多周期CPU

一、实验要求与内容

1. 设计实现多周期MIPS-CPU，可执行如下指令：
    - **add, sub, and, or, xor, nor, slt**
    - **addi**,**andi**,**ori**,**xori**,**slti**
    - **lw**,**sw**
    - **beq**,**bne**,**j**

    其中寄存器堆中R0内容恒定为0，存储器容量为256*32位
2. **DDU**：**Debug and Display Unit**，调试和显示单元
    - 下载测试时，用于控制CPU运行方式和显示运行结果
    - 数据通路中寄存器堆和存储器均需要增加1个读端口，供DDU读取并显示其中内容
3. 数据通路

    1588672209455
4. DDU显示模块

![1588672295832]((/多周期CPU/2.png)

二、实验数据通路

![1588672387778]((/多周期CPU/3.png)

注：

- I1代表I型指令，I3代表beq和bne型指令
- 对每个阶段的任务分析见下文对代码段的分析

三、代码逻辑分析

1.定义寄存器堆

```
{%codeblock lang:verilog%}
module regfile(
input [4:0] ra0,
input [4:0] ra1,
input [4:0] wa,
input [31:0] wd,
input we,
input rst,
input clk,
output reg [31:0] rd0,
output reg [31:0] rd1
    );
```

```
reg     [22:0]  cnt;
reg     [31:0] register [0:31];//32个寄存器

always @(posedge clk or posedge rst)
begin
     if(rst)
     begin
         register[0:31]=0; //寄存器初始化
     end
     else
     begin
     if(we) //写寄存器
          register[wa]<=wd;
     else
          register[wa]<=register[wa];
     end
end

always @(*)
begin
    if(rst)//读寄存器
     begin
        rd0=0;
        rd1=0;
      end
      else
      begin
         rd0=register[ra0];
         rd1=register[ra1];
      end
end

endmodule
{%endcodeblock%}
```

2.定义控制信号

```
{%codeblock lang:verilog%}
module control_code(
input [5:0] opcode,
input [5:0] funct,
input clk,
output reg regdst,
output reg memtoreg,
output reg regwrite,
output reg memread,
output reg memwrite,
output reg [1:0] ALUop,
output reg pcwritecond,
output reg [2:0] pcsource,
output reg j,//便于在取指结束后就判断是否为j型指令
output reg addi,//便于之后进行立即数扩展时判断是算术扩展还是逻辑扩展
output reg [2:0] alusrcB
    );
```

```
always @(opcode,funct)
begin
   case(opcode)
   6'b000000://R型
   begin
     if(funct==6'd0)
        regwrite=0;//对于32'h0的空指令，写寄存器信号为0
     else
        regwrite=1;
     regdst=1;
     memtoreg=0;
     memread=0;
     memwrite=0;
     ALUop=10;
     pcwritecond=0;
     pcsource=3'b000;
     j=0;
     addi=0;
     alusrcB=3'b000;//alusrcB的值按照上图的数据通路来定义
    end
```

```
6'b100011://lw
begin
    regdst=0;
    memtoreg=1;
    regwrite=1;
    memread=1;
    memwrite=0;
    ALUop=00;
    pcwritecond=0;
    pcsource=3'b000;
    j=0;
    addi=0;
    alusrcB=3'b010;
 end

 6'b101011://sw
 begin
     regdst=0;
     memtoreg=0;
     regwrite=0;
     memread=0;
     memwrite=1;
     ALUop=00;
     pcwritecond=0;
     pcsource=3'b000;
     j=0;
     addi=0;
     alusrcB=3'b010;
  end

  6'b000100,6'b000101://beq,bne
  begin
      regdst=0;
```

```
        memtoreg=0;
        regwrite=0;
        memread=0;
        memwrite=0;
        ALUop=01;
        pcwritecond=1;
        pcsource=3'b001;
        j=0;
        addi=0;
        alusrcB=3'b000;
     end

    6'b000010://j
    begin
        regdst=0;
        memtoreg=0;
        regwrite=0;
        memread=0;
        memwrite=0;
        ALUop=01;
        pcwritecond=0;
        pcsource=3'b010;
        j=1;
        addi=0;
        alusrcB=3'b011;
    end

    default://I型
    begin
        regdst=0;
        memtoreg=0;
        regwrite=1;
        memread=0;
        memwrite=0;
        ALUop=11;
        pcwritecond=0;
        pcsource=3'b000;
        j=0;
        if(opcode==6'b001000)
        addi=1;
        else
        addi=0;
        alusrcB=3'b010;
     end
endcase
```

