

**课程设计报告**

名 称： 硬件设计与实践

院 系： 计算机系

班 级：

学 号：

学生姓名： 宣朋羽

指导教师：

设计周数： 2

成 绩：

日期：

**硬件设计与实践**

**任 务 书**

**一、 目的与要求**

1.目的

1.1 培养学生在计算机硬件方面的动手实践能力；

1.2 熟悉VHDL硬件描述语言及开发环境，了解硬件系统开发的基本过程；

1.3 把前期学习阶段的知识和方法系统化，用以解决实际问题，为毕业设计做准备。

2.要求

2.1 运用所学的知识和方法采用最佳的解决问题思路，完成本次实践课程；

2.2 对要完成的内容进行详细的分析和设计，必要时画出指令的流程图；

2.3 编写简洁代码完成实验；

2.4 认真书写设计报告，包括设计实践过程中遇到的问题、解决办法，也包括课程体会及对本次实践的建议和意见。

**二、 主要内容**

利用THINPAD教学计算机硬件平台，采用VHDL语言完成如下任务：

（1）16 位ALU设计

要求：① 完成算术运算（加、减），逻辑运算（与、或、非、异或）和移位（逻辑左移、逻辑右移、算术左移、算术右移、循环左移、循环右移）。

② 设置标志位（进位、溢出、全零、符号位）。

③ 利用数码管显示时钟周期，在THINPAD平台上验证结果。

（2）存储器实验

设计一个状态机和内存读写逻辑，完成对存储器RAM的访问。

要求：① 将手拨开关上的数据，写入到RAM的给定地址的存储单元中；若不重新设置地址，可连续写入后继单元。

② 读出指定地址存储单元中的数据，送到LED上显示。

（3）16位CPU设计

实现一个基于MIPS指令集的多周期CPU，数据总线16位，地址总线18位，具有8个16位的通用寄存器。七条指令分别为：ADDU SUBU SLL LI NOT LW SW。

要求：① 按照取指、译码、执行、访存和写回五个工作周期，分析指令的指令流程。

② 根据指令流程，设计处理器各功能部件。

③ 通过手拨开关和LED灯，验证CPU运行的正确性。

**备注：**本次实验要求以小组为单位完成，自由结组，每组3-4人。

**三、 进度计划**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **设计(实验)内容** | **完成时间** | **备注** |
| 1 | 接受课题并进行分析 | 2019.8.26 |  |
| 2 | 根据任务书选题，查阅资料，设计方案 | 2019.8.26-2019.8.28 |  |
| 3 | 熟悉开发环境，学习开发工具 | 2019.8.29 |  |
| 4 | 根据要实现的任务划分功能模块、模块功能实现、代码设计，在实验箱上调试 | 2019. 8.30-2019.9.4 |  |
| 5 | 对照任务书检查、完善系统并完成验收 | 2019.9.5 |  |
| 6 | 撰写并提交实验报告 | 2019.9.6 | 下午4点前交 |

**四、 成果要求**

1. 较完善的完成任务书所列主要内容，可查看实验结果并通过验收；

2. 详细的写出实验报告，要求规范、认真，并及时提交实验报告；

3. 杜绝报告抄袭行为，一经发现均判为不及格。

**五、 考核方式**

平时表现（10%）+检查验收（50%）+实验报告（40%）

学生姓名：

指导教师：刘书刚、王晓霞、张铭泉

**16位ALU设计**

1. 实验要求
2. 完成算术运算（加、减），逻辑运算（与、或、非、异或）和移位（逻辑左移、逻辑右移、算术左移、算术右移、循环左移、循环右移）。
3. 设置标志位（进位、溢出、全零、符号位）。
4. 利用数码管显示时钟周期，在THINPAD平台上验证结果。
5. 实验设计说明
6. 使用状态机实现操作数操作码的输入以及结果的输出，结果输出在led灯上，标志位显示在地址线指示灯addr[0]到addr[3]上。
7. 程序一共可以实现十四种运算，除了实验要求外，还包括ADC和SBB。
8. 每一种操作都会进行sf、zf状态位的判断。其中ADD、SUB、SLL、SRL、SRA,SLA、ROR、ROL、ADC、SBB会影响cf和of标志位。
9. 标志位为？则为无意义，为x则为不影响。
10. 实验部分代码
11. 状态转换

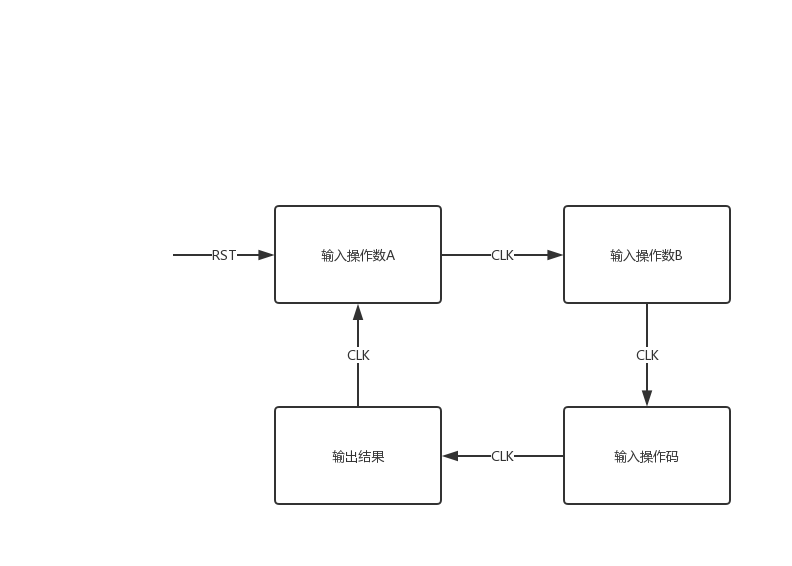
case state is

when 0 => state <= 1; a <= INPUT; stateCnt <= not "1000000";

when 1 => state <= 2; b <= INPUT; stateCnt <= not "1111001";

when 2 => state <= 3; opCode <= input(3 downto 0); stateCnt <= not "0100100";

when 3 => state <= 0; OUTPUT<= y;stateCnt <= not "0110000"; zfl<=zflag(0); cfl<=cflag(0); sfl<=sflag(0); ofl<=oflag(0);

end case;

1. 部分操作码代码

**--add**

when "0000" =>

y<= a + b;

if (a(15) = b(15)) then

if (y(15)/=b(15)) then

oFlag(0) <= '1';

else

oFlag(0) <= '0';

end if;

else oFlag(0) <= '0';

end if;

if(conv\_integer(b)+conv\_integer(a)>=2\*\*16)then

cflag(0) <= '1';

else

cflag(0) <= '0';

end if;

**--xor**

when "0100" => y<= a xor b;

**--srl**

when "0111" =>

if(a(15)='0')then

oFlag(0) <= '0';

else

oFlag(0) <= '1';

end if;

if(conv\_integer(b)>16)then

cflag(0)<='0';

else

cflag(0)<=a(conv\_integer(b)-1);

end if;

tempa <= to\_bitvector(a);

y<=to\_stdlogicvector(tempa srl conv\_integer(b));

**--sbb**

when "1001" =>

if(cflag(0)='1')then

y<= a + (not b) ;

else

y<= a + (not b) + 1;

end if;

if (a(15) /= b(15)) then

if (y(15)=b(15)) then

oFlag(0) <= '1';

else oFlag(0) <= '0';

end if;

else oFlag(0) <= '0';

end if;

ytemp <= ('0'&a) + ('0'&((not b)) + 1);

if(ytemp(16) = '1') then

cflag(0) <= '1';

else

cflag(0) <= '0';

end if;

