模型机设计报告

班级 计科 1808 班 姓名 张继伟 学号 201808010829

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

任务

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

数据格式与指令系统

本设计的主要目的是希望学生巩固在《数字逻辑》课程中学到的理论知识，并加以灵活

运用。因此，要设计的计算机非常简单。这台机器具有寄存器直接寻址和寄存器间接寻址两种寻址方式，除跳转指令为双字节指令外，其它指令均为单字节指令，字长为 8 位。

1、 数据格式

数据字采用 8 位二进制定点补码表示，其中最高位（第 7 位）为符号位，小数点可视为最左或最右，其数值表示范围分别为：$-1 \leqslant X < +1$ 或$-128 \leqslant X < +127$。

2、 寻址方式

指令的高 4 位为操作码，低 4 位分别用 2 位表示目的寄存器和源寄存器的编号，或表示寻址方式。共有 2 种寻址方式。

⑴ 寄存器直接寻址

当 R1 和 R2 均不是“11”时，R1 和 R2 分别表示两个操作数所在寄存器的地址（寄存器编号），其中 R1 为目标寄存器地址，R2 为源寄存器地址。R1 或 R2 的值 指定的寄存器 00 A 寄存器 01 B 寄存器 10 C 寄存器

⑵ 寄存器间接寻址

当 R1 或 R2 中有一个为“11”时，表示相应操作数的地址在 C 寄存器中。

3、 指令系统

指令系统有 16 条指令，具体格式见指令系统表。应该指出的是，各

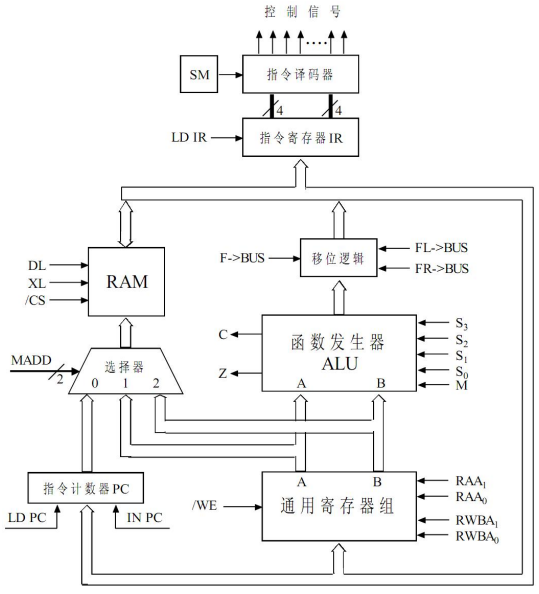
条指令的编码形式可以多种多样。为了叙述方便，下面采用汇编符号对指令进行描述，其中 R1 和 R2 分别表示“目标”和“源”寄存器，M 表示地址在寄存器 C 中的存贮单元。

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

数据通路及其说明

计算机的工作过程可以看作是许多不同的数据流和控制流在机器各部分之间的流数据流所经过的路径称作机器的数据通路。数据通路不同，指令执行所经过的操作过程就不同，机器的结构也就不一样。如何设计一个好的数据通路已经超出了本课程的范围，在此我们不予讨论。我们假设所设计的计算机的数据通路如图所示。



1、 数据传送类指令的执行过程

寄存器之间的传送

MOV R1，R2

要求完成的操作为（R2）→R1，执行过程为：

由 R2 的编码通过 RAA1、RAA0 从通用寄存器组 A 口读出 R2 的内容，在 S3～S0 和 M 的控制下，经 ALU 送入总线 BUS；由/WE 控制和 R1 的编码选择 RWBA1、RWBA0，将 BUS 上的数据写入通用寄存器 R1。

寄存器到内存的传送

MOV M，R2

要求完成的操作为（R2）→（C），执行过程为：

由 M 的编码 11 通过 RWBA1、RWBA0 从通用寄存器 B 口读出 C 寄存器中的地址，在 MADD=2 的控制下，地址通过选择器到达存储器 RAM 的地址输入端；由 R2 的编码通过 RAA1、RAA0 从通用寄存器组 A 口读出 R2 的内容，在 S3～S0 和 M 的控制下，经 ALU 送入总线 BUS，并在/CS 和 XL 控制下将 BUS 上的数据写入存储器 RAM。

内存到寄存器的传送

MOV R1，M

要求完成的操作为（（C））→R1，执行过程为：

由 M 的编码 11 通过 RAA1、RAA0 从通用寄存器 A 口读出 C 寄存器中的地址，在 MADD=1 的控制下，地址通过选择器到达存储器 RAM 的地址输入端，/CS 和 DL 使数据出现在 BUS 上；由/WE 控制和 R1 的编码选择 RWBA1、RWBA0，将 BUS 上的数据写入通用寄存器 R1。

2、 算术逻辑运算类指令的执行过程

ADD R1，R2

SUB R1，R2

OR R1，R2

这类指令的执行过程为：

由 R2 的编码通过 RAA1、RAA0 从通用寄存器组 A 口读出 R2 的内容，由 R1 的编码通过 RWBA1、RWBA0 从通用寄存器组 B 口读出 R1 的内容，在 S3～S0 和 M 的控制下，经 ALU 送入总线 BUS；由/WE 控制和 R1 的编码选择 RWBA1、RWBA0，将 BUS 上的数据写入通用寄存器 R1。其中 ADD 和 SUB 指令影响状态位 Cf 和 Zf。

3、 移位指令的执行过程

RSR R1

RSL R1

这类指令的执行过程为：

由 R1 的编码通过 RWBA1、RWBA0 从通用寄存器组 B 口读出 R1 的内容，在 S3～S0 和 M 的控制下通过 ALU，经移位逻辑循环右移或循环左移后送入总线 BUS；再由/WE 控制和 R1 的编码选择 RWBA1、RWBA0，将 BUS 上的数据写入通用寄存器 R1。但是，标准的 ALU 模块没有移位功能，需要在该模块出口与总线接口处增加一部分电路以实现相应的移位功能（此问题请同学们自行解决）。

4、 转移类指令的执行过程

JMP add

JZ add

JC add

这类指令为双字节指令，第一字节为指令码，第二字节为转移目标地址。这类指令的执行过程为：在 MADD=0 的控制下，程序计数器 PC 中的地址通过选择器到达存储器 RAM 的地址输入端，在/CS 和 DL 控制下转移地址从 RAM 中读出并送入 BUS；如果条件满足（IN PC=0）则在 LD PC 允许下将 BUS 上的地址打 PC，否则 PC 加 1 计数。

