《计算机组成原理》实验报告

年级、专业、班级	2022级计算机科学与技术03班/01班/06班	姓名	叶旭航，解吴雪，李佳玲
实验题目	实验二处理器译码实验		
实验时间	2024 年 4 月 2 日	实验地点	DS1410
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: 冯永

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 5月 18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 程序计数器(PC)

2.1.1 功能描述

d触发器结构，用于储存 PC(一个周期)

2.1.2 接口定义

2.1.3 逻辑控制

根据rst是否为1给pc和inst_ce赋值。当为1时，pc不变，inst_ce为0，rst为0，pc改变，inst_ce为1。

表 1: PC接口定义

信号名	方向	位宽	功能描述
clk	in	1	时钟周期
rst	in	1	复位信号
newpc	in	32	新地址
pc	out	32	输出的地址
Inst_ce	out	1	判断是否pc发生改变

2.2 加法器(Adder)

2.2.1 功能描述

用于计算下一条指令地址。

2.2.2 接口定义

表 2: Adder接口定义

信号名	方向	位宽	功能描述
Pc	in	32	初始位置
Pc_next	out	32	之后的位置
clk	in	1	时钟周期
rst	in	1	复位信号
Inst_ce	in	1	Pc部分的pc是否变换

2.2.3 逻辑控制

和上文的pc相结合，最后达成目的。

2.3 控制器(Controller)

2.3.1 功能描述

对于(a)main_decoder：负责判断指令类型，并生成相应的控制信号。

对于(b)alu_decoder：负责 ALU 模块控制信号的译码。

2.3.2 接口定义

表 3: main_decoder接口定义

信号名	方向	位宽	功能描述
op	in	6	指令
memtoreg	out	1	回写的数据来自于ALU计算的结果/存储器存储的数据
memwrite	out	1	是否要书写数据存储器
branch	out	1	是否为branch指令且满足branch的条件
regwrite	out	1	是否书写寄存器堆
jump	out	1	跳转
aluop	out	2	ALU选择

表 4: alu_decoder接口定义

信号名	方向	宽度	含义
funct	in	6	指令功能码
aluop	in	2	算术逻辑单元操作码
alucontrol	out	3	算术逻辑单元控制信号（指令的最终结果）

2.3.3 逻辑控制

对于(a)main_decoder：控制器输出的控制信号，用于控制器件的使能和多路选择器的选择，因此，根据不同指令的功能分析其所需要的路径，即可得到信号所对应的值。

对于(b)alu_decoder：先根据aluop判断是否为sw和beq，其他的操作要根据funct进一步判断，在这里用了三目操作符的方式。同时最后假如funct的数据不对，alucontrol=000不做任何操作。

2.4 存储器(Block Memory)

