

原理图：



Parameter	Value	Type
LPM_ADDRESS_CONTROL	REGISTERED	String
LPM_FILE	./RAM.mif	String
LPM_INDATA	REGISTERED	String
LPM_NUMWORDS	256	Unsigned Integer
LPM_OUTDATA	UNREGISTERED	String
LPM_WIDTH	8	Unsigned Integer
LPM_WIDTHAD	8	Unsigned Integer

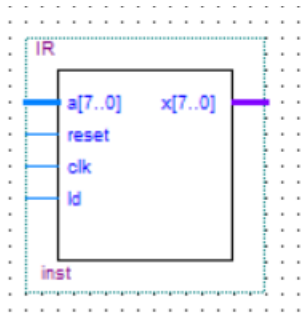
VHDL 代码：
调用了 LPM 库，略
仿真波形：（利用了三态门）



功能分析：

CLK	we	outenab	功能
⎍	0	0	Dio<=高阻态Z
⎍	1	0	Dio的数据写入address所指定的存储单元
⎍	0	1	address所指定的存储单元数据从dio输出

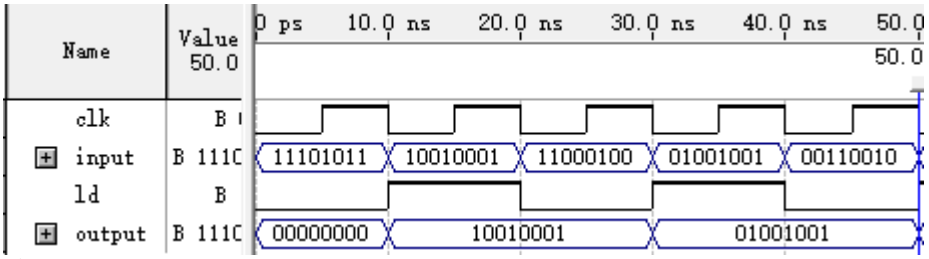
4、指令寄存器（IR）
接口设计：
（1）input 是 8 位的输入,output 是 8 位的输出。
（2）ld 代表将 BUS 总线上的数据传输入译码器，1 有效。
（3）clk 时钟信号接口，这个部件也需要时序控制。
原理图：



VHDL 代码：

```
library ieee;
use ieee.std_logic_1164.all;
--下降沿触发的八位数据寄存器
entity IR is
port(a:in std_logic_vector(7 downto 0);--input
     reset,clk,ld:in std_logic;--clock pulse, enable(1 is effective)
     x:out std_logic_vector(7 downto 0));--output
end IR;

architecture SSS of IR is
signal temp:std_logic_vector(7 downto 0):="00000000";
begin
    process(a,clk,ld,reset)
    begin
    if(reset='0')then
    temp<="00000000";
    else
        if (ld='1') then
            if(clk'event and clk='0') then
            temp<=a;
            end if;
        end if;
    end if;
    end process;
    x<=temp;
end SSS;
```