当数据通路设计好之后，就要进行详细电路设计，这时需要考虑其它各种因素，比如进行触发器 Cf 和 Zf 的设置。

控制器设计

有了指令系统和数据通路之后，就可以进入控制器设计阶段。控制器设计有两种方法，一种是组合逻辑实现方法，另一种是微程序实现方法。我们采用第一种方法。

1、微控制信号

指令寄存器 IR 接收到一条机器指令后，这条指令就被译码执行。指令通过译码产生出的各种控制信号在时钟信号的配合下控制着指令执行的全过程。为此，需要将执行每条指令所需的全部基本微操作的控制信号罗列出来，进行综合分析、化简，并落实到不同的周期、节拍之中，然后用各种逻辑门电路实现。以下是所用基本控制信号列表。

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。

2、指令周期与工作脉冲设置

指令同期与数据通路结构、指令执行方式有关。指令可以串行执行，也可以并行执行。

本设计采用串行工作方式，即“读取—执行—再读取—再执行……”。串行工作方式虽然工作速度和主机效率都要差一些，但它的控制简单。因此，本机指令周期可以确定为：

取一条指执行一条指令一个指令周期读取指令的时间随所使用的 RAM 的性能而异。执行一条指令所需工作脉冲的个数与宽度要依据控制流和数据流所经过的路径与各级门的最大延迟而定。例如，本机中写入 RAM 和寄存器组的操作显然不能发生在“执行阶段”的任意时刻，它必须是在运算结果已经产生，并被传送到总线的适当时刻才能“写”，这就需要工作脉冲来控制时序

控制台与控制指令设计

有了数据通路和控制器，还要有输入/输出部分，我们才能将编好的程序送入系统，看到运算结果。为简单起见，我们仅考虑最简单的 I/O 部分。虽然是最简单的，但仍必须有以下几个功能：

① 能够启动系统运行；
② 能够输入地址和程序；
③ 能够检查内存的内容；
④ 能够终止系统的运行；
⑤ 能够对系统进行初始化；
⑥ 能够提供某些出错信息，如溢出等。

1、 命令键设置及实现

为了实现控制台的功能，我们可以设置以下几个命令键：

(1) START 键：启动系统运行，起始地址由 PC 当前内容确定；

(2) STOP 键：终止系统运行；

(3) INA 键：输入地址，将开关上设置的数作为地址送入 PC 中；

(4) INP 键：输入程序，将开关上设置的数作为机器指令代码送入当前 PC 所指定的内存单元中；

(5) CHP 键：检查程序，将当前 PC 所指定的内存单元的内容送显示器显示；

(6) RESTART 键：系统总复位命令。

还可以设置一些其它命令键。在上面 6 个键中，INA、INP、CHP 不仅要启动系统运行，还需要在命令执行完后停机。要注意，这样的命令没有自己的指令格式，需要另外设计编码电路对这些命令进行编码，并将其送入指令寄存器 IR。

2、 输入输出部件

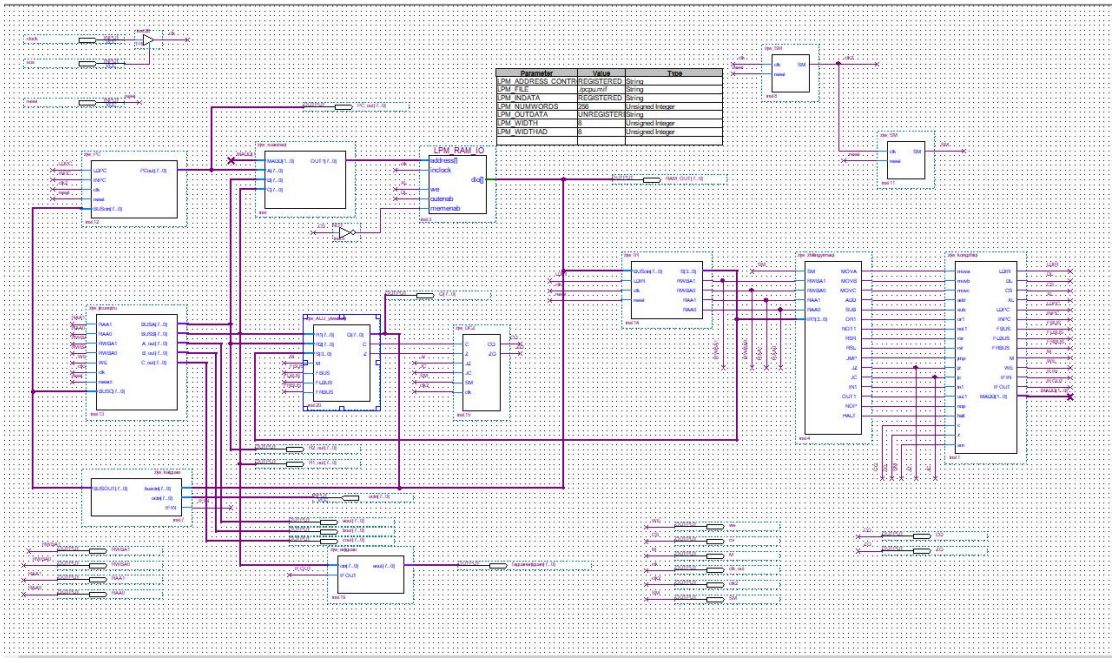
输入输出可以采用二进制、十进制或 ASCII 码。设备有多种多样，如 LCD 显示器及键