**--rol**

when "1011" =>

if(a(15)=a(14))then

oFlag(0) <= '0';

else

oFlag(0) <= '1';

end if;

cflag(0)<=a(16-(conv\_integer(b-1) mod 16+1));

tempa <= to\_bitvector(a);

y<=to\_stdlogicvector(tempa rol conv\_integer(b));

1. 实验结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 输入数据 | | | 实际输出 | | 与预期一致性 |
| 操作码 | 操作数A | 操作数B | 运算结果 | 标志位(cf,of,sf,zf) |
| 0000  (ADD) | 1000000000000001 | 1000000010000000 | 0000000010000001 | 1100 | 是 |
| 0001  (SUB) | 1000000000001000 | 1000000000001000 | 0000000000000000 | 0001 | 是 |
| 0010  (AND) | 1000100000010001 | 0000000000000000 | 0000000000000000 | xx01 | 是 |
| 0011  (OR) | 1000100000010001 | 0001000110001000 | 1001100110011001 | xx10 | 是 |
| 0100  (XOR) | 0000000000001001 | 0000000000001000 | 0000000000000001 | xx00 | 是 |
| 0101  (NOT) | 0000111100001111 | - | 1111000011110000 | xx10 | 是 |
| 0110  (SLL) | 1010000001111111 | 0000000000000001 | 0100000011111110 | 1100 | 是 |
| 0111  (SRL) | 0000110000011110 | 0000000000001000 | 0000000000001100 | ??00 | 是 |
| 1000  (ADC)  (上次cf为1) | 1000000000001000 | 100000000001000 | 0000000000010001 | 1100 | 是 |
| 1001  (SBB)  (上次cf为0) | 0000111100000000 | 110000000000000 | 0100111100000000 | 0000 | 是 |
| 1010  (SRA) | 0001110000001001 | 0000000000001001 | 0000000000001110 | ??00 | 是 |
| 1011  (ROL) | 1011111100000000 | 0000000000000001 | 0111111000000001 | 1100 | 是 |
| 1100  (SLA) | 0001110011100000 | 0000000000000010 | 0111000000000000 | ??00 | 是 |
| 1101  (ROR) | 0011111100000000 | 0000000000000001 | 0001111110000000 | 0000 | 是 |

1. 实验总结

这个实验难点在于状态机的实现、标志位的判断和数据类型的转换。在实现状态机的时候，要明确状态转换以及在每个状态执行的顺序语句的位置，否则程序在运行的时候会混乱。对于所有运算，zf和sf的判断都是一样的，只需要判断结果变量是不是等于零和结果变量数组的最高位是不是‘1’。对于会影响cf和of标志位的运算在判断标志位时比较麻烦，需要按照汇编语言判断方法进行判断，这里也会经常用到数据类型的转换，以满足同时对单个位和数值整体进行比较。

通过运算器实验，我知道了如何使用状态机实现简单的程序。同时加深了对计算机内部各种运算的理解，特别是加减法和移位操作，以及对于每种运算的标志位的判断。对于VHDL程序的编写更加熟练了，为以后的实验打下基础。

**存储器实验**

1. 实验要求
2. 设计一个状态机和内存读写逻辑，完成对存储器RAM的访问。
3. 将手拨开关上的数据，写入到RAM的给定地址的存储单元中；若不重新设置地址，可连续写入后继单元。
4. 读出指定地址存储单元中的数据，送到LED上显示。
5. 实验设计说明
6. 这个程序使用状态机来实现读写操作，根据要求，一共有3个状态，第二第三状态每个重复10次。
7. 在每个读或写周期里，需要再分出4个小状态，来实现芯片读写信号的时序输入。同时在左边七段数码管上显示大状态，右边显示小状态。
8. 初始状态oe、we、dbc送‘1’信号，将其关闭。
9. 在写周期里，先送地址到地址线、送数据到数据线，在将oe、we设为“10”，将芯片写使能端打开，再关闭写使能端，完成一次写入操作。之后将地址变量和数据变量都加一，利用cnt计数器再重复进行操作10次。
10. 在写周期里，先送地址到地址线、把数据线设为高阻态，再将读使能端打开，再将数据线上的数据送到led灯进行显示，再关闭读使能端，完成一次读取操作。之后将地址变量加一，利用cnt计数器再重复进行操作10次。
11. 实验部分代码
12. 第一次写，出入地址与数据

if(cnt1=0)then if(state1=0)then

stateCnt <= not "1111001";

stateCnt2 <= not "1000000";

addr\_output<=seg;

addrtemp<=seg;

state1<=1;--送地址，存数据

else if(state1=1)then

stateCnt2 <= not "1111001";

data\_output<=seg;

datatemp<=seg;

en<='0';

OE<='1';

WE<='0';

state1<=2;--写信号

else if(state1=2)then

stateCnt2 <= not "0100100";

WE<='1';

state1<=3;

else

stateCnt2 <= not "0110000";

cnt1<=cnt1+1;

state1<=0;--信号复位

end if;

end if;

end if;

1. 第一次读

if(cnt2=0)then

if(state1=0)then

stateCnt <= not "0100100";

stateCnt2 <= not "1000000";

addr\_output<=addrtemp; --送控制信号,地址

data\_output<=(others=>'Z');

en<='0';

state1<=1;

else if(state1=1)then

OE<='0';

stateCnt2 <= not "1111001";

state1<=2;

else if(state1=2)then

stateCnt2 <= not "0100100";

led\_output<=data\_output;--送数据

cnt2<=cnt2+1;

OE<='1';

state1<=0;

end if;

end if;

end if;

