《计算机组成原理》实验报告

年级、专业、班级		**姓名**	
实验题目	实验二处理器译码实验		
实验时间		**实验地点**	
实验成绩	优秀/良好/中等	**实验性质**	□验证性 ☑设计性 □综合性
教师评价： □算法/实验过程正确；　□源程序/实验内容提交；　□程序结构/实验步骤合理； □实验结果正确；　　□语法、语义正确；　　□报告规范； 其他: 评价教师:			
实验目的 (1)掌握单周期 CPU 控制器的工作原理及其设计方法。 (2)掌握单周期 CPU 各个控制信号的作用和生成过程。 (3)掌握单周期 CPU 执行指令的过程。 (4)掌握取指、译码阶段数据通路执行过程。			

报告完成时间: 2024 年 7 月 21 日

1 实验内容

1. PC。D 触发器结构, 用于储存 PC(一个周期)。需实现 2 个输入, 分别为 *clk, rst*, 分别连接时钟和复位信号; 需实现 2 个输出, 分别为 *pc, inst_ce*, 分别连接指令存储器的 *addra, ena* 端口。其中 *addra* 位数依据 coe 文件中指令数定义;
2. 加法器。用于计算下一条指令地址, 需实现 2 个输入,1 个输出, 输入值分别为当前指令地址 *PC*、*32'h4*;
3. Controller。其中包含两部分:
   (a). main_decoder。负责判断指令类型, 并生成相应的控制信号。需实现 1 个输入, 为指令 inst 的高 6 位 *op*, 输出分为 2 部分,控制信号有多个, 可作为多个输出, 也作为一个多位输出, 具体参照参考指导书进行设计;*aluop*, 传输至 alu_decoder, 使 alu_decoder 配合 *inst* 低 6 位 *funct*, 进行 ALU 模块控制信号的译码。
   (b). alu_decoder。负责 ALU 模块控制信号的译码。需实现 2 个输入,1 个输出, 输入分别为 *funct, aluop*; 输出位 *alucontrol* 信号。
   (c). 除上述两个组件, 需设计 controller 文件调用两个 decoder,对应实现 *op*,*funct* 输入信号, 并传入调用模块; 对应实现控制信号及 *alucontrol*, 并连接至调用模块相应端口。
4. 指令存储器。使用 Block Memory Generator IP 构造。(参考指导书)
   注意: Basic 中 Generate address interface with 32 bits 选项不选中;PortA Options 中 Enable Port Type 选择为 Use ENA Pin
5. 时钟分频器。将板载 100Mhz 频率降低为 1hz, 连接 PC、指令存储器时钟信号 clk。(参考数字逻辑实验)
   注意:Xilinx Clocking Wizard IP 可分的最低频率为 4.687Mhz, 因而只能使用自实现分频模块进行分频

2 实验设计

2.1 控制器 (Controller)

2.1.1 功能描述

Controller 模块接收指令的操作码 op 和功能码 Funct 字段, 处理并返回 CPU 中的控制信号 RegWrite,RegDst,AluSrc,Branch,MemWrite,MemtoReg,MemRead,Jump 和 ALUControl。

2.1.2 接口定义

表 1: Controller 接口定义

信号名	方向	位宽	功能描述
op	Input	5-bit	指令的操作码字段
Funct	Input	5-bit	指令的功能码字段
RegWrite	Input	5-bit	数据写入由写入寄存器输入端口指定的寄存器
RegDst	Input	5-bit	写入寄存器时, 目标寄存器的编号来自 rt/rd 字段
AluSrc	Input	5-bit	第二个 ALU 操作数来自第二个寄存器堆的输出/是指令第 16 位的符号拓展
Branch	Input	5-bit	指令不是/是分支指令
MemWrite	Input	5-bit	将写入数据输入端的数据写入地址输入端指定的存储单元中
MemtoReg	Input	5-bit	写入寄存器的数据来自 ALU/数据存储器
MemRead	Input	5-bit	输入地址对应的数据存储器的内容输出到读数据输出端口
Jump	Input	5-bit	指令不是/是跳转指令
ALUControl	Input	5-bit	ALU 的控制信号

