《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 2 班
学 生 姓 名 ：	黄梓林
学　　号 ：	16337102
时　　间 ：	2017　年　11　月　26　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** （**说明：以助记符表示，是汇编指令；以代码表示，是机器指令**）

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

特别说明：**immediate 是从 PC+4 地址开始和转移到的指令之间指令条数。**immediate 符号扩展之后左移 2 位再相加。为什么要左移 2 位？由于跳转到的指令地址肯定是 4 的倍数（每条指令占 4 个字节），最低两位是“00”，因此将 immediate 放进指令码中的时候，是右移了 2 位的，也就是以上说的“指令之间指令条数”。

（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。

说明：由于 MIPS32 的指令代码长度占 4 个字节，所以指令地址二进制数最低 2 位均为 0，将指令地址放进指令代码中时，可省掉！这样，除了最高 6 位操作码外，还有 26 位可用于存放地址，事实上，可存放 28 位地址了，剩下最高 4 位由 pc+4 最高 4 位拼接上。

==> 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

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。**）

CPU 在处理指令时，一般需要经过以下几个步骤：

(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。

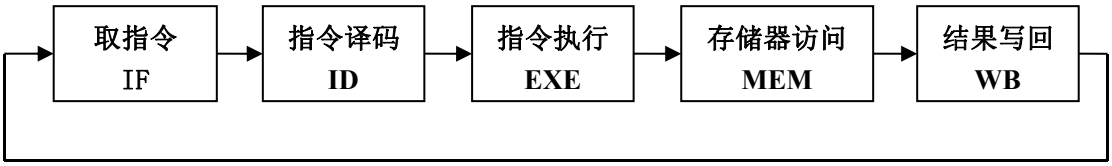
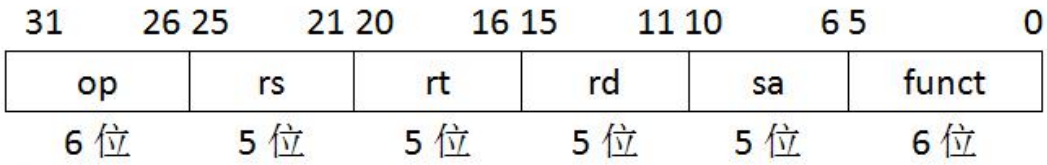


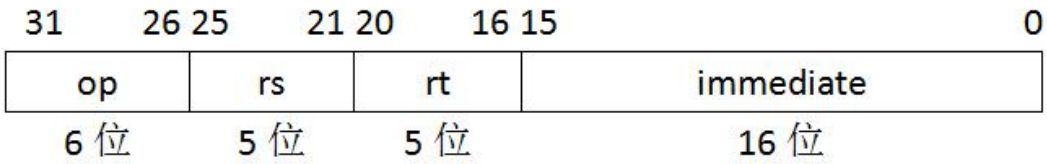
图 1　单周期 CPU 指令处理过程

MIPS 指令的三种格式：

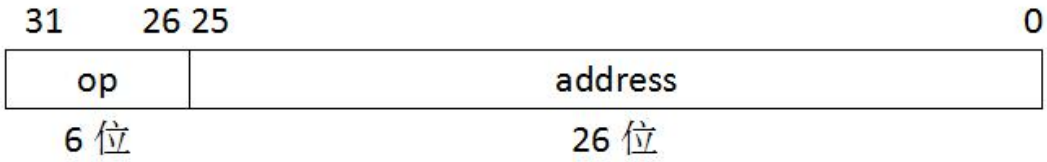
R 类型：



I 类型：



J 类型：



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。

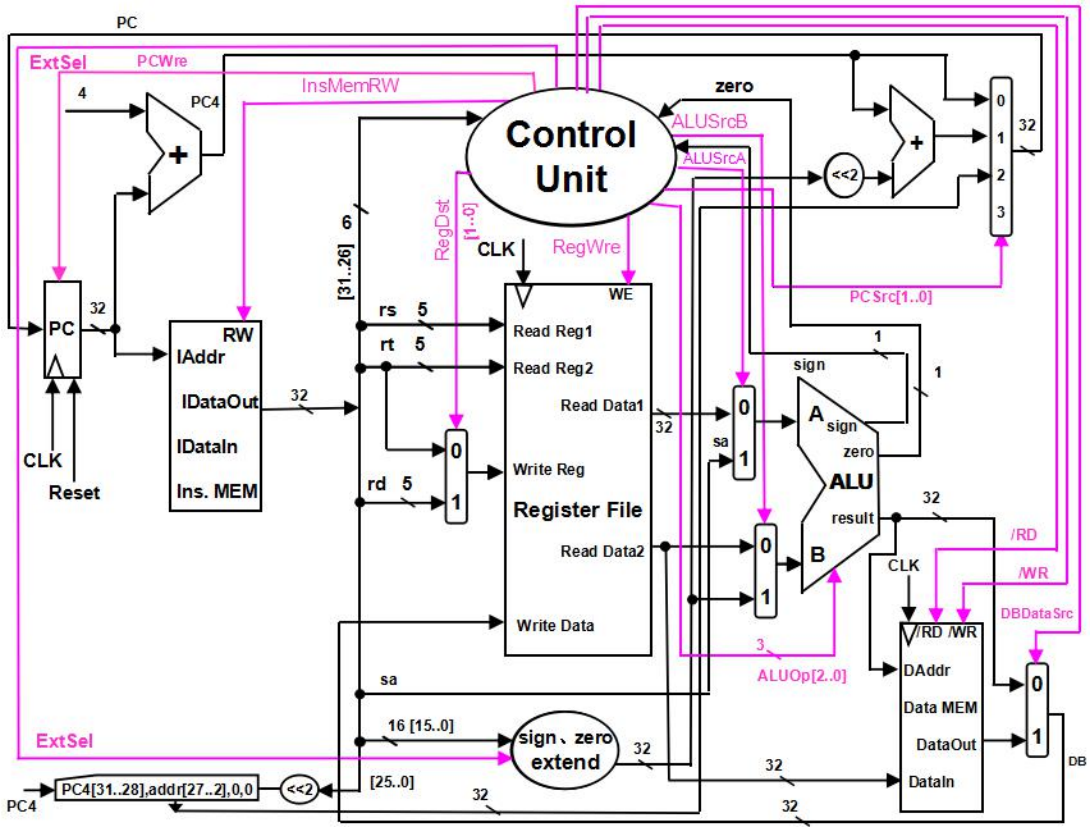


图 2　单周期 CPU 数据通路和控制线路图

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	写指令存储器	读指令存储器(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)； 01：pc<－pc+4+(sign-extend)**immediate**，相关指令：beq(zero=1)、bne(zero=0)、bgtz(sign=0，zero=0)； 10：pc<－{(pc+4)[31..28],addr[27..2],0,0}，相关指令：j； 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 不带符号

110	if (A<B &&(A[31] == B[31]))
	Y = 1;
	else if (A[31] && !B[31] Y = 1;
	else Y = 0;
		比较 A 与 B
		带符号

111	Y = A ⊕ B	异或

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、**CPU设计的思想、方法**:

流程图中包含了CPU的各个模块以及各个模块之间的控制信号，更标明了模块之间的数据通路；通过观察、研究流程图，我们可以确定CPU中的模块的数量以及种类，还有模块们的输入输出，从而正确构建各模块；而且，通过对数据通路的研究，我们可以得知各模块之间是如何协同合作的，在执行不同的指令时，有哪些模块参与了工作，又有哪些模块处于闲置状态，这既有助于我们编写顶层模块，将各模块的功能统合；又有助于我们在测试CPU时跟踪各条指令的执行，找出错误模块所在。

控制信号表中列出了CPU内部模块之间用于通信的所有信号，并且指明了各通信信号与不同指令之间的关系；通过研究控制信号表，我们可以正确编写控制单元，是在执行不同指令时，各个模块都能收到正确的控制信号，正常工作；更可以在测试CPU，执行不同指令时，检查控制信号的值，判断各个模块的工作状态，找出错误模块。

CPU模块:

PC：作用为更新PC的值；输入为时钟信号，重置信号，停止信号，下一个PC的值；输出为更新后的PC

always@(posedge clk or negedge reset) begin//时钟上升沿到来或重置信号下降

沿到来触发

```
        if(reset == 1'b1)//若重置信号为1，将PC置零
            curPC<=0;
        else if(PCWre)//若停滞信号为1，将PC停止
            curPC<=curPC;
        else
            curPC<=nextPC;
    end
```

ROM:作用为存储指令，根据PC的值输出指令；输入为读指令信号，指令地址，输出为该地址所存指令

```
    reg[7:0] rom [99:0];//顺序可能有问题
    initial begin
    $readmemb("C:/Users/ASUS/AppData/Roaming/Xilinx/Vivado/single_CPU/rom_data.txt",rom);//数据文件从0地址开始存放
    end

    always@(InsMemRW or addr) begin//当指令地址变化时触发
        if(InsMemRW == 1)begin//读指令信号为1，读存储器。（大端数据存储模式）
            IDataOut[31:24] = rom[addr];
            IDataOut[23:16] = rom[addr+1];
            IDataOut[15:8] = rom[addr+2];
            IDataOut[7:0] = rom[addr+3];
        end
    end
```

RegFile:寄存器组，用作CPU内部存储；输入为时钟信号，寄存器写信号，写寄存器编码，读寄存器1、2编码，写入寄存器的数据；输出为从寄存器读出的数据。

```
    assign   ReadData1 = regFile[ReadReg1];//读出寄存器中的数据
```

```
assign   ReadData2 = regFile[ReadReg2];//读出寄存器中的数据


always@(negedge clk) //以时钟边缘触发
    if(RegWre == 1&& WriteReg != 0)//当写寄存器信号为1以及所写寄存器编号不
        regFile[WriteReg]<=WriteData; 为零时写寄存器
```

controlUnit：控制单元，负责调度其他各模块的工作；输入为指令的操作码，判断ALU计算结果是否为零的zero信号，是否为有符号数的sign信号；输出为PCWre，PC停止信号；ExtSel，拓展数信号；InsMemRW，指令读写信号；RegDst，写寄存器编号选择信号；RegWre，写寄存器信号；PCSrc，PC值选择信号；ALUSrcA，ALU操作数A选择信号；ALUSrcB，ALU操作数B选择信号；ALUOp，ALU功能码；RD，读内存信号；WR，写内存信号；DBDataSrc，写寄存器数据选择信号。

```
always@(opcode or zero or sign)//当操作码，zero、sign信号之一改变时触发
begin
    case(opcode)
      6'b000000://add指令
      begin
            PCWre<=0;//pc更改
            ALUSrcA<=0;//ALU操作数A为rs寄存器数据
            ALUSrcB<=0;//ALU操作数B为rt寄存器数据
            DBDataSrc<=0;//写回寄存器的数据为ALU运算结果
            RegWre<=1;//读寄存器
            insMemRW<=1;//读指令
            RD<=1;//无读内存
            WR<=1;//无写内存
            RegDst<=1;//写寄存器为rd
            PCSrc[1:0]<=2'b00;//PC=PC+4
```

```
        ALUOp[2:0]<=3'b000; //使用ALU的加法功能
end
……
6'b010000://ori
  begin
        PCWre<=0;//pc更改
        ALUSrcA<=0;//ALU操作数A为rs寄存器数据
        ALUSrcB<=1;//ALU操作数B为立即数
        DBDataSrc<=0;//写回寄存器的数据为ALU运算结果
        ExtSel<=0;//无符号拓展立即数
        RegWre<=1;//读寄存器
        insMemRW<=1;//读指令
        RD<=1;//无读内存
        WR<=1;//无写内存
        RegDst<=0;写寄存器为rt
        PCSrc[1:0]<=2'b00;//PC=PC+4
        ALUOp[2:0]<=3'b011;//使用ALU的或运算功能
  end
  ……
 6'b011000://sll
  begin
       PCWre<=0;
       ALUSrcA<=1;//ALU操作数A为移位数sa
       ALUSrcB<=0;//ALU操作数B为rt寄存器数据
       DBDataSrc<=0;//写回寄存器的数据为ALU运算结果
       RegWre<=1;//读寄存器
       insMemRW<=1;//读指令
       RD<=1;//无读内存
       WR<=1;//无写内存
```

```
        RegDst<=1;写寄存器为rd
        PCSrc[1:0]<=2'b00;//PC=PC+4
        ALUOp[2:0]<=3'b010;使用ALU的左移运算功能
    end
    ……
6'b011100://slt
    begin
        PCWre<=0;//pc更改
        ALUSrcA<=0;//ALU操作数A为rs寄存器数据
        ALUSrcB<=0;//ALU操作数B为rt寄存器数据
        DBDataSrc<=0;//写回寄存器的数据为ALU运算结果
        RegWre<=1;//读寄存器
        insMemRW<=1;//一直读指令
        RD<=1;//无读内存
        WR<=1;//无写内存
        RegDst<=1;写寄存器为rd
        PCSrc[1:0]<=2'b00;//PC=PC+4
        ALUOp[2:0] <= (sign == 0) ? 3'b101 : 3'b110;//若ALU操作数为有符
                                                    号数，使用用符号数比较功能；否
                                                    则，使用无符号数比较功能
    end

6'b100110://sw,存到寄存器内容和立即数相加的地址中，不是存到寄存器中，
                不需写寄存器
    begin
        PCWre<=0;//pc更改
        ALUSrcA<=0;//ALU操作数A为rs寄存器数据
        ALUSrcB<=1;//ALU操作数B为立即数
        ExtSel<=1;//有符号拓展立即数
```

```
            RegWre<=0;//无写寄存器
            insMemRW<=1;//读指令
            RD<=1;//无读内存
            WR<=0;//写内存
            PCSrc[1:0]<=2'b00;//PC=PC+4
            ALUOp[2:0]<=3'b000; //计算地址使用ALU的加法功能
    end
    ……

6'b110000://beq
    begin
        PCWre<=0;//pc更改
        ALUSrcA<=0;//ALU操作数A为rs寄存器数据
        ALUSrcB<=0;//ALU操作数B为rt寄存器数据
        ExtSel<=1;//有符号拓展立即数
        RegWre<=0;//无写寄存器
        insMemRW<=1;//一直读指令
        RD<=1;//无读内存
        WR<=1;//无写内存
        ALUOp[2:0]<=3'b111;使用ALU的异或运算功能
        PCSrc[1:0]<= (zero == 0)?2'b00:2'b01;//若ALU运算结果为0，PC跳
                                                转；否则，PC+4
    end
    ……

6'b111000://j
    begin
            PCWre<=0;//pc更改
            RegWre<=0;//无写寄存器
```

```
                insMemRW<=1;//一直读指令
                RD<=1;//无读内存
                WR<=1;//无写内存
                PCSrc[1:0]<=2'b10; //pc={(pc+4)[31..28],addr[27..2],0,0}
            end

        6'b111111://halt
            begin
                PCWre<=1;//PC不变
                RegWre<=0;//无写寄存器
             end
    endcase
```