1. 实验结果

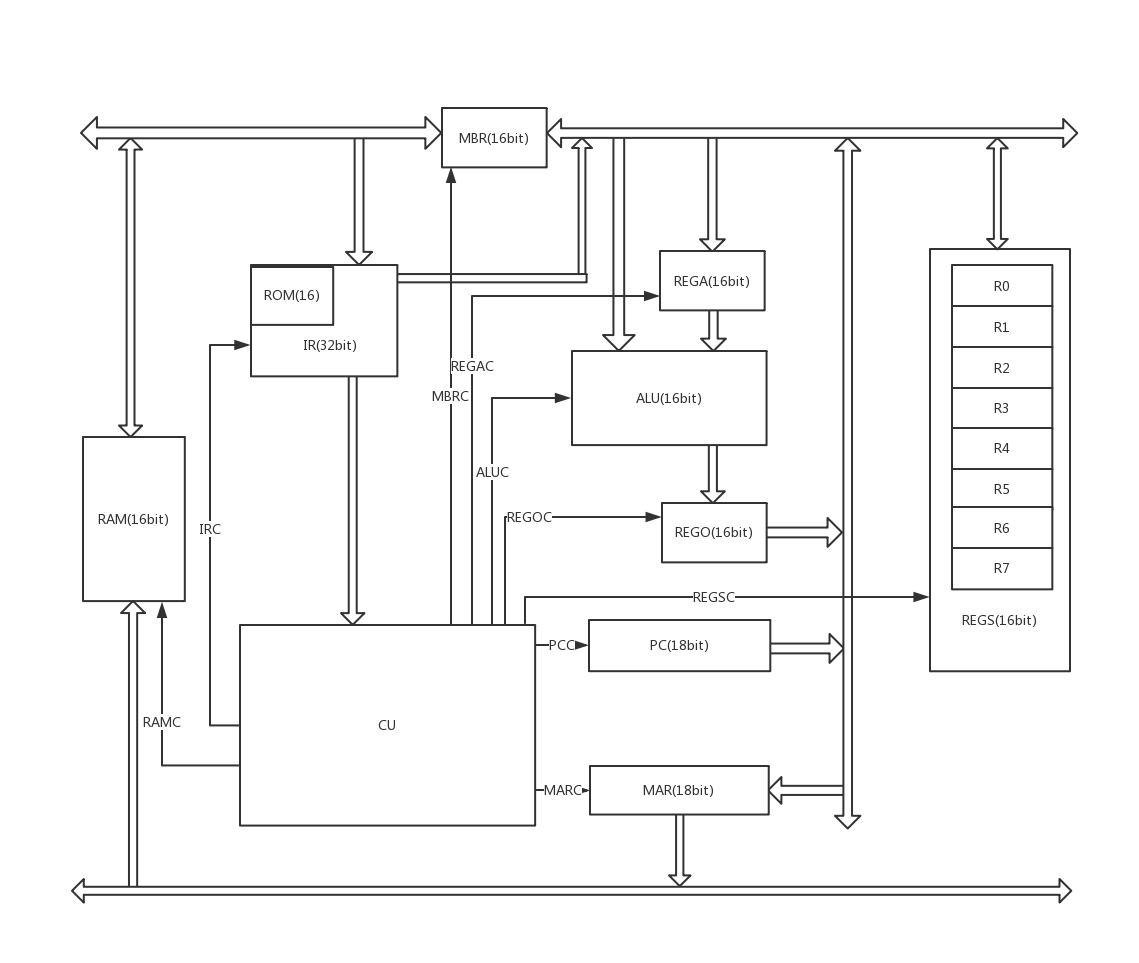
可以在地址线上看到，地址有二十次变化，前十次根据输入的地址依次加一，将数据送入sram。后十次地址的值与前十次一样，此时为从sram读数据。

可以在数据线上看到，数据有二十次变化，前十次根据输入的数据依次加一，被送入sram。后十次数据的值与前十次一样，此时是从sram读出的数据。

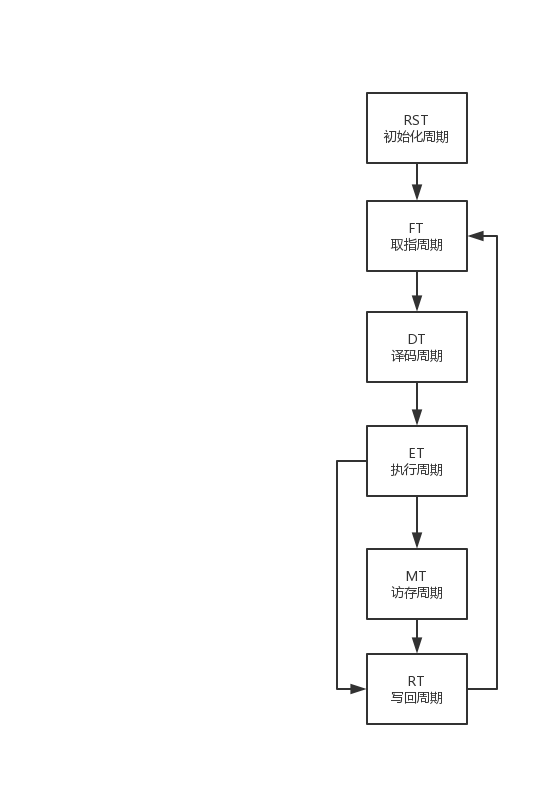
可以看到we,oe,en标志位随着状态机变化。

1. 实验总结
2. 利用小状态实现读或写周期中信号传入的时序。最开始做实验时没有参考sram读写周期时序图，导致数据总是不能写入内存，后来严格参照时序图才正确的完成了信号时序的输入。
3. 在最开始数据之后高八位写入了内存，后来经过查资料发现，串口与sram公用数据线，在对sram进行读写操作时，应将DBC端口致‘1’将其关闭，让其让出数据线。
4. 通过存储器实验我深刻的理解了内存的读写时序图，理解了如何通过信号的输入让程序实现这个时序图。读写的关键之处就是使能信号与数据传入的顺序。因为我们之前不知道dbc端口的存在，导致我们浪费了很多时间去解决数据线信号的问题，但是这也加深了我对总线的理解，如何通过使能信号控制各设备使用总线传输数据。

**16位CPU设计**

1. 实验要求
2. 实现一个基于MIPS指令集的多周期CPU，数据总线16位，地址总线18位，具有8个16位的通用寄存器。七条指令分别为：ADDU SUBU SLL LI NOT LW SW。
3. 按照取指、译码、执行、访存和写回五个工作周期，分析指令的指令流程。
4. 根据指令流程，设计处理器各功能部件。
5. 通过手拨开关和LED灯，验证CPU运行的正确性。
6. 实验设计说明
7. cpu原理图

另外，每个元件都有一根CLK连接至CU，CU还有CLOCK与RST输入，CU还有两个7段数码管的输出，REGS还有16位led灯的输出，IR还有一位SW的输入

1. 指令周期
2. 元件说明
3. RAM 内存读写控制器（并不是内存）

case RAMC is

when "000" =>data<=(others=>'Z');--地址线送高阻

when "001" =>OE<='0';--读1

when "010" =>OE<='1';data<=(others=>'Z');--读2

when "011" => WE<='0';--写1

when "100" => WE<='1';--写2

when others => en<='0';dbc<='1';OE<='1';WE<='1';

end case;

地址线上的数据由MAR提前准备好，写时MBR提前将数据打入外总线，内存控制器只需控制sram的控制线。

1. MBR 数据缓冲寄存器

case MBRC is

when "000" =>REG<= outdata;indata <= (others=>'Z');--外总线到寄存器

when "001" =>outdata<= REG;indata <= (others=>'Z');--寄存器到外总线

when "010" =>REG<= indata;outdata <= (others=>'Z');--内总线到寄存器

when "011" =>indata<= REG;outdata <= (others=>'Z');--寄存器到内总线

when others => outdata <= (others=>'Z');indata <= (others=>'Z');

end case;

数据缓冲寄存器是双口寄存器，分别连接内总线与外总线，但同一时刻，只允许单口单向，一个口工作时，另一个口总是高阻态。

1. IR（以及ROM） 指令寄存器

if (READY = '0') then

--li r1 10

romdata(0)<="0001000000000001";

romdata(1)<="0000000000001010";

case IRC is

when "000" =>REG(31 downto 16) <= romdata(i);i<=i+1;--ROM到IR1

when "001" => REG(15 downto 0) <= romdata(i);i<=i+1;--ROM到IR2

when "010" => OUTPUTCU <= REG(31 downto 0);OUTPUTIN <= (others=>'Z'); --IR到CU

when "011" => OUTPUTIN <= REG(31 downto 16);--IR1到内总线

when "100" => OUTPUTIN <= REG(15 downto 0);--IR2到内总线

when "101" => null;

when others => OUTPUTIN <= (others=>'Z');

end case;

elsif (READY = '1') then

case IRC is

when "000" => REG(31 downto 16) <= INPUT;OUTPUTIN <= (others=>'Z');--外总线到IR1

when "001" => REG(15 downto 0) <= INPUT;OUTPUTIN <= (others=>'Z');--外总线到IR2

when "010" => OUTPUTCU <= REG(31 downto 0);OUTPUTIN <= (others=>'Z');--IR到CU

when "011" => OUTPUTIN <= REG(31 downto 16);--IR1到内总线

when "100" => OUTPUTIN <= REG(15 downto 0);--IR2到内总线

when "101" => null;

when others => OUTPUTIN <= (others=>'Z');i<=0;

end case;

end if;

