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

这一节主要描述各个模块的功能、接口、逻辑控制方法(状态机控制方法)等。

2.1 控制器 (Controller)

2.1.1 功能描述

控制器输入指令 IR 的 opcode 和 funct 码，输出 memtoreg、memwrite、pcsrc、alusrc、regdst、regwrite、branch、jump 和 alucontrol 信号。

32 位 MIPS 指令在不同类型指令中分别有不同结构。但 IR[31:26] 表示的 opcode，IR[5:0] 表示的 funct, 为译码阶段明确指令控制信号的主要字段。根据实验要求，实现 R、lw、sw、beq、addi、j 型指令的译码。

由于 Controller 包含 Main decoder 和 ALU decoder，因此分模块编写，Controller 顶层调用。

Main decoder 输入 IR[31:26] 作为 opcode，输出 memtoreg、memwrite、pcsrc、alusrc、regdst、regwrite、branch、jump 和 aluop，ALU decoder 输入 aluop 和 IR[5:0] 作为 funct，输出 alu control

2.1.2 接口定义

信号名	方向	位宽	功能描述
op	input	6-bit	opcode 码，即 IR[31:26]
funct	input	6-bit	funct 码，即 IR[5:0]
aluctrl	output	3-bit	ALU 控制信号
memtoreg	output	1-bit	用于标识写回 register 的数据来自 ALU 计算结果 (R 型) or 存储器读取的数据 (lw)
memwrite	output	1-bit	用于标识是否需要写数据寄存器 (sw)
pcsrc	output	1-bit	用于标识下一个地址是否为 PC+4
alusrc	output	1-bit	传入 ALU B 端口的值是 32 位立即数 (addi) or 寄存器堆读取的数据（R/sw/beq）
regdst	output	1-bit	写入存储器堆的地址是 rt(lw 时赋 0)or rd(R 型赋 1)（配合 regwrite 信号）
regwrite	output	1-bit	是否需要写寄存器堆（R 型、lw 需要写回寄存器堆）
branch	output	1-bit	是否满足 branch 指令的跳转条件
jump	output	1-bit	是否为 jump 指令
alucontrol	output	1-bit	ALU 控制信号，只要求实现 R 型指令的 add、sub、and、or、slt

表 1: Controller 接口定义

上述信号除了 alucontrol 由 ALU decoder 负责译码输出，其余均在 Main decoder 中输出。中间信号 aluop 由 Main decoder 输出并输入 ALU decoder。

2.1.3 逻辑控制

Controller 包含 Main decoder 和 ALU decoder，分模块编写 Main decoder 和 ALU decoder，然后由 Controller 顶层调用。Main decoder 输入 IR[31:26] 作为 opcode，输出 memtoreg、memwrite、pcsrc、alusrc、regdst、regwrite、branch、jump 和 aluop，ALU decoder 接 IR[5:0] 作为 funct 和受 Main decoder 输出的 aluop，输出 alu control。

对于 Main decoder，根据输入的 opcode 判断指令类型，采用 case 语句译码得到各元件的控制信号。对应控制信号译码如下：

对于 ALU decoder，采用嵌套 case 语句，根据输入 funct 和 aluop，输出对应的 alu control。需

instruction	op[5:0]	regwrite	regdst	alusrc	branch	memwrite	memtoreg	aluop[1:0]
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
addi	001000	1	0	1	0	0	0	00
j	000010	0	X	X	X	0	X	XX

表 2: Main decoder 控制信号译码

要使用 ALU 的指令有 lw、sw、beq 和 R-type。输入输出对应如下：

opcode	aluop	operation	funct	alu function	alu control
lw	00	Load word	XXXXXX	Add	010
sw	00	Store word	XXXXXX	Add	010
beq	01	Branch equal	XXXXXX	Subtract	110
R-type	10	Add	100000	Add	010
R-type	10	Subtract	100010	Subtract	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

表 3: ALU decoder 控制信号译码

2.2 存储器 (Block Memory)

