计算机组成原理 实验报告

姓名：魏钊　学号：PB18111699　实验日期：2020-5-10

一、实验题目：

实验三 单周期 CPU

二、实验目的：

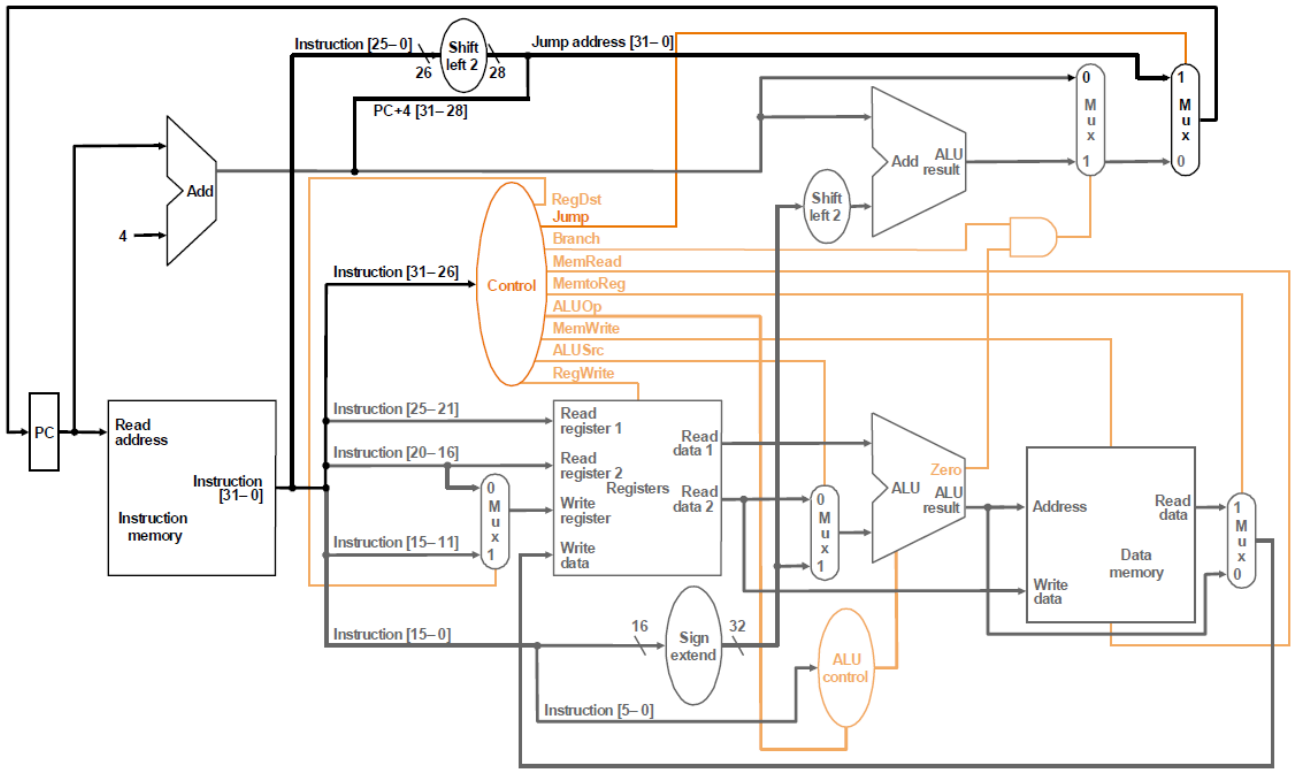
1. 理解计算机硬件的基本组成、结构和工作原理；
2. 掌握数字系统的设计和调试方法；
3. 熟练掌握数据通路和控制器的设计和描述方法。