根据实际情况，IR有两种工作模式，一是ROM-IR模式，而是RAM-IR模式。前者直接将指令写入ROM，并根据时间推进依次读取。后者是指令存放在RAM中，根据PC指向的地址取指。

指令都是32位的，但是内外总线都是16位的，因此IR分高（IR1）低（IR2）两部分。IR与CU的连线是32位的，IR与内外总线的连线是16位的。

1. REGA 运算器输入缓冲寄存器

case REGAC is

when "00" => REG <= INPUT;--内总线到寄存器

when "01" => OUTPUT <= REG;--寄存器到ALU的A口

when others => OUTPUT <= (others=>'Z');

end case;

1. ALU 运算器

case ALUC is

when "000" => OUTPUT <= INPUTa + INPUTb;--a+b

when "001" => OUTPUT <= INPUTa - INPUTb;--a-b

when "011" => OUTPUT <= to\_stdlogicvector(to\_bitvector(INPUTb) sll conv\_integer(INPUTa(10 downto 6)));--详见sll1指令介绍

when "100" => OUTPUT <= to\_stdlogicvector(to\_bitvector(INPUTb) sll conv\_integer(INPUTa));--b左移（a）位

when "101" => OUTPUT <= not INPUTb;--b取反

when "110" => null;

when others => OUTPUT <= (others => 'Z');

end case;

运算器能执行五种运算，算术加、算术减、逻辑左移、取反。其中011状态执行的是逻辑左移操作，但是左移位数这个值来自立即数，详见指令sll1的介绍。

1. REGO 运算器输出缓冲寄存器

case REGOC is

when "00" => REG <= INPUT;--ALU到寄存器

when "01" => OUTPUT <= REG;--寄存器到内总线

when others => OUTPUT <= (others=>'Z');

end case;

该寄存器作为ALU的输出与内总线间的缓冲。

1. PC 程序计数器

case PCC is

when "00" => REG <= "000000000000000000";--初始化PC

when "01" => OUTPUT <= REG;--PC到内总线

when "10" => REG <= REG + 1;--PC+1->PC

when others => OUTPUT <= (others=>'Z');

end case;

程序计数器是18位的，并且能够进行自加操作。

1. MAR 地址缓冲寄存器

case MARC is

when "00" => REG <= INPUT;--内总线到MAR

when "01" => REG(17 downto 16) <= INPUT(1 downto 0);--内总线低两位到MAR2

when "10" => OUTPUT <= REG;--MAR到地址线

when others => OUTPUT <= (others=>'Z');

end case;

地址寄存器是18位的，因为内总线元件大多是16位的，所以将MAR分为MAR1（低16位）和MAR2（高两位）。

1. REGS 寄存器阵列（8个通用寄存器）

type regarray is array (0 to 7) of bit\_vector(15 downto 0);

signal ramdata : regarray;

begin

process(CLK)

begin

if(CLK'event and CLK = '1')then

case REGSC(4 downto 3) is

when "00" => ramdata(conv\_integer(REGSC(2 downto 0))) <= to\_bitvector(DATA);led<=DATA;

when "01" => DATA <= to\_stdlogicvector(ramdata(conv\_integer(REGSC(2 downto 0))));led<="1000000000000100";

when others => DATA <= (others=>'Z');led<="1110000000000111";

end case;

end if;

end process;

寄存器阵列将数据存放在二维数组中，宽度为16位，深度为8位。为了实验效果，寄存器阵列的连接了16位led灯，以展示内总线上的数据。

1. 内总线

内总线宽度18位，除了PC和MAR是18位的寄存器，其余原件与内总线的连续都是16位的。连接示意图如下

INBUS 0

INBUS 1

INBUS 15

INBUS 16

INBUS 17

BUS16 0

BUS16 1

BUS16 15

BUS18 0

BUS18 1

BUS18 15

BUS18 16

BUS18 17

内总线

1. CU 控制器

控制器代码较多，参见下文“实验部分代码”。

CU按照指令时间操作表设计，并在适当的时候将原件置为空操作、高阻态状态。

高阻态状态是为了避免元件在工作期间内接收到了多个clk上升沿而多次进行不可重复的操作，例如PC+1->PC。空操作状态是为了保持元件的输出的同时避免不可重复操作。

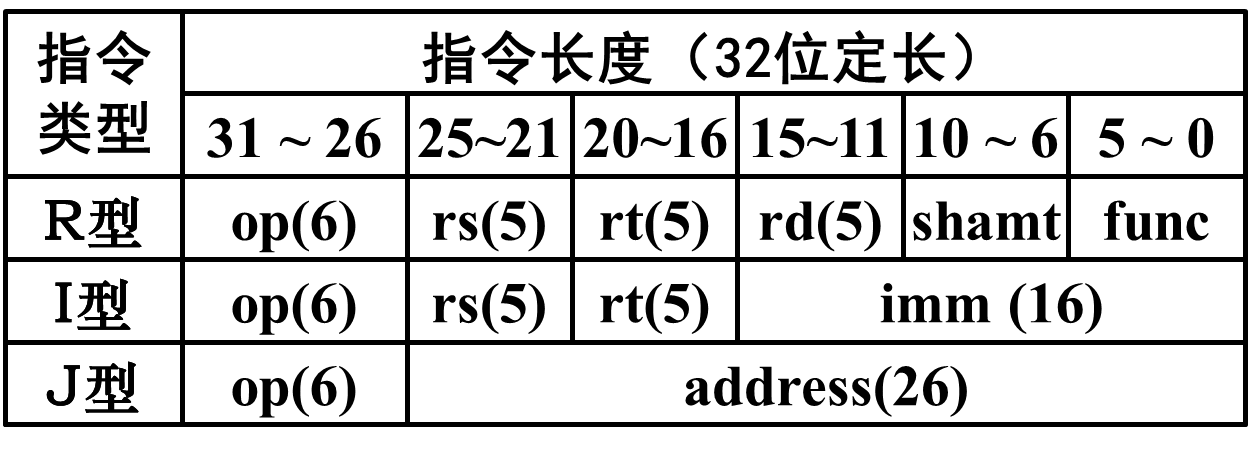
为了实验效果，CU连接了两个7段数码管，左数码管显示的是机器周期，右数码管显示的是外节拍数（CLOCK）的个位数。

CU使用CLOCK（外节拍）来控制CU的执行，自己生成CLK（内节拍）来控制其他元件的执行，以消除抖动以及控制信号时延。

CU能够识别11钟指令，分别是 ADDU1, ADDU2，SUBU1, SUBU2, SUBU3, SLL1, SLL2, LI, NOT, LW, SW.

