“计算机设计与实践”处理器实验设计报告

姓名：王叶

班级：1403103

学号：1140310311

哈尔滨工业大学计算机学院

2016 年 7 月

寄存器 0 号值为 0，请勿通过运算修改！

目录

一．实验概述

1.实验环境

Xilinx ISE9.1，Modelsim
Cop2000 实验台

2.实验目的

（1）掌握 Xilinx ISE 集成开发环境使用方法
（2）掌握 VHDL 语言
（3）掌握 FPGA 编程方法及硬件调试手段
（4）深刻理解处理器结构和计算机系统的整体工作原理

3.实验内容

根据计算机组成原理课程所学的知识和本课程所讲的设计思想，设计并实现一个给定指令系统的处理器。设计一个简单的 RISC 处理器，在给定的指令集下构建，支持十条指令。

二．详细设计及整体框图

1.CPU 接口信号定义

信号名	位数	方向	来源/去向	意义
RST	1	I	处理器板	高电平复位信号
CLK	1	I	处理器板	系统时钟
ABUS	16	O	主存储器	地址总线
DBUS	16	I/O	主存储器	数据总线
nmreq	1	O	主存储器	存储器片选
nrd	1	O	主存储器	存储器读
nwr	1	O	主存储器	存储器写
nbhe	1	O	主存储器	高字节访问允许
nble	1	O	主存储器	低字节访问允许
nPREQ	1	O	外设	端口片选信号
nPRD	1	O	外设	端口读有效
nPWR	1	O	外设	端口写有效

信号名	位数	方向	来源/去向	意义
K0	8	I	外设端口	端口 0
K1	8	I	外设端口	端口 1
K2	8	I	外设端口	端口 2
K3	8	I	外设端口	端口 3
S5	8	O	数码管	地址高八位
S4	8	O	数码管	地址低八位
S2	8	O	数码管	指令高八位
S1	8	O	数码管	指令低八位
S0	8	O	数码管	外设输出
S3	8	O	数码管	外设输出

2.指令格式设计

指令的操作码（opcode）为 5 位，寄存器号为 3 位，立即数或地址（未扩展）为 8 位，端口号为 2 位，其余位为 0。

要求寄存器 0 的值始终默认为 0

参照 MIPS 的指令格式，将指令分为以下三类：

a.5+3+3+5

Opcode（15~11）	Rs（10~8）	Rt（7~5）	00000（4~0）

ADD,SUB,MOV 都含有两个寄存器号，后五位为 0

b.5+3+8

Opcode（15~11）	Rs（10~8）	Immediate/addr（7~0）

MVI,STA,LDA,JZ,JMP 都含有一个寄存器号和一个八位数

其中，JZ 指令是否发生跳转取决于 RS 寄存器内部数据是否为 0
不妨将 JMP 指令设计为相同的格式，它使用的寄存器为 REG（0），其内部数据始终保持为 0，故始终发生跳转。

c.5+3+2+6

Opcode（15~11）	Rs（10~8）	Port_n（7~6）	000000（5~0）

IN,OUT 指令都含有一个寄存器号和 2 位的端口号，其余位为 0

操作码（**opcode**）的分配

00000	ADD
00001	SUB
00010	MOV
00011	MVI
00100	STA
00101	LDA
00110	JZ
00111	JMP
01000	IN
01001	OUT

3.微操作的定义

定义 Ad1(IR)为第一个寄存器号，Ad2(IR)为第二个寄存器号，Ad(IR)为后八位数字

取指阶段

PC -> MAR, 1-R, M(MAR) -> MDR,MDR -> IR

执行运算

ADD RS, RT　　REG(AD1(IR))+REG(AD2(IR)) -> REG(AD1(IR)), PC+1->PC;

SUB RS, RT　　REG(AD1(IR))-REG(AD2(IR)) -> REG(AD1(IR)), PC+1->PC

MOV RS, RT　　REG(AD2(IR)) -> REG(AD1(IR)), PC+1->PC;

MVI RS, X　　X->REG(AD1(IR)), PC+1->PC;

访存指令

STA RS, X　　REG(R7)//AD(IR)->MAR, 1->W,
REG(AD1(IR))->MDR,
MDR->M(MAR),
PC+1->PC;

LDA RS, X　　REG(R7)//AD(IR)->MAR,1->R,
M(MAR)->MDR,
MDR->REG(AD1(IR)),
PC+1->PC;

跳转指令

JZ RS,X　　ZERO(REG(AD1(IR)))*REG(R7)//AD(IR) + NZERO(REG(AD1(IR)))*(PC+1) -> PC;

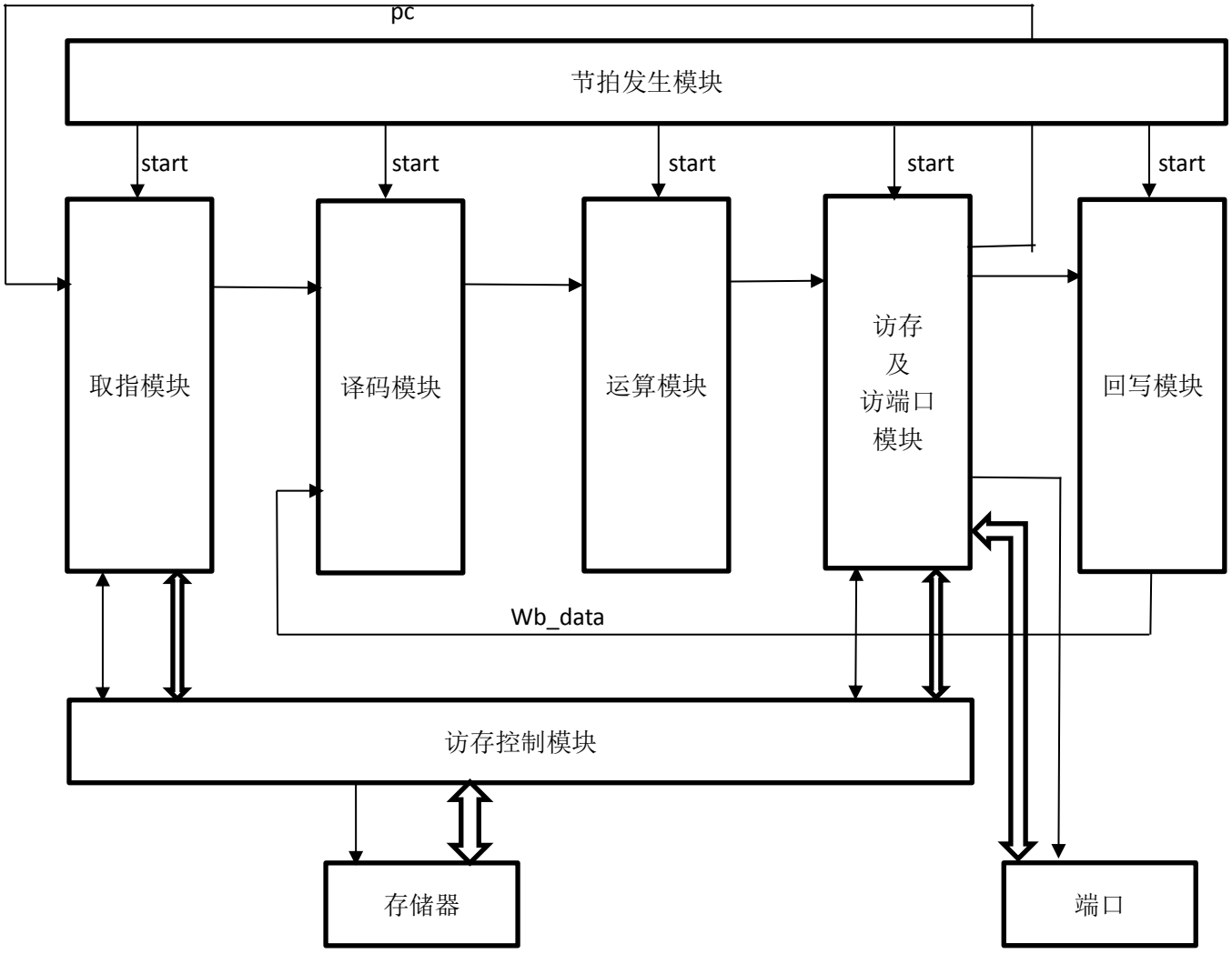
JMP X　　REG(R7)//AD(IR) -> PC;（实际微操作同 JZ，REG(AD1(IR))为寄存器 0，其值必须设为 0）