2.2.1 类型选择

存储器类型选择 Single Port ROM，并且注意勾选 Generate address interface with 32bits，将地址长度设置为 32 位。具体参数如图 1。

2.2.2 参数设置

端口宽度选择 32bits，深度随意，本实验中大于 6 即可（coe 文件中有 6 条指令）。此处选择深度为 1024。并且需要使能端和重置信号。具体端口参数选项如图 2。其他选项需加载 coe 文件，如图 3。参数选择完成后生成 ROM 如图 4。

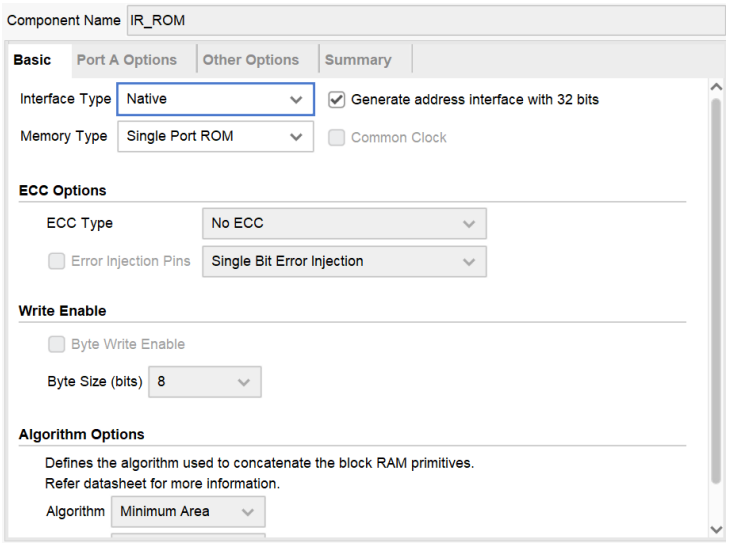

图 1: 存储器类型选择

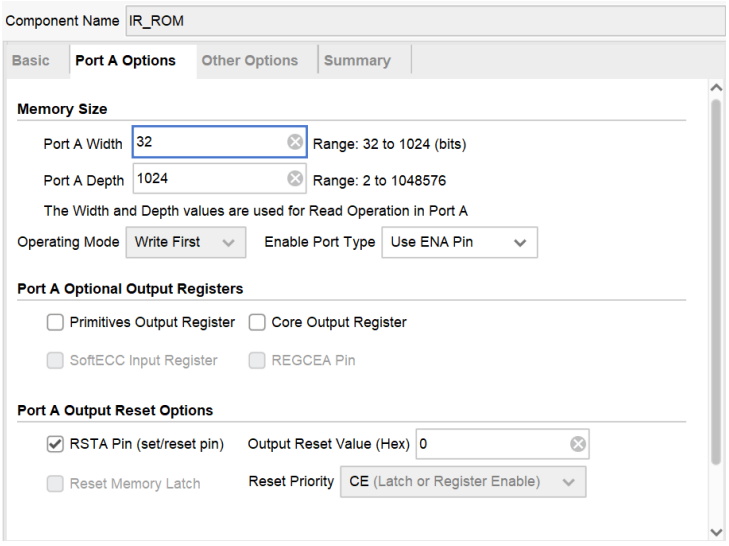

图 2: ROM 参数设置

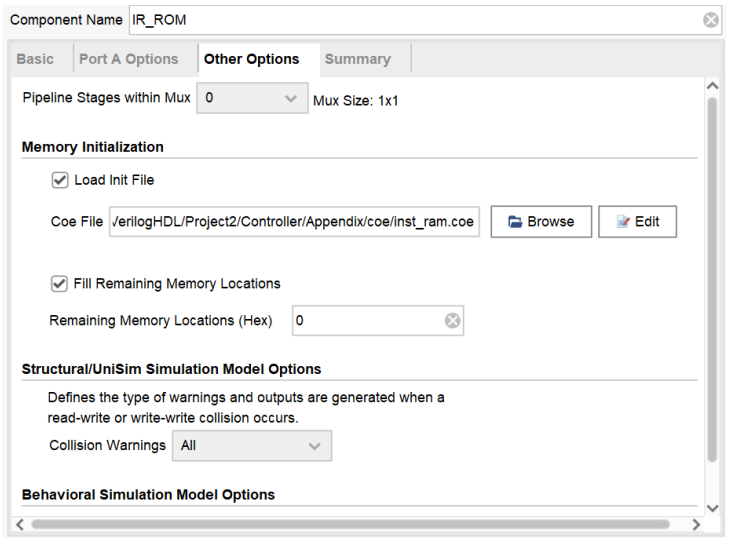

图 3: ROM 参数设置

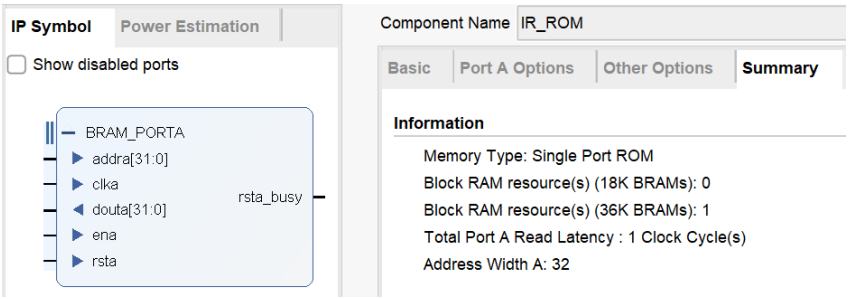

图 4: 生成 ROM