1. 指令、指令示例、指令流程

（特别说明，每条指令的译码周期只有一拍，仅仅是分支识别，没有特别操作，因此指令流程中没有操作）

本实验使用MIPS指令集架构，只有R型和I型指令。

1. ADDU1

**示例：**

ADDU1 R0, R1, R2 (空) （功能码）

000000 00000 00001 00010 00000 000000

**解释：**

R型指令，R0 + R1 -> R2

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** R0->REGA ,R1+REGA->REGO ,REGO->R2

**RT** PC->MAR

1. ADDU2

**示例：**

ADDU2 R0, R1, 立即数11

000001 00000 00001 0000000000001011

**解释：**

I型指令，R0 + 11 -> R1

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** R0->REGA ,IR2+REGA->REGO ,REGO->R1

**RT** PC->MAR

1. SUBU1

**示例：**

SUBU1 R0, R1, R2 (空) （功能码）

000000 00000 00001 00010 00000 000001

**解释：**

R型指令，R0 - R1 -> R2

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** R0->REGA ,REGA-R1->REGO ,REGO->R2

**RT** PC->MAR

1. SUBU2

**示例：**

SUBU2 R0, R1, 立即数13

000010 00000 00001 0000000000001101

**解释：**

I型指令，R0 - 13 -> R1

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** R0->REGA ,REGA-IR2->REGO ,REGO->R1

**RT** PC->MAR

1. SUBU3

**示例：**

SUBU3 R0, R1, 立即数13

000011 00000 00001 0000000000001101

**解释：**

I型指令，13 – R0 -> R1

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** IR2->REGA ,REGA-R0->REGO ,REGO->R1

**RT** PC->MAR

1. SLL1

**示例：**

SLL1 R0, (空), R1 立即数5 （功能码）

000000 00000 00000 00001 00101 000010

**解释：**

R型指令，R0 << 5 -> R1

因为立即数存放在指令的第10到第6位，同时为了简化连接线，IR2全部送入REGA，在ALU中执行左移操作时，操作数b只取这5位

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** IR2->REGA ,R0<<REGA(10 – 6位)->REGO ,REGO->R1

**RT** PC->MAR

1. SLL2

**示例：**

SLL2 R0, R1 ,R2 (空) （功能码）

000000 00000 00001 00010 00000 000110

**解释：**

R型指令，R0 << R1 -> R2

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** R1->REGA ,R0<<REGA->REGO ,REGO->R2

**RT** PC->MAR

1. LI

**示例：**

LI (空) R1, 立即数7

000100 00000 00001 0000000000000111

**解释：**

I型指令，7 -> R1

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** IR2-> R0

**RT** PC->MAR

1. NOT

**示例：**

NOT R0, (空) R1 (空) （功能码）

000000 00000 00000 00001 00000 000011

**解释：**

R型指令，not R0 -> R1

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** not R0->REGO ,REGO->R1

**RT** PC->MAR

1. LW

**示例：**

LW R0, R1, R2 (空) （功能码）

000000 00000 00001 00010 00000 000100

**解释：**

R型指令，[R0(15 to 0) R1(1 to 0)] -> R2

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** R0->MAR1 ,R1->MAR2

**MT** M->MBR ,MBR->R2

**RT** PC->MAR

1. SW

**示例：**

SW R0, R1, R2 (空) （功能码）

000000 00000 00001 00010 00000 000101

**解释：**

R型指令，R2 -> [R0(15 to 0) R1(1 to 0)]

**指令流程图：**

**FT** M->IR1 ,PC+1->PC ,PC->MAR ,M->IR2 ,PC+1->PC

**DT**

**ET** R2->MBR ,R0->MAR1 ,R2->MAR2

**RT** MBR->M ,PC->MAR

1. 测试小程序

LI R1 , 10;

0001000000000001 0000000000001010

LI R2 , 9;

0001000000000010 0000000000001001

LI R3 , 4;

0001000000000011 0000000000000100

ADDU R1 ,R2 ,R4;

0000000000100010 0010000000000000

SW R1 ,R2 ,R4;

0000000000100010 0010000000000101

SUBU R4 ,R3 ,R4;

0000000010000011 0010000000000001

SLL R4 ,R4 ,1;

0000000010000000 0010000001000010

NOT R4 ,R4;

0000000010000000 0010000000000011

LW R1 ,R2 ,R5;

0000000000100010 0010100000000100

1. 实验部分代码
2. CU代码

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity CU is

port(

ALUC : out STD\_LOGIC\_VECTOR (2 downto 0);

RAMC : out STD\_LOGIC\_VECTOR (2 downto 0);

REGOC : out STD\_LOGIC\_VECTOR (1 downto 0);

REGAC : out STD\_LOGIC\_VECTOR (1 downto 0);

PCC : out STD\_LOGIC\_VECTOR (1 downto 0);

REGSC : out STD\_LOGIC\_VECTOR (4 downto 0);

MARC : out STD\_LOGIC\_VECTOR (1 downto 0);

MBRC : out STD\_LOGIC\_VECTOR (2 downto 0);

IRC : out STD\_LOGIC\_VECTOR (2 downto 0);

INPUTIR : in STD\_LOGIC\_VECTOR (31 downto 0);

led1: out STD\_LOGIC\_VECTOR (6 downto 0);

led2: out STD\_LOGIC\_VECTOR (6 downto 0);

CLK : out STD\_LOGIC;

CLOCK : in STD\_LOGIC;

RST : in STD\_LOGIC

);

end CU;

architecture Behavioral of CU is

signal stateTerm: integer range 0 to 5;

signal stateIns: integer range 0 to 11;

signal state: integer range 0 to 19;

begin

process(CLOCK,RST)

begin

if (RST = '0')then

pcc <= "00";

ALUC<="111";

RAMC<="111";

REGOC<="11";

REGAC<="11";

REGSC<="11111";

MARC<="11";

MBRC<="111";

IRC<="111";

led1<="0111111";

led2<="0111111";

clk <= '0';

stateTerm <= 0;

state <= 0;

stateIns <= 0;

elsif(CLOCK'event and CLOCK = '1' )then

case stateTerm is

when 0 => led1<="0111111";

when 1 => led1<="0000110";

when 2 => led1<="1011011";

when 3 => led1<="1001111";

when 4 => led1<="1100110";

when 5 => led1<="1101101";

end case;

case state is

when 0 => led2<="0111111";

when 1 => led2<="0000110";

when 2 => led2<="1011011";

when 3 => led2<="1001111";

when 4 => led2<="1100110";

when 5 => led2<="1101101";

when 6 => led2<="1111101";

when 7 => led2<="0000111";

when 8 => led2<="1111111";

when 9 => led2<="1101111";

when 10 => led2<="0111111";

when 11 => led2<="0000110";

when 12 => led2<="1011011";

when 13 => led2<="1001111";

when 14 => led2<="1100110";

when 15 => led2<="1101101";

when 16 => led2<="1111101";

when 17 => led2<="0000111";

when 18 => led2<="1111111";

when 19 => led2<="1101111";

end case;

if(stateTerm = 0) then --rst

--pc->mar

case state is

when 0 => state <= 1; clk <= '1';

when 1 => state <= 2; clk <= '0';pcc <= "01";

when 2 => state <= 3; clk <= '1';

when 3 => state <= 4; clk <= '0';MARC<="00";

when 4 => state <= 5; clk <= '1';

when 5 => state <= 6; clk <= '0';MARC<="10";pcc <= "11";

when 6 => stateTerm <= 1;state <= 0; clk <= '1';

when others => null;

end case;

elsif(stateTerm = 1) then --ft

case state is

when 0 => state <= 1; clk <= '0';MARC<="10";pcc <= "10";

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';RAMC<="000";pcc <= "11";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';RAMC<="001";

