多周期CPU

浙江大学

本科实验报告

课程名称	计算机组成
实验名称	多周期CPU设计实验
小组成员	高宇、张佳文、付健豪
指导老师	陆魁军

2020年8月14日

一、实验目的

1. 掌握多周期CPU的工作原理；
2. 掌握9条常规指令多周期CPU的设计与开发，包括LW, SW, ADD, SUB, AND, OR,NOR,BEQ,和J指令。

二、实验内容

1. 实现多周期CPU的各个模块；
2. 完成CPU模块的综合、仿真和FPGA实现。

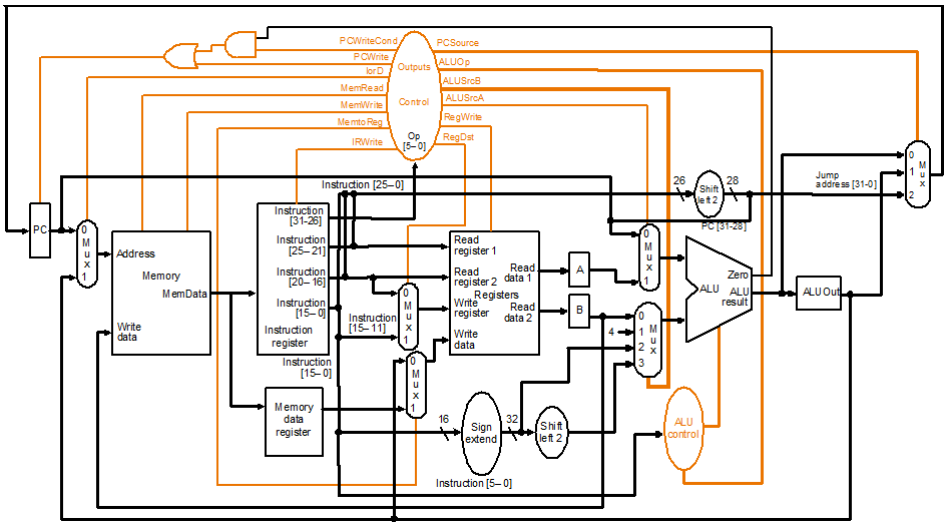
三、主要仪器设备

1. 装有Xilinx ISE 14.7的计算机　　1台
2. SWORD开发板　　1套

四、实验原理

在多周期CPU中，每条指令的执行需要多个时钟周期，也就是说需要将整个CPU的执行过程分成几个阶段。每个阶段需要具体执行的任务由控制单元发出的信号进行控制，多个基本部件在控制器的调度下进行工作。由于同一指令在多个时钟周期被用到，故需要增加IR寄存器，用于在未来几个时钟周期内产生控制信号直到该指令完成。

CPU执行指令时，一般需要经过取指、指令译码、执行、存储器访问和写回等几个步骤。当然，每条指令经过的步骤不同。多周期CPU相应的电路图如下：



为完成多周期CPU，我们需要ALU、寄存器组、pc寄存器读写、内存读写、控制器等5个基本组件，其中IR寄存器的读写我们放到了内存读写组件中。最后，通过top模块将基本组件连接起来并在top模块中添加了调试模块。下面分组件具体介绍。

alu.v实现了算术逻辑运算单元模块。

ALU采用分级控制的方法：先由CPU控制器产生控制信号ALUOp、ALUSrcA和ALUSrcB，其中ALUOp控制ALU进行何种操作，ALUSrcA和ALUSrcB决定了ALU操作数的来源，具体对应关系如下

信号名称	信号值	信号值所对应效果
ALUOp	00	加法操作
	01	减法操作
	10	根据指令的funct字段来决定操作类型
ALUSrcA	0	第一个操作数为PC
	1	第一个操作数为指令中第一个源寄存器
ALUSrcB	00	第二个操作数为指令中第二个源寄存器
	01	第二个操作数为常数4
	10	第二个操作数为指令低16位的符号扩展
	11	第二个操作数为指令低16位左移2位后的符号扩展

从上面可以看出，ALU单元的输入除三个控制信号外，还需要指令的funct字段、PC、指令两个源寄存器对应的内容以及指令低16位。另外，每一个周期之后需要将ALU的运算结果锁存到ALUOut寄存器中，故还需要时钟信号输入。R类型指令funct字段对应的具体指令关系如下

funct字段值	对应具体类型
100000	add
100010	sub
100100	and
100101	or
100111	nor

alu.v的代码如下

```
`timescale 1ns / 1ps

module alu(//input
                clk,ALUOp,ALUSrcA,ALUSrcB,funct,
                PC,regA,regB,IR_low16,
                //output
                result,ALUOut,zero //zero指示当前result是否为0,供beq
```

```
判断用
    );
    input clk;
    input[1:0] ALUOp;
    input ALUSrcA;
    input[1:0] ALUSrcB;
    input[5:0] funct;
    input[31:0] PC,regA,regB;
    input[15:0] IR_low16;

    output reg[31:0] result,ALUOut;
    output reg zero;

    reg[31:0] A;
    reg[31:0] B;
    wire[31:0] ext;
    wire[31:0] ext2;

    initial begin
        result=0;
    end

    assign ext={{16{IR_low16[15]}},IR_low16[15:0]};//按字寻址,无需左移
    assign ext2={{14{IR_low16[15]}},2'b00,IR_low16[15:0]};
    always@*
    begin
        A=(ALUSrcA==0)?PC:regA;
        case(ALUSrcB)
            2'b00:B=regB;
            2'b01:B=1;
            2'b10:B=ext;
            2'b11:B=ext2;
        endcase
        case(ALUOp)
            2'b00:result=A+B;
            2'b01:result=A-B;
            2'b10:
                case(funct)
                    6'b100000:result=A+B;
```

```
                    6'b100010:result=A-B;
                    6'b100100:result=A&B;
                    6'b100101:result=A|B;
                    6'b100111:result=~(A|B);
                endcase
        endcase
        zero=(result==0)?1:0;
    end

    always@(posedge clk) //每一周期结束将ALU计算结果锁存到ALUOut中
    begin
        ALUOut<=result;
    end

endmodule
```

regs.v实现了寄存器组。

寄存器文件是CPU中存放数据的主要场所。寄存器共有32个，每一个寄存器均为32位。该模块通过控制单元输出的控制信号对寄存器组进行相应的读或写操作。针对寄存器组的控制信号有RegDst、RegWrite和MemtoReg。控制信号含义如下