end

endmodule
{%endcodeblock%}

3.ALU运算器

{%codeblock lang:verilog%}
module CTR(
input [1:0] ALUop,
input [5:0] funct,

```
input [5:0] opcode,
output reg [3:0] ALUctr);
always @(ALUop or funct)
casex({ALUop,funct})
    8'b10100000:ALUctr=4'b0000;//add
    8'b10100010:ALUctr=4'b0001;//sub
    8'b10100100:ALUctr=4'b0010;//and
    8'b10100101:ALUctr=4'b0011;//or
    8'b10100110:ALUctr=4'b0100;//xor
    8'b10100111:ALUctr=4'b0101;//nor
    8'b10101010:ALUctr=4'b0110;//slt,slti
    8'b01xxxxxx:ALUctr=4'b0001;//beq,bne(相减)
    8'b00xxxxxx:ALUctr=4'b0000;//lw,sw(相加)
    8'b11xxxxxx://I型指令
    begin
    casex({ALUop,opcode})
    8'b11001000:ALUctr=4'b0000;//addi
    8'b11001100:ALUctr=4'b0010;//andi
    8'b11001101:ALUctr=4'b0011;//ori
    8'b11001110:ALUctr=4'b0100;//xori
    8'b11001010:ALUctr=4'b0110;//slti
    endcase
    end
endcase
endmodule

module alu(
input [31:0] input1,
input [31:0] input2,
input [1:0] ALUop,
input [5:0] funct,
input [5:0] opcode,
input [3:0] aluctr,
input clk,
input rst,
output reg[31:0] alures,
output reg zero);
CTR control(.ALUop(ALUop),.funct(funct),.opcode(opcode),.ALUctr(aluctr));
always @(*)
begin
   if(rst)
   alures=32'h0000;
   else
   begin
   case(aluctr)
      4'b0000://add
         alures=input1+input2;
      4'b0001://sub
      begin
         alures=input1-input2;
         if(alures==0)
            zero=1;
            else
                zero=0;
        end
        4'b0010://and
            alures=input1 & input2;
        4'b0011://or
            alures=input1 | input2;
        4'b0100://xor
            alures=input1 ^ input2;
        4'b0101://nor
            alures=~(input1 | input2);
        4'b0110://slt
        begin
            if(input1 < input2)
                alures=32'd1;
            else
                alures=32'd0;
        end
      endcase
    end
  end
endmodule
{%endcodeblock%}
```

