COD_Lab3 单周期 CPU 设计

PB18111707 吕瑞

一. 实验目标

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。

二. 实验内容

1. 单周期 CPU 的设计与仿真

待设计的单周期CPU可以执行如下6条指令：

• **add**: rd <- rs + rt; op = 000000, funct = 100000

R类型：

31-26	25-21	20-16	15-11	10-6	5-0
op	rs	rt	rd	shamt	func
6位	5位	5位	5位	5位	6位

• **addi**: rt <- rs + imm; op = 001000

lw: rt <- M(rs + addr); op = 100011

sw: M(rs + addr) <- rt; op = 101011

beq: if (rs = rt) then pc <- pc + 4 + addr << 2

else pc <- pc + 4; op = 000100

I类型：

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

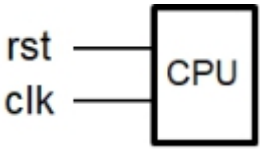
• **j**: pc <- (pc+4)[31:28] | (add<<2)[27:0]; op = 000010

J类型：

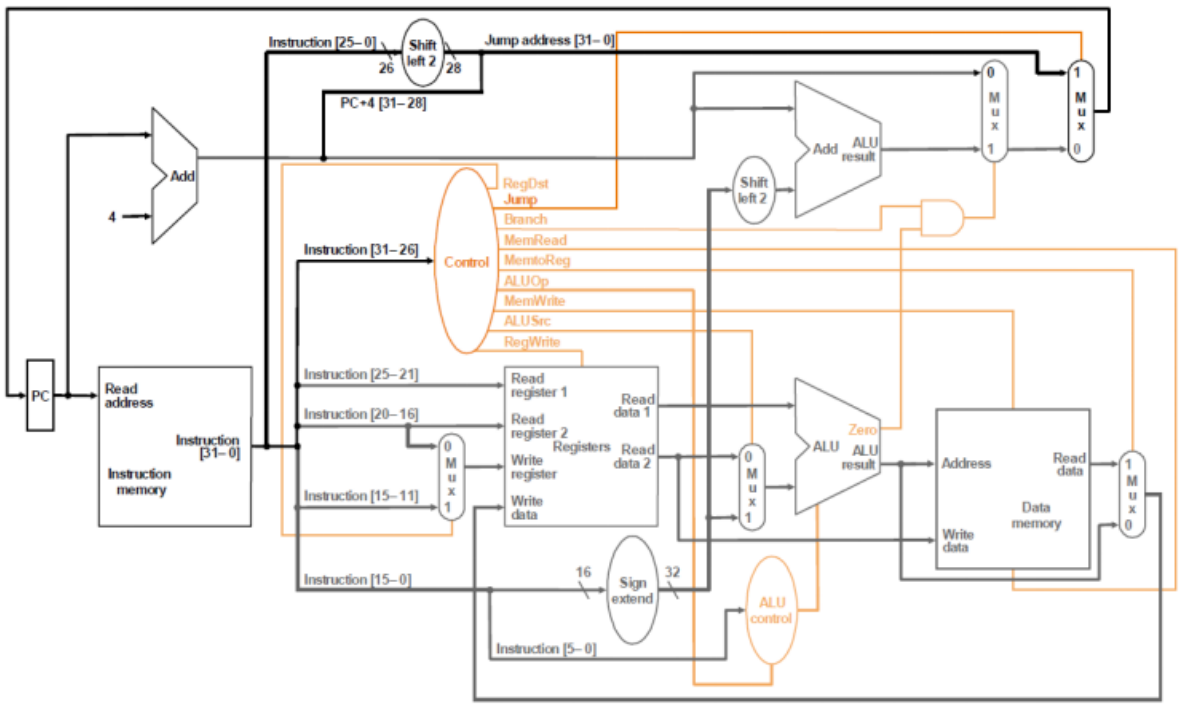
31-26	25-0
op	address
6位	26位

待设计的CPU的逻辑符号如图所示，端口声明如下：

```
module cpu_one_cycle    //单周期CPU

(input clk,          //时钟（上升沿有效）
 input rst               //异步复位，高电平有效
);

……

endmodule
```



（1）设计 CPU 数据通路如下：



（2）控制单元和 ALU 设计

表-2 控制信号

指令	RegDst	Jump	Branch	MemRead	MemtoReg	MemWrite	ALUSrc	RegWrite	ALUOp
add	1	0	0	0	0	0	0	1	0
addi	0	0	0	0	0	0	1	1	0
lw	0	0	0	1	1	0	1	1	0
sw	x	0	0	0	x	1	1	0	0
beq	x	0	1	0	x	0	0	0	1
j	x	1	x	0	x	0	x	0	x

表-2 ALU模块功能表

m	y
000	a + b
001	a - b
010	a & b
011	a | b
100	a ^ b
其他	x

```
module ControlUnit(
    input [5:0]Opcode,
    output RegDst,
    output Jump,
    output Branch,
    output MemRead,
    output MemtoReg,
    output [2:0]ALUOp,
    output MemWrite,
    output ALUSrc,
    output RegWrite
    );
   parameter op_add   =  6'b000000,
             op_addi = 6'b001000,
             op_lw = 6'b100011,
             op_sw = 6'b101011,
             op_beq = 6'b000100,
             op_j = 6'b000010;

   assign RegDst = (Opcode == op_add )? 1:0;
   assign Jump = (Opcode == op_j)? 1:0;
   assign Branch = (Opcode == op_beq)? 1:0;
   assign MemRead = (Opcode == op_lw)? 1:0;
   assign MemtoReg = (Opcode == op_lw)? 1:0;

   assign ALUOp[0] = (Opcode == op_beq)? 1:0;
   assign ALUOp[1] = 0;
   assign ALUOp[2] = 0;

   assign MemWrite = (Opcode == op_sw)? 1:0;
```

```
   assign ALUSrc = (Opcode == op_addi || Opcode == op_lw || Opcode == op_sw)? 1:0;
   assign RegWrite = (Opcode == op_add || Opcode == op_addi || Opcode == op_lw)? 1:0;


endmodule
```