盘、打印机等。为简单起见，本机采用最简单的二进制开关作为输入部件，发光二极管作为

输出部件。发光二极管所显示的内容要由寄存器保存

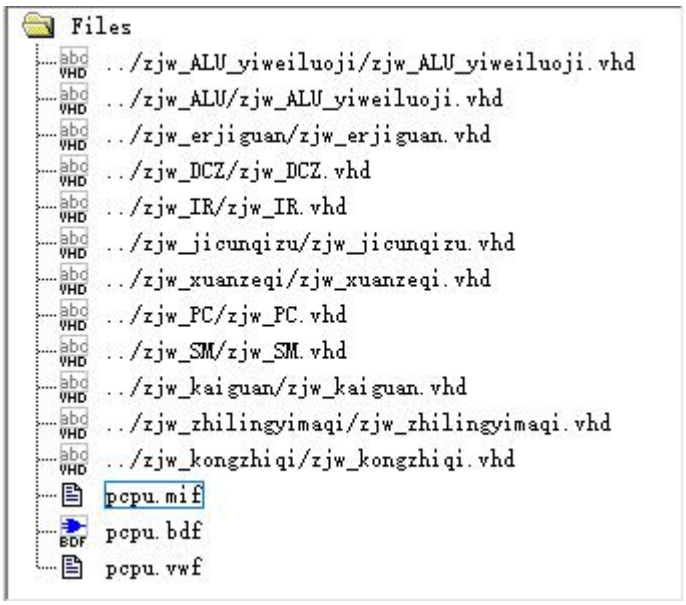
三、详细设计

3.1 设计的整体架构



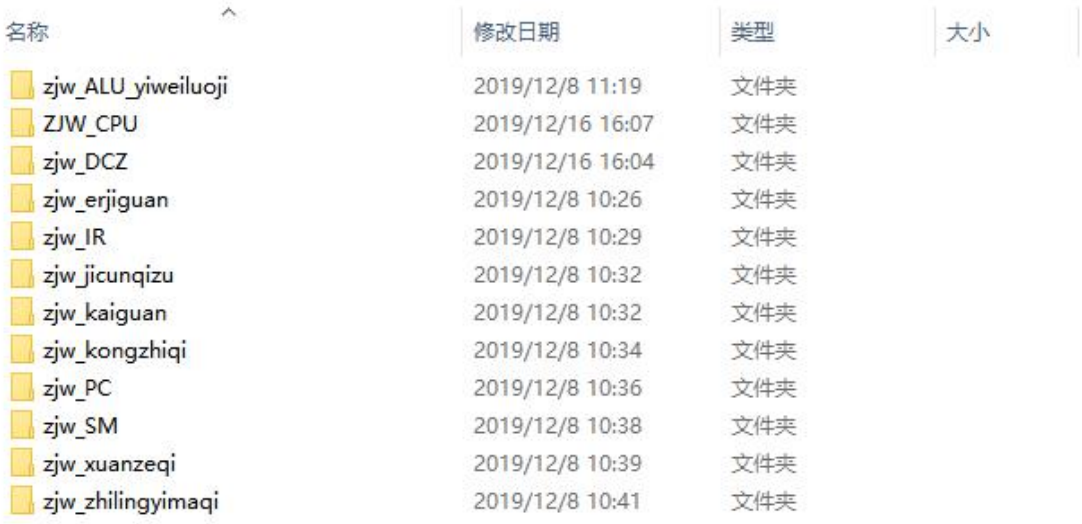
上图便是我设计的 cpu 的 bdf

cpu 工程中各个文件



cpu 工程中的文件

cpu 设计中每个元件的文件夹



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. PC 计数器

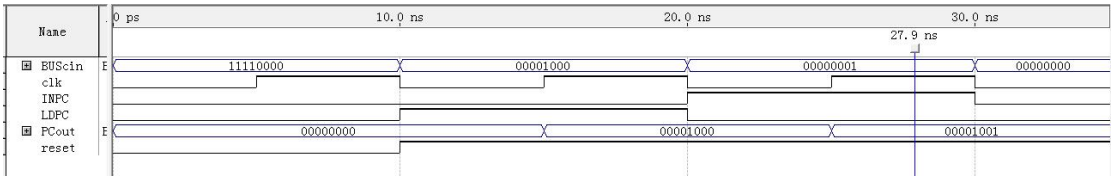
Vhdl

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   entity zjw_PC is
5   port(
6       LDPC,INPC,clk,reset:in std_logic;
7       BUScin:in std_logic_vector(7 downto 0);
8       PCout:out std_logic_vector(7 downto 0)
9   );
10  end zjw_PC;
11  architecture pfive of zjw_PC is
12  signal Q:std_logic_vector(7 downto 0);
13  begin
14      process(clk,reset)
15      begin
16          if(reset='0')then
17          Q<="00000000";
18          elsif(clk'event and clk='1')then
19              if(LDPC='1')then
20              Q<=BUScin;
21              elsif(INPC='1')then
22              Q<=Q+1;
23              end if;
24          end if;
25      end process;
26      PCout<=Q;
27  end pfive;
```

Pc 计数器一共有五个输入端口，一个输出端口，其中输入端口有一个为时钟信号，剩下 2 个为控制信号，一个复位信号，最后一个为总线的输入，

功能为当 reset 复位信号为 0 时，PC 内的数据清 0，当 inpc 为 1 时 PC=PC+1；当 ldpc 为 1 时 PC=cin

功能仿真结果



结果分析

当 reset 为 0 时，输出为 00000000

当 reset 为 1 时，ldpc 为 1 时，输出为输入即 00001000

当 reset 为 1 时 ，inpc 为 1 时，输出并不为输入，而是 pc+1 即为 00001001

2. 选择器

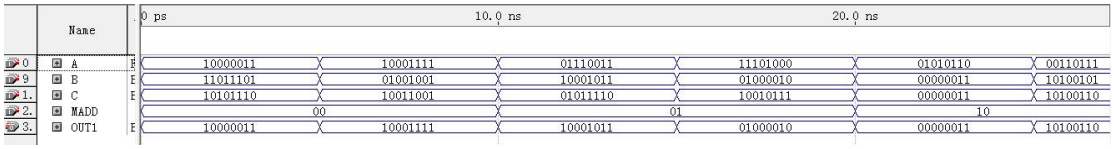
Vhdl

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity zjw_xuanzeqi is
4   port(
5       MADD:in std_logic_vector(1 downto 0);
6       A,B,C:in std_logic_vector(7 downto 0);
7       OUT1:out std_logic_vector(7 downto 0)
8   );
9   end zjw_xuanzeqi;
10  architecture psix of zjw_xuanzeqi is
11  begin
12      process(MADD)
13      begin
14          if(MADD="00")then
15          OUT1<=A;
16          elsif(MADD="01")then
17          OUT1<=B;
18          elsif(MADD="10")then
19          OUT1<=C;
20          end if;
21      end process;
22  end psix;
```