仿真波形：



功能分析：

clk	ld	功能
⌐	1	output<=input

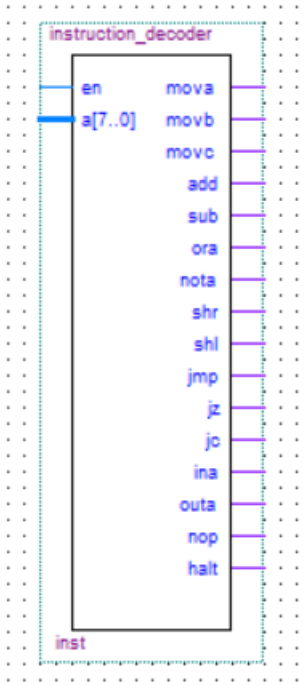
5、指令译码器

接口设计：

（1）输入的 IR 是 8 位的指令，SM 为时钟信号。

（2）输出对应的是各个指令所需要的操作，传输入控制台具体执行

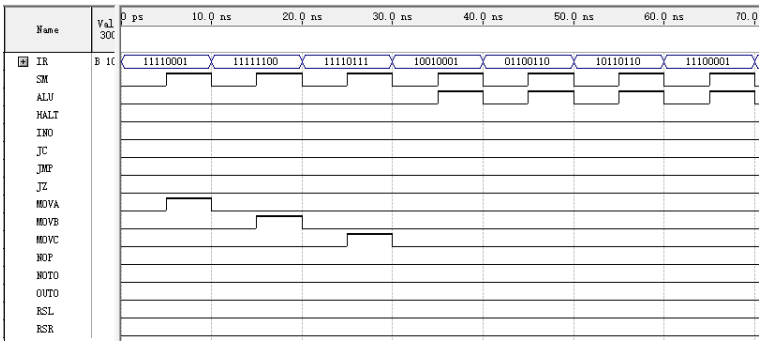
原理图：

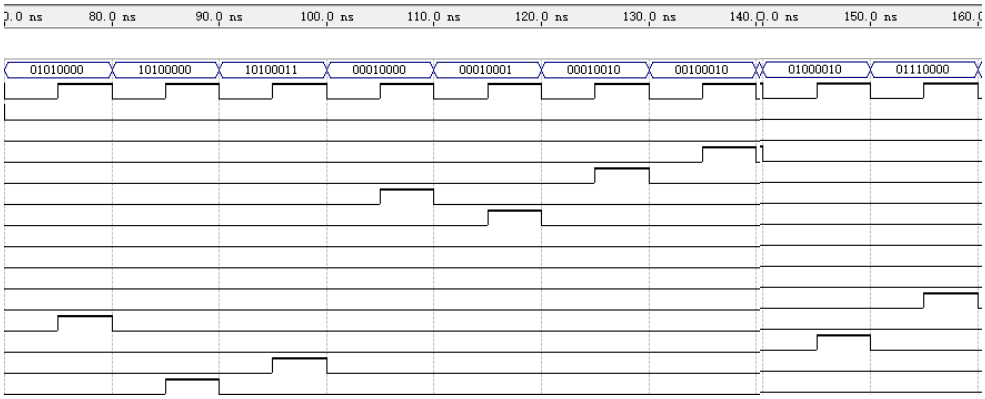


VHDL 代码：

```
library ieee;
use ieee.std_logic_1164.all;

entity instruction_decoder is
port(en : in bit;
 a: in std_logic_vector(7 downto 0);
 mova,movb,movc,add,sub,ora,nota,shr,shl,jmp,jz,jc,ina,outa,nop,halt: out bit);
 end instruction_decoder;

architecture SSS of instruction_decoder is
begin
process(a,en)
 begin
 mova<='0';movb<='0';movc<='0';add<='0';sub<='0';ora<='0';nota<='0';shr<='0';shl<='0';jmp<='0';jz<='0';jc<='0';ina<='0';outa<='0';nop<='0';halt<='0';
if (en='1') then
    if(a(7 downto 2)="111111") then movb<= '1';
    elsif (a(7 downto 4)="1111" and a(1 downto 0)="11") then movc<= '1';
    elsif (a(7 downto 4)="1111") then mova<= '1';
    elsif (a(7 downto 4)="1001") then add<= '1';
    elsif (a(7 downto 4)="0110") then sub<= '1';
    elsif (a(7 downto 4)="1011") then ora<= '1';
    elsif (a(7 downto 4)="0101") then nota<= '1';
    elsif (a(7 downto 4)="1010" and a(1 downto 0)="00") then shr<= '1';
    elsif (a(7 downto 4)="1010" and a(1 downto 0)="11") then shl<= '1';
    elsif (a(7 downto 0)="00010000") then jmp<= '1';
    elsif (a(7 downto 0)="00010001") then jz<= '1';
    elsif (a(7 downto 0)="00010010") then jc<= '1';
    elsif (a(7 downto 4)="0010") then ina<= '1';
    elsif (a(7 downto 4)="0100") then outa<= '1';
    elsif (a(7 downto 0)="01110000") then nop<= '1';
    elsif (a(7 downto 0)="10000000") then halt<= '1';
    end if;
 end if;
 end process;
 end SSS;
```

波形仿真：





功能分析：

指令译码器的功能就是对于输入进的各种指令进行读取分析，输出一系列的控制信号。这里没有直接将控制信号输出，因为这样会使得代码繁琐、大量重复，考虑简洁性，使用了控制器（见下一个部件）。这里我加了个 AND 的指令，操作码设置为 1110。

表 1　　指 令 系 统 表

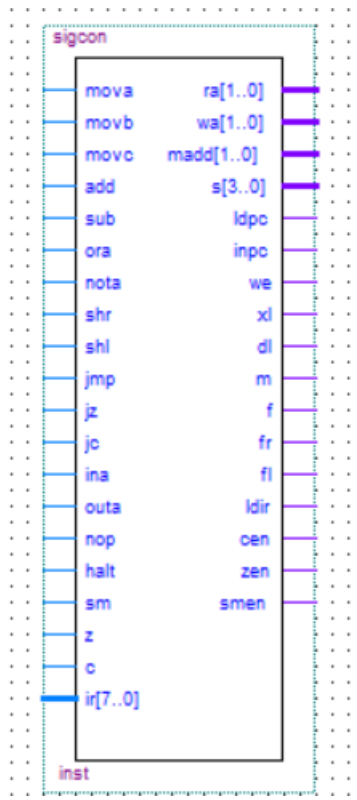
汇编符号	功能	编码
MOV　R1，R2	（R2）→ R1	1111　R1 R2
MOV　M，R2	（R2）→（C）	1111　11 R2
MOV　R1，M	((C)) →R1	1111　R1 11
ADD　R1，R2	（R1）+（R2）→ R1	1001　R1 R2
SUB　R1，R2	（R1）-（R2）→ R1	0110　R1 R2
OR　R1，R2	（R1）∨（R2）→ R1	1011　R1 R2
NOT　R1	/（R1）→ R1	0101　R1 XX
RSR　R1	（R1）循环右移一位→ R1	1010　R1 00
RSL　R1	（R1）循环左移一位→ R1	1010　R1 11
JMP　add	add → PC	0001　00 00，address
JZ　add	结果为 0 时 add → PC	0001　00 01，address
JC　add	结果有进位时 add → PC	0001　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