RegisterMux：写寄存器选择器；输入为rt寄存器编号，rd寄存器编号，写寄存器选择信号；输出为写寄存器编号

```
assign      WriteReg = (RegDst==0) ? rt : rd;//写寄存器选择信号为0时，写寄存
                                              器为rt；否则，为rd
```

ALU：逻辑运算模块；输入为ALU操作码，ALU操作数A，ALU操作数B；输出为判断ALU计算结果是否为零的zero信号，是否为有符号数的sign信号，以及ALU运算结果

```
always@(ALUopcode or rega or regb) begin//操作码或操作数改变时触发
    case(ALUopcode)
        3'b000:result <= rega + regb;//加法
        3'b001:result <= rega - regb;//减法
        3'b010:result <= regb << rega;//左移
        3'b011:result <= rega | regb;//或
        3'b100:result <= rega & regb;//与
        3'b101:result <= (rega < regb)?1:0;//无符号数比较
        3'b110:begin//有符号数比较
```

```
                if(rega < regb && (rega[31] == regb[31] ) )
                    result <= 1;
                else if(rega[31] == 1 && regb[31] == 0)
                    result <= 1;
                else
                    result <= 0;
                end
            3'b111:result <= rega ^ regb;//异或
            default: result <= 32'h 00000000;
        endcase
    end
    assign zero = (result == 0)?1:0;//结果为0时，zero为1；反之，为0
    assign sign = (result[31] == 0)?0:1;//结果的符号位为0时，sign为0；反之，为1
```

RAM：数据存储器；输入为时钟信号，内存地址，写入数据，读内存、写内存信号；输出为从内存读出数据

```
    reg[7:0] ram[0:60];//内存
    //读内存
  assign dataOut[7:0]=(RD==0)?ram[address+3]:8'bz;
  assign   dataOut[15:8]=(RD==0)?ram[address+2]:8'bz;
  assign   dataOut[23:16]=(RD==0)?ram[address+1]:8'bz;
  assign   dataOut[31:24]=(RD==0)?ram[address]:8'bz;
    //写内存
    always@(negedge clk) begin//时钟下降沿触发
        if(WR == 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
```

extend：拓展立即数模块；输入为拓展立即数信号，被拓展数；输出为拓展结果

assign extended = (ExtSel == 0) ? 32'h00000000 | needEx : (needEx[15] == 0) ? 32'h00000000 | needEx : 32'hffff0000 | needEx;//信号为0，进行无符号拓展；反之，进行有符号拓展

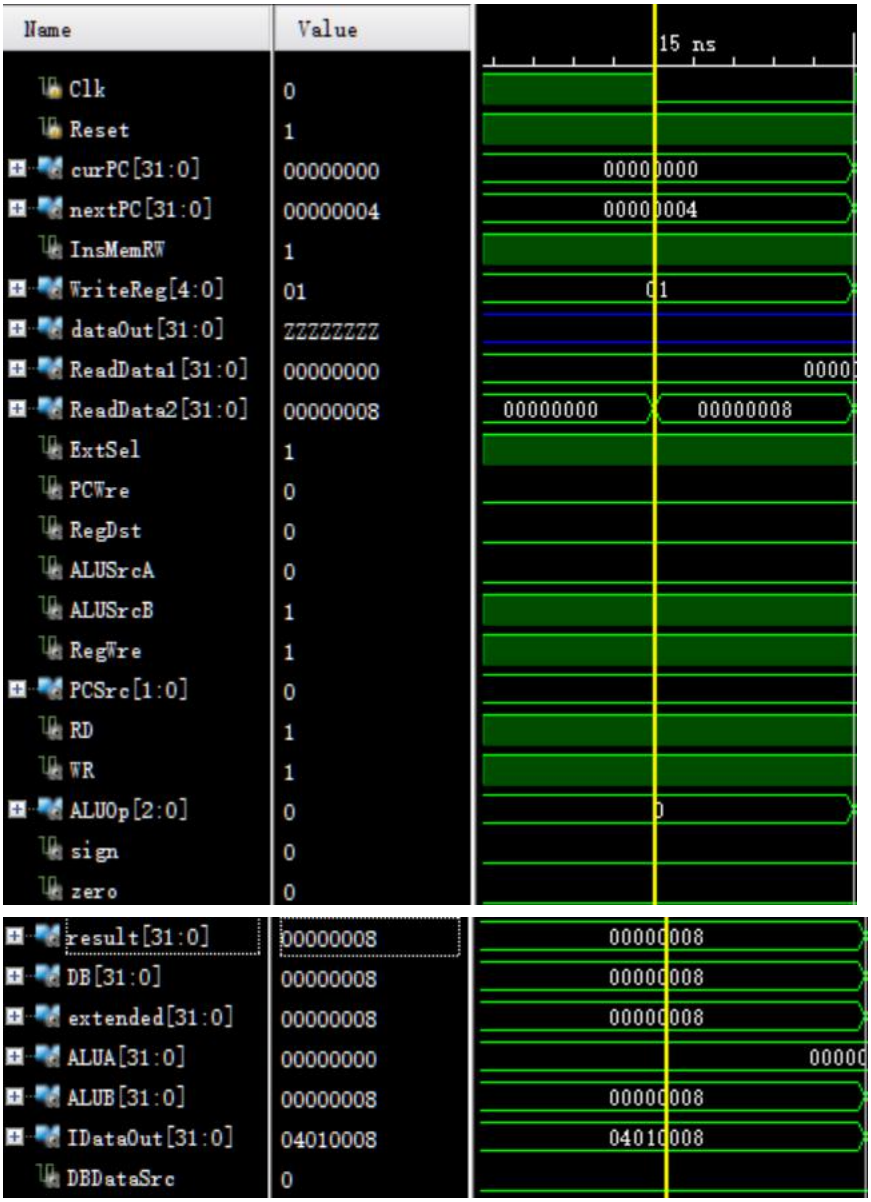
2、**CPU正确性**：

测试代码：

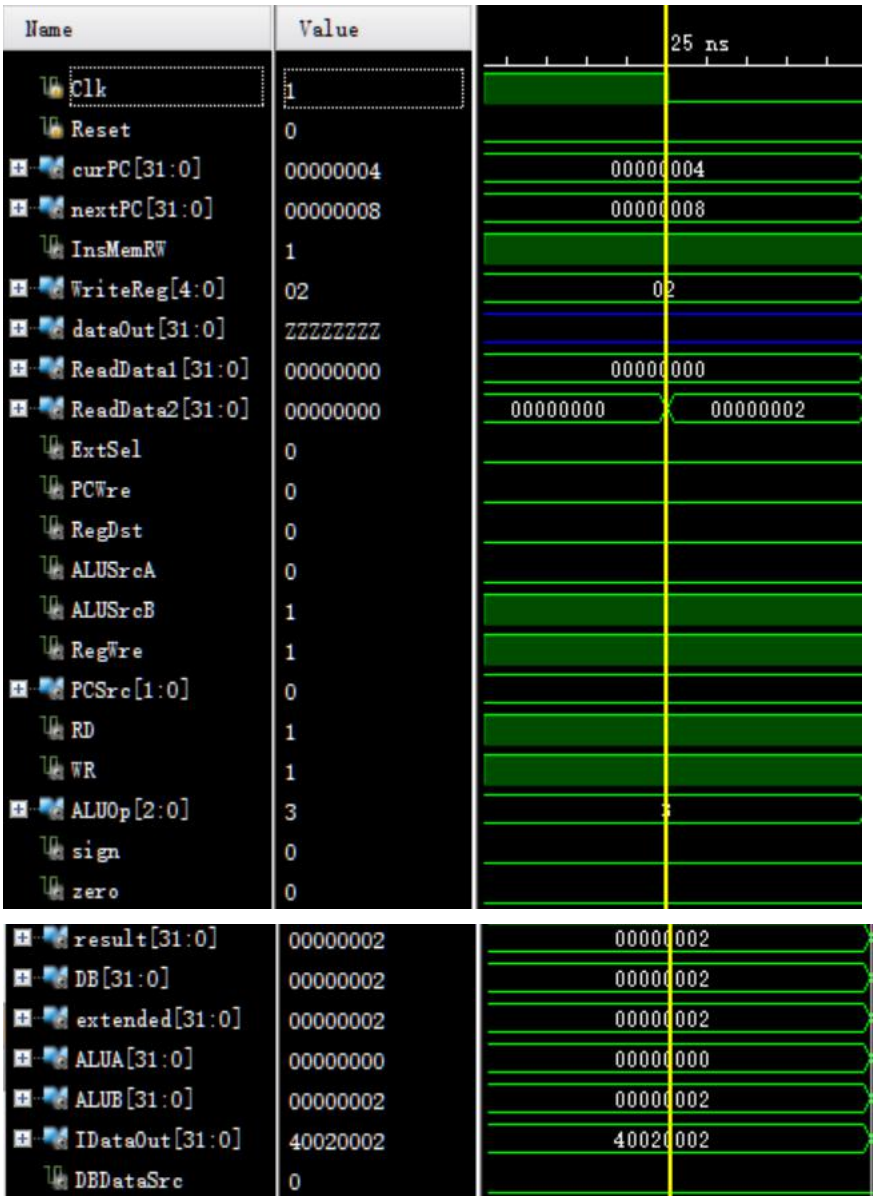
地址	汇编程序	指令代码					
		op（6）	**rs(5)**	**rt(5)**	**rd(5)/immediate (16)**	**16 进制数代码**	
0x00000000	addi $1,$0,8	000001	00000	00001	0000 0000 0000 1000	=	04010008
0x00000004	ori $2,$0,2	010000	00000	00010	0000 0000 0000 0010	=	40020002
0x00000008	add $3,$2,$1	000000	00010	00001	0001 1000 0000 0000	=	00411800
0x0000000c	sub $5,$3,$2	000010	00011	00010	0010 1000 0000 0000	=	08622800
0x00000010	and $4,$5,$2	010001	00101	00010	0010 0000 0000 0000	=	44A22000
0x00000014	or $8,$4,$2	010010	00100	00010	0100 0000 0000 0000	=	48824000
0x00000018	sll $8,$8,1	011000	00000	01000	0100 0000 0100 0000	=	60084040
0x0000001c	**bne $8,$1,-2 (≠,转 18)**	110001	01000	00001	1111 1111 1111 1110	=	C501FFFE
0x00000020	slt $6,$2,$1	011100	00010	00001	0011 0000 0000 0000	=	70413000
0x00000024	slt $7,$6,$0	011100	00110	00000	0011 1000 0000 0000	=	70C03800
0x00000028	addi $7,$7,8	000001	00111	00111	0000 0000 0000 1000	=	04E70008
0x0000002c	**beq $7,$1,-2 (=, 转 28)**	110000	00111	00001	1111 1111 1111 1110	=	C0E1FFFE
0x00000030	sw $2,4($1)	100110	00001	00010	0000 0000 0000 0100	=	98220004
0x00000034	lw $9,4($1)	100111	00001	01001	0000 0000 0000 0100	=	9C290004
0x00000038	**bgtz $9,1 (>0,转 40)**	110010	01001	00000	0000 0000 0000 0001	=	C9200001
0x0000003c	**halt**	111111	00000	00000	0000000000000000	=	FC000000
0x00000040	**addi $9,$0,-1**	000001	00000	01001	1111 1111 1111 1111	=	0409FFFF
0x00000044	**j 0x00000038**	111000	00000	00000	0000 0000 0000 1110	=	E000000E
0x00000048							
0x0000004c							