三、实验平台：

ISE / Vivado（暂不支持其他 Verilog HDL 开发环境的检查）

四、实验过程：

首先根据下图设计出 CPU 模块，CPU 中较为重要的是控制器。根据相关指令需要的信号设计控制器。



CPU：

Control：

```
module Control(
    input   [5:0] OpCode,
    output  jump,
    output  RegDst,
    output  Branch,
    output  MemRead,
    output  MemtoReg,
    output  ALUOp,
    output  MemWrite,
    output  ALUSrc,
    output  RegWrite
    );
assign RegDst=(OpCode == 6'b0)?1'b1:1'b0;//add时写地址为rd否则为rt
assign jump=(OpCode == 6'b10)?1'b1:1'b0;//是否为J指令
assign Branch=(OpCode == 6'b100)?1'b1:1'b0;//是否为beq指令
assign MemRead=(OpCode == 6'b100011)?1'b1:1'b0;//是否为lw
assign MemtoReg=(OpCode == 6'b100011)?1'b1:1'b0;//是否为lw
assign ALUOp=(OpCode == 6'b100)?1'b1:1'b0;//beq时为减法判断是否为0，否则均为加法
assign MemWrite=(OpCode == 6'b101011)?1'b1:1'b0;//是否为sw
assign ALUSrc=((OpCode == 6'b0)|(OpCode == 6'b100))? 1'b0 : 1'b1;//add和beq时为0，其他为1
assign RegWrite=(OpCode == 6'b1000|OpCode == 6'b0|OpCode == 6'b100011)?1'b1:1'b0;//是否写回寄存器
endmodule
```

ALUControl：

```
module ALUControl(
    input   ALUOp,
    input [5:0] Funct,
    output[2:0] Sign//ALU控制信号
    );
assign Sign=(ALUOp==1'b1) ? 3'b001:3'b000;//ALUop为1时beq判断两数是否为0，为减法，否则为加法
endmodule
```

MUX：

```
module MUX
#(parameter WIDTH = 32)
(
    input   m,
    input   [WIDTH-1:0] a,b,
    output  [WIDTH-1:0] y
);
assign y=(m==1)?b:a;
endmodule
```

寄存器堆：

```
module register_file
#(parameter WIDTH = 32)
(
    input clk,                  //时钟（上升沿有效）
    input [4:0] ra0,                //读端口0地址
    output [WIDTH-1:0] rd0,     //读端口0数据
    input [4:0] ra1,                //读端口1地址
    output [WIDTH-1:0] rd1,     //读端口1数据
    input [4:0] wa,                 //写端口地址
    input we,                   //写使能，高电平有效
    input [WIDTH-1:0] wd        //写端口数据
);
reg [4:0] addr_reg;
reg [WIDTH-1:0] mem[0:31];

assign rd0=mem[ra0];//异步读
assign rd1=mem[ra1];

always@(*)
    mem[0]=0;

always@(posedge clk)
begin
    if(we)
        if(~(wa==5'b0))
            mem[wa]<=wd;//同步写
    mem[0]=0;
end
endmodule
```

ALU:

```
module ALU
    #(parameter WIDTH = 32)
    (
    output reg [WIDTH-1:0] y,       //运算结果
    output reg zf,                  //零标志
    output reg cf,                  //进位/借位标志
    output reg of,                  //溢出标志
    output reg sf,                  //最高位是否为1
    input [WIDTH-1:0] a, b,     //两操作数
    input [2:0] m                       //操作类型
    );
    always@(*)
        case (m)
            3'b000://加
            begin
                {cf,y}=a+b;
                of=(~a[WIDTH-1]&~b[WIDTH-1]&y[WIDTH-1])|(a[WIDTH-1]&b[WIDTH-1]&~y[WIDTH-1]);//溢出标志
                sf=y[WIDTH-1];//符号标志
                zf=~|y;//零标志
            end

            3'b001://减
            begin
                {cf,y}=a-b;
                of=(~a[WIDTH-1]&b[WIDTH-1]&y[WIDTH-1])|(a[WIDTH-1]&~b[WIDTH-1]&~y[WIDTH-1]);//溢出标志
                sf=y[WIDTH-1];//符号标志
                zf=~|y;//零标志
            end
```

```
            3'b010://与
            begin
                cf=1'bx;
                of=1'bx;
                y=a&b;
                sf=y[WIDTH-1];//符号标志
                zf=~|y;//零标志
            end

            3'b011://或
            begin
                cf=1'bx;
                of=1'bx;
                y=a|b;
                sf=y[WIDTH-1];//符号标志
                zf=~|y;//零标志
            end

            3'b100://异或
            begin
                cf=1'bx;
                of=1'bx;
                y=a^b;
                sf=y[WIDTH-1];//符号标志
                zf=~|y;//零标志
            end

            default:
            begin
                  cf=1'bx;
                  of=1'bx;
                  y=32'bx;
                  sf=1'bx;
                  zf=1'bx;
              end
        endcase
endmodule
```