3 实验过程记录

3.1 问题 1:top 文件输出信号没有分配引脚

问题描述: 根据项目所给管脚分布文件进行比特流生成，发现 memtoreg、memwrite、alusrc、regdst、regwrite、branch、jump、aluctrl 没有分配引脚，生成比特流失败。

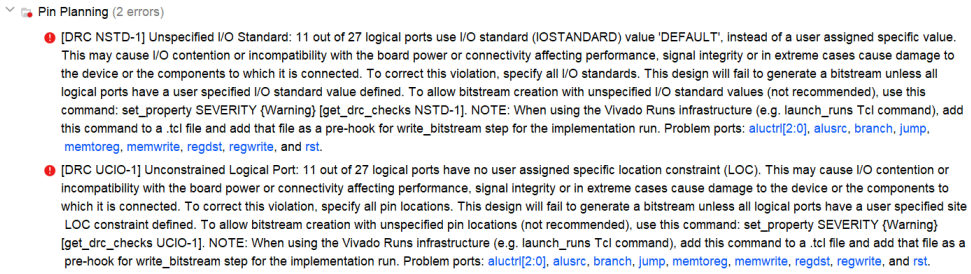

图 5: 问题 1 报错

解决方案: 将 memtoreg、memwrite、alusrc、regdst、regwrite、branch、jump、aluctrl[2]、aluctrl[1]、aluctrl[0] 依次分配给 N14、J13、U17、K15、H17、V17、R18、T15、V16、U16。然后比特流生成成功。

3.2 问题 2: 开发板无法显示 Z 信号

问题描述: 在 Controller 模块一些指令对应某些控制信号为空，一开始考虑输出 X 信号，但 X 信号只能用于仿真，真实电路中并不存在。于是采用 Z 信号，但在生成比特流过程中失败。

