中山大學
SUN YAT-SEN UNIVERSITY

# 《计算机组成原理实验》

# 实验报告

## （实验二）

| | | |
|---|---|---|
| **学 院 名 称** | ： | 数据科学与计算机学院 |
| **专业（班级）** | ： | 16 计算机类 4 班 |
| **学 生 姓 名** | ： | 郑映雪 |
| **学　　号** | ： | 16337327 |
| **时　　间** | ： | 2017　年　11　月　22　日 |

成 绩 ：

# 实验二 ： 单周期CPU设计与实现

## 一. 实验目的

1. 掌握单周期CPU数据通路图的构成、原理及其设计方法；
2. 掌握单周期CPU的实现方法，代码实现方法；
3. 认识和掌握指令与CPU的关系；
4. 掌握测试单周期CPU的方法；
5. 掌握单周期CPU的实现方法。

## 二. 实验内容

设计一个单周期 CPU，该 CPU 至少能实现以下指令功能操作。指令与格式如下:

**==> 算术运算指令**

（1）add rd , rs, rt

| | | | | |
|---|---|---|---|---|
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | reserved |

功能：rd←rs + rt。reserved 为预留部分，即未用，一般填“0”。

（2）addi rt , rs ,immediate

| | | | |
|---|---|---|---|
| 000001 | rs(5 位) | rt(5 位) | immediate(16 位) |

功能：rt←rs + (sign-extend)immediate；immediate 符号扩展再参加“加”运算。

（3）sub rd , rs , rt

| | | | | |
|---|---|---|---|---|
| 000010 | rs(5 位) | rt(5 位) | rd(5 位) | reserved |

功能：rd←rs - rt

**==> 逻辑运算指令**

（4）ori rt , rs ,immediate

| | | | |
|---|---|---|---|
| 010000 | rs(5 位) | rt(5 位) | immediate(16 位) |

功能：rt←rs | (zero-extend)immediate；immediate 做“0”扩展再参加“或”运算。

（5）and rd , rs , rt

| | | | | |
|---|---|---|---|---|
| 010001 | rs(5 位) | rt(5 位) | rd(5 位) | reserved |

功能：rd←rs & rt；逻辑与运算。

（6）or rd , rs , rt

| | | | | |
|---|---|---|---|---|
| 010010 | rs(5 位) | rt(5 位) | rd(5 位) | reserved |

功能：rd←rs | rt；逻辑或运算。

**==>移位指令**

（7）sll　rd, rt,sa

| | | | | | |
|---|---|---|---|---|---|
| 011000 | 未用 | rt(5 位) | rd(5 位) | sa | reserved |

功能：rd<－rt<<(zero-extend)sa，左移 sa 位 ，(zero-extend)sa

**==>比较指令**

（8） slt　rd, rs, rt　 带符号数

| | | | | |
|---|---|---|---|---|
| 011100 | rs(5 位) | rt(5 位) | rd(5 位) | reserved |

功能：if (rs<rt)　rd =1 else　rd=0，具体请看表 2 ALU 运算功能表，带符号

**==> 存储器读/写指令**

（9）sw　rt ,immediate(rs) 写存储器

| | | | |
|---|---|---|---|
| 100110 | rs(5 位) | rt(5 位) | immediate(16 位) |

功能：memory[rs+ (sign-extend)immediate]←rt；immediate 符号扩展再相加。即将 rt 寄存器的内容保存到 rs 寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中。

（10) lw　rt , immediate(rs) 读存储器

| | | | |
|---|---|---|---|
| 100111 | rs(5 位) | rt(5 位) | immediate(16 位) |

功能：rt ← memory[rs + (sign-extend)immediate]；immediate 符号扩展再相加。即读取 rs 寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中的数，然后保存到 rt 寄存器中。

**==> 分支指令**

（11）beq　rs,rt,immediate

| | | | |
|---|---|---|---|
| 110000 | rs(5 位) | rt(5 位) | immediate(16 位) |

功能：if(rs=rt) pc←pc + 4 + (sign-extend)immediate <<2　else pc ←pc + 4

（12）bne　rs,rt,immediate

| | | | |
|---|---|---|---|
| 110001 | rs(5 位) | rt(5 位) | immediate |

功能：if(rs!=rt) pc←pc + 4 + (sign-extend)immediate <<2　else pc ←pc + 4

特别说明：与 beq 不同点是，不等时转移，相等时顺序执行。

（13）bgtz　rs,immediate

| | | | |
|---|---|---|---|
| 110010 | rs(5 位) | 00000 | immediate |

功能：if(rs>0) pc←pc + 4 + (sign-extend)immediate <<2　else pc ←pc + 4

**==>跳转指令**

（14）j　addr

| | |
|---|---|
| 111000 | addr[27..2] |

功能：pc <－{(pc+4)[31..28],addr[27..2],0,0}，无条件跳转。

**==> 停机指令**

（15）halt

| | |
|---|---|
| 111111 | 00000000000000000000000000(26 位) |

功能：停机；不改变 PC 的值，PC 保持不变。

## 三. 实验原理

### 1、CPU设计原理:

#### ①单周期CPU：

单周期 CPU 指的是一条指令的执行在一个时钟周期内完成，然后开始下一条指令的执行，即一条指令用一个时钟周期完成。电平从低到高变化的瞬间称为时钟上升沿，两个相邻时钟上升沿之间的时间间隔称为一个时钟周期。时钟周期一般也称振荡周期（如果晶振的输出没有经过分频就直接作为 CPU 的工作时钟，则时钟周期就等于振荡周期。若振荡周期经二分频后形成时钟脉冲信号作为 CPU 的工作时钟，这样，时钟周期就是振荡周期的两倍。）

#### ②CPU 处理指令的步骤：

(1) 取指令(IF)：根据程序计数器 PC 中的指令地址，从存储器中取出一条指令，同时，PC 根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入 PC，当然得到的“地址”需要做些变换才送入 PC。

(2) 指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3) 指令执行(EXE)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

(4) 存储器访问(MEM)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

(5) 结果写回(WB)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

单周期 CPU，是在一个时钟周期内完成这五个阶段的处理。

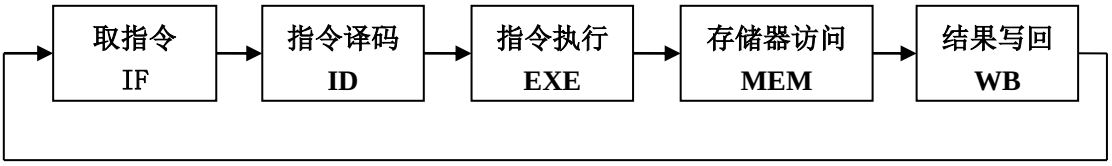



图 1　单周期 CPU 指令处理过程

#### ③MIPS 指令的三种格式

**R 类型：**

| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |
|---|---|---|---|---|---|
| op | rs | rt | rd | sa | funct |
| 6 位 | 5 位 | 5 位 | 5 位 | 5 位 | 6 位 |

**I 类型：**

| 31 26 | 25 21 | 20 16 | 15 0 |
|---|---|---|---|
| op | rs | rt | immediate |
| 6 位 | 5 位 | 5 位 | 16 位 |

**J 类型：**

| 31 26 | 25 0 |
|---|---|
| op | address |
| 6 位 | 26 位 |

其中，

op：为操作码；

rs：只读。为第 1 个源操作数寄存器，寄存器地址（编号）是 00000~11111，00~1F；

rt：可读可写。为第 2 个源操作数寄存器，或目的操作数寄存器，寄存器地址（同上）；

rd：只写。为目的操作数寄存器，寄存器地址（同上）；

sa：为位移量（shift amt），移位指令用于指定移多少位；

funct：为功能码，在寄存器类型指令中（R 类型）用来指定指令的功能与操作码配合使用；

immediate：为 16 位立即数，用作无符号的逻辑操作数、有符号的算术操作数、数据加载（Load）/数据保存（Store）指令的数据地址字节偏移量和分支指令中相对程序计数器（PC）的有符号偏移量；

address：为地址。

**④单周期 CPU 的数据通路**

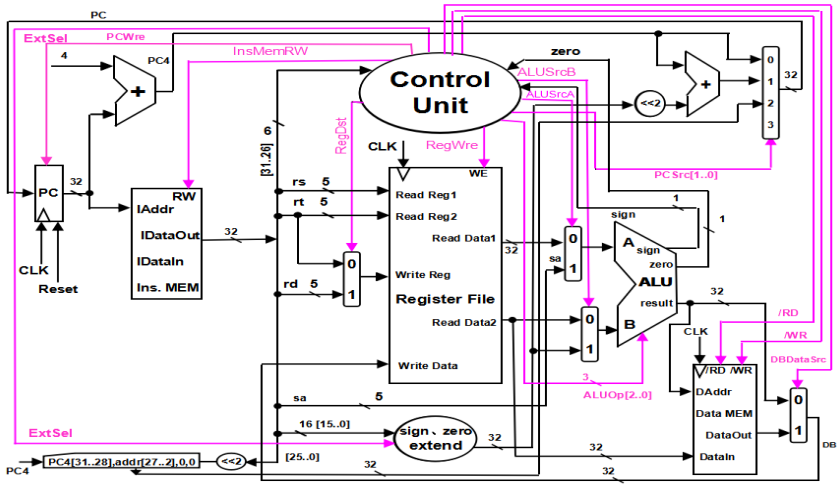


**图 2　单周期 CPU 数据通路和控制线路图**

图 2 是一个简单的基本上能够在单周期 CPU 上完成所要求设计的指令功能的数据通路和必要的控制线路图。其中指令和数据各存储在不同存储器中，即有指令存储器和数据存储器。访问存储器时，先给出内存地址，然后由读或写信号控制操作。对于寄存器组，先给出寄存器地址，读操作时，输出端就直接输出相应数据；而在写操作时，在 WE 使能信号为 1 时，在时钟边沿触发将数据写入寄存器。图中控制信号作用如表 1 所示，表 2 是 ALU 运算功能表。

**表 1 控制信号的作用**

| 控制信号名 | 状态“0” | 状态“1” |
|---|---|---|
| Reset | 初始化 PC 为 0 | PC 接收新地址 |
| PCWre | PC 不更改，相关指令：halt | PC 更改，相关指令：除指令 halt 外 |
| ALUSrcA | 来自寄存器堆 data1 输出，相关指令：add、sub、addi、or、and、ori、beq、bne、bgtz、slt、sw、lw | 来自移位数 sa，同时，进行(zero-extend)sa，即 {{27{0}},sa}，相关指令：sll |
| ALUSrcB | 来自寄存器堆 data2 输出，相关指令：add、sub、or、and、sll、slt、beq、bne、bgtz | 来自 sign 或 zero 扩展的立即数，相关指令：addi、ori、sw、lw |
| DBDataSrc | 来自 ALU 运算结果的输出，相关指令：add、addi、sub、ori、or、and、slt、sll | 来自数据存储器（Data MEM）的输出，相关指令：lw |
| RegWre | 无写寄存器组寄存器，相关指令：beq、bne、bgtz、sw、halt、j | 寄存器组写使能，相关指令：add、addi、sub、ori、or、and、slt、sll、lw |
| InsMemRW | 读指令存储器 | 写指令存储器(Ins. Data) |
| /RD | 读数据存储器，相关指令：lw | 输出高阻态 |
| /WR | 写数据存储器，相关指令：sw | 无操作 |
| RegDst | 写寄存器组寄存器的地址，来自 rt 字段，相关指令：addi、ori、lw | 写寄存器组寄存器的地址，来自 rd 字段，相关指令：add、sub、and、or、slt、sll |
| ExtSel | (zero-extend)immediate(0 扩展)，相关指令：ori | (sign-extend)immediate (符号扩展)，相关指令：addi、sw、lw、bne、bne、bgtz |
| PCSrc[1..0] | 00：pc<－pc+4，相关指令：add、addi、sub、or、ori、and、slt、sll、sw、lw、beq(zero=0)、bne(zero=1)、bgtz(sign=1，或 zero=1);<br>01：pc<－pc+4+(sign-extend)immediate，相关指令：beq(zero=1)、bne(zero=0)、bgtz(sign=0，zero=0);<br>10：pc<－{(pc+4)[31..28],addr[27..2],0,0}，相关指令：j;<br>11：未用 | |
| ALUOp[2..0] | ALU 8 种运算功能选择(000-111)，看功能表 | |