指令存储器：256 x 32位ROM，IP例化，分布式存储器

数据存储器：256 x 32位RAM，IP例化，分布式存储器

CPU:

为了配合 DBU，输入输出端口做出了改变。

```
module CPU(
    input   rst,
    input   clk,
    input   read,//使用DBU读CPU内部数据使能
    input   [7:0]   m_rf_addr,//地址
    output  [31:0]  rf_data,//寄存器数据
    output  [31:0]  m_data,//mem数据
    output  reg  [31:0] PC,
    output  [31:0] PC_next,
    output  [31:0] Instruction,
    output  [31:0] Read_data1,//寄存器堆读出数据
    output  [31:0] Read_data2,
    output  [31:0] Alu_out,//ALU输出
    output  [31:0] Read_data,//DM中读出数据
    //控制信号
    output  jump,
    output  RegDst,
    output  Branch,
    output  MemRead,
    output  MemtoReg,
    output  [2:0] Sign,//ALU控制
    output  MemWrite,
    output  ALUSrc,
    output  RegWrite,
    output  zero
    );
```

```
wire [31:0] PC_plus_4;//PC+4
wire [31:0] Jump_addr;//跳转地址
wire [4:0]  Write_R;//要写的寄存器地址
wire [31:0] Write_D;//写端口数据
wire [31:0] OP;//ALU的第二个操作数
wire [31:0] add_imm;//指令15-0拓展
wire [31:0] mux0_out;//MUX0的输出
wire zf,cf,of;//ALU相关输出
wire [4:0]  rf_addr;
wire [7:0]  dm_addr;
reg  [31:0] NPC;

//控制信号
wire  ALUOp;

assign  m_data=Read_data;
assign  rf_data=Read_data1;
assign  rf_addr=(read==1)?m_rf_addr[4:0]:Instruction[25:21];
assign  dm_addr=(read==1)?m_rf_addr:Alu_out[9:2];

IM im(.a(PC[9:2]),.spo(Instruction));
DM dm(.a(dm_addr),.d(Read_data2),.clk(~clk),.we(MemWrite),.spo(Read_data));

Control control(.OpCode(Instruction[31:26]),.jump(jump),.RegDst(RegDst),.Branch(Branch),
                .MemRead(MemRead),.MemtoReg(MemtoReg),.ALUOp(ALUOp),.MemWrite(MemWrite),
                .ALUSrc(ALUSrc),.RegWrite(RegWrite));

ALUControl Alucontrol(.ALUOp(ALUOp),.Funct(Instruction[5:0]),.Sign(Sign));


register_file Registers(.clk(~clk),.ra0(rf_addr),.rd0(Read_data1),.ra1(Instruction[20:16]),
                        .rd1(Read_data2),.wa(Write_R),.we(RegWrite),.wd(Write_D));

ALU Alu(.m(Sign),.a(Read_data1),.b(OP),.y(Alu_out),.zf(zf),.cf(cf),.of(of));
//依据数据通路图，从上到下，从左到右MUX
MUX mux0(.m(zero&Branch),.a(PC_plus_4),.b(({add_imm[29:0],2'b00}+PC_plus_4)),.y(mux0_out));
MUX mux1(.m(jump),.a(mux0_out),.b(Jump_addr),.y(PC_next));
MUX mux2(.m(RegDst),.a(Instruction[20:16]),.b(Instruction[15:11]),.y(Write_R));
MUX mux3(.m(ALUSrc),.a(Read_data2),.b(add_imm),.y(OP));
MUX mux4(.m(MemtoReg),.a(Alu_out),.b(Read_data),.y(Write_D));

assign PC_plus_4=PC+32'b100;//PC+4
assign Jump_addr={PC_plus_4[31:28], Instruction[25:0], 2'b00};
assign add_imm={16'b0,Instruction[15:0]};
assign zero=(zf==1'b1&&cf==1'b0&&of==1'b0);

always @(posedge rst or posedge clk)
        if (rst)
            PC <= 32'b0;
        else
            PC <= NPC;

always @(negedge clk or posedge rst)
    if(rst)
        NPC<=32'b0;
    else
        NPC<=PC_next;
endmodule
```