4. 7段数码管控制逻辑

```
{%codeblock lang:verilog%}
module segc(
input   clk,
input   rst,
input   [31:0] in,
output  [7:0]   an,
output  [6:0]   seg
    );
wire    rst_n;
wire    cout;
reg     pulse_1hz;
reg     [7:0]   seg_sel;
reg     [3:0]   seg_din;
reg     [22:0]  cnt;

seg_ctrl        seg_ctrl(
.x              (seg_din),
.sel            (seg_sel),
.an             (an),
.seg            (seg)
    );
 always@(posedge clk or posedge rst)
 begin
    if(rst)
        cnt <= 23'h0;
    else if(cnt<23'd5000000)
        cnt <= cnt + 1;
    else
        cnt <= 23'h0;
 end
always@(posedge clk )
begin
     case(cnt[15:13])
        3'b000:
        begin
            seg_sel <= 8'b1111_1110;
            seg_din <= in[3:0];
        end
        3'b001:
        begin
            seg_sel <= 8'b1111_1101;
            seg_din <= in[7:4];
        end
        3'b010:
        begin
            seg_sel <= 8'b1111_1011;
            seg_din <= in[11:8];
        end
        3'b011:
        begin
            seg_sel <= 8'b1111_0111;
            seg_din <= in[15:12];
        end
        3'b100:
        begin
            seg_sel <= 8'b1110_1111;
            seg_din <= in[19:16];
        end
        3'b101:
        begin
            seg_sel <= 8'b1101_1111;
            seg_din <= in[23:20];
        end
        3'b110:
        begin
            seg_sel <= 8'b1011_1111;
            seg_din <= in[27:24];
        end
        3'b111:
        begin
            seg_sel <= 8'b0111_1111;
            seg_din <= in[31:28];
        end
        endcase
end
endmodule
```

```
module seg_ctrl(
input [3:0] x,
input[7:0] sel,
output [7:0] an,
output reg [6:0] seg
);
assign an=sel;
always @(x)
begin
case(x)//从0到F7段数码管上每个引脚电平的控制
    4'b0000:seg=7'b1000000;
    4'b0001:seg=7'b1111001;
    4'b0010:seg=7'b0100100;
    4'b0011:seg=7'b0110000;
    4'b0100:seg=7'b0011001;
    4'b0101:seg=7'b0010010;
    4'b0110:seg=7'b0000010;
    4'b0111:seg=7'b1111000;
    4'b1000:seg=7'b0000000;
    4'b1001:seg=7'b0010000;
    4'b1010:seg=7'b0100000;
    4'b1011:seg=7'b0000011;
    4'b1100:seg=7'b1000110;
    4'b1101:seg=7'b0100001;
    4'b1110:seg=7'b0000110;
    4'b1111:seg=7'b0001110;
endcase
end
endmodule
{%endcodeblock%}
```

5.顶层模块

1.例化存储器

```
{%codeblock lang:verilog%}
dist_mem_gen_0 memory(
.a(writeaddr_mem), // input wire [15 : 0] a 10
.d(B), // input wire [11 : 0] d 11
.dpra(readaddr_mem), // input wire [15 : 0] dpra 12
.clk(cpuclk), // input wire clk 13
.we(memwrite1), // input wire we 14
.dpo(readdata),
.spo(readdata1) ); // output wire [11 : 0] dpo
{%endcodeblock%}
```