**相关部件及引脚说明：**

**Instruction Memory：指令存储器，**

Iaddr，指令存储器地址输入端口

IDataIn，指令存储器数据输入端口（指令代码输入端口）

IDataOut，指令存储器数据输出端口（指令代码输出端口）

RW，指令存储器读写控制信号，为 0 写，为 1 读

**Data Memory：数据存储器，**

Daddr，数据存储器地址输入端口

DataIn，数据存储器数据输入端口

DataOut，数据存储器数据输出端口

/RD，数据存储器读控制信号，为 0 读

/WR，数据存储器写控制信号，为 0 写

**Register File：寄存器组**

Read Reg1，rs 寄存器地址输入端口

Read Reg2，rt 寄存器地址输入端口

Write Reg，将数据写入的寄存器端口，其地址来源 rt 或 rd 字段

Write Data，写入寄存器的数据输入端口

Read Data1，rs 寄存器数据输出端口

Read Data2，rt 寄存器数据输出端口

WE，写使能信号，为 1 时，在时钟边沿触发写入

**ALU： 算术逻辑单元**

result，ALU 运算结果

zero，运算结果标志，结果为 0，则 zero=1；否则 zero=0

sign，运算结果标志，结果最高位为 0，则 sign=0，正数；否则，sign=1，负数

**表 2 ALU 运算功能表**

| ALUOp[2..0] | 功能 | 描述 |
|---|---|---|
| 000 | Y = A + B | 加 |
| 001 | Y = A － B | 减 |
| 010 | Y = B<<A | B 左移 A 位 |
| 011 | Y = A ∨ B | 或 |
| 100 | Y = A ∧ B | 与 |
| 101 | Y= (A<B) ?1: 0 | 比较 A 与 B<br>不带符号 |
| 110 | if (A<B &&(A[31] == B[31] ))<br>Y = 1;<br>else if ( A[31] && !B[31] Y = 1;<br>else Y = 0; | 比较 A 与 B<br>带符号 |
| 111 | Y = A ⊕ B | 异或 |

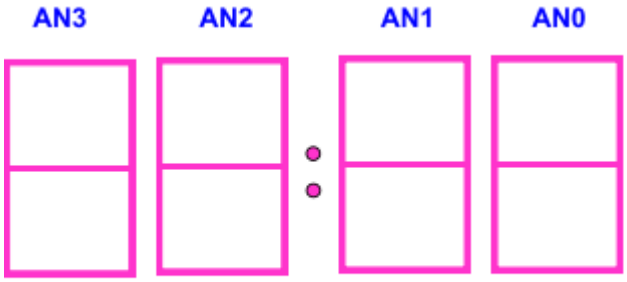
**2、显示原理**

分频：所谓“分频”，就是把输入信号的频率变成成倍地低于输入频率的输出信号。文献资料上所谓用计数器的方法做“分频器”的方法，只是众多方法中的一种。它的原理是：

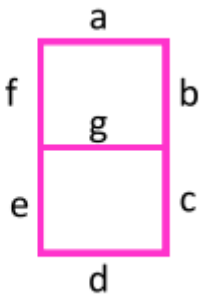
把输入的信号作为计数脉冲，由于计数器的输出端口是按一定规律输出脉冲的，所以对不同的端口输出的信号脉冲，就可以看作是对输入信号的“分频”。至于分频频率是怎样的，由选用的计数器所决定。如 果是十进制的计数器那就是十分频，如果是二进制的计数器那就是二分频，还 有四进制、八进制、十六进制等等。以此类推。

在 Vivado中，clock 在 W5引脚的默认频率是一亿赫兹，则需进行时钟分频至190hz进行扫描显示，一次只显示一位数字，但由于190hz非常大，每次显示的时间差非常小，所以可以有四位数同时显示的效果。

七段阴极数码管中，a~g的暗亮可以组成各个数的表示（如图所示），即可通过控制每个数a~g的暗亮来将二进制数直观地表达为十六进制数。




4个七段数码显示器




**3、按键消抖原理**

按键消抖通常的按键所用开关为机械弹性开关（本实验中手动输入的CLK即是），当机械触点断开、闭合时，由于机械触点的弹性作用，一个按键开关在闭合时不会马上稳定地接通，在断开时也不会一下子断开。因而在闭合及断开的瞬间均伴随有一连串的抖动，为了不产生这种现象而作的措施就是按键消抖。检测出键闭合后执行一个延时程序，让前沿抖动消失后再一次检测键的状态，如果仍保持闭合状态电平，则确认为真正有键按下。延时恰好避开了抖动期。

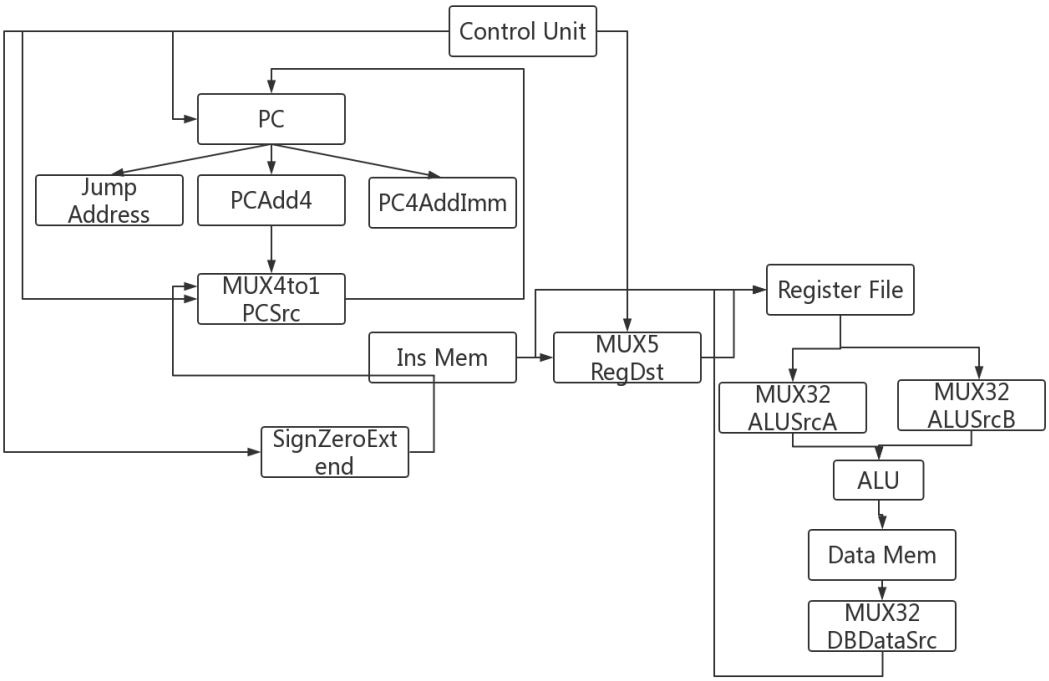
**四.实验器材**

电脑一台，Xilinx Vivado 软件一套，Basys3板一块。

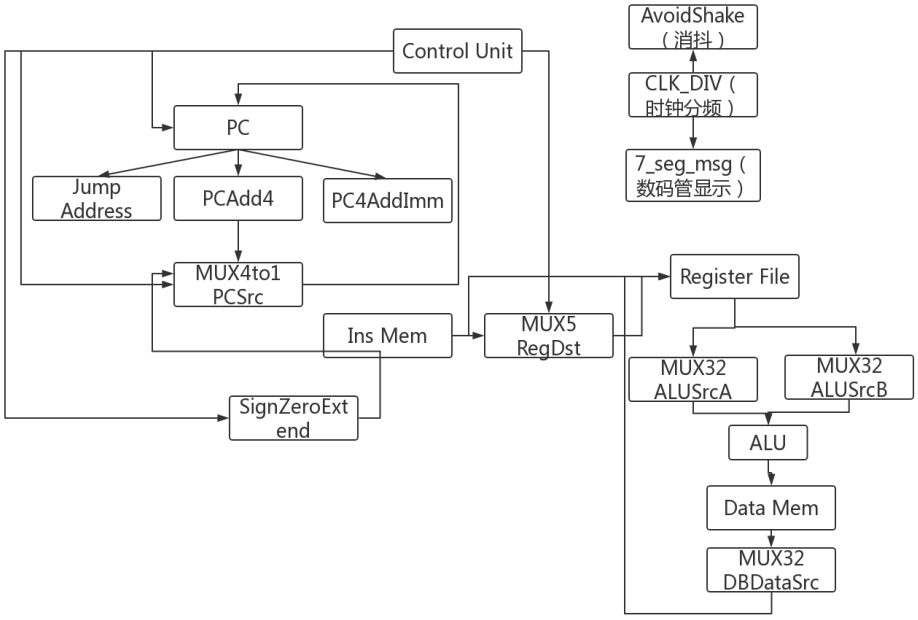
## 五.实验过程与结果

### 1、实验过程：

**①遵循单周期CPU的数据通路图，在顶层模块中分各个模块，设计图如下：**




**②设计按键消抖、七段数码管显示、时钟分频模块。最终设计图为：**



**③填写控制信号与指令关系表：**

注：该表中x的部分，我将根据方便计算的规则设置0或1

| 指令 | Reset | PCWre | ALUSrcA | ALUSrcB | DBDataSrc | RegWre | InsMemRW | RD | WR | RegDst | ExtSel | PCSrc[1:0] | ALUOp[2:0] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| add | 1 | 1 | 0 | 0 | 0 | 1 | x | x | x | 1 | x | 00 | 000 |
| addi | 1 | 1 | 0 | 1 | 0 | 1 | x | x | x | 0 | 1 | 00 | 000 |
| sub | 1 | 1 | 0 | 0 | 0 | 1 | x | x | x | 1 | x | 00 | 001 |
| ori | 1 | 1 | 0 | 1 | 0 | 1 | x | x | x | 0 | 0 | 00 | 011 |
| and | 1 | 1 | 0 | 0 | 0 | 1 | x | x | x | 1 | x | 00 | 100 |
| or | 1 | 1 | 0 | 0 | 0 | 1 | x | x | x | 1 | x | 00 | 011 |
| sll | 1 | 1 | 1 | 0 | 0 | 1 | x | x | x | 1 | x | 00 | 010 |
| slt | 1 | 1 | 0 | 0 | 0 | 1 | x | x | x | 1 | x | 00 | 110 |
| sw | 1 | 1 | 0 | 1 | x | 0 | x | x | 0 | x | 1 | 00 | 000 |
| lw | 1 | 1 | 0 | 1 | 1 | 1 | x | 0 | x | 0 | 1 | 00 | 000 |
| beq | 1 | 1 | 0 | 0 | x | 0 | x | x | x | x | 1 | 00(zero=0) 01(zero=1) | 001 |
| bne | 1 | 1 | 0 | 0 | x | 0 | x | x | x | x | 1 | 00(zero=1) 01(zero=0) | 001 |
| bgtz | 1 | 1 | 0 | 0 | x | 0 | x | x | x | x | 1 | 01(sign=0 zero=0) 00(其它 | 001 |
| j | 1 | 1 | x | x | x | 0 | x | x | x | x | x | 10 | 000 |
| halt | 1 | 0 | x | x | x | 0 | x | x | x | x | x | xx | 000 |