2.1.3 逻辑控制

Controller 模块将接受到的操作码 op 交由 Maindec 模块生成各控制信号, 然后再将控制信号 ALUOp 和功能码 Funct 一并交由 Aludec 模块生成 ALU 控制信号 ALUControl; 译码规则见下表:

表 2: Maindec 译码表

Instruction	op[5:0]	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	MemRead	Jump	ALUOp
R-type	000000	1	1	0	0	0	0	0	0	10
lw	000000	1	0	1	0	0	1	1	0	00
sw	000000	0	X	1	0	1	X	0	0	00
beq	000000	0	X	0	1	0	X	0	0	01
addi	000000	1	0	1	0	0	0	0	0	00
j	000000	0	X	X	X	0	X	0	1	XX

表 3: Aludec 译码表

Opcode	ALUOp	Operation	Funct	ALU Function	ALUControl
lw	00	Loar word	XXXXXX	Add	010
sw	00	Store word	XXXXXX	Add	010
beq	01	Branch if equal	XXXXXX	Sub	110
R-type	10	Add	100000	Add	010
R-type	10	Sub	100010	Sub	110
R-type	10	And	100100	And	000
R-type	10	Or	100101	Or	001
R-type	10	Set on less than	101010	SLT	111

2.2 存储器 (Block Memory)

2.2.1 类型选择

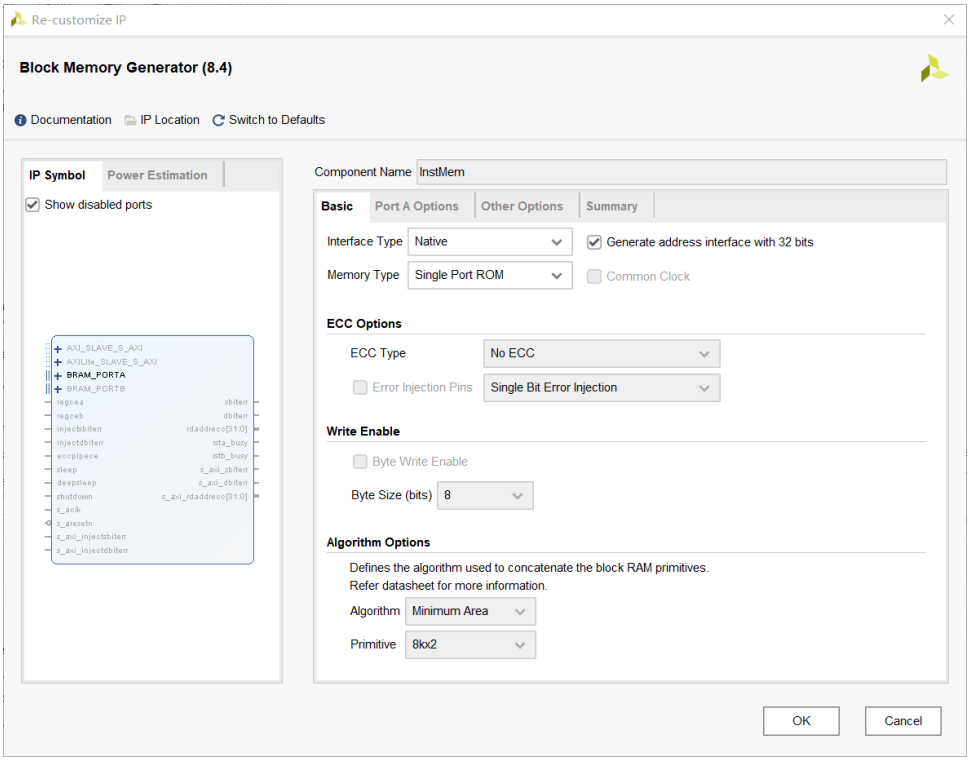

图 1: Basic

2.2.2 参数设置

指令存储器 Instruction Memory(InstMem) 各参数设置如下图所示:

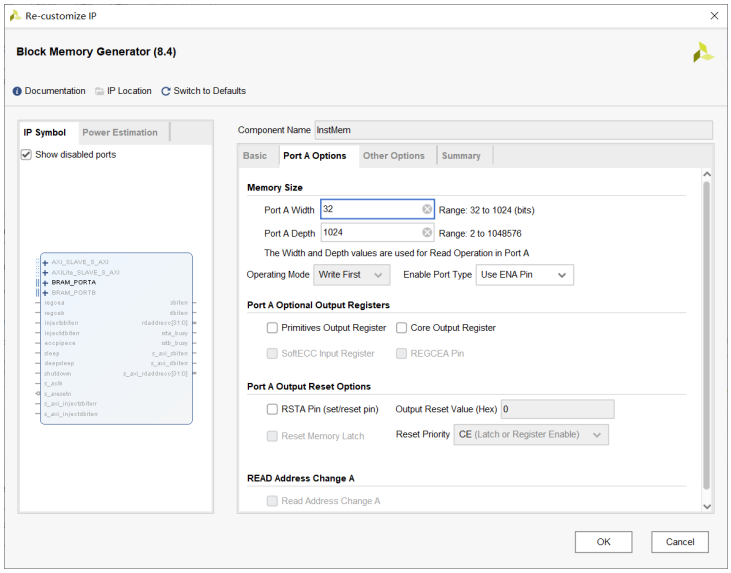

图 2: Port A Options

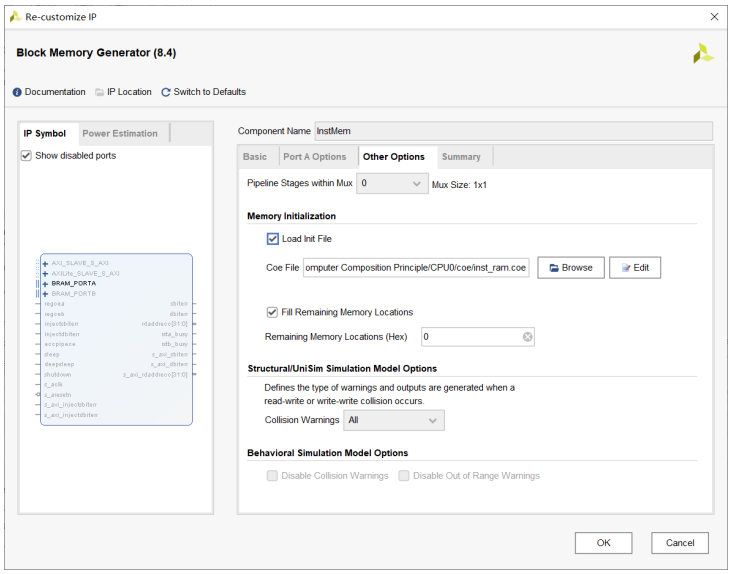

图 3: Other Options

3 实验过程记录

3.1 Controller

3.1.1 问题 1:ALUOp 信号处于未知状态

问题描述:ALUOp 位宽实际应为 2-bit,Maindec 中定义的 ALUOp 位宽为 1-bit。
解决方案:修改 Maindec 中定义的 ALUOp 位宽为 2-bit。

3.2 PC

3.2.1 问题 1: 丢失指令寄存器中的第一条指令

问题描述:没有为 PC 模块中的 reg 型输出指令地址 pc 赋初值 0, 导致第一次读取时指令地址无效。
解决方案:为 PC 模块中的 reg 型输出指令地址 pc 赋初值 0。

