***2023***



**硬件综合训练 课程设计报告**

|  |  |
| --- | --- |
| 题 目： | 5段流水CPU设计 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计算机21本硕博 |
| 学 号： | U202115674 |
| 姓 名： | 姚晨炫 |
| 电 话： | 15105919611 |
| 邮 件： | deadfffool@gmail.com |

目 录

[1 课程设计概述 3](#_Toc141690915)

[1.1 课设目的 3](#_Toc141690916)

[1.2 设计任务 3](#_Toc141690917)

[1.3 设计要求 3](#_Toc141690918)

[1.4 技术指标 4](#_Toc141690919)

[2 总体方案设计 6](#_Toc141690920)

[2.1 单周期CPU设计 6](#_Toc141690921)

[2.2 中断机制设计 8](#_Toc141690922)

[2.3 流水CPU设计 9](#_Toc141690923)

[2.4 气泡式流水线设计 9](#_Toc141690924)

[2.5 数据转发流水线设计 10](#_Toc141690925)

[2.6 动态分支预测机制（未完成部分可不写，直接删除） 10](#_Toc141690926)

[3 详细设计与实现 11](#_Toc141690927)

[3.1 单周期CPU 实现 11](#_Toc141690928)

[3.2 中断机制实现 13](#_Toc141690929)

[3.3 流水CPU实现 14](#_Toc141690930)

[3.4 气泡式流水线实现 14](#_Toc141690931)

[3.5 数据转发流水线实现 14](#_Toc141690932)

[3.6 动态分支预测机制实现 15](#_Toc141690933)

[4 实验过程与调试 16](#_Toc141690934)

[4.1 测试用例和功能测试 16](#_Toc141690935)

[4.2 可自行安排章节 16](#_Toc141690936)

[4.3 性能分析 16](#_Toc141690937)

[4.4 主要故障与调试 17](#_Toc141690938)

[4.5 实验进度 19](#_Toc141690939)

[5 设计总结与心得 21](#_Toc141690940)

[5.1 课设总结 21](#_Toc141690941)

[5.2 课设心得 21](#_Toc141690942)

[参考文献 23](#_Toc141690943)

# 课程设计概述

## 课设目的

计算机组成原理是计算机专业的核心基础课。该课程力图以“培养学生现代计算机系统设计能力”为目标，贯彻“强调软/硬件关联与协同、以CPU设计为核心/层次化系统设计的组织思路，有效地增强对学生的计算机系统设计与实现能力的培养”。课程设计是完成该课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有一定规模的指令系统的简单计算机系统。所设计的系统能在LOGISIM仿真平台和FPGA实验平台上正确运行，通过检查程序结果的正确性来判断所设计计算机系统正确性。

课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行中央处理器底层电路的实现、故障分析与定位、系统调试等环节的综合锻炼，进一步提高学生分析和解决问题的能力。

## 设计任务

本课程设计的总体目标是利用FPGA以及相关外围器件，设计五段流水CPU，要求所设计的流水CPU系统能支持自动和单步运行方式，能正确地执行存放在主存中的程序的功能，对主要的数据流和控制流通过LED、数码管等适时的进行显示，方便监控和调试。尽可能利用EDA软件或仿真软件对模型机系统中各部件进行仿真分析和功能验证。在学有余力的前提下，可进一步扩展相关功能。

## 设计要求

1. 根据课程设计指导书的要求，制定出设计方案；
2. 分析指令系统格式，指令系统功能。
3. 根据指令系统构建基本功能部件，主要数据通路。
4. 根据功能部件及数据通路连接，分析所需要的控制信号以及这些控制信号的有效形式；
5. 设计出实现指令功能的硬布线控制器；
6. 调试、数据分析、验收检查；
7. 课程设计报告和总结。

## 技术指标

1. 支持规定的32位RISC-V指令，具体见表格1.1；
2. 在CCAB扩展指令集中支持2条C类运算指令，1条M类存储指令，1条B类分支指令。
3. 支持多级嵌套中断，利用中断触发扩展指令集测试程序；
4. 支持5段流水机制，可处理数据冒险，结构冒险，分支冒险；
5. 能运行由自己所设计的指令系统构成的一段测试程序，测试程序应能涵盖所有指令，程序执行功能正确。
6. 能运行教师提供的标准测试程序，并自动统计执行周期数
7. 能自动统计各类分支指令数目，如不同种类指令的条数、冒险冲突次数、插入气泡数目、load-use冲突次数、动态分支预测流水线能自动统计预测成功与失败次数。
8. 表 1.1 指令集

