***2022***



**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： |  |
| 学 号： |  |
| 姓 名： | [作者] |
| 电 话： |  |
| 邮 件： |  |

目 录

[1 课程设计概述 1](#_Toc117855126)

[1.1 课设目的 1](#_Toc117855127)

[1.2 设计任务 1](#_Toc117855128)

[1.3 设计要求 1](#_Toc117855129)

[1.4 技术指标 2](#_Toc117855130)

[2 总体方案设计 4](#_Toc117855131)

[2.1 单周期CPU设计 4](#_Toc117855132)

[2.2 中断机制设计 8](#_Toc117855133)

[2.3 流水CPU设计 10](#_Toc117855134)

[2.4 气泡流水线设计 11](#_Toc117855135)

[2.5 重定向流水线设计 12](#_Toc117855136)

[2.6 基于重定向的动态分支预测机制 13](#_Toc117855137)

[3 详细设计与实现 16](#_Toc117855138)

[3.1 单周期CPU 实现 16](#_Toc117855139)

[3.2 中断机制实现 24](#_Toc117855140)

[3.3 流水CPU实现 28](#_Toc117855141)

[3.4 气泡式流水线实现 29](#_Toc117855142)

[3.5 重定向流水线实现 30](#_Toc117855143)

[3.6 动态分支预测机制实现 31](#_Toc117855144)

[4 实验过程与调试 33](#_Toc117855145)

[4.1 测试用例和功能测试 33](#_Toc117855146)

[4.2 性能分析 36](#_Toc117855147)

[4.3 主要故障与调试 36](#_Toc117855148)

[4.4 实验进度 37](#_Toc117855149)

[5 设计总结与心得 39](#_Toc117855150)

[5.1 课设总结 39](#_Toc117855151)

[5.2 课设心得 39](#_Toc117855152)

[参考文献 41](#_Toc117855153)

# 课程设计概述

## 课设目的

本课程力图以“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”。硬件综合训练课程是完成该计算机组成原理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指令集（指令集任选），具体见表 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，具体任务每位同学不一样，指令编号详见见公文包中的任务分配；
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，并能运行教师提供的标准测试程序，并自动统计执行周期数自动统计执行周期数
7. 能自动统计各类能自动统计各类无条件分支指令数目，分支指令数目，条件分支成功次数、条件分支成功次数、插入气泡数目插入气泡数目、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。动态分支预测流水线能自动统计预测成功与失败次数。

表 1.1 指令集

| **#** | **指令助记符** | **指令类型** | **简单功能描述** | **备注** |
| --- | --- | --- | --- | --- |
| 1 | ADD | R | 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 | I | 立即数加 |
| 3 | AND | R | 与 |
| 4 | ANDI | I | 立即数与 |
| 5 | SLLI | I | 逻辑左移 |
| 6 | SRAI | I | 算数右移 |
| 7 | SRLI | I | 逻辑右移 |
| 8 | SUB | R | 减 |
| 9 | OR | R | 或 |
| 10 | ORI | I | 立即数或 |
| 11 | XOR | I | 或非/立即数异或 |
| 12 | LW | I | 加载字 |
| 13 | SW | S | 存字 |
| 14 | BEQ | B | 相等跳转 |
| 15 | BNE | B | 不相等跳转 |
| 16 | SLT | R | 小于置数 |
| 17 | STLI | I | 小于立即数置数 |
| 18 | SLTU | R | 小于无符号数置数 |
| 19 | JAL | J | 转移并链接 |
| 20 | JALR | I | 转移到指定寄存器 |
| 21 | ECALL | I | 系统调用 | if ($a7==34) LED输出 $a0的值 else等待 Go按键暂停。  注意显示逻辑需要考虑如何锁存过去的数据，否则数据一闪而过。 |
| 22 | CSRRSI | I | 访问CSR寄存器 | 中断相关，可简化为开中断 |
| 23 | CSRRCI | I | 访问CSR寄存器 | 中断相关，可简化为关中断 |
| 24 | URET | I | 中断返回 | 清中断，mEPC送 PC，开中断 |
| 25 | SLL | R | 逻辑左移 | ccab扩展指令 |
| 26 | SLR | R | 逻辑右移 |
| 27 | LB | I | 加载字节 |
| 28 | BGE | B | 大于等于跳转 |

# 总体方案设计

## 单周期CPU设计

本阶段实现的是基于硬布线控制器的RISC-V单周期CPU，支持24条基本指令和4条扩展指令(SLL,SRL,LB,BGE)。

总体结构图如图 2.1所示。

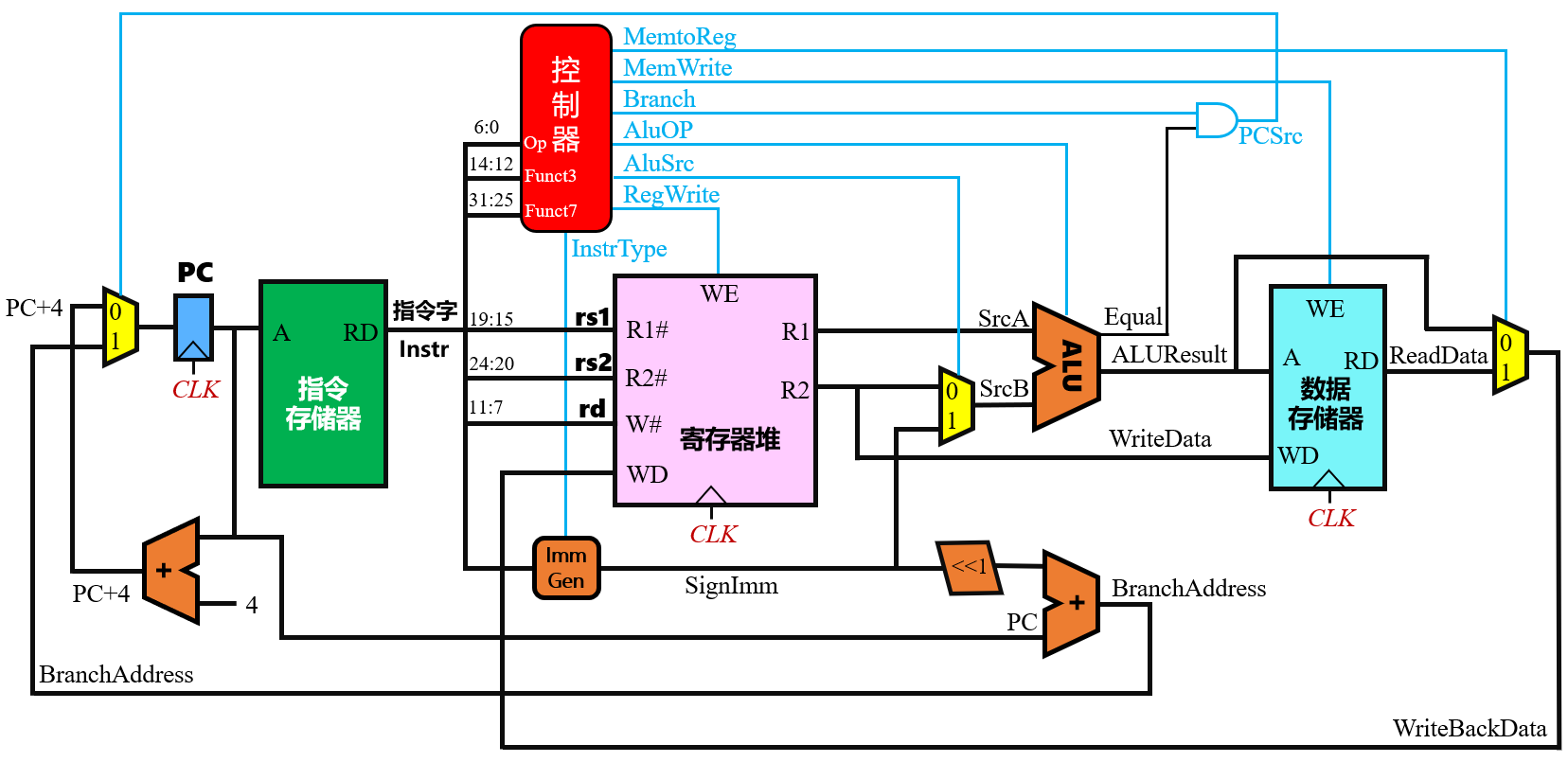


图 2.1 总体结构图

### 主要功能部件

#### 程序计数器PC

表 2.1程序计数器引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Data | 输入 | 32 | 下一条指令地址 |
| Enable | 输入 | 32 | 使能，~halt |
| Clock | 输入 | 1 | 时钟信号 |
| Clear | 输入 | 32 | 总复位 |
| Output | 输出 | 32 | 当前要运行的指令地址 |

#### 指令存储器IM

输入：PC的11~2位即指令的地址编号

输出：指令字IR

#### 运算器

表 2.2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 |
| --- | --- | --- | --- |
| a\_i | 输入 | 32 | 操作数a |
| b\_i | 输入 | 32 | 操作数b |
| 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 |
| Equal | 输出 | 1 | a\_i==b\_i |
| EG | 输出 | 1 | a\_i>=b\_i |

表 2.3运算器功能描述

| ALUOP | 功能 |
| --- | --- |
| 0000 | Result = X << Y 逻辑左移 （Y取低五位） Result2=0 |
| 0001 | Result = X >>>Y 算术右移 （Y取低五位） Result2=0 |
| 0010 | Result = X >> Y 逻辑右移 （Y取低五位） Result2=0 |
| 0011 | Result = (X \* Y)[31:0]; Result2 = (X \* Y)[63:32] 无符号乘法 |
| 0100 | Result = X/Y; Result2 = X%Y 无符号除法 |
| 0101 | Result = X + Y (Set OF/UOF) |
| 0110 | Result = X - Y (Set OF/UOF) |
| 0111 | Result = X & Y 按位与 |
| 1000 | Result = X | Y 按位或 |
| 1001 | Result = X⊕Y 按位异或 |
| 1010 | Result = ~(X |Y) 按位或非 |
| 1011 | Result = (X < Y) ? 1 : 0 符号比较 |
| 1100 | Result = (X < Y) ? 1 : 0 无符号比较 |

#### 寄存器堆RF

表 2.4寄存器引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| RDin | 输入 | 32 | 写入的数据 |
| WE | 输入 | 1 | 写使能 |
| W# | 输入 | 5 | 写入的寄存器编号 |
| R1# | 输入 | 5 | 读寄存器编号rs1 |
| R2# | 输入 | 5 | 读寄存器编号rs2 |
| clk | 输入 | 1 | 时钟信号 |
| R1 | 输出 | 1 | 寄存器rs1的数据 |
| R2 | 输出 | 1 | 寄存器rs2的数据 |

#### 数据储存器

表 2.5数据储存器引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| MDin | 输入 | 32 | 写入的数据 |
| Addr | 输入 | 10 | 地址 |
| Store | 输入 | 1 | 写使能 |
| clk | 输入 | 1 | 时钟信号 |
| rst | 输入 | 1 | 清零 |
| Dout | 输出 | 32 | 输出的数据 |