用于在 FPGA 中高效地存储和访问指令数据。

2.4.1 存储器Basic界面

2.4.2 存储器Part A Options界面

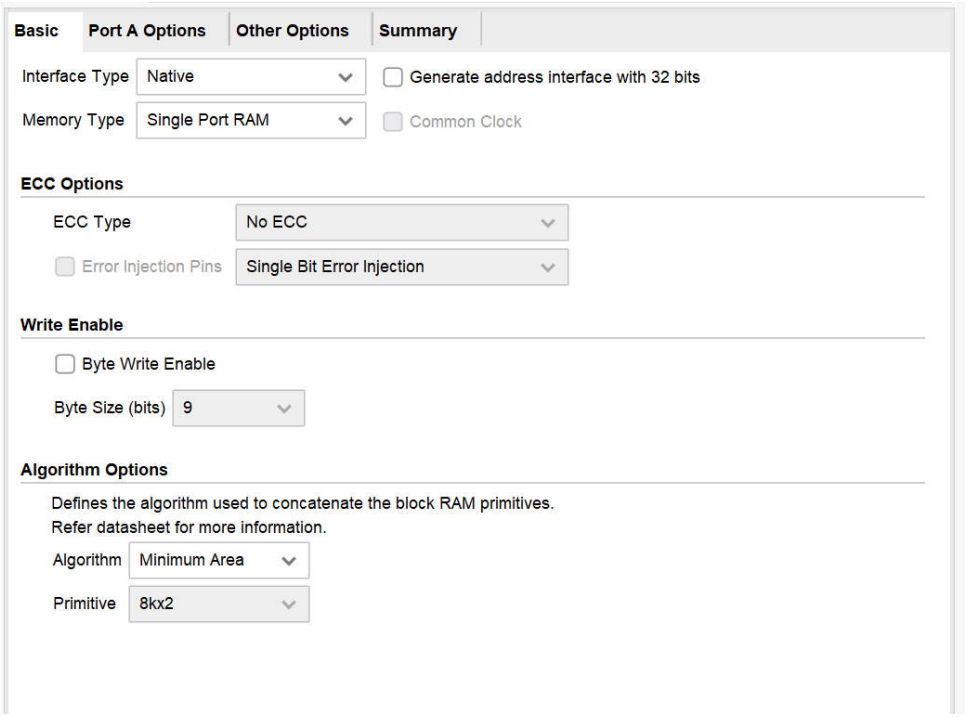

图 1: 存储器Basic界面

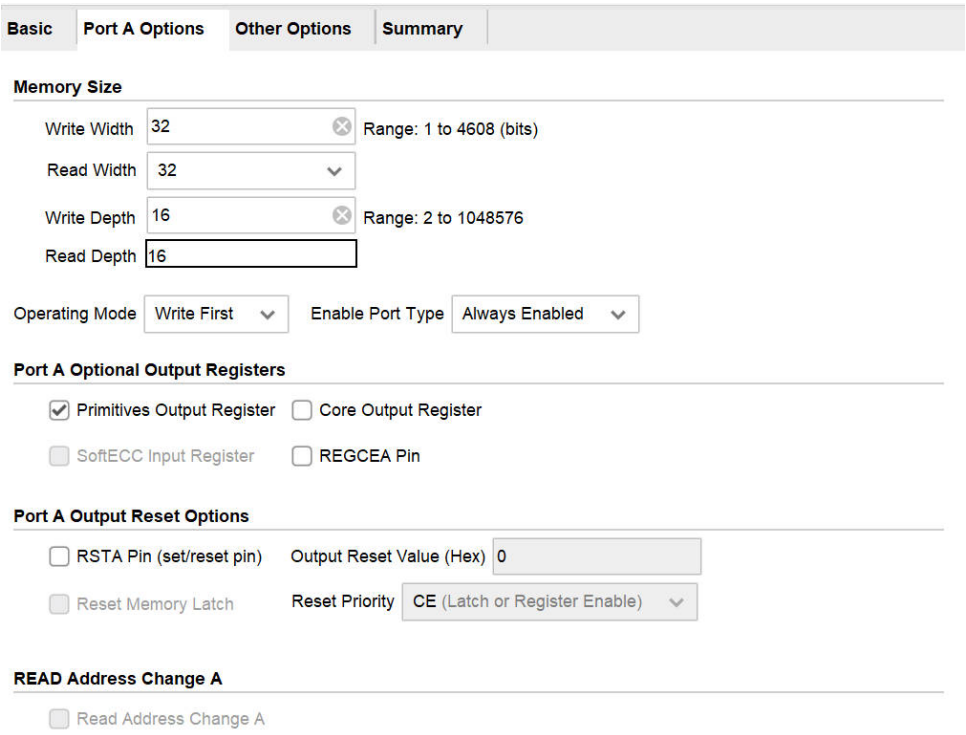

图 2: 存储器Part A Options界面

2.4.3 存储器Other Options界面

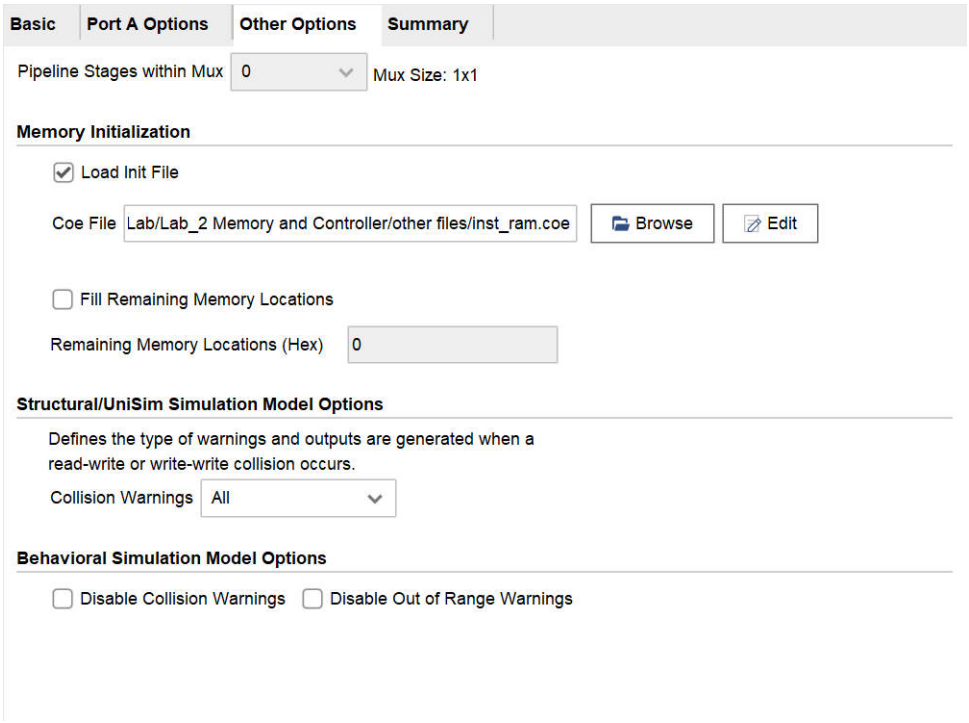

图 3: 存储器Other Options界面

3 实验过程记录

3.1 问题1: 程序计数单元的实现

问题描述：用 PC 模块和加法器共同构成一个完整的程序计数器功能单元。

解决方案：PC 模块存储当前的程序计数器值并控制指令使能信号，而加法器模块用于计算下一个程序计数器值，每个时钟周期加 4。通过将这两个模块串联，实现了程序计数器在时钟信号下按照设定的增量递增的功能。在 adder 模块中实例化一个 PC 模块，可以确保在加法器模块内部的程序计数器值返回给 PC 中。

3.2 问题2: main_decoder 模块译码

问题描述：main_decoder 负责判断指令类型并生成相应的控制信号。输入的指令需要正确的对应输出。

解决方案：查询给出的附录文件（附录 A_coe 文件涉及指令一查表.pdf），利用 `case` 语句根据文件给出正确的各控制信号的输出。

3.3 问题3: alu_decoder 模块译码