ALU 模块设计见实验一。

(3) PC 模块设计

```
module PC(
    input clk,
    input rst,
    input [31:0]pcin,
    output [31:0]ExtOut
    );
    reg [31:0]pc;

    assign ExtOut = pc;
    always@(posedge clk or posedge rst)
    begin
        if(rst) pc = 0;
        else pc = pcin;
    end
endmodule
```

(4) 5 位宽选择器和 32 位宽选择器设计

```
module mux2to1_d5 #(parameter N = 5) // 32 位宽只需 N = 32 即可
   (output [N-1:0]out,
   input [N-1:0 ]a,b, // 输入数据
   input sel // 选择位
    );
reg [N-1:0]od;
assign out = od;
always @(*)
begin
    if(sel == 0)
        od = a;
    else if(sel == 1)
        od = b;
    else
        od = od;
end
endmodule
```

(5) 指令存储器、寄存器堆、数据存储器

```
dist_mem_gen_0 instuctions (.a(pout[9:2]),.spo(instruct));

RegFile reg_file
(.ra0(instruct[25:21]),.ra1(instruct[20:16]),.ra2(m_rf_addr),.wa(WR),.wd(WRd
),.rd0(Rd1),.rd1(Rd2),.rd2(rf_data),.clk(clk),.we(RegWrite));

dist_mem_gen_1 DataMem
(.a(ALU_result[9:2]),.dpra(m_rf_addr),.d(Rd2),.we(MemWrite),.spo(mem_Rd),.dp
o(m_data),.clk(clk)); // 字节寻址

```

其中，指令存储器是 256x32 的 ROM，寄存器堆（32x32）的设计见实验二， 数据存储器是 256x32 的单端口 RAM ，后来为了和 DBU 搭建接口，改为 双端口的 RAM。

(6) sign_extend

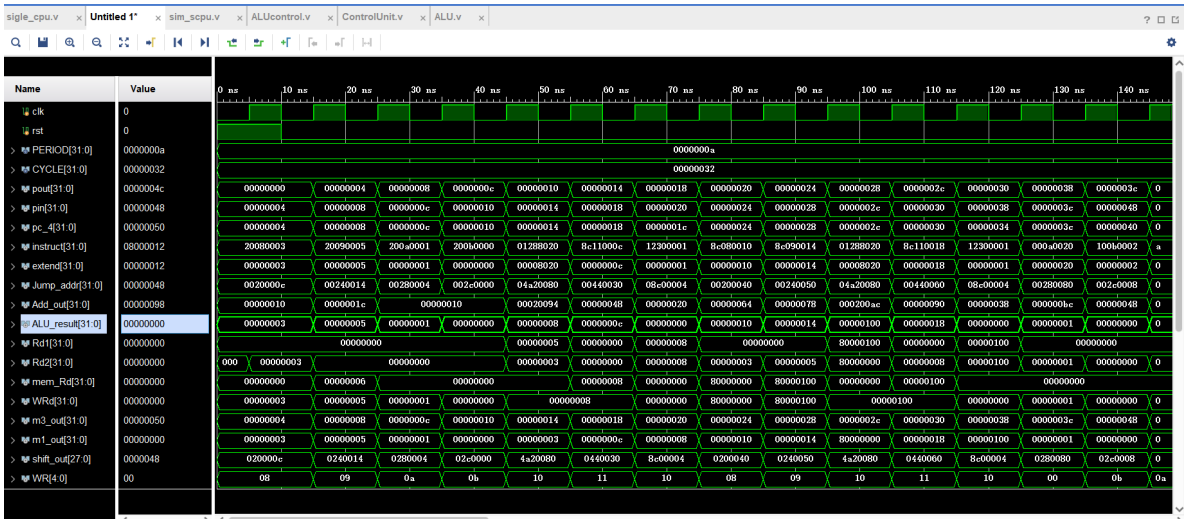
```
module sign_extend(
    input [15:0]imm,
    output [31:0]ExtOut
    );

    assign ExtOut[15:0] = imm;
    assign ExtOut[31:16] = 16'h0000;

endmodule
```