| **#** | **指令助记符** | **简单功能描述** | **备注** |
| --- | --- | --- | --- |
| 1 | ADD | 加法 | 指令格式与功能请参考 RISC-V32 指令集英文手册，或参考 RARS 模拟器 |
| 2 | ADDI | 立即数加 |
| 3 | AND | 与 |
| 4 | SLLI | 逻辑左移 |
| 5 | SRAI | 算数右移 |
| 6 | SRLI | 逻辑右移 |
| 7 | SUB | 减 |
| 8 | OR | 或 |
| 9 | ORI | 立即数或 |
| 10 | XORI | 或非/立即数异或 |
| 11 | LW | 加载字 |
| 12 | SW | 存字 |
| 13 | BEQ | 相等跳转 |
| 14 | ANDI | 立即数与 |
| 15 | BNE | 不相等跳转 |
| 16 | SLT | 小于置数 |
| 17 | SLTI | 小于立即置数 |
| 18 | SLTU | 小于无符号置数 |
| 19 | JAL | 转移并链接 |
| 20 | JALR | 转移到指定寄存器 |
| 21 | CSRRSI | 访问CSR寄存器 |
| 22 | CSRRCI | 访问CSR寄存器 |
| 23 | URET | 中断返回 | if ($a7==34) LED 输出$a0 的值 else 停机等待 Go 按键按下注意显示逻辑锁存住数据 |
| 24 | ECALL | 系统调用 |
| 25 | SRL | 逻辑右移 |  |
| 26 | LUI | 高位立即数加载 |  |
| 27 | SB | 低位字节写入内存 |  |
| 28 | BLT | 小于时分支跳转 |  |

# 总体方案设计

## 单周期CPU设计

本次单周期CPU设计我们采用程序存储器和数据存储器分开的设计，来保证一个周期内可以正确完成一条指令，通过硬布线控制器，我们可以设置好不同指令对应的控制信号与ALU的运算控制。

总体结构图如图 2.1所示。



图 2.1 总体结构图

### 主要功能部件

#### 程序计数器PC

程序计数器PC的思想是借助于寄存器，寄存器存入当前周期的PC的值输出端连接指令存储器，而输入端具有多种可能分别是分支于按序的下一条指令的地址，PC寄存器的使能至于ECALL指令有关，ECALL指令根据寄存器数据的不同作出print和halt的操作。

#### 指令存储器IM

指令存储采用rom存储器，提前将对应的程序写入存储器中，根据PC取出指令，利用分线器将不同字段送入不同部件，控制相关的字段送入硬布线程序控制器，寄存器相关字段送入寄存器堆，立即数字段通过相关扩展后送入ALU或者直接写入内存。

#### 运算器

运算器引脚及功能如下表2.1所示：

表 2.1算术逻辑运算单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| X | 输入 | 32 | 操作数X |
| Y | 输入 | 32 | 操作数Y |
| ALU\_OP | 输入 | 4 | 运算器功能码，具体功能见下表 |
| Result | 输出 | 32 | ALU运算结果 |
| Result2 | 输出 | 32 | ALU结果第二部分，用于乘法指令结果高位或除法指令的余数位，其他操作为零 |
| OF | 输出 | 1 | 有符号加减溢出标记，其他操作为零 |
| UOF | 输出 | 1 | 无符号加减溢出标记，其他操作为零 |
| Equal | 输出 | 1 | Equal=(x==y)?1:0, 对所有操作有效 |

#### 寄存器堆RF

由多个寄存器组成的寄存器堆，通过rs，rt控制寄存器堆的读，通过Rin和W来控制寄存器堆的写。

### 数据通路的设计

数据通路设计如下图所示

A diagram of a computer

Description automatically generated

图 2.2 数据通路图

### 控制器的设计

首先对于控制信号进行统计，包括各个主要部件所需要输入的控制信号，以及数据通路合并表中所示的具有多输入的主要部件需要进行输入选择的控制信号，并且对各个统计信号的各种取值情况进行定义，统计得到的控制信号以及说明如表 2.2。

表 2.2主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| MemtoReg | 0 | 不写回 |
| 1 | 数据存储器写回寄存器堆 |
| MemWrite | 0 | 数据存储器写禁止 |
| 1 | 数据存储器写允许 |
| ALU\_Src | 0 | 取寄存器堆中r2地址对应的内容 |
| 1 | 把立即数送入到ALU的输入 |
| RegWrite | 0 | 寄存器堆写不允许 |
| 1 | 寄存器堆写允许 |
| Ecall | 0 |  |
| 1 | 进行终止判断 |
| S\_TYPE | 0 | 不是S型指令 |
| 1 | 是S型指令 |
| BEQ | 0 | 非BEQ指令 |
| 1 | 是BEQ指令 |
| BNE | 0 | 非BNE指令 |
| 1 | 是BNE指令 |
| JAL | 0 | 非JAL指令 |
| 1 | 是JAL指令 |
| Result2 | 0 | 运算器结果为result1 |
| 1 | 结果为result2 |
| Memex | 0 | 数据寄存器内容输出不需要扩展 |
| 1 | 数据寄存器内容输出需要扩展 |
| pcimm | 0 | 分支关 |
| 1 | 分支开 |
| CSRRSI | 1 | 开中断启动 |
| 0 | 无 |
| CSRRCI | 1 | 关中断启动 |
| 0 | 无 |

对照所有控制信号，依次分析各条指令，分析该指令执行过程中需要哪些控制信号，对于与本条指令无关的控制信号，控制信号的取值一律为0，以简化控制器电路的设计。该控制信号表的框架如表 2.3所示。

表 2.3主控制器控制信号框架

| 指令 | ALU\_OP | MemtoReg | MemWrite | ALU\_Src | RegWrite | ecall | S\_Type | BEQ | BNE | ALU\_OP | MemtoReg | … |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|
| add | 5 |  |  |  | 1 |  |  |  |  |  |  | … |