解决方案: 将 Z 信号改为 0，即高电频才在开发板上亮 LED 灯。

3.3 问题 3:ROM 不接受 rst 信号

问题描述: 在顶层模块中调用 ROM 时无 rst 输入位置。

解决方案: 重建 IP 核，勾选 RSTA Pin 选项。

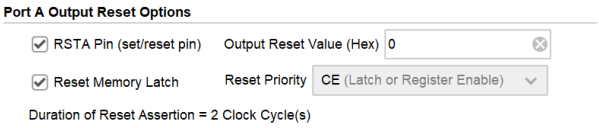

图 6: 问题 3 解决

3.4 问题 4: 重置信号不起作用

问题描述:一开始在设计分频模块时将 rst 信号加入,同时还将 rst 接入 pc,使得 addr 生成紊乱。在上板后发现拨动 rst 信号并不会重新读取。

解决方案:在分频模块只负责分频,不计入 rst 的重置功能。

3.5 问题 5:coe 文件中第一条指令被吞掉

问题描述:仿真过程中读出的指令从第二条 8'h00a42820 开始,第一条指令 8'h2002_0005 被吞掉。

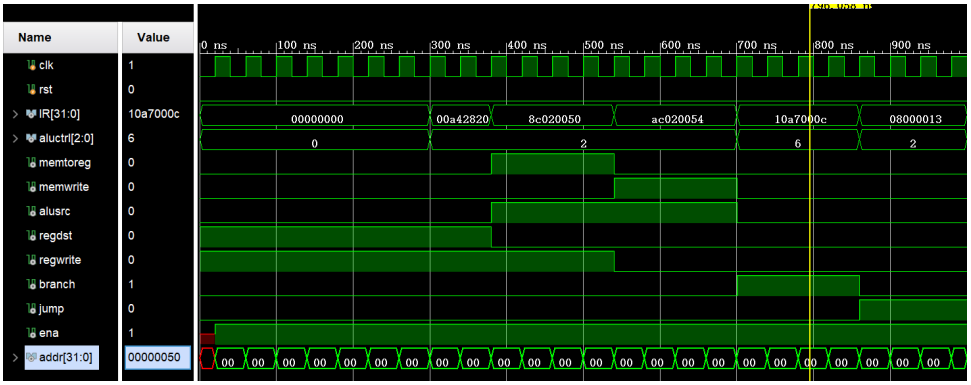

图 7: 问题 5 仿真错误

解决方案:在 pc 模块中对 addr 赋初值 0 后,问题成功解决。

3.6 问题 6: 读出指令有缺失

问题描述:编写 PC 模块时,为了模仿计算机上地址增减,在每个 clk 来临时 PC+4。但仿真发现读出的指令有缺失,中间的指令缺失。

解决方案: pc 输出的地址不能直接接入 ROM, ROM 接受字节地址, 因此每个地址间隔为 1 而不是 4,所以 pc 需要右移 2 位再输入 ROM。

3.7 问题 7: 开发板指令显示不全

问题描述: 生成比特流后,开发板上的 IR 指令显示在数码管上只能显示最低位,其他位不显示。

解决方案: 时钟频率不一致,分频后的时钟只需要接入 pc 模块和 ROM 存储器,display 模块内含分频功能,因此接入的时钟信号应该为 100MHz 的原始时钟信号而不是分频后 1Hz 的信号。

3.8 问题 8: 上板后每条指令间周期过长

问题描述: 10^8(10)=0101_1111_0101_1110_0001_0000_0000(2),约为 2^27。因此分频时将 cnt[27] 赋给 clk_division,但上板后发现指令间隔超过 1s。