波形图：

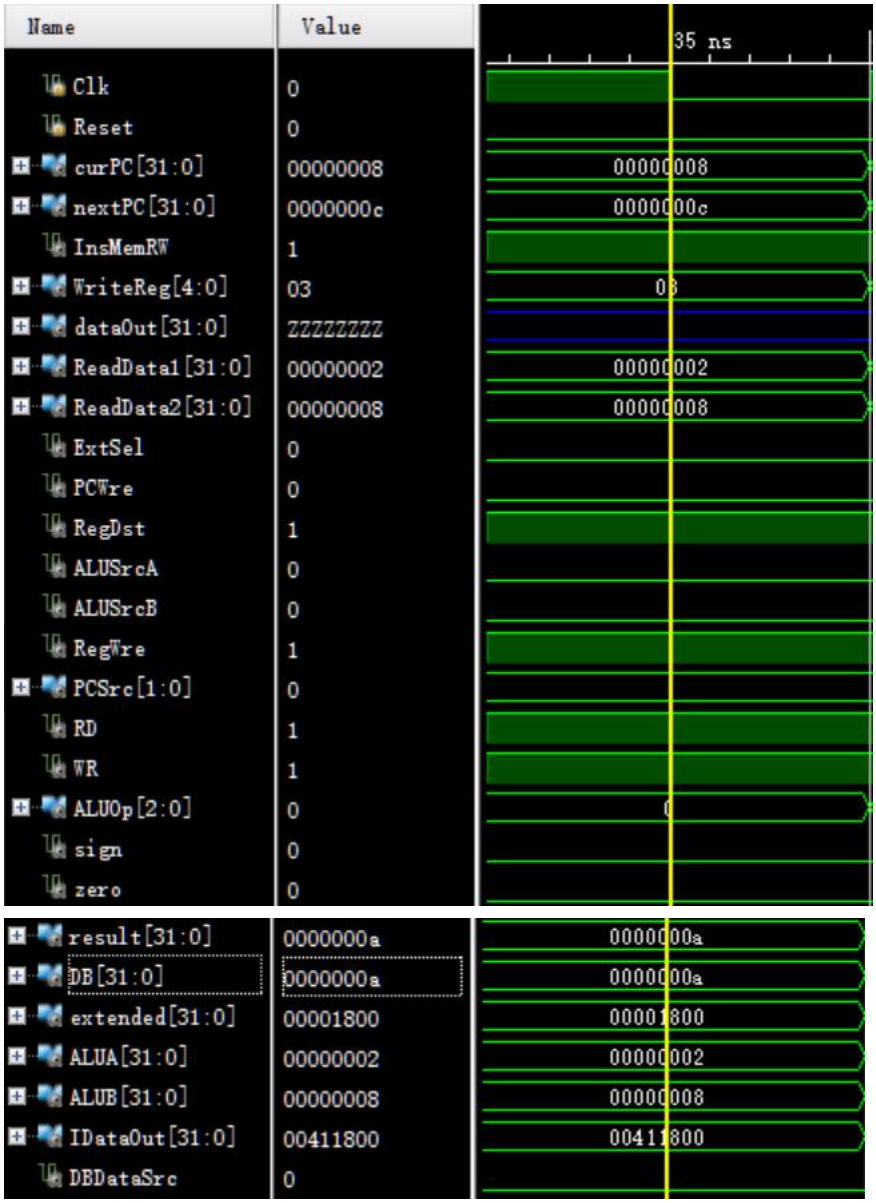
addi　$1,$0,8



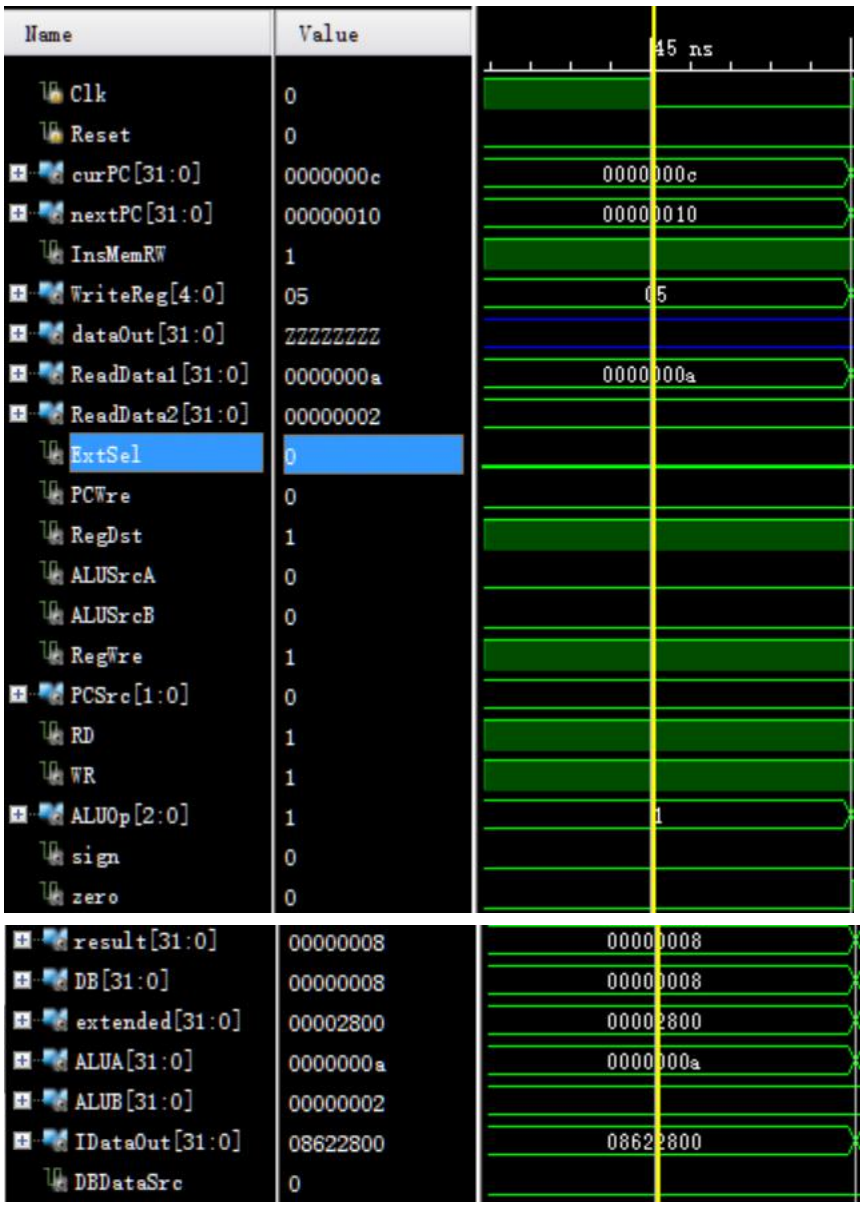
ori　$2,$0,2


Name
Value
25 ns
Clk
1
Reset
0
curPC[31:0]
00000004
nextPC[31:0]
00000008
InsMemRW
1
WriteReg[5:0]
02
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000000
ReadData2[31:0]
00000000
00000002
ExtSel
0
PCWre
0
RegDst
0
ALUSrcA
0
ALUSrcB
1
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
3
sign
0
zero
0
result[31:0]
00000002
DB[31:0]
00000002
extended[31:0]
00000002
ALUA[31:0]
00000000
ALUB[31:0]
00000002
IDataOut[31:0]
40020002
DBDataSrc
0

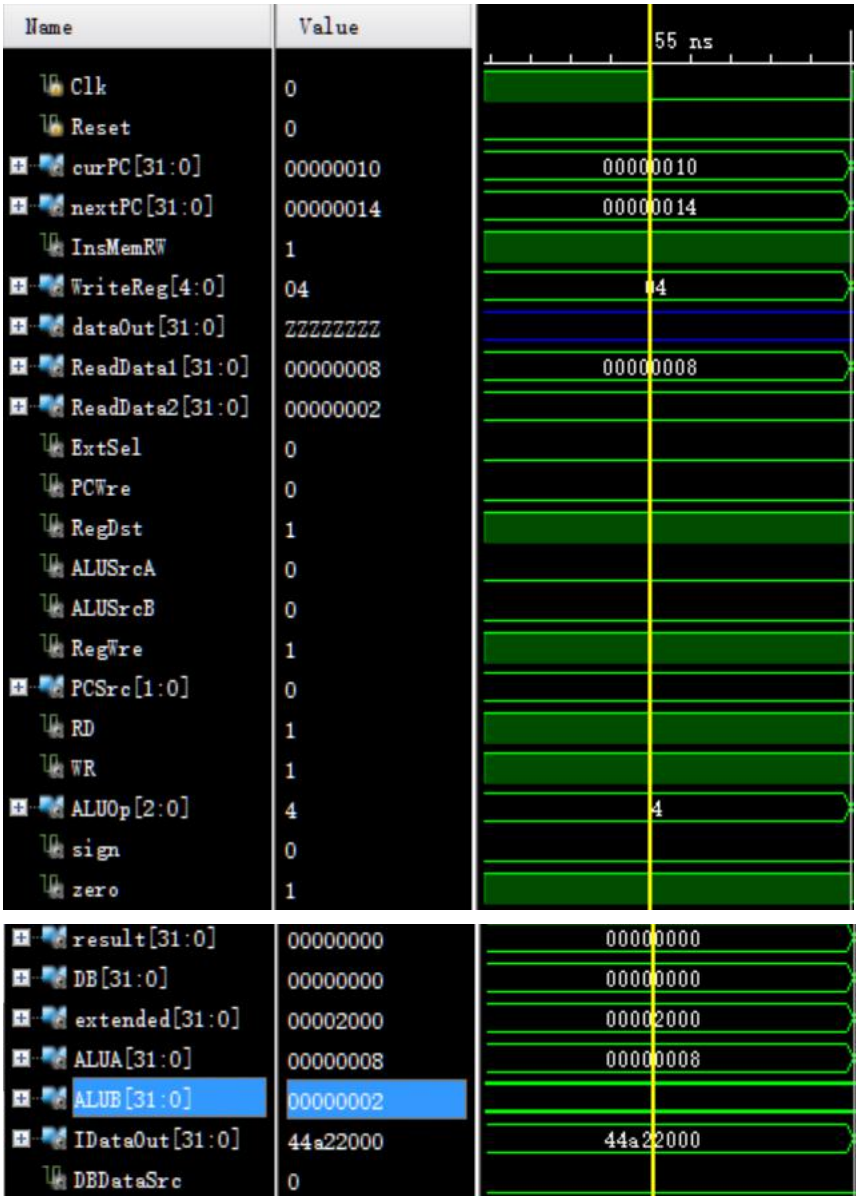
add $3,$2,$1


Name
Value
35 ns
Clk
0
Reset
0
curPC[31:0]
00000008
nextPC[31:0]
0000000c
InsMemRW
1
WriteReg[4:0]
03
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000002
ReadData2[31:0]
00000008
ExtSel
0
PCWre
0
RegDst
1
ALUSrcA
0
ALUSrcB
0
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
0
sign
0
zero
0
result[31:0]
0000000a
DB[31:0]
0000000a
extended[31:0]
00001800
ALUA[31:0]
00000002
ALUB[31:0]
00000008
IDataOut[31:0]
00411800
DBDataSrc
0

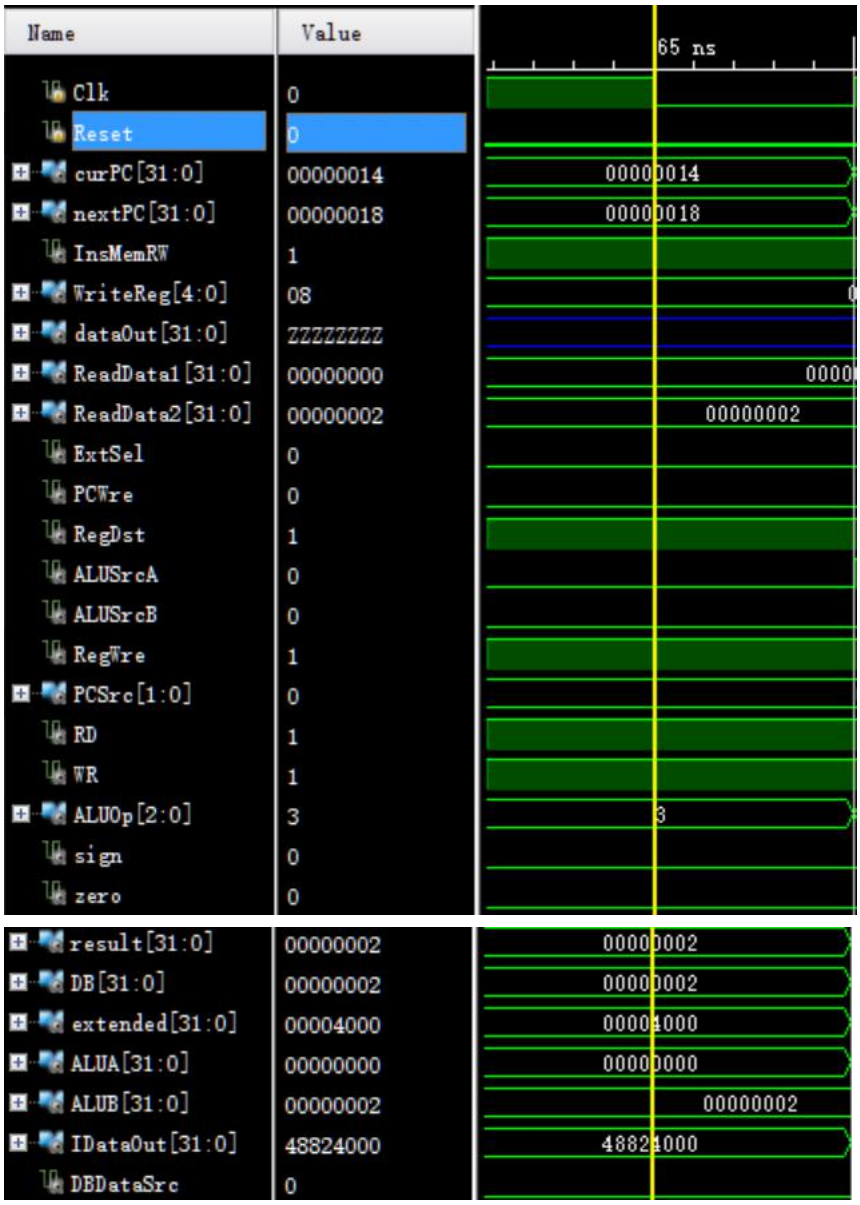
sub $5,$3,$2


Name
Value
45 ns
Clk 0
Reset 0
curPC[31:0] 0000000c
nextPC[31:0] 00000010
InsMemRW 1
WriteReg[4:0] 05
dataOut[31:0] zzzzzzzz
ReadData1[31:0] 0000000a
ReadData2[31:0] 00000002
ExtSel 0
PCWre 0
RegDst 1
ALUSrcA 0
ALUSrcB 0
RegWre 1
PCSrc[1:0] 0
RD 1
WR 1
ALUOp[2:0] 1
sign 0
zero 0
result[31:0] 00000008
DB[31:0] 00000008
extended[31:0] 00002800
ALUA[31:0] 0000000a
ALUB[31:0] 00000002
IDataOut[31:0] 08622800
DBDataSrc 0

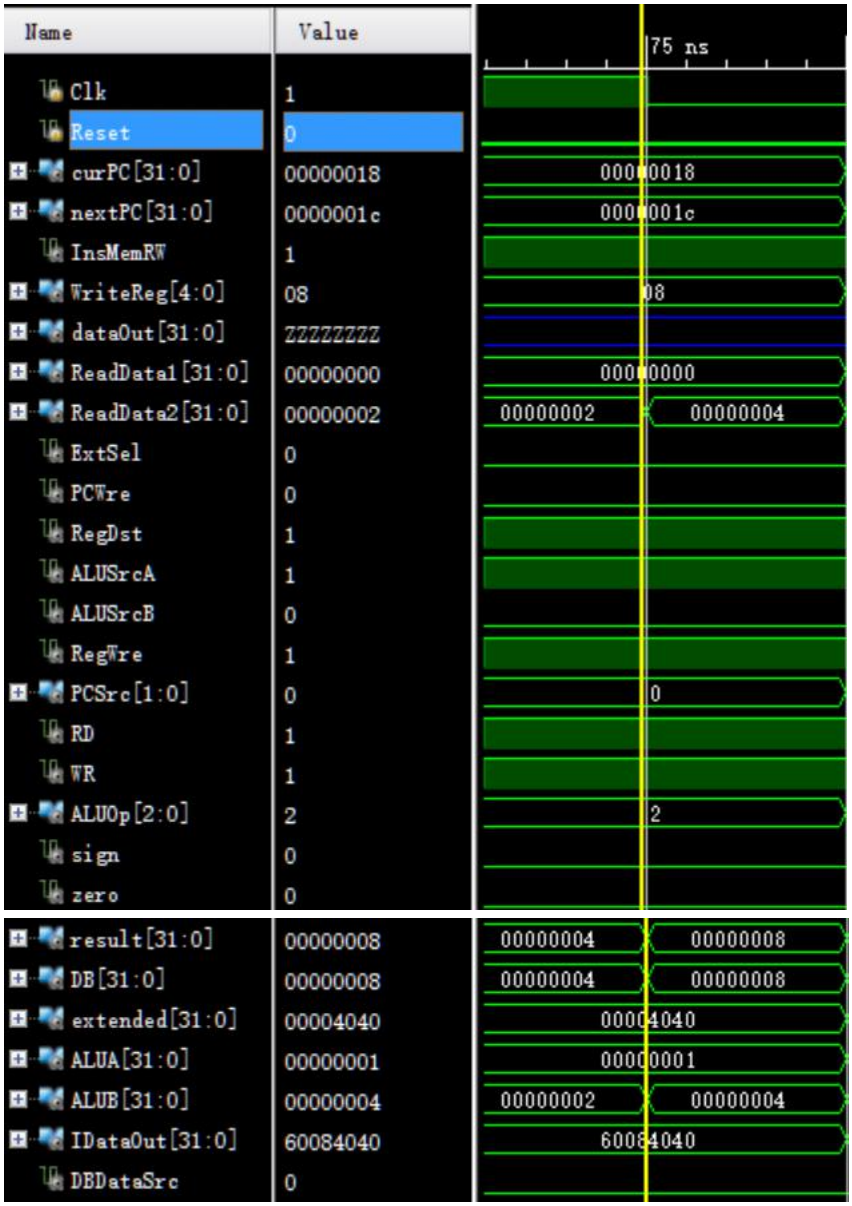
and $4,$5,$2


Name
Value
55 ns
Clk
0
Reset
0
curPC[31:0]
00000010
nextPC[31:0]
00000014
InsMemRW
1
WriteReg[4:0]
04
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000008
ReadData2[31:0]
00000002
ExtSel
0
PCWre
0
RegDst
1
ALUSrcA
0
ALUSrcB
0
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
4
sign
0
zero
1
result[31:0]
00000000
DB[31:0]
00000000
extended[31:0]
00002000
ALUA[31:0]
00000008
ALUB[31:0]
00000002
IDataOut[31:0]
44a22000
DBDataSrc
0