## 中断机制设计

### 总体设计

（1）单级中断：在单周期的数据通路与控制器不变的情况下，加入不同中断的响应装置，中断相关的寄存器，根据是否中断开始以及中断信号处理是否完成，加入相关的地址跳转信号。

（2）多级中断：增加多级中断的相关调度器，以及更改寄存器结构来适应多级中断中可能出现的更多断点地址。

### 硬件设计

（1）中断请求信号：硬件中断设计的核心是中断请求信号。使用程序包中的中断按键参考信号来实现中断请求信号。

（2）中断控制器：为了有效地管理多个中断请求，CPU通常会使用中断控制器。中断控制器负责接收中断请求，确定哪个中断应该被响应（优先级管理）以及通知CPU开始执行相应的中断服务例程。

（3）中断向量：每个中断都有一个唯一的标识号，称为中断向量。中断向量用于告诉CPU响应哪个中断。在多级中断的调度器中，使用优先编码器来调度。

（4）中断处理：一旦CPU确定要响应中断，它会保存当前的上下文（如寄存器状态）并跳转到中断服务例程的地址。在单周期CPU中，这通常涉及将中断服务例程的地址加载到程序计数器（PC）中，以便开始执行该例程。执行中断服务例程后，CPU通常会根据中断的类型进行一些额外的操作，如清除中断标志、保存中断发生时的寄存器状态等。

（5）中断返回：一旦中断服务例程完成，CPU需要恢复之前保存的上下文，以便从中断点继续执行。这包括恢复寄存器状态和重新设置PC。在我们的设计中，会根据控制信号加载EPC中寄存好的地址。

## 流水CPU设计

### 总体设计

理想流水线在设计时根据指令执行的特征分为了五个阶段，分别是取指阶段，译码阶段，运算阶段，访存阶段，写回阶段，在每个部分之间设置一个流水寄存器将整个过程都统一划分为5个时钟周期，从而保证了流水线的完整操作。

### 流水接口部件设计

流水线的接口均采用寄存器的方式便于传输和存储数据，寄存器负责传输相关的控制信号，IR，PC寄存器中的数据，以及读取和运算的数据内容。寄存器需要有统一的使能和复位信号，便于控制流水线的暂停和气泡。

### 理想流水线设计

理想流水线不考虑分支指令和读写数据的相关冲突，因此较为简单，只需要按序输入PC取出的指令即可。流水线中的每一个阶段对应的都是一条不同指令，理想流水线中没有气泡和重定向设计。

## 气泡式流水线设计

与理想流水线不同的是，气泡流水线考虑相关的指令相关，指令相关指在指令流水线中，如果某指令的某个阶段必须等到它前面的某条指令的某个阶段完成后才能开始，也就是两条指令间存在某种依赖关系，则两条指令存在指令相关。指令相关包括数据相关、结构相关、控制相关，指令相关会导致流水线冲突(冒险)，流水线冲突是指由于指令相关的存在，指令流水线出现“阻塞”或“暂停”，下一条指令不能在预期的时钟周期内加载到流水线中。流水线冲突包括结构冲突、控制冲突、数据冲突3种。

结构冲突是由于读写地址和数据均通过不同的结构冲突，但由于 RISC-V 寄存器堆的读写端口进入，读写逻辑可以并发操作，因此这种结构冲突并不存在。

分支指令会引起控制冲突，要解决控制冲突，在执行程序分支跳转时必须清除流水线中分支指令后续的若干条误取指令。

数据冲突时由于ID段取寄存器数据时会与EX段和MEM段存在相关的依赖关系，因此流水线中必须增加硬件逻辑来实现 ID 段与 EX、MEM 段指令的数据相关性检测。有了数据相关检测逻辑，当发生数据相关时，给ID/EX 流水寄存器一个同步清空信号 Flush即可；而要暂停IF、ID 段指令的执行，只需保证程序计数器 PC 和IF/ID流水寄存器的值不变即可。要做到这一点，只需要控制寄存器使能端即可，当使能端为1时，寄存器正常工作，为 0时则忽略时钟输入，寄存器值保持不变。

## 重定向流水线设计

气泡流水线通过延缓 ID 段取操作数动作的方式解决数据冲突问题，但大量气泡的插入会严重影响指令流水线的性能。重定向流水线的思路是先不考虑ID 段所取的寄存器操作数是否正确，等到指令实际使用这些寄存器操作数时再考虑其正确性问题。

重定向技术根据不同的冲突类型产生不同的控制信号，将需要更新的数据部分从流水线后段直接传入，不等待指令完全执行完毕，使得流水线的效率大大提高。

## 动态分支预测机制

采用重定向机制后，指令流水线中的数据相关基本不需要插入气泡就可解决，只有少数Load-Use 相关还需要插入一个气泡，流水线性能得到极大的提升。此时流水线中的控制冲突对流水线性能影响最大，基于加快经常性事件的原理，应优先考虑减少分支指令引起的分支延迟损失。为减少分支延迟损失，应尽可能提前执行分支指令，例如将分支指令放在ID 段完成。动态分支技术根据跳转历史的准确率对跳转策略不断调整，提高预测准确率，因为分支具有局部性，比如程序中for循环和while循环，动态分支预测就是利用这种局部性来进行预测