问题描述：alu_decoder 负责 ALU 模块控制信号的译码，需要配合 inst 低 6 位 funct，进行 ALU 模块控制信号的译码。

解决方案：同问题 2。查表后利用 `assign` 组合逻辑和判断语句给 alucontrol 赋值。

3.4 问题4: controller 中的额外端口输出

问题描述：未存在于 controller 调用的两个模块中的 pcsrc 的设计。

解决方案：虽然该实验要求中未要求该端口，但考虑到完整性最终在 controller 中设计了该端口。利用 `assign` 赋值为 branch 和 zero 变量的逻辑与操作，实现端口的设计。

3.5 问题5: 指令存储器实现

问题描述：使用 Block Memory Generator IP 构造指令存储器。

解决方案：根据指导书中的参考进行，导入给出的 coe 文件，并注意了地址深度的设置，此影响后续仿真和上板验证时的时间设置和指令循环。

3.6 问题6: 时钟分频

问题描述：实验要求在七段数码管上自动显示指令。控制显示频率需要时钟分频。

解决方案：`clk_div` 模块内部定义了一个计数器 count。没有复位信号时递增计数器 count 的值。当计数器达到一定值时，输出信号翻转一次，并将 count 清零，从而实现了时钟信号的分频。

3.7 问题7: 仿真调试

问题描述：自行编写 Testbench 仿真文件实现仿真。

解决方案：为实现仿真调试，需编写 Testbench 仿真文件并实例化 top 模块。设置时钟周期为 10ns，在每个时钟周期内输入地址 Address + 4，捕获输出信号并打印在控制台。通过运行仿真，可获得仿真图和控制台输出结果。