DBU：

按钮去抖动：

```
module Jitter_clr(
    input clk,
    input button,
    output button_clean
    );
parameter jitter_clr_bit=18;

reg [jitter_clr_bit:0] cnt;

always@(posedge clk)
begin
    if(button==0)
        cnt<=0;
    else
        if(cnt<(1<<jitter_clr_bit))
            cnt<=cnt+1;
end

assign button_clean=cnt[jitter_clr_bit];
endmodule
```

取边沿：

```
module EDG(
    input clk,
    input y,
    input rst,
    output p
    );
    parameter S0=2'b00;
    parameter S1=2'b01;
    parameter S2=2'b10;
    reg [1:0] state=S0,next_state;
    assign p=(state==S1);
    always@(posedge clk or posedge rst)
        if(rst)
            state<=S0;
        else
            state<=next_state;
    always@(*)
    begin
        case(state)
            S0:if(y)
                    next_state=S1;
                else
                    next_state=S0;
            S1:if(y)
                    next_state=S2;
                else
                    next_state=S0;
            S2:if(y)
                    next_state=S2;
                else
                    next_state=S0;
         endcase
     end
endmodule
```

DBU:

```
module DBU(
    input   succ,
    input   clk,
    input   rst,
    input   read,//读使能
    input   step,
    input   [2:0] sel,
    input   m_rf,
    input   inc,
    input   dec,
    output  reg [15:0] led,//16位led
    output  [7:0]  an,//数码管片选
    output  reg [31:0]  seg//8个八段数码管
    );
wire    step_clean,inc_clean,dec_clean;
wire    step_fin,inc_fin,dec_fin;
wire    clock;
wire    [31:0] PC;
wire    [31:0] PC_next,Instruction,Read_data2,Read_data1,Alu_out;
wire    [31:0] Read_data;//DM中读出数据
reg     [7:0]   m_rf_addr;
wire    [31:0]  rf_data;
wire    [31:0]  m_data;
wire    jump;
wire    RegDst;
wire    Branch;
wire    MemRead;
wire    MemtoReg;
wire    [2:0] ALUOp;//ALU控制
wire    MemWrite;
wire    ALUSrc;
wire    RegWrite;
wire    zero;
//相关按钮去抖动
Jitter_clr jitter_clr1(.clk(clk),.button(step),.button_clean(step_clean));
Jitter_clr jitter_clr2(.clk(clk),.button(inc),.button_clean(inc_clean));
Jitter_clr jitter_clr3(.clk(clk),.button(dec),.button_clean(dec_clean));
//相关按钮取边沿
EDG edg1(.clk(clk),.rst(rst),.y(step_clean),.p(step_fin));
EDG edg2(.clk(clk),.rst(rst),.y(inc_clean),.p(inc_fin));
EDG edg3(.clk(clk),.rst(rst),.y(dec_clean),.p(dec_fin));

CPU cpu(.rst(rst),.clk(clock),.Read_data1(Read_data1),.PC(PC),.PC_next(PC_next),
        .Instruction(Instruction),.read(read),.m_rf_addr(m_rf_addr),.rf_data(rf_data),
        .m_data(m_data),.jump(jump),.RegDst(RegDst),.Branch(Branch),.MemRead(MemRead),
        .MemtoReg(MemtoReg),.Sign(ALUOp),.MemWrite(MemWrite),.ALUSrc(ALUSrc),
        .RegWrite(RegWrite),.zero(zero),.Read_data2(Read_data2),.Alu_out(Alu_out),
        .Read_data(Read_data));

assign clock=(read==1)?1'b0:((succ==1)?clk:step_fin);

always@(posedge clk or posedge rst)
begin
    if(rst)
        m_rf_addr<=8'b0;
    else if((inc_fin==1'b1)&(dec_fin==1'b0))
    m_rf_addr<=m_rf_addr+8'b1;
    else if((dec_fin==1'b1)&(inc_fin==1'b0))
```