**④各模块设计：**

**a. controlunit：**

输入信号：in，即opcode

zero，零标志位，用于判断运算结果是否为0

sign，符号位，用于判断符号正负

输出信号：reg InsMemRW, PCWre, ExtSel, DBDataSrc, WR, RD, ALUSrcA, ALUSrcB, RegWre, RegDst,[2:0]PCSrc,[1:0]ALUOp。具体释义见实验原理中的信号解释表。

【control unit】关键代码：

```
always @(in or zero or sign)begin
        if (in == 6'b111000) PCSrc = 2'b10;
        else if (in == 6'b110000 && zero == 1 || in == 6'b110001 && zero == 0 || in == 6'b110010 && sign == 0 && zero == 0) PCSrc = 2'b01;
        else PCSrc = 2'b00;

        if (in == 6'b000000 || in == 6'b000001 || in == 6'b100110 || in == 6'b100111) ALUOp = 3'b000;
        else if (in == 6'b000010 || in == 6'b110000 || in == 6'b110001 || in == 6'b110010) ALUOp = 3'b001;
        else if (in == 6'b011000) ALUOp = 3'b010;
        else if (in == 6'b010000 || in == 6'b010010) ALUOp = 3'b011;
        else if (in == 6'b010001) ALUOp = 3'b100;
        else if (in == 6'b011100) ALUOp = 3'b110;
        else ALUOp = 3'b111;
```

```
//续上    InsMemRW = 0;
        PCWre = (in == 6'b111111) ? 0 : 1;
        ExtSel = (in == 6'b010000) ? 0 : 1;
        DBDataSrc = (in == 6'b100111) ? 1 : 0;
        WR = (in == 6'b100110) ? 0 : 1;
        RD = (in == 6'b100111) ? 0 : 1;
        ALUSrcA = (in == 6'b011000) ? 1 : 0;
        ALUSrcB = (in == 6'b000001 || in == 6'b010000 || in == 6'b100110 || in == 6'b100111) ? 1 : 0;
        RegWre = (in == 6'b110000 || in == 6'b110001 || in == 6'b110010 || in == 6'b100110 || in == 6'b111111 || in == 6'b111000) ? 0 : 1;
        RegDst = (in == 6'b000001 || in == 6'b010000 || in == 6'b100111) ? 0 : 1;
end
```

**b.PC相关模块，包括PC+4模块，PC+imm模块，Jump模块和选择MUX模块。**

输入信号：pcin，上一条PC的值。

in（用于jump模块），指令的25~0位。

imm（用于PC+imm模块），指令的immediate部分。

RST，用于PC的清零。

PCWre,用于PC的停止。

[1:0]PCSrc，用于Pcmux的选择。

CLK，时钟下降沿触发。

输出信号:Pcout，下一条PC的值。

【PC模块】相关代码：

```
always @(posedge CLK or negedge Reset) begin     //PC 总模块
    if (Reset == 0) PCout <= 0;
    else if (PCWre) PCout <= PCin;
end
always @(PC4 or imm) begin       //PC+imm 模块：
    PC4_imm = PC4 + (imm << 2);
end
always @(PCin) begin       //PC+4 模块
   PCout = PCin + 4;
end
always @(PC4 or PC4_imm or jump or PCSrc) begin       //选择模块
    if (PCSrc == 0) PCout = PC4;
    else if (PCSrc == 1) PCout = PC4_imm;
    else PCout = jump;
end
```

**c.InsMem模块**

输入信号：[31:0] addr,32位机器码（本实验中从文件中读取）

输出信号：[31:0]dataout，读取出的机器码输送出去。

【InsMem模块】相关代码：

```
reg [7:0] rom [99:0];
initial begin
      $readmemb ("C:/new/test.txt", rom);   //根据文件存放地址适当修改读取地址
end
always @(addr) begin   //为 0，读存储器，大端数据存储模式
            dataOut[31:24] = rom[addr];
            dataOut[23:16] = rom[addr+1];
            dataOut[15:8] = rom[addr+2];
            dataOut[7:0] = rom[addr+3];
end
```

**d.RegFile及其相关选择模块**

输入信号：

CLK,时钟触发。

RST,寄存器清零。

RegWre,是否写寄存器。

RegDst（用于读取选择模块），选择读取哪个寄存器。

[4:0] ReadReg1,读寄存器1地址。

[4:0] ReadReg2,读寄存器2地址。

[4:0] WriteReg,写寄存器。

[31:0] WriteData,往寄存器中写入数据。

输出信号：

[31:0] ReadData1,输出寄存器1的数据。

[31:0] ReadData2，输出寄存器2的数据。

【RegFile及其选择模块】相关代码

```
//输入寄存器类型的选择
    always @(RegDst or rt or rd) begin
        if (RegDst == 1) Write_Reg = rd;
        else Write_Reg = rt;
    end
//寄存器组
reg [31:0] regFile[1:31]; //寄存器必须用 reg 类型
    integer i;
    assign ReadData1 = (ReadReg1 == 0) ? 0 : regFile[ReadReg1]; //读寄存器数据
    assign ReadData2 = (ReadReg2 == 0) ? 0 : regFile[ReadReg2];
    wire clk1;
    always @ (negedge CLK or negedge RST) begin //必须用时钟边沿触发
        if (RST==0) begin
            for(i=1;i<32;i=i+1) begin
                if(i==WriteReg)    regFile[WriteReg] <= WriteData;
                else regFile[i] <= 0;
            end
        end
        else if(RegWre == 1 && WriteReg != 0) // WriteReg != 0，$0 寄存器不能修改
            regFile[WriteReg] <= WriteData; //  写寄存器
    end
```

**e.ALU及其相关选择模块**

输入信号:

[2:0] ALUopcode,根据aluopcode选择相应计算，具体见前表。

[31:0] rega,读入的第一个需要计算的数据。

[31:0] regb,读入的第二个需要计算的数据。

[31:0] Read_Data1,Read_Data2用于读取数据。（用于ALU选择模块）

[4:0] sa,偏移位，用于sll指令。（用于ALUA选择模块）

ALUSrcA,选择是用寄存器还是偏移量作为计算数据。(用于ALUA选择模块)

ALUSrcB,选择是用寄存器还是符号扩展为计算数据。(用于ALUB选择模块)

输出信号:

[31:0] result,计算结果。

zero,零标志位的结果。

sign，符号位的结果。

【ALU及其选择模块】相关代码

```
//第一个计算数的选择
    always @(ALUSrcA or sa or Read_Data1) begin
        if (ALUSrcA == 0) ALUA = Read_Data1;
        else ALUA = {{27{1'b0}}, sa};
    end
//第二个计算数的选择
    always@(ALUSrcB or Read_Data2 or Extend_out) begin
        if (ALUSrcB == 1) ALUB = Extend_out;
        else ALUB = Read_Data2;
    end
//ALU 部分代码
assign zero = (result==0)?1:0;
    assign sign = (result[31] == 1)?1:0;
    always @( ALUopcode or rega or regb ) begin
        case (ALUopcode)
            3'b000 : result = rega + regb;
            3'b001 : result = rega - regb;
            3'b010 : result = regb << rega;
            3'b100 : result = rega & regb;
            3'b011 : result = rega | regb;
            3'b101 : result = (rega < regb)?1:0; //  不带符号比较
            3'b110 : begin //  带符号比较
                if (rega<regb &&(( rega[31] == 0 && regb[31]==0) || (rega[31] == 1 &&
regb[31]==1))) result = 1;
                else if (rega[31] == 0 && regb[31]==1) result = 0;
                else if ( rega[31] == 1 && regb[31]==0) result = 1;
                else result = 0;
                end
            3'b111 : begin
                result = rega ^ regb;
            end
        endcase
    end
```

**f.DataMem及其相关选择模块**

输入信号:

CLK，用于时钟触发。

nRD，判断是否读取存储器。

nWR，判断是否写入存储器

[31:0] address，写入的地址。

[31:0]writeData,存储器的内存单元。

DBDataSrc（用于选择模块），选择输出到DB的数据是ALU还是内存里。

输出信号：

[31:0]dataout，读取出的数据。

[31:0]DB，输出给数据总线的数据。

【DataMem及其选择模块】相关代码

```
//DataMem 模块
    reg [7:0] ram [0:60]; //写寄存器
    assign Dataout[7:0] = (nRD==0)?ram[address + 3]:8'bz;
    assign Dataout[15:8] = (nRD==0)?ram[address + 2]:8'bz;
    assign Dataout[23:16] = (nRD==0)?ram[address + 1]:8'bz;
    assign Dataout[31:24] = (nRD==0)?ram[address ]:8'bz;
    always@(negedge CLK) begin    //读寄存器
        if( nWR==0 ) begin
            ram[address] <= writeData[31:24];
            ram[address+1] <= writeData[23:16];
            ram[address+2] <= writeData[15:8];
            ram[address+3] <= writeData[7:0];
        end
    end


//DB 选择模块
    always @(ALUresult or DataOut or DBDataSrc) begin
        if (DBDataSrc == 1) DB = DataOut;
        else DB = ALUresult;
    end
```

**g.符号扩展模块**

输入信号：

[15:0] in,指令低16位。

ExtSel,选择是符号扩展还是0扩展

输出信号：

[31:0]out，输出的扩展后的数。

【符号扩展】相关代码

```
always @(in or ExtSel) begin
    if (ExtSel == 0) out =   {{16{1'b0}}, in};
    else if (in[15] == 0) out = {{16{1'b0}}, in};
    else out = {{16{1'b1}}, in};
end
```

另附：

**消抖模块代码：**

```
reg tmp_in[2:0];
always @(posedge clk2) begin
    tmp_in[0] <= in;
    tmp_in[1] <= tmp_in[0];
    tmp_in[2] <= tmp_in[1];
end
assign out = (RST==1)? (tmp_in[1] | tmp_in[2]) :1 ;
```

**时钟分频代码：**

```
reg [40:0] q;
always @ (posedge clk) begin
    q <= q + 1;
end
assign clk190 = q[17]; // 190 Hz 用于扫描输出
```