信号名称	信号值为0时的效果	信号值为1时的效果
RegDst	目的寄存器为rd	目的寄存器为rt
RegWrite	无	将特定值写入目的寄存器
MemtoReg	写入目的寄存器的值来自ALU的计算结果	写入目的寄存器的值来自内存

可见，除控制信号外，输入还应有rs、rt、rd寄存器编号以及ALU计算结果和内存值，还有时钟和复位信号；输出为rs和rt对应寄存器的内容。此外还要有另外的一个调试数据输出端口。
regs.v代码如下

```
`timescale 1ns / 1ps

module regs(//input
                clk,rst,RegDst,RegWrite,MemtoReg,
                rs,rt,rd,ALUOut,MemContent,dbgReg,
                //output
                rsContent,rtContent,dbgContent
```

```
    );
    input clk,rst,RegDst,RegWrite,MemtoReg;
    input[4:0] rs,rt,rd,dbgReg;
    input[31:0] ALUOut,MemContent;
    output reg[31:0] rsContent,rtContent;
    output [31:0] dbgContent;

    reg[31:0] regFile[31:0];
    wire[4:0] RealRd;
    wire[31:0] WriteData;
    integer i;

    always@(posedge clk) begin
    rsContent<=regFile[rs];
    rtContent<=regFile[rt];
    end

    assign dbgContent=regFile[dbgReg];
    assign RealRd=(RegDst==0)?rd:rt; //选择写入寄存器编号
    assign WriteData=(MemtoReg==0)?ALUOut:MemContent; //选择写入寄存器的内容

    initial begin
        for(i=0;i<32;i=i+1) regFile[i]<=0;
    end
    always@(posedge clk or posedge rst)
    begin
        if(rst) begin
            for(i=0;i<32;i=i+1)
                regFile[i]<=0;
        end
        else begin
            if((RealRd!=0)&&(RegWrite==1)) //防止对r0写入数据
                regFile[RealRd]<=WriteData;
        end
    end

endmodule
```

pc.v实现了对PC寄存器的操作。

PC寄存器用于保存当前执行指令的地址。该单元根据控制器产生的信号来更新PC寄存器。针对PC寄存器的控制信号主要有PCWrite、PCWriteCond和PCSource。各信号值的具体效果如下

信号名称	信号值	信号值所对应效果
PCWrite	0	无
	1	根据PCSource写PC寄存器
PCWriteCond	0	无
	1	如果ALU输出为0则写PC
PCSource	00	写PC的内容为ALU输出
	01	写PC的内容为ALUOut寄存器
	10	写PC的内容为jump指令的目标地址

从上面可以看到，PC还需要ALU的zero和result输出以及ALUOut寄存器内容以及指令的低26位。
pc.v代码如下

```
`timescale 1ns / 1ps

module pc(//input
            clk,rst,PCWrite,PCWriteCond,PCSource,
            zero,result,ALUOut,IR_low26,
            //output
            PCvalue //PCvalue即为PC寄存器
    );
    input clk,rst,PCWrite,PCWriteCond,zero;
    input[1:0] PCSource;
    input[31:0] result,ALUOut;
    input[25:0] IR_low26;
    output reg[31:0] PCvalue;

    reg[31:0] nextPC;

    initial begin
        PCvalue=0;
    end

```

```
    always@(posedge clk or posedge rst)
    begin       //仅时钟上升沿更新PC寄存器
        if(rst) begin
            PCvalue<=0;
        end
        else begin
            PCvalue<=nextPC;
        end
    end

    always@*
    begin //nextPC根据控制信号更新
        if(PCWriteCond==1&&zero==1)
            nextPC=ALUOut;
        else if(PCWrite==1)
            begin
                case(PCSource)
                2'b00:nextPC=result;
                2'b10:nextPC[25:0]=IR_low26;
                endcase
            end
        else
            nextPC=PCvalue;
    end

endmodule
```

memory.v实现了对内存的读写。

我们将数据和指令放在同一个存储器中。该模块根据CPU产生的控制信号对特定位置的内存进行读取或写入，IR寄存器的写入也放到了该模块中。控制信号包括MemRead、MemWrite、IorD以及IRWrite，具体效果如下

信号名称	信号值为0时的效果	信号值为1时的效果
MemRead	无	读取特定地址的内存并放入到dataout中
MemWrite	无	将输入的内容写入到特定地址的内存中
IorD	内存访问地址由PC指定	内存访问地址由ALUOut寄存器指定
IRWrite	无	将内存单元dataout中的内容写入到IR寄存器