6、控制器：

接口设计：

包括输入的信号，以及所有指令译码器传过来的指令信号，由控制器具体执行发出信号。

原理图：

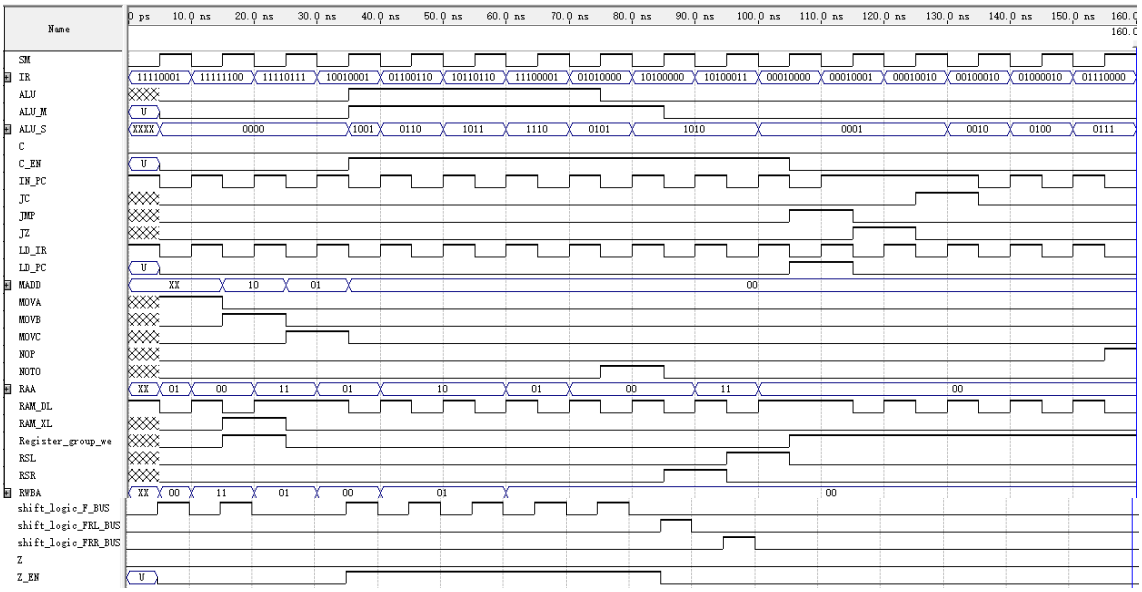


VHDL 代码：

```
library ieee;
use ieee.std_logic_1164.all;

entity sigcon is
    port(mova,movb,movc,add,sub,ora,nota,shr,shl,jmp,jz,jc,ina,outa,nop,halt,sm,z,c: in std_logic;
    ir: in std_logic_vector(7 downto 0);
    ra,wa,madd : out std_logic_vector(1 downto 0);
    s: out std_logic_vector(3 downto 0);
    ldpc,inpc,we,xl,dl,m,f,fr,fl,ldir,cen,zen,smen: out std_logic);
end sigcon;

architecture lyh of sigcon is
begin
process(IR,MOVA,MOVB,MOVC,add,sub,ora,nota,NOTa,shR,shL,JMP,JZ,Z,JC,C,NOP,SM,ina,outa)
begin

    we<=not(sm and(MOVA or MOVC or shR or shl or add or sub or ora or nota or ina) );


    wa<=IR(3 downto 2);
    RA<=IR(1 downto 0);


    INPC<=not sm;

    LDPC<=(sm and (jmp or (jc and c) or (jz and z)));

    LDIR<=not SM;

    DL<=(not sm) or (jmp or (jc and c) or (jz and z) or movc);
```

```
31
32      XL<=sm and movb;
33
34      M<=add or sub or ora or nota;
35
36      F<=(add or sub or ora or nota or MOVA or MOVB or outa);
37
38      fl<=shR and SM;
39
40      fr<=shL and SM;
41
42      CEN<=add or sub or ora or nota;
43
44      ZEN<=add or sub or ora or nota;
45
46      S<=IR(7 DOWNTO 4);
47
48      if(sm='0')then madd<="00";
49          elsif(movb='1')then madd<="10";
50          elsif(movc='1')then madd<="01";
51          elsif(jmp='1' or (jc='1' and c='1') or (jz='1' and z='1'))then madd<="00";
52          else madd<="XX";
53          end if;
54
55      smen<=not halt;
56      end process;
57  end lyh;
```

波形仿真：



功能分析：

将 M 信号改成了 M=1 时在 ALU 中进行操作，M=0 时选择输入进行直传。

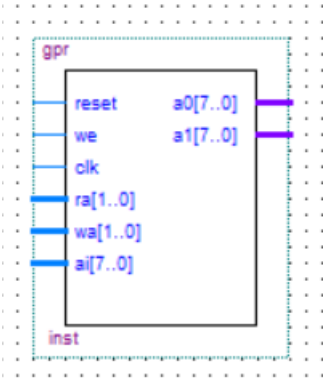
表 2　基本控制信号及功能表

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。
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。

7、通用寄存器组：

接口设计：

（1）i 是 8 位的输入,AO,BO,AR,BR,CR 是 8 位的输出。(AR、BR、CR 输出的分别是 A 寄存器、B 寄存器和 C 寄存器的值，AO、BO 输出的分别根据 RA 和 WA 的值决定）
（2）we 代表将 BUS 总线上的数据传输入寄存器组，0 有效。
（3）clk 时钟信号接口，这个部件也需要时序控制。
原理图：