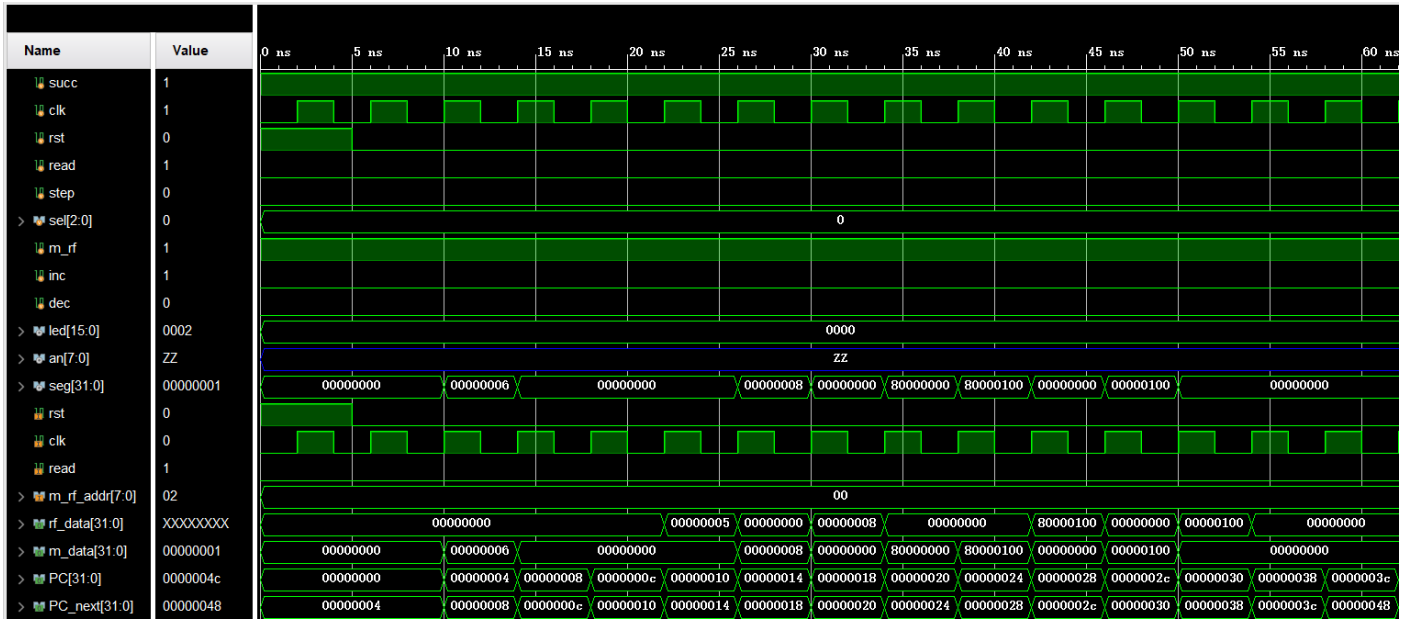
```
        m_rf_addr<=m_rf_addr-8'b1;
    else
        m_rf_addr<=m_rf_addr;
end

always@(*)
begin
    if(sel==3'b0)
    begin
        led={8'b0,m_rf_addr};
        if(m_rf==1)
            seg=m_data;
        else
            seg=rf_data;
    end
    else
    begin
    led={2'b00,sel,m_rf,jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zero};
    if(sel==3'b001)
    seg=PC_next;
    else if(sel==3'b010)
        seg=PC;
    else if(sel==3'b011)
        seg=Instruction;
    else if(sel==3'b100)
        seg=Read_data1;
    else if(sel==3'b101)
        seg=Read_data2;
    else if(sel==3'b110)
        seg=Alu_out;
    else if(sel==3'b111)
        seg=Read_data;
    end
end
endmodule
```