# 详细设计与实现

## 单周期CPU 实现

### 主要功能部件实现

1. 程序计数器（PC）
2. Logism实现：

使用一个32位寄存器实现程序计数器PC，触发方式为下降沿触发，输入为下一条将要执行的指令的地址，输出为当前执行指令的地址。Halt为停机信号，经过一个非门后与寄存器使能端相连，如图 3.1所示。

A computer screen shot of a computer

Description automatically generated

图 3.1程序计数器（PC）

1. FPGA实现：

程序计数器使用register寄存器的实例化，register的Verilog代码如下：

module register(CLK, RST, EN, Din, Dout);

parameter WIDTH = 32; // Register width, default is 32 bits

input CLK, RST, EN; // Clock, reset, and enable signals

input [WIDTH-1:0] Din; // Data input

output [WIDTH-1:0] Dout; // Data output

reg [31:0] ram; // Register

initial ram = 0; // Initialize the register to 0

always @(posedge CLK) begin

if (RST)

ram <= 0; // If reset signal is active, set register value to 0

else if (EN)

ram <= Din;

else

ram = ram; // Otherwise, keep the register value unchanged

end

assign Dout = ram; // Data output is the register value

endmodule

1. 程序计数器（PC）
2. Logism实现：

使用一个只读存储器ROM实现指令存储器（IM）。设置该只读存储器的地址位宽为10位，数据位宽为32位。因为PC中存储的指令地址有32位，而ROM地址线宽度有限，仅为10位，故将32位指令地址高位部分和字节偏移部分直接屏蔽，使用分线器只取32位指令地址的2-11位作为指令存储器的输入地址。如图 3.2所示。



图 3.2指令存储器（IM）

1. FPGA实现：

直接使用Vivado中自带的RAM，数据位宽为32位，地址位宽为4位。

指令存储器IM的Verilog代码如下：

module RegFile(Din, R1Adr, R2Adr, WAdr, WE, CLK, R1, R2);

input [31:0] Din; // Data input

input [4:0] WAdr, R1Adr, R2Adr; // Write address, R1 address, and R2 address

input WE, CLK; // Write enable and clock signal

output [31:0] R1, R2; // R1 and R2 outputs

reg [31:0] RAM[31:0]; // Register file, size is 32 registers

integer i;

initial begin

for (i = 0; i <= 31; i = i + 1) begin

RAM[i] = 0; // Initialize all registers to 0

end

end

assign R1 = RAM[R1Adr]; // R1 output is the value at the specified address

assign R2 = RAM[R2Adr]; // R2 output is the value at the specified address

always @(posedge CLK) begin

if (WE && WAdr != 0)

RAM[WAdr] <= Din;