4. 模块及节拍的划分

在一个时钟周期内，所有操作都可以同时进行，所以划分为将一个指令的执行划分为五个周期，每个周期需要一个节拍

T0	取指模块	取指周期
T1	译码模块	译码周期
T2	运算模块	运算周期
T3	访存及端口模块	读写外设及主存周期
T4	回写模块	回写周期

5. 处理器结构设计及功能描述

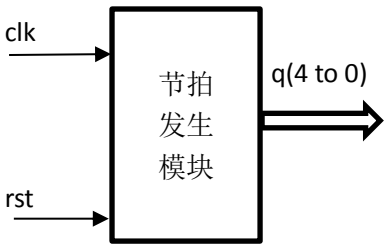


取指模块负责获取指令，译码模块负责按照指令获取操作数并且指导后续执行过程，运算模块负责运算，访存及端口模块负责读写主存和端口，回写模块完成。

节拍发生模块负责控制当前指令进行到的阶段，访存控制模块负责读写寄存器(决定当前哪个模块发来的读写信号有效)。

三．各功能模块结构设计框图、输入输出信号定义、功能描述

1.节拍发生模块



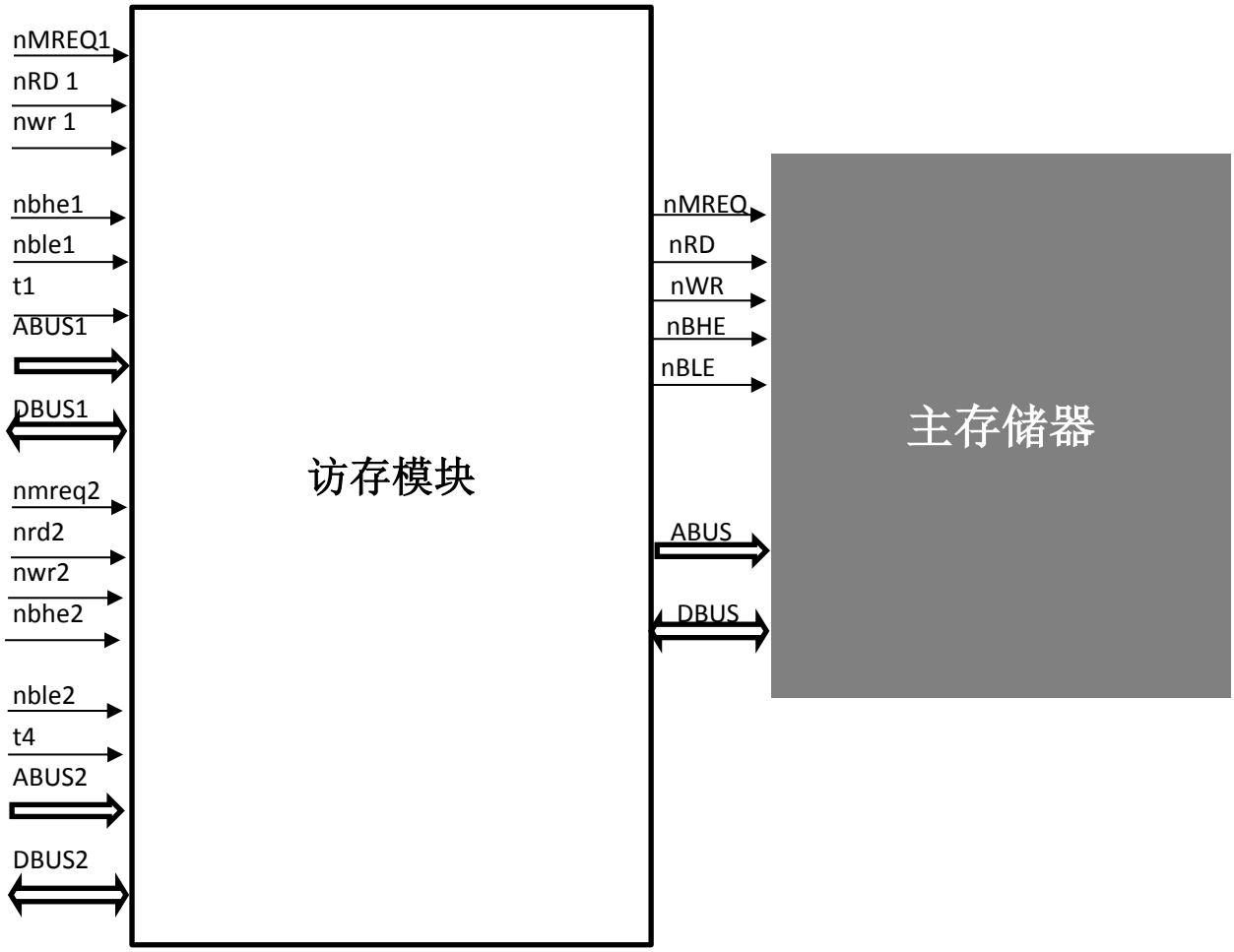
rst 为高电平时，系统复位，所有控制其他模块的信号均置零。
rst 为低电平时，clk 每出现一个上升沿，指令执行一个阶段。
用来控制其他模块的执行状态，及整体复位。
不同指令的节拍发生是一样的。

信号名	位数	方向	来源/去向	意义
clk	1	I	外部输入	时钟信号
rst	1	I	外部输入	高电平复位信号
q	5	O	其他模块及输出到外部	其他模块的起始信号

2. 访存模块

其他模块中，取指模块和访存及访端口模块都会对存储器进行读写操作，访存模块将根据指令当前进行到的阶段决定哪个模块对存储器的操作有效，并访问存储器。

不同指令都会在执行过程中两次使访存模块工作，一次是在第一个阶段，传送 PC 取回指令，另一次是在第四个阶段，有可能读写主存，将根据该指令的类型决定访存的相关信号是否有效。

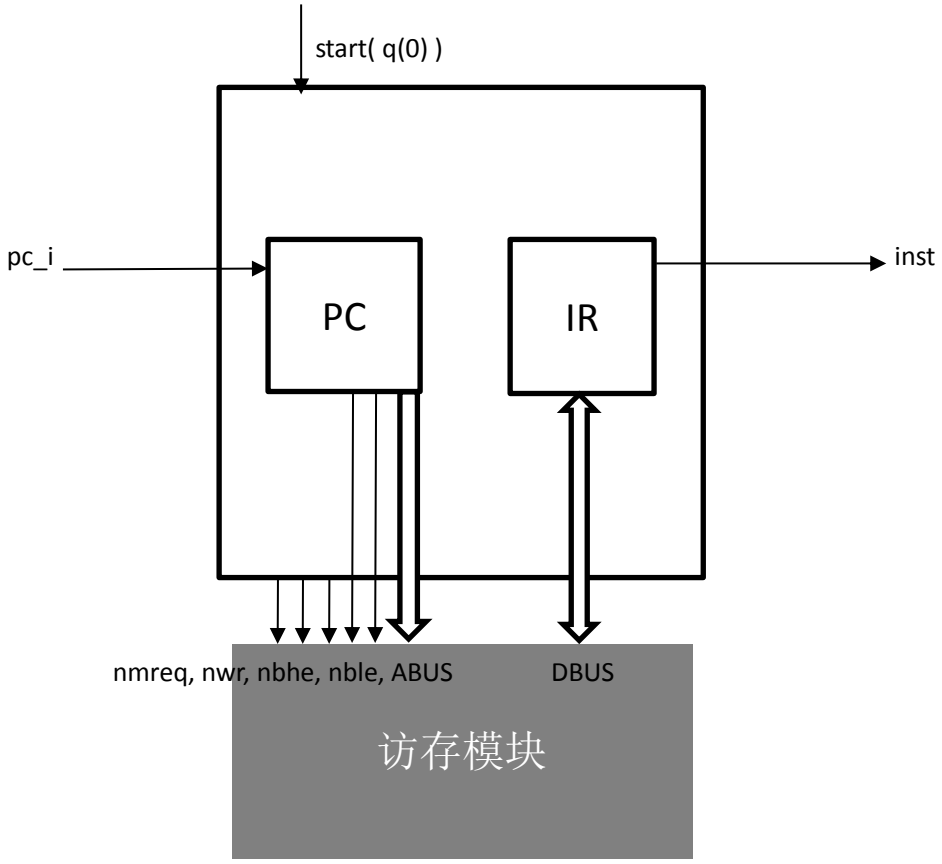


信号名	位数	方向	来源/去向	意义
T1	1	I	节拍发生模块	取指阶段开始的标志
nmreq1,nrd1,nwr1,nbhe1,nble1	1	I	取指模块	取指阶段对主存的操作信号
ABUS1	16	I	取指模块	取指阶段的地址
DBUS1	16	I/O	取指模块	指令线
T1	1	I	节拍发生模块	读写主存、端口阶段开始的标志
nmreq2,nrd2,nwr2,nbhe2,nble2	1	I	访存及访端口模块	访存及端口阶段对主存的操作信号
ABUS2	16	I	访存及访端口模块	读写数据的地址
DBUS2	16	I/O	访存及访端口模块	读写的数据
nmreq,nrd,nwr,nbhe,nble	1	O	主存储器	对主存储器的有效操作信号
ABUS	16	I	主存储器	地址总线
DBUS	16	I/O	主存储器	数据总线

3. 取指模块

是指令运行的第一个阶段，当起始信号到来时开始工作，根据当前 PC 向访存模块传递访存信号，取回

指令至 IR 寄存器。
不同类型指令在这一阶段的执行内容是一样的。
注：由于 rst 是一个重要的信号，不选择把本信号传入取指模块，而是在最大的整合模块中使之对 PC 有复位的决定作用。



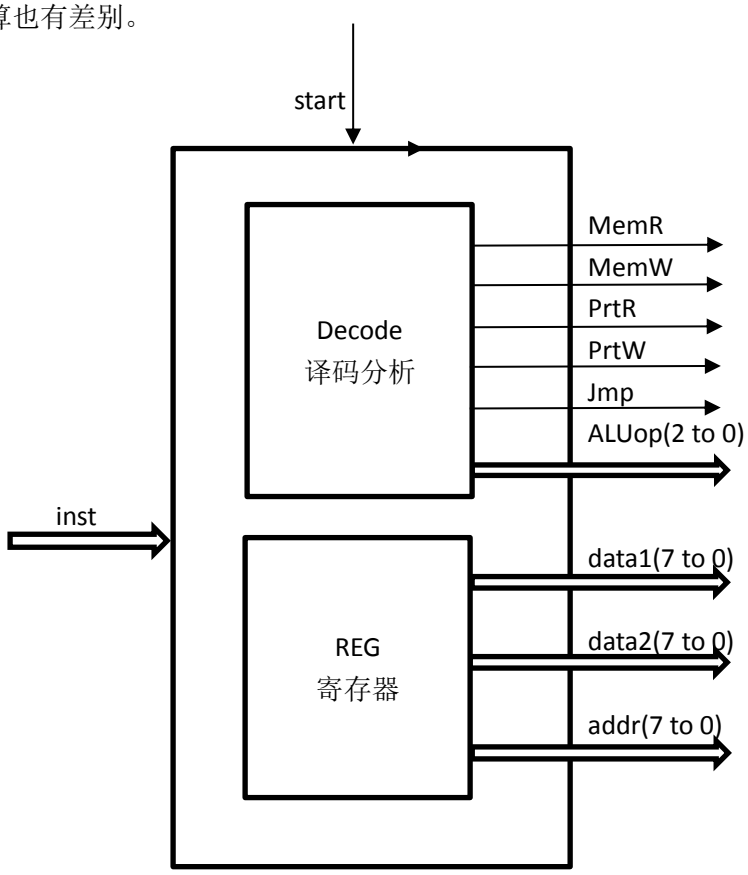
信号名	位数	方向	来源/去向	意义
Start	1	I	节拍发生模块	本模块开始工作的信号
pc_i	16	I	访存及访端口模块	当前指令的 PC
Inst	16	O	访存模块	相关访存信号
Nmreq	1	O		
nrd	1	O		
nwr	1	O		
Nbhe	1	O		
Nble	1	O		
ABUS	16	O		地址总线
DBUS	16	I/O		数据总线

4. 译码模块

根据指令 op 部分，决定后续三个阶段的操作（ALU 的运算、存储器/端口的读写、寄存器的回写），

以及是否跳转。
同时分析指令并取得寄存器中的操作数，完成地址的扩展。

取操作数、地址拓展对所有指令都执行相同的操作，但不同指令译码后传给后续阶段的信号不同，进行的运算也有差别。

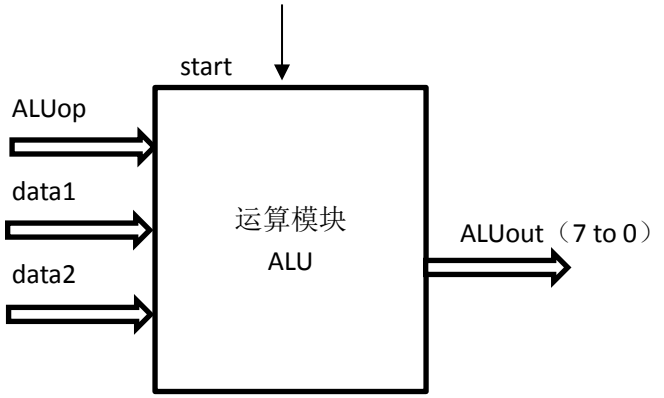


data2 根据指令类型不同，可能是第二个寄存器所代表的数据或指令后八位

信号名	位数	方向	来源/去向	意义
Start	1	I	节拍发生模块	本模块起始信号
Inst	16	I	取指模块	指令
MemW	1	O	访存及访端口模块	高电平写主存
MemR	1	O	访存及访端口模块	高电平读主存
PrtW	1	O	访存及访端口模块	高电平写端口
PrtR	1	O	访存及访端口模块	高电平读端口
Jmp	1	O	访存及访端口模块	高电平跳转
WB	1	O	回写模块	高电平在回写模块回写
ALUop	3	O	运算模块	ALU 运算类型码
data1	8	O	运算模块	操作数
data2	8	O	运算模块	操作数
Addr	16	O	访存及访端口模块	扩展地址