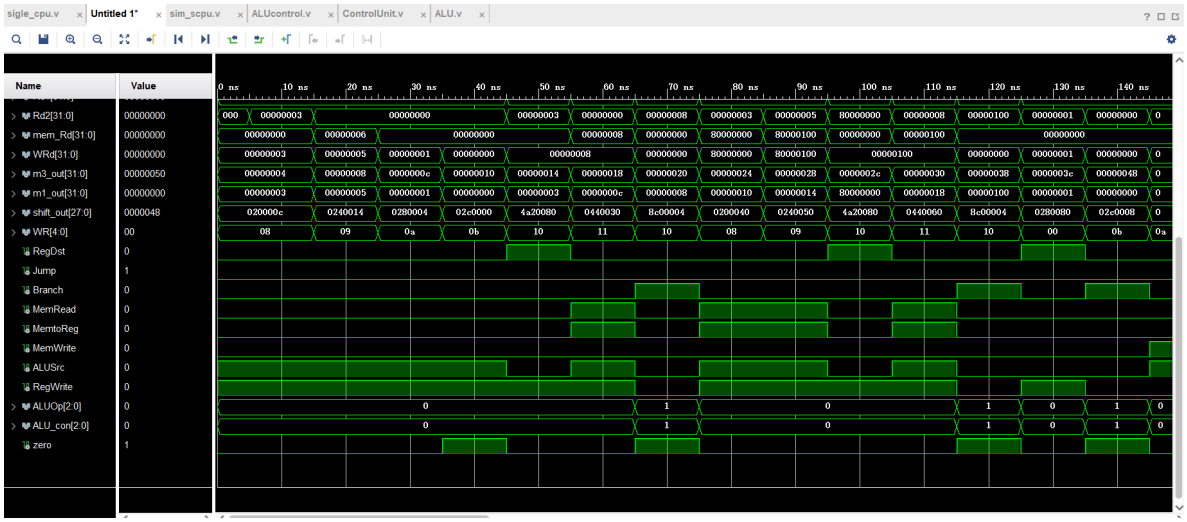
(7) 顶层模块设计

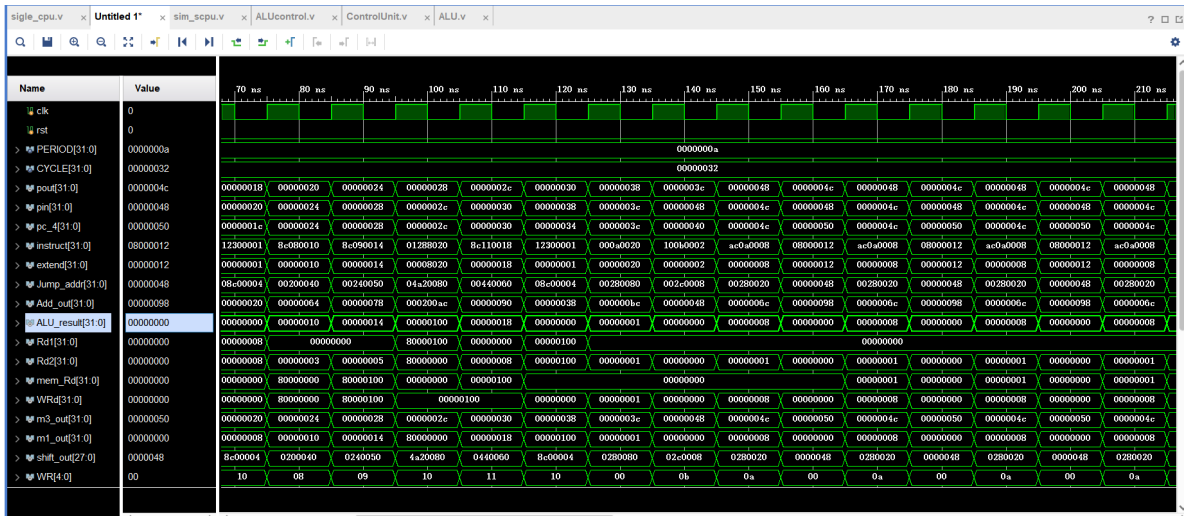
```
module sigle_cpu(
    input clk,
    input rst,
    input [7:0]m_rf_addr,
    output [31:0]m_data,
    output [31:0]rf_data,
    // dbu -- status
    output Jump,
    output Branch,
    output RegDst,
    output RegWrite,
    output MemRead,
    output MemtoReg,
    output MemWrite,
    output ALUSrc,
    output zero,
    output [2:0]ALUOp,
    output [31:0]pc_in,
    output [31:0]pc_out,
    output [31:0]instr,
    output [31:0]rf_rd1,
    output [31:0]rf_rd2,
    output [31:0]alu_y,
    output [31:0]m_rd
    );
    wire [31:0]
pin,pout,pc_4,instruct,extend,Jump_addr,Add_out,ALU_result,Rd1,Rd2,mem_Rd,W
Rd;
```