### 数据通路的设计

表 2.6指令系统数据通路框架

| 指令 | PC | IM | RF | | | | ALU | | | DM | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1# | R2# | W# | Din | A | B | OP | Addr | Din |
| R型 | PC+4 | PC | rs1 | rs2 | rd | ALU | R1 | R2 | 0-10 | R[1:0] | R2 |
| I型 | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] | 0-10 | R[1:0] | R2 |
| lw | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] |  | R[1:0] | R2 |
| sw | PC+4 | PC | rs1 | rs2 | rd | Dout | R1 | IR[31:25,11:7] |  | R[1:0] | R2 |
| J型 | PC+2IR[31,19:  12,20,30:21] | PC |  |  | rd | PC+4 | R1 | IR[31:20] |  | R[1:0] | R2 |
| B型 | PC+2IR[31,7,  30:25,11:8] | PC |  |  |  | ALU | R1 | R2 | 6,11,12 | R[1:0] | R2 |
| ecall | PC+4 | PC | 17 | 10 | rd | ALU | R1 | R2 |  | R[1:0] | R2 |

特殊情况：

对于I型指令jalr：PC=ALU

对于扩展指令中的lw型指令lb：RF的Din={24(R[7]),R[7:0]}

### 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 2.3。

表 2.7主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| RegWrite | 0-12 | 寄存器写使能 |
| MemWrite | 0-1 | 写内存控制信号 |
| AluOP | 0-1 | 运算器操作控制符（4位） |
| MemToReg | 0-1 | 寄存器写入数据来自存储器 |
| S\_Type | 0-1 | S型指令译码信号 |
| AluSrcB | 0-1 | 运算器B输入选择 |
| JALR | 0-1 | JALR指令译码信号 |
| JAL | 0-1 | JAL指令译码信号 |
| Beq | 0-1 | Beq指令译码信号 |
| Bne | 0-1 | Bne指令译码信号 |
| ecall | 0-1 | ecall指令译码信号 |
| LB | 0-1 | lb指令译码信号 |
| BGE | 0-1 | bge指令译码信号 |

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图 2.2所示。

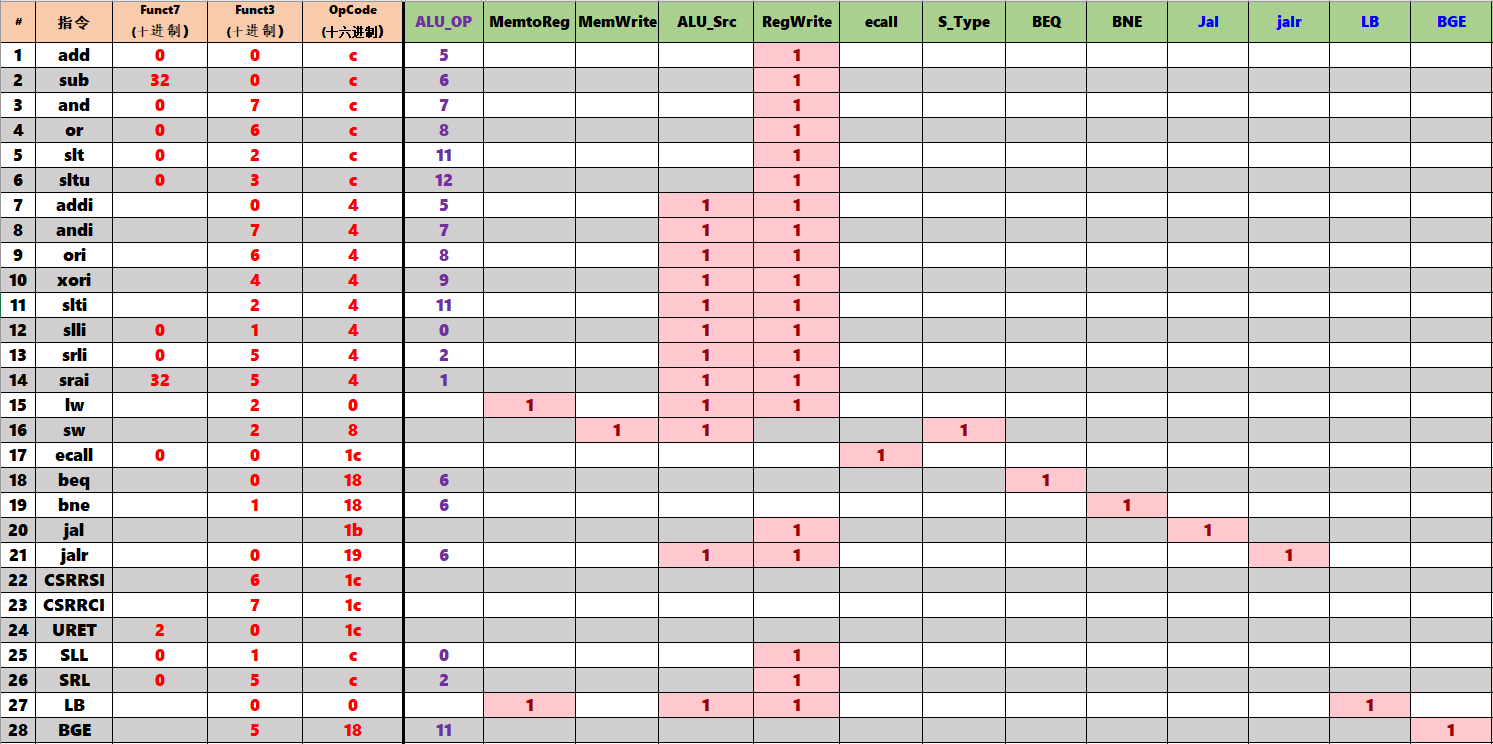


图 2.2 控制信号表框架

## 中断机制设计

### 总体设计

本次实验提供3个中断程序，优先级1<2<3

1. 中断屏蔽寄存器提供中断信号。
2. 有中断信号且中断使能IE打开时，发出中断请求INT。将入口送到PC，保存现场到mEPC，关闭IE，进入中断程序。
3. 单级中断：如果中断信号出现时其他中断程序正在执行，等待其结束，再响应正在等待的优先级最高的中断程序。

多级中断：如果中断请求时出现时比其优先级高的中断程序正在执行，等待其结束，再响应正在等待的优先级最高的中断程序。否则中断正在执行的中断程序，响应新的中断程序。需要实现开关中断的功能。

1. 返回时从mEPC中取出断点送到PC，清除当前中断信号，打开IE。多级中断返回时需要还恢复上一级的中断号。

### 硬件设计

一、单级中断：

1. 中断按键信号采样电路：使用任务书提供的参考电路
2. mEPC寄存器：响应中断时保存现场
3. 优先编码器：判断中断优先级
4. 多路选择器：选择中断入口。
5. now寄存器：保存当前正在运行的中断程序。

二、多级中断：在单机中断的基础上更改和增加部分硬件

1. mEPC寄存器有3个，分别保存三级中断的断点
2. 多级中断返回原来执行的程序时不仅要还原现场的指令地址，还要还原上一级中断号。为了实现这个功能，需要增加以下硬件：

（1）计数器：记录当前中断级数。

（2）2个中断号寄存器：当响应优先级更高的中断时，保存上一级中断的中断号，返回时送给now寄存器。

（3）last寄存器：保存上一级中断的中断号。

1. 3个中断程序都设置中断使能。当接收到csrrisi指令时，把当前中断号对应的中断使能打开，接收到csrrci则关闭。所以一共有4个使能，通过多路选择器来选择当前的使能。

三、重定向单级中断：将重定向流水线与单级中断结合，在流水线上做修改：

1. 在IF阶段就需要判断当前指令是否为中断返回指令，如果是，将中断返回地址送入PC中。不能用ID或EX阶段的URET指令，否则不能正确返回原来执行的程序。

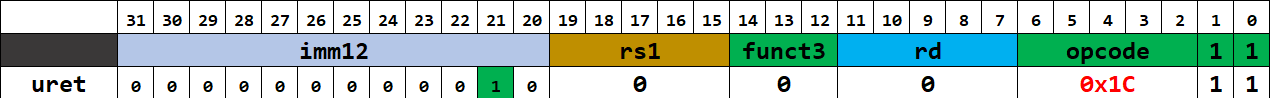


图 . URET指令格式

1. 当接收到中断请求时，将EX阶段的指令清零。保存EX阶段的PC作为中断返回地址。
2. 当收到中断请求，且EX阶段不是返回指令时，将返回地址送到PC。如果EX阶段恰好是返回指令，这条返回指令的地址将会作为现场保存在mEPC寄存器中，中断响应结束后执行这条指令，变成死循环，所以EX阶段不是返回指令时才能进行中断响应。

### 软件设计

1. 因为硬件设计已经考虑了现场的保存，在软件中就无需考虑。
2. 用RARS打开测试文件，编译后可以找到三个中断程序的入口。
3. 多级中断增加了csrrsi和csrrci指令，用于在中断响应过程中开关中断。

## 流水CPU设计

### 总体设计

指令执行过程分为5个阶段：取指令阶段（IF）、译码取数阶段（ID）、指令执行阶段（EX）、访存阶段（MEM）、写回阶段（WB）。在前4个执行阶段后面增加流水寄存器，用于将数据传递到下一阶段并锁存本段处理完成的数据，以保证本段的执行结果能在下一个时钟周期给下一阶段使用。所有寄存器、计数器采用统一的时钟进行同步。各段功能可独立并行工作。

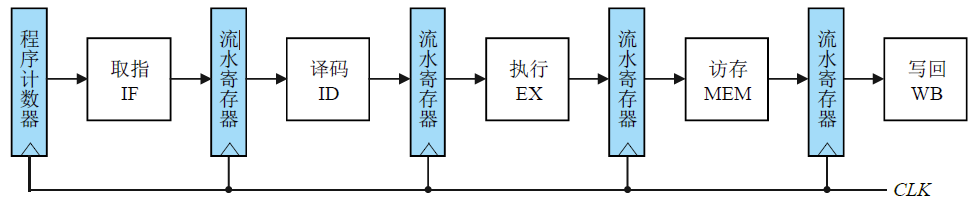


图 2.4 流水线示意图

### 流水接口部件设计

流水接口部件用于分割流水阶段，使用寄存器暂存数据并同时向下一阶段传递数据。流水接口支持同步清零和同步使能控制，每一个接口部件都按统一的时钟clk同步更新。

### 理想流水线设计

1. 所有跳转指令在EX阶段完成地址计算，流水也在EX阶段完成跳转。
2. RF设置为下降沿触发，以解决译码和写回阶段可能存在的结构冲突。
3. RF写回操作安排在WB阶段，控制信号和写入的数据都来自WB阶段。
4. LedData的输出由WB阶段的指令控制。