5. 运算模块

根据运算类型码以及操作数完成运算

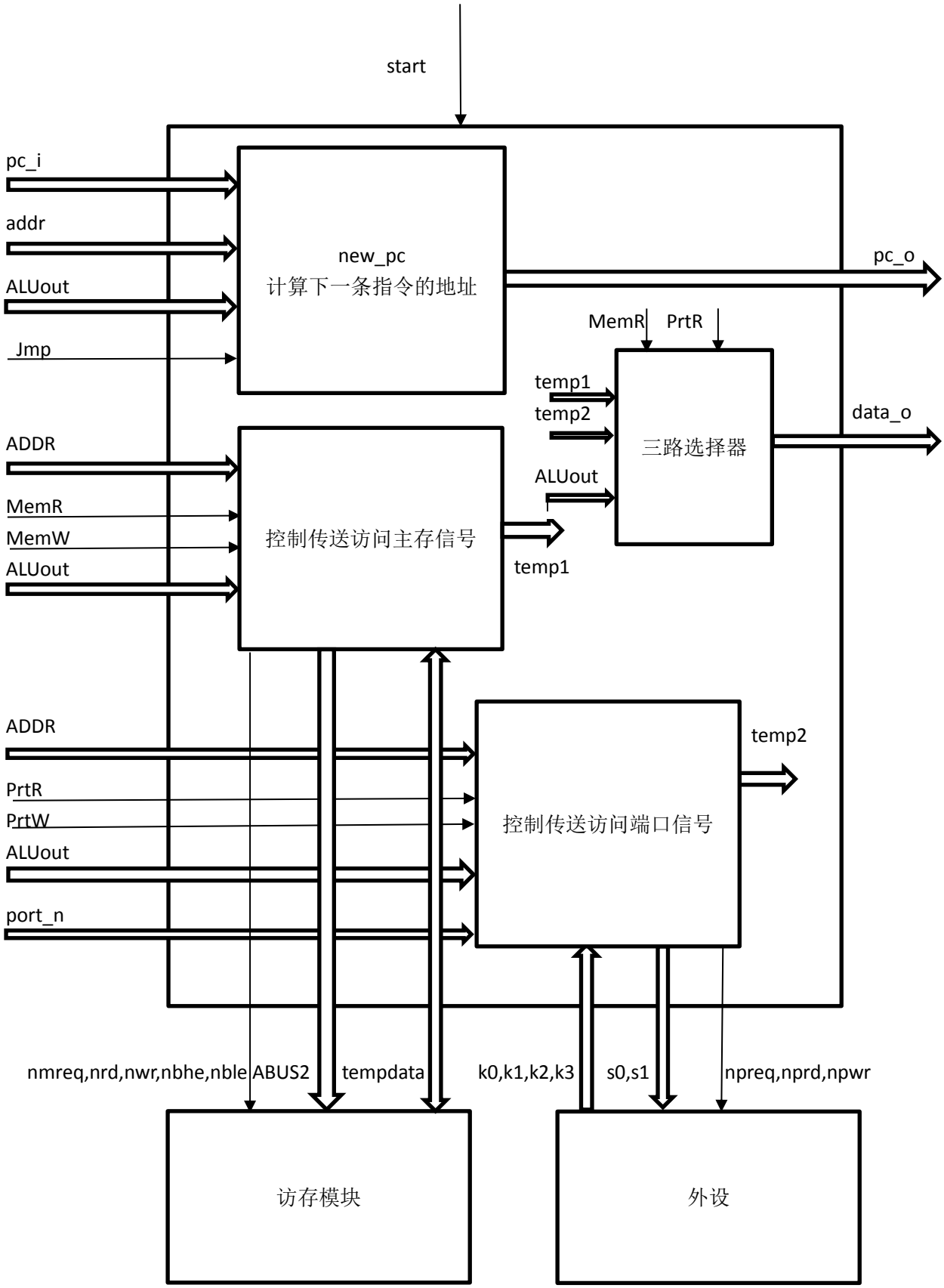


信号名	位数	方向	来源/去向	意义
Start	1	I	节拍发生模块	本模块起始控制信号
ALUop	1	I	译码模块	ALU 运算类型码
data1	8	I	译码模块	操作数
data2	8	I	译码模块	操作数
ALUout	8	O	访存及访端口模块	运算结果

6. 访存及访端口模块

根据译码结果得到的判断信号决定是否读写端口或主存，进行操作。
访存指令向访存模块发送正确的访存信号。
访端口指令向端口发送正确的访问端口信号。
运算类指令直接将 ALU 运算结果传给回写模块。

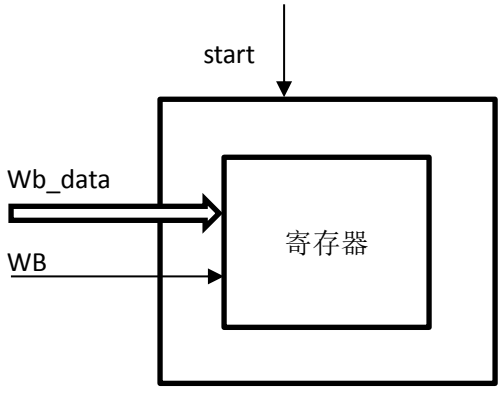
根据 ALU 运算结果和译码模块跳转的判断，决定 PC 新值是 PC+1/R7//X。


start
pc_i
addr
ALUout
Jmp
new_pc
计算下一条指令的地址
pc_o
MemR
PrtR
temp1
temp2
三路选择器
data_o
ADDR
MemR
MemW
ALUout
控制传送访问主存信号
ALUout
temp1
ADDR
PrtR
PrtW
ALUout
port_n
控制传送访问端口信号
temp2
nmreq,nrd,nwr,nbhe,nble
ABUS2
tempdata
k0,k1,k2,k3
s0,s1
npreq,nprd,npwr
访存模块
外设

信号名	位数	方向	来源/去向	意义
start	1	I	节拍发生模块	本模块起始信号
ALUout	8	I	运算模块	ALU 运算结果
port_n	2	I	取指模块（指令部分）	端口号
Addr	16	I	译码模块	扩展地址
MemW	1	I	译码模块	写主存
MemR	1	I	译码模块	读主存
PrtW	1	I	译码模块	写端口
PrtR	1	I	译码模块	读端口
Jmp	1	I	译码模块	跳转
pc_i	16	I	取指模块	当前 PC
pc_o	16	O	取指模块	下一 PC
data_o	8	O	回写模块	回写用的数据
ABUS2	16	O	访存模块	访存信号
tempdata	8	I/O	访存模块	
nmreq	1	O	访存模块	
nrd	1	O	访存模块	
nwr	1	O	访存模块	
nbhe	1	O	访存模块	
nble	1	O	访存模块	
k0, k1, k2, k3	8	I	外设端口	访外设模块
s0, s1	8	O	外设端口	
nPREQ	1	O	外设端口	
nPRD	1	O	外设端口	
nPWR	1	O	外设端口	

7. 回写模块

根据译码模块提供的是否回写的判断信号决定是否是否回写寄存器。

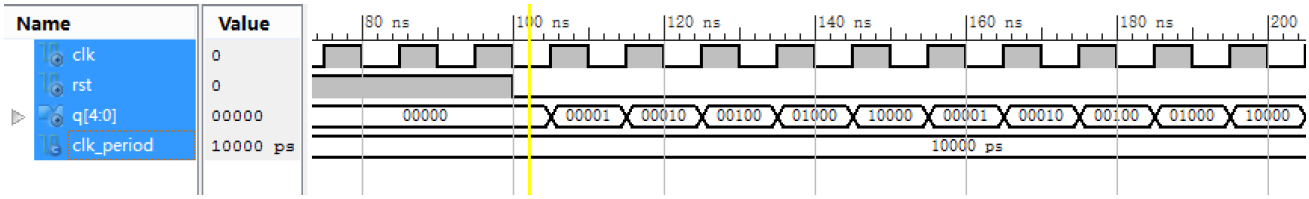


信号名	位数	方向	来源/去向	意义
start	1	I	节拍发生模块	本模块起始信号
Wb_data	8	I	访存及访端口模块	写回寄存器的数据
WB	1	I	译码模块	高电平写回寄存器

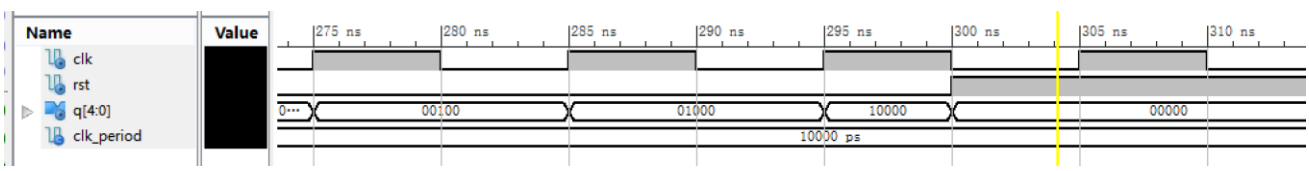
四．模块具体设计、代码及系统仿真波形

1. 节拍发生模块

```
begin

clku: bufgp port map(clk,clkgp);

flag <= '1' when temp = "00000" else '0';

process(clkgp,rst,flag)
begin
if rst = '1' then temp <= "00000";
elsif clkgp'event and clkgp = '1' then
      if flag = '1' then temp <= "00001";
      else temp <= temp(3 downto 0) & temp(4);
      end if;
      end if;
end process;

q <= temp;

end Behavioral;
```



（**rst** 无效后，节拍发生器输出节拍）



（**rst** 生效后，节拍发生全部置零）

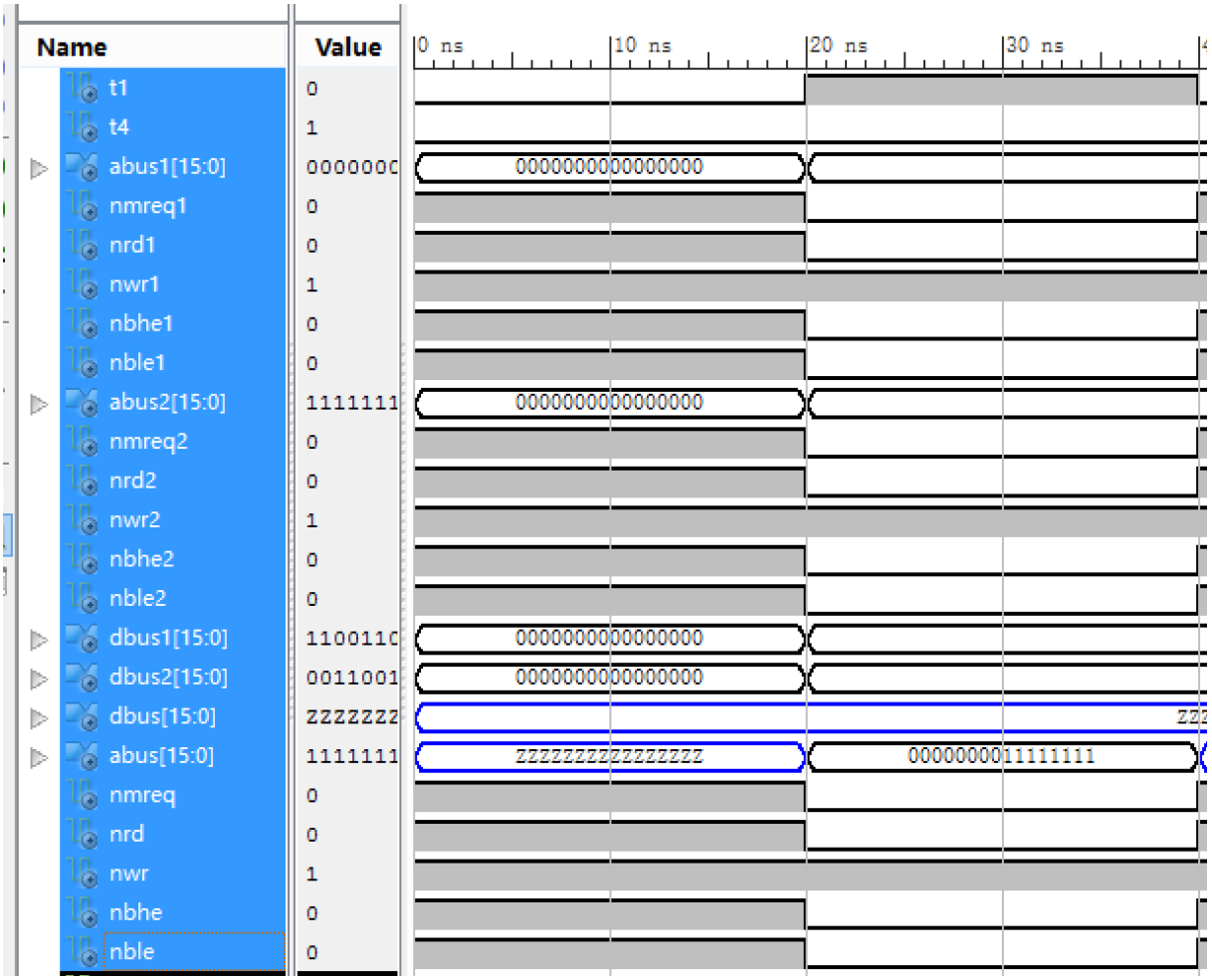
测试：

```
stim_proc: process
begin
    rst <= '1';
    wait for clk_period*10;
       rst <= '0';
       wait for clk_period*20;
       rst <= '1';
    wait;
end process;
```