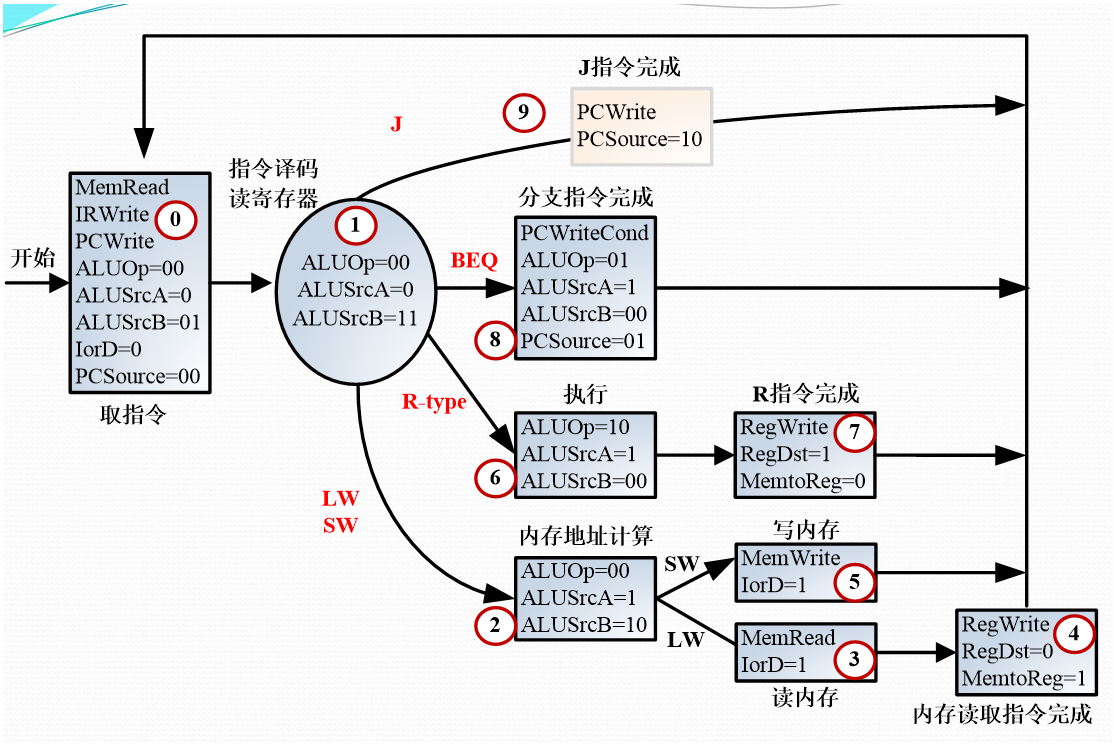
		中

对内存的直接读写通过调用RAM的IP模块来实现。IP核的导入将在后面陈述。
memory.v的代码如下

```
`timescale 1ns / 1ps

module memory(//input
                clk,MemRead,MemWrite,IorD,IRWrite,
                PC,ALUOut,wdata,
                //output
                dataout,IR
    );
    input clk,MemRead,MemWrite,IorD,IRWrite;
    input[31:0] PC,ALUOut,wdata; //wdata为可能写入内存的数据
    output[31:0] dataout;
    output reg[31:0] IR;

    wire[31:0] addr;

    assign addr=(IorD==0)?PC:ALUOut; //选择内存访问地址
    initial begin
        IR=0;
    end
    RAMIP ram(
            .clka(clk),
            .addra(addr[8:0]),
            .dina(wdata),
            .douta(dataout),
            .wea(MemWrite)
            );

    always@*
    begin
        if(IRWrite==1)
            IR<=dataout; //写IR寄存器
    end
endmodule
```

control.v 实现了CPU控制器。

控制器是CPU的核心模块。控制器根据指令的操作码和当前的CPU状态产生下一个状态以及控制信号。该模块主要实现了一个有限状态机，其在时钟信号的驱动下进行状态转换，一步步完成各条指令的执行。各个控制信号的具体效果我们已经在前述各模块中详细叙述。下面给出有限状态机的示意图