解决方案: 1s 的时钟间隔应该是 clk_division 从 0 到 1 再到 0 的过程,因此分频计数应该除以 2,即赋 cnt[26]。

3.9 问题 9: 上板子出现抖动

问题描述: 实验最后上板子阶段偶尔出现抖动,指令显示不清。

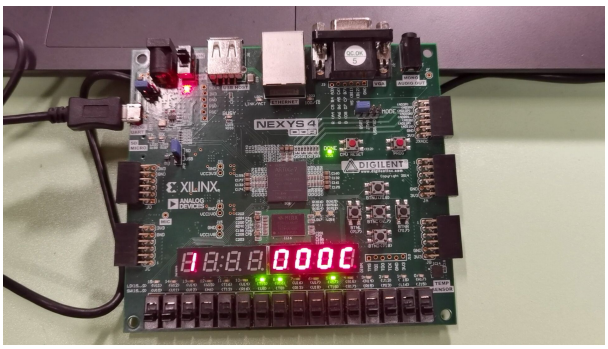

图 8: 问题 9 抖动

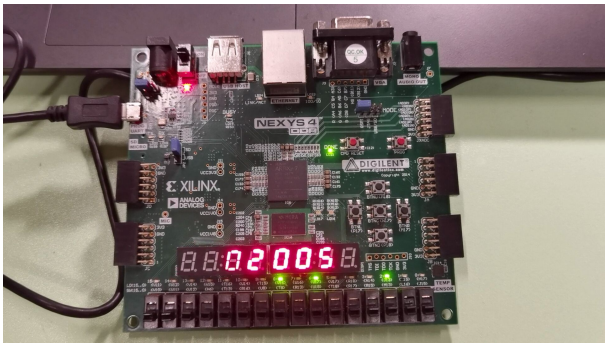

图 9: 问题 9 抖动

解决方案: 重置信号 rst 时将开关拨到底即可。

3.10 问题 10: 第一条指令和第二条指令间被插入 4 个周期的 8'h0000_0000

问题描述: 仿真过程中读出的第一条指令 8'h2002_0005 到第二条指令 8'h00a4_2820 之间被插入了 4 个周期的 8'h0000_0000。

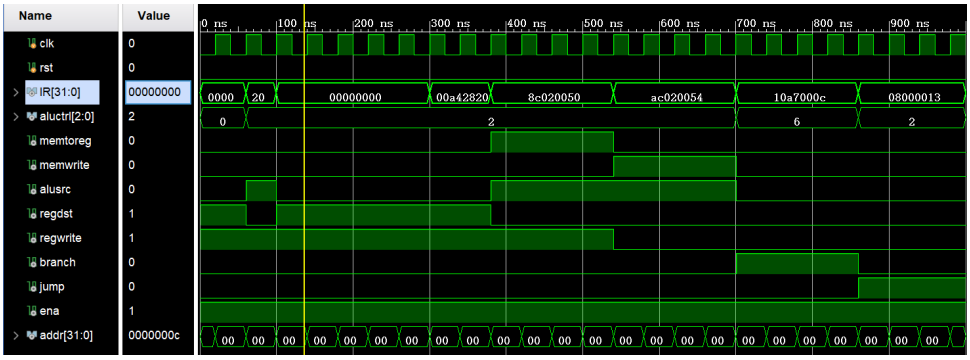

图 10: 问题 10 仿真错误

解决方案: debug4 天, 问了很多人, 后来发现是 ROM 损毁, 将项目文件放在另一台电脑上运行, 问题解决。也可以用 Distributed RAM 替代。