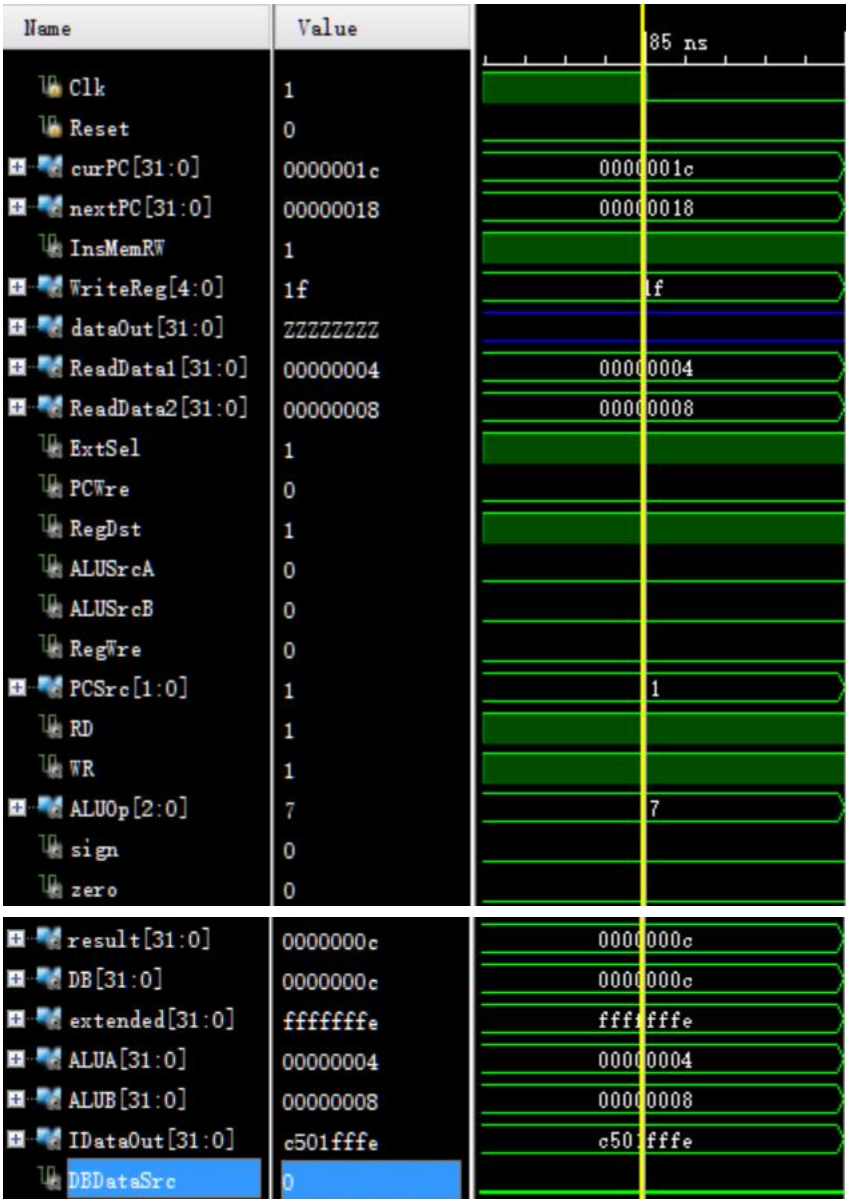
or $8,$4,$2


Name
Value
65 ns
Clk
0
Reset
0
curPC[31:0]
00000014
nextPC[31:0]
00000018
InsMemRW
1
WriteReg[4:0]
08
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000000
ReadData2[31:0]
00000002
ExtSel
0
PCWre
0
RegDst
1
ALUSrcA
0
ALUSrcB
0
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
3
sign
0
zero
0
result[31:0]
00000002
DB[31:0]
00000002
extended[31:0]
00004000
ALUA[31:0]
00000000
ALUB[31:0]
00000002
IDataOut[31:0]
48824000
DBDataSrc
0

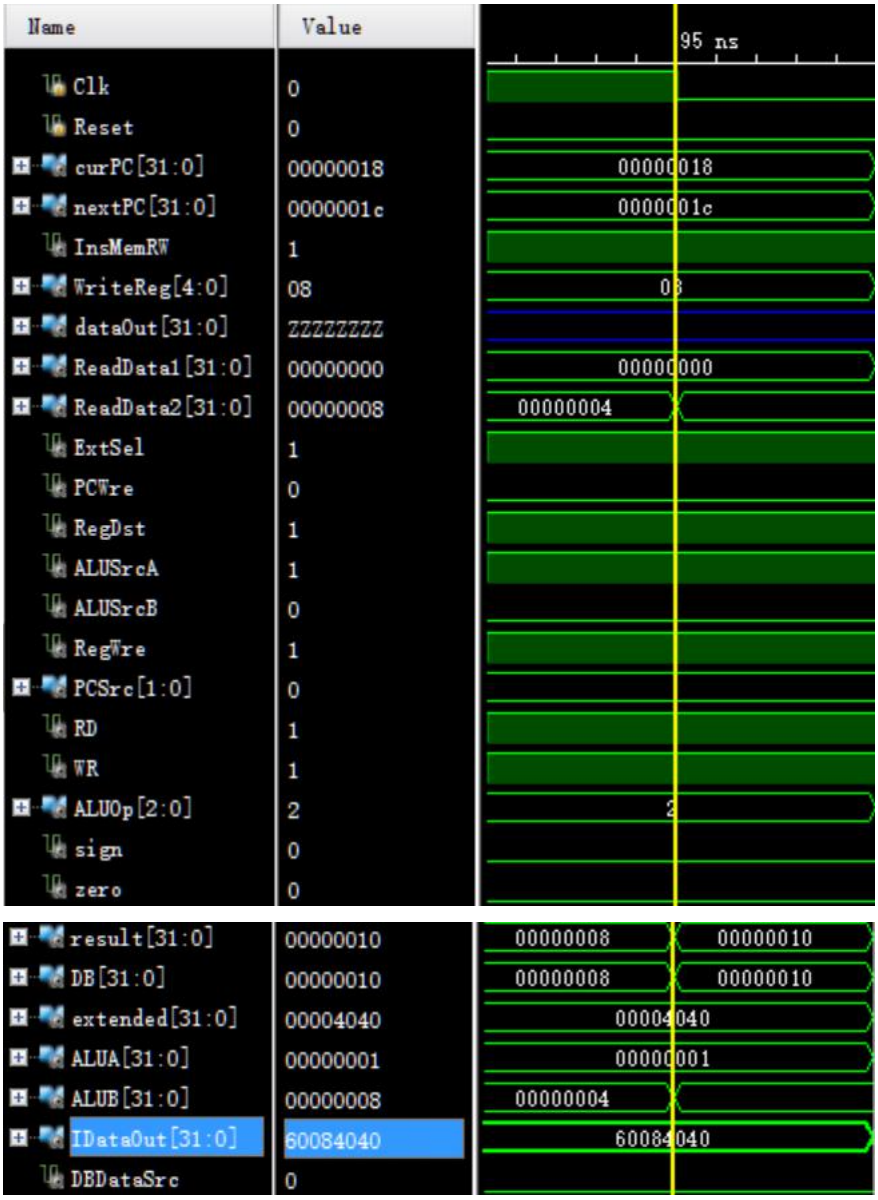
sll $8,$8,1


Name
Value
75 ns
Clk
1
Reset
0
curPC[31:0]
00000018
nextPC[31:0]
0000001c
InsMemRW
1
WriteReg[4:0]
08
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000000
ReadData2[31:0]
00000002
00000004
ExtSel
0
PCWre
0
RegDst
1
ALUSrcA
1
ALUSrcB
0
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
2
sign
0
zero
0
result[31:0]
00000008
00000004
00000008
DB[31:0]
00000008
00000004
00000008
extended[31:0]
00004040
ALUA[31:0]
00000001
ALUB[31:0]
00000004
00000002
00000004
IDataOut[31:0]
60084040
DBDataSrc
0

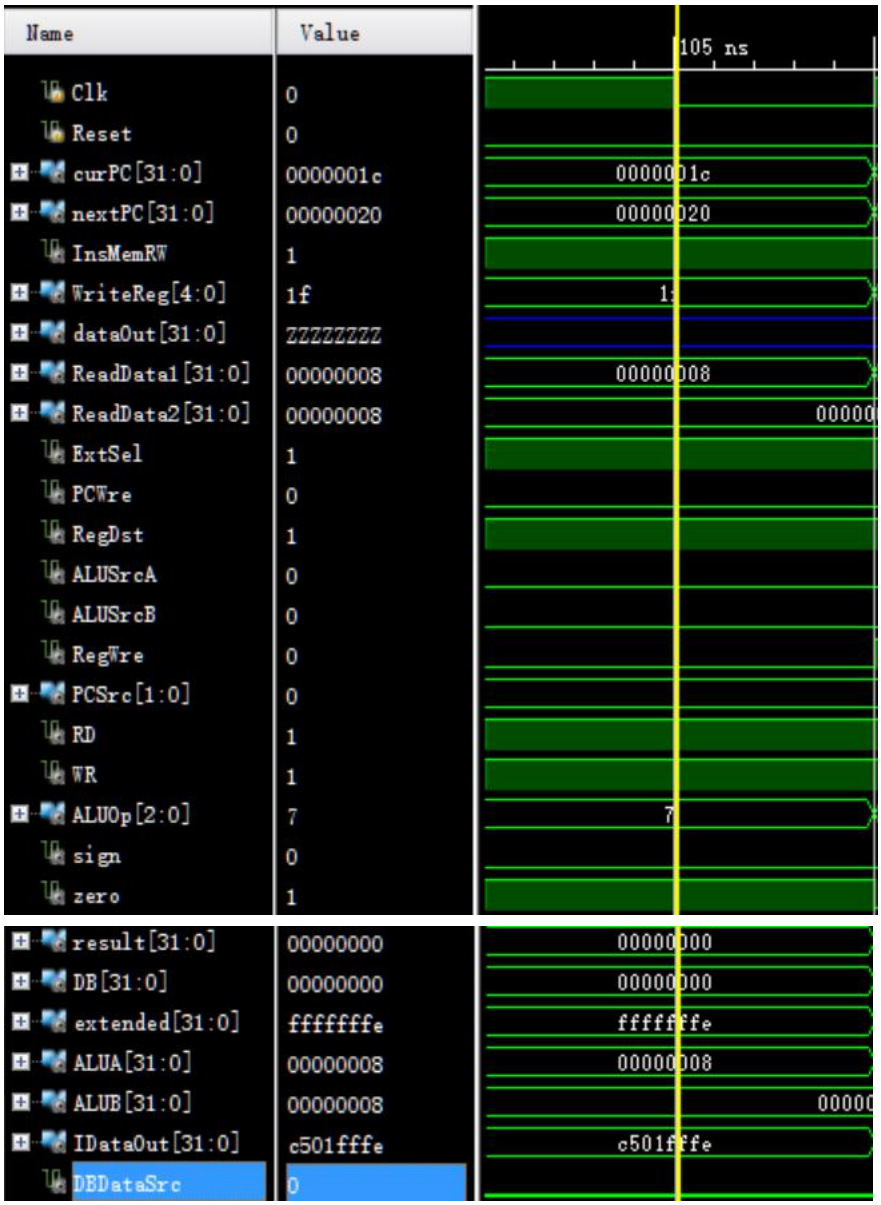
bne $8,$1,-2


Name
Value
85 ns
Clk
1
Reset
0
curPC[31:0]
0000001c
nextPC[31:0]
00000018
InsMemRW
1
WriteReg[4:0]
1f
dataOut[31:0]
zzzzzzzz
ReadData1[31:0]
00000004
ReadData2[31:0]
00000008
ExtSel
1
PCWre
0
RegDst
1
ALUSrcA
0
ALUSrcB
0
RegWre
0
PCSrc[1:0]
1
RD
1
WR
1
ALUOp[2:0]
7
sign
0
zero
0
result[31:0]
0000000c
DB[31:0]
0000000c
extended[31:0]
fffffffe
ALUA[31:0]
00000004
ALUB[31:0]
00000008
IDataOut[31:0]
c501fffe
DBDataSrc
0