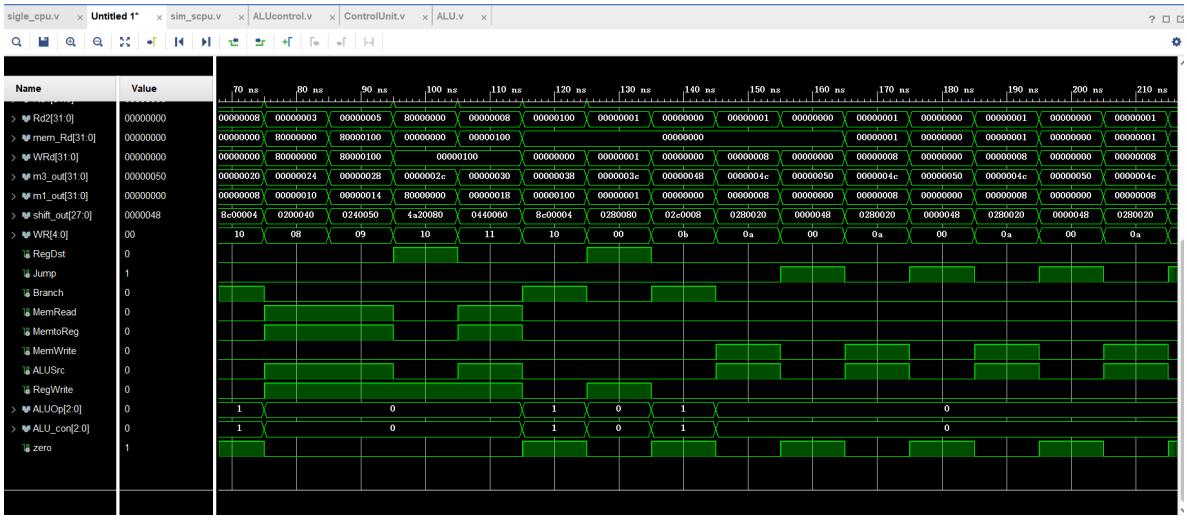
```
    wire [31:0] m3_out,m1_out;
    wire [27:0] shift_out;
    wire [4:0] WR;
    wire [2:0]ALU_con;

    assign Jump_addr = pc_4[31:28]|shift_out[27:0];

    assign pc_in = pin;
    assign pc_out = pout;
    assign instr = instruct;
    assign rf_rd1 = Rd1;
    assign rf_rd2 = Rd2;
    assign alu_y = ALU_result;
    assign m_rd = mem_Rd;

    PC pc_0 (.pcin(pin),.ExtOut(pout),.clk(clk),.rst(rst));

    mux2to1_d5 mux0
(.a(instruct[20:16]),.b(instruct[15:11]),.sel(RegDst),.out(WR));
    mux2to1_d32 mux1(.a(Rd2),.b(extend),.sel(ALUSrc),.out(m1_out));
    mux2to1_d32 mux3 (.a(pc_4),.b(Add_out),.sel(Branch&zero),.out(m3_out));
    mux2to1_d32 mux4 (.a(m3_out),.b(Jump_addr),.sel(Jump),.out(pin));
    mux2to1_d32 mux2 (.a(ALU_result),.b(mem_Rd),.sel(MemtoReg),.out(WRd));


    dist_mem_gen_0 instuctions (.a(pout[9:2]),.spo(instruct));
    RegFile reg_file
(.ra0(instruct[25:21]),.ra1(instruct[20:16]),.ra2(m_rf_addr),.wa(WR),.wd(WR
d),.rd0(Rd1),.rd1(Rd2),.rd2(rf_data),.clk(clk),.we(RegWrite));
    dist_mem_gen_1 DataMem
(.a(ALU_result[9:2]),.dpra(m_rf_addr),.d(Rd2),.we(MemWrite),.spo(mem_Rd),.d
po(m_data),.clk(clk)); // 字节寻址

    ControlUnit CU
(.Opcode(instruct[31:26]),.RegDst(RegDst),.Jump(Jump),.Branch(Branch),.MemR
ead(MemRead),.MemtoReg(MemtoReg),.ALUOp(ALUOp),.MemWrite(MemWrite),.ALUSrc(
ALUSrc),.RegWrite(RegWrite));

    shift_left shift(.addr(instruct[25:0]),.new_addr(shift_out));
    sign_extend new_sign (.imm(instruct[15:0]),.ExtOut(extend));

    ALU Add (.a(4),.b(pout),.m(3'b000),.y(pc_4));
    ALU Add_branch (.a(pc_4),.b(extend<<2),.m(3'b000),.y(Add_out));
    ALU ALU_0 (.a(Rd1),.b(m1_out),.m(ALU_con),.y(ALU_result),.zf(zero));

    ALUcontrol alu_control
(.funct(instruct[5:0]),.ALUOp(ALUOp),.opcode(ALU_con));

endmodule
```

(8) 仿真结果


sigle_cpu.v
Untitled 1*
sim_scpu.v
ALUcontrol.v
ControlUnit.v
ALU.v
Name
Value
clk
rst
PERIOD[31:0]
CYCLE[31:0]
pout[31:0]
pin[31:0]
pc_4[31:0]
instruct[31:0]
extend[31:0]
Jump_addr[31:0]
Add_out[31:0]
ALU_result[31:0]
Rd1[31:0]
Rd2[31:0]
mem_Rd[31:0]
WRd[31:0]
m3_out[31:0]
m1_out[31:0]
shift_out[27:0]
WR[4:0]


sigle_cpu.v
Untitled 1*
sim_scpu.v
ALUcontrol.v
ControlUnit.v
ALU.v
Name
Value
Rd2[31:0]
mem_Rd[31:0]
WRd[31:0]
m3_out[31:0]
m1_out[31:0]
shift_out[27:0]
WR[4:0]
RegDst
Jump
Branch
MemRead
MemtoReg
MemWrite
ALUSrc
RegWrite
ALUOp[2:0]
ALU_con[2:0]
zero


sigle_cpu.v
Untitled 1*
sim_scpu.v
ALUcontrol.v
ControlUnit.v
ALU.v
Name
Value
clk
rst
PERIOD[31:0]
CYCLE[31:0]
pout[31:0]
pin[31:0]
pc_4[31:0]
instruct[31:0]
extend[31:0]
Jump_addr[31:0]
Add_out[31:0]
ALU_result[31:0]
Rd1[31:0]
Rd2[31:0]
mem_Rd[31:0]
WRd[31:0]
m3_out[31:0]
m1_out[31:0]
shift_out[27:0]
WR[4:0]



通过波形图可以看出，取指、译码、执行、写回操作都能正常运行，最终跳转到 success 状态循环。

2. DBU 设计与仿真

为了方便下载和调试，需要设计一个调试单元 DBU，该单元用于控制 CPU 的运行方式，显示运行过程的中间状态和最终运行结果。 DBU 的端口与 CPU 以及 FPGA 开发版外设的连接图如下。