2.访存模块

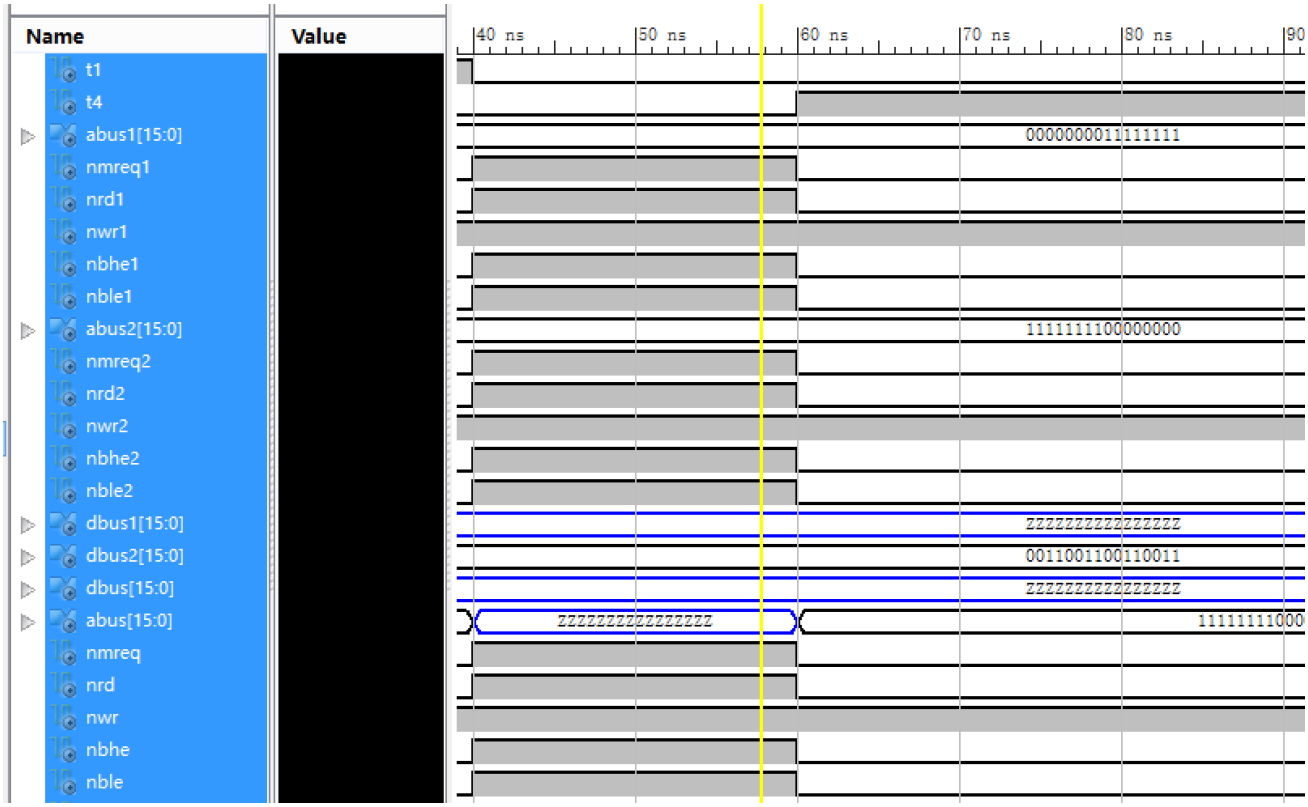
```
architecture Behavioral of Mctrl is

begin

ABUS    <= ABUS1      when t1 = '1'      else ABUS2      when t4 = '1' else (others =>'Z');
DBUS    <= DBUS2      when t4 = '1'      and    nWR2 = '0'    else (others =>'Z');

nMREQ <= nMREQ1    when t1 = '1'      else nMREQ2    when t4 = '1' else    '1';
nRD      <= nRD1        when t1 = '1'      else nRD2        when t4 = '1' else    '1';
nWR      <= nWR1        when t1 = '1'      else nWR2        when t4 = '1' else    '1';
nBHE    <= nBHE1      when t1 = '1'      else nBHE2      when t4 = '1' else    '1';
nBLE    <= nBLE1      when t1 = '1'      else nBLE2      when t4 = '1' else    '1';

DBUS1 <= DBUS ;--when t1 = '1' else (others=>'Z');
DBUS2 <= DBUS WHEN t4 = '1' else (others=>'Z');

end Behavioral;
```

主要使用锁存器结构，对所有指令，均为第一周期根据取指模块的访存数据对主存储器进行操作，第四周期根据访存及访端口模块的访存数据对主存储器进行操作。



上图对应测试：

```
wait for clk_period*2;    --由初始化状态  模拟进入取指状态
        t1 <= '1';
        t4 <= '0';
        ABUS1 <= "0000000011111111";    --由于是取指阶段，取指阶段提供的信号生效
        DBUS1 <= "ZZZZZZZZZZZZZZZZ";
        nMREQ1 <= '0';
        nRD1 <= '0';
        nWR1 <= '1';
        nBHE1 <= '0';
        nBLE1 <= '0';
        ABUS2 <= "1111111100000000";    --访存及访端口模块的信号不生效
        DBUS2 <= "0011001100110011";
        nMREQ2 <= '0';
        nRD2 <= '0';
        nWR2 <= '1';
        nBHE2 <= '0';
        nBLE2 <= '0';
        DBUS <= "ZZZZZZZZZZZZZZZZ";
```



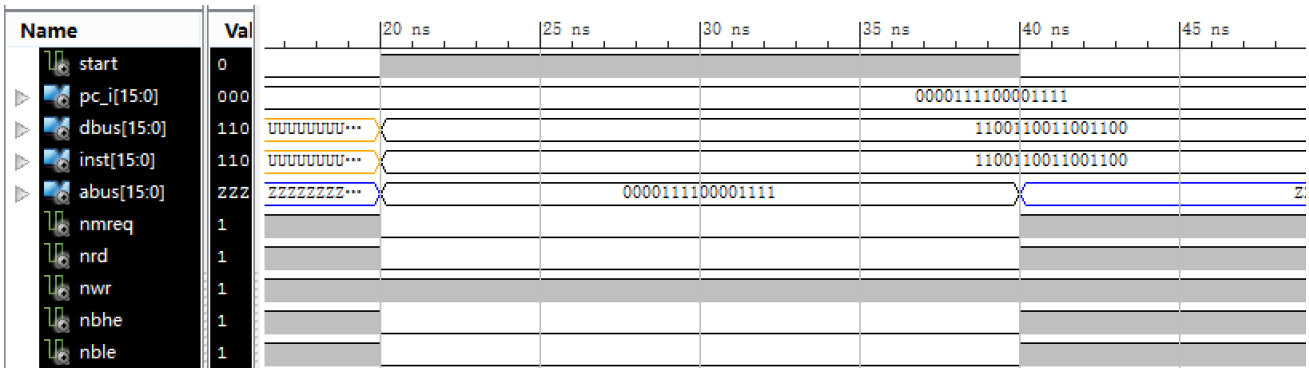

```
wait for clk_period*2;
    t1 <= '0';
    t4 <= '1';            --模拟进入访存及访端口模块
    ABUS1 <= "0000000011111111"; --取指模块的访存信号不生效
    DBUS1 <= "ZZZZZZZZZZZZZZZZ";
    nMREQ1 <= '0';
    nRD1 <= '0';
    nWR1 <= '1';
    nBHE1 <= '0';
    nBLE1 <= '0';
    ABUS2 <= "1111111100000000";
    DBUS2 <= "0011001100110011";
    nMREQ2 <= '0';
    nRD2 <= '0';
    nWR2 <= '1';
    nBHE2 <= '0';
    nBLE2 <= '0';
    DBUS <= "ZZZZZZZZZZZZZZZZ";
```

3.取指模块

根据 PC 指向访存模块传送信号取指令。

```
architecture Behavioral of IFstage is
```

```
    signal inst_t: std_logic_vector(15 downto 0);
    begin
    ABUS     <= pc_i when start = '1' else "ZZZZZZZZZZZZZZZZ";
    nMREQ    <= '0' when start = '1' else '1';
    nRD      <= '0' when start = '1' else '1';
    nWR      <= '1' ;
    nBHE     <= '0' when start = '1' else '1';
    nBLE     <= '0' when start = '1' else '1';
    DBUS     <= "ZZZZZZZZZZZZZZZZ" when start = '1' else DBUS;
    inst_t <= DBUS when start = '1' else inst_t;
    inst <= inst_t;
end Behavioral;
```



测试：

```
    start <= '0';                    --节拍起始信号不生效
    pc_i <= "0000111100001111";
    DBUS <= "1100110011001100";
wait for clk_period*2;
    start <= '1';                    --开始取指
wait for clk_period*2;
    start <= '0';
```

4.译码模块(/回写模块)

根据指令指导后续操作进行。(根据相关回写信息回写存储器。)

指令	op	ALUop	对应 ALU 运算	MemR (读主存)	MemW (写主存)	PrtR (读端口)	PrtW (写端口)	Jmp (跳转)	Wb (回写)
ADD	00000	001	data1+data2	0	0	0	0	0	1
SUB	00001	010	data1-data2	0	0	0	0	0	1
MOV	00010	011	data2	0	0	0	0	0	1
MVI	00011	011	data2(7 to 0)	0	0	0	0	0	1
STA	00100	100	data1	0	1	0	0	0	0
LDA	00101	000(无关)	无关	1	0	0	0	0	1
JZ	00110	100	data1	0	0	0	0	1	0