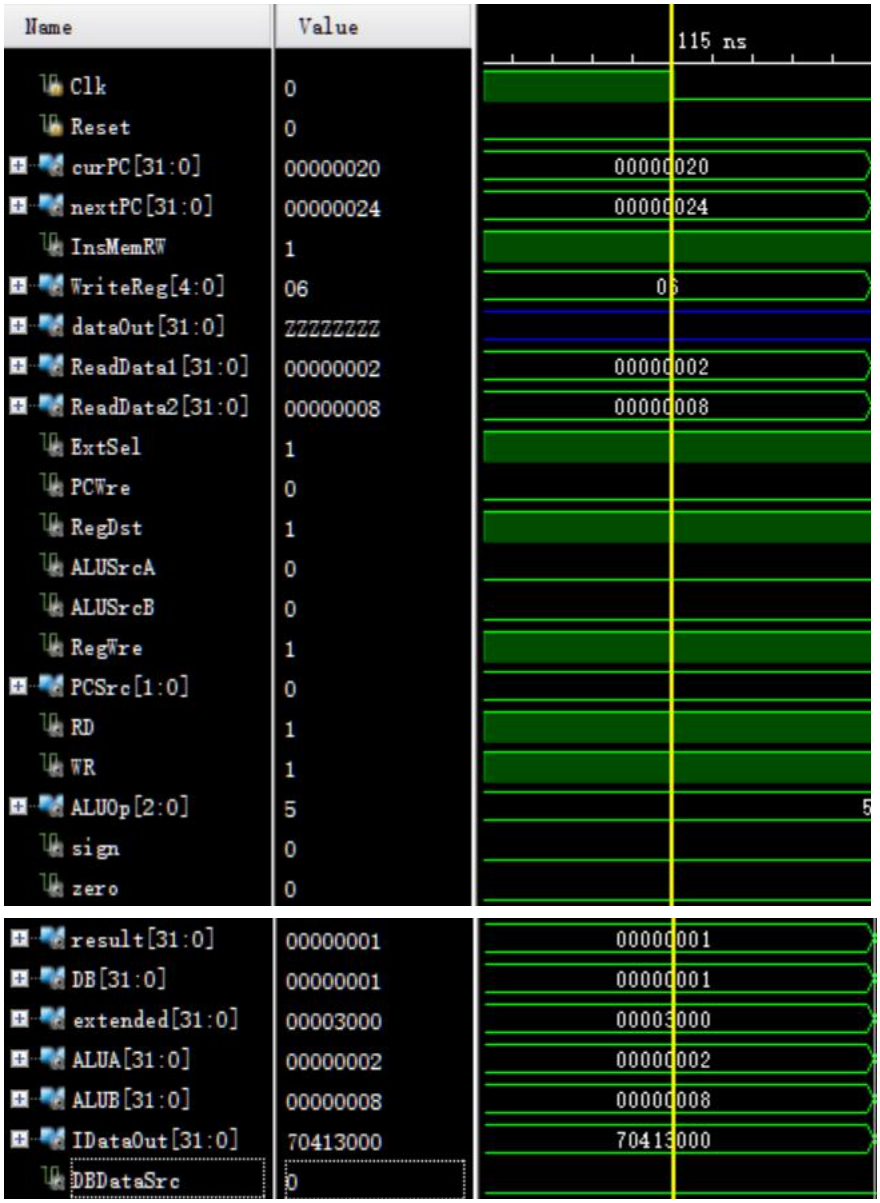
sll $8,$8,1


Name
Value
95 ns
Clk
0
Reset
0
curPC[31:0]
00000018
nextPC[31:0]
0000001c
InsMemRW
1
WriteReg[4:0]
08
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000000
ReadData2[31:0]
00000008
00000004
ExtSel
1
PCWre
0
RegDst
1
ALUSrcA
1
ALUSrcB
0
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
2
sign
0
zero
0
result[31:0]
00000010
00000008
00000010
DB[31:0]
00000010
00000008
00000010
extended[31:0]
00004040
00004040
ALUA[31:0]
00000001
00000001
ALUB[31:0]
00000008
00000004
IDataOut[31:0]
60084040
60084040
DBDataSrc
0

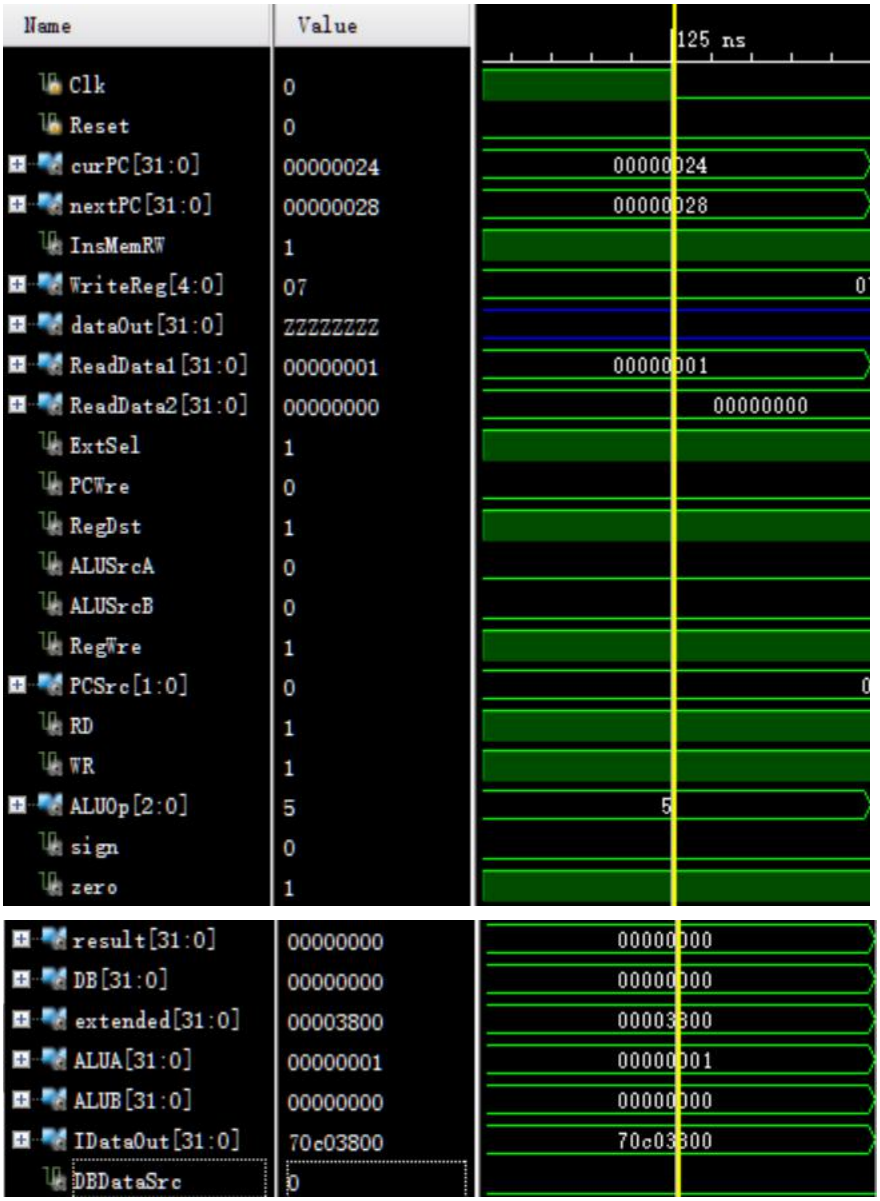
bne $8,$1,-2


Name
Value
105 ns
Clk
0
Reset
0
curPC[31:0]
0000001c
nextPC[31:0]
00000020
InsMemRW
1
WriteReg[4:0]
1f
dataOut[31:0]
zzzzzzzz
ReadData1[31:0]
00000008
ReadData2[31:0]
00000008
ExtSel
1
PCWre
0
RegDst
1
ALUSrcA
0
ALUSrcB
0
RegWre
0
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
7
sign
0
zero
1
result[31:0]
00000000
DB[31:0]
00000000
extended[31:0]
fffffffe
ALUA[31:0]
00000008
ALUB[31:0]
00000008
IDataOut[31:0]
c501fffe
DBDataSrc
0

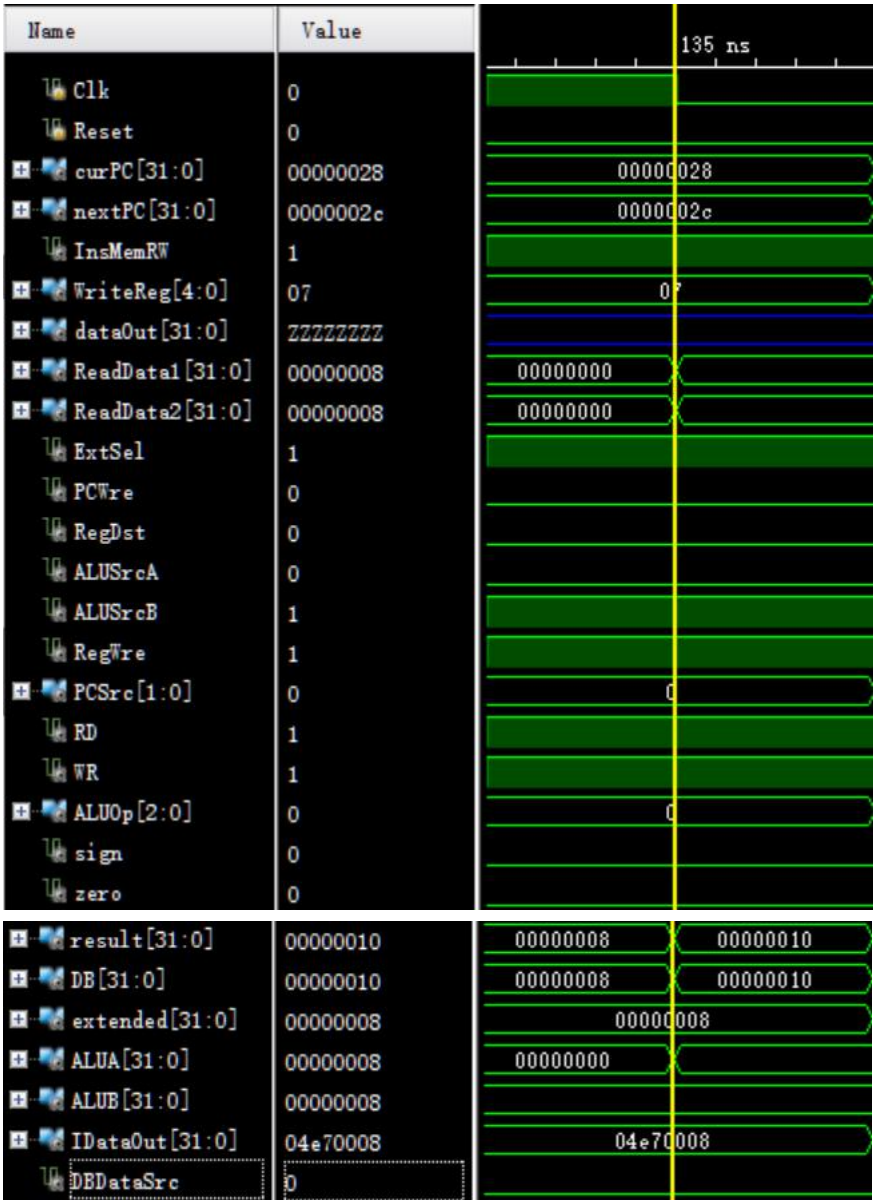
slt $6,$2,$1


Name
Value
115 ns
Clk
0
Reset
0
curPC[31:0]
00000020
00000020
nextPC[31:0]
00000024
00000024
InsMemRW
1
WriteReg[4:0]
06
06
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000002
00000002
ReadData2[31:0]
00000008
00000008
ExtSel
1
PCWre
0
RegDst
1
ALUSrcA
0
ALUSrcB
0
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
5
5
sign
0
zero
0
result[31:0]
00000001
00000001
DB[31:0]
00000001
00000001
extended[31:0]
00003000
00003000
ALUA[31:0]
00000002
00000002
ALUB[31:0]
00000008
00000008
IDataOut[31:0]
70413000
70413000
DBDataSrc
0

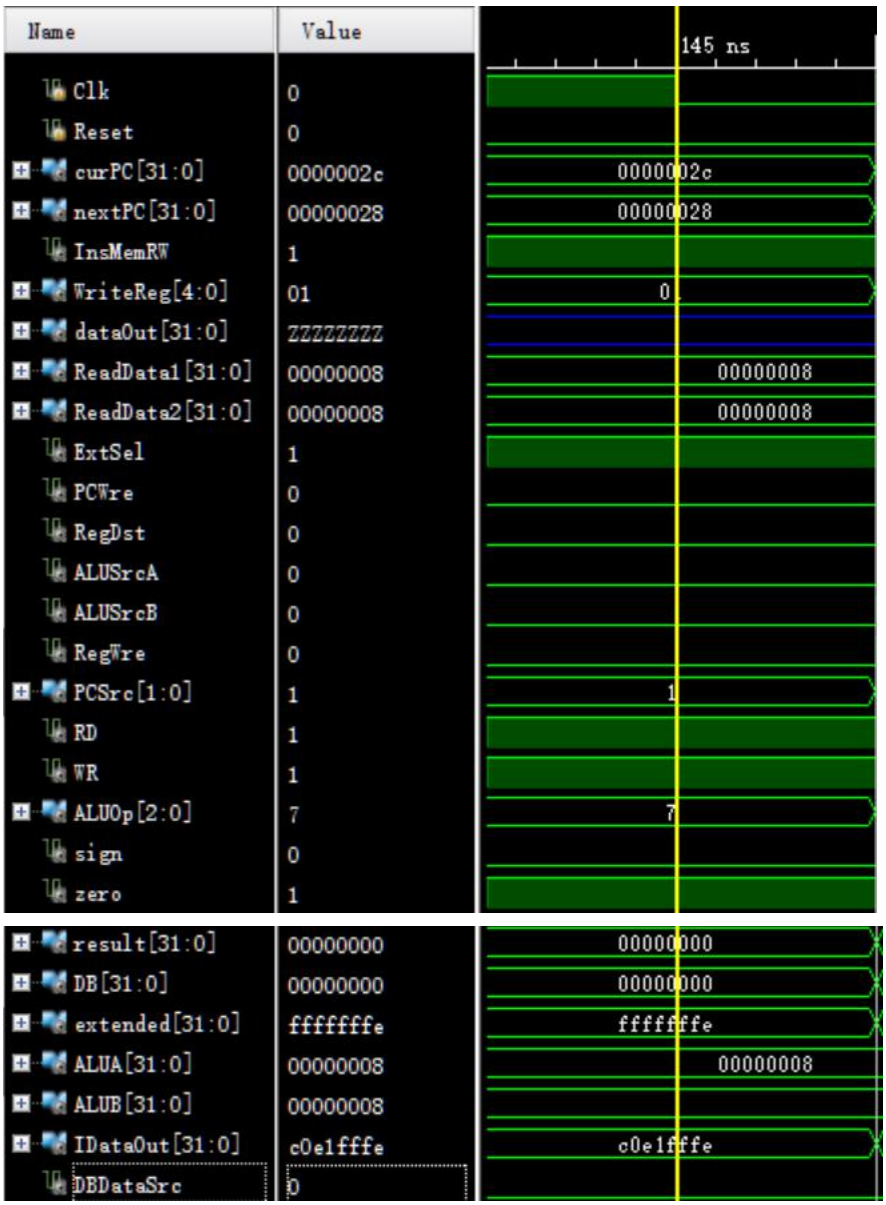
slt $7,$6,$0


Name
Value
125 ns
Clk
0
Reset
0
curPC[31:0]
00000024
nextPC[31:0]
00000028
InsMemRW
1
WriteReg[4:0]
07
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000001
ReadData2[31:0]
00000000
ExtSel
1
PCWre
0
RegDst
1
ALUSrcA
0
ALUSrcB
0
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
5
sign
0
zero
1
result[31:0]
00000000
DB[31:0]
00000000
extended[31:0]
00003800
ALUA[31:0]
00000001
ALUB[31:0]
00000000
IDataOut[31:0]
70c03800
DBDataSrc
0