## 气泡流水线设计

### 总体设计

气泡流水线用流水阻塞和插入气泡的方式解决分支冲突和数据冲突。

### 分支冲突处理

当检测到分支跳转成功信号时，将ID/IF和IF/EX流水寄存器清零。

### 数据冲突处理

处理数据冲突需要增加寄存器使用判断和数据相关判断逻辑。



图 2.5 寄存器使用判断逻辑

数据冲突有以下两种可能：

1. 当在EX阶段和MEM阶段出现读操作和写操作的目的寄存器为同一寄存器
2. 执行ecall指令时，对10和17号寄存器进行写操作。

产生数据冲突时，暂停流水线，并将ID/EX流水寄存器清零。

表 2.8气泡逻辑引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| EX.RegWrite | 输入 | 1 |  |
| MEM. RegWrite | 输入 | 1 |  |
| ecall | 输入 | 1 | ID阶段的ecall指令 |
| BranchTaken | 输入 | 1 | 高电平分支跳转成功 |
| EX.rd | 输入 | 5 |  |
| MEM.rd | 输入 | 5 |  |
| ID.IR | 输入 | 32 |  |
| stall | 输出 | 1 | 流水线暂停信号 |
| flush | 输出 | 1 | ID/EX插入气泡 |

## 重定向流水线设计

### 总体设计

重定向流水线在产生数据冲突时，将MEM和WB阶段的正确数据覆盖EX段的错误数据，无需插入气泡，可以大大优化流水线的性能。需要解决的问题分为分支冲突和数据冲突，其中数据冲突的访问指令需要另外处理

### 冲突处理

分支冲突的处理与气泡流水线相同。

寄存器使用逻辑和气泡流水线相同。

数据冲突有以下两种可能：

1. 当在EX阶段和MEM阶段出现读操作和写操作的目的寄存器为同一寄存器
2. 或当执行ecall指令时对10和17号寄存器进行写操作时。在产生数据冲突时，暂停流水线，并将ID/EX流水寄存器同步清零。ecall指令的冲突又分为以下两种：rs1=17且进行写操作；rs2=10且进行写操作。

### 重定向设计

重定向选择分为以下三种：

1. EX段产生数据冲突：EX段的R1和R2重定向为MEM段的ALU输出。
2. MEM段产生数据冲突且EX段不产生数据冲突：EX段的R1和R2重定向为WB段的RDin，即写入寄存器的数据。
3. EX段存在数据冲突且EX段指令是访存指令(即LoadUse)，不能采用重定向方式，需要暂停流水线，在将ID/EX流水寄存器清零，插入气泡来消除冲突。

表 2.9重定向逻辑引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| EX.RegWrite | 输入 | 1 | EX阶段的Regwrite指令 |
| MEM. RegWrite | 输入 | 1 | MEM阶段的Regwrite指令 |
| ecall | 输入 | 1 | ID阶段的ecall指令 |
| BranchTaken | 输入 | 1 | 高电平分支跳转成功 |
| EX.rd | 输入 | 5 | EX阶段的rd |
| MEM.rd | 输入 | 5 | MEM阶段的rd |
| EX.MemToReg | 输入 | 1 | EX阶段的MemToReg指令 |
| ID.IR | 输入 | 32 | ID阶段的IR |
| stall | 输出 | 1 | 流水线暂停信号 |
| flush | 输出 | 1 | ID/EX插入气泡 |
| rs1.Fwd | 输出 | 2 | R1的重定向选择序号 |
| rs2.Fwd | 输出 | 2 | R2的重定向选择序号 |

## 基于重定向的动态分支预测机制

### 总体设计

动态分支预测依据分支指令的分支跳转历史，不断的对预测策略进行动态调整。每一条分支指令执行时，将分支指令地址、分支目标地址、是否发生跳转等信息送入BTB表，BTB以分支指令地址为关键字，在BTB表内进行全相联并发比较，如果数据缺失，表示当前分支指令不在 BTB表中，需要将该分支指令的相关信息载入，并设置合适的分支预测历史位初值，以方便后续预测注意载入过程中可能涉及到淘汰。如果数据命中，表明当前分支指令历史分支信息已存放在BTB表中，此时需要根据本次分支是否发生跳转的信息调整对应表项中的分支预测历史位，以提升预测准确率，并且处理与淘汰相关的置换标记信息即可。

### 状态机设计

动态分支预测根据跳转指令的历史跳转信息预测是否要跳转。该逻辑使用用双位预测状态机来实现。状态转换图如图 2.6所示。

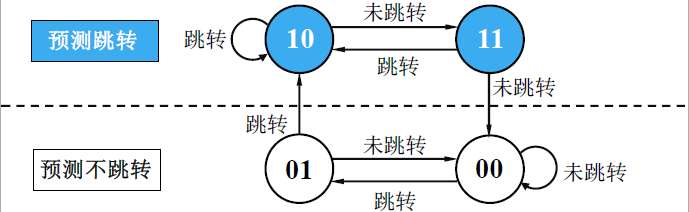


图 2.6 双预测位状态转换图

根据状态图，得到真值表



图 . 跳转预测状态机真值表

### cache设计

Cache用于保存历史跳转信息，采用八路全相连映射，使用LRU策略进行更新。若分支指令的地址在cache中命中，读取对应cache中的跳转地址作为预测的跳转地址。若未命中，在空的寄存器或在最久未访问的寄存器中写入跳转地址。

### 跳转逻辑设计

跳转的判断和执行需要对两个阶段进行处理：

1. IF阶段：在cache中查找当前PC的分支历史记录，如果命中，根据当前预测状态决定下一条指令地址是否要取目标跳转地址。
2. EX阶段：比较正确的跳转地址和IF阶段的预测信息判断是否预测成功。如果成功，跳转并更新 BTB中对应行的状态信息。如果预测失败，在ID/EX流水寄存器插入气泡，并在 IF阶段执行原来的指令。状态机的状态位随预测结果更新。

表 2.10分支预测逻辑引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| clk | 输入 | 1 | 同步时钟信号 |
| Branch | 输入 | 1 | 分支指令 |
| BranchTaken | 输入 | 1 | 分支成功 |
| BranchTaken | 输入 | 1 | 高电平分支跳转成功 |
| IF.PC | 输入 | 32 |  |
| EX.PC | 输入 | 32 |  |
| BranchAddr | 输入 | 32 | 分支跳转地址 |
| JumpAddr | 输出 | 32 | 预测的跳转地址 |
| PredictJump | 输出 | 1 | 是否预测跳转 |

# 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，将此控制信号通过非门取反之后和时钟相与，当需要进行停机时，Halt控制信号为1，经过非门之后为0，与时钟信号相与，屏蔽时钟信号，使整个电路停机。如图 3.1所示。

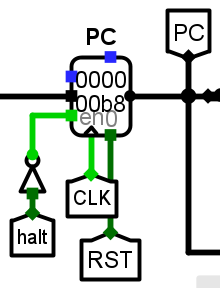


图 3.1程序计数器（PC）

1. FPGA实现：

程序计数器PC的Verilog代码如下：

initial begin

Q <= 0;

end

always @(posedge clk or posedge rst) begin

if (rst) Q <= 0;

else if (~halt&start) Q <= D;

end

1. 指令存储器（IM）
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。

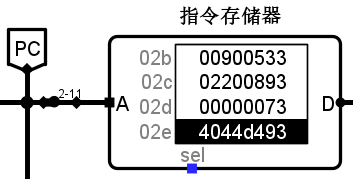


图 3.2指令存储器（IM）

1. FPGA实现：

使用logisim转verilog工具转出rom。

指令存储器IM的Verilog代码如下：

always @ (Address)

begin

case(Address)

0 : Data = 1049747;

…

default : Data = 0;

endcase

end

1. 运算器(ALU)
2. Logism实现：

采用提供的预先实现好的寄存器

1. FPGA实现：

运算器ALU的Verilog代码如下：

always@(a\_i or b\_i)

begin

case(alu\_op[3:0])

4'd0: result<= a\_i<<b\_i[4:0];

4'd1: result<= a\_i[31]?~(32'hffffffff>>b\_i[4:0])|(a\_i>>b\_i[4:0])

: a\_i>>b\_i[4:0];

4'd2: result<= a\_i>>b\_i[4:0];

4'd5: result<= a\_i+b\_i;

4'd6: result<= a\_i-b\_i;

4'd7: result<= a\_i&b\_i;

4'd8: result<= a\_i|b\_i;

4'd9: result<= a\_i^b\_i;

4'd10: result<= ~(a\_i|b\_i);

4'd11: result<= a\_i[31]>b\_i[31] || ~a\_i[31]&&~b\_i[31]&&a\_i<b\_i

|| a\_i[31]&&b\_i[31]&&a\_i>b\_i;

4'd12: result<= (a\_i<b\_i)?1 : 0;

default: result<=0;

endcase

E<=(a\_i==b\_i)?1 : 0;

GE<=(a\_i[31]==0&&b\_i[31]==0&&a\_i<b\_i)?0 : (a\_i[31]==1&&b\_i[31]

==1&&a\_i>b\_i)?0 : (a\_i[31]==1&&b\_i[31]==0)?0 : 1;

end

1. 寄存器堆(RF)
2. Logism实现：

采用cs3410.jar文件中预先实现好的寄存器组

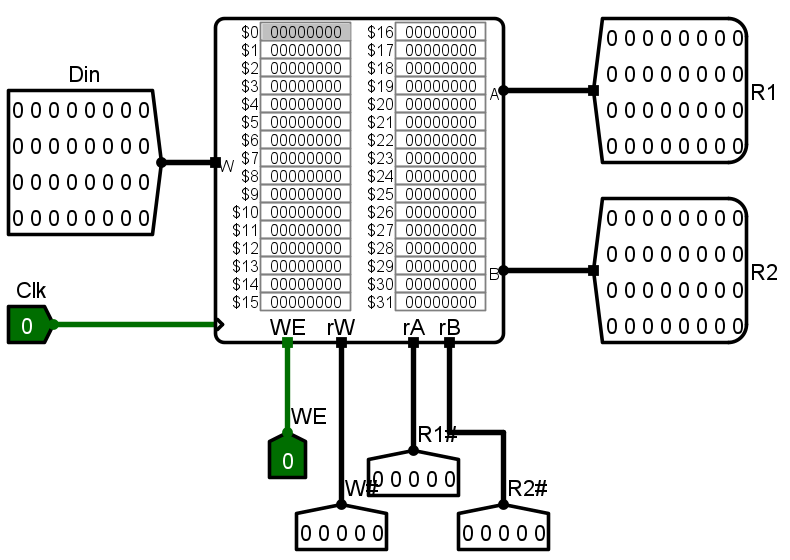


图 3.3指令存储器（RF）

1. FPGA实现：

注意要先将寄存器初始化为0，否则输出为z。

寄存器堆RF的Verilog代码如下：

initial begin

for(k=0;k<32;k=k+1)

regifile[k]=0;

read\_reg1\_data <= 0;

read\_reg2\_data <= 0;

end

always @(posedge clk) begin

if(write\_read\_ena && write\_reg\_addr) begin

regifile[write\_reg\_addr] <= data\_in;

read\_reg1\_data <= regifile[read\_reg1\_addr];

read\_reg2\_data <= regifile[read\_reg2\_addr];

end

1. 数据储存器(DM)
2. Logism实现：

使用一个存储器RAM实现指令存储器（DM）。设置该存储器的地址位宽为10位，数据位宽为32位。如图 3.4所示。

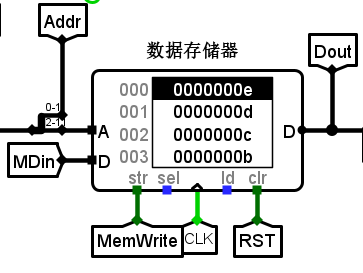


图 3.4指令存储器（RF）

1. FPGA实现：

使用logisim转verilog工具转出ram。注意要先将储存器初始化为0，否则输出为z。

数据储存器DM的Verilog代码如下：

reg [31:0] mem[1023:0];

integer k;

initial begin

for(k=0;k<1024;k=k+1)

mem[k]=0;

end

always @ (posedge clk) begin

if (we)

mem[addr] <= d;

end

assign q = mem[addr];