分析：使用Dual Port RAM,其中a为写地址，d为读出的数据，dpra为读地址，we为写使能。

2.定义状态和操作数

{%codeblock lang:verilog%}
//定义状态
assign IF=4'd1,
ID = 4'd2,
EX_R = 4'd3,
EX_I1 = 4'd4,
EX_LW = 4'd5,
EX_I3 = 4'd6,
EX_J = 4'd7,
MEM_R = 4'd8,
MEM_I1 = 4'd9,
MEM_LW = 4'd10,
MEM_SW = 4'd11,
MEM_I3 = 4'd12,
WB_LW = 4'd13,
EX_SW = 4'd14;
//定义操作数
assign opcode=IR[31:26],
funct = IR[5:0],
rs = IR[25:21],
rt = IR[20:16],
rd = IR[15:11],
shamt = IR[10:6],
imm = IR[15:0],
jump = IR[25:0];

```
#### 3.定义选择器信号

{%codeblock lang:verilog%}
//多路选择器
assign IorD_mux = IorD?ALUout[9:2]:PC[9:2];
assign writereg_mux = regdst?rd:rt;//写寄存器值
assign writedata_mux = memtoreg?MDR:ALUout;//写存储器值
assign ALUsrcA_mux = A;
assign data=mem?readdata:A;
//ALUsrcB
always @(posedge cpuclk)
begin
     if(ALUsrcB==3'b000)
        ALUsrcB_mux=B;
     else if(ALUsrcB==3'b001)
        ALUsrcB_mux=32'd4;
     else if(ALUsrcB==3'b010)
        ALUsrcB_mux=sign_extend;
     else if(ALUsrcB==3'b011)
        ALUsrcB_mux=shift_left2;
end
//PCwritecond&zero
always @(current_state,PC_change,PCwritecond,zero,opcode)
begin
    if(current_state==EX_I3)
    begin
    if(opcode==6'b000100)
       PC_change=PCwritecond & zero ;
    else if(opcode==6'b000101)
       PC_change=PCwritecond & ~zero;
    else
       PC_change=0;
    end
    else
       PC_change=0;
{%endcodeblock%}

分析：

* 在MEM_LW阶段时，IorD为1，此时访问存储器的地址为ALU计算出的地址。
* 当为beq和bne型指令时，PCwritecond为1
* 当PC_change为1时，PC更新为当前PC+im<<2的值，其余指令PC_change均为0

#### 4.扩展位操作

{%codeblock lang:verilog%}
assign sign_extend=(imm[15]&(addi|PCwritecond))?{16'hffff,imm}:{16'h0000,imm};
{%endcodeblock%}

分析：当指令为addi、beq和bne类型且最高位为1时进行算术扩展，其余进行逻辑扩展

#### 5.由状态机控制每个周期运作

{%codeblock lang:verilog%}
always @(posedge cpuclk or posedge rst)
begin
     if(rst)
     begin
        current_state=4'd1;
         PC<=32'h0000_0000;
         IorD<=0;
    end
     else
     begin
     case(current_state)
     IF://PC+4,IR更新，当前状态切换到ID
     begin
        IorD<=0;
        PC<=PC+4;
        IR<=readdata;
        memwrite1<=0;
        current_state<=ID;
     end
     ID://如果为空指令，则继续取下一条指令
     begin
       if(IR==32'h0000)
            current_state<=IF;//状态机跳转到下一状态
        else//否则根据操作码不同，切换到不同状态
        begin
         if(opcode==6'd0)
             current_state<=EX_R;
         else if(opcode==6'b100011)
             current_state<=EX_LW;
         else if(opcode==6'b101011)
             current_state<=EX_SW;
         else if(opcode==6'b000100 || opcode==6'b000101)
```

```
            current_state<=EX_I3;
        else if(opcode==6'b000010)
            current_state<=EX_J;
        else
            current_state<=EX_I1;
        end
    end
    EX_R:
    begin
        current_state<=MEM_R;
     end
     MEM_R:
         current_state<=IF;
     EX_I1:
     begin
         current_state<=MEM_I1;
       end
       MEM_I1:
           current_state<=IF;
       EX_LW:
       begin
            current_state<=MEM_LW;
         end
         EX_SW:
         begin
             current_state<=MEM_SW;
         end
         MEM_LW:
         begin
             IorD<=1;//该信号置1，从存储器中取相应值
             current_state<=WB_LW;
          end
          WB_LW:
          begin
              IorD<=0;//该信号置0
              current_state<=IF;
          end
          MEM_SW:
          begin