3.3 InstMem

3.3.1 问题 1: 指令存储器无有效输出

问题描述:指令存储器模块的接口顺序与图示不一致, 导致模块间端口连接错位。
解决方案:修改实例化储存器模块的端口顺序。

4 实验结果及分析

4.1 仿真图

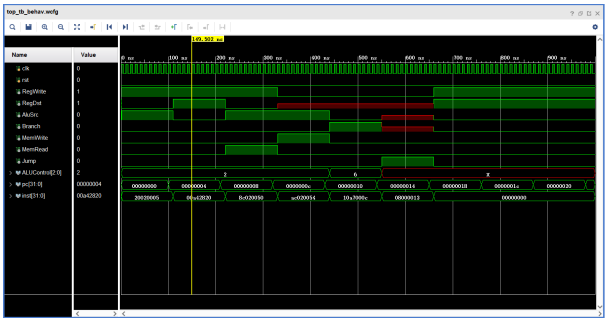

图 4: 仿真图

4.2 控制台输出

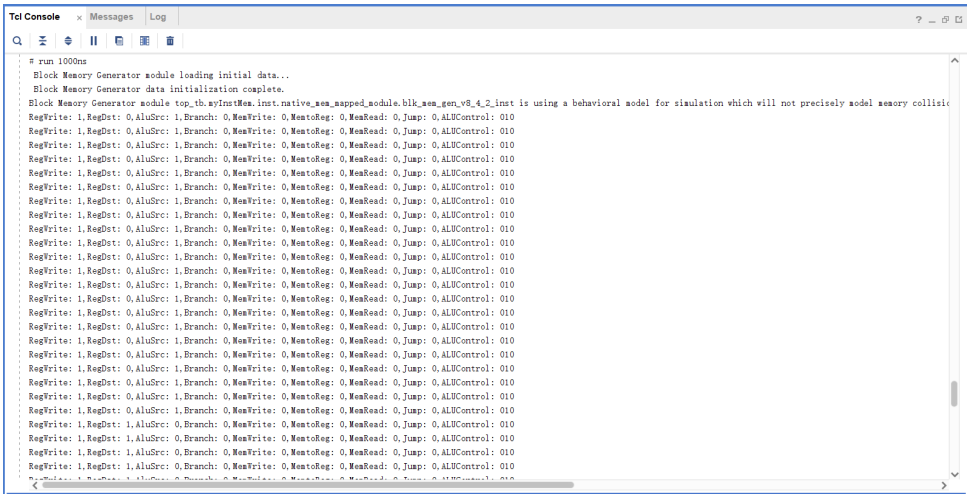

图 5: 控制台输出

4.3 开发板验证