### 数据通路的实现

本次课程设计采用的工程化的设计模式，一次性构建所有的数据通路。主要实现方法为，对于每一条指令，将其改写成RTL（Register Transfer Level），忽略控制类信号，仅保留数据类信号，根据RTL功能填写对应指令的数据通路表，描述五大部件之间的连接关系，记录各部件输入端数据来源。

根据总体方案设计中数据通路设计那一小节的详细内容，具体分析每一条指令在执行过程中各个主要部件的输入和输出端口的连接，完成指令系统数据通路表的填写，如表 3.1所示。

表 3.1指令系统数据通路表

| 指令 | PC | IM | RF | | | | ALU | | | DM | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1# | R2# | W# | Din | A | B | OP | Addr | Din |
| add | PC+4 | PC | rs1 | rs2 | rd | ALU | R1 | R2 | 5 | R[1:0] | R2 |
| sub | PC+4 | PC | rs1 | rs2 | rd | ALU | R1 | R2 | 6 | R[1:0] | R2 |
| and | PC+4 | PC | rs1 | rs2 | rd | ALU | R1 | R2 | 7 | R[1:0] | R2 |
| or | PC+4 | PC | rs1 | rs2 | rd | ALU | R1 | R2 | 8 | R[1:0] | R2 |
| slt | PC+4 | PC | rs1 | rs2 | rd | ALU | R1 | R2 | 11 | R[1:0] | R2 |
| sltu | PC+4 | PC | rs1 | rs2 | rd | ALU | R1 | R2 | 12 | R[1:0] | R2 |
| addi | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] | 5 | R[1:0] | R2 |
| andi | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] | 7 | R[1:0] | R2 |
| ori | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] | 8 | R[1:0] | R2 |
| xori | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] | 9 | R[1:0] | R2 |
| slti | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] | 11 | R[1:0] | R2 |
| slli | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] | 0 | R[1:0] | R2 |
| srli | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] | 2 | R[1:0] | R2 |
| srai | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] | 1 | R[1:0] | R2 |
| lw | PC+4 | PC | rs1 |  | rd | ALU | R1 | IR[31:20] |  | R[1:0] | R2 |
| sw | PC+4 | PC | rs1 | rs2 | rd | Dout | R1 | IR[31:25,11:7] |  | R[1:0] | R2 |
| ecall | PC+4 | PC | 17 | 10 | rd | ALU | R1 | R2 |  | R[1:0] | R2 |
| beq | PC+2\*IR[31,  7,30:25,11:8] | PC |  |  |  | ALU | R1 | R2 | 6 | R[1:0] | R2 |
| bne | PC+2\*IR[31,  7,30:25,11:8] | PC |  |  |  | ALU | R1 | R2 | 6 | R[1:0] | R2 |
| jal | PC+2\*IR[31,19:12,20,30:21] | PC |  |  | rd | PC+4 | R1 | IR[31:20] |  | R[1:0] | R2 |
| jalr | ALU | ALU | rs1 |  | rd | ALU | R1 | IR[31:20] | 6 | R[1:0] | R2 |
| uret | RET\_PC | PC |  |  |  |  |  |  |  |  |  |
| sll | PC+4 | PC | rs1 | rs2 | rd | ALU | R1 | R2 | 0 | R[1:0] | R2 |
| srl | PC+4 | PC | rs1 | rs2 | rd | ALU | R1 | R2 | 2 | R[1:0] | R2 |
| lb | PC+4 | PC | rs1 |  | rd | {24(R[7]),R[7:0]} | R1 | IR[31:20] |  | R[1:0] | R2 |
| bge | PC+2\*IR[31,7,30:25,11:8] | PC |  |  |  | ALU | R1 | R2 | 11 | R[1:0] | R2 |

在完成指令系统数据通路表的填写之后，根据列出的数据通路表，进行多指令数据通路的合并输入数，表，将各个主要功能部件进行连接，根据数据通路合并表的最终结果，对于所有的多输入部件使用多路选择器进行输入选择。最终便可以完成数据通路的搭建。