选择器一共五个端口，三个输入端口，一个控制信号端口，一个输出端口

当控制信号 madd 分别为 00 01 10 选择器分别输出 a b c 三个输入

功能仿真



结果分析

当 madd 为 00 时，输出与 a 口一致，

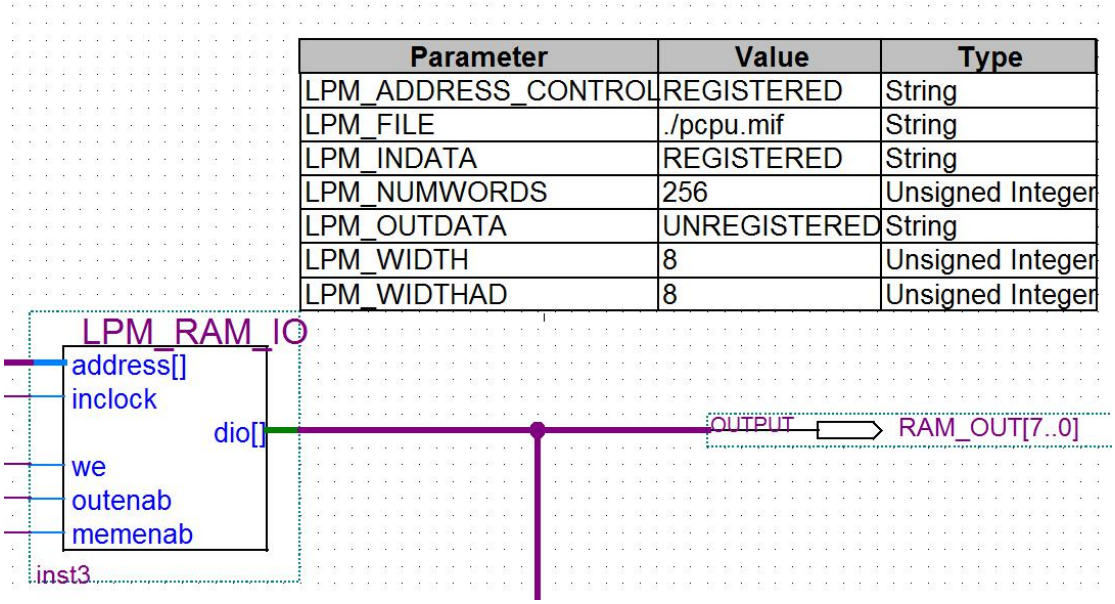
01 与 b 口一致

10 与 c 口一致

3. RAM

bsf

Parameter	Value	Type
LPM_ADDRESS_CONTROL	REGISTERED	String
LPM_FILE	./pcpu.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



为基础的 LPM_RAM_IO 元件能够实现读写功能

Ram 对应的 mif 文件

Addr	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13
0	00100000	00100100	00101000	11110001	11111101	11110011	10010001	01100001	10110001	10100000	10100011	01010000	00010000	00010000
14	00000000	00000000	01110000	01000000	10000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
28	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
42	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
70	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
84	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
98	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
112	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
126	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
140	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
154	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
168	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
182	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
196	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
210	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
224	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
238	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
252	00000000	00000000	00000000	00000000										

4. IR 寄存器

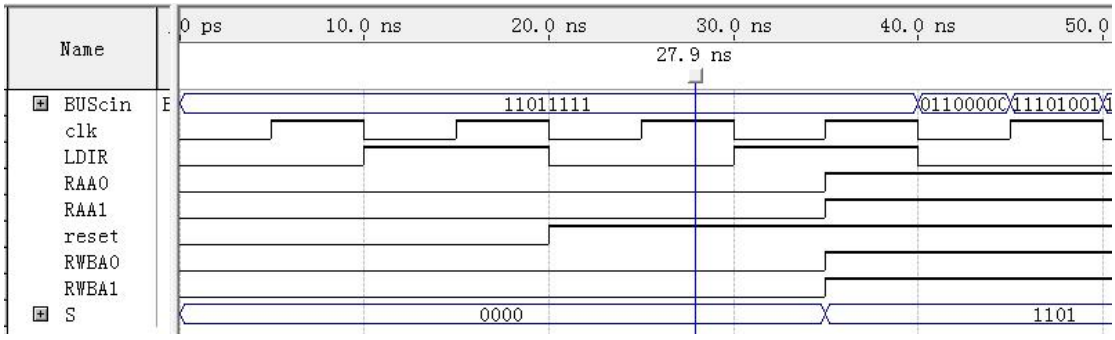
Vhdl

```
library ieee;
use ieee.std_logic_1164.all;
entity zjw_IR is
port(
    BUScin:in std_logic_vector(7 downto 0);
    LDIR,clk,reset:in std_logic;
    S:out std_logic_vector(3 downto 0);
    RWBA1,RWBA0,RAA1,RAA0:out std_logic
);
end zjw_IR;
architecture peight of zjw_IR is
signal Q:std_logic_vector(7 downto 0);
begin
    process(LDIR,clk,reset)
    begin
        if(reset='0')then
            Q<="00000000";
        elsif(clk'event and clk='1')then
            if(LDIR='1')then
                Q<=BUScin;
            end if;
        end if;
    end process;
    S<=Q(7 downto 4);
    RWBA1<=Q(3);
    RWBA0<=Q(2);
    RAA1<=Q(1);
    RAA0<=Q(0);
end peight;
```