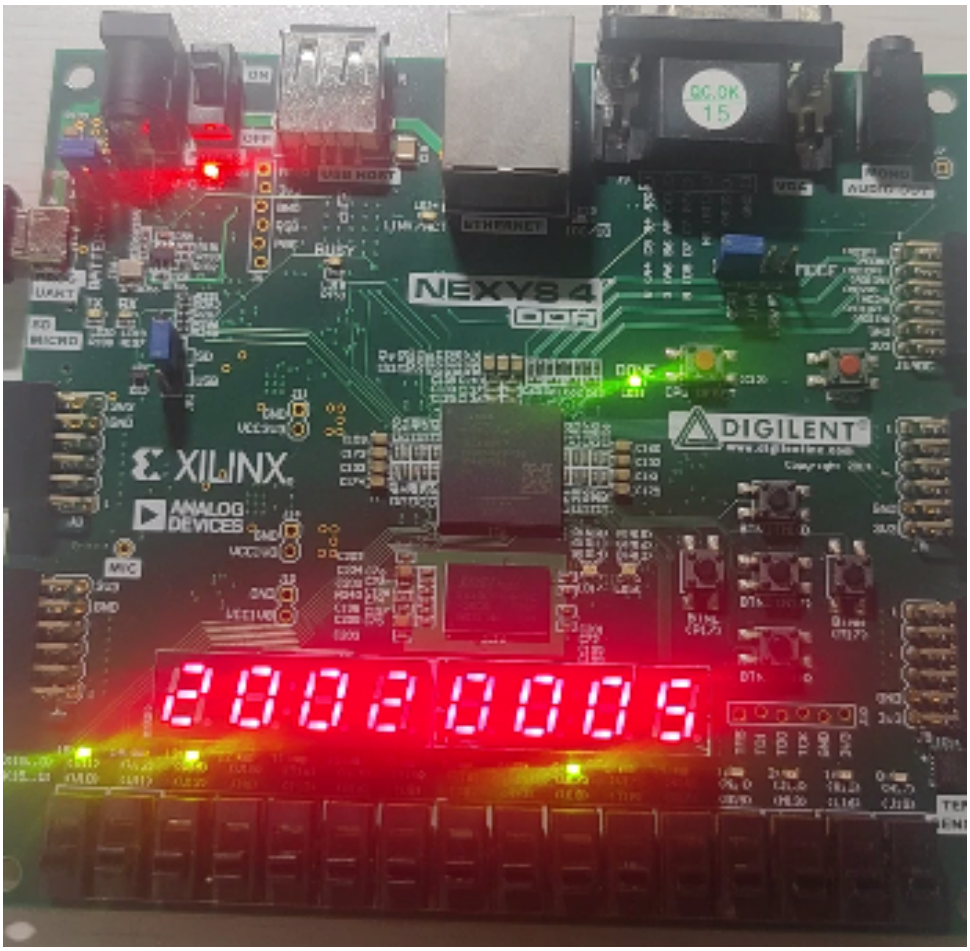

图 6: 指令 1

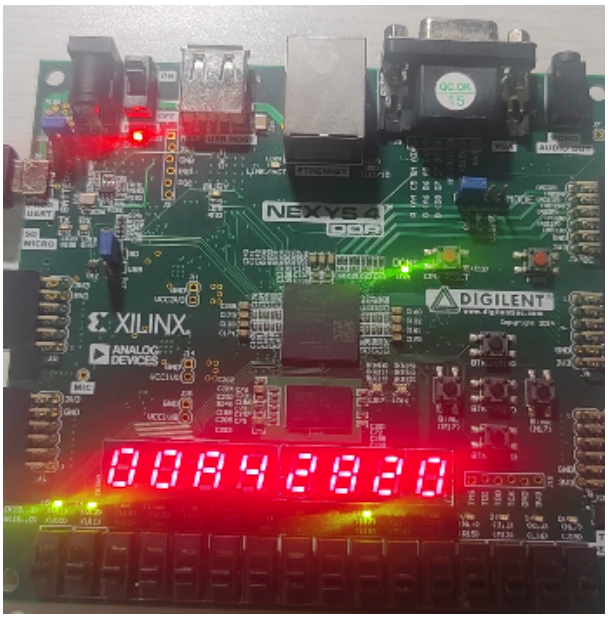
NEXYS 4
DDR
XILINX
ANALOG DEVICES
DIGILENT
00A42820

图 7: 指令 2

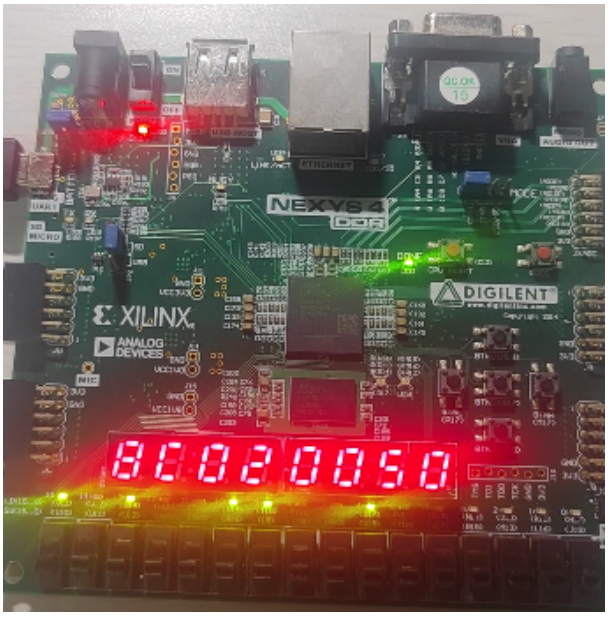
NEXYS 4
DDR
XILINX
ANALOG DEVICES
DIGILENT
BC020050