               memwrite1<=1;//写存储器信号置1
               current_state<=IF;
          end
          EX_I3://I3代表beq和bne类型指令
          begin
              if(PC_change)
              begin
                  shift_left2=(PC+(sign_extend<<2));//PC更新
                   PC<=shift_left2;
                   IorD<=0;
                   current_state<=MEM_I3;
          end
          MEM_I3:
          begin
               current_state<=IF;
          end
          EX_J:
          begin
              PC<=jump_shift;//PC更新
```

```
                IorD<=0;
                current_state<=IF;
            end
          endcase
       end
   end
{%endcodeblock%}

#### 6.定义DDU模块

{%codeblock lang:verilog%}
assign cpuclk=cont?clk_50:step;//控制单步和连续执行
assign readaddr_reg =(choose & ~mem)?DDU_addr[4:0]:rs;
assign readaddr_mem=(choose & mem)?DDU_addr:IorD_mux;
assign data=mem?readdata:A;
always @(posedge clk_50 or posedge rst)
begin
     if(rst)
     begin
        inc_ctr<=1;
        dec_ctr<=1;
        DDU_addr<=8'd0;
     end
     else
     begin
         if(inc & inc_ctr)
         begin
            DDU_addr<=DDU_addr+1;
            inc_ctr<=0;
         end
         if(~inc & ~inc_ctr)
             inc_ctr<=1;
         if(dec & dec_ctr)
         begin
              DDU_addr<=DDU_addr-1;
              dec_ctr<=0;
         end
        if(~dec & ~dec_ctr)
              dec_ctr<=1;
     end
end
{%endcodeblock%}

分析：

* choose用于控制是否将读入地址作为访存或访问寄存器的地址
* DDU_addr为读入地址
* data为7段数码管上要显示的值，当mem为1时显示存储器中值，mem为0时显示相应寄存器堆中值
* 其中inc_ctr和dec_ctr为reg类型值，用来控制使在每个周期时钟上升沿当inc为1时，DDU_addr只递
增/递减一次。


## 四、仿真源码及顶层模块完整源码

* 顶层模块完整源码

  {%codeblock lang:verilog%}
  module CPU(
    input cont,
    input step,
    input mem,
    input inc,
    input dec,
    input clk,
    input choose,
    input rst,
    output [7:0] an,
    output [6:0] seg,
    output [7:0] MEMORY
        );

    wire [31:0] data;
    reg [7:0] DDU_addr;
    reg [7:0] DDU;
    reg run;
    reg step_ctr;
    reg inc_ctr;
    reg dec_ctr;
    reg [3:0] current_state;
    reg [3:0] next_state;
    reg [31:0] PC,IR;
    reg PCwrite;
    reg IorD;
    reg IRwrite;
    reg ALUsrcA;
    reg [31:0] writedata_mem;
    reg PC_change;
    reg [31:0] PCsource_mux;
    reg memwrite1;
    reg [31:0] ALUsrcB_mux;
    reg [31:0] shift_left2;
    wire [5:0] opcode_DDU;
    wire cpuclk;
    wire [2:0] ALUsrcB;
    wire [31:0] ALUout_shift;
    wire [2:0] PCsource;
    wire PCwritecond;
    wire addi;
    wire j;
    wire memread;
    wire memwrite;
    wire memtoreg;
    wire [1:0] ALUop;
    wire regwrite;
    wire regdst;
    wire [31:0] A,B;
    wire [31:0] readdata;
    wire [31:0] readdata1;
    wire [4:0] readaddr_reg;
    wire [31:0] MDR;
    wire zero;
    wire set_1;
    wire [3:0] aluctr;
    wire [7:0] readaddr_mem,writeaddr_mem;
    wire [31:0] sign_extend;
    wire [5:0] opcode,funct;
```

```
  wire [4:0] rs,rt,rd,shamt;
  wire [15:0] imm;
  wire [25:0] jump;
  wire [31:0] jump_shift;
  wire [31:0] ALUout;
  wire [31:0] writedata_mux,ALUsrcA_mux;
  wire [7:0] IorD_mux;
  wire [4:0] writereg_mux;
  wire locked;
  wire clk_50;
  wire [3:0]
IF,ID,EX_R,EX_I1,EX_LW,EX_SW,EX_I3,EX_J,MEM_R,MEM_I1,WB_LW,MEM_SW,MEM_I3,MEM_LW;

  //定义状态
  assign IF=4'd1,
  ID = 4'd2,
  EX_R = 4'd3,