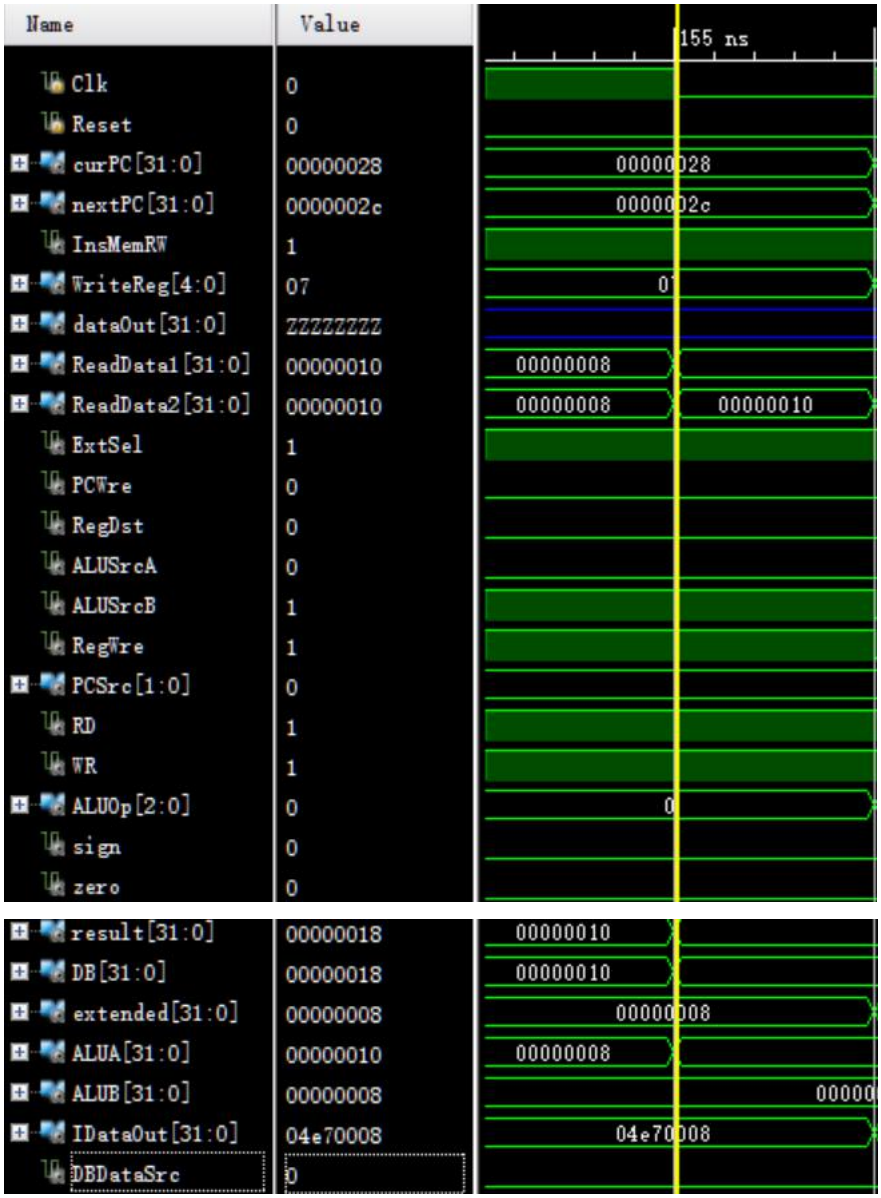
addi $7,$7,8


Name
Value
135 ns
Clk
0
Reset
0
curPC[31:0]
00000028
nextPC[31:0]
0000002c
InsMemRW
1
WriteReg[4:0]
07
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000008
ReadData2[31:0]
00000008
ExtSel
1
PCWre
0
RegDst
0
ALUSrcA
0
ALUSrcB
1
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
0
sign
0
zero
0
result[31:0]
00000010
DB[31:0]
00000010
extended[31:0]
00000008
ALUA[31:0]
00000008
ALUB[31:0]
00000008
IDataOut[31:0]
04e70008
DBDataSrc
0
00000028
0000002c
00000000
00000008
00000010
04e70008

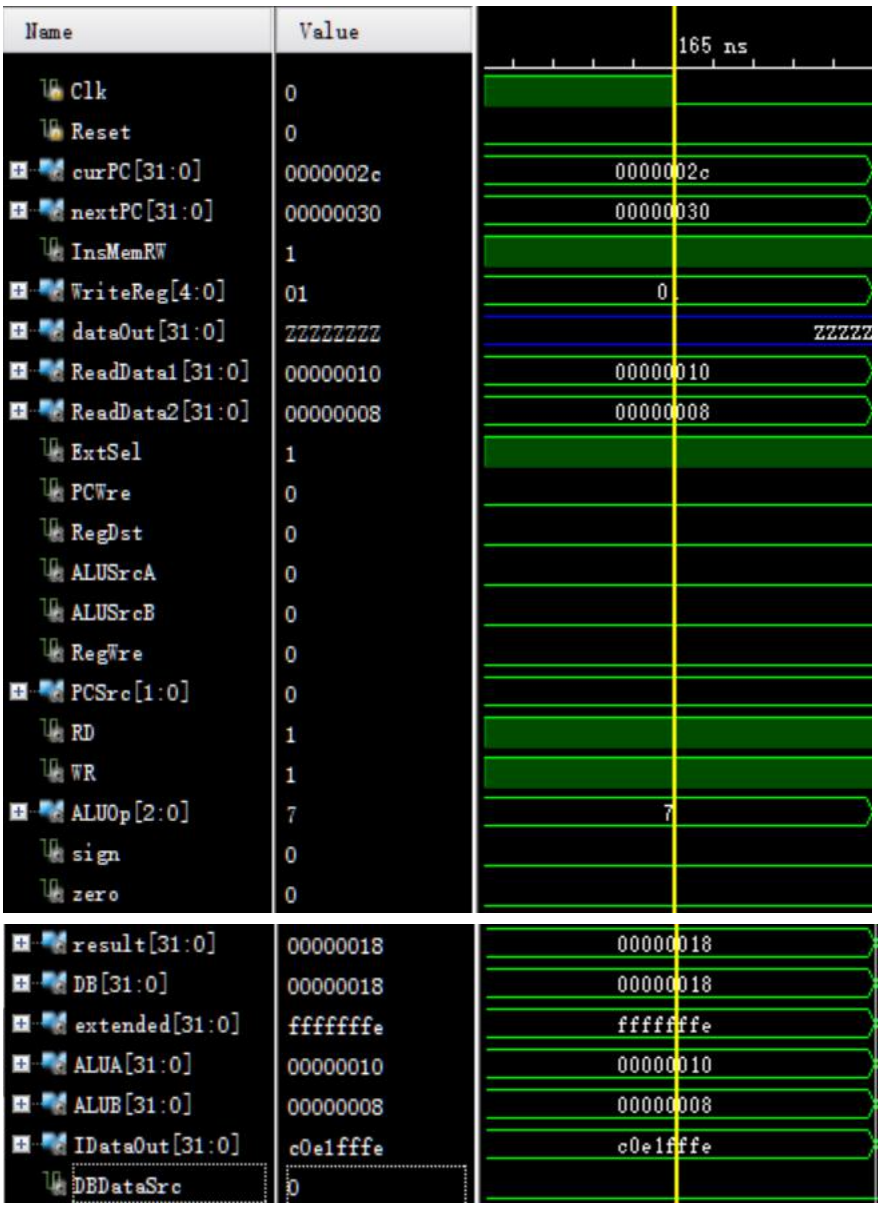
beq $7,$1,-2


Name
Value
145 ns
Clk
0
Reset
0
curPC[31:0]
0000002c
nextPC[31:0]
00000028
InsMemRW
1
WriteReg[4:0]
01
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000008
ReadData2[31:0]
00000008
ExtSel
1
PCWre
0
RegDst
0
ALUSrcA
0
ALUSrcB
0
RegWre
0
PCSrc[1:0]
1
RD
1
WR
1
ALUOp[2:0]
7
sign
0
zero
1
result[31:0]
00000000
DB[31:0]
00000000
extended[31:0]
fffffffe
ALUA[31:0]
00000008
ALUB[31:0]
00000008
IDataOut[31:0]
c0e1fffe
DBDataSrc
0

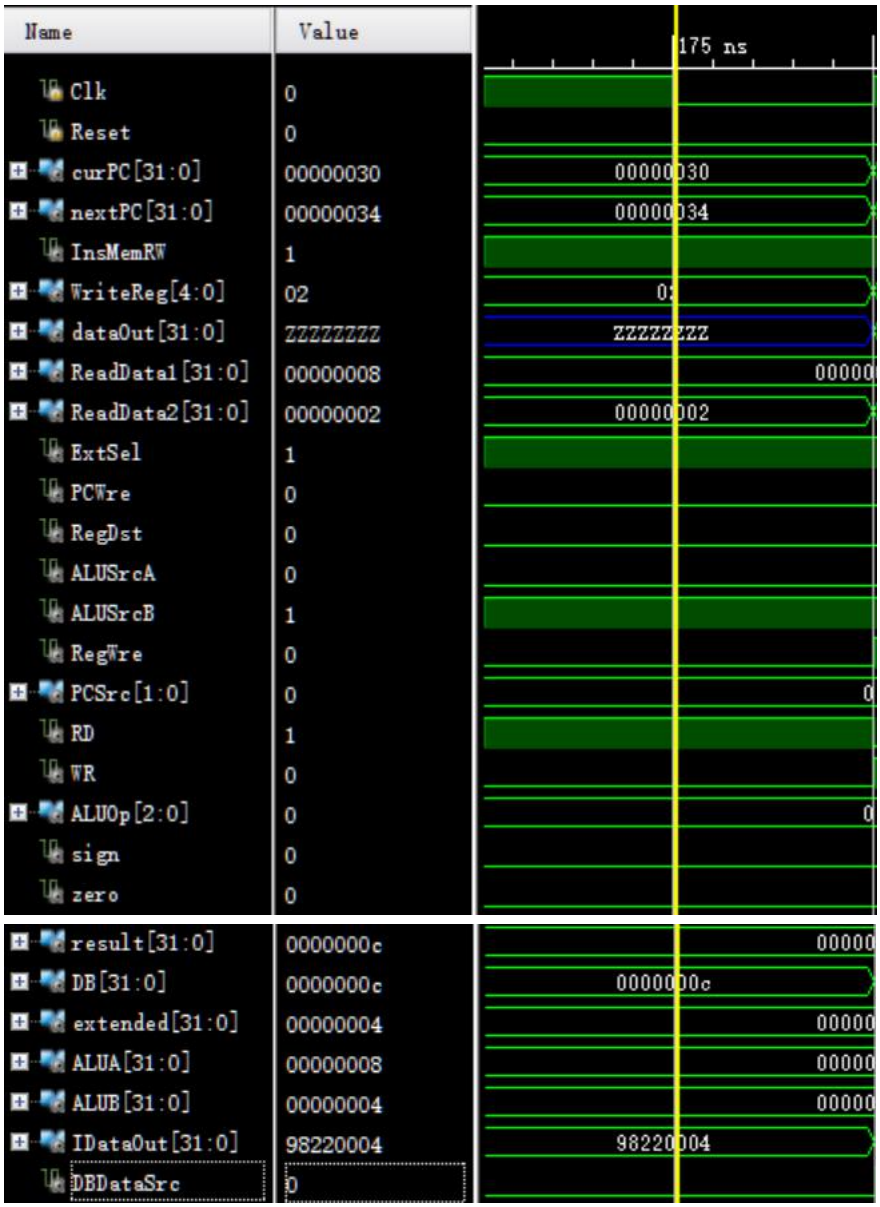
addi $7,$7,8


Name
Value
155 ns
Clk
0
Reset
0
curPC[31:0]
00000028
nextPC[31:0]
0000002c
InsMemRW
1
WriteReg[4:0]
07
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000010
ReadData2[31:0]
00000010
ExtSel
1
PCWre
0
RegDst
0
ALUSrcA
0
ALUSrcB
1
RegWre
1
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
0
sign
0
zero
0
result[31:0]
00000018
DB[31:0]
00000018
extended[31:0]
00000008
ALUA[31:0]
00000010
ALUB[31:0]
00000008
IDataOut[31:0]
04e70008
DBDataSrc
0