下一PC地址的逻辑实现。Branch表示B型指令跳转成功

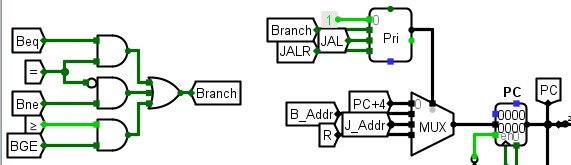


图 3.5 下址实现

RDin选择逻辑实现

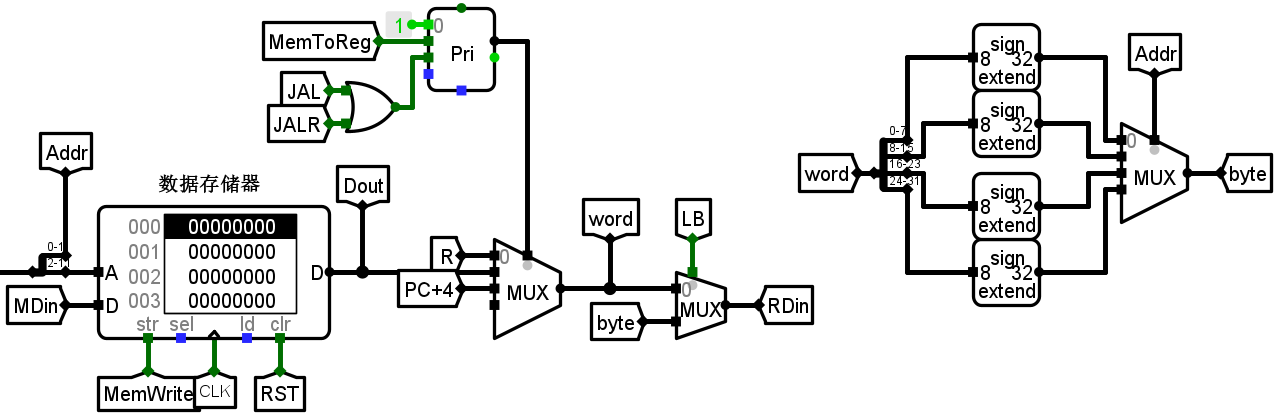


图 3.6 RDin数据选择

Led输出和周期计数实现

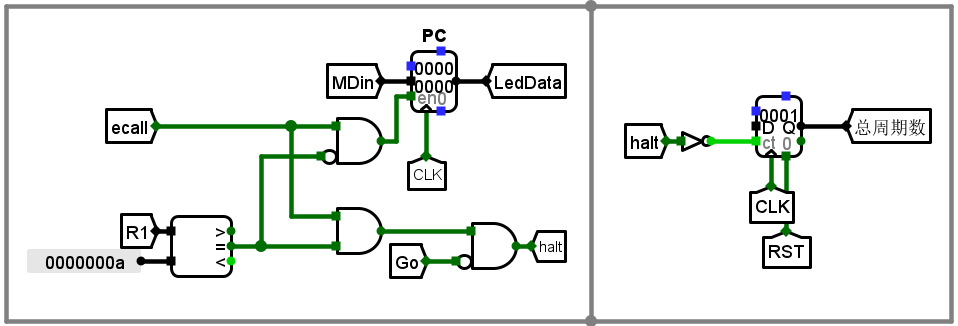


图 . Led输出和周期计数

总数据通路结构图如图 3.8所示

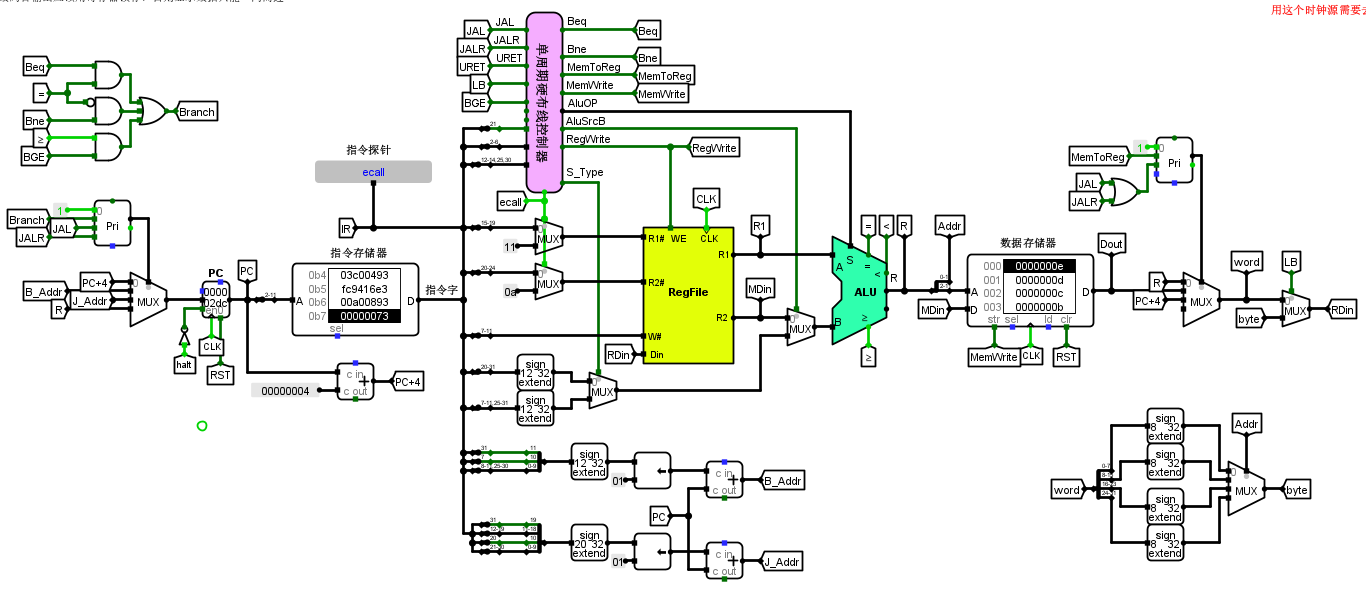


图 3.8 单周期CPU数据通路（Logism）

在Vivado中使用Verilog语言搭建的数据通路的原理图如图 3.9所示。

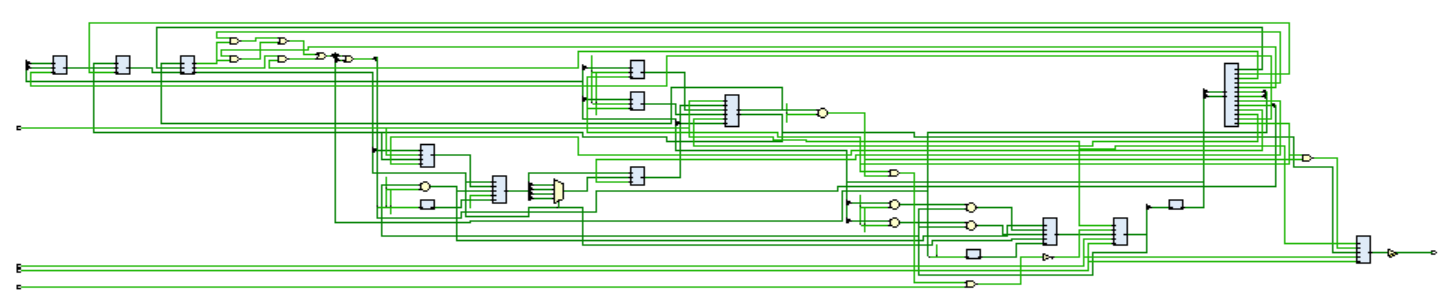


图 3.9单周期CPU数据通路（FPGA）

### 控制器的实现

1. Logism实现

填写图2.1 所示的控制信息表，使用Excel的预先定义好的函数得到相应电路图的规范表达式，使用Logisim 的自动生成电路功能完成电路，最终实现的控制器如图3.7 所示。

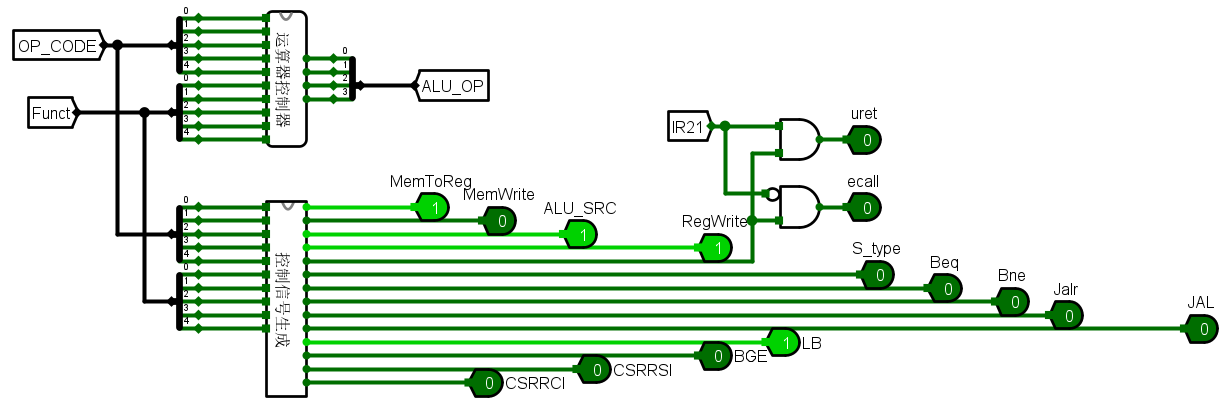


图 . 硬布线控制器

1. FPGA实现

根据在表格中得到的各个控制信号，使用case语句对指令进行分类。代码过于冗长，故只取R型指令的代码举例如下：

always@(\*) begin

{ALU\_SRC,BGE,Beq,Bne,JAL,Jalr,LB,MemToReg,MemWrite,RegWrite,S\_type,ecall}

= 12'b0;

ALU\_OP <= 4'd0;

case(OP[4:0])

5'hc:begin

RegWrite<=1'b1;

case(Funct[2:0])

3'd0:begin

case(Funct[4])

1'd0: ALU\_OP<=4'd5;

1'd1: ALU\_OP<=4'd6;

endcase

end

3'd7: ALU\_OP<=4'd7;

…

3'd5: ALU\_OP<=4'd2;

endcase

end

endcase

end

以此类推，最终便可以实现整个主控制器中所有控制信号的生成。

## 中断机制实现

只截取中断逻辑部分进行说明。

所有寄存器和触发器未说明的均为上升沿。now寄存器的数据为当前正在运行的中断程序的中断号。last寄存器的数据为上一级中断号。(没有执行的中断号就为0)

### 单级中断实现

中断按键信号采样电路采用提供的参考电路。

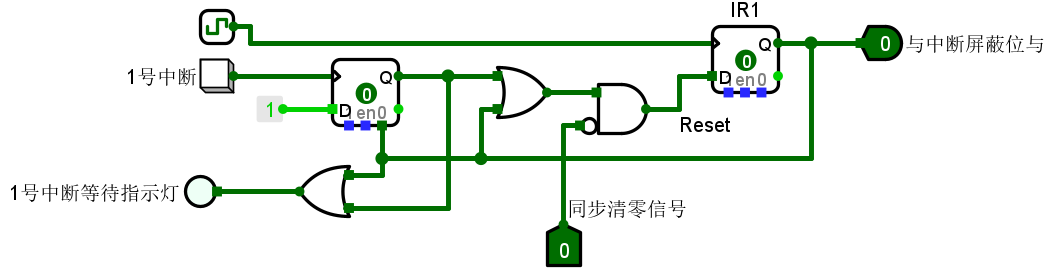


图 . 中断信号采样电路

中断使能和中断请求的逻辑实现如下图。

发出中断请求后的下一个时钟周期关闭中断使能，接收到返回指令后的下一个时钟周期打开中断使能。当有中断信号且中断使能打开时，发出中断请求。

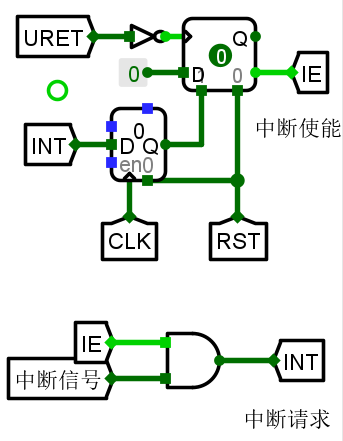


图 . 中断使能和中断请求逻辑

发出中断请求的下一时钟周期，now寄存器更新。收到返回指令时将当前中断信号清除。

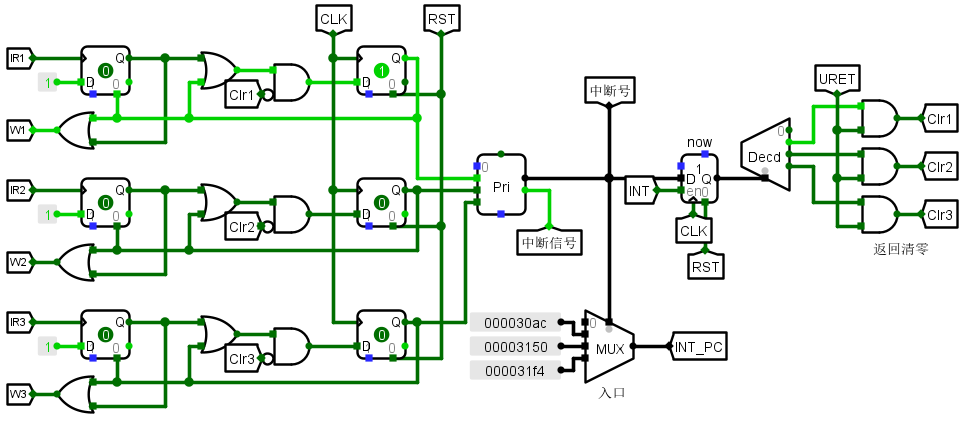


图 . 中断入口、中断号更新

mEPC寄存器为下降沿。在中断使能关闭后将现场保存到寄存器中。不能是上升沿，否则保存的地址的指令是最后一条已经执行过的指令。

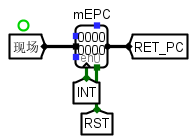


图 . 保存现场

中断入口和返回地址送入PC的逻辑实现如图所示

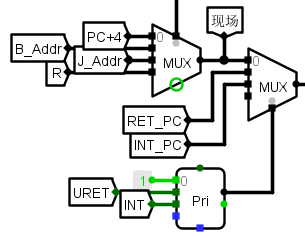


图 . 中断入口和返回地址的送入

### 多级中断实现

为每个中断程序都设置使能，开关中断指令只对当前执行的中断号的使能有效。

当中断信号和now相等时，不开中断，防止中断一直调用本身。

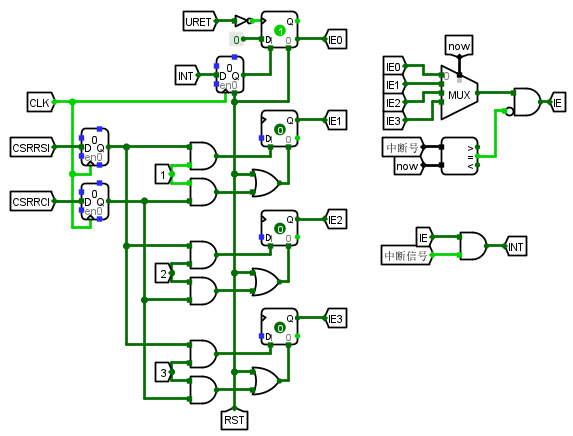


图 . 中断使能和中断请求

计数器计算中断级数，响应新一级中断时加1，返回时减1。

因为最多只有三级中断，所以用2个寄存器来保存上一级中断号。响应新一级中断时将上一级中断号保存在寄存器中。返回时将上一级中断号送给last寄存器。

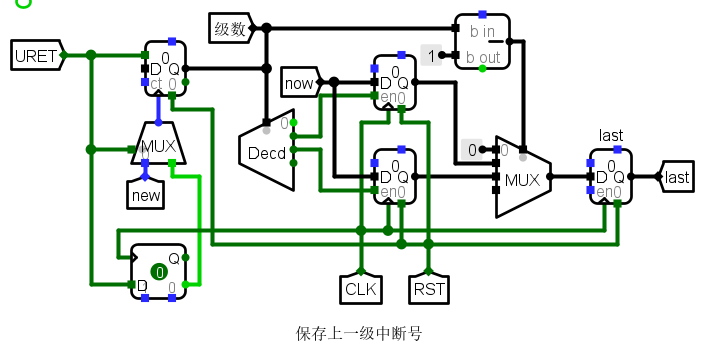


图 . 保存上一级中断号

now寄存器的更新逻辑在单级中断的基础上做了修改。当新的中断号优先级高于now时，new变为高电平，响应新的中断程序。当中断程序返回时，now更新为上一级中断号last。