  EX_I1 = 4'd4,
  EX_LW = 4'd5,
  EX_I3 = 4'd6,
  EX_J = 4'd7,
  MEM_R = 4'd8,
  MEM_I1 = 4'd9,
  MEM_LW = 4'd10,
  MEM_SW = 4'd11,
  MEM_I3 = 4'd12,
  WB_LW = 4'd13,
  EX_SW = 4'd14;
  //定义操作数
  assign opcode=IR[31:26],
  funct = IR[5:0],
  rs = IR[25:21],
  rt = IR[20:16],
  rd = IR[15:11],
  shamt = IR[10:6],
  imm = IR[15:0],
  jump = IR[25:0];

  segc  segc(
  .clk  (clk_50),
  .rst  (rst),
  .in   (data),
  .an   (an),
  .seg  (seg));

  //例化内存
  dist_mem_gen_0 memory(
  .a(writeaddr_mem), // input wire [15 : 0] a 10
  .d(B), // input wire [11 : 0] d 11
  .dpra(readaddr_mem), // input wire [15 : 0] dpra 12
  .clk(cpuclk), // input wire clk 13
  .we(memwrite1), // input wire we 14
  .dpo(readdata),
  .spo(readdata1) ); // output wire [11 : 0] dpo

  //例化时钟
  clk_wiz_0 clk_wiz_0(
  .clk_in1(clk),
.clk_out1(clk_50),
.locked(locked),
.reset(rst)
);

//例化寄存器堆
regfile REGFILE(
.ra0(readaddr_reg),
.ra1(rt),
.wa(writereg_mux),
.wd(writedata_mux),
.we(regwrite),
.rst(rst),
.clk(cpuclk),
.rd0(A),
.rd1(B));

//例化控制信号
control_code CONTROL(
.opcode(opcode),
.funct(funct),
.clk(cpuclk),
.regdst(regdst),
.memtoreg(memtoreg),
.regwrite(regwrite),
.memread(memread),
.memwrite(memwrite),
.ALUop(ALUop),
.pcwritecond(PCwritecond),
.pcsource(PCsource),
.j(j),
.addi(addi),
.alusrcB(ALUsrcB));

//例化ALU
alu ALU(
.input1(ALUsrcA_mux),
.input2(ALUsrcB_mux),
.ALUop(ALUop),
.funct(funct),
.opcode(opcode),
.aluctr(aluctr),
.clk(cpuclk),
.rst(rst),
.alures(ALUout),
.zero(zero));

assign opcode_DDU=readdata1[31:26];
assign MEMORY=PC[9:2];
assign cpuclk=cont?clk_50:step;
assign readaddr_reg =(choose & ~mem)?DDU_addr[4:0]:rs;
assign readaddr_mem=(choose & mem)?DDU_addr:IorD_mux;
//assign writeaddr_mem = (choose & mem)?DDU_addr:ALUout[9:2];
assign writeaddr_mem = ALUout[9:2];
assign MDR = IorD?readdata:MDR;
//扩展位操作
assign sign_extend=(imm[15]&(addi|PCwritecond))?{16'hffff,imm}:{16'h0000,imm};
//移位操作
```

```
//assign shift_left2=(PC+(sign_extend<<2));
assign jump_shift={PC[31:28],jump,2'b00};

//多路选择器
assign IorD_mux = IorD?ALUout[9:2]:PC[9:2];
assign writereg_mux = regdst?rd:rt;
assign writedata_mux = memtoreg?MDR:ALUout;
assign ALUsrcA_mux = A;
assign data=mem?readdata:A;
//assign data=readdata1;

always @(posedge clk_50 or posedge rst)
begin
     if(rst)
     begin
        inc_ctr<=1;
        dec_ctr<=1;
        DDU_addr<=8'd0;
     end
     else
     begin
         if(inc & inc_ctr)
         begin
            DDU_addr<=DDU_addr+1;
            inc_ctr<=0;
         end
         if(~inc & ~inc_ctr)
             inc_ctr<=1;
         if(dec & dec_ctr)
         begin
              DDU_addr<=DDU_addr-1;
              dec_ctr<=0;
         end
        if(~dec & ~dec_ctr)
              dec_ctr<=1;
     end
end

always @(posedge cpuclk)
begin
     if(ALUsrcB==3'b000)
        ALUsrcB_mux=B;
     else if(ALUsrcB==3'b001)
        ALUsrcB_mux=32'd4;
     else if(ALUsrcB==3'b010)
        ALUsrcB_mux=sign_extend;
     else if(ALUsrcB==3'b011)
        ALUsrcB_mux=shift_left2;
end

//PCwritecond&zero
always @(current_state,PC_change,PCwritecond,zero,opcode)
begin
    if(current_state==EX_I3)
    begin
    if(opcode==6'b000100)
       PC_change=PCwritecond & zero ;
    else if(opcode==6'b000101)
```

```
        PC_change=PCwritecond & ~zero;