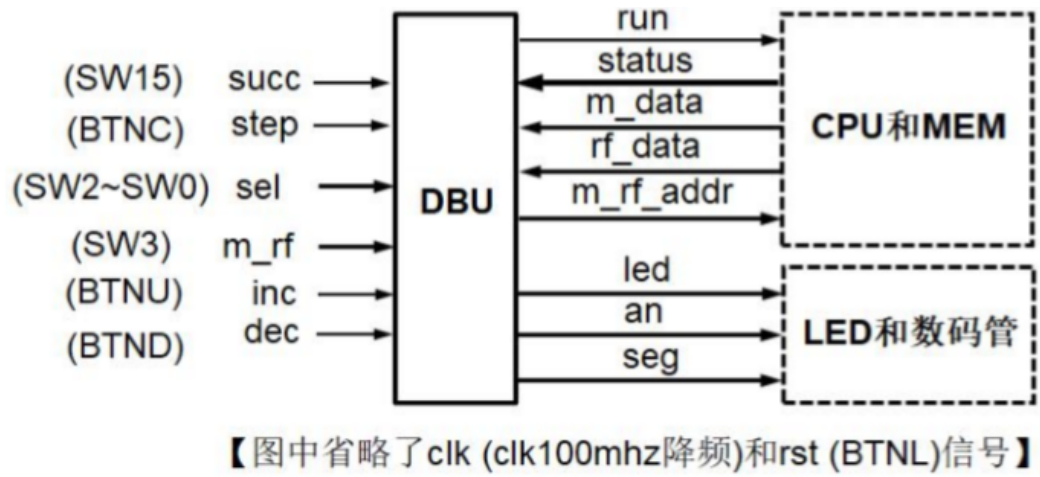


图-3 调试单元端口及其连接图

- 控制 CPU 运行方式

  – succ=1：控制 CPU 连续执行指令，run=1（一直维持）

  – succ=0：控制 CPU 执行一条指令，每按动 step 一次，run 输出维持一个时钟周期 的脉冲

- sel=0：查看 CPU 运行结果 (存储器或者寄存器堆内容)

```
- m_rf：  1，查看存储器(MEM)；0，查看寄存器堆(RF)

- m_rf_addr：  MEM/RF 的调试读口地址(字地址)，复位时为零

- inc/dec：m_rf_addr 加 1 或减 1

- rf_data/m_data：从 RF/MEM 读取的数据字

- 16 个 LED 指示灯显示 m_rf_addr

- 8 个数码管显示 rf_data/m_data
```

- sel=1~7：查看 CPU 运行状态（status）
  – 12个LED指示灯(SW11~SW0) 依次显示控制器的控制信号

  (Jump,Branch,Reg_Dst, RegWrite, MemRead, MemtoReg, MemWrite, ALUOp, ALUSrc)和ALUZero。

  其中 ALUOp 为 3 位 – 8 个数码管显示由 sel 选择的一个 32 位数据
- sel=1：pc_in,PC 的输入数据
- sel=2：pc_out,PC 的输出数据
- sel=3：instr, 指令存储器的输出数据
- sel=4：rf_rd1, 寄存器堆读口 1 的输出数据
- sel=5：rf_rd2, 寄存器堆读口 2 的输出数据
- sel=6：alu_y,ALU 的运算结果
- sel=7：m_rd, 数据存储器的输出数据

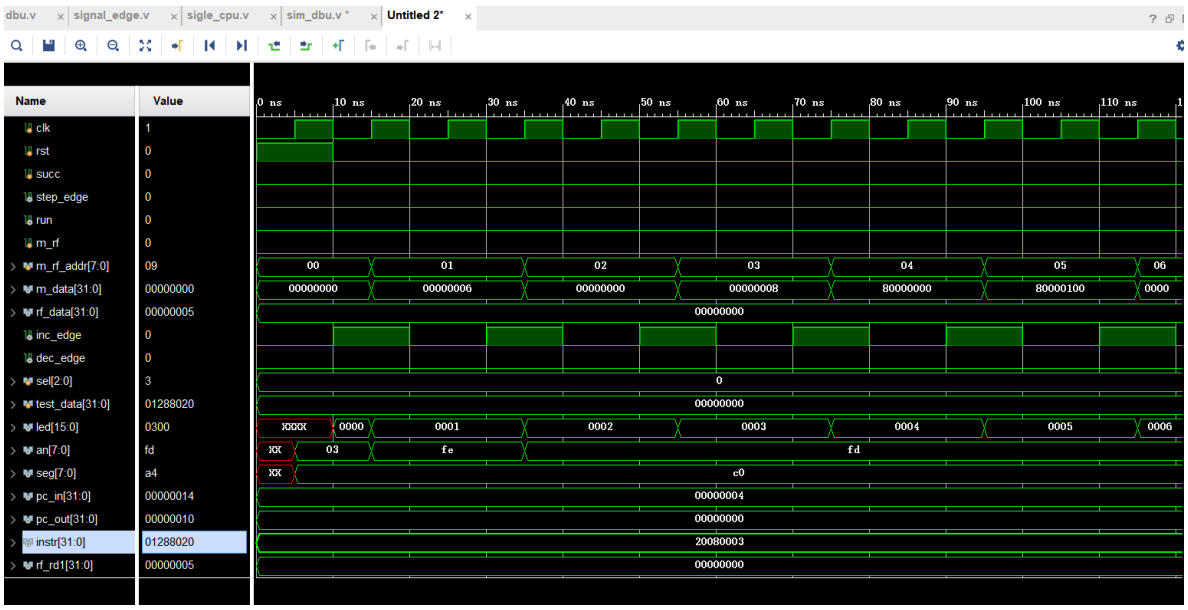
```
module dbu(
   input clk,
   input rst,
   input succ, // SW 控制 CPU 执行指令的条数
   input step, // button 单步执行指令
   input [2:0]sel, // SW2~SW0 查看 CPU 运行的结果
   input m_rf, // SW3 1，查看 MEM，0，查看 RF
   input inc, // button m_rf_addr +1
   input dec,  // button m_rf_addr -1
   output [7:0]SEG, // 数码管显示十六进制数字
   output reg [7:0]AN, // 片选信号
   output reg [15:0]led // led 灯显示
    );
    wire [31:0]m_data,rf_data;
    wire
Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUSrc,zero;
    wire [2:0]ALUOp;
    wire [31:0]pc_in,pc_out,instr,rf_rd1,rf_rd2,alu_y,m_rd ;
    wire run;
    reg [7:0]m_rf_addr;
    reg [31:0]test_data;

    wire step_edge,inc_edge,dec_edge;

    //数码管扫描信号
    reg [2:0] cntsel;
    reg [19:0]cnt;
    wire pulse_cnt;
    wire [7:0]seg;
    reg [3:0] data;

    assign SEG = seg;

    sigle_cpu
cpu(.clk(clk&run),.rst(rst),.m_rf_addr(m_rf_addr),.m_data(m_data),.rf_data
(rf_data),.Jump(Jump),.Branch(Branch),.RegDst(RegDst),.MemRead(MemRead),.M
emtoReg(MemtoReg),.ALUOp(ALUOp),.MemWrite(MemWrite),.ALUSrc(ALUSrc),.RegWr
ite(RegWrite),.zero(zero),.pc_in(pc_in),.pc_out(pc_out),.instr(instr),.rf_
rd1(rf_rd1),.rf_rd2(rf_rd2),.alu_y(alu_y),.m_rd(m_rd));