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';IRC<="000";--M->ir1

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';pcc<= "01";IRC<="111";RAMC<="010";

when 9 => state <= 10; clk <= '1';

when 10 => state <= 11; clk <= '0';MARC<="00";pcc <= "10";

when 11 => state <= 12; clk <= '1';

when 12 => state <= 13; clk <= '0';MARC<="10";pcc <= "11";

when 13 => state <= 14; clk <= '1';

when 14 => state <= 15; clk <= '0';RAMC<="001";

when 15 => state <= 16; clk <= '1';

when 16 => state <= 17; clk <= '0';IRC<="001";

when 17 => state <= 18; clk <= '1';

when 18 => state <= 19; clk <= '0';IRC<="010";RAMC<="010";--M->ir2

when 19 => stateTerm <= 2;state <= 0; clk <= '1';

when others => null;

end case;

elsif(stateTerm = 2) then --dt

case INPUTIR(31 downto 26) is

when "000000" =>

case INPUTIR(5 downto 0) is

when "000000" => stateIns<=1;stateTerm <= 3;state <= 0;--addu1

when "000001" => stateIns<=2;stateTerm <= 3;state <= 0;--subu1

when "000010" => stateIns<=3;stateTerm <= 3;state <= 0;--sll1

when "000011" => stateIns<=4;stateTerm <= 3;state <= 0;--not

when "000100" => stateIns<=5;stateTerm <= 3;state <= 0;--lw

when "000101" => stateIns<=6;stateTerm <= 3;state <= 0;--sw

when "000110" => stateIns<=7;stateTerm <= 3;state <= 0;--sll2

when others => stateTerm <= 0;

end case;

when "000001" => stateIns<=8;stateTerm <= 3;state <= 0;--addu2

when "000010" => stateIns<=9;stateTerm <= 3;state <= 0;--subu2

when "000011" => stateIns<=10;stateTerm <= 3;state <= 0;--subu3

when "000100" => stateIns<=11;stateTerm <= 3;state <= 0;--li

when others => stateTerm <= 0;

end case;

elsif(stateTerm = 3) then --pt

case stateIns is

when 1 => --addu1--addu1--

case state is

when 0 => state <= 1; clk <= '0';REGSC<="01"&INPUTIR(23 downto 21);--r0->rega

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';REGAC<="00";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';REGSC<="01"&INPUTIR(18 downto 16);REGAC<="01";--rega+r1->rego

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';ALUC<="000";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';REGOC<="00";REGSC<="11111";REGAC<="11";ALUC<="110";

when 9 => state <= 10; clk <= '1';

when 10 =>state<= 11; clk <= '0';REGOC<="01";ALUC<="111";--rego->r2

when 11 => state <= 12; clk <= '1';

when 12 => state <= 13; clk<='0';REGSC<="00"&INPUTIR(13 downto 11);

when 13 => state <= 14; clk <= '1';

when 14 => state <= 15; clk <= '0';REGOC<="11";REGSC<="11111";

when 15 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 2 => --subu1

case state is

when 0 => state <= 1; clk <= '0';REGSC<="01"&INPUTIR(23 downto 21);--r0->rega

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';REGAC<="00";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';REGSC<="01"&INPUTIR(18 downto 16);REGAC<="01";--rega-r1->rego

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';ALUC<="001";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';REGSC<="11111";REGOC<="00";REGAC<="11";ALUC<="110";

when 9 => state <= 10; clk <= '1';

when 10 =>state <=11; clk <= '0';REGOC<="01";ALUC<="111";--rego->r2

when 11 => state <= 12; clk <= '1';

when 12 => state <=13;clk <= '0';REGSC<="00"&INPUTIR(13 downto 11);

when 13 => state <= 14; clk <= '1';

when 14 => state <= 15; clk <= '0';REGOC<="11";REGSC<="11111";

when 15 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 3 => --sll1

case state is

when 0 => state <= 1; clk <= '0';IRC<="100";

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';REGAC<="00";IRC<="101";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';IRC<="111";REGSC<="01"&INPUTIR(23 downto 21);REGAC<="01";--r1<<(rega)->rego

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';ALUC<="011";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';REGSC<="11111";REGOC<="00";REGAC<="11";ALUC<="110";

when 9 => state <= 10; clk <= '1';

when 10 =>state <= 11;clk <= '0';REGOC<="01";ALUC<="111";--rego->r2

when 11 => state <= 12; clk <= '1';

when 12 => state <= 13;clk <='0';REGSC<="00"&INPUTIR(13 downto 11);

when 13 => state <= 14; clk <= '1';

when 14 => state <= 15; clk <= '0';REGOC<="11";REGSC<="11111";

when 15 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 4 => --not

case state is

when 0 => state <= 1; clk <= '0';REGSC<="01"&INPUTIR(23 downto 21);--not r0->rego

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';ALUC<="101";

when 3 => state <= 4; clk <= '1';

when 4 =>state<=5;clk<= '0';REGSC<="11111";REGOC<="00";ALUC<="110";

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';REGOC<="01";ALUC<="111";--rego->r2

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';REGSC<="00"&INPUTIR(13 downto 11);

when 9 => state <= 10; clk <= '1';

when 10 => state <= 11; clk <= '0';REGOC<="11";REGSC<="11111";

when 11 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 5 => --lw

case state is

when 0 => state <= 1; clk <= '0';REGSC<="01"&INPUTIR(23 downto 21);--r0->mar1

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';MARC<="00";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';REGSC<="01"&INPUTIR(18 downto 16);MARC<="11";--r2->mar2

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';MARC<="01";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';REGSC<="11111";MARC<="11";

when 9 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 6 => --sw

case state is

when 0 => state <= 1; clk <= '0';REGSC<="01"&INPUTIR(13 downto 11);--r2->mbr

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';MBRC<="010";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';MBRC<="001";

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';REGSC<="01"&INPUTIR(23 downto 21);--r0->mar1

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';MARC<="00";

when 9 => state <= 10; clk <= '1';

when 10 => state <= 11; clk <= '0';REGSC<="01"&INPUTIR(18 downto 16);MARC<="11";--r2->mar2

when 11 => state <= 12; clk <= '1';

when 12 => state <= 13; clk <= '0';MARC<="01";

when 13 => state <= 14; clk <= '1';

when 14 => state <= 15; clk <= '0';REGSC<="11111";MARC<="11";

when 15 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 7 => --sll2

case state is

when 0 => state <= 1; clk <= '0';REGSC<="01"&INPUTIR(23 downto 21);--r0->rega

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';REGAC<="00";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';REGSC<="01"&INPUTIR(18 downto 16);REGAC<="01";--r1<<(rega)->rego

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';ALUC<="100";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';REGSC<="11111";REGOC<="00";REGAC<="11";ALUC<="110";

when 9 => state <= 10; clk <= '1';

when 10 => state <= 11;clk <='0';REGOC<="01";ALUC<="111";--rego->r2

when 11 => state <= 12; clk <= '1';

when 12 => state <= 13; clk<='0';REGSC<="00"&INPUTIR(13 downto 11);

when 13 => state <= 14; clk <= '1';

when 14 => state <= 15; clk <= '0';REGOC<="11";REGSC<="11111";

when 15 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 8 => --addu2

case state is

when 0 => state <= 1; clk <= '0';REGSC<="01"&INPUTIR(23 downto 21);--r0->rega

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';REGAC<="00";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';IRC<="100";REGAC<="01";REGSC<="11111";--rega+绔嬪嵆鏁>rego

