***2021***



**计算机组成原理 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： | CS1806 |
| 学 号： | U201814788 |
| 姓 名： | 刘美 |
| 电 话： | 15673827636 |
| 邮 件： | [1527796339@qq.com](mailto:13456@qq.com) |

目 录

[1 课程设计概述 3](#_Toc22433)

[1.1 课设目的 3](#_Toc6707)

[1.2 设计任务 3](#_Toc22300)

[1.3 设计要求 3](#_Toc5366)

[1.4 技术指标 4](#_Toc13909)

[2 总体方案设计 6](#_Toc22370)

[2.1 单周期CPU设计 6](#_Toc25325)

[2.2 中断机制设计 11](#_Toc14758)

[2.3 流水CPU设计 13](#_Toc14625)

[2.4 气泡式流水线设计 14](#_Toc8361)

[2.5 重定向流水线设计 15](#_Toc9557)

[2.6 动态分支预测机制 16](#_Toc24056)

[3 详细设计与实现 18](#_Toc14030)

[3.1 单周期CPU 实现 18](#_Toc24577)

[3.2 中断机制实现 21](#_Toc21879)

[3.3 流水CPU实现 25](#_Toc15701)

[3.4 气泡式流水线实现 26](#_Toc6827)

[3.5 重定向流水线实现 28](#_Toc15045)

[3.6 动态分支预测机制实现 30](#_Toc3409)

[4 实验过程与调试 35](#_Toc11156)

[4.1 性能分析 35](#_Toc17355)

[4.2 主要故障与调试 35](#_Toc17430)

[4.3 实验进度 36](#_Toc17041)

[5 团队任务 38](#_Toc31352)

[5.1 团队任务设计 38](#_Toc25709)

[5.2 团队分工 38](#_Toc11902)

[6 设计总结与心得 39](#_Toc16810)

[6.1 课设总结 39](#_Toc22650)

[6.2 课设心得 39](#_Toc30140)

[参考文献 40](#_Toc16595)

# 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. 支持表 1.1前27条基本32位MIPS指令；
2. 支持教师指定的4条扩展指令；
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。

表 1.1 指令集

| **#** | **指令助记符** | **简单功能描述** | **备注** |
| --- | --- | --- | --- |
| 1 | ADD | 加法 | 指令格式参考MIPS32指令集，最终功能以MARS模拟器为准。 |
| 2 | ADDI | 立即数加 |
| 3 | ADDIU | 无符号立即数加 |
| 4 | ADDU | 无符号数加 |
| 5 | AND | 与 |
| 6 | ANDI | 立即数与 |
| 7 | SLL | 逻辑左移 |
| 8 | SRA | 算数右移 |
| 9 | SRL | 逻辑右移 |
| 10 | Sub | 减 |
| 11 | OR | 或 |
| 12 | ORI | 立即数或 |
| 13 | NOR | 或非 |
| 14 | LW | 加载字 |
| 15 | SW | 存字 |
| 16 | BEQ | 相等跳转 |
| 17 | BNE | 不相等跳转 |
| 18 | SLT | 小于置数 |
| 19 | STI | 小于立即数置数 |
| 20 | SLTU | 小于无符号数置数 |
| 21 | J | 无条件转移 |
| 22 | JAL | 转移并链接 |
| 23 | JR | 转移到指定寄存器 | If $v0==10 halt(停机指令)  else 数码管显示$a0值 |
| 24 | SYSCALL | 系统调用 |
| 25 | MFC0 | 访问CP0 | 中断相关，可简化，选做 |
| 26 | MTC0 | 访问CP0 | 中断相关，可简化，选做 |
| 27 | ERET | 中断返回 | 异常返回，选做 |
| 28 | SRLV | 逻辑可变左移 |  |
| 29 | SUBU | 无符号减法 |  |
| 30 | LHU | 加载半字（无符号） |  |
| 31 | BGEZ | 大于等于0跳转 |  |

# 总体方案设计

## 单周期CPU设计

单周期CPU（24条指令）本次我们采用的方案是微程序控制，且主、控存分开的方案，即采用微程序控制方式，实现主存储器（MM）和微程序控制存储器（CM）不共用一个存储器的方式完成方案的设计。同时在实施的过程中，全部电路由Logisim仿真实现。