一共有 9 个端口其中一个数据输入端口，一个复位信号端口，两个控制信号端口，5 个输出端口

功能为 Reset 为 0 时，IR 寄存器中储存的值归 0；Reset 为 1 时，当 LDIR 为 1 时，输出

功能仿真



当 reset 为 0 时无论 LDIR 所有的输出都为 0

当 reset 为 1，LDIR 为 1 时，在适中的上升沿，输出与输入相符

5. 指令译码器

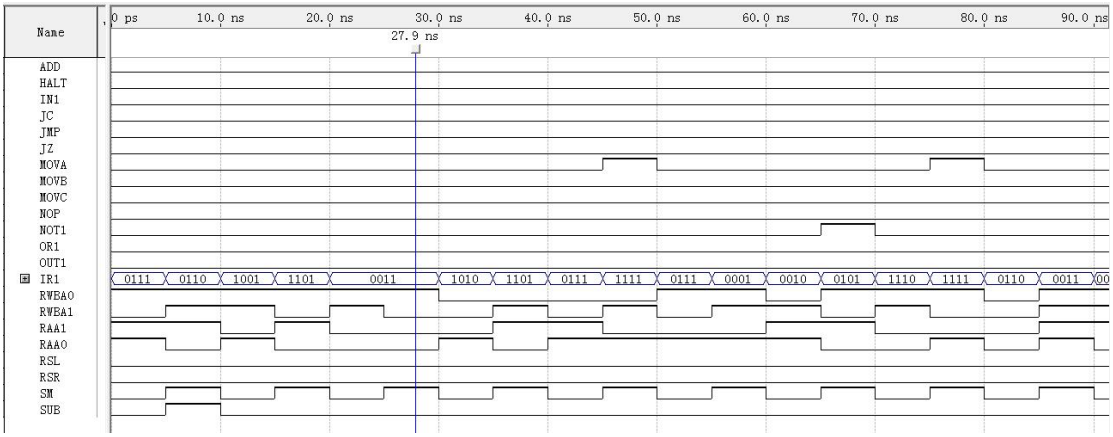
Vhdl

```
library ieee;
use ieee.std_logic_1164.all;
entity zjw_zhilingyimaqi is
port(
    SM,RWBA1,RWBA0,RAA1,RAA0:in std_logic;
    IR1:in std_logic_vector(3 downto 0);
    MOVA,MOVB,MOVC,ADD,SUB,OR1,NOT1,RSR,RSL,JMP,JZ,JC,IN1,OUT1,NOP,HALT:out std_logic
);
end entity zjw_zhilingyimaqi;

architecture pp of zjw_zhilingyimaqi is
signal R1:std_logic_vector(1 downto 0);
signal R2:std_logic_vector(1 downto 0);
begin
    R1<=RWBA1&RWBA0;
    R2<=RAA1&RAA0;
    process(SM,IR1,R1,R2)
    begin
        MOVA<='0';
        MOVB<='0';
        MOVC<='0';
        ADD<='0';
        SUB<='0';
        OR1<='0';
        NOT1<='0';
        RSR<='0';
        RSL<='0';
        JMP<='0';
        JZ<='0';
        JC<='0';
        IN1<='0';
        OUT1<='0';
        NOP<='0';
        HALT<='0';
        if(SM='1') then
            if(IR1="1111") then
                if(R1="11") then
                    MOVB<='1';
                elsif (R2="11") then
                    MOVC<='1';
                else
                    MOVA<='1';
                end if;
            elsif(IR1="1001") then
                ADD<='1';
            elsif(IR1="0110") then
             elsif(IR1="0110") then
                 SUB<='1';
             elsif(IR1="1011") then
                 OR1<='1';
             elsif(IR1="0101") then
                 NOT1<='1';
             elsif(IR1="1010") then
                 if(R2="00")then
                     RSR<='1';
                 elsif(R2="11")then
                     RSL<='1';
                 end if;
             elsif(IR1="0001" and R1="00")then
                 if(R2="00") then
                     JMP<='1';
                    JMP<='1';
                elsif(R2="01") then
                    JZ<='1';
                elsif(R2="10") then
                    JC<='1';
                end if;
            elsif(IR1="0010") then
                IN1<='1';
            elsif(IR1="0100") then
                OUT1<='1';
            elsif(IR1="0111") then
                NOP<='1';
            elsif(IR1="1000") then
                HALT<='1';
            end if;
        end if;
    end process;
end architecture pp;
```

一共有 22 个端口，其中一个类时钟信号输入端口，5 个指令输入端口，16 个输出端口

功能：在 sm（类时钟信号）的执行周期，根据输入的 ir 以及剩下的四位输入合成的 8 位向量，输出对应的 16 个操作的执行与否

功能仿真



当输入满足要求的指令后在 sm 的执行周期，将会有对应的操作端口输出为 1

6. 控制器

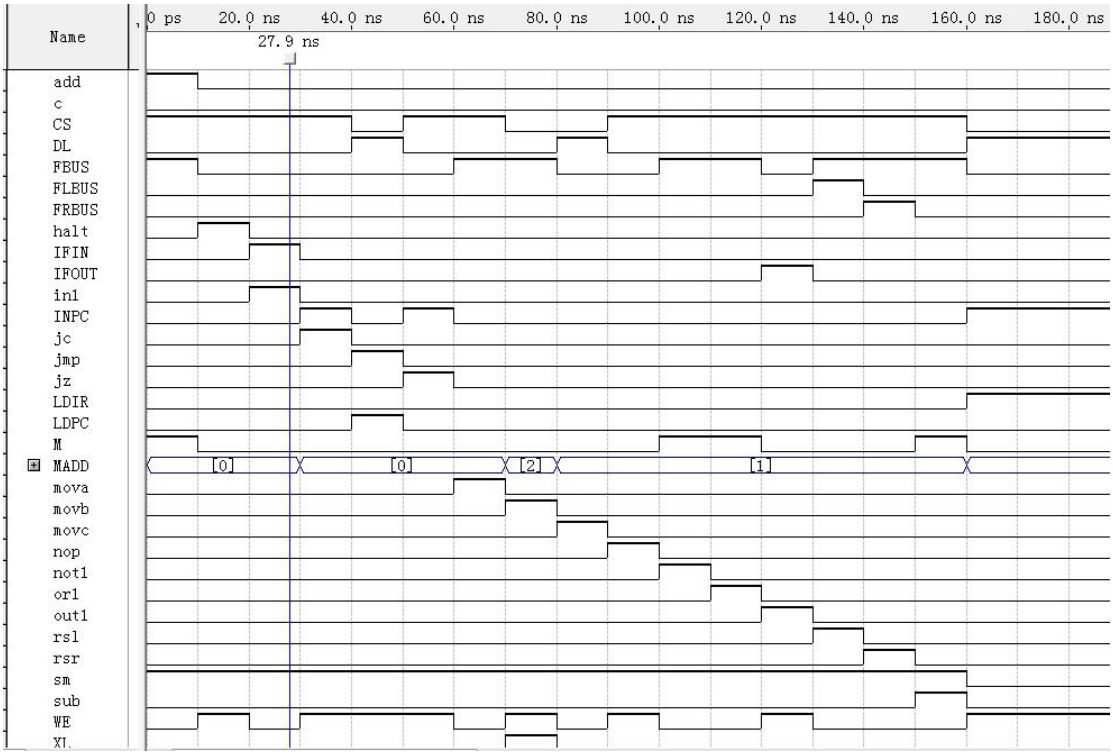
Vhdl

```
library ieee;
use ieee.std_logic_1164.all;
entity zjw_kongzhiqi is
 port(mova,movb,movc,add,sub,orl,notl,rsr,rsl,jmp,jz,jc,inl,outl,nop,halt,c,z,sm : in std_logic;
      LDIR,DL,CS,XL,LDPC,INPC,FBUS,FLBUS,FRBUS,M,WE,IFIN,IFOUT:out std_logic;
      MADD : out std_logic_vector(1 downto 0)
       );
end zjw_kongzhiqi;

architecture p of zjw_kongzhiqi is
begin
 LDIR <= '1' when (sm='0') else '0';
 DL <= '1' when (sm='0' or movc='1' or jmp='1' or (jz='1' and z='1') or (jc='1' and c='1')) else '0';
 XL <= '1' when (movb='1') else '0';
 cs <= '0' when (sm='0' or movb='1' or movc='1' or jmp='1' or (jz='1' and z='1') or (jc='1' and c='1')) else '1';
 LDPC <= '1' when (jmp='1' or (jz='1' and z='1') or (jc='1' and c='1')) else '0';
 INPC <= '1' when (sm='0' or (jc='1' and c='0') or (jz='1' and z='0')) else '0';
 FBUS <= '1' when (mova='1' or movb='1' or add='1' or sub='1' or orl='1' or notl='1' or rsr='1' or rsl='1') else '0' ;
 FLBUS<='1' when (rsl='1') else '0';
 FRBUS<='1' when (rsr='1') else '0';
 M<='1' when (orl='1' or notl='1' or add='1' or sub='1') else '0';
 WE<='1' when (sm='0' or movb='1' or outl='1' or nop='1' or halt='1' or jmp='1' or jz='1' or jc='1') else '0';
 MADD<="00" when (sm='0' or jmp='1' or jz='1' or jc='1') else
        "01" when (movc='1') else
        "10" when (movb='1');
 IFIN <= '1' when (inl='1') else '0';
 IFOUT <= '1' when (outl='1') else '0';
 end p;
```