图 8: 指令 3

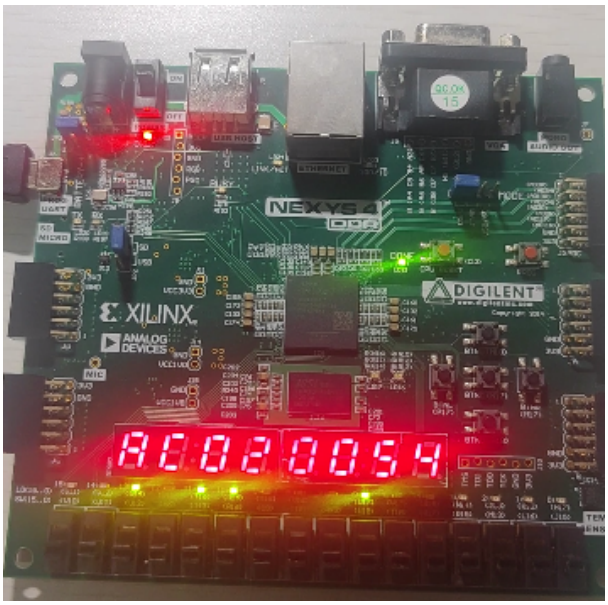

图 9: 指令 4

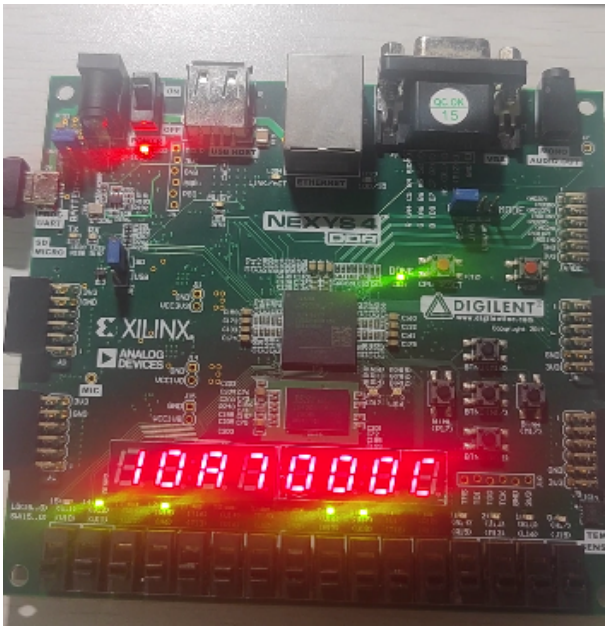

图 10: 指令 5

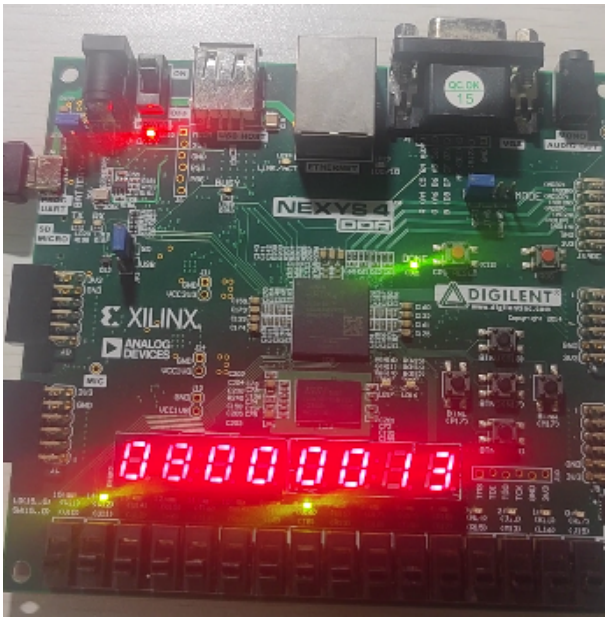

图 11: 指令 6

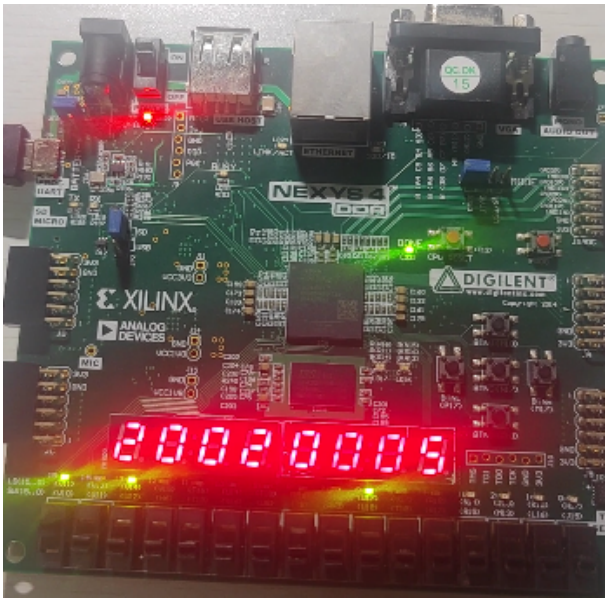

图 12: 复位

A Top 代码

```
module Top(clk,rst,RegWrite,RegDst,AluSrc,Branch,MemWrite,MemtoReg,MemRead,Jump,
    ALUControl,enable,segs);
    input clk;
    input rst;
    output RegWrite;
    output RegDst;
    output AluSrc;
    output Branch;
    output MemWrite;
    output MemtoReg;
    output MemRead;
    output Jump;
    output [2:0] ALUControl;

    output [7:0] enable;
    output [6:0] segs;

    wire div;
    wire [31:0] pc;
    wire inst_en;
    wire [31:0] inst;

    Clock_1HZ myClock_1HZ(clk,div);
    PC myPC(div,rst,pc,inst_en);
    InstMem myInstMem(.addra(pc),.clka(clk),.douta(inst),.ena(inst_en));
    Controller myController(inst[31:26],inst[5:0],RegWrite,RegDst,AluSrc,Branch,
        MemWrite,MemtoReg,MemRead,Jump,ALUControl);
    Display myDisplay(clk,1,inst,enable,segs);
endmodule
```

B Clock_1HZ 代码

```
module Clock_1HZ(clk,clk_1hz);
    //parameter MAX_CNT = 32'd10;
    parameter MAX_CNT = 32'd1_0000_0000;

    input clk;
    output reg clk_1hz;

    integer cnt=0;

    always @(posedge clk)
    begin
        if(cnt==MAX_CNT) begin
            cnt<=0;
```

```
            clk_1hz<=1'b1;
        end
        else begin
            cnt<=cnt+1'b1;
            clk_1hz<=1'b0;
        end
    end
endmodule
```

C PC 代码

```
module PC(clk,rst,pc,inst_ce);
    input clk;
    input rst;