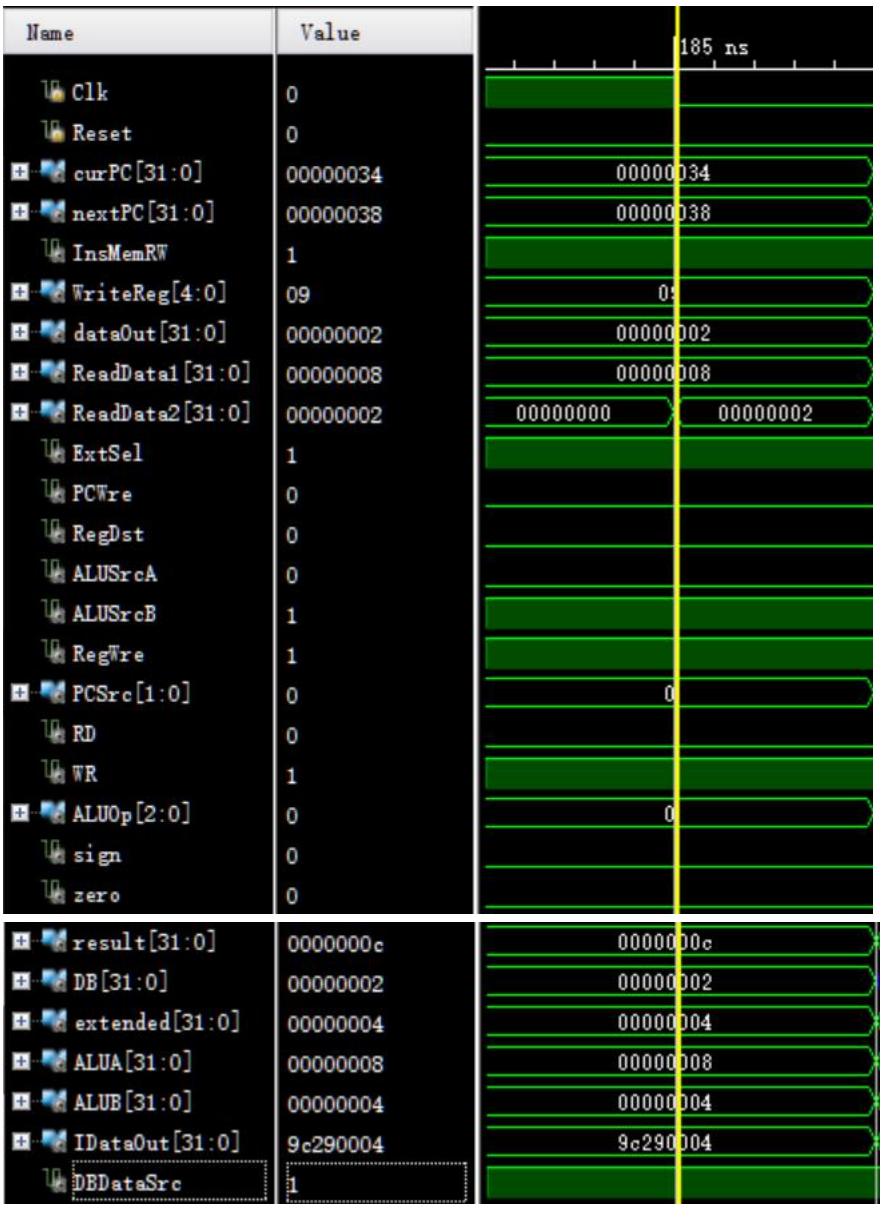
beq $7,$1,-2


Name
Value
165 ns
Clk
0
Reset
0
curPC[31:0]
0000002c
nextPC[31:0]
00000030
InsMemRW
1
WriteReg[4:0]
01
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000010
ReadData2[31:0]
00000008
ExtSel
1
PCWre
0
RegDst
0
ALUSrcA
0
ALUSrcB
0
RegWre
0
PCSrc[1:0]
0
RD
1
WR
1
ALUOp[2:0]
7
sign
0
zero
0
result[31:0]
00000018
DB[31:0]
00000018
extended[31:0]
fffffffe
ALUA[31:0]
00000010
ALUB[31:0]
00000008
IDataOut[31:0]
c0e1fffe
DBDataSrc
0

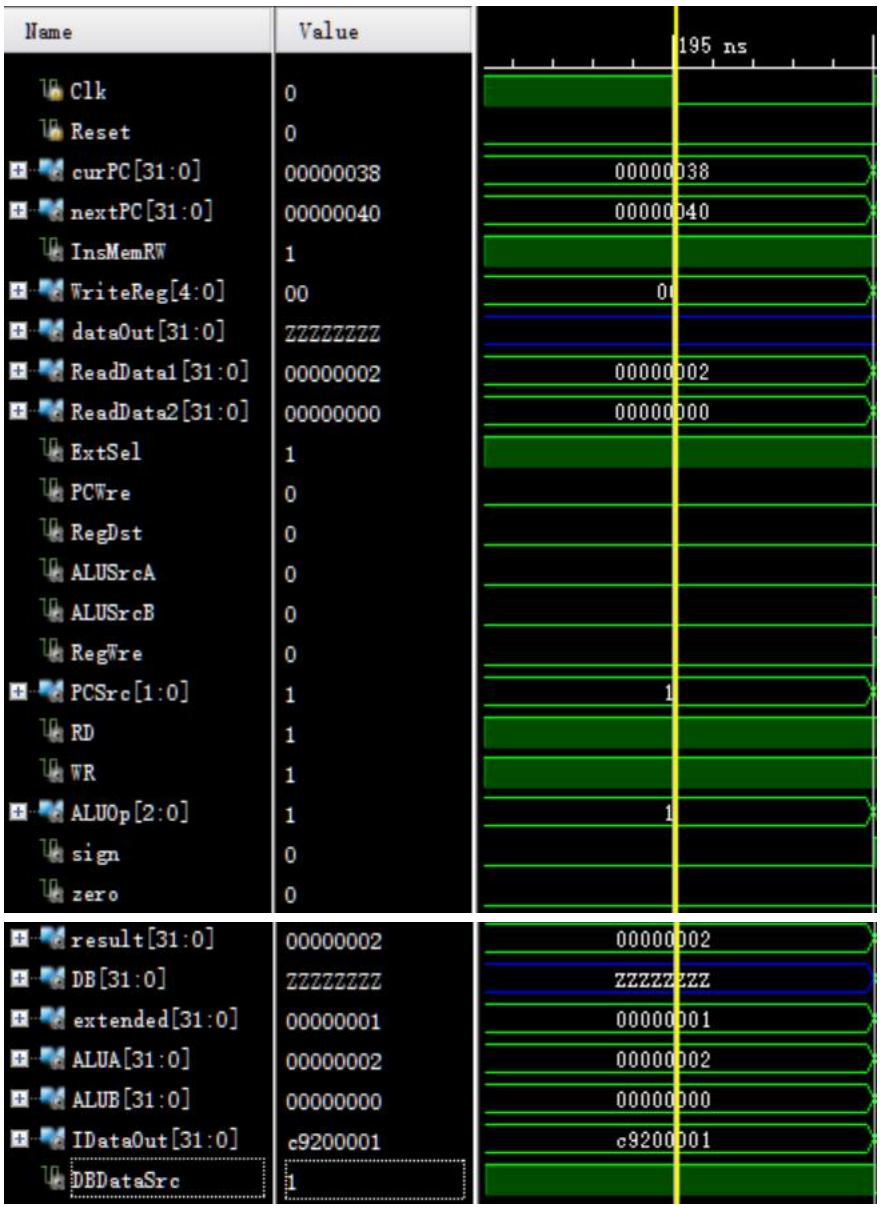
sw　$2,4($1)


Name
Value
175 ns
Clk
0
Reset
0
curPC[31:0]
00000030
nextPC[31:0]
00000034
InsMemRW
1
WriteReg[4:0]
02
dataOut[31:0]
ZZZZZZZZ
ReadData1[31:0]
00000008
ReadData2[31:0]
00000002
ExtSel
1
PCWre
0
RegDst
0
ALUSrcA
0
ALUSrcB
1
RegWre
0
PCSrc[1:0]
0
RD
1
WR
0
ALUOp[2:0]
0
sign
0
zero
0
result[31:0]
0000000c
DB[31:0]
0000000c
extended[31:0]
00000004
ALUA[31:0]
00000008
ALUB[31:0]
00000004
IDataOut[31:0]
98220004
DBDataSrc
0

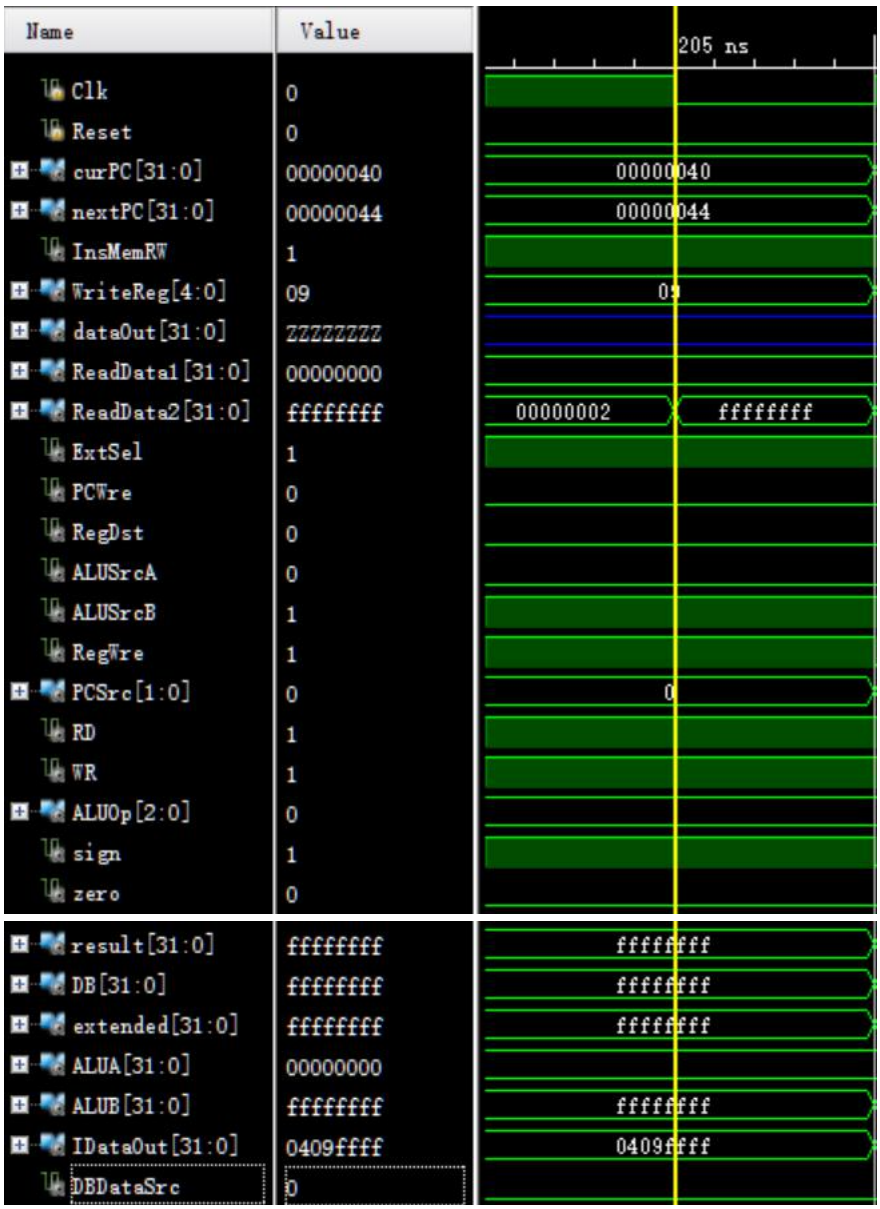
lw $9,4($1)


Name
Value
185 ns
Clk
0
Reset
0
curPC[31:0]
00000034
nextPC[31:0]
00000038
InsMemRW
1
WriteReg[4:0]
09
dataOut[31:0]
00000002
ReadData1[31:0]
00000008
ReadData2[31:0]
00000002
00000000
00000002
ExtSel
1
PCWre
0
RegDst
0
ALUSrcA
0
ALUSrcB
1
RegWre
1
PCSrc[1:0]
0
RD
0
WR
1
ALUOp[2:0]
0
sign
0
zero
0
result[31:0]
0000000c
DB[31:0]
00000002
extended[31:0]
00000004
ALUA[31:0]
00000008
ALUB[31:0]
00000004
IDataOut[31:0]
9c290004
DBDataSrc
1

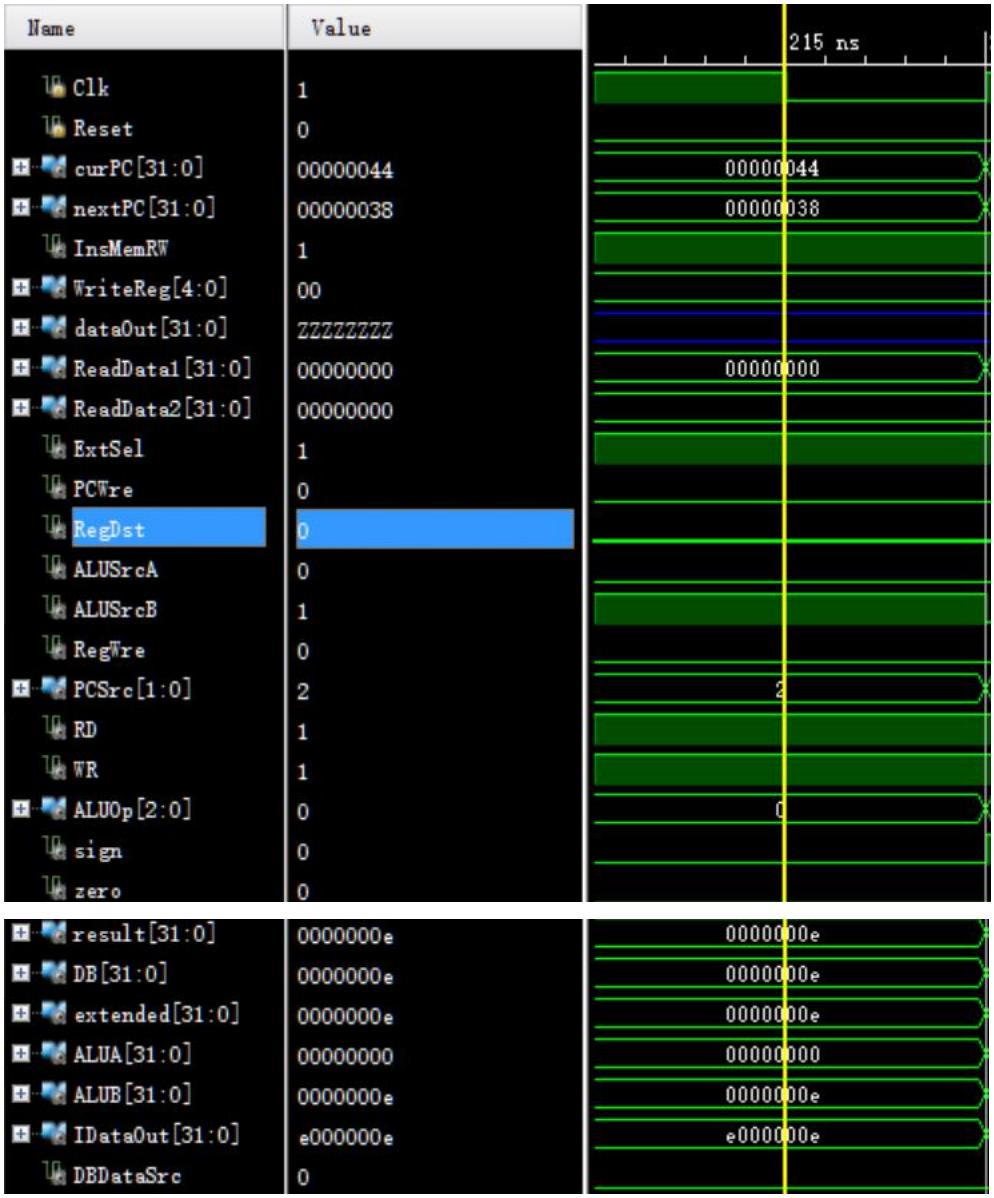
bgtz $9,1


Name
Value
195 ns
Clk 0
Reset 0
curPC[31:0] 00000038
nextPC[31:0] 00000040
InsMemRW 1
WriteReg[4:0] 00
dataOut[31:0] ZZZZZZZZ
ReadData1[31:0] 00000002
ReadData2[31:0] 00000000
ExtSel 1
PCWre 0
RegDst 0
ALUSrcA 0
ALUSrcB 0
RegWre 0
PCSrc[1:0] 1
RD 1
WR 1
ALUOp[2:0] 1
sign 0
zero 0
result[31:0] 00000002
DB[31:0] ZZZZZZZZ
extended[31:0] 00000001
ALUA[31:0] 00000002
ALUB[31:0] 00000000
IDataOut[31:0] c9200001
DBDataSrc 1