end

endmodule

### 控制器的实现

1. 硬布线控制器
2. Logisim实现

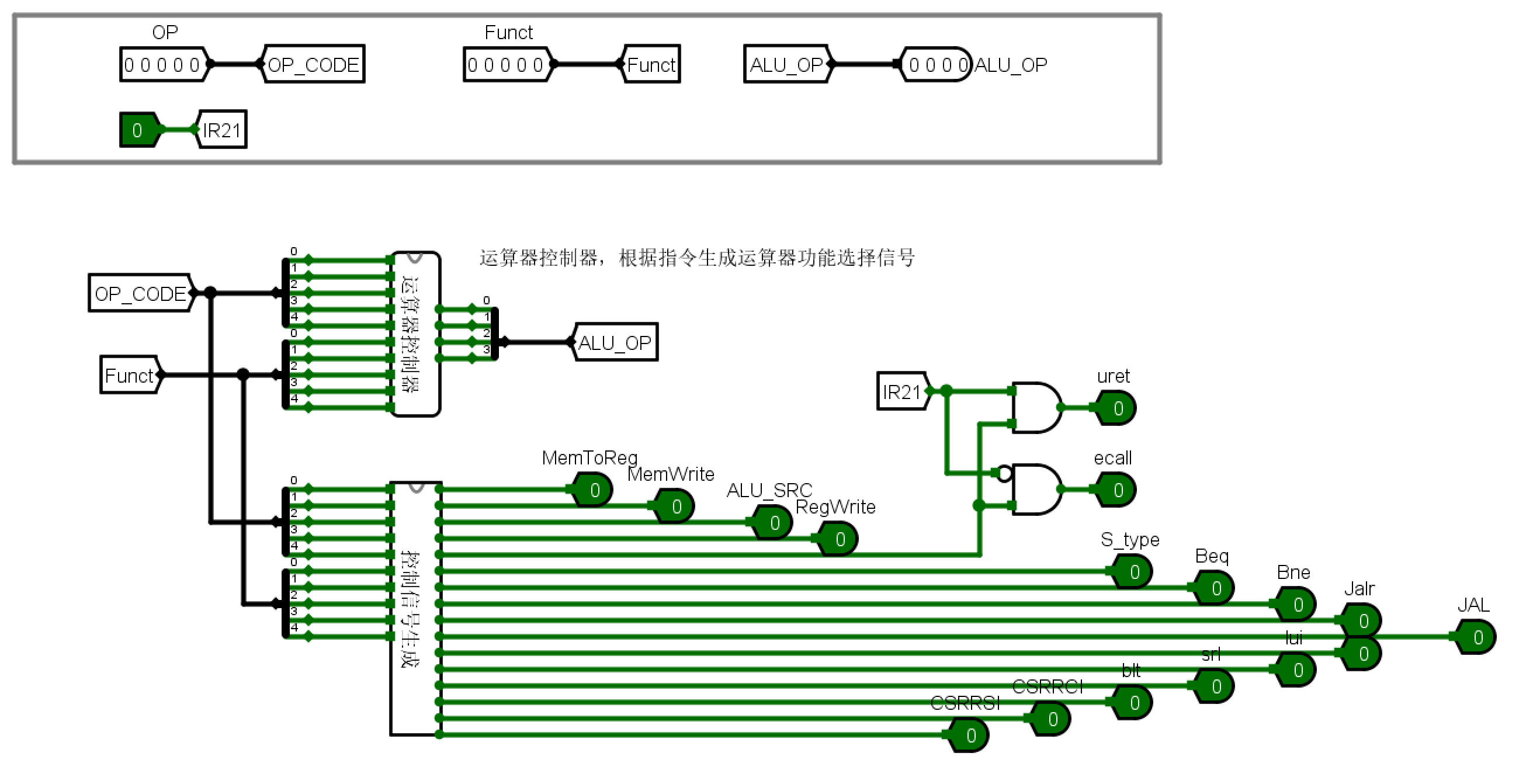


图 3.3硬布线控制器

硬布线控制器通过OP\_CODE和Funct来综合判断对应的控制型号和运算方式，运算器控制器和控制信号通过execl自动生成，如下图所示



图 3.3控制信号生成

根据生成的逻辑表达式，利用Logisim电路仿真实现硬布线控制器的设计

1. FPGA实现

是用assign语句，如图3.3根据不同的指令情况分配不同的信号，一下是ALUOP的assign语句，控制信号同理。

assign ALUOP =(Funct7 == 7'b0000000 && Funct3 == 3'b000 && OpCode == 5'b01100) ? 4'b0101 : // add

(Funct7 == 7'b0100000 && Funct3 == 3'b000 && OpCode == 5'b01100) ? 4'b0110 : // sub

(Funct7 == 7'b0000000 && Funct3 == 3'b111 && OpCode == 5'b01100) ? 4'b0111 : // and

(Funct7 == 7'b0000000 && Funct3 == 3'b110 && OpCode == 5'b01100) ? 4'b1000 : // or

(Funct7 == 7'b0000000 && Funct3 == 3'b010 && OpCode == 5'b01100) ? 4'b1011 : // slt

(Funct7 == 7'b0000000 && Funct3 == 3'b011 && OpCode == 5'b01100) ? 4'b1100 : // sltu

(Funct3 == 3'b000 && OpCode == 5'b00100) ? 4'b0101 : // addi

(Funct3 == 3'b111 && OpCode == 5'b00100) ? 4'b0111 : // andi

(Funct3 == 3'b110 && OpCode == 5'b00100) ? 4'b1000 : // ori

(Funct3 == 3'b100 && OpCode == 5'b00100) ? 4'b1001 : // xori

(Funct3 == 3'b010 && OpCode == 5'b00100) ? 4'b1011 : // slti

(Funct7 == 7'b0000000 && Funct3 == 3'b001 && OpCode == 5'b00100) ? 4'b0000 : // slli

(Funct7 == 7'b0000000 && Funct3 == 3'b101 && OpCode == 5'b00100) ? 4'b0010 : // srli

(Funct7 == 7'b0100000 && Funct3 == 3'b101 && OpCode == 5'b00100) ? 4'b0001 : // srai

