《计算机组成原理》实验报告

年级、专业、班级	2021级计算机科学与技术03班，05班	姓名	任俊璇，张梓健
实验题目	实验二处理器译码实验		
实验时间	2023 年 04 月 03 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: 2023年 4月 20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

接收存储器传来的32位指令，调用Main decoder和Alu decoder，生成memtoreg,memwrite, branch,alusrc,regdst,regwrite,jump,alucontrol信号。

2.1.2 接口定义

表 1: 接口定义模版

信号名	方向	位宽	功能描述
op	input	6-bit	指令op码。
funct	input	6-bit	指令funct码。
memtoreg	output	1-bit	回写的数据来自于 ALU 计算的结果还是存储器读取的数据。
memwrite	output	1-bit	是否需要写数据存储器。
alusrc	output	1-bit	送入 ALU B 端口的值是立即数的 32 位扩展还是寄存器堆读取的值。
regdst	output	1-bit	写入寄存器堆的地址是 rt 还是 rd,0 为 rt,1 为 rd。
regwrite	output	1-bit	是否需要写寄存器堆。
branch	output	1-bit	是否为 branch 指令，且满足 branch 的条件。
jump	output	1-bit	是否为 jump 指令。
alucontrol	output	1-bit	ALU 控制信号，代表不同的运算类型。

2.1.3 逻辑控制

controller模块接收指令的op码和funct码，先调用Main decoder，根据op得到memtoreg,memwrite, branch,alusrc,regdst,regwrite,jump,aluop，然后调用Alu decoder，根据aluop和funct得到alucontrol。

2.2 存储器(Block Memory)