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

运算器部分主要包括程序计数器PC,指令存储器IM，运算器和寄存器堆RF等。具体设计思路如下。

#### 程序计数器PC

程序计数器PC使用一个寄存器缓存下一条执行的指令地址，并由于存在条件分支指令和无条件转移指令，下一条指令执行的地址不一定是顺序地址PC+4，所以需要根据具体的指令译码信号来选择，所以还需要设计一个选择器。选择器的选择端是由多个跳转分支指令信号组成的三位宽数字Sel={beq/bne,JMP,JR}。选择端Sel=100代表此时指令为beq或bne指令且分支跳转成功，跳转的地址为对应指令字指定的PC+4+Imm；当Sel=011时，代表此时指令为JR无条件跳转指令，跳转的地址为寄存器R[rs]的值；当Sel=010时，代表此时指令为J或者JAL无条件跳转指令，两者的跳转地址为{PC+4(31:28),IR(25:0),2b’0}。选择器的输出端输出下一条指令的地址并送入到寄存器中，在时钟的触发下，送入指令存储器根据PC值获取指令。

#### 指令存储器IM

指令存储器中存放的是二进制的MIPS指令。在Logisim中，该存储器通过ROM实现，由于一条指令只有4个字节，且按字节读取，所以只需要取PC的2到11位作为输入送入ROM存储器中。

#### 运算器

运算器能够根据指令的译码信号进行一些基本的算术逻辑运算。本实验中运算器使用的是Logisim自带的ALU器件，其引脚和实现的功能如表2.1所示。运算器的功能码如表2.2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 |

表 2.2 运算器功能码对应表

| ALU\_OP | 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

寄存器堆使用Logisim自带的电路封装。该寄存器堆包含MIPS32个通用寄存器并实现读写寄存器的操作。寄存器的各个端口对应的功能如表2.3所示。

表 2.3 寄存器堆RF引脚功能

| 引脚 | 功能 |
| --- | --- |
| R1# | 读寄存器1的编号 |
| R2# | 读寄存器2的编号 |
| W# | 写寄存器的编号 |
| Din | 写入寄存器的值 |
| WE | 写使能信号，为1时代表写入寄存器 |
| CLK | 时钟端 |
| R1 | R1号寄存器的值 |
| R2 | R2号寄存器的值 |

### 数据通路的设计

在完成各个运算部件的设计后，我们根据每条指令的译码信号，获得每个运算部件引脚需要的信息，得到如2.4的表格。

表 2.4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 | rs | rt | rd | alu | r1 | r2 | 5 |  |  |
| ADDI | PC+4 | PC | rs |  | rt | alu | r1 | SignImm | 5 |  |  |
| ADDIU | PC+4 | PC | rs |  | rt | alu | r1 | UnsignImm | 5 |  |  |
| ADDU | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 5 |  |  |
| AND | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 7 |  |  |
| ANDI | PC+4 | PC | rs |  | rt | alu | r1 | SignImm | 7 |  |  |
| SLL | PC+4 | PC |  | rt | rd | alu | r2 | SignImm | 0 |  |  |
| SRA | PC+4 | PC |  | rt | rd | alu | r2 | SignImm | 1 |  |  |
| SRL | PC+4 | PC |  | rt | rd | alu | r2 | SignImm | 2 |  |  |
| SUB | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 6 |  |  |
| OR | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 8 |  |  |
| ORI | PC+4 | PC | rs |  | rt | alu | r1 | SignImm | 8 |  |  |
| NOR | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 10 |  |  |
| LW | PC+4 | PC | rs |  | rd | ReadData | r1 | SignImm | 5 | alu |  |
| sw | PC+4 | PC | rs | rt |  |  | r1 | SignImm | 5 | alu | r2 |
| BNE/BEQ | PC+4+SignImm{32:0} | PC | rs | rt |  |  | r1 | r2 |  |  |  |
| J | {PC+4{31:28},  IR{25:0},2b’0} | PC | rs | rt |  |  | r1 | r2 |  |  |  |
| JAL | {PC+4{31:28},  IR{25:0},2b’0} | PC | rs |  | R[31] | ReadData | r1 | r2 |  |  |  |
| JR | R[rs] | PC | rs |  | rt | ReadData | r1 | SignImm |  | alu | r2 |
| Syscall | PC+4 | 2# | 4# |  |  |  | r1 | r2 |  |  |  |