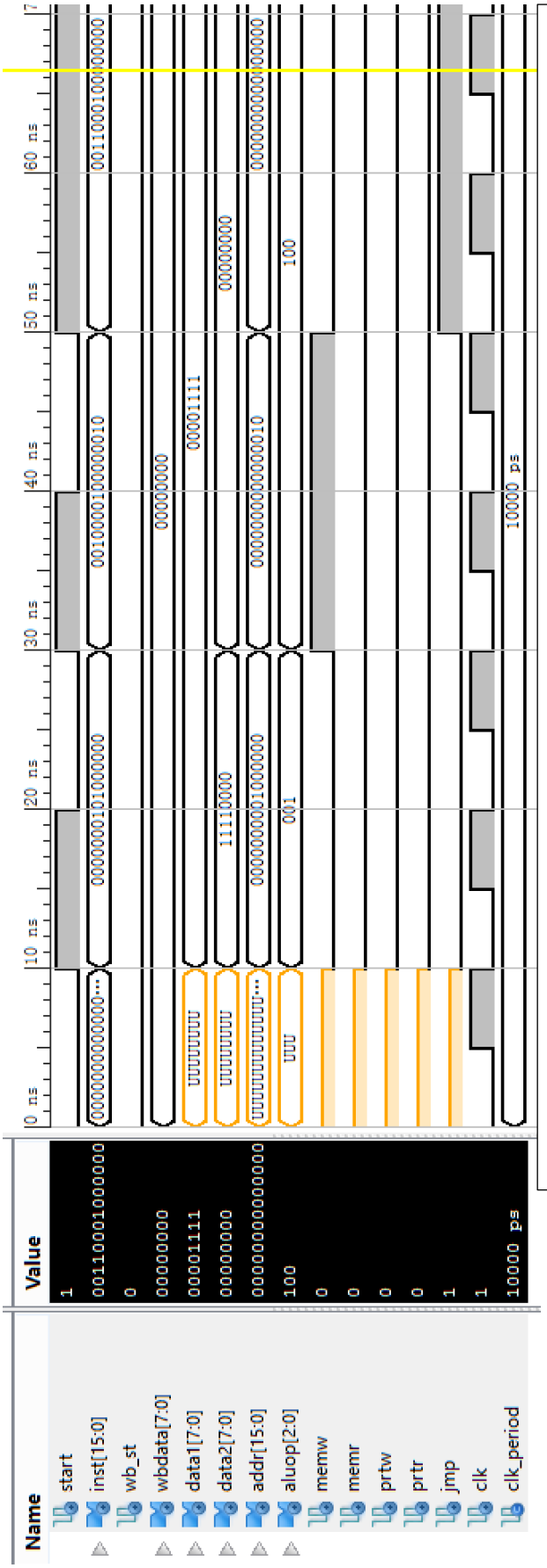
指令	op	ALUop	对应 ALU 运算	MemR (读主存)	MemW (写主存)	PrtR (读端口)	PrtW (写端口)	Jmp (跳转)	Wb (回写)
JMP	00111	100	data1(0 寄存器)	0	0	0	0	1	0
IN	01000	000(无关)	无关	0	0	1	0	0	1
OUT	01001	100	data1	0	0	0	1	0	0

data1 为 指令 10~8 所代表的寄存器内容
data2 为 指令 10~8 所代表的寄存器内容 或 指令 7~0 所代表的立即数

JZ 要求 寄存器内容为零发生跳转，即 10~8 位寄存器内容作为 ALU 输出结果
JMP 要求 直接发生跳转，设置使 JMP 指令同样依靠 10~8 位所代表寄存器内容作为 ALU 输出结果
JMP 指令 10~8 位为“000”，所以 0 寄存器的内容必须为 0，不应该进行修改。

```
architecture Behavioral of IDstage is
type r is array (7 downto 0) of std_logic_vector(7 downto 0);
signal reg: r :=("00000000","00000000","00000000","00000000","00000000","00000000","00000000",
"00000000");
signal rs,rt: std_logic_vector(2 downto 0);
signal opcode : std_logic_vector(4 downto 0);
signal imm_8    : std_logic_vector(7 downto 0);
signal Wb :std_logic;
signal ALUop_t: std_logic_vector(2 downto 0);
signal mw,mr,pw,pr,j: std_logic;

begin
--拆分指令
opcode <= (inst(15),inst(14),inst(13),inst(12),inst(11)) when start = '1' else opcode;
rs <= (inst(10),inst(9),inst(8)) when start = '1' else rs;
rt <= (inst(7),inst(6),inst(5)) when start = '1' else rt;
imm_8 <= (inst(7),inst(6),inst(5),inst(4),inst(3),inst(2),inst(1),inst(0)) when start = '1' else imm_8;
--写回寄存器 当写回周期到来且需要写回时
reg(0) <= WbData when Wb = '1' and Wb_st = '1' and rs = "000" else reg(0);
reg(1) <= WbData when Wb = '1' and Wb_st = '1' and rs = "001" else reg(1);
reg(2) <= WbData when Wb = '1' and Wb_st = '1' and rs = "010" else reg(2);
reg(3) <= WbData when Wb = '1' and Wb_st = '1' and rs = "011" else reg(3);
reg(4) <= WbData when Wb = '1' and Wb_st = '1' and rs = "100" else reg(4);
reg(5) <= WbData when Wb = '1' and Wb_st = '1' and rs = "101" else reg(5);
reg(6) <= WbData when Wb = '1' and Wb_st = '1' and rs = "110" else reg(6);
reg(7) <= WbData when Wb = '1' and Wb_st = '1' and rs = "111" else reg(7);

with (opcode) select
ALUop_t <=
    "001" when "00000", --add
    "010" when "00001", --sub
```

```
        "011" when "00010" | "00011", --mov,mvi
        "100" when "00100" | "00110" | "00111" | "01001", --sta,jz,jmp,out
        "000" when others;
with (opcode) select
Wb <=
        '1' when "00000"|"00001"|"00010"|"00011"|"00101"|"01000",
        '0' when others;
with (opcode) select
mw <=
        '1' when "00100",
        '0' when others;
with (opcode) select
mr <=
        '1' when "00101",
        '0' when others;
with (opcode) select
pr <=
        '1' when "01000",
        '0' when others;
with (opcode) select
pw <=
        '1' when "01001",
        '0' when others;
with (opcode) select
j <=
        '1' when "00110"|"00111",
        '0' when others;

process(start,imm_8,rs,rt,reg,ALUop_t,mw,mr,pw,pr,j)
begin
if start = '1' then
        data1 <= reg(conv_integer(rs));
        if opcode = "00011" then data2 <= imm_8;
        else data2 <= reg(conv_integer(rt));
        end if;
        ALUop <= ALUop_t;
        addr    <=(reg(7)&imm_8);
        MemW    <= mw;    MemR    <= mr;
        PrtW        <= pw;    PrtR    <= pr;
        Jmp      <= j;
else null;
end if;
end process;
end Behavioral;
```



测试：

（存储器默认值设为 REG(1)=”00001111”; REG(2)=”11110000”;）

start <= '0';

inst <= "0000000000000000";

Wb_st <= '0';

Wbdata <= "00000000";

wait for clk_period*1; --加法指令：取 reg1，reg2 作为操作数，地址扩展和立即数正常计算但不使用

start <= '1';

inst <= "0000000101000000";

Wb_st <= '0';

Wbdata <= "00000000";

wait for clk_period*1;

start <= '0';

wait for clk_period*1; --STA 指令：取 reg1，地址扩展使用，reg2 和立即数正常计算但不使用

start <= '1';

inst <= "0010000100000010";

Wb_st <= '0';

Wbdata <= "00000000";

wait for clk_period*1;

start <= '0';

wait for clk_period*1; --JZ 指令：取 reg1，地址扩展使用，reg2 和立即数正常计算但不使用

start <= '1';

inst <= "0011000100000000";

Wb_st <= '0';

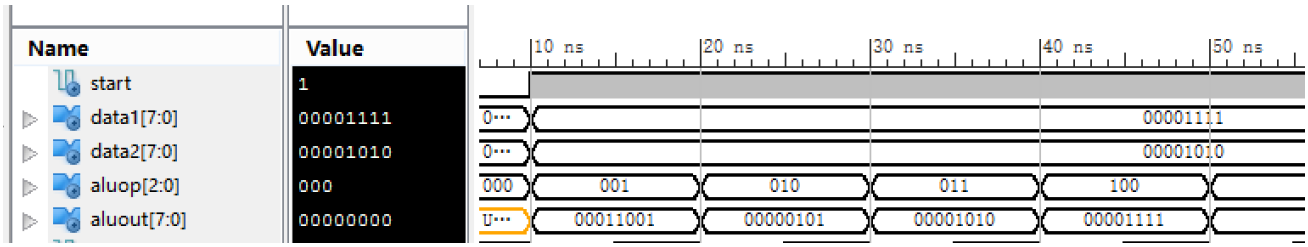
Wbdata <= "00000000";

5.运算模块

根据 ALUop 对操作数进行计算

指令	ALUop	对应 ALU 运算
ADD	001	data1+data2
SUB	010	data1-data2
MOV	011	data2
MVI	011	data2(7 to 0)
STA	100	data1
LDA	000(无关)	无关“00000000”
JZ	100	data1
JMP	100	data1(0 寄存器)
IN	000(无关)	无关“00000000”
OUT	100	data1

```
architecture Behavioral of ALU is

signal result : std_logic_vector(7 downto 0);

begin

with (ALUop) select
result <=
      conv_std_logic_vector(conv_integer(data1)+conv_integer(data2),8) when "001",
      conv_std_logic_vector(conv_integer(data1)-conv_integer(data2),8) when "010",
      data2 when "011",
      data1 when "100",
      "00000000" when others;--000

process(start,result)
begin
if start = '1' then ALUout <= result;
end if;
end process;

end Behavioral;
```