**七段码输出代码**

```
reg [1:0] s;
reg [3:0] digit;
wire [3:0] aen;
assign aen = 4'b1111; //一共四位数码管，每一位显示什么数字
always @ (* )
    case (s)
        0: //数码管最低位
        begin
            case (change)
                2'b00:digit=pcnext2;
                2'b01:digit=rsdata2;
                2'b10:digit=rtdata2;
                2'b11:digit=dbdata2;
            endcase
```

```
//接上          end
                1:
                begin
                    case (change)
                        2'b00:digit=pcnext1;
                        2'b01:digit=rsdata1;
                        2'b10:digit=rtdata1;
                        2'b11:digit=dbdata1;
                    endcase
                end
                2:
                begin
                    case (change)
                        2'b00:digit=pcpre2;
                        2'b01:digit=rsaddr2;
                        2'b10:digit=rtaddr2;
                        2'b11:digit=result2;
                    endcase
                end
                3:
                begin
                    case (change)
                        2'b00:digit=pcpre1;
                        2'b01:digit=rsaddr1;
                        2'b10:digit=rtaddr1;
                        2'b11:digit=result1;
                    endcase
                end
            endcase
        always @ ( * )
            case (digit)
                4'b0000 : dispcode = 8'b1100_0000; //0 '0'-亮灯，'1'-熄灯
                4'b0001 : dispcode = 8'b1111_1001; //1
                4'b0010 : dispcode = 8'b1010_0100; //2
                4'b0011 : dispcode = 8'b1011_0000; //3
                4'b0100 : dispcode = 8'b1001_1001; //4
                4'b0101 : dispcode = 8'b1001_0010; //5
                4'b0110 : dispcode = 8'b1000_0010; //6
                4'b0111 : dispcode = 8'b1101_1000; //7
                4'b1000 : dispcode = 8'b1000_0000; //8
                4'b1001 : dispcode = 8'b1001_0000; //9
                4'b1010 : dispcode = 8'b1000_1000; //A
                4'b1011 : dispcode = 8'b1000_0011; //b
                4'b1100 : dispcode = 8'b1100_0110; //C
```

```
//接上          4'b1101 : dispcode = 8'b1010_0001; //d
                4'b1110 : dispcode = 8'b1000_0110; //E
                4'b1111 : dispcode = 8'b1000_1110; //F
                default : dispcode = 8'b0000_0000;
        endcase
    always @ (posedge clk190)
        begin //每次  190HZ  的频率闪过就选择亮哪一盏灯
            s<=s+1;
        end

     //确保每个数码管上都有显示数字
    always @ ( *)
    begin
        an = 4'b1111; //共阴极，全部不亮
        if (aen[s] == 1) //控制哪一位亮
            an[s] = 0;
    end
```

**⑤顶层模块设计，调用各个模块即可。**

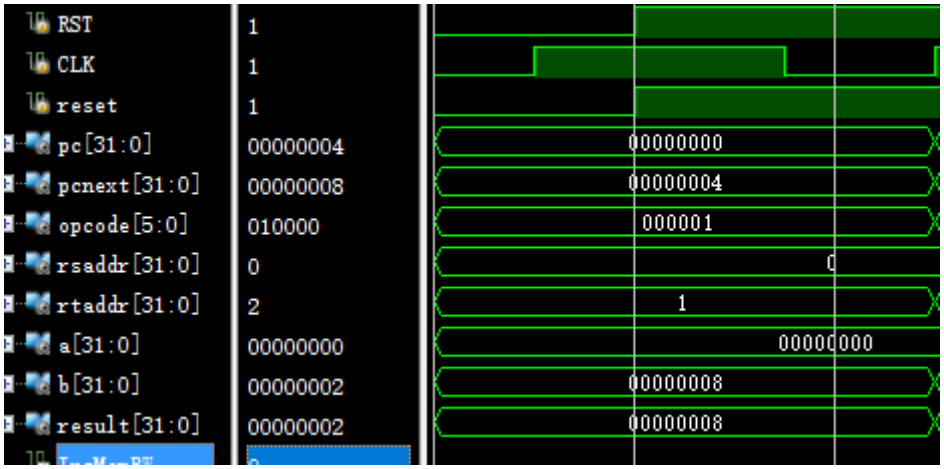
**⑥运行VIVADO，进行仿真、综合和烧录，对比输入机器码的文件观察各个数据。**