### 控制器的设计

在实验的数据通路中，我们涉及到很多的控制信号以及选择信号，如寄存器和存储器的写控制端，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，为了使电路看起来更加简洁美观，我们使用一个子电路来封装这些控制信号。首先我们对控制信号进行统计，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 2.5。

表 2.5 主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| MemToReg | 0/1 | 写内存控制信号，为1时代表向内存RAM中写入值 |
| RegWrite | 0/1 | 寄存器写使能信号，为1时代表向寄存器中写入值 |
| AluOP | 0000~1100 | 见表2-2 |
| MemToReg | 0 | 寄存器写入数据来自运算器ALU的结果 |
| 1 | 寄存器写入数据来自存储器 |
| RegDst | 0 | 写入寄存器编号为rt |
| 1 | 写入寄存器编号为rd |
| AluSrcB | 0 | 运算器B来自寄存器输出R2 |
| 1 | 运算器B来自符号拓展 |
| SignedExt | 0/1 | 为0代表对立即数无符号拓展，为1代表对立即数有符号拓展 |
| JR | 0/1 | 寄存器跳转指令译码信号，为1代表当前为JR指令 |
| JAL | 0/1 | 为1时代表当前指令为JAL指令 |
| JMP | 0/1 | 无条件分支控制信号 |
| Beq | 0/1 | Beq指令译码信号 |
| Bne | 0/1 | Bne指令译码信号 |
| Syscall | 0/1 | Syscall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。

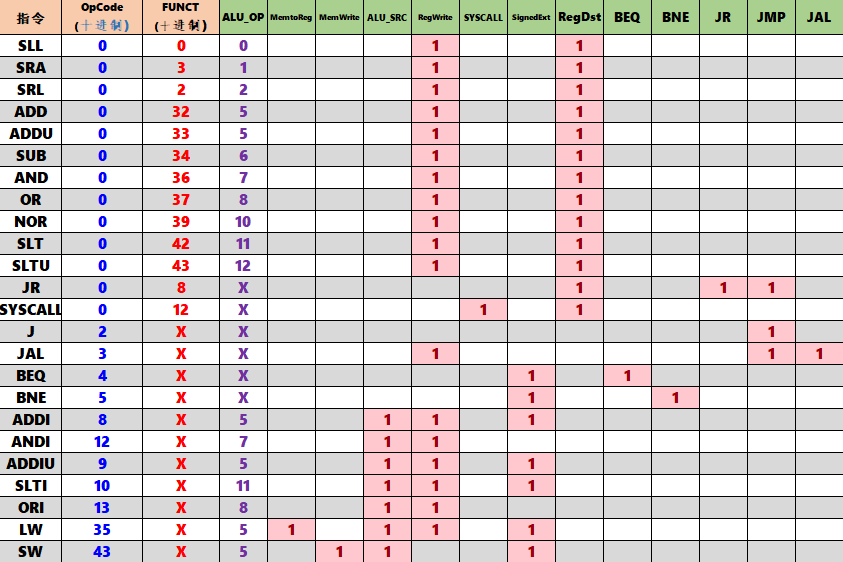


图 2.2 指令控制信号真值表

## 中断机制设计

### 总体设计

在本实验中的中断是异步中断，其中断请求与指令无关，CPU需要在指令执行结束后才响应外部中断。为了实现MIPS中断处理，需要从硬件和软件两个方面进行设计。在硬件方面，我们需要实现中断请求生成逻辑电路、中断仲裁与中断识别电路、异常程序计数器EPC以及中断使能寄存器IE；在软件方面我们要实现相关的指令支持，包括中断返回指令、开关中断指令等，还要实现程序支持，改造主程序和中断服务程序。单中断和多中断的执行过程如图2.3所示。