    output reg [31:0] pc=32'b0;
    output inst_ce;

    assign inst_ce=1'b1;

    always@(posedge clk or posedge rst)
    begin
        if(rst) begin
            pc<=32'b0;
        end
        else begin
            pc<=pc+4;
        end
    end
endmodule
```

D Controller 代码

```
module Controller(op,Funct,RegWrite,RegDst,AluSrc,Branch,MemWrite,MemtoReg,MemRead,
    Jump,ALUControl);
    input [5:0] op;
    input [5:0] Funct;
    output RegWrite;
    output RegDst;
    output AluSrc;
    output Branch;
    output MemWrite;
    output MemtoReg;
    output MemRead;
    output Jump;
    output [2:0] ALUControl;
```

```
    wire [1:0] ALUOp;

    Maindec myMaindec(op,RegWrite,RegDst,AluSrc,Branch,MemWrite,MemtoReg,MemRead,
        Jump,ALUOp);
    Aludec myAludec(ALUOp,Funct,ALUControl);
endmodule
```

E Maindec 代码

```
module Maindec(op,RegWrite,RegDst,AluSrc,Branch,MemWrite,MemtoReg,MemRead,Jump,
    ALUOp);
    input [5:0] op;
    output RegWrite;
    output RegDst;
    output AluSrc;
    output Branch;
    output MemWrite;
    output MemtoReg;
    output MemRead;
    output Jump;
    output [1:0] ALUOp;

assign {RegWrite,RegDst,AluSrc,Branch,MemWrite,MemtoReg,MemRead,Jump,ALUOp}=
        (op==6'b00_0000)?10'b11_0000_0010://R-type
        (op==6'b10_0011)?10'b10_1001_1000://lw
        (op==6'b10_1011)?10'b0X_101X_0000://sw
        (op==6'b00_0100)?10'b0X_010X_0001://beq
        (op==6'b00_1000)?10'b10_1000_0000://addi
        (op==6'b00_0010)?10'b0X_XX0X_01XX:10'bXX_XXXX_XXXX;//j
endmodule
endmodule
```