**2、实验结果：**

**i.仿真结果：**

波形从上到下依次是 当前pc、下一条pc、指令的opcode、rs地址、rt地址、两个运算数和运算结果。

1.addi  $1,$0,8

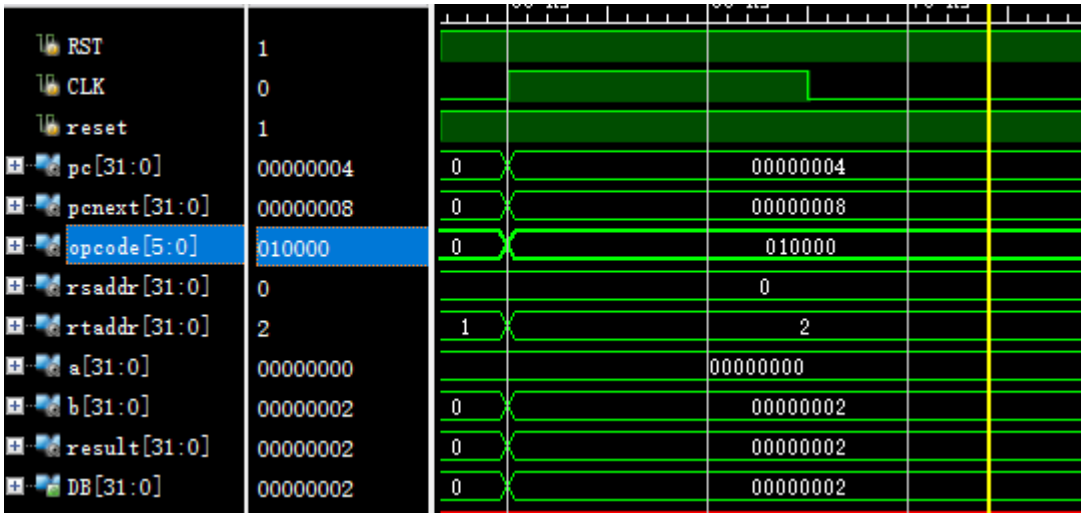


由图可知

当前pc：00　下条pc：04　rs编号：0　rt编号：1　opcode:000001执行addi操作　两个运算数分别为0、8

结果为 8 符合指令，执行正确。

2.ori　$2,$0,2

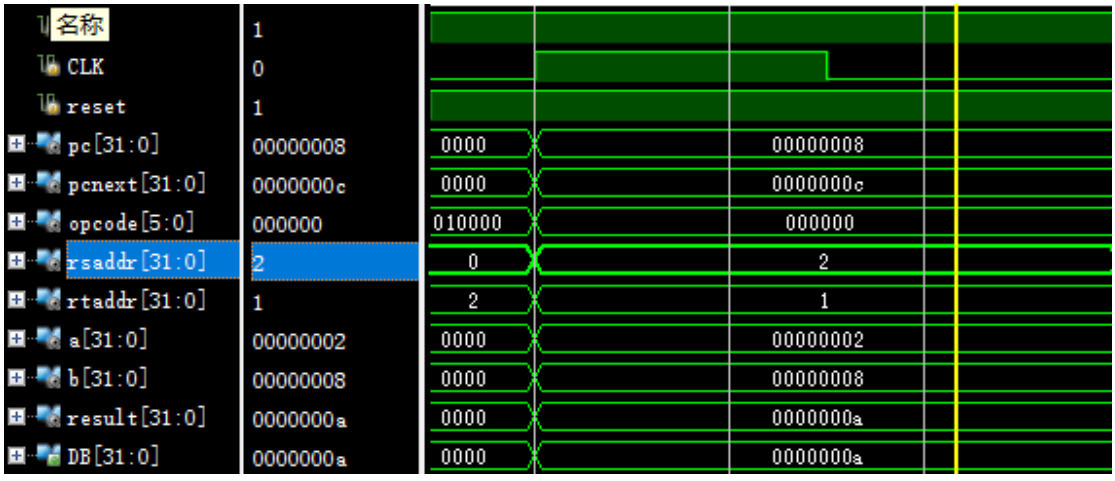



由图可知

当前pc：04　下条pc：08　rs编号：0　rt编号：2　opcode:010000执行ori操作　两个运算数分别为0、2

结果为 2 符合指令，执行正确。

3.add　$3,$2,$1

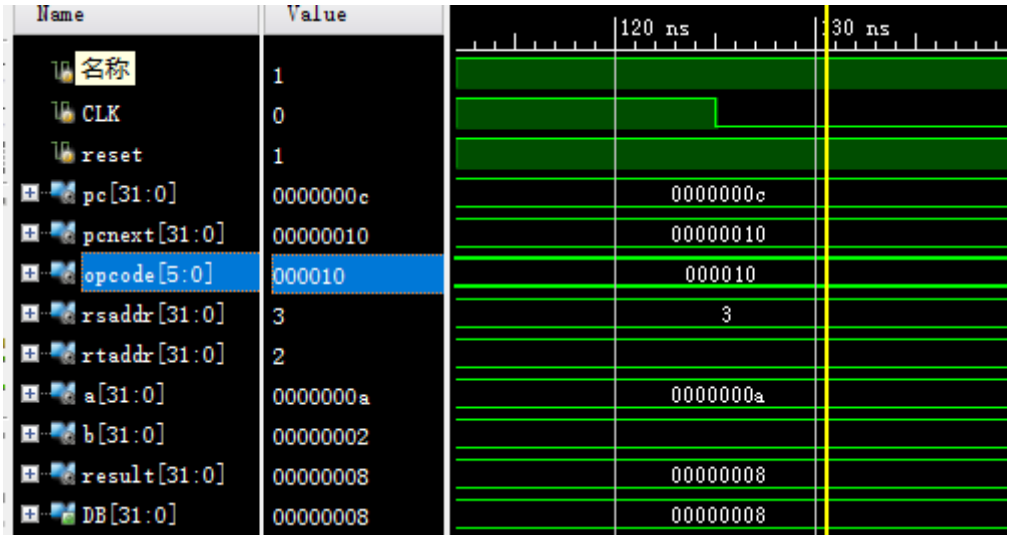



由图可知

当前pc：08　下条pc：0C　rs编号：2　rt编号：1　opcode:000000执行ADD操作两个运算数分别为2 、8

结果为 a 符合指令，执行正确。

4.sub　$5,$3,$2

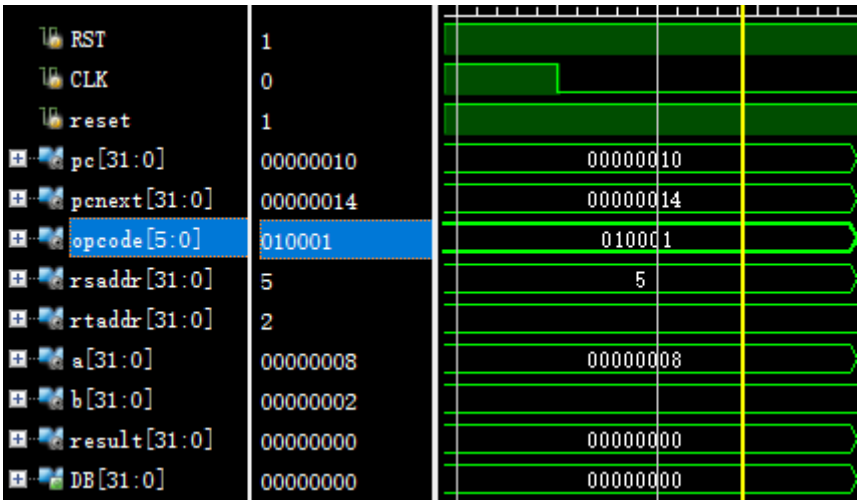



由图可知

当前pc：0C　下条pc：10　rs编号：3　rt编号：2　opcode:000010执行sub操作两个运算数分别为a 、2

结果为 8 符合指令，执行正确。

5.and　$4,$5,$2



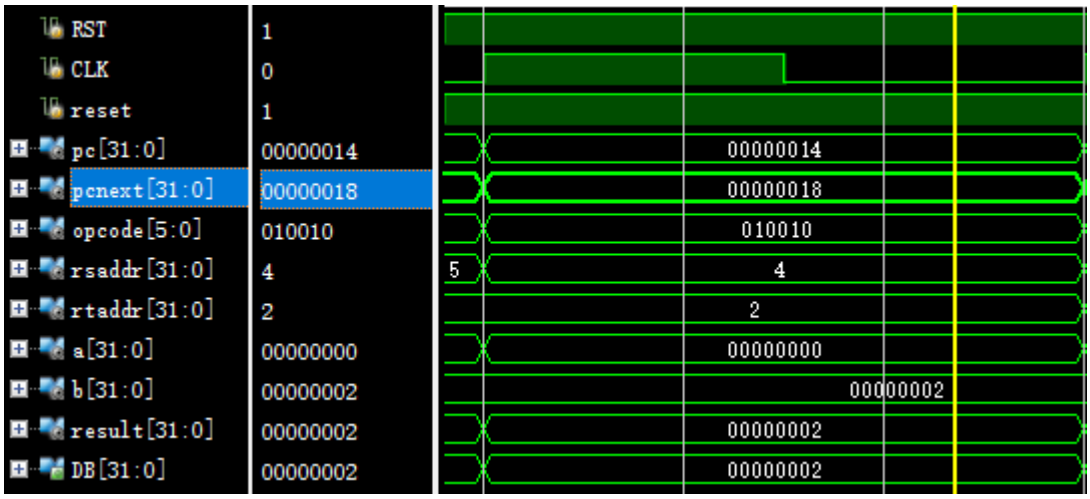

由图可知

当前pc：10　下条pc：14　rs编号：5　rt编号：2　opcode:010001执行and操作

两个运算数分别为8、0

结果为 0 符合指令，执行正确。

6.or $8,$4,$2

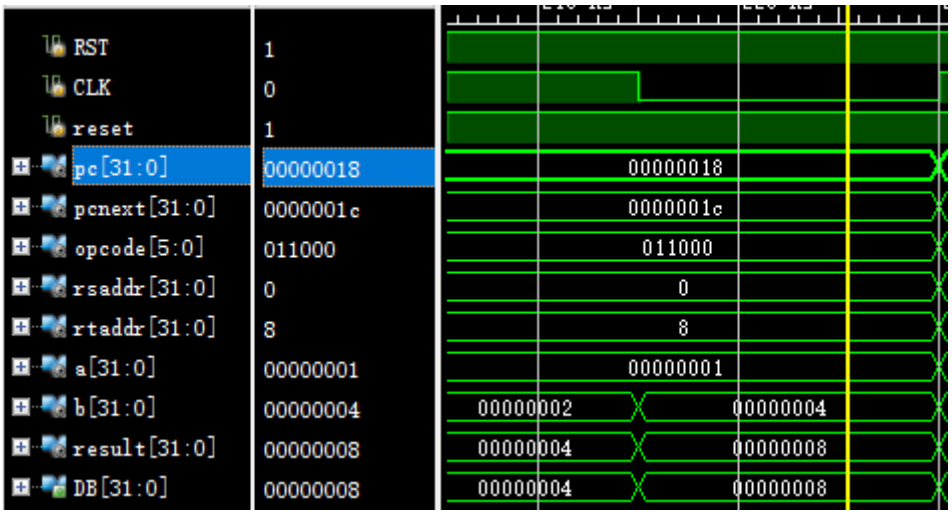



由图可知

当前pc：14 下条pc：18 rs编号：4 rt编号：2 opcode:010010执行or操作 两个运算数分别为0、2

结果为 2 符合指令，执行正确。

7.sll $8,$8,1

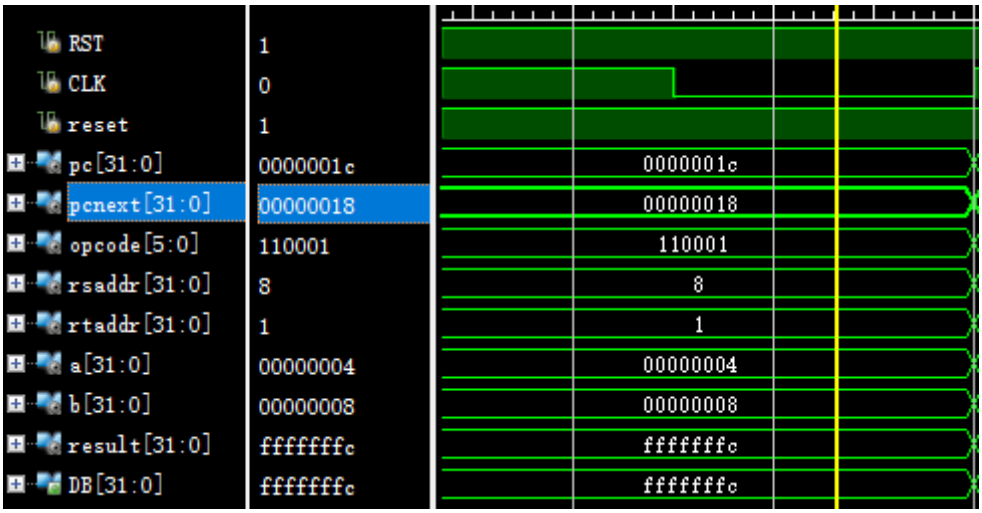



由图可知

当前pc：18 下条pc：1C rs编号：0 rt编号：8 opcode:011000执行sll操作 两个运算数分别为1、2（10）

结果为 4（100） 符合指令，执行正确。

8.bne $8,$1,-2 (≠,转18)

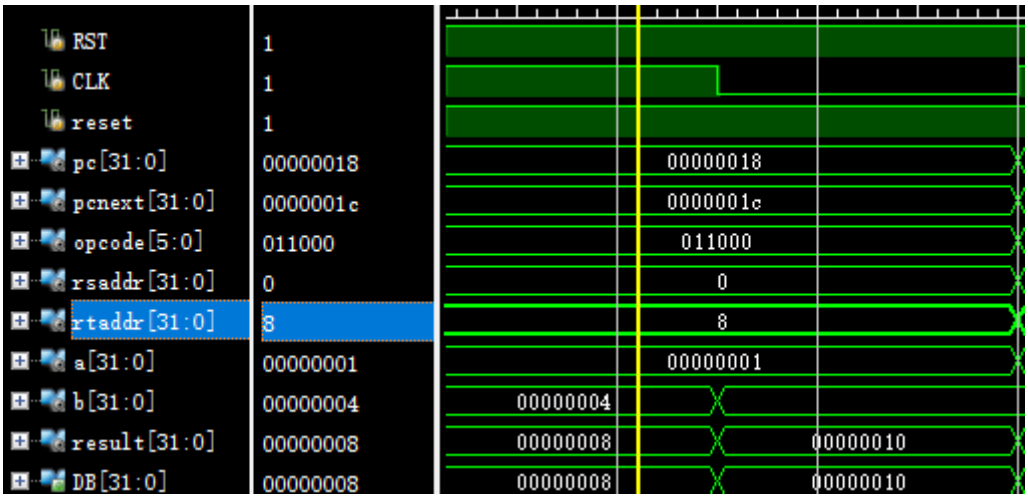



由图可知

当前pc：1C 下条pc：18 rs编号：8 rt编号：1 opcode:110001执行bne操作两个运算数分别为4、8

结果为 -2的补码fffffffc，表示PC即将跳转到18，符合指令，执行正确。

9.sll $8,$8,1

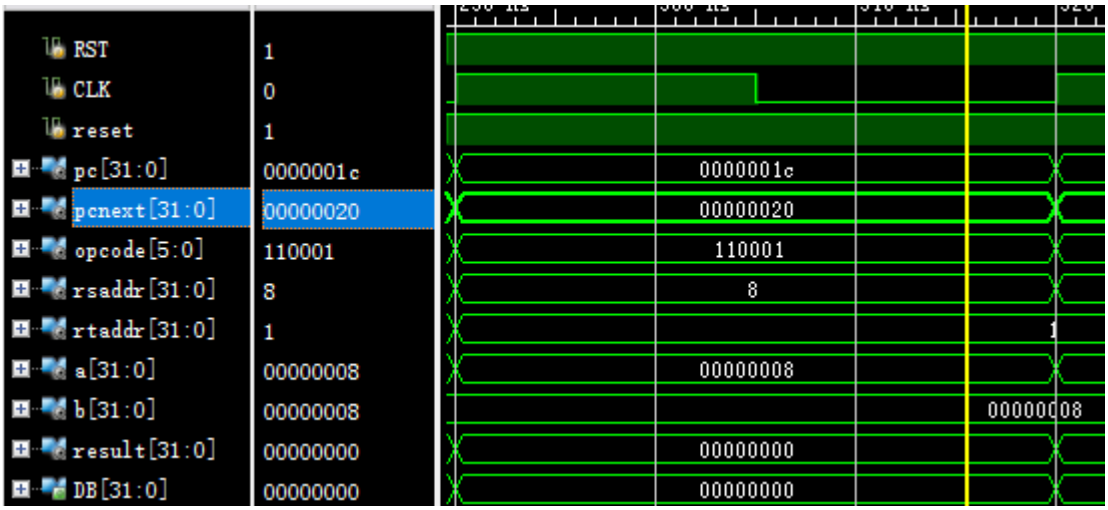



由图可知

当前pc：18 下条pc：1C rs编号：0 rt编号：8 opcode:011000执行sll操作两个运算数分别为1、4（100）

结果为 8（1000） 符合指令，执行正确。

10.bne $8,$1,-2 (≠,转18)