control.v代码如下

```
`timescale 1ns / 1ps

//信号控制单元模块：ControlUnit
//输入：时钟信号clk，零标志位zero，符号位标志sign
//输出：各个控制信号
module control(
    input clk,
    input rst,
    input [5:0] opcode,
    //input zero,
    //input sign,
    output reg RegWrite,
    output reg PCWrite,
    output reg IRWrite,
    output reg PCWriteCond,
    output reg IorD,
```

```
17      output reg MemRead,
18      output reg MemWrite,
19      output reg MemtoReg,
20      output reg RegDst,
21      output reg ALUSrcA,
22      output reg [1:0] ALUSrcB,
23      output reg [1:0] ALUOp,
24      output reg [1:0] PCSource,
25      output reg [4:0] beat
26      //output reg [2:0] out_state
27      );
28      //先将对应情况的阶段和对应情况的指令设置成常数方便进行编写代码
29      parameter [3:0] s0=4'b0000, s1=4'b0001, s2=4'b0010, s3=4'b00
    11, s4=4'b0100, s5=4'b0101, s6=4'b0110, s7=4'b0111, s8=4'b1000,
    s9=4'b1001;
30      parameter [5:0] Rtype=6'b000000, SW=6'b101011, LW=6'b100011,
    BEQ=6'b000100, J=6'b000010, HALT=6'b111111; //指令名常量
31      reg [3:0] state, next_state;    //state为当前所状处的状态，next_
    state是当前状态的下一个状态
32      reg [31:0] count;
33
34      //1.先对各个输出信号及当前阶段进行初始化
35      initial begin
36          RegWrite = 0;
37          PCWrite = 0;
38          IRWrite = 0;
39          PCWriteCond = 0;
40          IorD = 0;
41          MemRead = 0;
42          MemWrite = 0;
43          MemtoReg = 0;
44          RegDst = 0;
45          ALUSrcA = 0;
46          ALUSrcB = 2'b00;
47          ALUOp = 2'b00;
48          PCSource = 2'b00;
49          beat = 5'b00000;
50          count=32'h00000000;
51            state = s0;
52              next_state = s0;
```

```
    end
    //2.D触发器模块：并行对当前阶段进行更新
    always @(posedge clk or posedge rst) begin
    if(rst) state <= s0;
    else state <= next_state;
        //if(rst) begin
        //    state <= s0;
        //end
        //else begin
        //    state <= next_state;
        //end
            //out_state = state;
    end
    //3.阶段转移模块：确定下一个阶段
    always @* begin
        case(state)
            //当前阶段：s0
            s0: begin
                beat = 5'b00001;
                PCWrite = 1;
                PCWriteCond = 0;
                IorD = 0;
                MemRead = 1;
                MemWrite = 0;
                IRWrite = 1;
                MemtoReg = 0;
                ALUSrcA = 0;
                RegWrite = 0;
                RegDst = 0;
                PCSource = 2'b00;
                ALUOp = 2'b00;
                ALUSrcB = 2'b01;
                next_state = s1;
                count=count+32'h00000001;
            end
            //当前阶段：s1
            s1: begin
                beat = 5'b00010;
                PCWrite = 0;
                PCWriteCond = 0;
```

```
93                  IorD = 0;
94                  MemRead = 0;
95                  MemWrite = 0;
96                  IRWrite = 0;
97                  MemtoReg = 0;
98                  ALUSrcA = 0;
99                  RegWrite = 0;
100                 RegDst = 0;
101                 PCSource = 2'b00;
102                 ALUOp = 2'b00;
103                 ALUSrcB = 2'b11;
104                 case(opcode)
105                     LW: next_state = s2;
106                     SW: next_state = s2;
107                     Rtype: next_state = s6;
108                     BEQ: next_state = s8;
109                     J: next_state = s9;
110                 endcase
111             end
112             //当前阶段：s2
113             s2: begin
114                 beat = 5'b00100;
115                 PCWrite = 0;
116                 PCWriteCond = 0;
117                 IorD = 0;
118                 MemRead = 0;
119                 MemWrite = 0;
120                 IRWrite = 0;
121                 MemtoReg = 0;
122                 ALUSrcA = 1;
123                 RegWrite = 0;
124                 RegDst = 0;
125                 PCSource = 2'b00;
126                 ALUOp = 2'b00;
127                 ALUSrcB = 2'b10;
128                 case(opcode)
129                     LW: next_state = s3;
130                     SW: next_state = s5;
131                     default next_state = s2;
132                 endcase
```

```
133             end
134             //当前阶段：s3
135             s3: begin
136                 beat = 5'b01000;
137                 PCWrite = 0;
138                 PCWriteCond = 0;
139                 IorD = 1;
140                 MemRead = 1;
141                 MemWrite = 0;
142                 IRWrite = 0;
143                 MemtoReg = 0;
144                 ALUSrcA = 0;
145                 RegWrite = 0;
146                 RegDst = 0;
147                 PCSource = 2'b00;
148                 ALUOp = 2'b00;
149                 ALUSrcB = 2'b00;
150                 next_state = s4;
151             end
152             //当前阶段：s4
153             s4: begin
154                 beat = 5'b10000;
155                 PCWrite = 0;
156                 PCWriteCond = 0;
157                 IorD = 0;
158                 MemRead = 0;
159                 MemWrite = 0;
160                 IRWrite = 0;
161                 MemtoReg = 1;
162                 ALUSrcA = 0;
163                 RegWrite = 1;
164                 RegDst = 1;
165                 PCSource = 2'b00;
166                 ALUOp = 2'b00;
167                 ALUSrcB = 2'b00;
168                 next_state = s0;
169             end
170             //当前阶段：s5
171             s5: begin
172                 beat = 5'b01000;
```

```
173                 PCWrite = 0;
174                 PCWriteCond = 0;
175                 IorD = 1;
176                 MemRead = 0;
177                 MemWrite = 1;
178                 IRWrite = 0;
179                 MemtoReg = 0;
180                 ALUSrcA = 0;
181                 RegWrite = 0;
182                 RegDst = 0;
183                 PCSource = 2'b00;
184                 ALUOp = 2'b00;
185                 ALUSrcB = 2'b00;
186                 next_state = s0;
187             end
188             //当前阶段：s6
189             s6: begin
190                 beat = 5'b00100;
191                 PCWrite = 0;
192                 PCWriteCond = 0;
193                 IorD = 0;
194                 MemRead = 0;
195                 MemWrite = 0;
196                 IRWrite = 0;
197                 MemtoReg = 0;
198                 ALUSrcA = 1;
199                 RegWrite = 0;
200                 RegDst = 0;
201                 PCSource = 2'b00;
202                 ALUOp = 2'b10;
203                 ALUSrcB = 2'b00;
204                 next_state = s7;
205             end
206             //当前阶段：s7
207             s7: begin
208                 beat = 5'b01000;
209                 PCWrite = 0;
210                 PCWriteCond = 0;
211                 IorD = 0;
212                 MemRead = 0;
```

```
213                 MemWrite = 0;
214                 IRWrite = 0;
215                 MemtoReg = 0;
216                 ALUSrcA = 0;
217                 RegWrite = 1;
218                 RegDst = 0;
219                 PCSource = 2'b00;
220                 ALUOp = 2'b00;
221                 ALUSrcB = 2'b00;
222                 next_state = s0;
223             end
224             //当前阶段：s8
225             s8: begin
226                 beat = 5'b00100;
227                 PCWrite = 0;
228                 PCWriteCond = 1;
229                 IorD = 0;
230                 MemRead = 0;
231                 MemWrite = 0;
232                 IRWrite = 0;
233                 MemtoReg = 0;
234                 ALUSrcA = 1;
235                 RegWrite = 0;
236                 RegDst = 0;
237                 PCSource = 2'b01;
238                 ALUOp = 2'b01;
239                 ALUSrcB = 2'b00;
240                 next_state = s0;
241             end
242             //当前阶段：s9
243             s9: begin
244                 beat = 5'b00100;
245                 PCWrite = 1;
246                 PCWriteCond = 0;
247                 IorD = 0;
248                 MemRead = 0;
249                 MemWrite = 0;
250                 IRWrite = 0;
251                 MemtoReg = 0;
252                 ALUSrcA = 0;
```

```
253                 RegWrite = 0;
254                 RegDst = 0;
255                 PCSource = 2'b10;
256                 ALUOp = 2'b00;
257                 ALUSrcB = 2'b00;
258                 next_state = s0;
259             end
260         endcase
261     end
262 endmodule
```