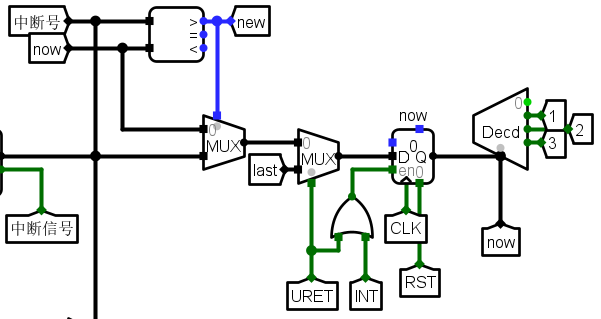


图 . 优先级更高的中断响应和返回

多级中断的3个mEPC都为下降沿，分别保存三个中断的现场。

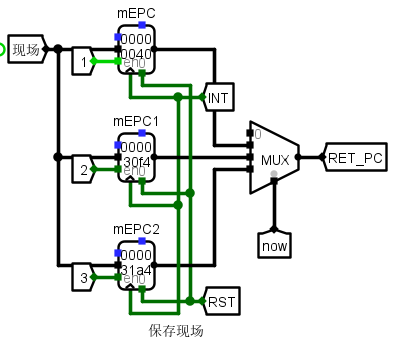


图 . 保存现场

### 重定向流水线单级中断实现

只截取了在重定向和中断逻辑修改的部分进行说明。

收到中断请求时，EX阶段指令清零，一个时钟周期后将清零信号恢复。

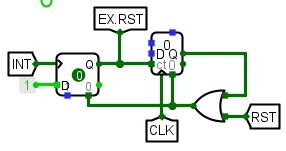


图 . EX阶段清零逻辑

当收到中断请求，且EX阶段不是返回指令时，将返回地址送到PC。

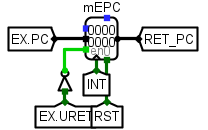


图 . 保存现场

在IF阶段，根据RISC-V的指令格式确定指令是否为URET。

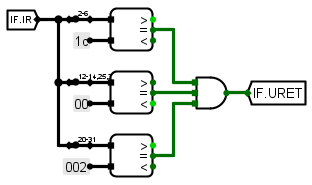


图 . URET指令判断

## 流水CPU实现

### 流水接口部件实现

下图为一个接口的logisim实现，支持同步清零、同步使能控制和时钟上升沿同步更新。其他接口的流水寄存器原理相同，不再贴图。

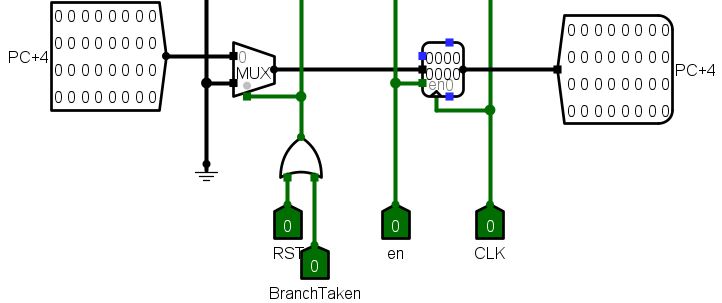


图 . 流水线接口

### 理想流水线实现

在单周期的基础上插入流水寄存器，并对数据通路做修改

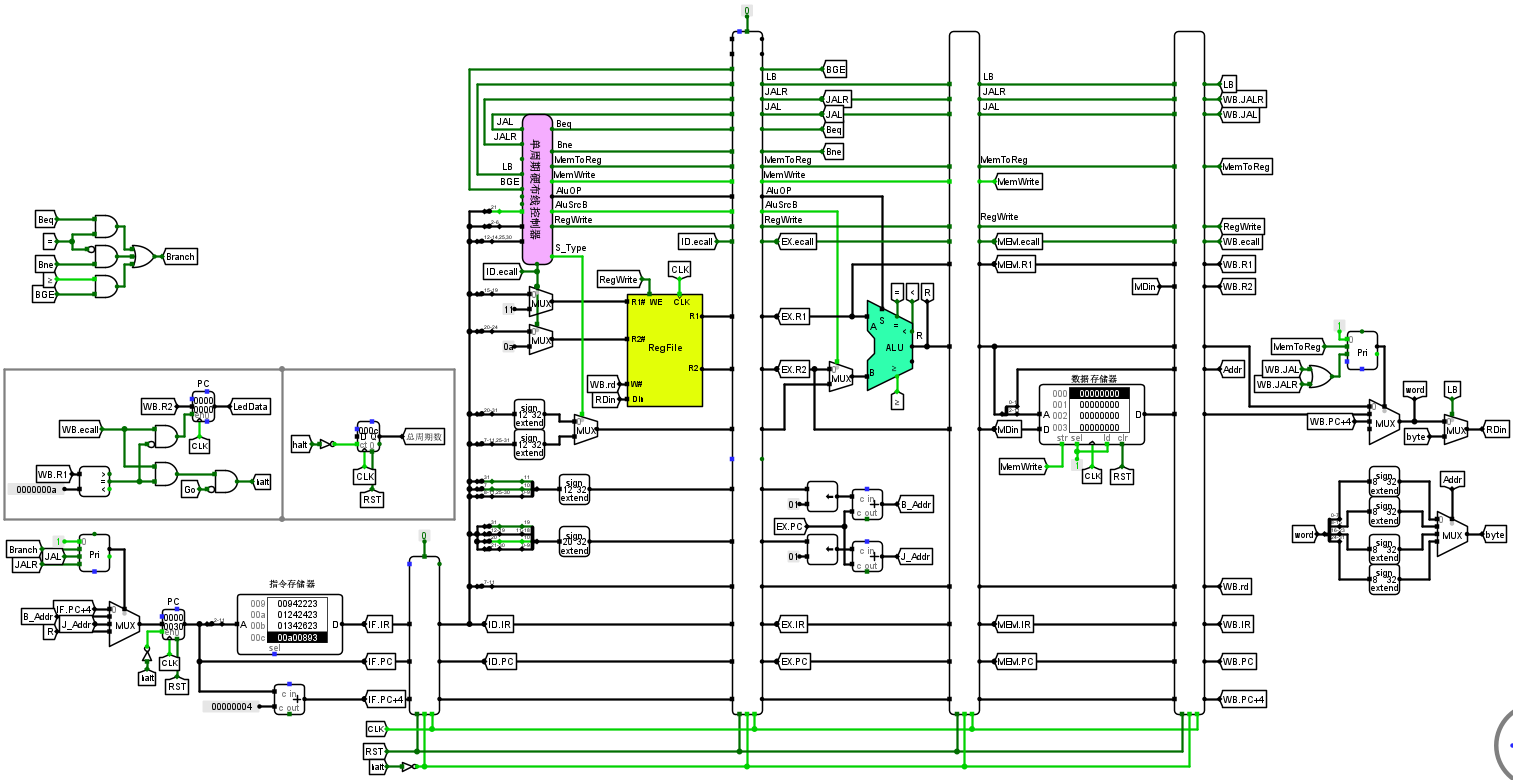


图 . 理想流水线实现

## 气泡式流水线实现

只截取气泡相关部分进行说明。

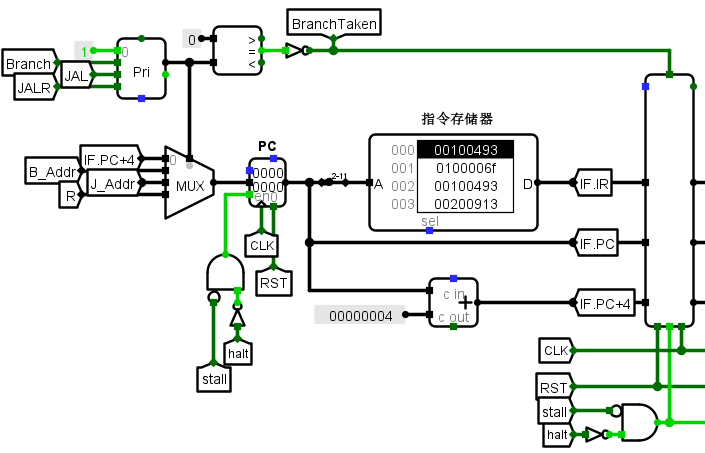


图 . IF/ID处理

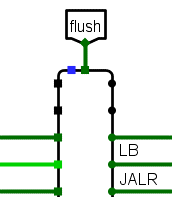


图 . ID/EX处理

分支指令成功时，BranchTaken和flush为高电平，IF/ID和ID/EX接口清零。

数据冲突时，stall和flush为高电平，PC计数器和流水线暂停，ID/EX接口清零。