上图测试代码：

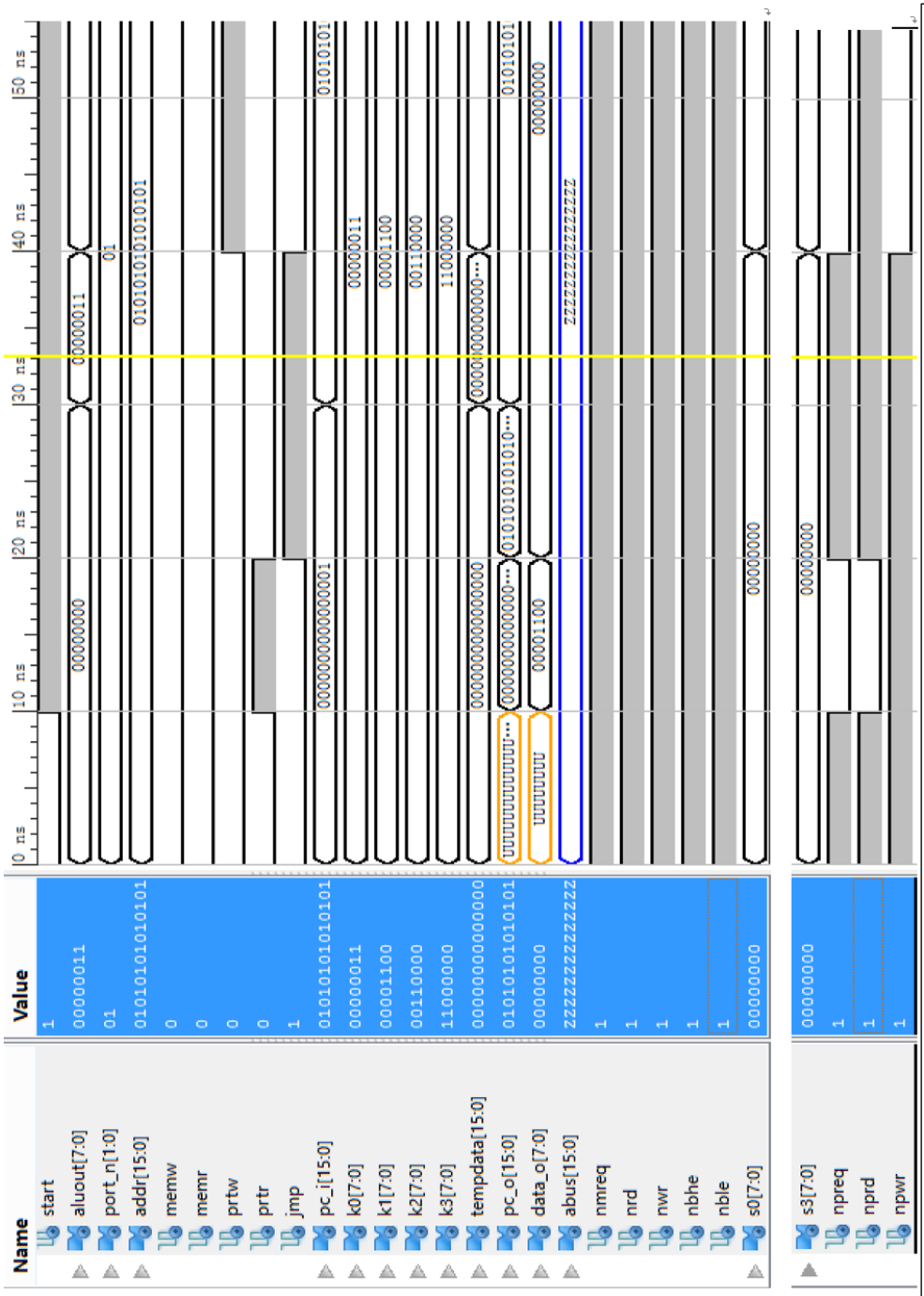
```
start <= '0';
data1 <= "00000000";
data2 <= "00000000";
ALUop <= "000";
wait for clk_period*1;      --op 001
start <= '1';
data1 <= "00001111";
data2 <= "00001010";
ALUop <= "001";
wait for clk_period*1;    --op010
ALUop <= "010";
wait for clk_period*1;    --op011
ALUop <= "011";
wait for clk_period*1;    --op100
ALUop <= "100";
wait for clk_period*1;    --op000
ALUop <= "000";
```

6.访存及访端口模块

根据译码产生的信号决定对主存/端口的操作，以及下一条指令的地址。

指令	MemR (读主存)	MemW (写主存)	PrtR (读端口)	PrtW (写端口)	Jmp (跳转)	PC 新值
ADD	0	0	0	0	0	PC+1
SUB	0	0	0	0	0	PC+1
MOV	0	0	0	0	0	PC+1
MVI	0	0	0	0	0	PC+1
STA	0	1	0	0	0	PC+1
LDA	1	0	0	0	0	PC+1
JZ	0	0	0	0	1	R7//X 或 PC+1（ALUout 和 Jmp 决定）
JMP	0	0	0	0	1	R7//X
IN	0	0	1	0	0	PC+1
OUT	0	0	0	1	0	PC+1

```
architecture Behavioral of MemPort is
signal pc_t: std_logic_vector(15 downto 0);
signal tempdata1 : std_logic_vector(7 downto 0);
begin

with port_n select
tempdata1<=    k0 when "00",
                        k1 when "01",
                        k2 when "10",
                        k3 when "11",
                        "ZZZZZZZZ" when others;
nPREQ <= '0' when (PrtW = '1' or PrtR = '1') and start = '1' else '1';
nPRD    <= '0' when start = '1' and PrtR = '1' else '1';
nPWR    <= '0' when start = '1' and PrtW = '1' else '1';
s3 <= ALUout when start = '1' and PrtW = '1' else "00000000";
s0 <= ALUout when start = '1' and PrtW = '1' else "00000000";

--计算新指令地址
pc_t <= addr when ALUout = "00000000" and Jmp = '1' else conv_std_logic_vector(conv_integer(pc_i)+1,16);
nMREQ <= '0' when MemW = '1' or MemR = '1' else '1';
nRD     <= '0' when MemR = '1' else '1';
nWR     <= '0' when MemW = '1' else '1';
nBHE    <= '0' when MemW = '1' or MemR = '1' else '1';
nBLE    <= '0' when MemW = '1' or MemR = '1' else '1';
tempdata <= "00000000"&ALUout when MemR = '0' else (others => 'Z');
ABUS <= addr when MemW = '1' or MemR = '1' else (others => 'Z');

process(start,pc_t,tempdata1,tempdata,MemW,MemR,PrtR,PrtW,ALUout)
begin
if start = '1' then
      pc_o <= pc_t;
      if MemW = '1' or PrtW = '1' or Jmp = '1' then data_o <="00000000";
      elsif MemR = '1' then data_o <= tempdata(7 downto 0);
      elsif PrtR = '1' then data_o <= tempdata1;
      else data_o <= ALUout;
      end if;
end if;
end process;
end Behavioral;
```



测试：

```
start <= '0';
ALUout <= "00000000";
port_n <= "01" ;
addr <="0101010101010101";
MemW <='0';
MemR <='0';
PrtW <='0';
PrtR <='0';
Jmp    <='0';
pc_i <="0000000000000001";

tempdata<="ZZZZZZZZZZZZZZZZ";
k0 <= "00000011";
k1 <= "00001100";
k2 <= "00110000";
k3 <= "11000000";

wait for clk_period*1;
start <= '1';
PrtR    <= '1';

wait for clk_period*1;
start <= '1';
PrtR    <= '0';
Jmp <= '1';

wait for clk_period*1;
start <= '1';
ALUout <= "00000011";
pc_i <= pc_o;
Jmp <= '1';

wait for clk_period*1;
start <= '1';
ALUout <= "00001111";
Jmp <= '0';
PrtW <='1';
```

7.回写模块

根据译码阶段是否回写的判断访问寄存器

代码同译码模块。
Wb_start 代表回写阶段的开始，当本条指令 Wb 信号为 1 时，用 Wb_data 写回预先保存好的寄存器

8.整合代码

```
signal q: std_logic_vector(4 downto 0);
signal ABUS1,DBUS1: std_logic_vector(15 downto 0);
signal ABUS2,DBUS2: std_logic_vector(15 downto 0);
signal nMREQ1,nRD1,nWR1,nBHE1,nBLE1,nMREQ2,nRD2,nWR2,nBHE2,nBLE2: std_logic;
signal pc_t: std_logic_vector(15 downto 0) ;
signal port_n :std_logic_vector(1 downto 0);
signal inst :std_logic_vector(15 downto 0);
signal data1:std_logic_vector(7 downto 0);
signal data2: std_logic_vector(7 downto 0);
signal addr : std_logic_vector(15 downto 0);
signal ALUop: std_logic_vector(2 downto 0);
signal MemW, MemR, PrtW, PrtR, Jmp : std_logic;
signal ALUout: std_logic_vector(7 downto 0);
signal pc_o    : std_logic_vector(15 downto 0);
signal data_o: std_logic_vector(7 downto 0);
signal s5temp,s4temp: std_logic_vector(7 downto 0);

begin
port_n <= inst(7 downto 6) when q(1) = '1' else port_n;
mybeat : Beat        port map(clk,rst,q);
myIF:        IFstage port map(q(0),pc_t,inst,ABUS1,DBUS1,nMREQ1,nRD1,nWR1,nBHE1,nBLE1);

myID:      IDstage port map(q(1),inst,data1,data2,addr,ALUop,q(4),data_o,MemW,MemR,PrtW,PrtR,Jmp);

myALU:   ALU    port map(q(2),data1,data2,ALUop,ALUout);

myMP:    MemPort port map(q(3),ALUout,port_n,addr,MemW,MemR,PrtW,PrtR,Jmp,pc_t,pc_o,data_o,ABUS2,
           DBUS2,nMREQ2,nRD2,nWR2,nBHE2,nBLE2,k0,k1,k2,k3,s0,s3,nPREQ,nPRD,nPWR);

myMem :    Mctrl      port map(q(0),q(3),ABUS1,DBUS1,nMREQ1,nRD1,nWR1,nBHE1,nBLE1,ABUS2,DBUS2,
           MREQ2,nRD2,nWR2,nBHE2,nBLE2,ABUS,DBUS,nMREQ,nRD,nWR,nBHE,nBLE);

pc_t <= "0000000000000000" when rst = '1' else pc_o when q(4) = '1' else pc_t;
s5temp <= "00000000" when q = "00000" else pc_t(15 downto 8) when q(0) = '1' else s5temp;
s4temp <= "00000000" when q = "00000" else pc_t(7 downto 0) when q(0) = '1' else s4temp;
s5 <= s5temp;    s4 <= s4temp;
B <= q ;
s2<= inst(15 DOWNTO 8) ; s1<= inst(7 DOWNTO 0);
```

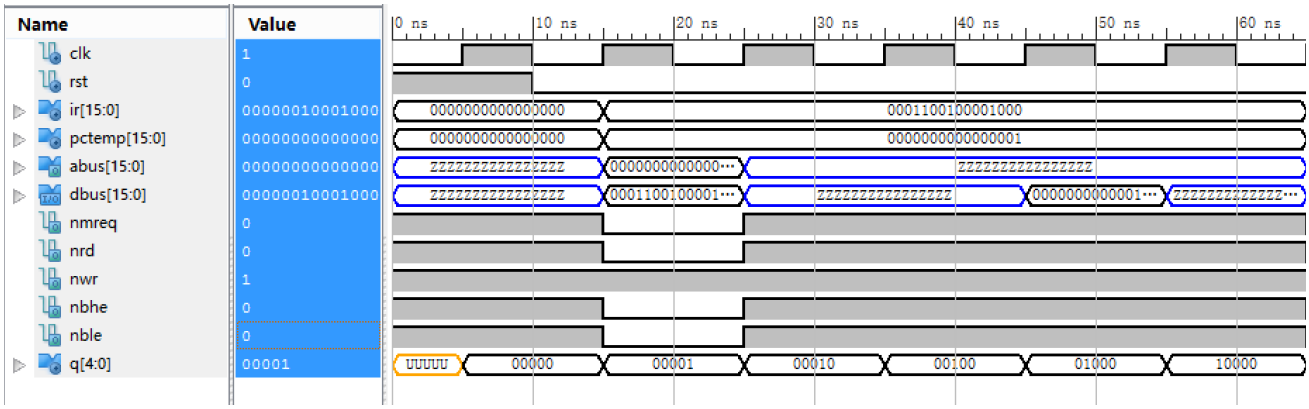
end Behavioral;