4 实验结果及分析

4.1 仿真图

仿真每隔 100 个时钟周期取出一条指令, 并将指令和控制信号在控制台输出。如图 11 中, 将 ROM 中 6 条指令全部取出。

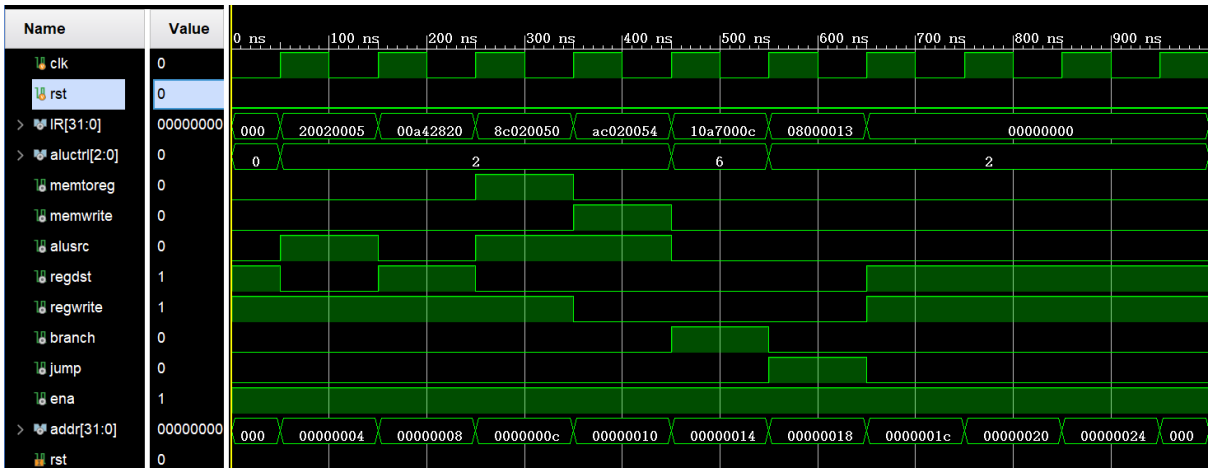

图 11: 仿真图

4.2 控制台输出

```
# run 1000ns
WARNING: This core is supplied with a behavioral model. To model cycle-accurate behavior you must run timing simulation.
instruction:00000000 aluctrl:000 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
instruction:20020005 aluctrl:010 memtoreg:0 memwrite:0 alusrc:1 regdst:0 regwrite:1 branch:0 jump:0
instruction:20020005 aluctrl:010 memtoreg:0 memwrite:0 alusrc:1 regdst:0 regwrite:1 branch:0 jump:0
instruction:00a42820 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
instruction:00a42820 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
instruction:8c020050 aluctrl:010 memtoreg:1 memwrite:0 alusrc:1 regdst:0 regwrite:1 branch:0 jump:0
instruction:8c020050 aluctrl:010 memtoreg:1 memwrite:0 alusrc:1 regdst:0 regwrite:1 branch:0 jump:0
instruction:ac020054 aluctrl:010 memtoreg:0 memwrite:1 alusrc:1 regdst:0 regwrite:0 branch:0 jump:0
instruction:ac020054 aluctrl:010 memtoreg:0 memwrite:1 alusrc:1 regdst:0 regwrite:0 branch:0 jump:0
instruction:10a7000c aluctrl:110 memtoreg:0 memwrite:0 alusrc:0 regdst:0 regwrite:0 branch:1 jump:0
instruction:10a7000c aluctrl:110 memtoreg:0 memwrite:0 alusrc:0 regdst:0 regwrite:0 branch:1 jump:0
instruction:08000013 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:0 regwrite:0 branch:0 jump:1
instruction:08000013 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:0 regwrite:0 branch:0 jump:1
instruction:00000000 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
instruction:00000000 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
instruction:00000000 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
instruction:00000000 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
instruction:00000000 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
instruction:00000000 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
instruction:00000000 aluctrl:010 memtoreg:0 memwrite:0 alusrc:0 regdst:1 regwrite:1 branch:0 jump:0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'top_sim_behav' loaded.
```