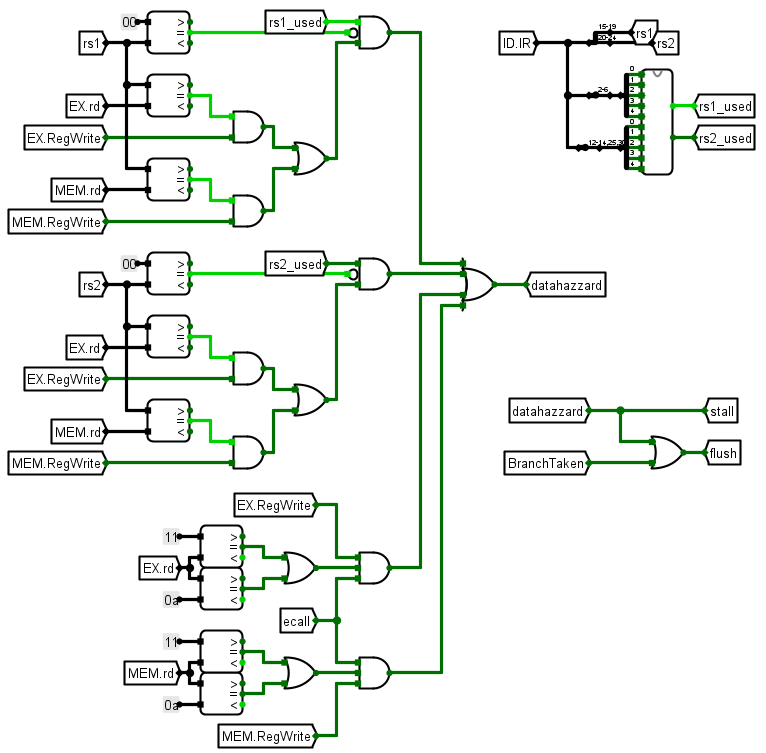


图 . 气泡逻辑实现

## 重定向流水线实现

只截取重定向相关部分进行说明。

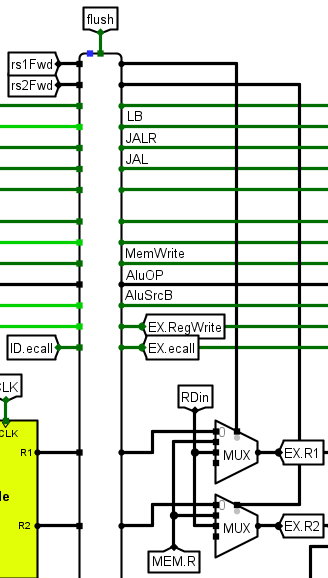


图 . 重定向选择实现

rs1Fwd和rs2Rwd为重定向选择信号。选择逻辑不再赘述。

分支指令成功时，BranchTaken和flush为高电平，IF/ID和ID/EX接口清零。

数据冲突且访存相关时，stall和flush为高电平，PC计数器和流水线暂停，ID/EX接口清零。

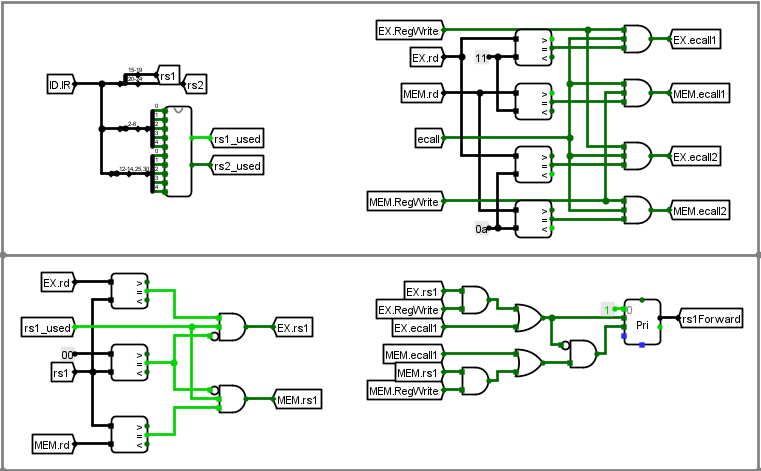


图 3.29 重定向逻辑实现1

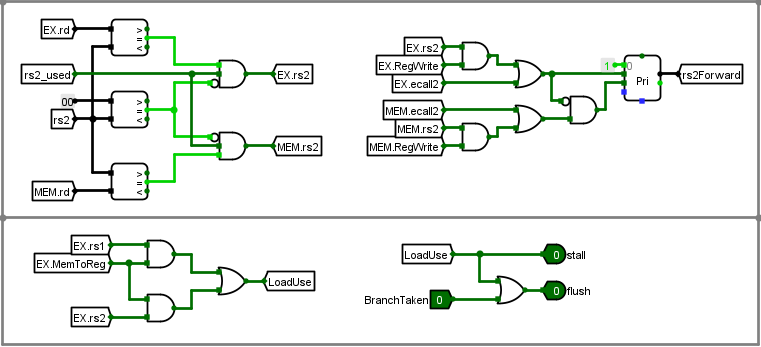


图 . 重定向逻辑实现2

## 动态分支预测机制实现

只截取了分支预测相关部分。

当收到分支指令时，BTB逻辑查询跳转地址，并判断是否跳转。

如果EX阶段的跳转判断和真正的分支跳转不同，则判断错误，error为高电平。

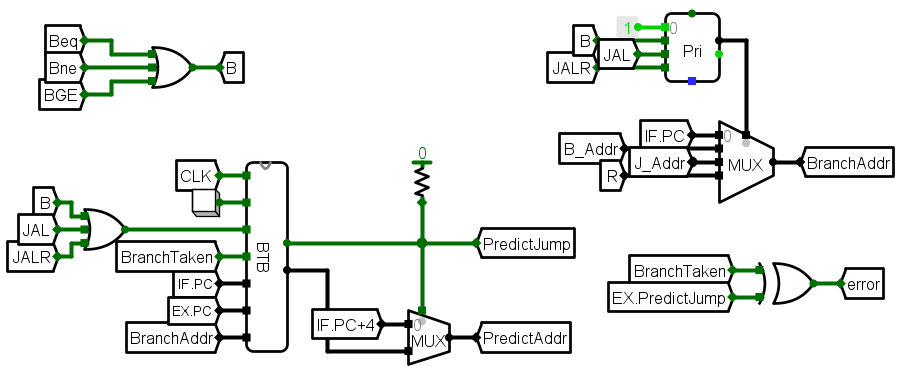


图 . 动态分支预测

如果判断正确，跳转到预测的指令，否则运行原来的指令。

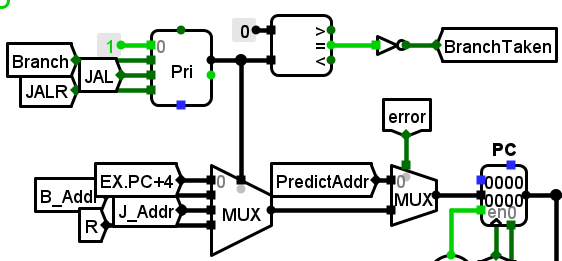


图 . 预测判断与下址

BTB的实现：

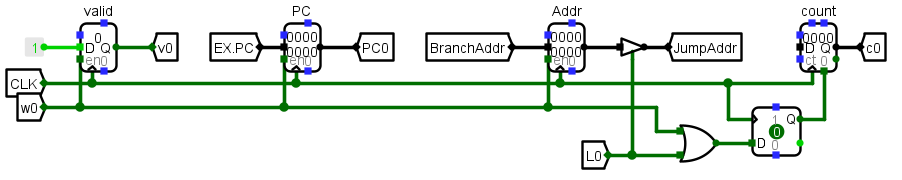


图 . cache结构

IF阶段和EX阶段PC命中逻辑实现如图 3.34所示。

L表示load，地址输出信号。H表示hit，跳转地址命中信号。

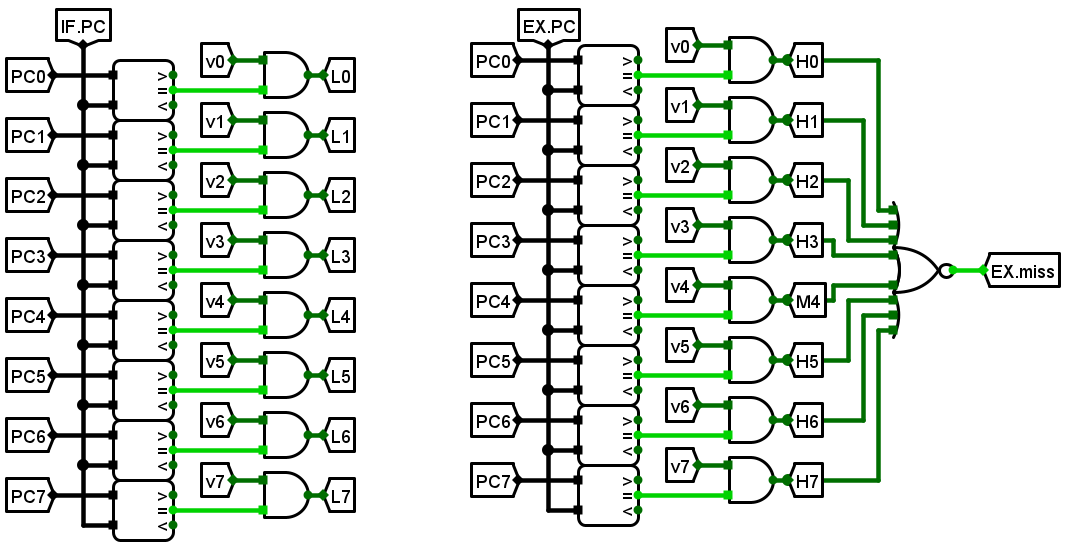


图 . 加载控制和命中判断

Cache写逻辑实现如图 3.35所示。W表示write。当指令为分支指令且未命中时，若8个cache有空余，数据写入空的cache，否则选择最久未访问的cache写入。