配合 Memory 做测试

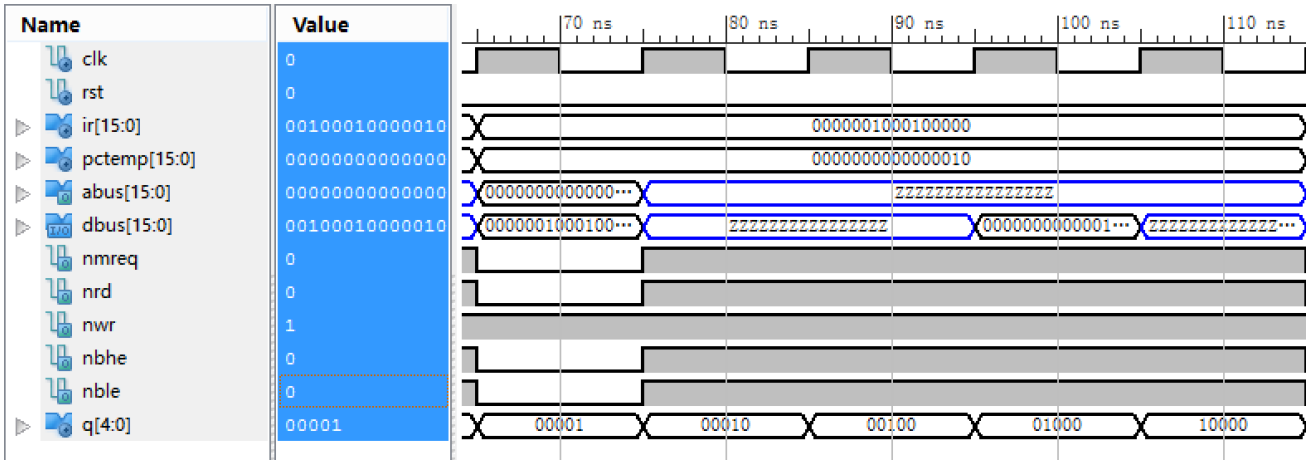
--0 空指令 --1 mvi r1,8 --2 add r2,r1

--3 sta r2,10 --4 lda r3,10 --5 jmp r3,8

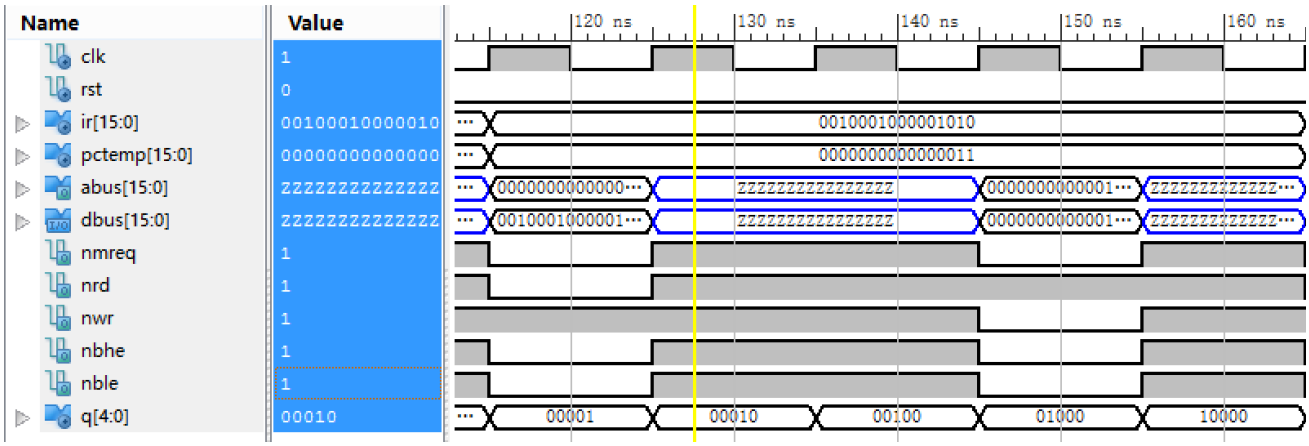
--6 sub r3,r1 --7 jz r3,1



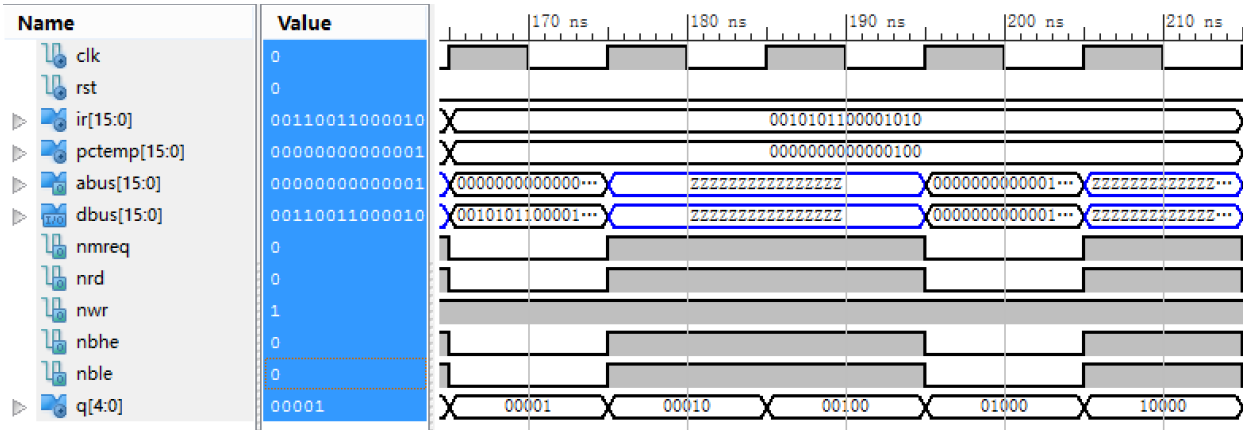
（pc=0 空指令 pc=1 mvi r1,8）空指令 置数



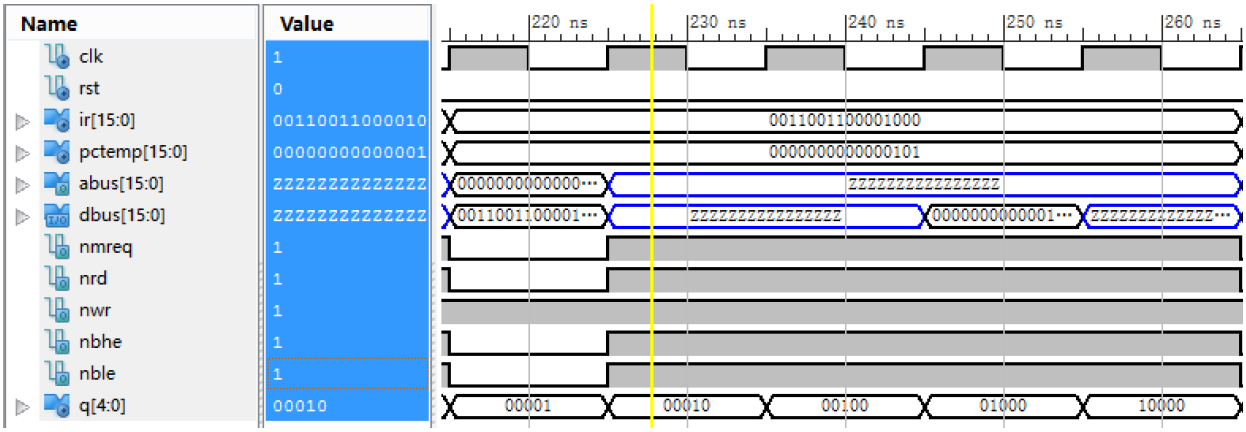
（pc=2 add r2,r1）加法



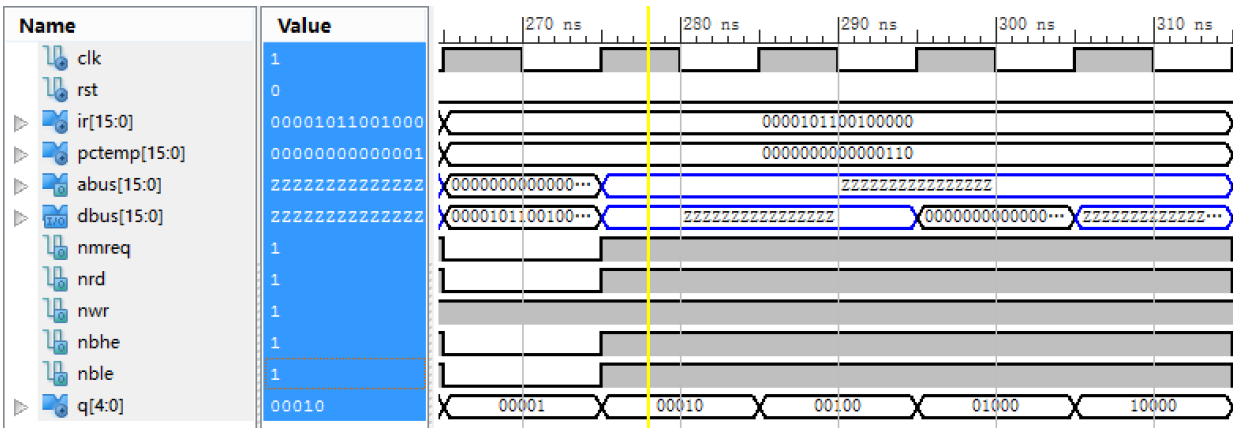
(pc=3 sta r2,10)存数



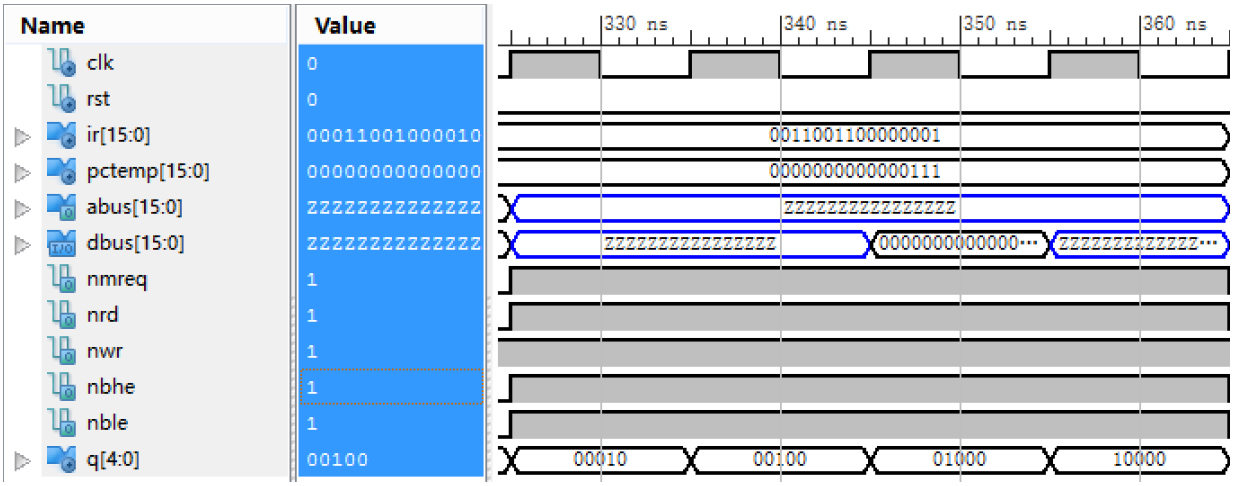
(pc = 4 lda r3,10)取数



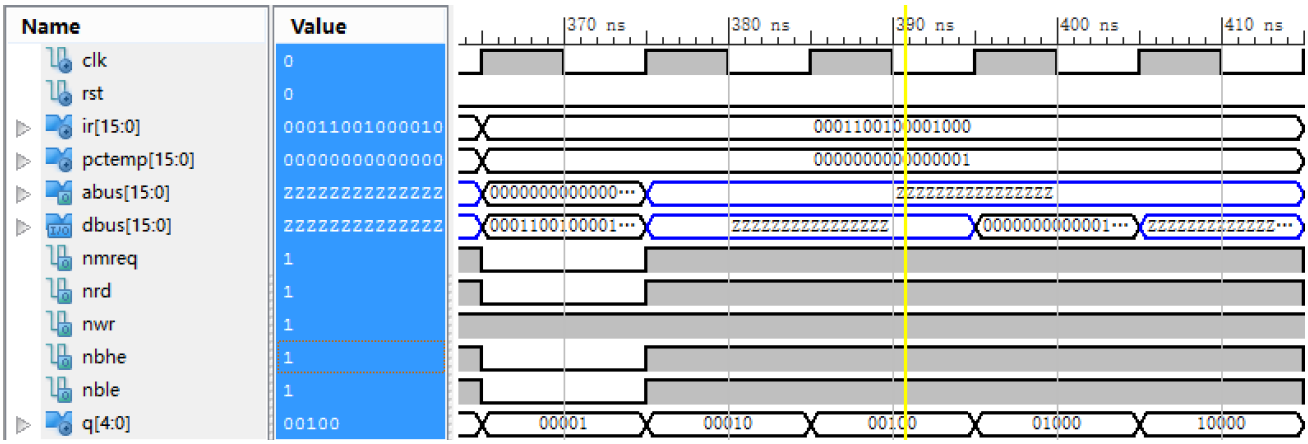
(pc=5 jz r3,8)不发生跳转



(pc=6 sub r3,r1)