图 12: 控制台输出

4.3 电路图

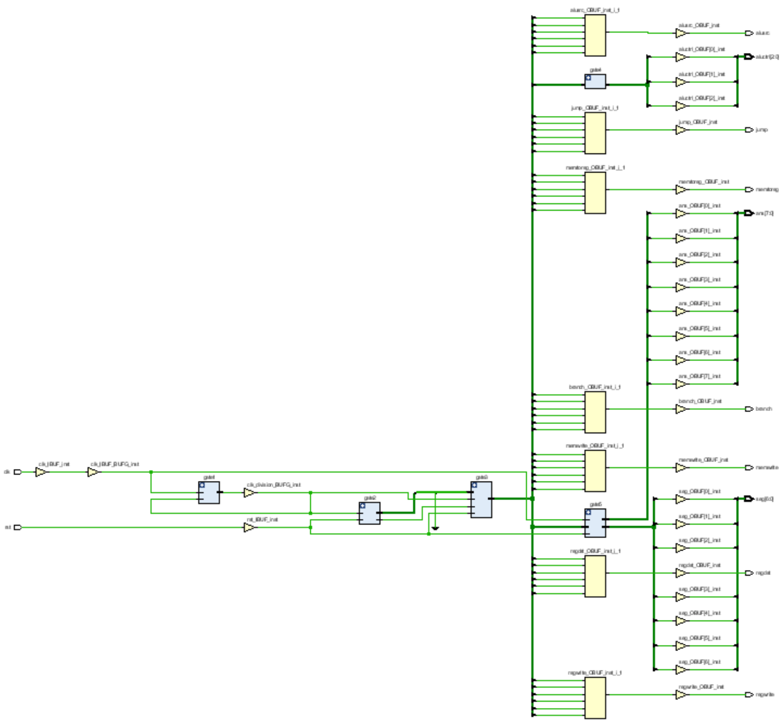

图 13: 电路图

4.4 上板图

上板实验，初始状态如图 14，根据 clk 每个周期读取一条指令，一共 6 条指令，如图 15，读取结束后回到初始状态，重置后如图 16。

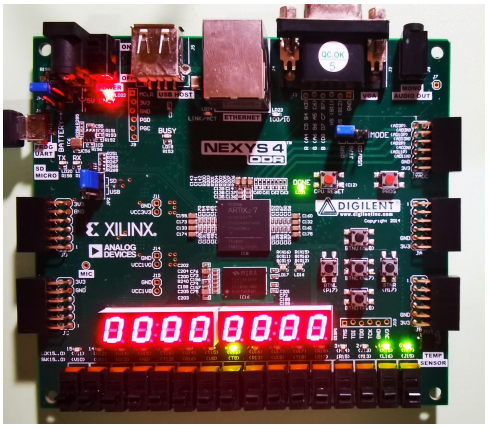

图 14: 初始状态

图 15: 读取指令

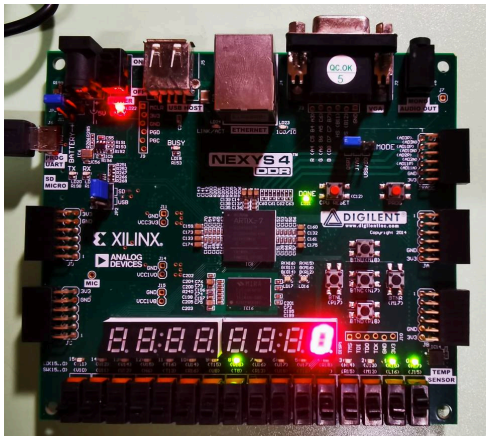

图 16: 重置状态

5 代码附录

A 顶层模块

```
module top(
    input wire clk,rst,
    output wire[2:0] aluctrl,
    output wire memtoreg,memwrite,alusrc,regdst,regwrite,branch,jump,
    output wire[6:0] seg,
    output wire[7:0] ans
);
    wire clk_division;
    clk_div gate1(clk,clk_division);

    wire ena;
    wire[31:0] addr;                    //32bits指令的地址（0~1023）
    pc gate2(clk_division,rst,addr,ena);

    wire[31:0] IR;                      //32bits指令
    DisM gate3(addr>>2,clk_division,ena,rst,IR);  //传入ROM的是字节地址，因此要
        右移2位

    controller gate4(IR[31:26],IR[5:0],aluctrl,memtoreg,memwrite,alusrc,regdst,
        regwrite,branch,jump);

    display gate5(clk,rst,IR,seg,ans);  //display中已经分频

endmodule
```

B 时钟分频模块

```
module clk_div(
```

```
    input clk,
    output clk_division
);
    reg[27:0] cnt=0;
    assign clk_division=cnt[26];    //26th从0~1~0才是2^26
    always@(posedge clk)begin
        cnt=cnt+1;
    end

endmodule
```