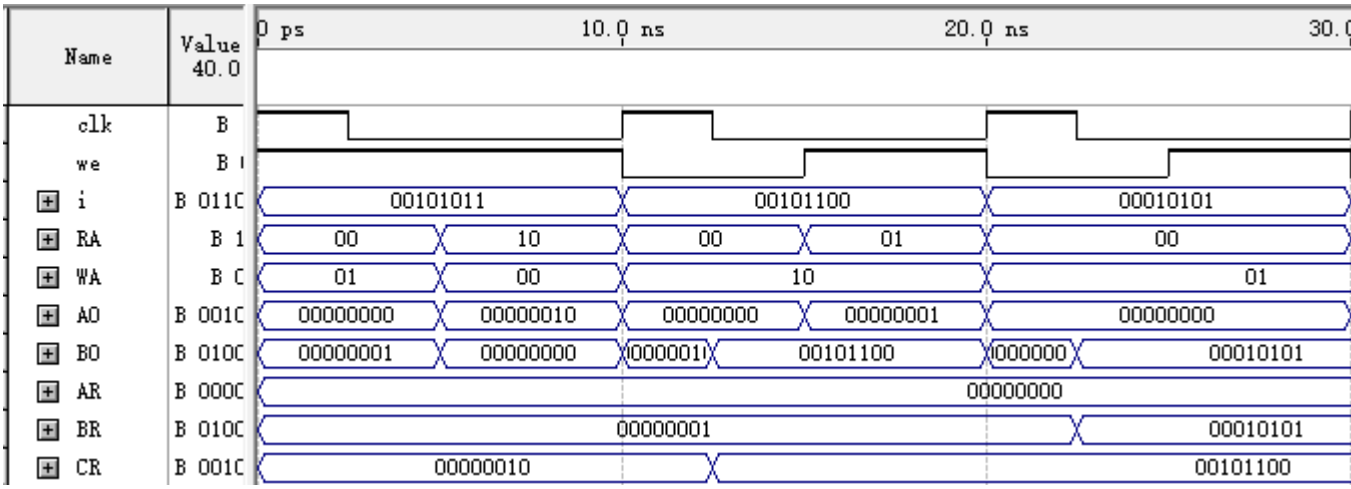


VHDL 代码：

```
 1   library ieee;
 2   use ieee.std_logic_1164.all;
 3  entity gpr is
 4  port(
 5       reset,we,clk: in std_logic;
 6       ra,wa: in std_logic_vector(1 downto 0);
 7       i: in std_logic_vector(7 downto 0);
 8       a0,b0,aa,bb,cc: out std_logic_vector(7 downto 0));
 9   end gpr;
10
11  architecture lyh of gpr is
12       signal a: std_logic_vector(7 downto 0):="10000001";
13       signal b: std_logic_vector(7 downto 0):="10000001";
14       signal c: std_logic_vector(7 downto 0):="00000000";
15  begin
16       process(clk,we,reset,ra,wa)
17       begin
18       if(reset='0')then
19       a<="10000001";
20       b<="10000001";
21       c<="00000000";
22       else
23               if(we='0')then
24                       if(clk'event and clk='0')then
25                           if(wa="00")then a<=i;
26                           elsif(wa="01")then b<=i;
27                           elsif(wa="10")then c<=i;
28                           else a<=a;b<=b;c<=c;
29                           end if;
30                       end if;
```

```
31                  end if;
32
33                  if(ra="00")then a0<=a;
34                      elsif(ra="01")then a0<=b;
35                      elsif(ra="10")then a0<=c;
36                      else a0<=c;
37                  end if;
38                  if(wa="00")then b0<=a;
39                      elsif(wa="01")then b0<=b;
40                      elsif(wa="10")then b0<=c;
41                      else b0<=c;
42                  end if;
43
44                  aa<=a;
45                  bb<=b;
46                  cc<=c;
47      end if;
48      end process;
49  end lyh;
```

波形仿真：

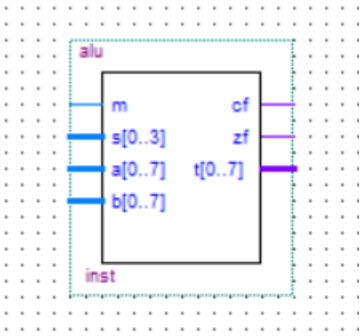


功能分析：

CLK	WE	RAA[1..0]	RWBA[1..0]	功能
	1	00或01或10	00或01或10	根据RAA[1..0]的值从A,B,C中选择一个寄存器的值由AO口输出 如RAA[1..0]=00，AO<=A寄存器的值 RAA[1..0]=01，AO<=B寄存器的值 RAA[1..0]=10，AO<=C寄存器的值 根据RWBA[1..0]的值从选择A,B,C中选择一个寄存器的值由BO口输出， 如RWBA[1..0]=00，BO<=A寄存器的值 RWBA[1..0]=01，BO<=B寄存器的值 RWBA[1..0]=10，BO<=C寄存器的值
┐	0	XX	00或01或10	根据RWBA[1..0]的值,将外部输入写入A,B,C三个寄存器中的一个寄存器内。

8、函数发生器（ALU）

接口设计：
（1）使能端信号 en，为 0 直传，为 1 计算。
（2）4 位控制信号 S，代表 ALU 执行的各种操作。
（3）A,B,output 三个 8 位接口，代表通用寄存器 AO、BO 口传来的数据和 ALU 的运算结果。
（4）C 输出代表进位，Z 输出代表运算结果。结果为 00000000 时输出 1，否则为 0，在跳转指令中起作用。
原理图：

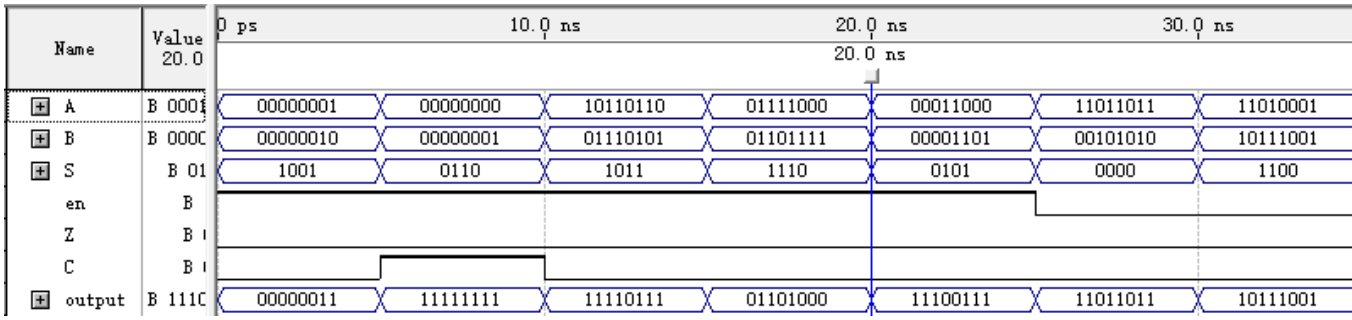


VHDL 代码：

```
library ieee;
use ieee.std_logic_1164.all;

entity ALU is
    port(m: in std_logic;
        s: in std_logic_vector(3 downto 0);
        a,b: in std_logic_vector(7 downto 0);
        cf,zf: out std_logic;
        t: out std_logic_vector(7 downto 0));
end ALU;

architecture re of ALU is

    component ebaad is
        port(a,b: in std_logic_vector(7 downto 0);
            c: out std_logic_vector(7 downto 0);
            jcc: out std_logic);
    end component;

    component ebsub is
        port(a,b: in std_logic_vector(7 downto 0);
            c: out std_logic_vector(7 downto 0);
            jzz: out std_logic);
    end component;

    component ebor is
        port(a,b: in std_logic_vector(7 downto 0);
            c: out std_logic_vector(7 downto 0));
    end component;