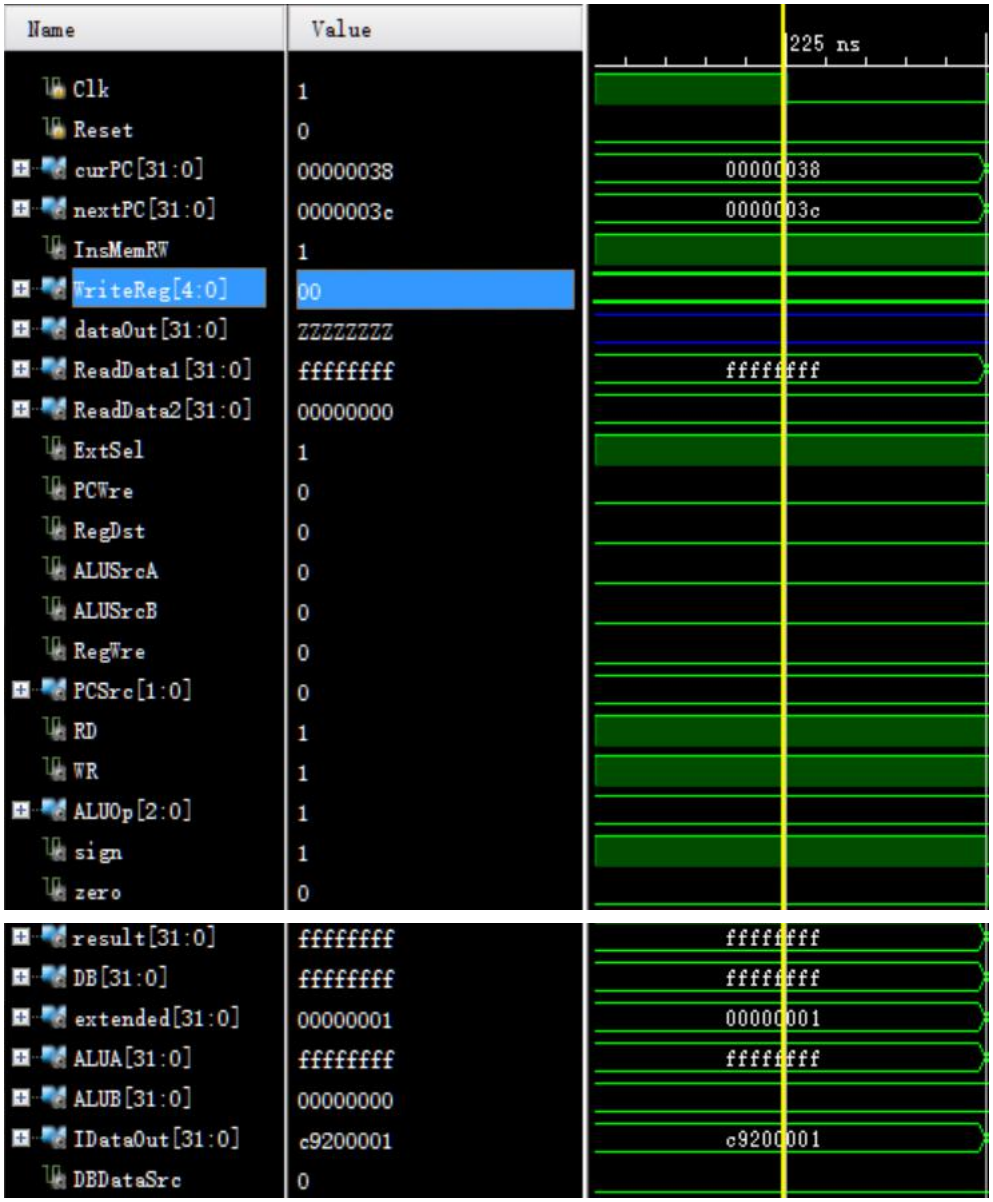
addi $9,$0,-1



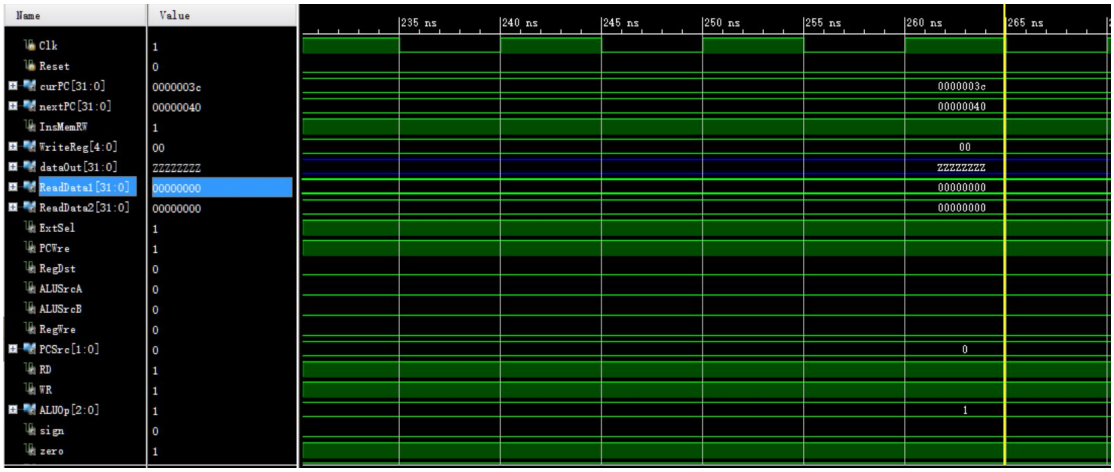
j　0x00000038

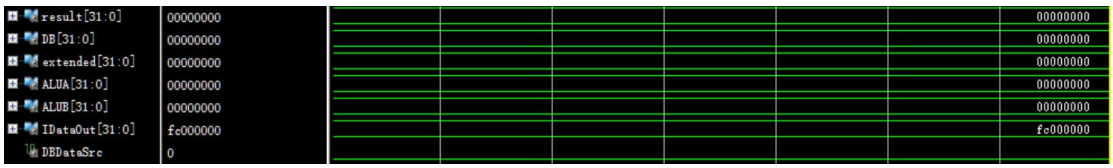

Name
Value
215 ns
Clk 1
Reset 0
curPC[31:0] 00000044
nextPC[31:0] 00000038
InsMemRW 1
WriteReg[4:0] 00
dataOut[31:0] ZZZZZZZZ
ReadData1[31:0] 00000000
ReadData2[31:0] 00000000
ExtSel 1
PCWre 0
RegDst 0
ALUSrcA 0
ALUSrcB 1
RegWre 0
PCSrc[1:0] 2
RD 1
WR 1
ALUOp[2:0] 0
sign 0
zero 0
result[31:0] 0000000e
DB[31:0] 0000000e
extended[31:0] 0000000e
ALUA[31:0] 00000000
ALUB[31:0] 0000000e
IDataOut[31:0] e000000e
DBDataSrc 0

bgtz $9,1


Name
Value
225 ns
Clk 1
Reset 0
curPC[31:0] 00000038
nextPC[31:0] 0000003c
InsMemRW 1
WriteReg[4:0] 00
dataOut[31:0] zzzzzzzz
ReadData1[31:0] ffffffff
ReadData2[31:0] 00000000
ExtSel 1
PCWre 0
RegDst 0
ALUSrcA 0
ALUSrcB 0
RegWre 0
PCSrc[1:0] 0
RD 1
WR 1
ALUOp[2:0] 1
sign 1
zero 0
result[31:0] ffffffff
DB[31:0] ffffffff
extended[31:0] 00000001
ALUA[31:0] ffffffff
ALUB[31:0] 00000000
IDataOut[31:0] c9200001
DBDataSrc 0

halt


Name
Value
235 ns
240 ns
245 ns
250 ns
255 ns
260 ns
265 ns
Clk 1
Reset 0
curPC[31:0] 0000003c
nextPC[31:0] 00000040
InsMemRW 1
WriteReg[4:0] 00
dataOut[31:0] zzzzzzzz
ReadData1[31:0] 00000000
ReadData2[31:0] 00000000
ExtSel 1
PCWre 1
RegDst 0
ALUSrcA 0
ALUSrcB 0
RegWre 0
PCSrc[1:0] 0
RD 1
WR 1
ALUOp[2:0] 1
sign 0
zero 1



3、**实现**：仿真结果正确后，确认CPU能正常工作，因此让CPU在Basys3板上运行只需增加一个显示模块，改变顶层模块的输入输出，并让CPU的输入输出与板上的接口对应起来。

增加的显示模块：

display，输入为时钟信号，显示内容选择信号，当前PC值，下次更新的PC值，当前运行指令，rs寄存器值，rt寄存器值，ALU计算结果，DB总线数据；输出为LED位选信号、段选信号

```
always@(posedge clk)
begin
    if(count_clk == 26'hffffff2)
    begin
        count_clk = 26'h0;
    end
    else
    begin
        count_clk = count_clk+1'h1;
    end
end//分频
……
always@(sel)//显示内容选择
begin
    case(sel)
        4'b0111:
        begin
        if(SW_in==2'b00)
            disp_temp=curPC[7:4];
        else if(SW_in==2'b01)
            disp_temp={{3{0}},IDataOut[25]};
```

```
else if(SW_in==2'b10)
        disp_temp={{3{0}},IDataOut[20]};
else
        disp_temp=result[7:4];
end
4'b1011:
begin
if(SW_in==2'b00)
        disp_temp=curPC[3:0];
else if(SW_in==2'b01)
        disp_temp=IDataOut[24:21];
else if(SW_in==2'b10)
        disp_temp=IDataOut[19:16];
else
        disp_temp=result[3:0];
end
4'b1101:
begin
if(SW_in==2'b00)
        disp_temp=nextPC[7:4];
else if(SW_in==2'b01)
        disp_temp=ReadData1[7:4];
else if(SW_in==2'b10)
        disp_temp=ReadData2[7:4];
else
        disp_temp=DB[7:4];
end
4'b1110:
begin
if(SW_in==2'b00)
```

```
                disp_temp=nextPC[3:0];
            else if(SW_in==2'b01)
                disp_temp=ReadData1[3:0];
            else if(SW_in==2'b10)
                disp_temp=ReadData2[3:0];
            else
                disp_temp=DB[3:0];
            end
            default:disp_temp=4'b1010;
        endcase
end
```

将代码综合、分配好接口、生成bit文件并下载到板上后，接通电源，运行CPU。

指令	当前PC	下条PC	RS寄存器地址	RS寄存器值	RT寄存器地址	RT寄存器值	ALU 结果	DB总线数据
addi $1,$0,8	00	04	00	00	01	08	08	08
ori $2,$0,2	04	08	00	00	02	02	02	02
add $3,$2,$1	08	0c	02	02	01	08	0A	0A
sub $5,$3,$2	0c	10	03	10	02	02	08	08
and $4,$5,$2	10	14	05	08	02	02	00	00
or	14	18	04	00	02	02	02	02

$8,$4,$2								
sll $8,$8,1	18	1c	00	00	08	02	04	04
bne $8,$1,-2	1c	18	08	04	01	08	0c	00
sll $8,$8,1	18	1c	00	00	08	04	08	08
bne $8,$1,-2	1c	20	08	08	01	08	00	00
slt $6,$2,$1	20	24	02	02	01	08	01	01
slt $7,$6,$0	24	28	06	01	00	00	00	00
addi $7,$7,8	28	2c	07	00	07	00	08	08
beq $7,$1,-2	2c	28	07	08	01	08	00	00
addi $7,$7,8	28	2c	07	08	07	08	10	10
beq $7,$1,-2	2c	30	07	10	01	08	18	00
sw $2,4($1)	30	34	01	08	02	02	0c	00
lw $9,4($1)	34	38	01	08	09	00	0c	02
bgtz	38	40	09	02	00	00	02	00

$9,1								
addi $9,$0,-1	40	44	00	00	09	02	FF	FF
j 0x0000 0038	44	38	00	00	00	00	00	00
bgtz $9,1	38	3c	09	FF	00	00	FF	00
halt	3c	40	00	00	00	00	00	00

六. 实验心得

体会和建议:

（1）简单的模块的编写也必须认真对待，bug往往出现在最掉以轻心的地方。在编写ALU模块时，为了图方便，我直接复制粘贴了老师所给的参考代码，可在debug时发现，CPU能运行，但计算的结果总是有错误，本以为是在编写复杂的控制单元时代码有错，检查多次无果后，才将目光转移到ALU模块；检查过后，发现ALU模块的操作码对应的功能与设计的不符，且ALU没有左移功能；修改过后再次进行仿真，发现CPU进入了死循环，对各个模块进行排查后并未发现明显bug，再对波形图数值进行观察，发现左移1位的运算结果会随着循环次数增加而改变（一次左移多位），再次对ALU进行检查，发现左移运算的操作数搞反了，修改后，仿真结果基本正确。

（2）要仔细各个模块的触发条件，当触发条件书写正确时，模块能按我们预想地工作。在仿真的初始阶段，发现每条跳转语句都跳转失败，检查波形图中控制信号的值后，发现控制信号赋值出错；返回检查控制单元，发现控制单元没有及时地更新控制信号，即在每条指令中，控制单元都只被调用了一次，思考后发现，控制单元的触发条件出错，原为指令操作码改变触发，改为任一输入改变出发后，跳转指令可正常执行。

（3）CPU状态的初始化正确与否，决定了CPU后续运算的正确与否。在仿真时，发现CPU进入死循环，检查波形图后，发现原因是第一条测试指令只被执行了一半，即运算结果正确，但没有写回到目的寄存器；检查时钟信号，发现由于初始化不正确，第一条指令的

运行周期只有正常的一半，修改后仿真结果正确；而在烧板时，发现LED始终显示为0000，按下按键也没有丝毫反应，因仿真结果正确，考虑是初始化出错，将PC更新的触发条件改为时钟的下降沿，写内存、写寄存器的触发条件改为时钟的上升沿，再进行烧录，CPU正常运行。（猜测原因是U18产生的是一个负脉冲）