2.2.1 类型选择

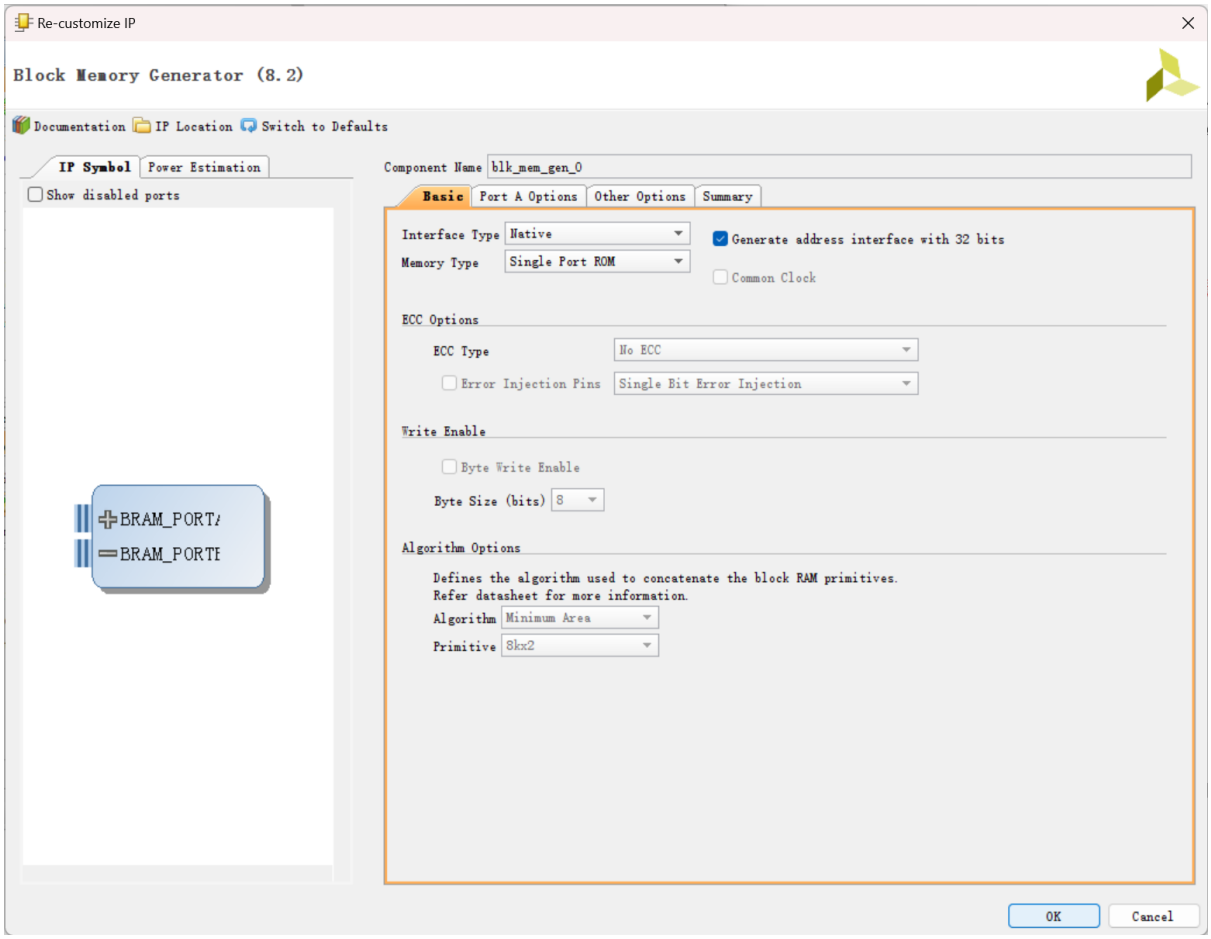

图 1: 类型选择为单端口rom

2.2.2 参数设置

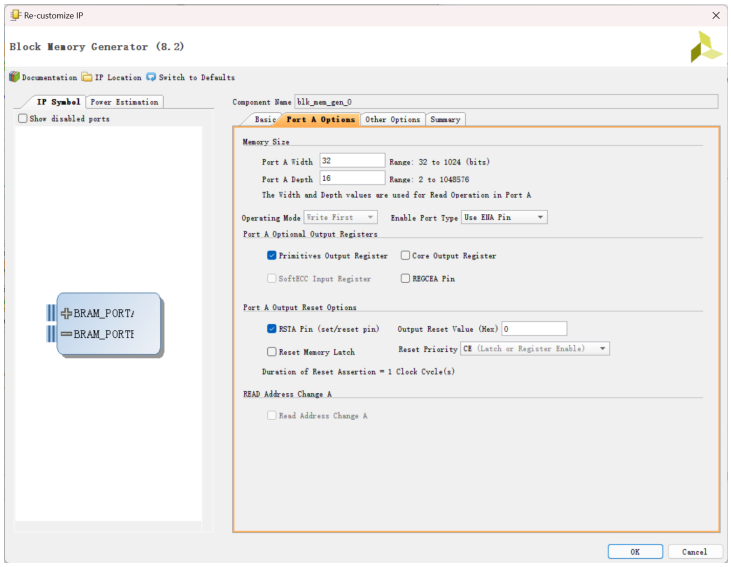

图 2: 参数设置1

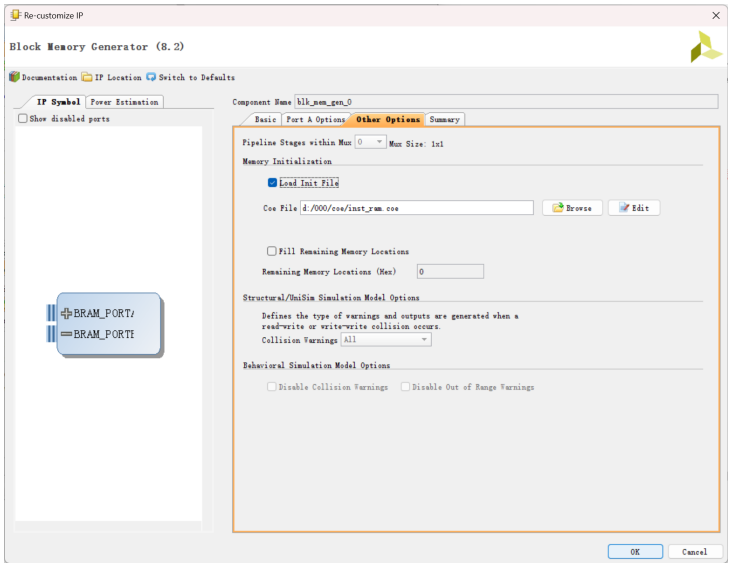

图 3: 参数设置2

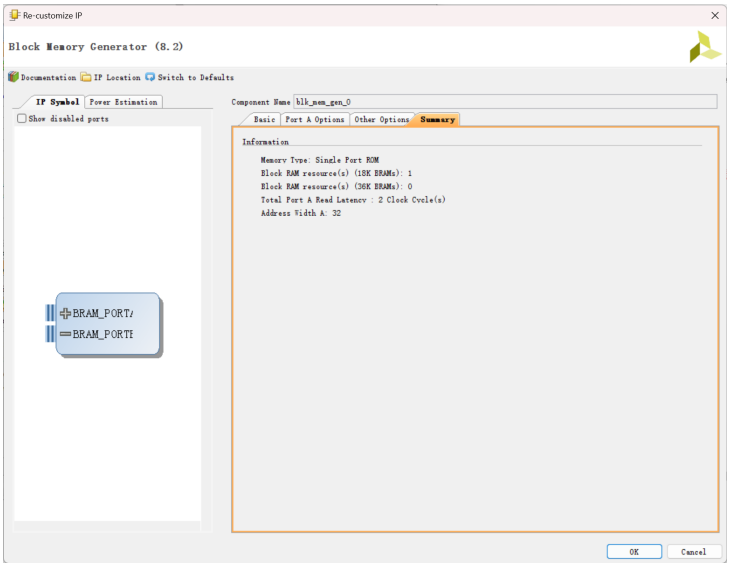

图 4: 参数设置3

```
memory_initialization_radix = 16;
memory_initialization_vector =
20020005,
00a42820,
8c020050,
ac020054,
10a7000c,
08000013,
```