```

```
    component ebnot is
        port(a: in std_logic_vector(7 downto 0);
        c: out std_logic_vector(7 downto 0));
    end component;

    signal t1,t2,t3,t4: std_logic_vector(7 downto 0);
    signal tcf,tzf: std_logic;
    begin
    g0:ebaad port map(a,b,t1,tcf);
    g1:ebsub port map(a,b,t2,tzf);
    g2:ebor port map(a,b,t3);
    g3:ebnot port map(b,t4);
    process(m,s,a,b)
        begin
            if(m='0')then
                if(s="1111")then t<=a;
                elsif(s="1010" or s="0100")then t<=b;
                end if;
            elsif(s="1001")then
                t<=t1;cf<=tcf;zf<='0';
            elsif(s="0110")then
                t<=t2;cf<='0';zf<=tzf;
            elsif(s="1011")then
                t<=t3;cf<='0';zf<='0';
            elsif(s="0101")then
                t<=t4;cf<='0';zf<='0';
            else t<="ZZZZZZZZ";
            end if;
    end process;

end;
```

波形仿真：



功能分析：

ALU 的功能主要是实现加、减、或等运算，由 S 控制，另外 ALU 作为数据通路，在特定的指令下，也执行直传操作，直传的数据来源可能有两个，AO 口和 BO 口。这里的 ALU 我定义了两个新的控制信号 S，是“1100”和“0000”。这两个指令，在指令表中没有定义，这里额外定义不会冲突。前者表示直传 B，后者表示直传 A。

S	en	功能
1001	1	output<=A+B(ADD)
0110	1	output<=A-B(SUB)
1011	1	output<=A∨B(OR)
1110	1	output<=A∧B(AND)
0101	1	output<=/A(NOT)
0000	0	output<=A(直传A)
1100	0	output<=B(直传B)

9、移位逻辑

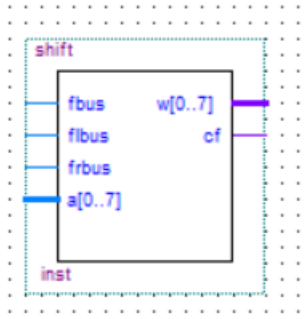
接口设计：

（1）INPUT 和 OUTPUT 是 8 位的输出和输出。

（2）三个控制信号：FBUS 为 1 则直传，FR 为 1 右移，FL 为 1 左移。

（3）C 代表被移除的那一位，即左移时输出原最左一位，右移时输出原最右一位。

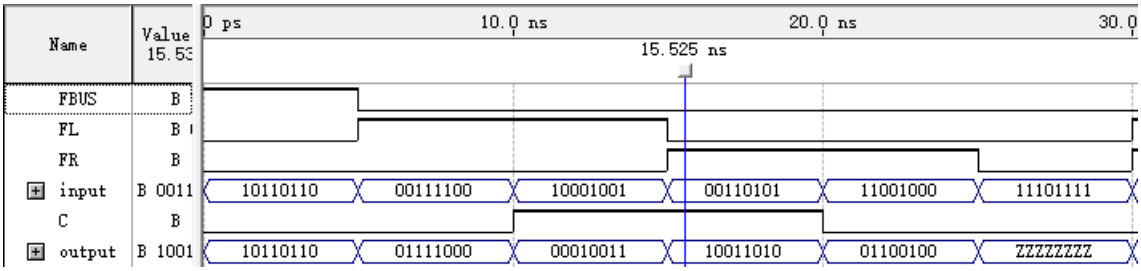
原理图：



VHDL 代码：

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity shift is
4       port(fbus,flbus,frbus: in std_logic;
5       a: in std_logic_vector(0 to 7);
6       w: out std_logic_vector(0 to 7);
7       cf: out std_logic);
8   end shift;
9
10  architecture SSS of shift is
11  begin
12  process(flbus,frbus,fbus)
13  begin
14      if(fbus='1')then
15              cf<='0';
16              w<=a;
17      elsif(frbus='1')then
18          cf<=a(0);
19          FOR n in 6 downto 0 LOOP
20              w(n)<= a(n+1);
21          END LOOP;
22          w(7)<=a(0);
23      elsif(flbus='1')then
24          cf<=a(7);
25          FOR n IN 1 TO 7 LOOP
26              w(n)<= a(n-1);
27          END LOOP;
28          w(0)<=a(7);
29      else w<="ZZZZZZZZ";
30      end if;
31      end process;
32  end;
```

波形仿真：



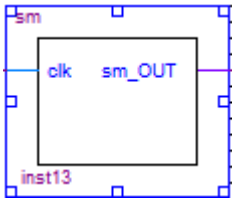
功能分析：

移位逻辑部件的功能类似于 ALU，也是两方面：一方面根据信号控制进行数据的处理，左移和右移，并将移除的位输出；另一方面作为数据通路，在特殊信号控制下进行直传。

F_BUS	FL	FR	功能
1	0	0	直传
0	1	0	循环左移一位
0	0	1	循环右移一位
0	0	0	输出高阻态

10、时钟信号控制

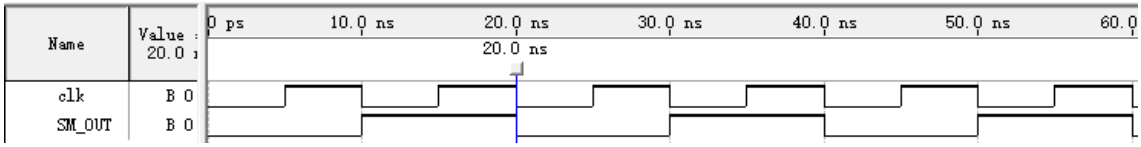
原理图：



VHDL 代码：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity sm is
port(clk: in std_logic;
     sm_OUT:out std_logic);
end sm;

architecture behavior of sm is
signal num: std_logic_vector(1 downto 0):="00";
begin
    process(clk)
    begin
    if(clk'event and clk='0') then
        num<=num+1;
    end if;
    end process;
    sm_OUT<=num(0);
end behavior;
```