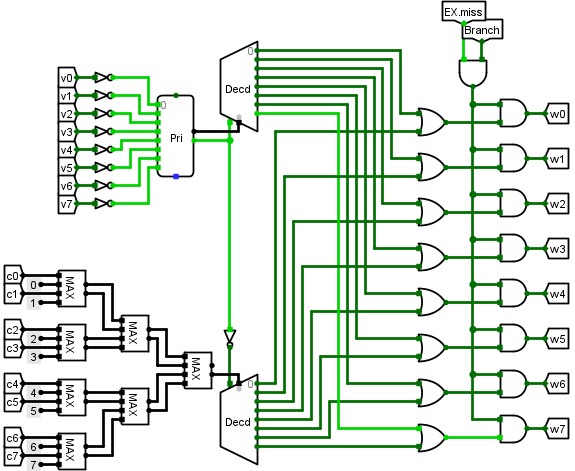


图 . cache写操作逻辑（LRU策略）

填写图 2.7所示的控制信息表，使用Excel的预先定义好的函数得到相应电路图的规范表达式，使用Logisim 的自动生成电路功能完成FSM电路。

最终实现的跳转预测逻辑实现如下。

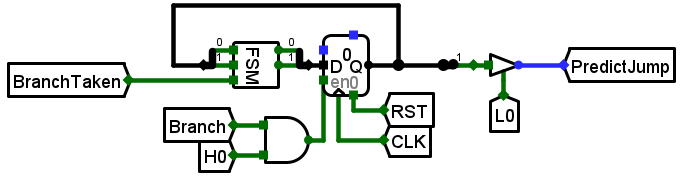


图 3.36 状态机实现

# 实验过程与调试

## 测试用例和功能测试

测试用例使用给定的测试包中的hex文件。

未说明的都为risc-v-benchmark\_ccab.hex文件。

### 单周期ccab测试用例

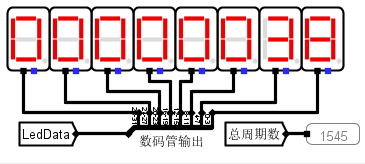


图 . 基本指令测试



图 . 基本指令测试DM结果

按go进行ccab测试

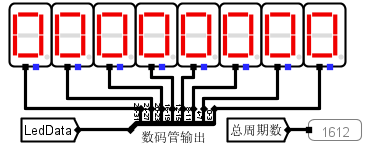


图 . SLL测试结果

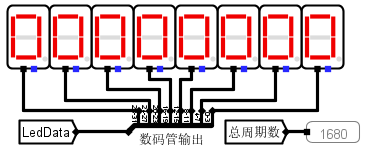


图 . SLR测试结果

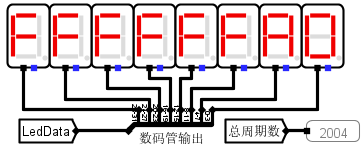


图 . LB测试结果



图 . LB指令DM结果

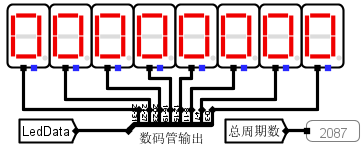


图 . BGE指令测试结果

### 理想流水线测试用例

测试用例为risc-v-理想流水线测试.hex

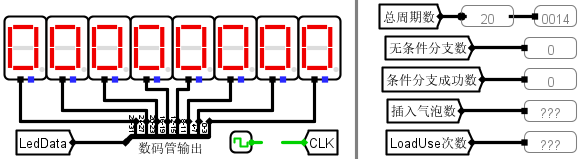


图 . 理想流水线测试

### 气泡流水线测试用例

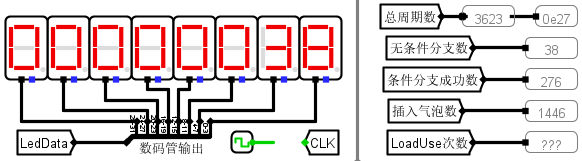


图 . 气泡流水线测试

### 重定向流水线测试用例

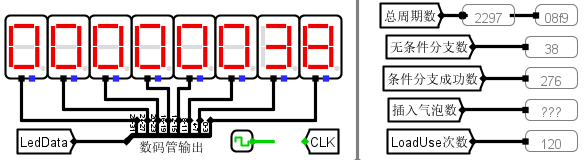


图 . 重定向流水线测试

### 重定向分支预测测试用例

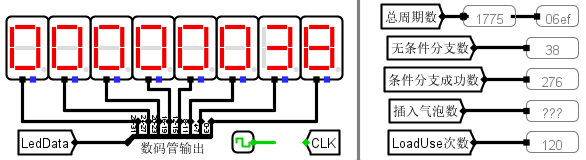


图 . 重定向分支预测设置

### 单级中断测试用例

测试用例为risc-v单级中断测试程序.hex

中断测试信号用提供的模拟器。

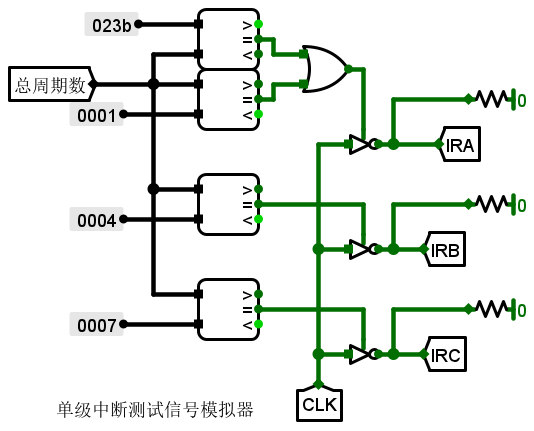


图 . 单级中断测试信号模拟器

中断相应顺序为1->3->2->1，符合预期。

### 多级中断测试用例

测试用例为risc-v多级中断测试(EPC硬件堆栈保护).hex

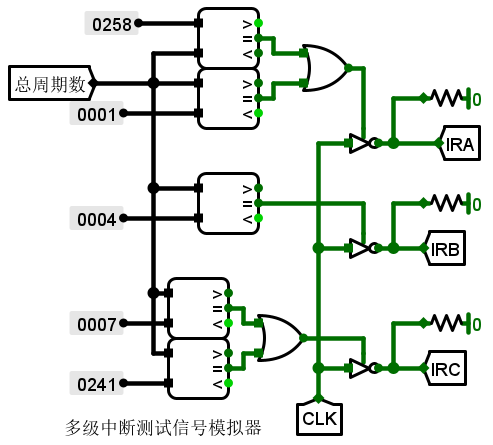


图 . 多级中断测试信号模拟器

中断相应顺序为1->3->2->1->3->1，符合预期。

### 重定向单级中断测试用例

测试用例为risc-v单级中断测试程序.hex，信号模拟器同单级中断。

中断相应顺序为1->3->2->1，符合预期。

## 性能分析

根据周期数，重定向分支预测的性能优于重定向流水，优于气泡流水。

## 主要故障与调试

### cache的LRU策略故障

logisim重定向动态分支预测：cache的LRU策略更新错误，且测试周期数不符合要求。

**故障现象：**当跳转指令未命中时，cache更新的不是最久没访问的寄存器。

**原因分析：**当归并比较出最久未使用的寄存器后，需要等待一个周期再将其清零，否则会直接跳到第二久未访问的寄存器。

**解决方案：**再计数器的清零端前加D触发器，将清零信号延迟一个时钟周期。

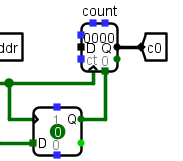


图 . LRU策略实现

### 重定向单级中断返回故障

Logisim重定向单级中断：返回地址错误。

**故障现象：**当退出一个中断程序，响应下一个中断程序时，返回地址变为上一中断程序的URET指令的地址，程序进入死循环。

**原因分析：**响应下一个中断时，EX阶段恰好是返回指令，这条返回指令的地址将会作为现场保存在mEPC寄存器中，下一个中断程序返回后会执行这条指令，结果又返回了这条指令，变成了死循环。

**解决方案：**EX阶段不是返回指令时才能进行中断响应。给mEPC寄存器加上使能端：~EX.URET

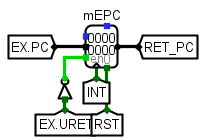


图 . 重定向中断修改

### FPGA板srai指令故障

Verilog实现单周期CPU：算数右移指令故障。

**故障现象：**指令为srai s1,s1,1027。80000000后应该依次输出f0000000，ff000000，fff00000…。但实际输出为10000000，01000000，00100000.

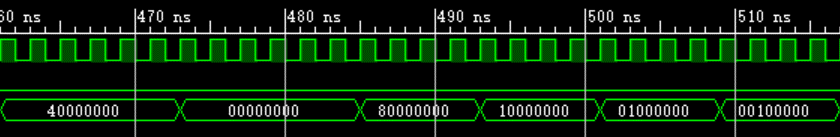


图 .程序故障图

**原因分析：**算术右移指令出错。Vivado不支持算术右移符号>>>，必须手动实现

**解决方案：**修改ALU模块的算数右移代码，实现符号位的右移扩展。

result<= a\_i[31]?~(32'hffffffff>>b\_i[4:0])|(a\_i>>b\_i[4:0]) : a\_i>>b\_i[4:0];

修改后运行成功。

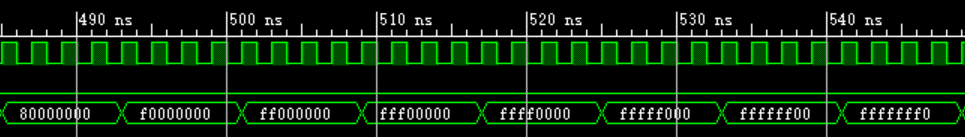


图 .程序运行图

## 实验进度

表 . 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，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。学习理想流水线相关理论知识，完成流水线接口设计，实现了理想流水线CPU并且通过了测试。 |
| 第四天 | 完成Logism理想流水线CPU的故障报告。学习气泡流水线相关理论知识，完成气泡流水线中数据冲突和分支处理逻辑设计，正在调试。 |
| 第五天 | 实现了气泡流水线CPU并且通过了测试，完成气泡流水线故障报告。学习重定向流水线相关理论知识。 |
| 第六天 | 完成重定向逻辑设计。实现了重定向流水线CPU并且通过了测试。 |
| 第七天 | 完成重定向流水线故障报告。学习分支预测相关理论知识。完成分支预测逻辑中的状态机逻辑设计。 |
| 第八天 | 完成分支预测逻辑中cache和LRU策略的逻辑设计。实现了分支预测，但周期数不符合要求，还在调试。 |
| 第九天 | 完成分支预测CPU设计，周期数符合要求。完成故障报告。学习单级中断相关理论知识。 |
| 第十天 | 完成单级中断逻辑设计，测试结果符合要求。完成重定向单级中断设计。 |
| 第十一天 | 设计多级中断逻辑。还在调试。 |
| 第十二天 | 继续调试多级中断，并完成多级中断逻辑设计，测试结果符合要求。 |
| 第十三天 | 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。 |
| 第十四天 | 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。 |

# 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。设计并完成了单周期CPU、理想流水线CPU、气泡流水线CPU、定向流水线CPU、支持中断以及分支预测的重定向流水线、以及流水中断CPU 且扩展设计了ccmb 四条指令。设计了什么，实现了什么，完成了什么…
2. 功能总结。实现了单周期和流水线CPU ，支持包括CCMB 指令在内的31 条指令，理想流水线将单周期CPU 的运行过程流水化，气泡流水线CPU 使理想流水线能够处理数据以及结构冲突，重定向流水线通过重定向的方式减少了气泡数从而提高流水线的运行效率，单周期实现了单级中断和三级可嵌套外部中断，流水中断的支持单级中断，而分支预测通过BHT 表以及相应的预测机制减少了程序运行的周期数。
3. 设计并完成了FPGA板的单周期CPU的实现。

## 课设心得

本次课设历时两周，实现了从基本的单周期 CPU到五段重定向流水动态分支预测机制、单级多级中断及其与流水线CPU的结合，以及FPGA板单周期CPU的任务。这两周过得很充实也很有成就感。现在再来回顾整个课程设计的整个过程，有不少的细节值得我去深思与体会。

在单周期CPU和三种流水线CPU的设计和完成过程中，有教材和课件的指导，网络上也能搜到很多相关的资料，所以整个过程还算比较顺利和迅速。然而在设计分支预测逻辑后，开始遇到各种各样的困难。虽然有详细的课件作为指导，但将理论描述在logisim中实现仍然有很大难度，首先是状态机的设计，然后是cache的LRU策略的实现，最后是分支预测逻辑与重定向流水线的接口设计，每个阶段都遇到了很多细节上的问题，尤其是对各个寄存器的更新数据的监视。接下来花了很多时间的是多级中断，在实现中断嵌套中开关中断的逻辑时调试了很久，寄存器的更新数据的监视仍然是一大问题，有时候脑子里一团混乱想放弃了，最后还是坚持了下来。最后完成的任务是Verilog的单周期CPU实现。由于上学期选了verilog的课，在课堂上实现过简单的CPU，所以在时钟分频、各个部件的实现、数据通路的实现上有一定的理论和实践基础，但仿真调试仍然调了很久。首先因为verilog没有默认值，如果不在initial中赋值，输出的就是未知状态，其次verilog不支持算数右移，ALU和所有位扩展器都必须另外改。FPGA板功能实现的时候是整个课设过程中最有成就感的时刻。

这次课设提醒了我自学能力的重要性、规范化命名在工程实现中的必要性，和调试在项目设计与实现过程中的作用，也让我复习了上学期组成原理和verilog的部分知识，并学习了流水线和分支预测的理论与实践知识，同时提升了我查找资料，寻找问题、解决问题的能力。

然而对于本次课程设计，我还有一些小小的建议和改进。关于提供的logisim转verilog工具，我在上面花了挺多时间，转出来比较好用的只有rom、ram和优先编码器，转换的其他部件有很多没用的接口，应用到代码中会导致函数调用语句和函数内容太长，编写代码和调试过程中难以找到重要的变量，不如自己写一个。而且数据通路是无法转换的。我在网上有查到关于如何用vivado生成rom和ram 的教程，可以提供一个相关的教程，不提供转换工具，既可以帮助同学们熟悉和学习vivado的功能，也避免在转换工具上浪费时间。

最后在这里也感谢老师们在群里对各种问题的耐心解答，也感谢在课程设计中提供了帮助和建议的同学。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，秦磊华，胡迪青.计算机组成原理实践教程.北京：清华大学出版社，2018.
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:** [作者] |
|  |