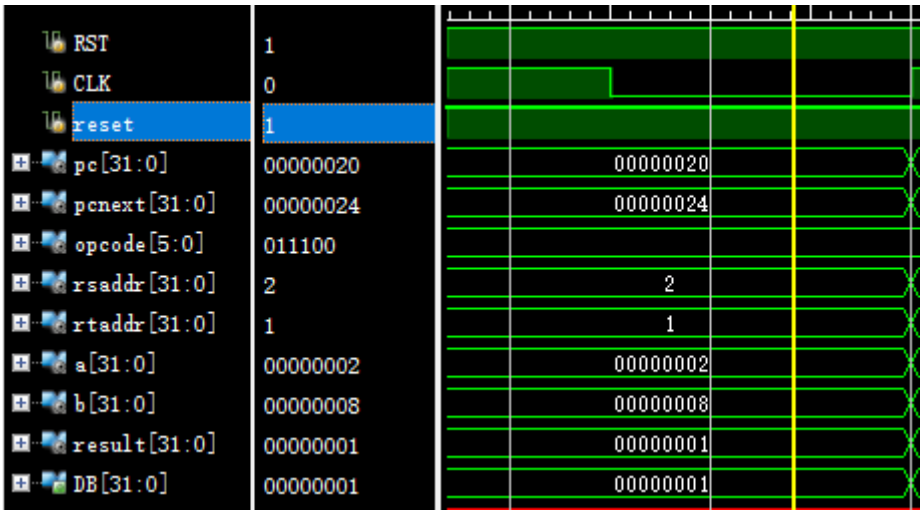



由图可知

当前pc：1C 下条pc：20 rs编号：8 rt编号：1 opcode:110001执行bne操作两个运算数分别为8、8

结果为相等不跳转，符合指令，执行正确。

11. slt $6,$2,$1

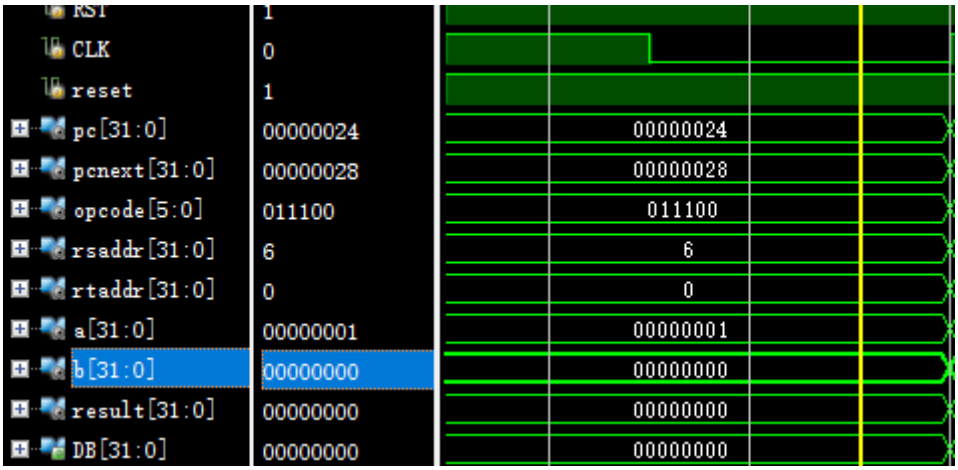



由图可知

当前pc: 20 下条pc: 24 rs编号: 2 rt编号: 1 opcode:011100执行slt操作 两个运算数分别为2、8

结果为小于则置位，运算结果为1，符合指令，执行正确。

12. slt　$7,$6,$0

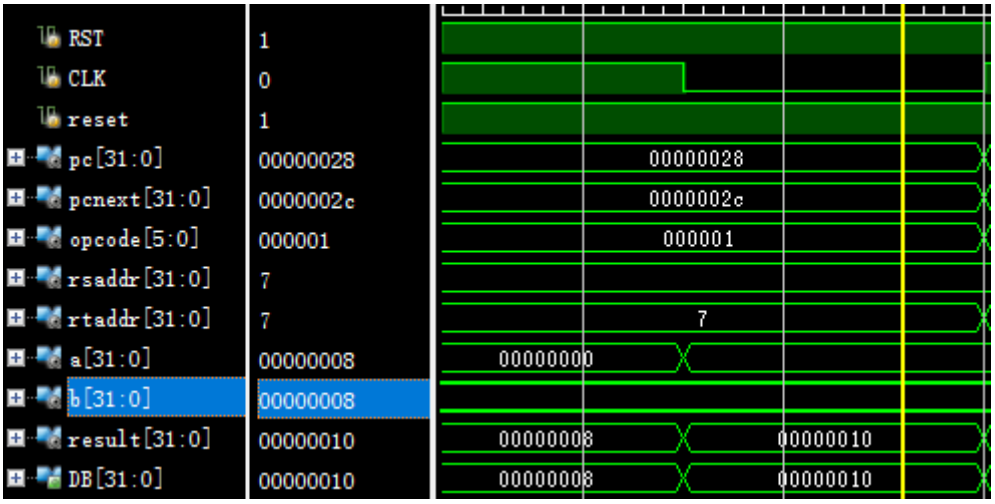



由图可知

当前pc: 24　下条pc: 28　rs编号: 6　rt编号: 0　opcode:011100执行slt操作　两个运算数分别为1、0

结果为大于则不置位，运算结果为0，符合指令，执行正确。

13. addi $7,$7,8

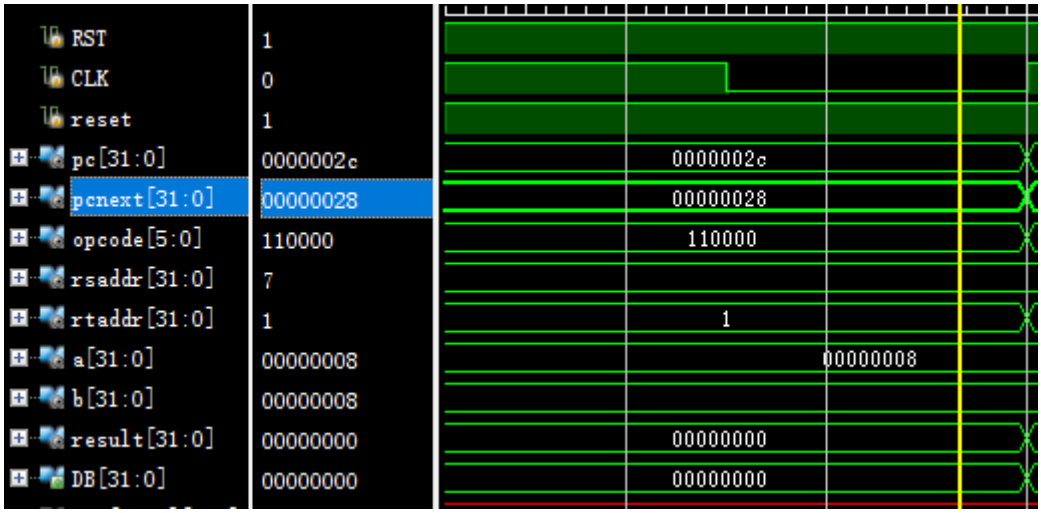



由图可知

当前pc：28　下条pc：2c　rs编号：7　rt编号：7　opcode:000001执行addi操作两个运算数分别为0、8

结果为8，符合指令，执行正确。

14. beq $7,$1,-2 (≠,转28)

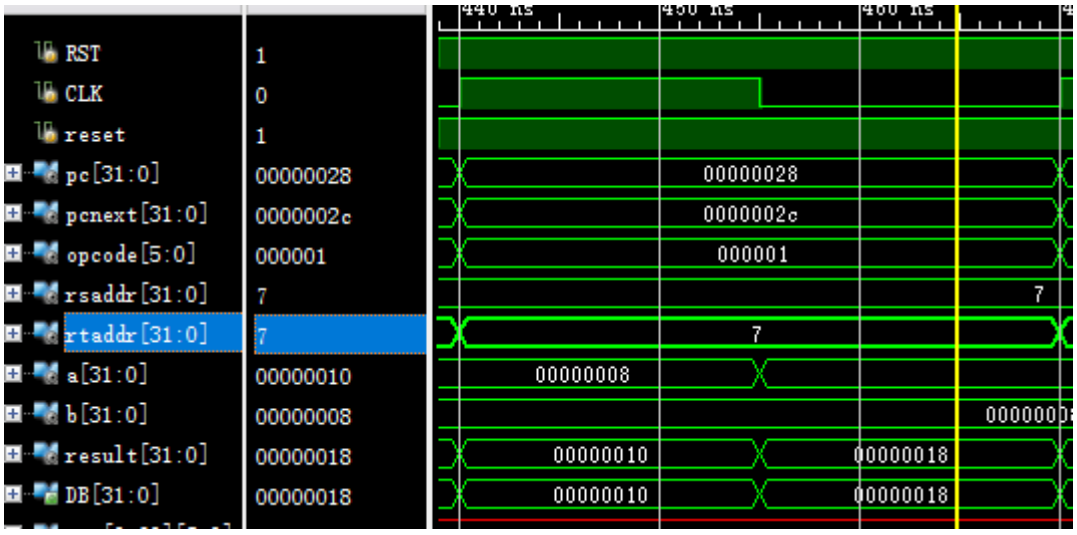



由图可知

当前pc:2c 下条pc:28 rs编号:7 rt编号:1 opcode:110000执行beq操作 两个运算数分别为8、8

相减结果为0，pc跳转至28，符合指令，执行正确。

15. addi $7,$7,8

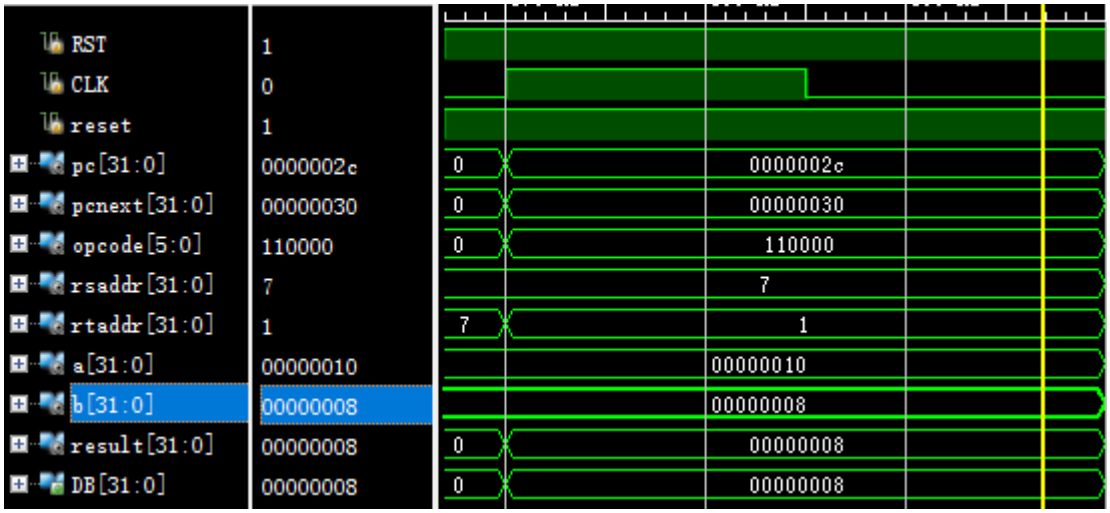



由图可知

当前pc：28 下条pc：2c rs编号：7 rt编号：7 opcode:000001执行addi操作 两个运算数分别为8、8

结果为10（16d），符合指令，执行正确。

16. beq $7,$1,-2 (≠,转28)