```

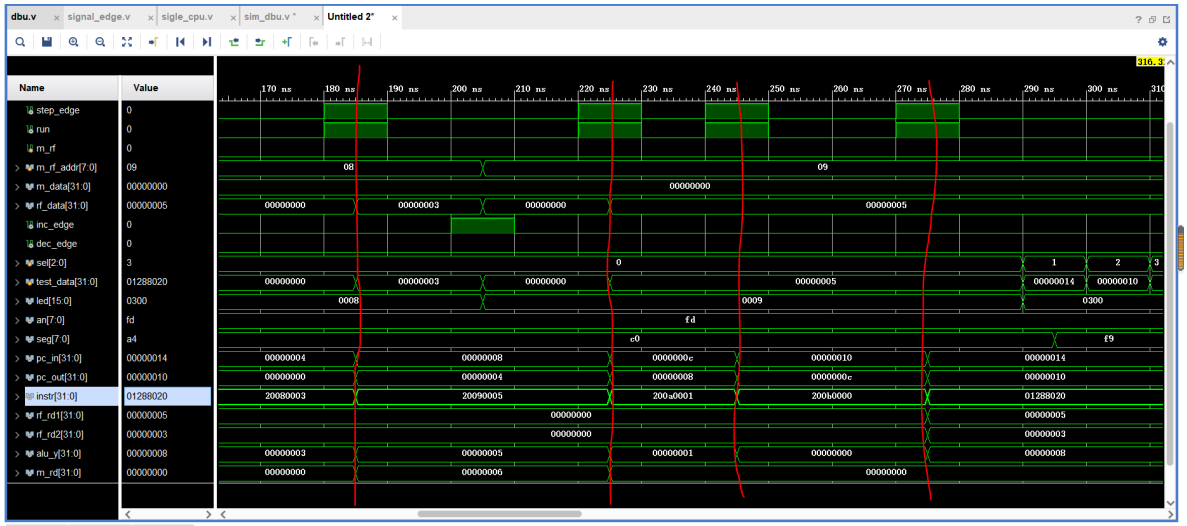
```
    // 按键信号取边沿
    signal_edge step_out(.clk(clk),.button(step),.button_redge(step_edge));
    signal_edge inc_out(.clk(clk),.button(inc),.button_redge(inc_edge));
    signal_edge dec_out(.clk(clk),.button(dec),.button_redge(dec_edge));

    //----------控制 CPU 运行方式----------//
    assign run = step_edge|succ;

    //---------- sel 查看 CPU 运行结果----------//

    reg [7:0]new_m_rf_addr;
    always@(posedge clk or posedge rst)
    begin
        if(rst) m_rf_addr <= 0;
        else m_rf_addr <= new_m_rf_addr;
    end

    // 组合逻辑实现 sel 控制信号的对应功能
    always@(*)
    begin
        if(rst)
        begin
            test_data = 0;
        end
        else
        begin
            case(sel)
                3'b000:
                begin
                    led[7:0] = m_rf_addr; // led 显示
                    led[15:8] = 0;
                    if(m_rf) begin test_data = m_data; end
                    else if (~m_rf) begin test_data = rf_data;end
                    else test_data = 0;

                    if(inc_edge && ~dec_edge) new_m_rf_addr =  m_rf_addr + 1;
                    else if (~inc_edge && dec_edge) new_m_rf_addr = m_rf_addr - 1;
                    else new_m_rf_addr = m_rf_addr;
                end
                3'b001:
                begin
                    led[12:0] = {Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zero};
                    led[15:13] = 0;
                    test_data = pc_in; // sel=1：pc_in,PC 的输入数据
                end
                3'b010:
                begin
                    led[12:0] = {Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zero};
                    led[15:13] = 0;
                    test_data = pc_out; // sel=2：pc_out,PC 的输出数据
                end
                3'b011:
                begin
```