C PC 模块

```
module pc(
    input clk,rst,
    output reg[31:0] addr=0,     //连接指令寄存器的输入地址端口
    output wire ena              //连接指令寄存器的使能端口
);
    reg [31:0] cnt=0;            //地址计数，不断+4
    assign ena=(addr<1024)?1:0; //ROM设置的端口Depth为1024，因此>1024的地址为无
        效地址
//    assign ena =1;
    always@(posedge clk)begin
        if(rst)begin
            cnt=0;
            addr=cnt;
        end
        else begin
           cnt=cnt+4;
           addr=cnt;
        end
    end
endmodule
```

D Controller 模块

```
module controller(
    input wire[5:0] op,
    input wire[5:0] funct,
    output wire[2:0] aluctrl,
    output wire memtoreg,memwrite,alusrc,regdst,regwrite,branch,jump
);
    wire[1:0] aluop;
    maindec gate1(op,aluop,memtoreg,memwrite,alusrc,regdst,regwrite,branch,jump
        );
```

```
    aludec gate2(aluop,funct,aluctrl);

endmodule
```

E Main decoder 模块

```
module maindec(
    input [5:0] op,                //输入opcode
    output reg[1:0] aluop,        //输出alu控制信号
    output reg memtoreg,memwrite,alusrc,regdst,regwrite,branch,jump
);
    always@(*)begin
        case(op)
            6'b000000:begin      //R型
                {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump}=7'
                    b1100000;  //顺序按指导书表5
                aluop=2'b10;
            end
            6'b100011:begin      //lw
                {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump}=7'
                    b1010010;
                aluop=2'b00;
            end
            6'b101011:begin      //sw
//                {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump}=7'
    b0z101z0;
                {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump}=7'
                    b0010100;
                aluop=2'b00;
            end
            6'b000100:begin      //beq
                {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump}=7'
                    b0001000;
                aluop=2'b01;
            end
            6'b001000:begin      //I型(addi)
                {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump}=7'
                    b1010000;
                aluop=2'b00;
            end
            6'b000010:begin      //jump
                {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump}=7'
                    b0000001;
                aluop=2'b00;
            end
            default:begin
                {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump}='b0;
```

```
                aluop=2'b00;
            end
        endcase
    end
endmodule
```

F ALU decoder 模块

```
module aludec(
    input wire[1:0] aluop,
    input wire[5:0] funct,
    output reg [2:0] aluctrl
);
    always@(*)begin
        case(aluop)
            2'b00:aluctrl=010;              //lw、sw
            2'b01:aluctrl=110;              //beq
            2'b10:begin                     //R型指令
                case(funct)
                    6'b100000:aluctrl=010;  //add
                    6'b100010:aluctrl=110;  //sub
                    6'b100100:aluctrl=000;  //and
                    6'b100101:aluctrl=001;  //or
                    6'b101010:aluctrl=111;  //slt
                endcase
            end
            default:aluctrl='b0;            //写成z不好上板子
        endcase
    end
endmodule
```

G 仿真文件

```
module top_sim();
    reg clk=0,rst=0;
    wire[31:0] IR;
    wire[2:0] aluctrl;
    wire memtoreg,memwrite,alusrc,regdst,regwrite,branch,jump;

    wire ena;
    wire[31:0] addr;
    pc gate2(clk,rst,addr,ena);
    DisM gate3(addr>>2,clk,ena,rst,IR);
    controller gate4(IR[31:26],IR[5:0],aluctrl,memtoreg,memwrite,alusrc,regdst,
        regwrite,branch,jump);
```

```
    always #50 begin
        clk<=~clk;
        $display("instruction:%h␣aluctrl:%b␣memtoreg:%b␣memwrite:%b␣alusrc:%b␣
            regdst:%b␣regwrite:%b␣branch:%b␣jump:%b",IR,aluctrl,memtoreg,
            memwrite,alusrc,regdst,regwrite,branch,jump);
    end

endmodule
```