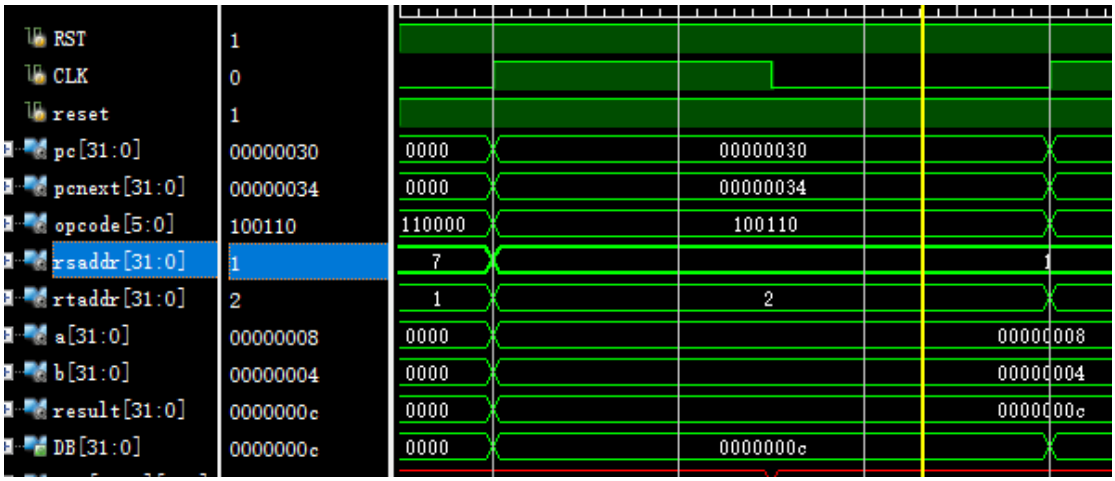



由图可知

当前pc:2c 下条pc:30 rs编号:7 rt编号:1 opcode:110000执行beq操作 两个运算数分别为10、8

相减结果为8，pc不跳转，而是加4，符合指令，执行正确。

17. sw $2,4($1)

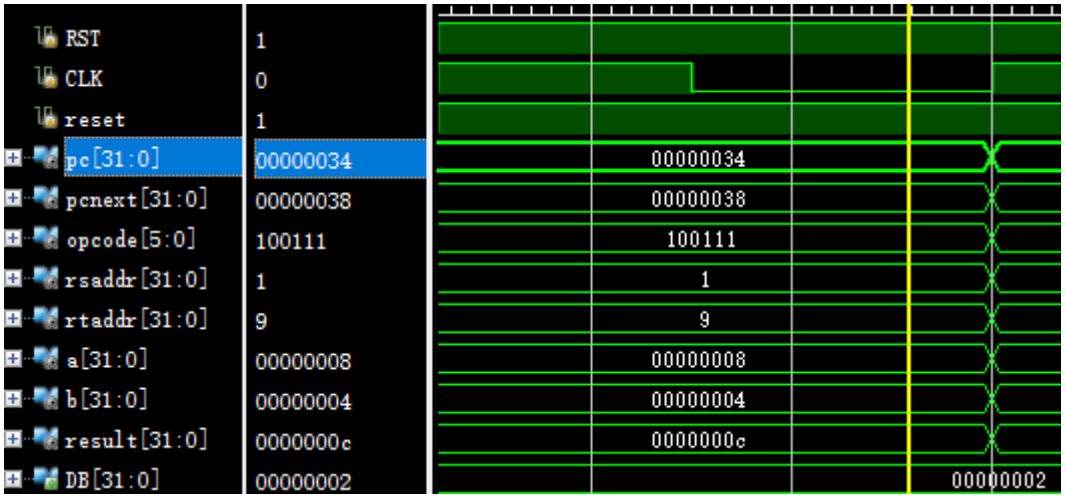



由图可知

当前pc:30 下条pc:34 rs编号:1 rt编号:2 opcode:100110执行sw操作 两个运算数分别为8、4

相加结果为c，将c寄存器的数（0）存入了2号内存单元，符合指令，执行正确。

18. lw $9,4($1)

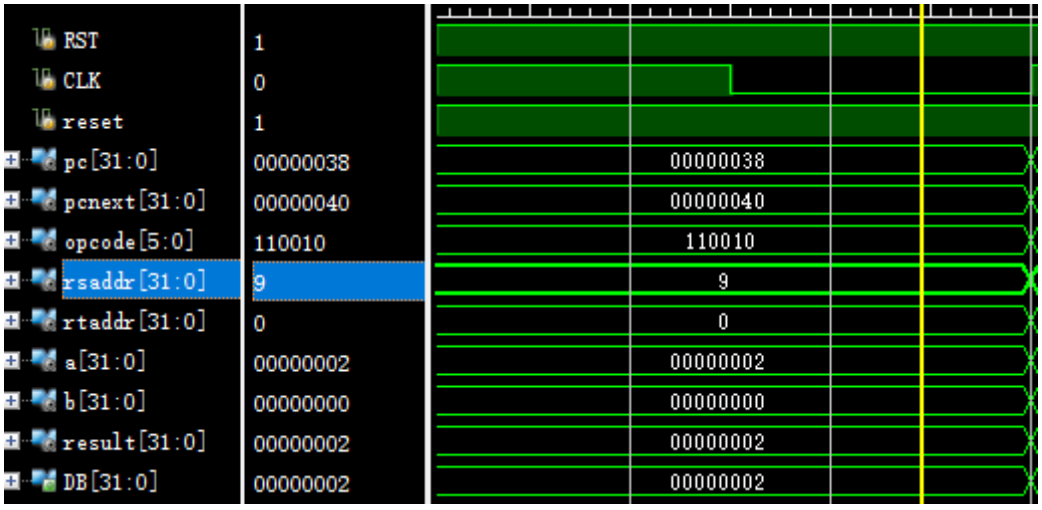



由图可知

当前pc: 34　下条pc: 38　rs编号: 1　rt编号: 9　opcode:100111执行lw操作　两个运算数分别为8、4

相加结果为c，将c单元的数存入了2号寄存器，符合指令，执行正确。

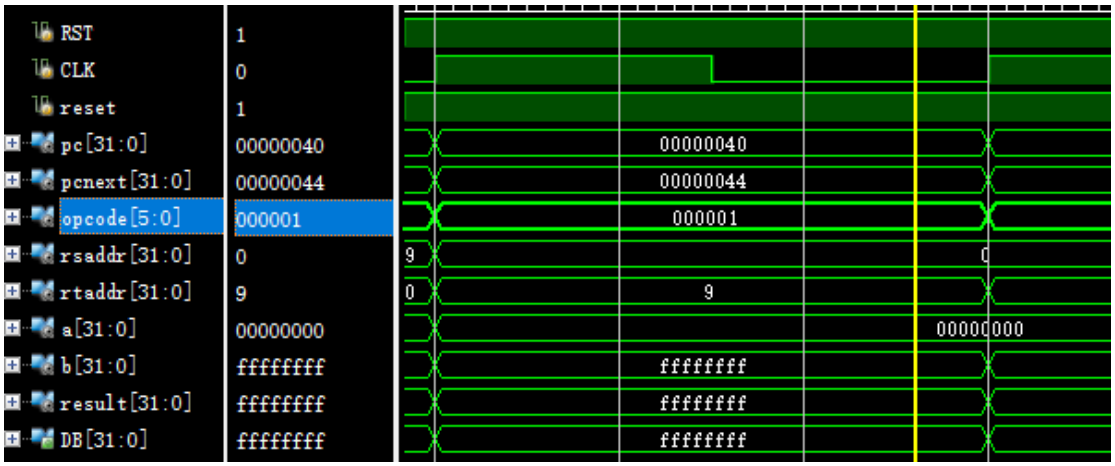
19. bgtz　$9,1 (>0,转40)




由图可知

当前pc：38　下条pc：40　rs编号：9　rt编号：0　opcode:110010执行bgtz操作两个运算数分别为2、0。2>0，故pct跳转至imm值处，符合指令，执行正确。

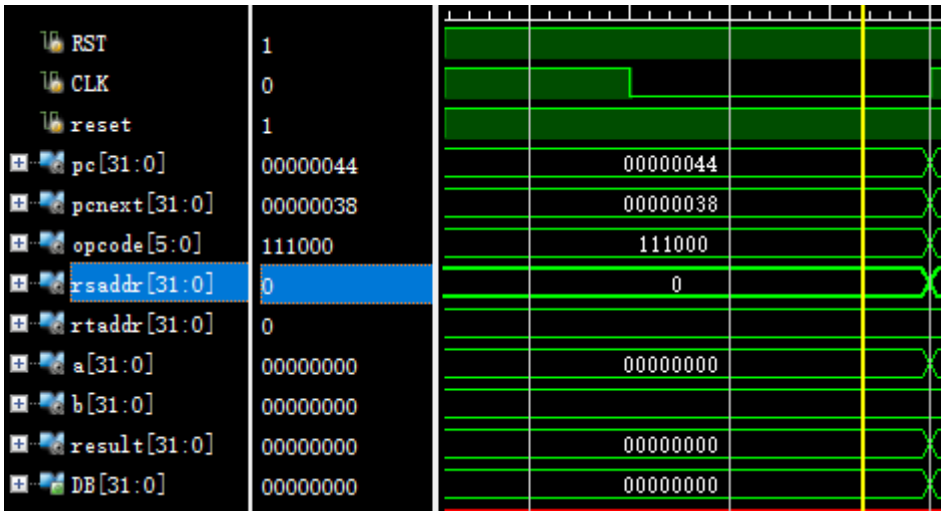
20. addi　$9,$0,-1




由图可知

当前pc：40　下条pc：44　rs编号：0　rt编号：9　opcode:000001执行addi操作两个运算数分别为0，-1（ffffffff为其补码），结果为-1（ffffffff为其补码），符合指令，执行正确。

21.j　0x00000038




由图可知

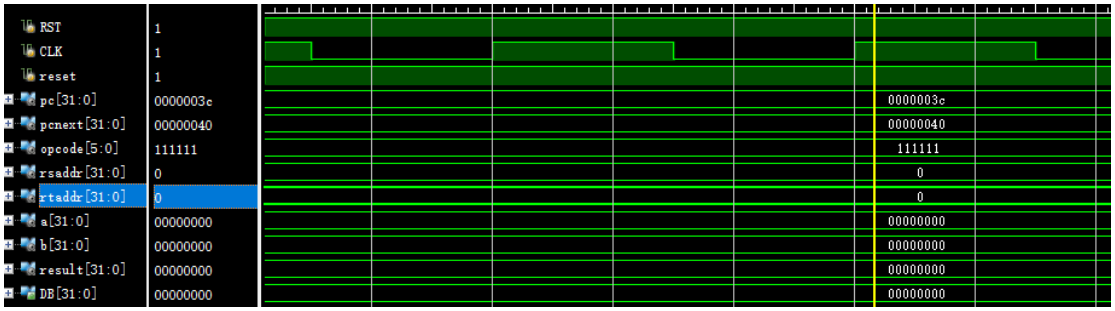
当前pc：44　下条pc：38　rs编号：0　rt编号：0　opcode:111000执行j操作 ，运算数为，PC跳转到了38，符合指令，执行正确。

21. bgtz　$9,1 (>0,转40)


由图可知

当前pc：38　下条pc：3c　rs编号：9　rt编号：0　opcode:110010执行bgtz操作两个运算数分别为-1、0。-1<0，故pc=pc+4，符合指令，执行正确。