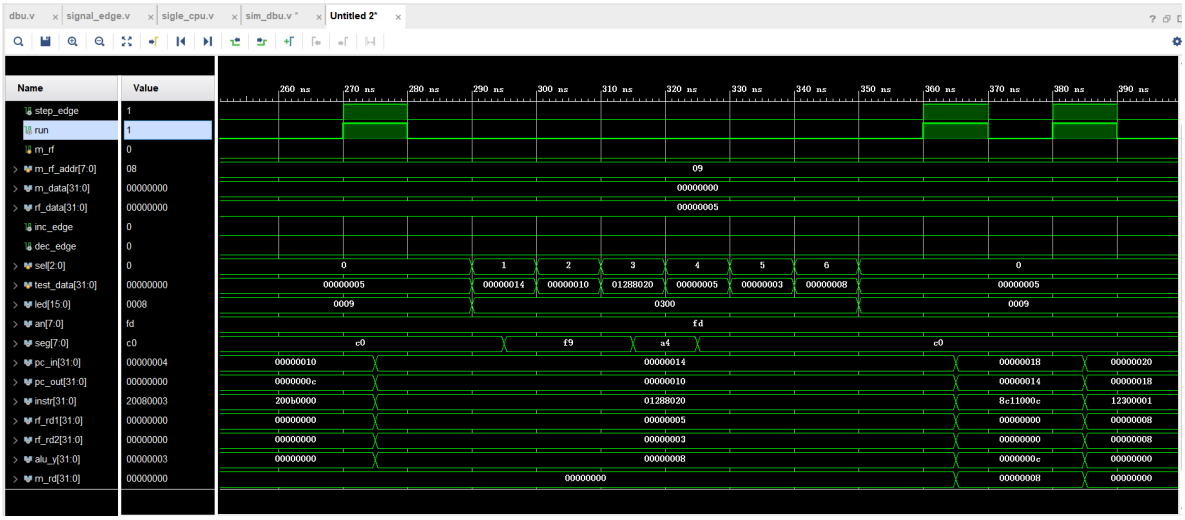
```
                    led[12:0] = {Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zero};
                    led[15:13] = 0;
                    test_data = instr; // sel=3：instr，指令存储器的输出数据
                end
                3'b100:
                begin
                    led[12:0] = {Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zero};
                    led[15:13] = 0;
                    test_data = rf_rd1; // sel=4：rf_rd1，寄存器堆读口 1 的输出数据
                end
                3'b101:
                begin
                    led[12:0] = {Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zero};
                    led[15:13] = 0;
                    test_data = rf_rd2; // sel=5：rf_rd2，寄存器堆读口 2 的输出数据
                end
                3'b110:
                begin
                    led[12:0] = {Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zero};
                    led[15:13] = 0;
                    test_data = alu_y; // sel=6：alu_y,ALU 的运算结果
                end
                3'b111:
                begin
                    led[12:0] = {Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zero};
                    led[15:13] = 0;
                    test_data = m_rd; // sel=7：m_rd，数据存储器的输出数据
                end
                default:
                begin
                    led[12:0] = {Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zero};
                    led[15:13] = 0;
                    test_data = 0;
                end
            endcase
        end
    end

    //----------数码管扫描部分：显示十六进制数据----------//
    always @(posedge clk)
    begin
        if (rst)
            cnt <= 20'b0;
        else if(cnt == 20'h186a0)//设定扫描频率 10^5
            cnt <= 20'b0;
        else
            cnt <= cnt + 20'b1;
    end
    assign pulse_cnt = (cnt == 20'b1)? 1'b1:1'b0;
    always @(posedge clk)
```

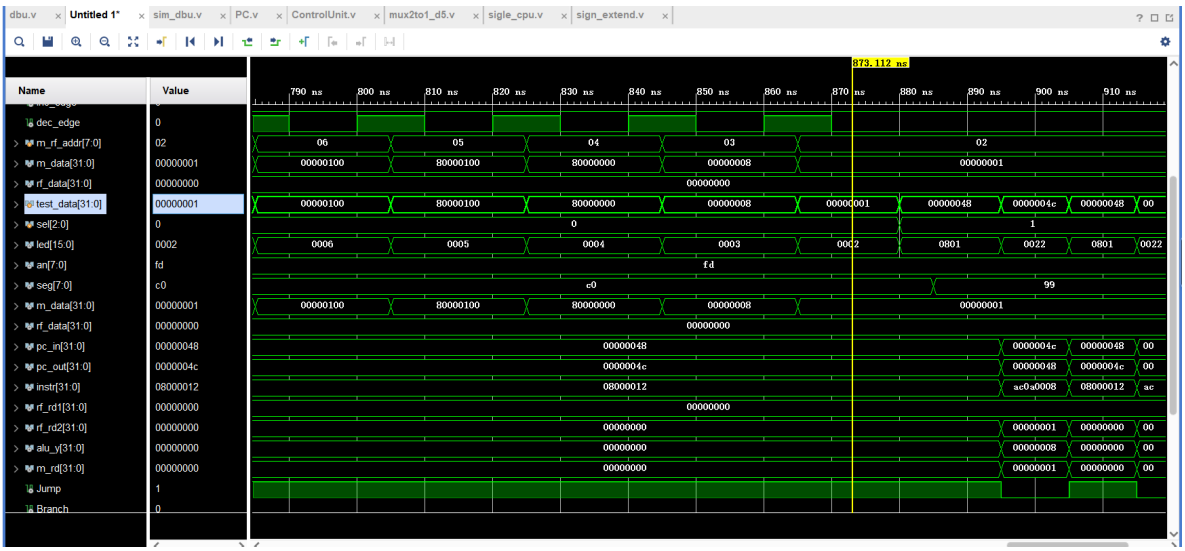
```
    begin
        if (rst)
            cntsel <= 0;
        if(pulse_cnt)
            cntsel <= cntsel + 3'b01;
    end

    dist_mem_gen_2 dist_mem_2(
    .a(data),
    .spo(seg)
    );

    always @(posedge clk)
    begin
            case(cntsel)
                3'h0:AN     <= 8'b1111_1110;
                3'h1:AN     <= 8'b1111_1101;
                3'h2:AN     <= 8'b1111_1011;
                3'h3:AN     <= 8'b1111_0111;
                3'h4:AN     <= 8'b1110_1111;
                3'h5:AN     <= 8'b1101_1111;
                3'h6:AN     <= 8'b1011_1111;
                3'h7:AN     <= 8'b0111_1111;
                default: AN <= 8'b11;
            endcase
    end

    always @(posedge clk)
    begin
            case(cntsel)
                3'h0:data   <= test_data[3:0];
                3'h1:data   <= test_data[7:4];
                3'h2:data   <= test_data[11:8];
                3'h3:data   <= test_data[15:12];
                3'h4:data   <= test_data[19:16];
                3'h5:data   <= test_data[23:20];
                3'h6:data   <= test_data[27:24];
                3'h7:data   <= test_data[31:28];
                default: data <= 32'b0;
            endcase
    end

endmodule
```