波形仿真：



sm 是 clk 的二分频，sm=0 为取指令周期，sm=1 为执行指令周期。

四、 系统测试

4.1 测试环境

Quartus II 9.0

仿真类型：Functional

4.2 测试代码

（使用模型机实现的指令编写一至两个程序测试模型机的正确性。）

测试一：

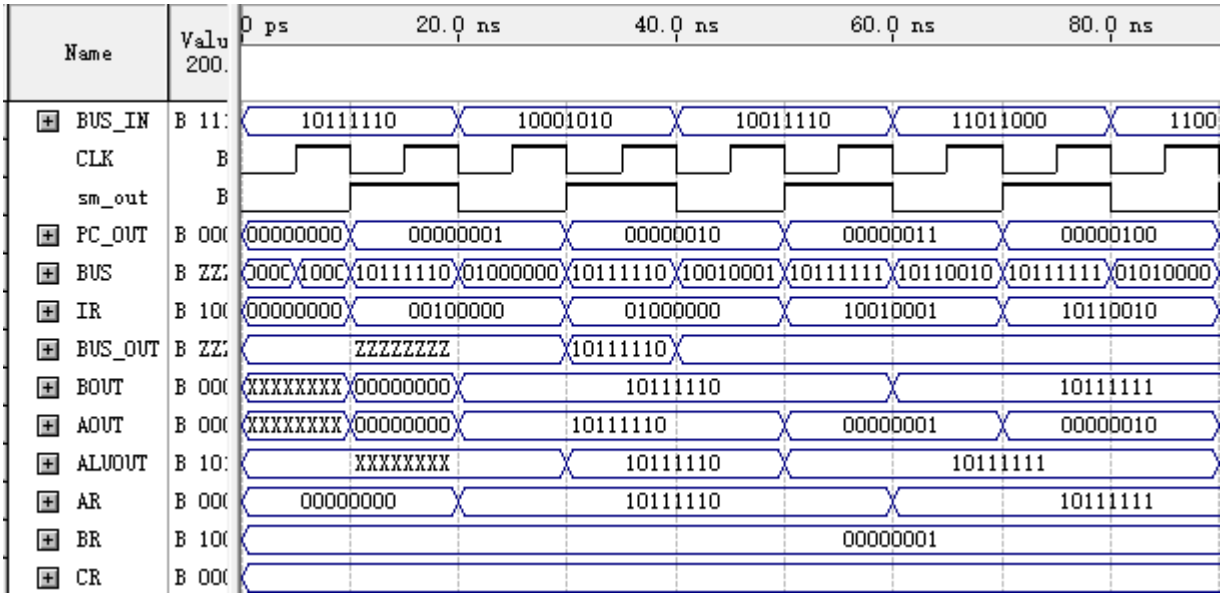
Addr	+000	+001	+010	+011	+100	+101	+110	+111
00000000	00100000	01000000	10010001	10110010	01010000	11110100	01100001	00010001
00001000	00011000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00010000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00011000	11110111	10000000	00000000	00000000	00000000	00000000	00000000	00000000