(Funct3 == 3'b010 && OpCode == 5'b00000) ? 4'b0101 : // lw

(Funct3 == 3'b010 && OpCode == 5'b01000) ? 4'b0101 : // sw

(Funct3 == 3'b000 && OpCode == 5'b11000) ? 4'b1011 : // beq

(Funct3 == 3'b001 && OpCode == 5'b11000) ? 4'b1011 : // bne

(OpCode == 5'b11011) ? 4'b0101 : // jal

(Funct3 == 3'b000 && OpCode == 5'b11001) ? 4'b0101 : // jalr

4'b0000; //default

Endmodule

1. ALU
2. Logisim设计

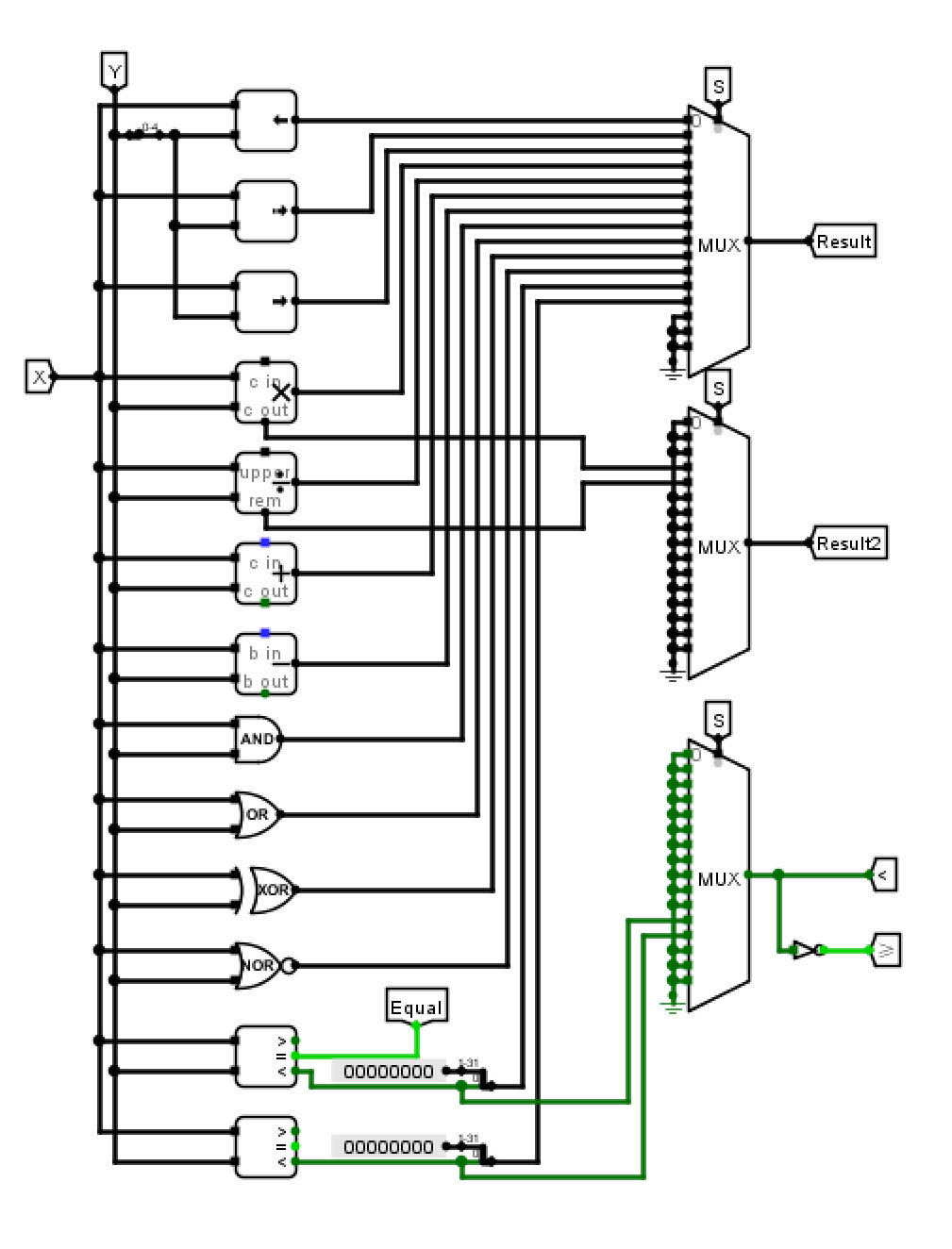


图 3.4 ALU设计电路图

ALU设计中，X，Y作为输入，根据ALUOP控制信号的选择不同设置不同的运算器，然后输出运算结果

1. FPGA实现

与硬布线相似，根据不同的输入，利用assign语句来选择不同的运算器，使用组合逻辑和线网形。

assign ALU\_result = (ALU\_op == 4'b0000)? x<<y[4:0]:

(ALU\_op == 4'b0001)? $signed($signed(x)>>>y[4:0]):

(ALU\_op == 4'b0010)? x>>y[4:0]:

(ALU\_op == 4'b0011)? x\*y[31:0]:

(ALU\_op == 4'b0100)? x/y:

(ALU\_op == 4'b0101)? x+y:

(ALU\_op == 4'b0110)? x-y:

(ALU\_op == 4'b0111)? x&y:

(ALU\_op == 4'b1000)? x|y:

(ALU\_op == 4'b1001)? x^y:

(ALU\_op == 4'b1010)? ~(x|y):

(ALU\_op == 4'b1011)? ($signed(x)<$signed(y))?1:0:

(ALU\_op == 4'b1100)? ((x<y)?1:0):

32'b0;

## 中断机制实现

### XXX

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)

### XXXX

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)

## 流水CPU实现

### 流水接口部件实现

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)

### 理想流水线实现

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)

## 气泡式流水线实现

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)

## 数据转发流水线实现

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)

## 动态分支预测机制实现

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)

# 实验过程与调试

## 测试用例和功能测试

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

### 测试用例1

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

### 测试用例1

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

## 可自行安排章节

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

## 性能分析

**分析不同方案时钟周期数差异**, XXXXX XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

## 主要故障与调试

### XXX故障

理想流水线： 接口处数据传输问题。

**故障现象：**执行halt指令时控制信号无法通过ID/EX接口。

**原因分析：**如图 4.1，寄存器设置为上升沿刷新，但当D端有输入且clk变化一个周期后寄存器中的数据仍然没有改变，检查时注意到此时ALUControl端输入的值为不确定，这是由于在控制器电路中未给halt指令相应的ALUControl值，为设计控制器时的失误。如图 4.1，寄存器设置为上升沿刷新，但当D端有输入且clk变化一个周期后寄存器中的数据仍然没有改变，检查时注意到此时**ALUControl**端输入的值为不确定，这是由于在控制器电路中未给halt指令相应的ALUControl值，为设计控制器时的失误。