一共有 16 个操作输入端口，2 个判断信号输入端口，1 个类时钟 sm 输入端口，14 个控制信号输出端口

功能，根据输入的操作的执行与否以及 cz 标志，在 sm 的执行周期与控制周期发出其余元件的控制信号

功能仿真



所有信号与输入的关系皆与实验要求相符

7. 寄存器组

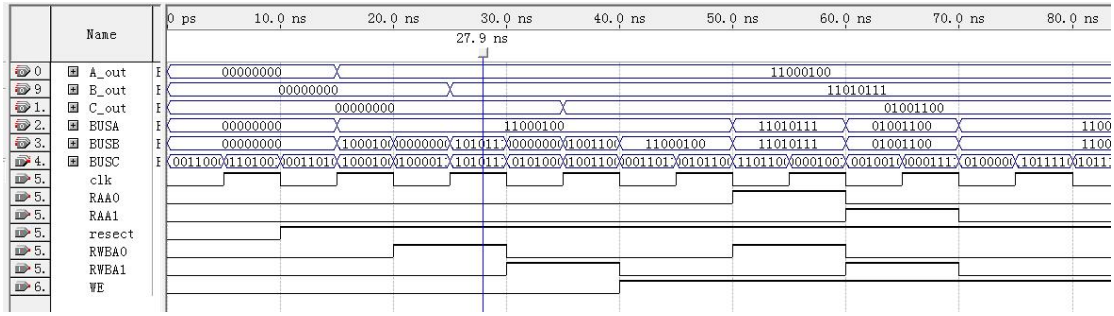
Vhdl

```
library ieee;
use ieee.std_logic_1164.all;
entity zjw_jicunqizu is
port(
    RAA1,RAA0,RWBA1,RWBA0,WE,clk,resect:in std_logic;
    BUSC:in std_logic_vector(7 downto 0);
    BUSA,BUSB:out std_logic_vector(7 downto 0);
    A_out,B_out,C_out : out std_logic_vector(7 downto 0));
end zjw_jicunqizu;
architecture pseven of zjw_jicunqizu is
signal A,B,C,OUTA,OUTB:std_logic_vector(7 downto 0);
signal S1,S2:std_logic_vector(1 downto 0);
begin
    S1<=RWBA1&RWBA0;
    S2<=RAA1&RAA0;
    process(clk,resect,A,B,C,BUSC)
    begin
        if(resect='0')then
            A<="00000000";
            B<="00000000";
            C<="00000000";
        elsif(clk'event and clk='1')then
            if(WE='0')then
                if(S1="00")then
                    A<=BUSC;
                elsif(S1="01")then
                    B<=BUSC;
                elsif(S1="10")then
                    C<=BUSC;
                end if;
            end if;
        end if;
    end process;
    BUSA<=A when S2="00" else
            B when S2="01" else
            C when S2="10" or S2="11";
    BUSB<=A when S1="00" else
            B when S1="01" else
            C when S1="10" or S1="11";
    A_out<=A;
    B_out<=B;
    C_out<=C;
end pseven;
```