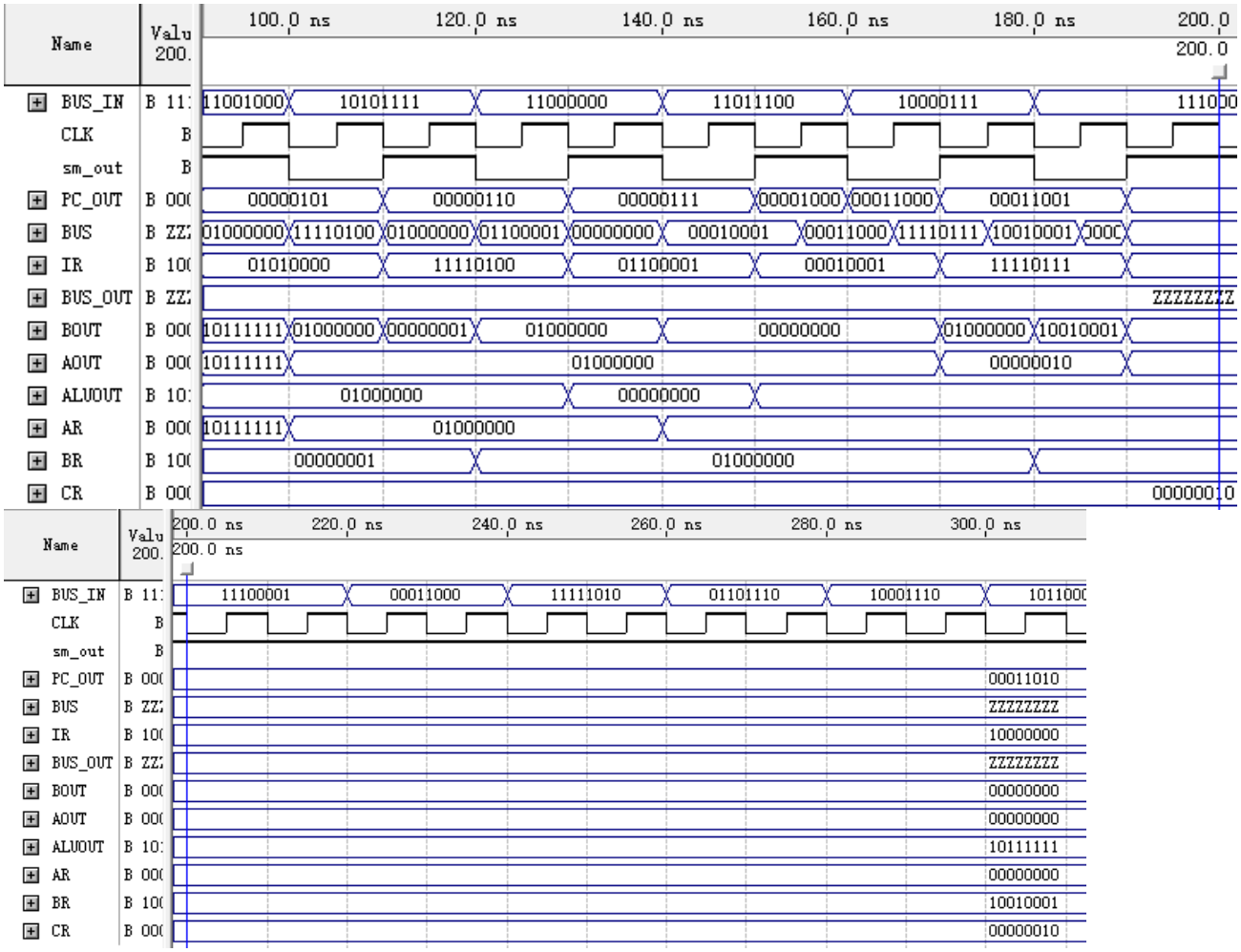
测试二：

Addr	+000	+001	+010	+011	+100	+101	+110	+111
00000000	01110000	10000000	10101011	10100100	00010010	00100100	00000000	00000000
00001000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00010000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00011000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00100000	00000000	00000000	00000000	00000000	11111100	00010000	00000100	00000000

4.3 测试结果

测试一：





（1）IN 指令，0.0-10.0ns 取指令，10.0-20.0ns 执行指令，指令编码为 00100000，将总线上的输入加载入寄存器 A，此时总线上的输入为 10111110，故执行完这个指令后寄存器 A 的存储的值由初始值 00000000 变为 10111110；

（2）OUT 指令，20.0-30.0ns 取指令，30.0-40.0ns 执行指令，指令编码为 01000000，将寄存器 A 中的值输出到发光二极管，由于没有线路板原件，故只能模拟，输出到总线为寄存器 A 中的值 10111110；

（3）ADD 指令，40.0-50.0ns 取指令，50.0-60.0ns 执行指令，指令编码为 10010001，将寄存器 A 中的值 10111110 和寄存器 B 中的值 00000001 相加后存入寄存器 A，故执行完这个指令后寄存器 A 中的值变为 10111111；

（4）OR 指令，60.0-70.0ns 取指令，70.0-80.0ns 执行指令，指令编码为 10110001，将寄存器 A 中的值 10111111 和寄存器 B 中的值 00000001 按位相或后存入寄存器 A，故执行完这个指令后寄存器 A 中的值仍为 10111111；

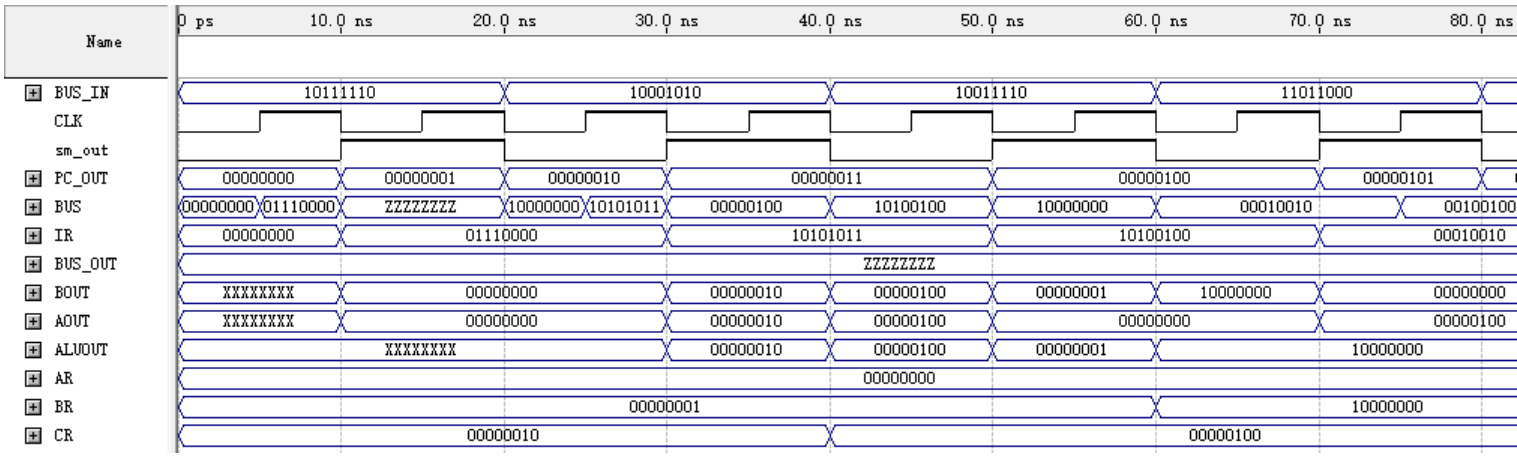
（5）NOT 指令，80.0-90.0ns 取指令，90.0-100.0ns 执行指令，指令编码为 01010000，将寄存器 A 中的值 10111111 按位取反后存入寄存器 A，故执行完这个指令后寄存器 A 中的值变为 01000000；