**解决方案：**在控制器中给halt指令（OP为12）一个ALUControl信号0000一边让控制信号在接口处顺利传递。

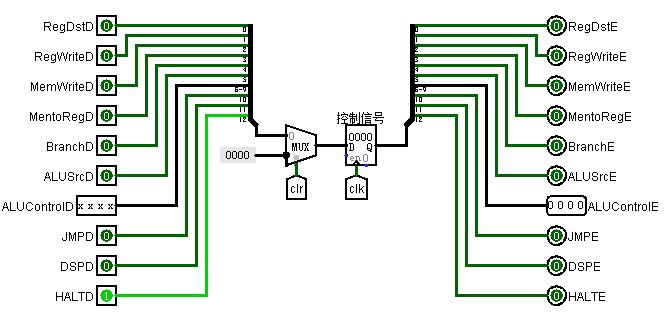


图 4.1无法向寄存器中写入数据示意图

### XXX故障

Verilog实现动态分支预测：状态机转移错误。

**故障现象：**如图 4.2所示，当判断信号judge为10b，即判断错误，predict\_f信号为10b，即判断指令不发生跳转，可判断指令最终发生了跳转，状态state[1]应向11b方向转移，即由01b转移至10b，可此时状态并未发生相应转移。

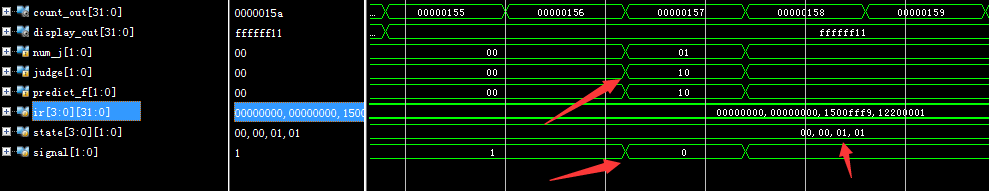


图 4.2程序故障图

**原因分析：**添加测试信号signal用来判断程序在此时进入了判断状态的case语句中的哪一个状态分支，结果显示，程序竟然进入了状态为00b的分支，再通过仔细分析可以得到，此模块的敏感变量表内为judge、predict\_t两个信号，在编写时采用的是电平触发，可能会产生毛刺，并最终导致状态机状态转移错误。

**解决方案：**修改动态分支预测器的敏感变量表，将此模块修改为时序控制，使用时钟的下降沿进行控制，在上一个时钟上升沿，分支指令进入EX段，并且判断出分支预测结果的正误，随后的时钟下降沿，使用判断结果作为依据进行状态机的状态转移，便可实现状态机状态的正确转移。

### XXX故障

Verilog实现动态分支预测：预测错误之后重新计算地址错误。

**故障现象：**如图 4.3，judge判断信号为10b，表示预测错误，nop信号，即分支跳转信号为0，表示分支指令没有发生跳转，此时情况为分支指令并未发生跳**转而**动态分支预测器预测指令会跳转，此时需要删去误取的指令，并将不跳转的地址重新计算，应为0x0000316c，然而此时计算出的不跳转地址为0x00003158，错误。

**原因分析：**分析地址计算模块发现代码编写错误。当预测器预测指令跳转而实际指令并未跳转时，应计算出此条分支指令的下一条指令的地址，即在此**分支指令**的地址基础上加上4，而之前的代码在计算时，直接在如今处于的IF段的指令的地址上加上4，必然导致程序出错。