一共有一个总线数据输入端口，5 个控制信号输入端口，一个复位信号输入端口，1 个时钟输入端口，以及五个输出端口

功能：可以实现对寄存器组中 abc 三个寄存器的读与写，并且将 abc 三个寄存器内容输出

功能仿真



当 reset 为 0 时，abc 三个寄存器内容皆为 00000000

当 reset 为 1，we 为 0，rwba1，rwba0 分别为 00 01 10 时，在时钟的上升沿，abc 三个寄存器的值分别等于输入

We 为 1 后 rwba1.rwba0，raa1，raa0 分别为 0000 0101 1010 AB 两个接口分别输出的为 abc 三个寄存器此前储存的值

8. ALU

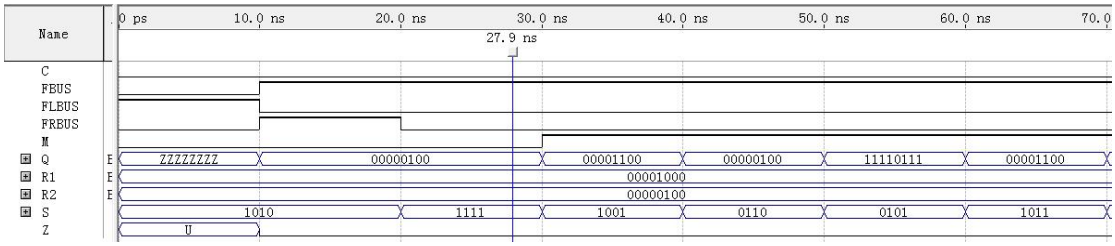
Vhdl

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.std_logic_unsigned.all;
 4  use ieee.std_logic_arith.all;
 5  entity zjw_ALU_yiweiluoji is
 6  port(
 7      R1,R2:in std_logic_vector(7 downto 0);
 8      S:in std_logic_vector(3 downto 0);
 9      M,FBUS,FLBUS,FRBUS:in std_logic;
10      Q:out std_logic_vector(7 downto 0);
11      C,Z:out std_logic
12  );
13  end entity zjw_ALU_yiweiluoji;
14  architecture pten of zjw_ALU_yiweiluoji is
15  signal ZZ:std_logic_vector(8 downto 0);
16  begin
17      process(S,R1,R2,FBUS)
18      begin
19          if(M='1')then
20              if(S="1011") then--OR
21                  ZZ<='0'&(R1 or R2);
22              elsif(S="0101") then--NOT
23                  ZZ<='0'&not(R1);
24              end if;
25              if(S="1001") then--ADD
26                  ZZ<=('0'&R1)+('0'&R2);
27              elsif(S="0110") then--SUB
28                  ZZ<=('0'&R1)-('0'&R2);
29              end if;
30          elsif(FBUS='1')then
31              if(FRBUS='1')then--RSR
32                  ZZ<=R1(0)&'0'&R1(7 downto 1);
33              elsif(FLBUS='1')then--RSL
34                  ZZ<=R1&'0';
35              end if;
36              if(S="1111")then--MOV
37                  ZZ<='0'&R2;
38              end if;
39          elsif(FBUS='0')then
40              ZZ<="0ZZZZZZZZ";
41          end if;
42          if(zz(7 downto 0)="00000000")then
41          end if;
42          if(zz(7 downto 0)="00000000")then
43              Z<='1';
44          else
45              Z<='0';
46          end if;
47      end process;
48      Q<=ZZ(7 downto 0);
49      C<=ZZ(8);
50  end architecture pten;
```

一共有两个数据输入端口，5 个控制信号输入端口，一个结果输出端口，两个标志位输出端口

功能：根据控制信号，对进入 alu 的两个数据执行加减或非循环左右移操作

功能仿真



当 FBUS=0 时 q 没有输出

当 FBUS=1 时

当 FLBUS、FRBUS 分别为 1 0 时 q 为 r1 循环左移的结果 为 01 时 q 为 r1 循环右移结果

当 s 为 1111 时 q 为 r2 为 1001 时 q 为 r1+r2 为 0110 时 q 为 r1−r2 为 0101 q 为−r1 为 1011 时 q 为 r1 or r2

9. CZ 暂存器

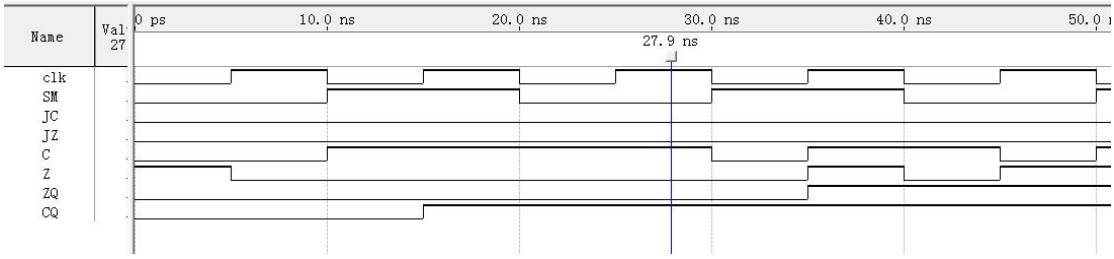
Vhdl

```
library ieee;
use ieee.std_logic_1164.all;
entity zjw_DCZ is
port(
    C,Z,JZ,JC,SM,clk:in std_logic;
    CQ,ZQ:out std_logic);
end zjw_DCZ;
architecture pnine of zjw_DCZ is
signal X,Y:std_logic;
begin
    process(JZ,JC,C,Z)
    begin
        if(clk'event and clk='1')then
            if(SM='1')then
                X<=C;
                Y<=Z;
            end if;
        end if;
    end process;
    CQ<=X;
    ZQ<=Y;
end pnine;
```

一共有 2 个标志输入信号，2 个控制信号，2 个时钟信号，2 个标志输出信号

功能：在执行周期时钟上升沿将 cz 放出（方便整个 cpu 的时钟分配的调节）

功能仿真



当 sm 为 1 时时钟上升沿 cz 被放出

10.（模拟）二极管

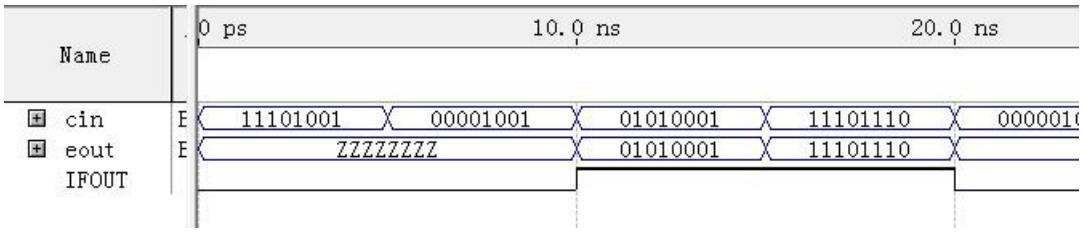
Vhdl

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

entity zjw_erjiguan is
 port(cin : in std_logic_vector(7 downto 0);
      IFOUT : in std_logic;
      eout : out std_logic_vector(7 downto 0));
end zjw_erjiguan;

architecture peleven of zjw_erjiguan is
begin
process(IFOUT)
 begin
if(IFout='1') then
 eout<=cin;
else
 eout<="ZZZZZZZZ";
 end if;
 end process;
 end peleven;
```