3.8 问题8: 约束文件实现

问题描述：利用约束文件实现管脚配置。

解决方案：根据给出的 Nexys4 板子的约束文件做出调整，时钟连接板载时钟，rst 连接板上实体按键，七段数码管显示指令后上板观察验证是否无误。

4 实验结果及分析

4.1 仿真图

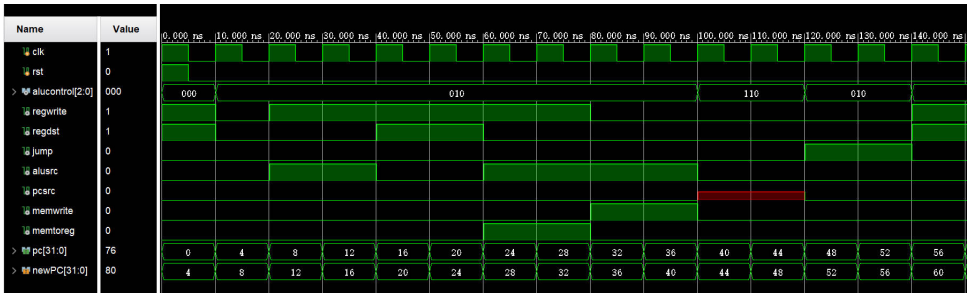

图 4: 仿真图

4.2 控制台输出

```
Simulation started
clk=0, rst=1, alucontrol=000, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=xxxxxxx, ans=11111110
clk=0, rst=0, alucontrol=000, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=xxxxxxx, ans=11111110
clk=0, rst=0, alucontrol=000, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=1111111, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=0, jump=0, alusrc=1, pcsrc=0, memwrite=0, memtoreg=0, seg=1111111, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=0, jump=0, alusrc=1, pcsrc=0, memwrite=0, memtoreg=0, seg=0100100, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0100100, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=0, jump=0, alusrc=1, pcsrc=0, memwrite=0, memtoreg=1, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=0, alusrc=1, pcsrc=0, memwrite=1, memtoreg=0, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=0, alusrc=1, pcsrc=0, memwrite=1, memtoreg=0, seg=1001100, ans=11111110
clk=0, rst=0, alucontrol=110, regwrite=0, regdst=0, jump=0, alusrc=0, pcsrc=x, memwrite=0, memtoreg=0, seg=1001100, ans=11111110
clk=0, rst=0, alucontrol=110, regwrite=0, regdst=0, jump=0, alusrc=0, pcsrc=x, memwrite=0, memtoreg=0, seg=0110001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=1, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0110001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=1, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0000110, ans=11111110
clk=0, rst=0, alucontrol=000, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0000110, ans=11111110
clk=0, rst=0, alucontrol=000, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=1, jump=0, alusrc=1, pcsrc=0, memwrite=0, memtoreg=0, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=0, jump=0, alusrc=1, pcsrc=0, memwrite=0, memtoreg=0, seg=0100100, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0100100, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=1, regdst=0, jump=0, alusrc=1, pcsrc=0, memwrite=0, memtoreg=1, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=0, alusrc=1, pcsrc=0, memwrite=1, memtoreg=0, seg=0000001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=0, alusrc=1, pcsrc=0, memwrite=1, memtoreg=0, seg=1001100, ans=11111110
clk=0, rst=0, alucontrol=110, regwrite=0, regdst=0, jump=0, alusrc=0, pcsrc=x, memwrite=0, memtoreg=0, seg=1001100, ans=11111110
clk=0, rst=0, alucontrol=110, regwrite=0, regdst=0, jump=0, alusrc=0, pcsrc=x, memwrite=0, memtoreg=0, seg=0110001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=1, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0110001, ans=11111110
clk=0, rst=0, alucontrol=010, regwrite=0, regdst=0, jump=1, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0000110, ans=11111110
clk=0, rst=0, alucontrol=000, regwrite=1, regdst=1, jump=0, alusrc=0, pcsrc=0, memwrite=0, memtoreg=0, seg=0000110, ans=11111110
```

图 5: 控制台输出

A Controller代码

A.1 main_decoder代码

```
    module main_decoder(
        input [5:0] op,
        output memtoreg,memwrite,
        output branch,alusrc,
        output regdst,regwrite,
        output wire jump,
        output wire [1:0] aluop
        );

    reg [8:0] controls;
    assign {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump,aluop} =
        controls;
    always @(*) begin
            case (op)
                6'b000000:controls <= 9'b110000010;//R-TYRE
                6'b100011:controls <= 9'b101001000;//LW
                6'b101011:controls <= 9'b001010000;//SW
                6'b000100:controls <= 9'b000100001;//BEQ
                6'b001000:controls <= 9'b101000000;//ADDI
                6'b000010:controls <= 9'b000000100;//J
                default:  controls <= 9'b000000000;//illegal op
            endcase
        end
    endmodule
```

A.2 alu_decoder代码

```
module alu_decoder(
    input [5:0] funct,
    input [1:0] aluop,
    output wire [2:0] alucontrol
);
    assign alucontrol = (aluop == 2'b00) ? 3'b010 :   // sw
                        (aluop == 2'b01) ? 3'b110 :  // beq
                        (aluop == 2'b10) ? (
                            (funct == 6'b100000) ? 3'b010 :  // add
                            (funct == 6'b100010) ? 3'b110 :  // sub
                            (funct == 6'b100100) ? 3'b000 :  // and
                            (funct == 6'b100101) ? 3'b001 :  // or
                            (funct == 6'b101010) ? 3'b111 :  // slt
                            3'b000
                        ) : 3'b000;

endmodule
```