F Aludec 代码

```
module Aludec(ALUOp,Funct,ALUControl);
    input [1:0] ALUOp;
    input [5:0] Funct;
    output [2:0] ALUControl;

    assign ALUControl=(ALUOp==2'b00)?3'b010://Add
                      (ALUOp==2'b01)?3'b110://Sub
                      (ALUOp==2'b10&&Funct==6'b10_0000)?3'b010://Add
                      (ALUOp==2'b10&&Funct==6'b10_0010)?3'b110://Sub
                      (ALUOp==2'b10&&Funct==6'b10_0100)?3'b000://And
```

```
                    (ALUOp==2'b10&&Funct==6'b10_0101)?3'b001://Or
                    (ALUOp==2'b10&&Funct==6'b10_1010)?3'b111:3'bXXX;//SLT
endmodule
```

G Display 代码

```
module Display(clk,rst,display,enable,segs);
    parameter MAX_CNT=32'h20000;

    input clk;
    input rst;
    input [31:0] display;
    output reg [7:0] enable;
    output reg [6:0] segs;

    reg div=1'b0;
    reg [2:0] cnt8=3'b0;
    reg [3:0] digits;

    integer cnt=0;

    always@(posedge clk or negedge rst)
    begin
        if(!rst)
        begin
            div=1'b0;
            cnt=0;
        end
        else
        begin
            if(cnt==MAX_CNT)
            begin
                div=1'b1;
                cnt=0;
            end
            else
            begin
                div=1'b0;
                cnt=cnt+1;
            end
        end
    end

    always@(posedge div or negedge rst)
    begin
        if(!rst)
        begin
```

```
            cnt8=3'b0;
            enable=8'b1111_1111;
        end
        else
        begin
             cnt8=cnt8+1'b1;
             case(cnt8)
                3'b000:enable=8'b1111_1110;
                3'b001:enable=8'b1111_1101;
                3'b010:enable=8'b1111_1011;
                3'b011:enable=8'b1111_0111;
                3'b100:enable=8'b1110_1111;
                3'b101:enable=8'b1101_1111;
                3'b110:enable=8'b1011_1111;
                3'b111:enable=8'b0111_1111;
                default:enable=8'b1111_1111;
             endcase
        end
end

always@(cnt8 or display)
begin
        case(cnt8)
        3'b000:digits=display[3:0];
        3'b001:digits=display[7:4];
        3'b010:digits=display[11:8];
        3'b011:digits=display[15:12];
        3'b100:digits=display[19:16];
        3'b101:digits=display[23:20];
        3'b110:digits=display[27:24];
        3'b111:digits=display[31:28];
        default:digits=4'b0;
        endcase
end

always@(digits)
begin
        case(digits)
            4'h0:segs = 7'b000_0001;//0
            4'h1:segs = 7'b100_1111;//1
            4'h2:segs = 7'b001_0010;//2
            4'h3:segs = 7'b000_0110;//3
            4'h4:segs = 7'b100_1100;//4
            4'h5:segs = 7'b010_0100;//5
            4'h6:segs = 7'b010_0000;//6
            4'h7:segs = 7'b000_1111;//7
            4'h8:segs = 7'b000_0000;//8
            4'h9:segs = 7'b000_0100;//9
            4'ha:segs = 7'b000_1000;//a
```

```
            4'hb:segs = 7'b110_0000;//b
            4'hc:segs = 7'b011_0001;//c
            4'hd:segs = 7'b100_0010;//d
            4'he:segs = 7'b011_0000;//e
            4'hf:segs = 7'b011_1000;//f
            default:segs = 7'b111_1111;
        endcase
    end
endmodule
```