（pc=7 jz　r3,1）发生跳转



回到 PC=1

五．管脚定义

指令、地址及端口输出一并输出到数码管，输入由拨码开关决定。
节拍发生器共有五个灯，第一个亮起时应该有指令，第四个亮起时可能有端口输出。
地址在每条指令执行完毕后变更。
端口输入应该在该条指令执行前拨好。
RST 回到 0 时，按动时钟信号开始工作。

```
NET "nMREQ" LOC = "p159";
NET "nWR" LOC = "p171";
NET "nRD" LOC = "p188";
NET "nBHE" LOC = "p189";
NET "nBLE" LOC = "p191";
NET "rst" LOC = "p33";
NET "clk" LOC = "p92";
```

```
NET "ABUS<0>" LOC = "p153";
NET "ABUS<1>" LOC = "p154";
NET "ABUS<2>" LOC = "p155";
NET "ABUS<3>" LOC = "p156";
NET "ABUS<4>" LOC = "p157";
NET "ABUS<5>" LOC = "p173";
NET "ABUS<6>" LOC = "p174";
NET "ABUS<7>" LOC = "p175";

NET "ABUS<8>" LOC = "p176";
NET "ABUS<9>" LOC = "p186";
NET "ABUS<10>" LOC = "p185";
NET "ABUS<11>" LOC = "p208";
NET "ABUS<12>" LOC = "p207";
NET "ABUS<13>" LOC = "p206";
NET "ABUS<14>" LOC = "p205";
NET "ABUS<15>" LOC = "p187";

NET "DBUS<0>" LOC = "p160";
NET "DBUS<1>" LOC = "p161";
NET "DBUS<2>" LOC = "p162";
NET "DBUS<3>" LOC = "p163";
NET "DBUS<4>" LOC = "p167";
NET "DBUS<5>" LOC = "p168";
NET "DBUS<6>" LOC = "p169";
NET "DBUS<7>" LOC = "p170";

NET "DBUS<8>" LOC = "p202";
NET "DBUS<9>" LOC = "p201";
NET "DBUS<10>" LOC = "p200";
NET "DBUS<11>" LOC = "p199";
NET "DBUS<12>" LOC = "p195";
NET "DBUS<13>" LOC = "p194";
NET "DBUS<14>" LOC = "p193";
NET "DBUS<15>" LOC = "p192";

NET "k0<0>" LOC = "p103";
NET "k0<1>" LOC = "p102";
NET "k0<2>" LOC = "p101";
NET "k0<3>" LOC = "p100";
NET "k0<4>" LOC = "p97";
NET "k0<5>" LOC = "p96";
NET "k0<6>" LOC = "p95";
NET "k0<7>" LOC = "p94";
```

```
NET "k1<0>" LOC = "p87";
NET "k1<1>" LOC = "p86";
NET "k1<2>" LOC = "p85";
NET "k1<3>" LOC = "p84";
NET "k1<4>" LOC = "p82";
NET "k1<5>" LOC = "p81";
NET "k1<6>" LOC = "p80";
NET "k1<7>" LOC = "p79";

NET "k2<0>" LOC = "p73";
NET "k2<1>" LOC = "p72";
NET "k2<2>" LOC = "p71";
NET "k2<3>" LOC = "p70";
NET "k2<4>" LOC = "p66";
NET "k2<5>" LOC = "p65";
NET "k2<6>" LOC = "p64";
NET "k2<7>" LOC = "p63";

NET "k3<0>" LOC = "p47";
NET "k3<1>" LOC = "p48";
NET "k3<2>" LOC = "p49";
NET "k3<3>" LOC = "p50";
NET "k3<4>" LOC = "p53";
NET "k3<5>" LOC = "p54";
NET "k3<6>" LOC = "p55";
NET "k3<7>" LOC = "p56";

NET "s0<0>" LOC = "p223";
NET "s0<1>" LOC = "p222";
NET "s0<2>" LOC = "p221";
NET "s0<3>" LOC = "p220";
NET "s0<4>" LOC = "p218";
NET "s0<5>" LOC = "p217";
NET "s0<6>" LOC = "p216";
NET "s0<7>" LOC = "p215";

NET "s1<0>" LOC = "p235";
NET "s1<1>" LOC = "p234";
NET "s1<2>" LOC = "p232";
NET "s1<3>" LOC = "p231";
NET "s1<4>" LOC = "p230";
NET "s1<5>" LOC = "p229";
NET "s1<6>" LOC = "p228";
NET "s1<7>" LOC = "p224";
```

```
NET "s2<0>" LOC = "p7";
NET "s2<1>" LOC = "p6";
NET "s2<2>" LOC = "p5";
NET "s2<3>" LOC = "p4";
NET "s2<4>" LOC = "p3";
NET "s2<5>" LOC = "p238";
NET "s2<6>" LOC = "p237";
NET "s2<7>" LOC = "p236";

NET "s3<0>" LOC = "p19";
NET "s3<1>" LOC = "p18";
NET "s3<2>" LOC = "p17";
NET "s3<3>" LOC = "p13";
NET "s3<4>" LOC = "p12";
NET "s3<5>" LOC = "p11";
NET "s3<6>" LOC = "p10";
NET "s3<7>" LOC = "p9";

NET "s4<0>" LOC = "p28";
NET "s4<1>" LOC = "p27";
NET "s4<2>" LOC = "p26";
NET "s4<3>" LOC = "p25";
NET "s4<4>" LOC = "p24";
NET "s4<5>" LOC = "p23";
NET "s4<6>" LOC = "p21";
NET "s4<7>" LOC = "p20";

NET "s5<0>" LOC = "p74";
NET "s5<1>" LOC = "p68";
NET "s5<2>" LOC = "p67";
NET "s5<3>" LOC = "p57";
NET "s5<4>" LOC = "p52";
NET "s5<5>" LOC = "p46";
NET "s5<6>" LOC = "p42";
NET "s5<7>" LOC = "p31";

NET "nPREQ" LOC = "p110";
NET "nPRD" LOC = "p111";
NET "nPWR" LOC = "p203";

NET "B<0>" LOC = "p125";
NET "B<1>" LOC = "p124";
NET "B<2>" LOC = "p109";
```

NET "B<3>" LOC = "p108";
NET "B<4>" LOC = "p107";

六．处理器功能测试程序

地址	内容	指令	指令(16)	结果
0000000000000000	MVI R2,0	0001101000000000	1A08	R2=0
0000000000000001	MVI R1, 8	0001100100000000	1908	R1=8
0000000000000010	ADD R2, R1	0000001000100000	0220	R2=8
0000000000000011	STA R2, 10	0010001000001010	220A	M(10)=8
0000000000000100	LDA R3, 10	0010101100001010	2B0A	R3=M(10)=8
0000000000000101	JZ R3, 8	0011001100001000	3308	为零则跳转
0000000000000110	SUB R3, R1	0000101100100000	0B20	R3=0
0000000000000111	JZ R3, 1	0011001100000001	3301	为零则跳转
0000000000001000	IN R1, 1	0100000101000000	4140	R1=K1
0000000000001001	OUT R1, 0	0100100100000000	4A00	S0=S3=K1

上载存储器可见 M(10)已经被修改为 0000000000001000

(如果尝试添加指令，则应该删去前面修改 M(10)内容的相关指令)

修改指令时请勿修改 R(0)值，将 R(0)内容保持为 0

下载结果：

地址	指令(16)	结果	判断
0000000000000000	1A08	R2=0	
0000000000000001	1908	R1=8	
0000000000000010	0220	R2=8	
0000000000000011	220A	M(10)=8	
0000000000000100	2B0A	R3=M(10)=8	
0000000000000101	3308	不发生跳转	前面所有运算指令执行正确
0000000000000110	0B20	R3=0	
0000000000000111	3301	发生跳转	减法指令运算成功 跳转指令成功
0000000000000001	1908	R1=8	
0000000000000010	0220	R2=16	
0000000000000011	220A	M(10)=16	
0000000000000100	2B0A	R3=M(10)=16	
0000000000000101	3308	不发生跳转	JZ 判断正确
0000000000000110	0B20	R3=8	
0000000000000111	3301	不发生跳转	
0000000000001000	4140	R1=K1 输入 15	
0000000000001001	4A00	S0=S3=K1 输出 15	成功读入并输出 IN OUT 指令正确

七．设计、调试、波形、下载过程中遇到的问题及解决方法

设计部分：

相关信号不明确，需要在设计时明确 10 条指令在各个阶段所执行的操作的具体内容。

理清各个模块之间的信号传输。

调试部分：

调试不稳定，通过减少代码中的边缘触发和 process 进程的代码量，尽量使用”a <= b when (condition) else a;”类似的锁存器构造完成条件语句。

Process 的敏感信号表应该包含所有的输入信号，否则容易出现错误。

inout 类型的线应该尽可能减少。

波形部分：

波形存在较多 U,X 通过修改给信号赋初值减少 U 的存在，当 X 出现时，是由于对同信号的重复赋值导致。

下载过程中：

取指周期过后，指令并未能正确取出，但根据波形判断数据并未错误传送，而是由于 DBUS 同时传了 16 位正常数和高阻，导致无法正确获取指令，当修改代码使得 DBUS 不被同时赋值为要写回的数据和高阻，对存储器的操作恢复正常。

FPGA 出错可能由于代码本身错误（如 PROCESS 的敏感信号表未包含全部输入信号）导致。如果反复尝试仍然有出错提示应该考虑代码 simulate 过程中的 warning 信息。

八．实验体会

硬件程序的设计，应该严格遵循相应的开发步骤，自顶向下，从系统整体要求出发，将设计内容逐渐细化，最后完成需求。自顶向下的方法可以保持逻辑清楚。

FPGA 芯片功能灵活可扩展，未来将成为硬件发展的趋势。要掌握集成的 PLD/FPGA 开发环境（XILINX 等），了解 FPGA 开发的基本流程：系统划分->代码输入->编译->代码功能仿真->综合->芯片仿真。

更具体的掌握了 VHDL 语言。其模块内部的信号，应该尽可能精简，输入输出信号减少对 inout 类型的使用，掌握如何书写并行、顺序语句、条件控制语句。

掌握了 VHDL 本质上并发的特性。

下载调试过程，需要综合考虑代码以及实验台多方面因素。

熟练了 FPGA 芯片的相关操作。