top.v是整个CPU的顶层模块。

顶层模块负责将各个子模块进行信号连接。另外，我们在top.v中放置了调试模块用来显示CPU内部状态。我们首先叙述一下调试模块。调试模块在开发板上的显示区域主要分LED灯和七段码两部分。LED灯使用最右边的8个，从左至右的前三个依次显示手动时钟信号、自动时钟信号和复位信号；其余5位以增量形式显示当前节拍数；LED灯的显示通过调用SPIO模块来实现，只需要更改P_Data输入端口。七段码使用最右边的四个，并通过最右边的7个拨动开关进行控制；从左至右的前两个拨动开关选择显示内容，后面5个设置寄存器编号。
添加好调试模块后，只需将各个模块用定义在top中的变量连接起来即可。
top.v的代码如下

```
1  `timescale 1ns / 1ps
2
3  module top(
4      input wire clk,                  //时钟
5      input wire RSTN,                     //复位
6
7      input [3:0] BTN_Y,               //按键输入
8      input [7:0] SW,
9
10     output readn,                    //三色led灯
11     output CR,                           //三色信号灯
12     output RDY,                          //三色信号灯
13
14     output [4:0] BTN_X,            //输出按键
15
16     output wire led_clk,           //串行移位时钟
17     output wire led_sout,          //串行输出
```

```
    output wire led_clrn,          //LED显示清零
    output wire LED_PEN,           //LED显示刷新使能

    output seg_clk, //串行移位时钟
    output seg_sout,    //七段显示数据(串行输出)
    output SEG_PEN, //七段码显示刷新使能
    output seg_clrn,    //七段码显示汪零);//显示7段码

    output Buzzer
);
    reg[15:0] disp_num;
    wire[3:0] blink,dots;
    wire [3:0] BTN_OK;
    //wire [15:0]SW_OK,
    wire [31:0] P_Data;          //并行输入，用于串行输出数据
    wire [31:0] Div;

    wire RegWrite,PCWrite,IRWrite,PCWriteCond,IorD;
    wire MemRead,MemWrite,MemtoReg,RegDst,ALUSrcA,ALUZero;
   wire[1:0] ALUSrcB,ALUOp,PCSource;
   wire[4:0] beat;
    wire[31:0] PC,IR,ALUOut,ALUResult,MemContent;
    wire[31:0] regAContent,regBContent,dbgContent;
    wire dbgclk,rst_cpu,clk_cpu;
    reg [3:0] BTN_cnt;

    clk_div U4(clk,rst,SW2,Div,Clk_CPU);//分频
    always @ (posedge clk_cpu or posedge rst_cpu) begin
        if (rst_cpu == 1) //BTN_cnt为时钟计数
            BTN_cnt = 4'b0;
        else begin
            BTN_cnt = BTN_cnt + 1;
        end
    end

    SAnti_jitter U1(clk,RSTN,readn,BTN_Y,BTN_X,Key_out,RDY,pulse
_out,BTN_OK,SW_OK,CR,rst);
    assign dbgclk=BTN_OK[0];
    assign rst_cpu=BTN_OK[1];
    assign clk_cpu=(SW[7]==0)?dbgclk:clk; //选择自动时钟或手动时钟
```

```
57
58      SPIO U2(clk,rst,Div[20],1,P_Data,counter_set,LED_out,led_clk
    ,led_sout,led_clrn,LED_PEN,GPIOf0);//显示led
59      assign P_Data[17:2]={{8{0}},dbgclk,clk,rst_cpu,beat};//LED
60
61      SSeg7_Dev U3(clk,rst,Div[20],1,Div[25],{16'h0000,disp_num},{
    4'b0000,dots},{4'b0000,blink},seg_clk,seg_sout,SEG_PEN,seg_clrn)
    ;//显示7段码
62      always@* //根据拨动开关选择七段码的输出
63      begin
64          if(SW[6:5]==2'b00)begin
65              disp_num[15:0]=dbgContent[15:0];
66          end
67          if(SW[6:5]==2'b01)begin
68              disp_num[15:0]=dbgContent[31:16];
69          end
70          if(SW[6:5]==2'b10)begin
71              disp_num[15:12]=BTN_cnt;
72              case(IR[31:26])
73              6'b000000:disp_num[11:8]=10;
74              6'b101011:disp_num[11:8]=11;
75              6'b100011:disp_num[11:8]=11;
76        6'b000100:disp_num[11:8]=11;//BEQ指令应为I型
77              6'b000010:disp_num[11:8]=12;
78              endcase
79              case(beat)
80              5'b00001:disp_num[7:4]=0;
81              5'b00010:disp_num[7:4]=1;
82              5'b00100:disp_num[7:4]=2;
83              5'b01000:disp_num[7:4]=3;
84              5'b10000:disp_num[7:4]=4;
85              endcase
86              disp_num[3:0]=PC[3:0];
87          end
88      end
89
90      assign Buzzer = 1;
91
92      control x_control( //控制器
93                  .clk(clk_cpu),
```