（6）MOVA 指令，100.0-110.0ns 取指令，110.0-120.0ns 执行指令，指令编码为 11110100，将寄存器 A 中的值存入寄存器 B，故执行完这个指令后寄存器 B 中的由 00000010 变为 01000000；

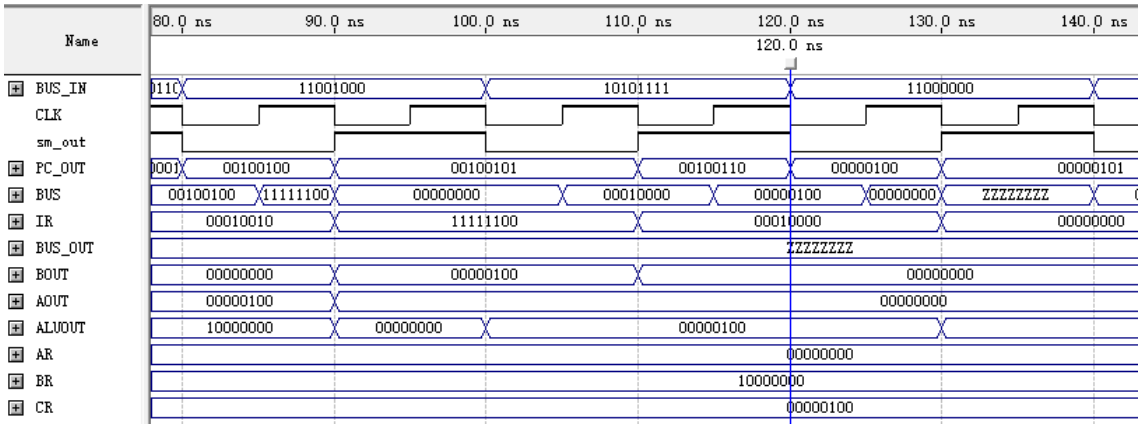
（7）SUB 指令，120.0-130.0ns 取指令，130.0-140.0ns 执行指令，指令编码为 01100001，将寄存器 A 中的值减去寄存器 B 中的值后得到的结果存入寄存器 A，故执行完这个指令后寄存器 A 的值变为 00000000，此时结果为 0，Z 标志位为 1；

（8）JZ 指令，140.0-150.0ns 取指令，150.0-160.0ns 执行指令，指令编码为 00010001，满足跳转条件，后面跟的 address 编码为 00011000，下一次取指令的地址将变为 00011000；

（9）MOVC 指令，160.0-170.0ns 取指令，170.0-180.0ns 执行指令，指令编码为 11110111，从寄存器 C 中读出地址 00000010，将 RAM 中该地址的值 10010001 存入寄存器 B 中，故执行完这个指令后寄存器 B 的值由 01000000 变为 10010001；
（10）HALT 指令，180.0-190.0ns 取指令，190.0-200.0ns 执行指令，指令编码为 10000000，后序指令都不将执读取和执行。
上述指令操作结果均符合预期。



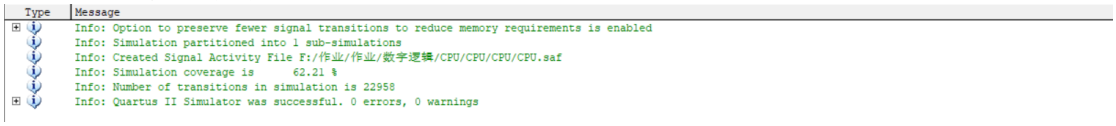
测试二：



（1）NOP 指令：0.0-10.0ns 取指令，10.0-20.0ns 执行指令，指令编码为 01110000，将 PC 的值加 1；
（2）RSL 指令：20.0-30.0ns 取指令，30.0-40.0ns 执行指令，指令编码为 10101011，将寄存器 C 中的值循环左移一位，故执行完这个指令后寄存器 C 中的值由 00000010 变为 00000100；
（3）RSR 指令：40.0-50.0ns 取指令，50.0-60.0ns 执行指令，指令编码为 10100100，将寄存器 B 中的值循环右移一位，故执行完这个指令后寄存器 B 中的值由 00000001 变为 10000000，且 C 标志位变为 1；
（4）JC 指令：60.0-70.0ns 取指令，70.0-80.0ns 执行指令，指令编码为 00010010，满足跳转条件，后面跟的 address 为 00100100，故下一次取指令的地址将变为 00100100；
（5）MOVB 指令：80.0-90.0 取指令，90.0-100.0 执行指令，指令编码为 11111100，读出寄存器 A 中的值 00000000 和寄存器 C 中的值 00000100，将寄存器 A 的值写入 RAM 中地址为寄存器 C 中的值的单元，由下一条 JUM 指令检测其正确性。
（6）JMP 指令：100.0-110.0 取指令，110.0-120.0 执行指令，指令编码为 00010000，后面跟的地址为 00000100，故下一次取指令的地址将变为 00000100；
可观测到下一次取出的指令为 00000000，而不是 mif 文件中 00000100 对应的 00010010 正是上次 MOVB 指令造成的结果。

上述指令操作结果均符合预期。

4.4 模型机性能分析



进行时序仿真，找到执行正确命令的**最短**时间周期为 26.2ns

五、 总结

（设计总结与心得）

在设计 CPU 时，要思路清晰，将每一个部件进行连接和仿真，并尝试了在 FPGA 板子上进行仿真，是一次不错的尝试。