DBU 仿真结果


dbu.v
signal_edge.v
sigle_cpu.v
sim_dbu.v *
Untitled 2*
Name
Value
clk 1
rst 0
succ 0
step_edge 0
run 0
m_rf 0
m_rf_addr[7:0] 09
m_data[31:0] 00000000
rf_data[31:0] 00000005
inc_edge 0
dec_edge 0
sel[2:0] 3
test_data[31:0] 01288020
led[15:0] 0300
an[7:0] fd
seg[7:0] a4
pc_in[31:0] 00000014
pc_out[31:0] 00000010
instr[31:0] 01288020
rf_rd1[31:0] 00000005
0 ns 10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90 ns 100 ns 110 ns
00 01 02 03 04 05 06
00000000 00000006 00000000 00000008 80000000 80000100 0000
00000000
0
00000000
XXXX 0000 0001 0002 0003 0004 0005 0006
XX 03 fe fd
XX c0
00000004
00000000
20080003
00000000


dbu.v
signal_edge.v
sigle_cpu.v
sim_dbu.v *
Untitled 2*
Name
Value
step_edge 0
run 0
m_rf 0
m_rf_addr[7:0] 09
m_data[31:0] 00000000
rf_data[31:0] 00000005
inc_edge 0
dec_edge 0
sel[2:0] 3
test_data[31:0] 01288020
led[15:0] 0300
an[7:0] fd
seg[7:0] a4
pc_in[31:0] 00000014
pc_out[31:0] 00000010
instr[31:0] 01288020
rf_rd1[31:0] 00000005
rf_rd2[31:0] 00000003
alu_y[31:0] 00000008
m_rd[31:0] 00000000
170 ns 180 ns 190 ns 200 ns 210 ns 220 ns 230 ns 240 ns 250 ns 260 ns 270 ns 280 ns 290 ns 300 ns
316.3
08 09
00000000
00000000 00000003 00000000 00000005
0 1 2 3
00000000 00000003 00000000 00000005 00000014 00000010
0008 0009 0300
fd
c0 f9
00000004 00000008 0000000c 00000010 00000014
00000000 00000004 00000008 0000000c 00000010
20080003 20090005 200a0001 200b0000 01288020
00000000 00000005
00000000 00000003
00000003 00000005 00000001 00000000 00000008
00000000 00000006 00000000


dbu.v
signal_edge.v
sigle_cpu.v
sim_dbu.v *
Untitled 2*
Name
Value
step_edge 1
run 1
m_rf 0
m_rf_addr[7:0] 08
m_data[31:0] 00000000
rf_data[31:0] 00000000
inc_edge 0
dec_edge 0
sel[2:0] 0
test_data[31:0] 00000000
led[15:0] 0008
an[7:0] fd
seg[7:0] c0
pc_in[31:0] 00000004
pc_out[31:0] 00000000
instr[31:0] 20080003
rf_rd1[31:0] 00000000
rf_rd2[31:0] 00000000
alu_y[31:0] 00000003
m_rd[31:0] 00000000
260 ns 270 ns 280 ns 290 ns 300 ns 310 ns 320 ns 330 ns 340 ns 350 ns 360 ns 370 ns 380 ns 390 ns
09
00000000
00000005
0 1 2 3 4 5 6 0
00000005 00000014 00000010 01288020 00000005 00000003 00000008 00000005
0009 0300 0009
fd
c0 f9 a4 c0
00000010 00000014 00000018 00000020
0000000c 00000010 00000014 00000018
200b0000 01288020 Bc11000c 12300001
00000000 00000005 00000000 00000008
00000000 00000003 00000000 00000008
00000000 00000008 0000000c 00000000
00000000 00000008 00000000



通过仿真结果可以看出，DBU 的接口都运行正常，在成功执行 `sw $t2,8($0) #全部测试通过，存储器地址0x08里值为1` 之后， test_data 也能正确显示数值 1。

三、实验总结

本次实验主要有以下收获：

1. 熟练掌握单周期 CPU 的数据通路和各个功能块的设计与实现，能够通过控制单元来实现一些简单的指令。
2. 能够熟练使用仿真波形调试代码。

> 本次实验如果能使用 FPGA 开发板应该会有趣很多，线上的 FPGA 毕竟功能还是有限。
>
> 希望能早点返校 QAQ