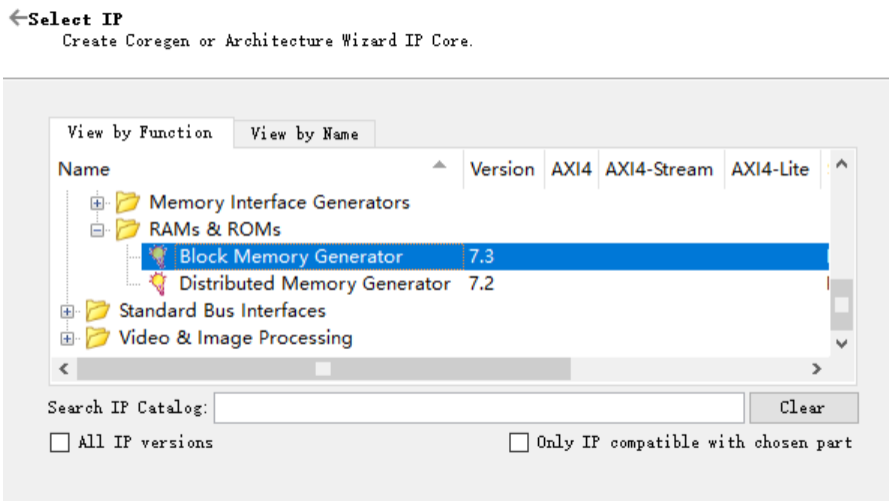
```
94                  .rst(rst_cpu),
95                  .opcode(IR[31:26]),
96                  .RegWrite(RegWrite),
97                  .PCWrite(PCWrite),
98                  .IRWrite(IRWrite),
99                  .PCWriteCond(PCWriteCond),
100                 .IorD(IorD),
101                 .MemRead(MemRead),
102                 .MemWrite(MemWrite),
103                 .MemtoReg(MemtoReg),
104                 .RegDst(RegDst),
105                 .ALUSrcA(ALUSrcA),
106                 .ALUSrcB(ALUSrcB),
107                 .ALUOp(ALUOp),
108                 .PCSource(PCSource),
109                 .beat(beat)
110                 );
111
112     memory x_memory( //内存读写
113                 .clk(clk_cpu),
114                 .MemRead(MemRead),
115                 .MemWrite(MemWrite),
116                 .IorD(IorD),
117                 .IRWrite(IRWrite),
118                 .PC(PC),
119                 .ALUOut(ALUOut),
120                 .wdata(regBContent),
121                 .dataout(MemContent),
122                 .IR(IR)
123                 );
124
125     pc x_pc( //PC寄存器读写
126             .clk(clk_cpu),
127             .rst(rst_cpu),
128             .PCWrite(PCWrite),
129             .PCWriteCond(PCWriteCond),
130             .PCSource(PCSource),
131             .zero(ALUZero),
132             .result(ALUResult),
133             .ALUOut(ALUOut),
```

```
134             .IR_low26(IR[25:0]),
135             .PCvalue(PC)
136             );
137 
138 
139     alu x_alu( //ALU
140                 .clk(clk_cpu),
141                 .ALUOp(ALUOp),
142                 .ALUSrcA(ALUSrcA),
143                 .ALUSrcB(ALUSrcB),
144                 .funct(IR[5:0]),
145                 .PC(PC),
146                 .regA(regAContent),
147                 .regB(regBContent),
148                 .IR_low16(IR[15:0]),
149                 .result(ALUResult),
150                 .ALUOut(ALUOut),
151                 .zero(ALUZero)
152                 );
153 
154     regs x_regs( //寄存器组
155                 .clk(clk_cpu),
156                 .rst(rst_cpu),
157                 .RegDst(RegDst),
158                 .RegWrite(RegWrite),
159                 .MemtoReg(MemtoReg),
160                 .rs(IR[25:21]),
161                 .rt(IR[20:16]),
162                 .rd(IR[15:11]),
163                 .dbgReg(SW[4:0]),
164                 .ALUOut(ALUOut),
165                 .MemContent(MemContent),
166                 .rsContent(regAContent),
167                 .rtContent(regBContent),
168                 .dbgContent(dbgContent)
169                 );
170 
171 endmodule
```

五、操作方法与实验步骤

1. 创建工程，选择正确的平台参数。
2. 添加在实验原理中贴出的代码，并将用于防抖动、LED显示和七段码显示的ngc文件和verilog文件导入到工程中。将top.v设为顶层模块。
3. 添加RAM的IP核。

   右键选择new source,选择源类型为IP，输入文件名"RAMIP",选择下图所示IP核。



在IP核配置界面选择单口RAM，读写宽度选择32bit，内存大小设置为512(或其它大小亦可,不要小于数据和指令所需要的大小)。用我们事先写好的coe文件初始化内存,coe文件内容在报告最后。其中,前14行存放的为指令，指令的具体意义请参见最后的表格。后面的行用于存放数据。