一共有一个数据输入端口，一个数据输出端口，一个控制信号端口

功能：由于没有硬件，模拟一下输出情况，当 ifout 为 1 时输出输入数据

功能仿真



当 IFOUT 为 0 时输出为高组态

当 IFOUt 为 1 时输出为输入

11.（模拟）开关

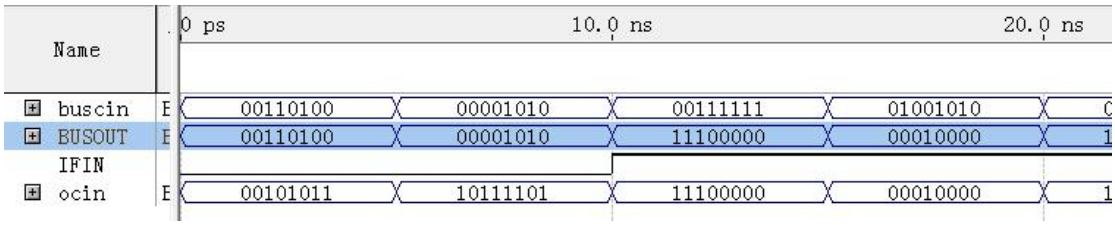
Vhdl

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

entity zjw_kaiguan is
 port(buscin,ocin : in std_logic_vector(7 downto 0);
       IFIN : in std_logic;
       BUSOUT : out std_logic_vector(7 downto 0)
        );
end zjw_kaiguan;
architecture ppp of zjw_kaiguan is
begin
     process(IFIN)
     begin
     if(IFIN = '1') then
     BUSOUT<=ocin;
     else
     BUSOUT<=buscin;
     end if;
     end process;
end ppp;
```

一共有两个数据输入端口，一个控制信号输入端口，一个数据输出端口

功能当 IFIN 为 1 时输出额外输入的值，为 0 时输出总线上输入的值（由于开关需要操作台，所以只能模拟一下）

功能仿真



当 IFIN 为 0 时 输出 BUSOUT 与 BUSCIN 一致 为 1 时输出与 OCIN 一致

12. SM 创造器

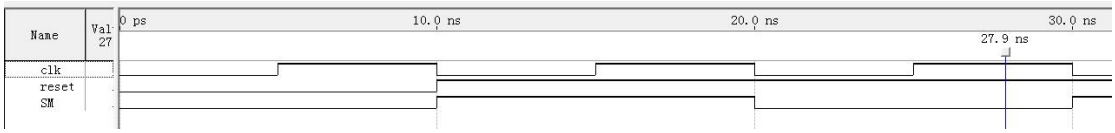
Vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity zjw_SM is
4  port(
5      clk,reset:in std_logic;
6      SM:out std_logic
7  );
8  end zjw_SM;
9  architecture pfour of zjw_SM is
10 signal Q1:std_logic;
11 begin
12     process(clk,reset)
13     begin
14         if(reset='0')then
15             Q1<='0';
16         elsif(clk'event and clk='0')then
17             if(Q1='0')then
18                 Q1<='1';
19             else
20                 Q1<='0';
21             end if;
22         end if;
23     end process;
24     SM<=Q1;
25 end pfour;
```

一共有一个时钟输入信号端一个时钟输出信号端，一个复位信号输入端

功能：将时钟周期增加一倍，频率减少一倍，即创造 sm

功能仿真



SM 成功变为 clk 频率的一半

四、系统测试

4.1 测试环境

QuartusII

4.2 测试代码

（使用你已实现的指令编写一至两个程序测试以检测模型机的正确性）

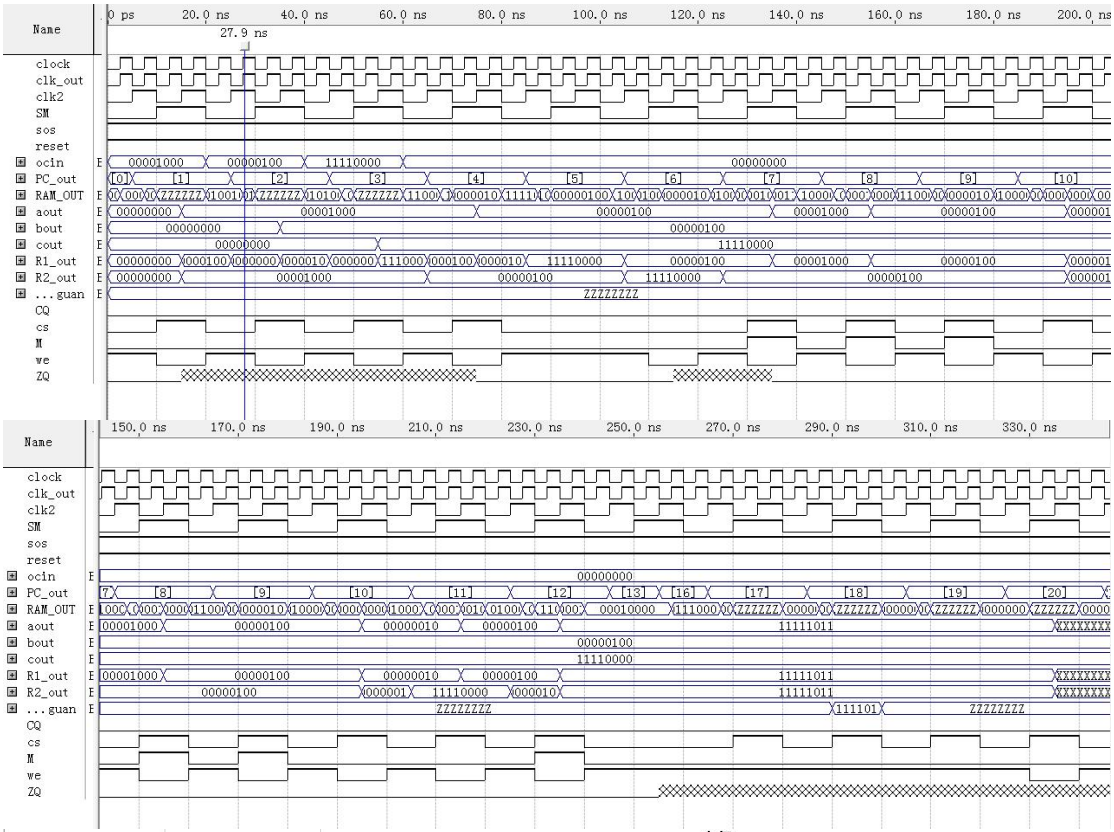
00100000	00100100	00101000	11110001	11111101	11110011	10010001	01100001
10110001	10100000	10100011	01010000	00010000	00010000	00000000	00000000
01110000	01000000	10000000	00000000	00000000	00000000	00000000	00000000

此为 ram 中的设计的指令,按照顺序为

输入 abc,mova,movb,movc,ADD,SUB,OR,RSR,RSL,NOT,JMP,NOP,OUT,HALT

4.3 测试结果

由于时钟信号的原因，pc 两个值（如 0,1）之间正下方对的 ram_out 即为 ram 当前值 ram（0），cpu 执行的时候取址与上一个执行周期有重复部分，即流水。



指令	功能	表征										
		ocin	RAM_out	aout	bout	cout	R1_out	R2_out	RAM_out″	aout″	PC_out	kaiguan
00100000	输入a	00001000		00001000								
00100100	输入b	00000100			00000100							
00101000	输入c	11110000				11110000						
11110001	mova			00001000	00000100		00001000	00000100				
11111101	movb				00000100	11110000	11110000	00000100				
11110011	movc			00000100		11110000	00000100	11110000				
10010001	ADD			00000100	00000100				00001000	00001000		
01100001	SUB			00001000	00000100				00000100	00000100		
10110001	OR			00000100	00000100				00000100	00000100		
10100000	RSR			00000100					00000010	00000010		
10100011	RSR			00000010					00000100	00000100		
01010000	NOT			00000100					11111011	11111011		
00010000-00010000	JMP		00100000								00010000	
01110000	NOP										00010001	
01000000	NOP			11111011								11111011
10000000	HALT											
00010001-00010000	JZ											
00010010-00010000	JC											

按照 ram 中储存的值的顺序，对应操作表，得出结果皆符合要求

五、总结

本次实验，让我对整个 cpu 有了更加深入的理解，从将之前的实验中的元件内容相结合，到新设计控制器等元件，然后将元件相连，然后在修改的过程中，我对 EDA 有了更加深入的理解。