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';ALUC<="000";IRC<="101";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';REGOC<="00";REGAC<="11";IRC<="111";ALUC<="110";

when 9 => state <= 10; clk <= '1';

when 10 => state <= 11;clk <='0';REGOC<="01";ALUC<="111";--rego->r1

when 11 => state <= 12; clk <= '1';

when 12 => state <= 13; clk<='0';REGSC<="00"&INPUTIR(18 downto 16);

when 13 => state <= 14; clk <= '1';

when 14 => state <= 15; clk <= '0';REGOC<="11";REGSC<="11111";

when 15 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 9 => --subu2

case state is

when 0 => state <= 1; clk <= '0';REGSC<="01"&INPUTIR(23 downto 21);--r0->rega

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';REGAC<="00";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';IRC<="100";REGAC<="01";REGSC<="11111";--rega-绔嬪嵆鏁>rego

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';ALUC<="001";IRC<="101";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';REGOC<="00";REGAC<="11";IRC<="111";ALUC<="110";

when 9 => state <= 10; clk <= '1';

when 10 => state <= 11;clk <='0';REGOC<="01";ALUC<="111";--rego->r1

when 11 => state <= 12; clk <= '1';

when 12 => state <= 13;clk<= '0';REGSC<="00"&INPUTIR(18 downto 16);

when 13 => state <= 14; clk <= '1';

when 14 => state <= 15; clk <= '0';REGOC<="11";REGSC<="11111";

when 15 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 10 => --subu3

case state is

when 0 => state <= 1; clk <= '0';IRC<="100";--绔嬪嵆鏁>rega

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';REGAC<="00";IRC<="101";

when 3 => state <= 4; clk <= '1';

when 4=>state<=5;clk <= '0';IRC<="111" ; REGAC<="01" ; REGSC<="01"&INPUTIR(23 downto 21);--rega-r0->rego

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';ALUC<="001";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';REGOC<="00";REGAC<="11";REGSC<="11111";ALUC<="110";

when 9 => state <= 10; clk <= '1';

when 10 => state <= 11; clk <= '0';REGOC<="01";ALUC<="111";--rego->r1

when 11 => state <= 12; clk <= '1';

when 12 => state <= 13;clk <='0';REGSC<="00"&INPUTIR(18 downto 16);

when 13 => state <= 14; clk <= '1';

when 14 => state <= 15; clk <= '0';REGOC<="11";REGSC<="11111";

when 15 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when 11 => --li

case state is

when 0 => state <= 1; clk <= '0';IRC<="100";--绔嬪嵆鏁>r1

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';REGSC<="00"&INPUTIR(18 downto 16);IRC<="101";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';IRC<="111";REGSC<="11111";

when 5 => state <= 0; stateTerm <= 4; clk <= '1';

when others => null;

end case;

when others => null;

end case;

elsif(stateTerm = 4) then --mt

case stateIns is

when 5 =>

case state is

when 0 => state <= 1; clk <= '0';MARC<="10";--M->mbr

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';RAMC<="000";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';RAMC<="001";

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';MBRC<="000";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';MBRC<="011";RAMC<="010";--mbr->r2

when 9 => state <= 10; clk <= '1';

when 10 => state <= 11; clk<='0';REGSC<="00"&INPUTIR(13 downto 11);

when 11 => state <= 12; clk <= '1';

when 12 => state <= 13; clk <= '0';MBRC<="111";REGSC<="11111";

when 13 => stateTerm <= 5;state <= 0; clk <= '1';

when others => null;

end case;

when others => state <= 0; stateTerm <= 5;

end case;

elsif(stateTerm = 5) then --rt

case stateIns is

when 6 =>

case state is

when 0 => state <= 1; clk <= '0';MARC<="10";--mbr->m

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';RAMC<="011";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';RAMC<="100";MBRC<="111";pcc <= "01";--pc->mar

when 5 => state <= 6; clk <= '1';

when 6 => state <= 7; clk <= '0';MARC<="00";

when 7 => state <= 8; clk <= '1';

when 8 => state <= 9; clk <= '0';MARC<="10";pcc <= "11";

when 9 => stateTerm <= 1;state <= 0; clk <= '1';

when others => null;

end case;

when others =>

case state is

when 0 => state <= 1; clk <= '0';pcc <= "01";--pc->mar

when 1 => state <= 2; clk <= '1';

when 2 => state <= 3; clk <= '0';MARC<="00";

when 3 => state <= 4; clk <= '1';

when 4 => state <= 5; clk <= '0';MARC<="10";pcc <= "11";

when 5 => stateTerm <= 1;state <= 0; clk <= '1';

when others => null;

end case;

end case;

else null;

end if;

end if;

end process;

end Behavioral;

1. 元件例化代码

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity CPU is

port(

ADDRCPU : out STD\_LOGIC\_VECTOR (17 downto 0);

DATACPU : inout STD\_LOGIC\_VECTOR (15 downto 0);

INDATACPU : out STD\_LOGIC\_VECTOR (15 downto 0);

ENCPU : inout STD\_LOGIC;

WECPU : inout STD\_LOGIC;

OECPU : inout STD\_LOGIC;

DBCCPU : inout STD\_LOGIC;

CLOCKCPU : in STD\_LOGIC;

RSTCPU : in STD\_LOGIC;

LEDCPU1:out STD\_LOGIC\_VECTOR (6 downto 0);

LEDCPU2:out STD\_LOGIC\_VECTOR (6 downto 0);

LEDCPU3: out STD\_LOGIC\_VECTOR (15 downto 0);

READYCPU : in STD\_LOGIC

);

end CPU;

architecture Behavioral of CPU is

component CU

port(

ALUC : out STD\_LOGIC\_VECTOR (2 downto 0);

RAMC : out STD\_LOGIC\_VECTOR (2 downto 0);

REGOC : out STD\_LOGIC\_VECTOR (1 downto 0);

REGAC : out STD\_LOGIC\_VECTOR (1 downto 0);

PCC : out STD\_LOGIC\_VECTOR (1 downto 0);

REGSC : out STD\_LOGIC\_VECTOR (4 downto 0);

MARC : out STD\_LOGIC\_VECTOR (1 downto 0);

MBRC : out STD\_LOGIC\_VECTOR (2 downto 0);

IRC : out STD\_LOGIC\_VECTOR (2 downto 0);

INPUTIR : in STD\_LOGIC\_VECTOR (31 downto 0);

led1: out STD\_LOGIC\_VECTOR (6 downto 0);

led2: out STD\_LOGIC\_VECTOR (6 downto 0);

CLK : out STD\_LOGIC;

CLOCK : in STD\_LOGIC;

RST : in STD\_LOGIC

);

end component;

component IR

port(

CLK : in STD\_LOGIC;

IRC : in STD\_LOGIC\_VECTOR (2 downto 0);

INPUT : in STD\_LOGIC\_VECTOR (15 downto 0);

OUTPUTCU : out STD\_LOGIC\_VECTOR (31 downto 0);

OUTPUTIN : out STD\_LOGIC\_VECTOR (15 downto 0);

READY:in STD\_LOGIC

);

end component;

component MAR

Port ( CLK : in STD\_LOGIC;

MARC : in STD\_LOGIC\_VECTOR (1 downto 0);

INPUT : in STD\_LOGIC\_VECTOR (17 downto 0);

OUTPUT : out STD\_LOGIC\_VECTOR (17 downto 0)

);

end component;

component MBR

port(

CLK : in STD\_LOGIC;

MBRC : in STD\_LOGIC\_VECTOR (2 downto 0);

outdata : inout STD\_LOGIC\_VECTOR (15 downto 0);

indata : inout STD\_LOGIC\_VECTOR (15 downto 0)

);

end component;

component PC

Port ( CLK : in STD\_LOGIC;

PCC : in STD\_LOGIC\_VECTOR (1 downto 0);