4. 分配引脚

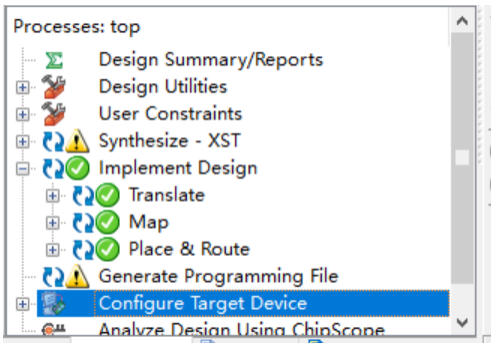
   ucf文件内容如下

```
#系统时钟
NET "clk"       LOC = AC18      | IOSTANDARD = LVCMOS18 ;
NET "RSTN"              LOC = W13       | IOSTANDARD = LVCMOS18 ;
NET "clk"       TNM_NET = TM_CLK ;
TIMESPEC TS_CLK_100M = PERIOD "TM_CLK"  10 ns HIGH 50%;
#LED串行接口
NET "led_clk"           LOC = N26   |IOSTANDARD = LVCMOS33 ;
NET "led_clrn"          LOC = N24   | IOSTANDARD = LVCMOS33 ;
NET "led_sout"          LOC = M26   | IOSTANDARD = LVCMOS33 ;
NET "LED_PEN"           LOC = P18   | IOSTANDARD = LVCMOS33 ;
#七段码串行接口
NET "seg_clk"           LOC = M24   | IOSTANDARD = LVCMOS33 ;
NET "seg_clrn"          LOC = M20   | IOSTANDARD = LVCMOS33 ;
NET "seg_sout"          LOC = L24   | IOSTANDARD = LVCMOS33 ;
```

```
15 NET "SEG_PEN"            LOC = R18    | IOSTANDARD = LVCMOS33 ;
16 #三色信号灯：Tri_LED
17 NET "RDY"                LOC = U21    | IOSTANDARD = LVCMOS33 ;#LED
   _nR0
18 NET "readn"              LOC = U22    | IOSTANDARD = LVCMOS33 ;#LED
   _nG0
19 NET "CR"                     LOC = V22    | IOSTANDARD = LVCMOS33 ;
   #LED_nB0
20
21 #阵列式按键
22 NET "BTN_x[0]"           LOC = V17    | IOSTANDARD = LVCMOS18 ;#ROW
   0
23 NET "BTN_x[1]"           LOC = W18    | IOSTANDARD = LVCMOS18 ;#ROW
   1
24 NET "BTN_x[2]"           LOC = W19    | IOSTANDARD = LVCMOS18 ;#ROW
   2
25 NET "BTN_x[3]"           LOC = W15    | IOSTANDARD = LVCMOS18 ;#ROW
   3
26 NET "BTN_x[4]"           LOC = W16    | IOSTANDARD = LVCMOS18 ;#ROW
   4
27 NET "BTN_y[0]"           LOC = V18    | IOSTANDARD = LVCMOS18 ;#COL
   0
28 NET "BTN_y[1]"           LOC = V19    | IOSTANDARD = LVCMOS18 ;#COL
   1
29 NET "BTN_y[2]"           LOC = V14    | IOSTANDARD = LVCMOS18 ;#COL
   2
30 NET "BTN_y[3]"           LOC = W14    | IOSTANDARD = LVCMOS18 ;#COL
   3
31 #switch
32 NET "SW[0]"              LOC = AA10   | IOSTANDARD = LVCMOS15 ;
33 NET "SW[1]"              LOC = AB10   | IOSTANDARD = LVCMOS15 ;
34 NET "SW[2]"              LOC = AA13   | IOSTANDARD = LVCMOS15 ;
35 NET "SW[3]"              LOC = AA12   | IOSTANDARD = LVCMOS15 ;
36 NET "SW[4]"              LOC = Y13    | IOSTANDARD = LVCMOS15 ;
37 NET "SW[5]"              LOC = Y12    | IOSTANDARD = LVCMOS15 ;
38 NET "SW[6]"              LOC = AD11   | IOSTANDARD = LVCMOS15 ;
39 NET "SW[7]"              LOC = AD10   | IOSTANDARD = LVCMOS15 ;
40
41
42 #ArDUNIO-Sword-002-Basic IO
```

```
43 NET "Buzzer"            LOC = AF24   | IOSTANDARD = LVCMOS33 ;
```

5. 生成bit文件并下载到开发板上进行验证。



六、实验结果与分析

Control仿真

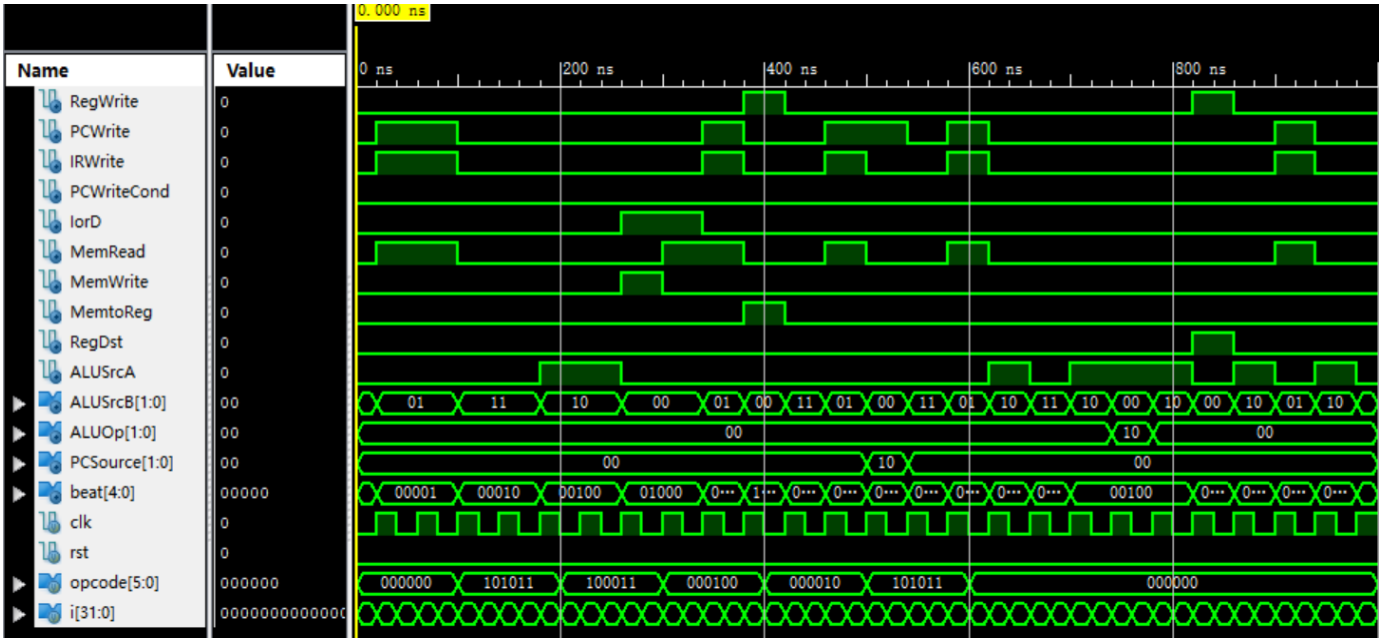
首先对控制单元进行了仿真验证，仿真代码如下

```
1  module controlSim;
2
3      // Inputs
4      reg clk;
5      reg rst;
6      reg [5:0] opcode;
7
8      // Outputs
9      wire RegWrite;
10     wire PCWrite;
11     wire IRWrite;
12     wire PCWriteCond;
13     wire IorD;
14     wire MemRead;
15     wire MemWrite;
16     wire MemtoReg;
17     wire RegDst;
18     wire ALUSrcA;
19     wire [1:0] ALUSrcB;
20     wire [1:0] ALUOp;
21     wire [1:0] PCSource;
22     wire [4:0] beat;
```

```
23
24      // Instantiate the Unit Under Test (UUT)
25      control uut (
26          .clk(clk),
27          .rst(rst),
28          .opcode(opcode),
29          .RegWrite(RegWrite),
30          .PCWrite(PCWrite),
31          .IRWrite(IRWrite),
32          .PCWriteCond(PCWriteCond),
33          .IorD(IorD),
34          .MemRead(MemRead),
35          .MemWrite(MemWrite),
36          .MemtoReg(MemtoReg),
37          .RegDst(RegDst),
38          .ALUSrcA(ALUSrcA),
39          .ALUSrcB(ALUSrcB),
40          .ALUOp(ALUOp),
41          .PCSource(PCSource),
42          .beat(beat)
43      );
44      integer i=0;
45      initial begin
46          // Initialize Inputs
47          clk = 0;
48          rst = 0;
49          opcode = 0;
50
51          // Wait 100 ns for global reset to finish
52          #100;
53
54          // Add stimulus here
55          opcode = 6'b101011;
56          #100;
57          opcode = 6'b100011;
58          #100;
59          opcode = 6'b000100;
60          #100;
61          opcode = 6'b000010;
62          #100;
```

```
63          opcode = 6'b101011;
64          #100;
65          opcode = 6'b000000;
66      end
67      always@ * begin
68          for(i=0;i<100;i=i+1) begin
69              #20;
70              clk <= ~clk;
71          end
72      end
73  endmodule
```