     else
        PC_change=0;
     end
     else
        PC_change=0;
end

always @(posedge cpuclk or posedge rst)
begin
     if(rst)
     begin
        current_state=4'd1;
         PC<=32'h0000_0000;
         IorD<=0;
    end
     else
     begin
     case(current_state)
     IF:
     begin
        IorD<=0;
        PC<=PC+4;
        IR<=readdata;
        memwrite1<=0;
        current_state<=ID;
     end
     ID:
     begin
       if(IR==32'h0000)
            current_state<=IF;
        else
        begin
         if(opcode==6'd0)
             current_state<=EX_R;
         else if(opcode==6'b100011)
             current_state<=EX_LW;
         else if(opcode==6'b101011)
             current_state<=EX_SW;
         else if(opcode==6'b000100 || opcode==6'b000101)
             current_state<=EX_I3;
         else if(opcode==6'b000010)
             current_state<=EX_J;
         else
             current_state<=EX_I1;
         end
       end
       EX_R:
       begin
           current_state<=MEM_R;
        end
        MEM_R:
            current_state<=IF;
        EX_I1:
        begin
            current_state<=MEM_I1;
         end
         MEM_I1:
                current_state<=IF;
            EX_LW:
            begin
                 current_state<=MEM_LW;
              end
              EX_SW:
              begin
                  current_state<=MEM_SW;
              end
              MEM_LW:
              begin
                  IorD<=1;
                  current_state<=WB_LW;
               end
               WB_LW:
               begin
                   IorD<=0;
                   current_state<=IF;
               end
               MEM_SW:
               begin
                    memwrite1<=1;
                    current_state<=IF;
               end
               EX_I3:
               begin
                   if(PC_change)
                   begin
                        shift_left2=(PC+(sign_extend<<2));
                        PC<=shift_left2;
                   end
                   IorD<=0;
                   current_state<=MEM_I3;
               end
               MEM_I3:
               begin
                    current_state<=IF;
               end
               EX_J:
               begin
                   PC<=jump_shift;
                   IorD<=0;

       current_state<=IF;
               end
             endcase
          end
      end
   endmodule
  {%endcodeblock%}
```

 * 仿真源码

```
   {%codeblock lang:verilog%}
   module CPU_t(

       );
   reg cont, step, mem, inc, dec, clk, choose, rst;
```

```
    wire [7:0] an;
    wire [6:0] seg;
    wire [7:0] MEMORY;
    integer i,j;
    CPU cpu(.cont(cont), .step(step),
.mem(mem),.inc(inc),.dec(dec),.clk_50(clk),.choose(choose),.rst(rst),.an(an),.se
g(seg),.MEMORY(MEMORY));
    initial
    begin
    clk=1;
    step=1;
    while(1)
    begin
    #5 clk=~clk;
    step=~step;
    end
    end

    initial
    begin
    rst=1;
    #20 rst=0;
    end

    initial
    begin
    cont=0;
    mem=0;
    choose=0;
    #1000 cont=1;
    #1000 mem=1;
    choose=1;
    end

    initial
    begin
    inc=0;
    dec=0;
    #3000
    for(i=0;1;i=i+1)
    begin
    #10 inc=~inc;
    end
    end

    endmodule
   {%endcodeblock%}
```