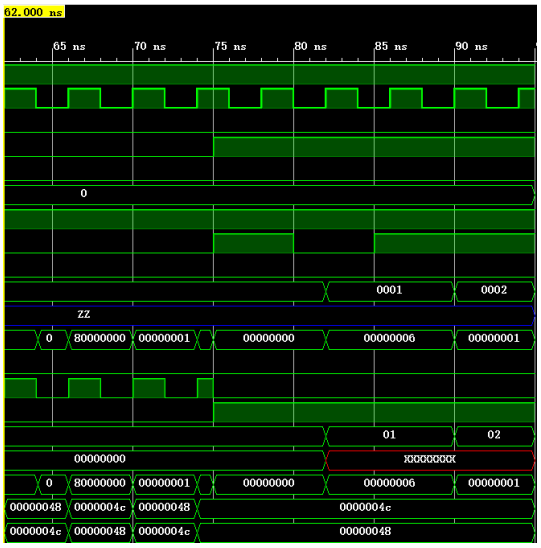
五、实验结果：

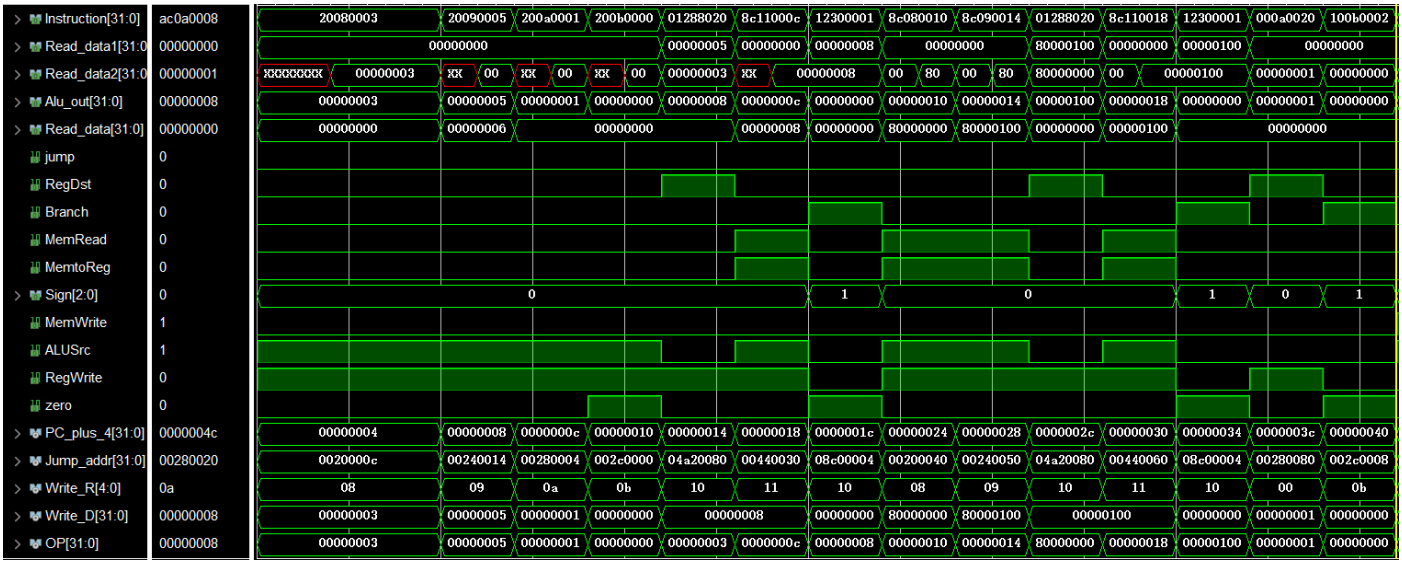
根据提供的 COE 文件，最终在 mem[8]中所存的是 1 即可。

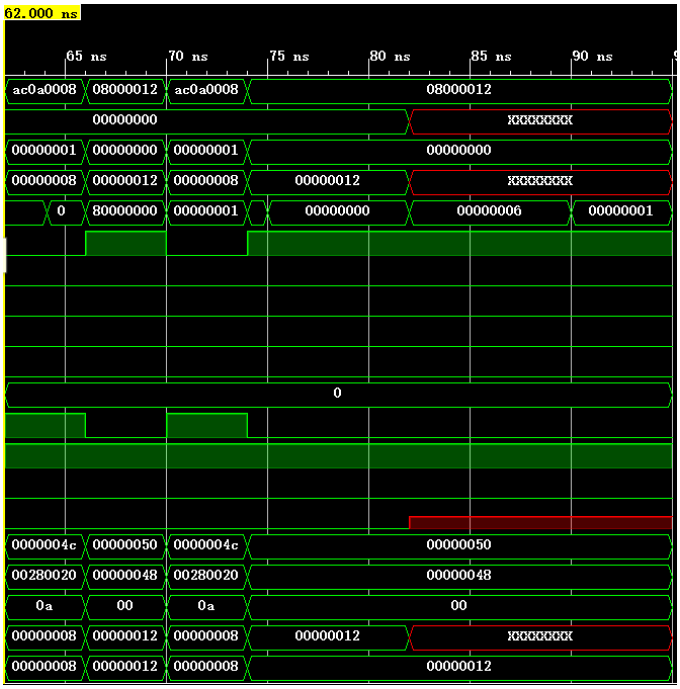
```
module DBU_test;
    reg     succ;
    reg     clk;
    reg     rst;
    reg     read;
    reg     step;
    reg     [2:0] sel;
    reg     m_rf;
    reg     inc;
    reg     dec;
    wire    [15:0] led;
    wire    [7:0]  an;//数码管片选
    wire    [31:0]  seg;//8个八段数码管
    DBU dbu(.succ(succ),.rst(rst),.clk(clk),.read(read),.step(step),
            .sel(sel),.m_rf(m_rf),.inc(inc),.dec(dec),.led(led),.an(an),.seg(seg));