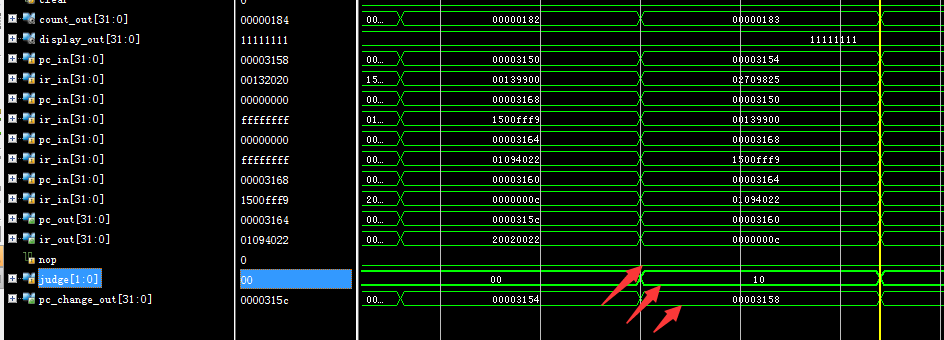


图 4.3程序故障图

**解决方案：**修改地址计算模块**，**当预测器预测指令跳转而实际指令并未跳转时，因为此时那条被判断的分支指令处于EX段，于是应当使用EX段指令的地址作为基础加上4，便可以得到修改错误预测之后的下一条指令的正确地址**。**

## 实验进度

表 4.1 课程设计进度表

| 时间 | 进度 |
| --- | --- |
| 第一天 | 复习组成原理CPU相关理论知识，阅读课设任务书，阅读MIPS指令手册，并列出CPU各部件的数据通路表，并完成数据通路的基本构建。 |
| 第二天 | 完成单周期CPU的控制信号表，使用Logisim搭建控制器，实现了单周期CPU并且通过了测试。完成部分Logism单周期CPU故障报告。 |
| 第三天 | 完成Logism单周期CPU的故障报告，并且通过了Logism单周期CPU的检查。使用Verilog实现了部分单周期CPU的重要部件，并通过仿真检查。 |
| 第四天 | 继续使用Verilog进行实现单周期CPU的工作，完成了所有部件的编写、控制器的编写，以及所有部件以及控制器的仿真测试，正在进行数据通路的拼接。 |
| 第五天 | 使用Verilog完成单周期CPU数据通路的连接，并且通过仿真测试。使用Verilog完成时钟分频以及七段数码管的代码编写，正在调试。 |
| 第六天 | 完成CPU电路的功能仿真和时序仿真，并成功将生成bit流烧入FPGA板内实现预计功能。 |
| 第七天 | 复习关于指令流水线的知识点，完成理想流水线的verilog代码，正在调试。 |
| 第八天 | 调试成功理想流水线verilog代码，并成功将bit流烧至FPGA板中。完成冒险处理中的数据冲突处理和分支处理代码编写，正在调试。 |
| 第九天 | 完成冒险处理中的数据冲突和分支处理，并成功烧入FPGA板内。完成数据重定向的Verilog代码的编写，正在调试。 |
| 第十天 | 完成数据重定向的Verilog代码并成功烧入FPGA板内。成功实现动态分支预测，预测成功率显著提高，并成功将代码烧入FPGA板内。 |

# 设计总结与心得

## 课设总结

基于对象的存储是为了克服当前基于块的存储存在的诸多难题，在存储接口和结构层次的重要发展。可以根据应用负载选择优化的存储策略。作了如下几点工作：

1. 完成方案总结（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）。
2. 功能总结（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）。
3. 其他需要总结的内容，（自行修订扩充）。

## 课设心得

本次课程设计可以说是迄今为止所有实验以及课程设计中难度最大的一门。两个星期从早到晚的不懈努力以及国庆节假期的辛苦加班才终于完成了整个课程设计的设计任务。现在再来回顾整个课程设计的整个过程，满满的成就感自是不用说，但是其中也有不少的细节值得我去深思与体会。

课程设计刚刚开始的时候，第一个任务是使用Logism设计单周期CPU，该任务和上学期实验课的任务类似，所以整个过程还算比较迅速。然而，在将该电路使用Verilog语言进行描述时，麻烦接踵而至。因为Logism对于电路是完全可视化的，连接过程清晰明了，但是使用Verilog语言进行数据通路时，因为各个部件的接口在定义时起名不是很规范，又因为只能使用各个wire型变量对于关键部件进行连接，而这些变量的名字在定义时同样没有做到一目了然，从而导致了我在完成了数据通路的连接后，使用了大量的时间对于整个电路进行排错和检查，最终还是将所有的接口和连线规范化的起了名字，才终于解决了问题。这也就说明了工程化、规范化对于电路设计或是程序设计的重要性。

紧接着，理想流水线CPU的设计并没有什么难度，但是使用插入气泡、数据重定向技术对于流水线CPU进行冒险处理时，因为这些方法书本上并没有，老师提供的PPT上也只有简单的一些描述，这就要求我不断地在网上搜索相关的知识内容，和小组内的成员进行相关探讨。随后的动态分支预测的设计难度更高，老师并没有提供任何相关的内容，这又使得我不断地去网上搜寻资料文献，阅读全英文的学术论文。从这一切也可以看出团队在一个工作中的重要性，以及自我学习能力的必要性。

然而对于本次课程设计，我还有一些小小的建议和改进。本次课程设计按照分组的机制，但是因为所有人所需要完成的工作相同，分组也只是更便于组内的同学在设计过程中进行互相交流，分组的作用并不是非常明显。分组内的同学们进度完全不同，且有些同学因为对于课程内容的掌握不是很好，逐渐边缘化，分组并不能调动所有同学加入到课程设计任务中。本课程设计的难度很大，但是时间又有一点不够充分，可以适当简化一下任务（并不是降低最终目标），使得在有限的时间内，在完成全部课程设计目标的基础上，适当减少任务中其中重复的繁杂的工作，以提高整个课程设计的效率。

最后在这里也感谢三位老师对于我在本次课程设计中无数问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。我相信组成原理课程设计必将成为我整个大学生涯中一段无比难忘的回忆。

# 参考文献

1. DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第4版).北京：机械工业出版社.
2. David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社
3. 谭志虎，秦磊华，吴非，肖亮.计算机组成原理. 北京：人民邮电出版社，2021年.
4. 谭志虎，周军龙，肖亮. 计算机组成原理实验指导与习题解析.北京：人民邮电出版社，2022.
5. 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011年.
6. 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008年.

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字: 姚晨炫** |