23.halt



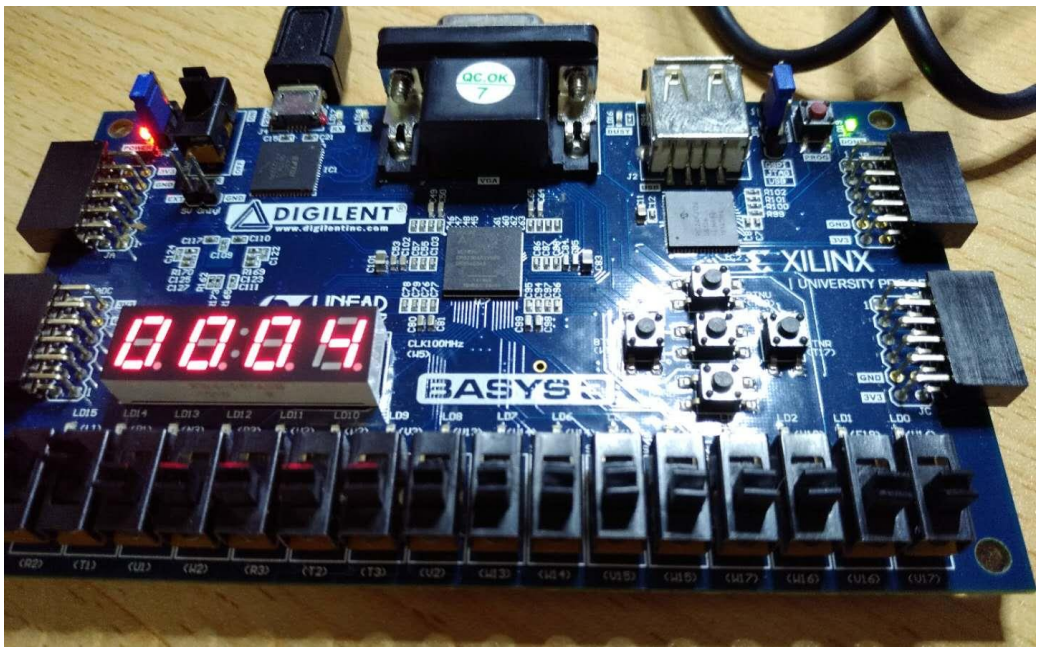

由图可知

当前pc:3c　下条pc:40　rs编号:0　rt编号:0　opcode:111111执行halt操作　两个运算数分别为0、0，PC不再跳转，寄存器不再写入，呈停机状态，符合指令，执行正确。

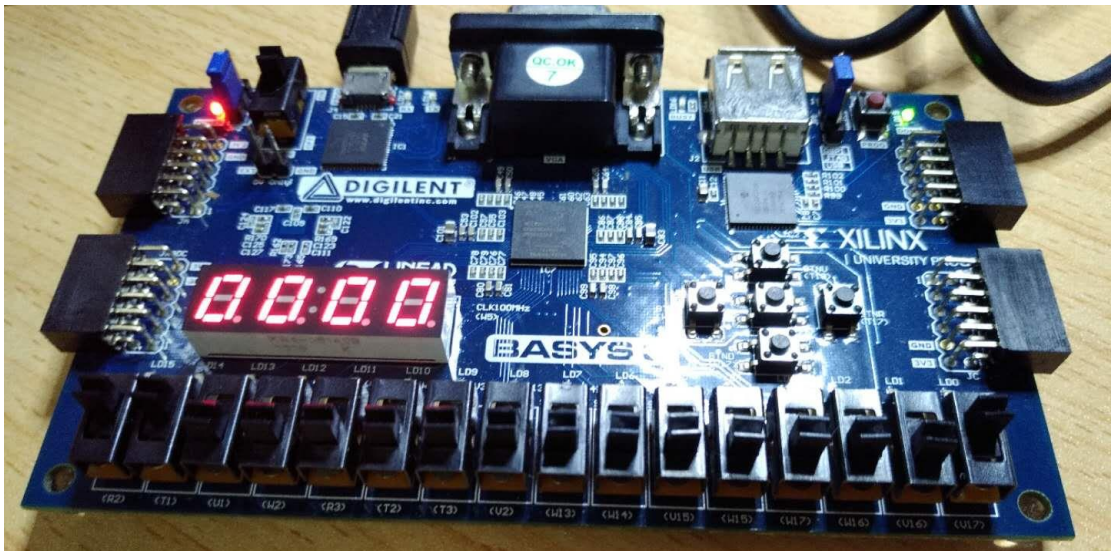
**（ii.）basys3板运行结果（部分）**

由于basys板运行结果有八十多张照片，所以实验报告里只放第一条指令的四个状态。板子的最左两个开关分别控制PC的reset和寄存器的RST，最右两个开关是切换显示的数据。

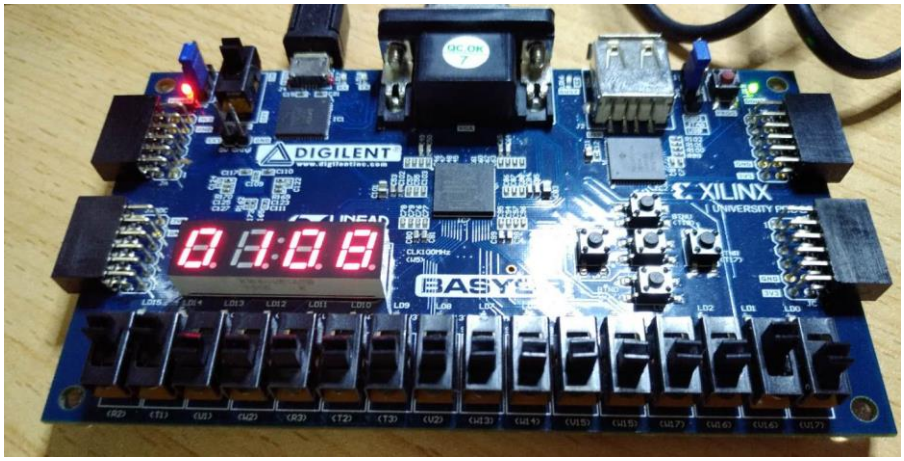
a.当前PC和下条PC：



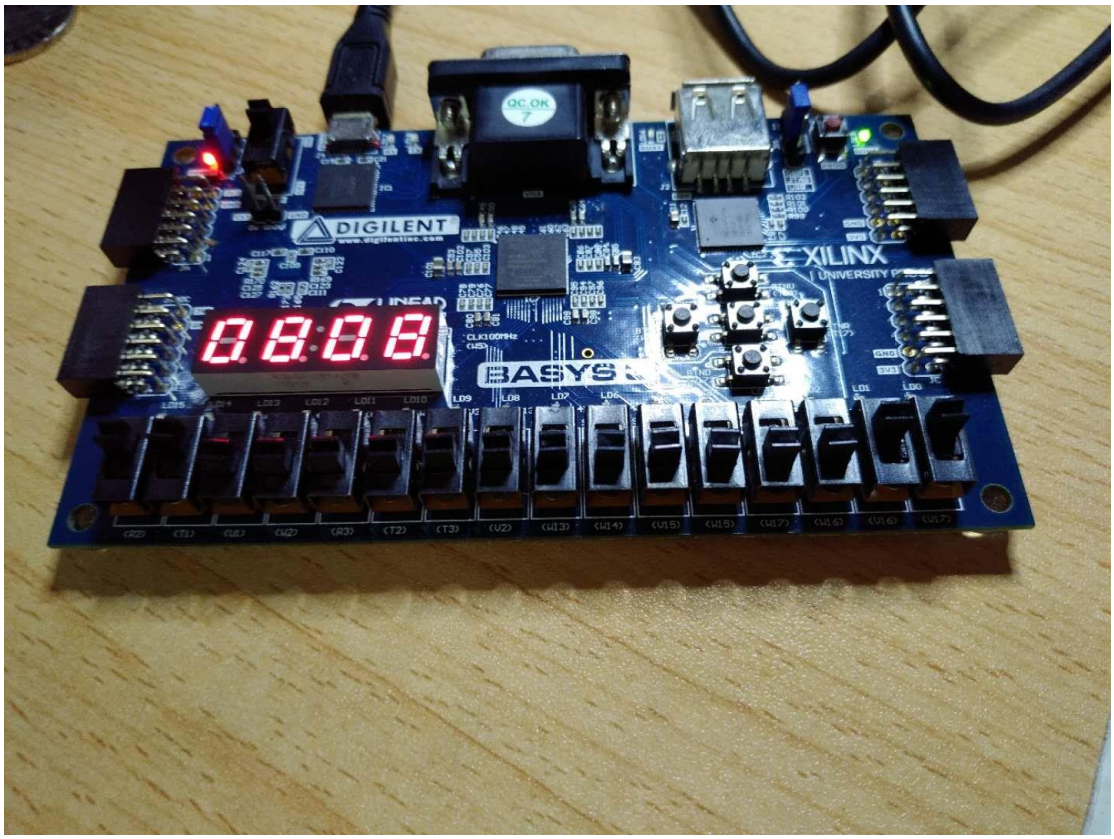

b.rs寄存器的地址、rs寄存器的值：




c. b.rt寄存器的地址、rs寄存器的值：



d.ALU结果和DB的值




……

（剩余的太多，不一一存放在报告里了，我会在检查实验时演示）

**六.实验心得**

1、这次单周期CPU的设计对我来说是一个挑战。之前的数电实验课我们使用过vivado用verilog语言写过电子时钟，CPU的难度相对它来说大了很多很多。这个实验从开始设计到完成实验报告花了我将近一周的所有空闲时间，实在是很费我的精力，不过我也学到了许多verilog的语法要点，也对vivado的操作更加熟悉。

2、这次实验中不同阶段都出现了错误，自己细心找了很久改了很久，也请教了班里做完的同学，终于成功解决。

一开始老是读不进txt文件里的机器码，我找了很久才发现我把“/”打成了“\”，一个方向之差浪费了我很多时间。

后来在仿真的时候，PC总是到了18之后就不动了。我找了很久代码的错误，还是没有

头绪。后来请教好朋友，他告诉我原来我的仿真在一个周期内结束了但是还可以延续到下一个周期继续仿真。那个按钮我从来没用过，自然不知道。我点下“继续仿真”后果然后面的波形就出现了。这个错误不是来自于我的代码，而是来自于我对vivado这个软件的不熟悉啊。

仿真时我的顶层模块全部设置成了output，但是在烧板时就会提示已经超过引脚数量。这时我将顶层模块删除不必要的output然后隐藏，重新写了一个sim文件复制进顶层文件的代码，依然可以成功仿真。到综合的时候我再把顶层模块再设置为可用，就可以完成烧录了。

烧板成功后我单步按键，但PC出现了18~1c无限死循环。我发现1号寄存器的数据没有写进去。可是RST一开始置0但已经开始送PC了，导致clk没有一个下降沿可以写数。这时我请教了已经完成了的朋友，他告诉我这个的解决方法是reset=0时clk置1，这样clk一接上按键就会为0，出现了一个由1到0的下降沿。但是不知道为什么在我的程序里没有用（我现在依旧没有思考清楚）。于是我自己又尝试了清零时如果碰到有需要送数的寄存器，则该寄存器不清零而是送数，居然解决了这个问题，真的是非常开心了哈哈哈。

3、2中的许多问题，我为了解决它们花了好几天的时间。从自己思考、询问同学到上网查资料一点一点地解决。但我觉得这时间花的是值得的。花时间在改正错误上才能更好地提醒自己下次写代码不要再犯类似的错误呀。同时我也明白了同学之间需要互相帮忙。比如室友的工程开始得比我晚，她的很多错误我之前也犯过，就可以很好地给予她帮助。同时我也很感谢解答我问题的同学们。