OUTPUT : inout STD\_LOGIC\_VECTOR (17 downto 0)

);

end component;

component RAM

Port (

CLK : in STD\_LOGIC;

RAMC : in STD\_LOGIC\_VECTOR (2 downto 0);

EN : out STD\_LOGIC;

WE:out STD\_LOGIC;

OE:out STD\_LOGIC;

DBC:out std\_logic

);

end component;

component REGA

Port ( CLK : in STD\_LOGIC;

REGAC : in STD\_LOGIC\_VECTOR (1 downto 0);

INPUT : in STD\_LOGIC\_VECTOR (15 downto 0);

OUTPUT : out STD\_LOGIC\_VECTOR (15 downto 0)

);

end component;

component REGO

Port ( CLK : in STD\_LOGIC;

REGOC : in STD\_LOGIC\_VECTOR (1 downto 0);

INPUT : in STD\_LOGIC\_VECTOR (15 downto 0);

OUTPUT : out STD\_LOGIC\_VECTOR (15 downto 0)

);

end component;

component REGS

Port ( CLK : in STD\_LOGIC;

REGSC : in STD\_LOGIC\_VECTOR (4 downto 0);

DATA : inout STD\_LOGIC\_VECTOR (15 downto 0);

led : out STD\_LOGIC\_VECTOR (15 downto 0)

);

end component;

component ALU

Port ( CLK : in STD\_LOGIC;

ALUC : in STD\_LOGIC\_VECTOR (2 downto 0);

INPUTa : in STD\_LOGIC\_VECTOR (15 downto 0);

INPUTb : in STD\_LOGIC\_VECTOR (15 downto 0);

OUTPUT : out STD\_LOGIC\_VECTOR (15 downto 0)

);

end component;

--signal carry\_out1 : STD\_LOGIC\_VECTOR (15 downto 0);--DBOUT

signal carry\_out2 : STD\_LOGIC\_VECTOR (17 downto 0);--DBIN

--signal carry\_out3 : STD\_LOGIC\_VECTOR (17 downto 0);--AB

signal carry\_out4 : STD\_LOGIC\_VECTOR (15 downto 0);--REGA\_ALU

signal carry\_out5 : STD\_LOGIC\_VECTOR (31 downto 0);--IR\_CU

signal carry\_out6 : STD\_LOGIC\_VECTOR (2 downto 0);--IRC

signal carry\_out7 : STD\_LOGIC\_VECTOR (2 downto 0);--RAMC

signal carry\_out8 : STD\_LOGIC\_VECTOR (2 downto 0);--MBRC

signal carry\_out9 : STD\_LOGIC\_VECTOR (1 downto 0);--REGAC

signal carry\_out10 : STD\_LOGIC\_VECTOR (2 downto 0);--ALUC

signal carry\_out11 : STD\_LOGIC\_VECTOR (1 downto 0);--REGOC

signal carry\_out12 : STD\_LOGIC\_VECTOR (4 downto 0);--REGSC

signal carry\_out13 : STD\_LOGIC\_VECTOR (1 downto 0);--PCC

signal carry\_out14 : STD\_LOGIC\_VECTOR (1 downto 0);--MARC

signal carry\_out15 : STD\_LOGIC;--OE

signal carry\_out16 : STD\_LOGIC;--EN

signal carry\_out17 : STD\_LOGIC;--DBC

signal carry\_out18 : STD\_LOGIC;--WE

signal carry\_out19 : STD\_LOGIC;--CLK

signal carry\_out20 : STD\_LOGIC\_VECTOR (15 downto 0);--ALU\_REGO

begin

U0:CU port map(

ALUC=>carry\_out10,

RAMC=>carry\_out7,

REGOC=>carry\_out11,

REGAC=>carry\_out9,

PCC=>carry\_out13,

REGSC=>carry\_out12,

MARC=>carry\_out14,

MBRC=>carry\_out8,

IRC=>carry\_out6,

INPUTIR=>carry\_out5,

LED1=>LEDCPU1,

LED2=>LEDCPU2,

CLK=>carry\_out19,

CLOCK=>CLOCKCPU,

RST=>RSTCPU

);

U1:IR port map(

CLK=>carry\_out19,

IRC=>carry\_out6,

INPUT=>DATACPU,

OUTPUTCU=>carry\_out5,

OUTPUTIN=>carry\_out2(15 downto 0),

READY=>READYCPU

);

U2:MAR port map(

CLK=>carry\_out19,

MARC=>carry\_out14,

INPUT=>carry\_out2,

OUTPUT=>ADDRCPU

);

U3:MBR port map(

CLK=>carry\_out19,

MBRC=>carry\_out8,

OUTDATA=>DATACPU,

INDATA=>carry\_out2(15 downto 0)

);

U4:PC port map(

CLK=>carry\_out19,

PCC=>carry\_out13,

OUTPUT=>carry\_out2

);

U5:RAM port map(

CLK=>carry\_out19,

RAMC=>carry\_out7,

EN=>ENCPU,

WE=>WECPU,

OE=>OECPU,

DBC=>DBCCPU

);

U6:REGA port map(

CLK=>carry\_out19,

REGAC=>carry\_out9,

INPUT=>carry\_out2(15 downto 0),

OUTPUT=>carry\_out4

);

U7:REGO port map(

CLK=>carry\_out19,

REGOC=>carry\_out11,

INPUT=>carry\_out20,

OUTPUT=>carry\_out2(15 downto 0)

);

U8:REGS port map(

CLK=>carry\_out19,

REGSC=>carry\_out12,

DATA=>carry\_out2(15 downto 0),

led=>LEDCPU3

);

U9:ALU port map(

CLK=>carry\_out19,

ALUC=>carry\_out10,

INPUTa=>carry\_out4,

INPUTb=>carry\_out2(15 downto 0),

OUTPUT=>carry\_out20

);

end Behavioral;

1. 实验结果

LI R1 , 10; 预计能在34时看到1010

LI R2 , 9;预计能在34时看到1001

LI R3 , 4;预计能在34时看到0100

ADDU R1 ,R2 ,R4;预计能在33时看到10011

SW R1 ,R2 ,R4;预计能在50时，在数据线上看到1011，地址线上看到0100…1010

SUBU R4 ,R3 ,R4;预计能在33时看到1111

SLL R4 ,R4 ,1;预计能在33时看到11110

NOT R4 ,R4;预计能在39时看到1111111111100001

LW R1 ,R2 ,R5;预计能在45时，在数据线上看到10011，在地址线上看到01…1010

实验结果与预期完全一致，说明CPU正常运行

1. 实验总结

本实验16位cpu设计是重中之重。

本次设计实验让我十分透彻的了解到了CPU的构造、工作原理，并且再一次加深了对VHDL的理解。CPU的设计非常复杂，但是我在查过资料后，总结出设计流程。确定指令集架构->确定指令周期->确定机器指令->CPU结构图->指令流程图（表）->基本元件实现->CU实现->元件例化->测试。

本实验cpu设计与常识有些不符的地方在内时钟与外时钟。该设计主要是因为外时钟为手动按钮，为避免按键抖动，才采用内外两时钟的设计。虽然这个设计避免了一些问题，但是从理论上可以看到CPU的性能大幅下降。另外还有IR与ROM的设计。原本应该将指令先存入RAM，CPU初始化后就从内存读取指令再进行操作，但是限于能力，查资料与尝试时间过长，最后改为了指令存入ROM的设计，为了减少对整体代码的修改，就将ROM放入了IR内。