仿真结果如下



将仿真图与实验原理一节控制器叙述时贴出的状态机进行比较，可见完全一致，控制器实现正确。

下载到开发板验证

七、心得体会

通过此次实验，进一步加深了对多周期CPU工作原理的认识，掌握了设计和开发支持常规指令多周期CPU的方法。在实验中，我们遇到了很多障碍。这些障碍，有的来自Verilog语法错误，比如wire和reg类型的混用；有的来自疏忽，比如变量的位数设置错误或忘记设置，调用模块时信号名称拼写错误；有的来自对

多周期CPU设计的不解，比如复位信号的处理，PC和IR寄存器何时读写等。我们还遇到了很多一开始令人不解的bug，但最终通过不断的调试观察CPU状态以及思考，解决了一个又一个的bug，得到了正确的结果。实验过后，对理论课上所学的多周期CPU设计理解更深了，比如ALUOut等寄存器的设置原因，同时Verilog编程也更加熟练了。

coe文件与两张表格

coe文件目录：**./src/memory.coe**

coe文件内容如下(其中前14行表示指令并与两张表格相对应):

```
MEMORY_INITIALIZATION_RADIX = 2;
MEMORY_INITIALIZATION_VECTOR =
10001100000000010000000000010100,
10001100000000010000000000010101,
00010000001000100000000000000011,
10001100000000110000000000010100,
00010000001000110000000000000001,
00000000001000100001100000100000,
10101100000000010000000000010110,
10001100000001000000000000010110,
00000000001000100001100000100000,
00000000001000110010000000100100,
00000000001001000010100000100010,
00000000001001000011000000100101,
00000000001001000011100000100111,
00001000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00010001001000100011001101000100,
00000000000100010010001000110011,
```

表格A

行号	指令	机器码	分割机器码
1	lw r1,80($0)	10001100000000010000000000010100,	100011-00000-00001-00000-00000-010100

2	lw r2,84($0)	10001100000000010000000000001 0101,	100011–00000–00010–0000000000010101
3	beq r1,r2,3	00010000001000010000000000000 00011,	000100–00001–00010–0000000000000011
4	lw r3,80($0)	10001100000000010000000000001 0100,	100011–00000–00011–0000000000010100
5	beq r1,r3,1	00010000001000010000000000000 00001,	000100–00001–00011–0000000000000001
6	add r3,r1,r2	00000000001000100001100000010 0000,	000000–00001–00010–00011–00000–100000
7	sw r1,88($0)	10101100000000001000000000001 0110,	101011–00000–00001–0000000000010110
8	lw r4,88($0)	10001100000001000000000000001 0110,	100011–00000–00100–0000000000010110
9	add r3,r1,r2	00000000001000100001100000010 0000,	000000–00001–00010–00011–00000–100000
10	and r4,r1,r3	00000000001000110010000000010 0100,	000000–00001–00011–00100–00000–100100
11	sub r5,r1,r4	00000000001001000010100000010 0010,	000000–00001–00100–00101–00000–100010
12	or r6,r1,r4	00000000001001000011000000010 0101,	000000–00001–00100–00110–00000–100101
13	nor r7,r1,r4	00000000001001000011100000010 0111,	000000–00001–00100–00111–00000–100111
14	J 0	00001000000000000000000000000 00000,	000010–00000000000000000000000000,

表格B

行号	指令	寄存器	Next PC	时钟计数值
1	lw r1,80($0)	$r1 = 0x11223344	0x1	0x5
2	lw r2,84($0)	$r2 = 0x00112233	0x2	0xA
3	beq r1,r2,3		0x3	0xD
4	lw r3,80($0)	$r3 = 0x11223344	0x4	0x12

5	beq r1,r3,1		0x6	0x15
6	add r3,r1,r2	$r3 = 0x11223344	该条指令被跳过执行	
7	sw r1,88($0)		0x7	0x19
8	lw r4,88($0)	$r4 = 0x11223344	0x8	0x1E
9	add r3,r1,r2	$r3 = 0x11335577	0x9	0x22
10	and r4,r1,r3	$r4 = 0x11221144	0xA	0x26
11	sub r5,r1,r4	$r5 = 0x00002200	0xB	0x2A
12	or r6,r1,r4	$r6 = 0x11223344	0xC	0x2E
13	nor r7,r1,r4	$r7 = 0xEEDDCCBB	0xD	0x32
14	J 0		0x0	0x35