3 实验过程记录

3.1 实验过程

创建 PC 模块:根据实验要求设计PC模块,以clk和rst为输入,输出pc和inst_ce。

创建 main_decde, alu_decode 模块:根据实验要求设计main_decde模块,以op为输入,输出regwrite, regdst,alusrc,branch,memwrite,memtoreg,jump,aluop信号;根据实验要求设计alu_decode模块,以aluop和funct为输入,输出alucontrol。

创建 Controller模块, 调用 main_decode, alu_decode: 根据实验要求设计Controller模块,以op和funct为输入,调用main_decode, alu_decode模块,输出memtoreg,memwrite,branch,alusrc,regdst, regwrite,jump,alucontrol。

使用 Block Memory, 导入 coe 文件:按照实验指导书调用 Xilinx 库 Block Memory Generator。

自定义顶层文件, 连接相关模块:根据要求调用以上模块。

编写 Testbench 仿真文件:按照实验要求编写仿真文件,时钟周期设为 10ns,每个时钟周期输入地址 Address + 4,捕获输出信号并打印在控制台。

下板验证：生成bit流文件，进行下板验证。

3.2 遇到的问题

3.2.1 问题一

不了解Block Memory应该如何接入，应该选择哪种类型

3.2.2 解决方案

通过阅读指导手册和网上查询，选择single port rom类型，并找到了对应的接口。

3.2.3 问题二

在仿真时，Block Memory的输出一直为0没有变化

3.2.4 解决方案

观察rtl图发现Block Memory的addra至于pc的第0位连接，其余位接地，最后发现问题是顶层模块中，没有添加中间变量pc。

4 实验结果及分析

4.1 仿真图

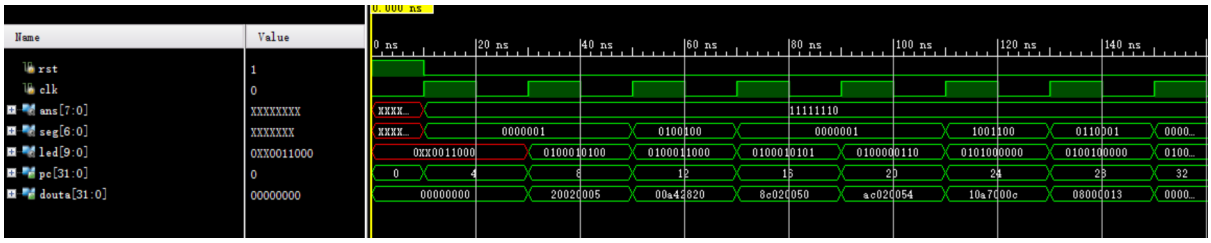

图 5: 仿真图

由图，led即为八种控制输出信号alucontrol,branch,jump,regwrite,regdst,alusrc,memwrite,memtoreg的输出，pc每过一个时钟周期就会自加4，block memory的输出为20020005,00a42820,8c020050，ac020054,10a7000c,080
换。

A Controller代码

A.1 controller

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

  main_decoder x(op,memtoreg,memwrite,branch,alusrc,regdst,regwrite,jump,aluop)
      ;
  alu_control y(funct,aluop,alucontrol);

endmodule
```

A.1.1 main_decoder

```
module main_decoder(op,memtoreg,memwrite,branch,alusrc,regdst,regwrite,jump,
    aluop);
  input wire[5:0] op;
  output wire memtoreg,memwrite,branch,alusrc,regdst,regwrite,jump;
  output wire[1:0] aluop;
  reg[8:0] controls;

  always @(*) begin
    case (op)
      6'b000000:controls <= 9'b110000010;//R-type
      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;//error
    endcase
  end
  assign {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump,aluop} =
      controls;
endmodule
```

A.1.2 alu_decoder

```
module alu_control(funct,aluop,alucontrol);
    input wire [5:0] funct;
    input wire [1:0] aluop;
```

```
    output wire [2:0] alucontrol;
    reg[2:0] result;
    always @(*) begin
    case (aluop)
      2'b00: result = 010;//lw or sw
      2'b01: result = 110;//beq
      2'b10: //R-type
      case (funct)
      6'b100000: result = 010;//add
      6'b100010: result = 110;//subtract
      6'b100100: result = 000;//and
      6'b100101: result = 001;//or
      6'b101010: result = 111;//set-on-less-than
      endcase
    endcase
  end
    assign alucontrol = result;
endmodule
```