    always
        #2 clk = ~ clk;
    initial
    begin
    #0  clk=0;
    #0  succ=1'b1;
    #0  step=0;
    #0  rst=1'b1;
    #0  read=1'b0;
    #0  sel=0;
    #0  m_rf=1;
    #0  inc=0;
    #0  dec=0;
    #5  rst=1'b0;
```

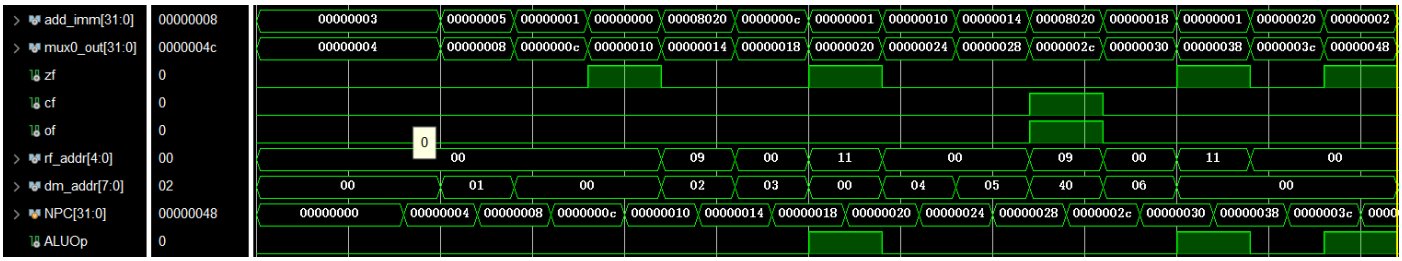
```
    #70 read=1'b1;
    #0  inc=1'b1;
    #5  inc=1'b0;
    #5  inc=1'b1;
    #10 $finish;
    end
endmodule
```

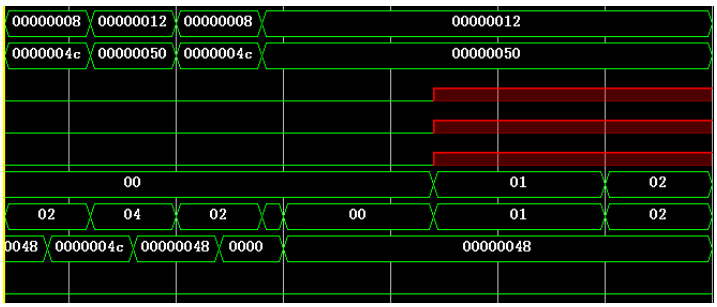












可以看到最终 seg 输出是 1，证明仿真测试成功。

六、 心得体会：

通过本次实验理解了计算机硬件的基本组成、结构和工作原理；掌握了数字系统的设计和调试方法；熟练掌握了数据通路和控制器的设计和描述方法。