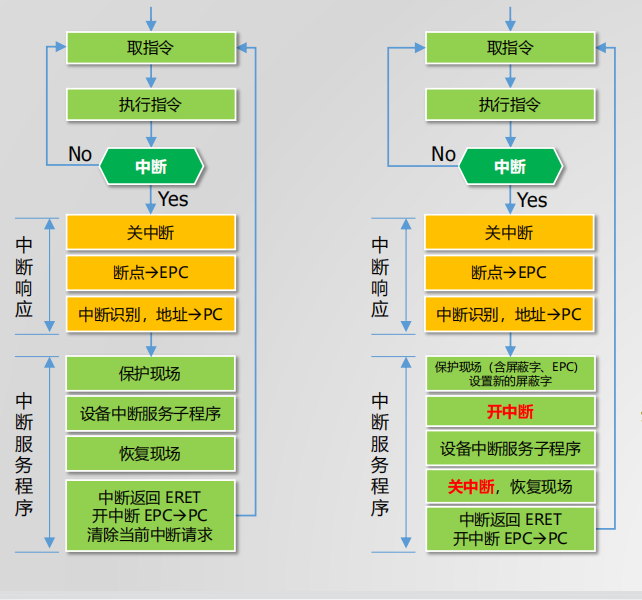


图2.3 中断执行流程图（左单中断，右多中断）

### 硬件设计

由于是外部中断，我们首先要实现按键中断外围电路，能够通过点击编号按钮产生对应的中断请求，锁存中断请求并在完成对应的中断请求服务后清除中断请求。

我们使用一个优先编码器实现中断优先级仲裁，在当有多个中断请求信号时，输出优先级最高的中断请求信号。然后获取三个中断服务程序的入口地址通过一个选择器输出相应的中断入口地址实现中断入口识别。

异常程序计数器EPC实现保护被中断程序的断点和现场，即当有中断请求产生时将当前的PC地址压入寄存器中。在单级中断中，当前执行的中断不可被其他中断请求打断，所以只需要保存一个断点，而在本实验的多级中断中，由于要求实现中断程序的嵌套执行，最坏情况下需要保存三个程序断点，所以我们需要利用三个寄存器构成一个EPC硬件堆栈来实现保存多个程序地址断点以及对应的中断号。

单级中断和多级中断分别利用中断隐指令、ERET信号和MFC0、MTC0实现中断使能寄存器IE来控制开关中断，只有在关闭中断后才可以响应当前的中断请求，在单级中断中，由于不涉及中断程序的嵌套，只有在中断程序执行完毕后才开中断，允许继续相应其他中断信号，而对多级中断而言，在优先级条件满足的情况下可以将当前执行的低优先级的中断打断而转去执行更高优先级的中断，所以要在低级的中断保护完现场后开中断。

### 软件设计

多级中断和单级中断程序中都存在与中断服务程序有关的指令来实现开关中断和中断返回。因此我们要对这些指令进行译码产生相应的指令控制信号。由图2.3可知在单级中断中，响应时根据中断隐指令关中断，在执行eret后执行开中断，所以我们只需要设计指令ERET，该指令由控制信号产生器根据指令译码产生。在多中断中，除了中断返回指令ERET以外还涉及到单独的开关中断——MFC0和MTC0，这两个信号也通过指令译码产生，两个控制信号指令译码的低六位均为0x10，区别在于MFC0的第25-21位为0x0，MTC0为0x4。

## 流水CPU设计

### 总体设计

我们将MIPS指令的执行过程细分为5个阶段：

* 取指令(IF)：从指令存储器中取出指令
* 指令译码(ID)：操作控制器对指令字译码，同时从寄存器堆取操作数
* 指令执行(EX)：执行运算操作或者地址计算
* 访存(MEM)：对存储器进行读写操作
* 写回(WB)：将指令执行结果写回寄存器堆

我们在每一个阶段后面都设计一个流水寄存器用于锁存本阶段处理完成的所有数据或结果，以保证本段的执行结果能够在下一个时钟周期给下一个阶段使用。

### 流水接口部件设计

流水接口部件实质上就是一个锁存器，用于锁存前段加工处理完毕的数据和控制信号，这些数据通常会横穿流水寄存器传递到下一段。五段流水线需要设计四个流水寄存器分别命名为：IF/ID, ID/EX, EX/MEM, MEM/WB。所有的流水接口部件都使用统一的始终CLK来同步，并通过设计同步清零实现清除流水寄存器中的数据，设计使能端实现流水寄存器的暂停，设计若干寄存器来锁存相应的数据。

不同流水寄存器锁存和传递的数据信息不尽相同。以气泡流水线为例，其具体用到的数据入表2.6所示。

表2.6 各流水部件需要传递的数据

| 流水部件 | 锁存和传递的数据 |
| --- | --- |
| IF/ID | PC, PC+4, IR |
| ID/EX | PC, PC+4, IR, RegWrite, MemToReg, MemWrite, AluOp, Alusrc, SignExt, JMP, JAL, JR, Beq, Bne, Syscall, RS,RT,写寄存器编号WriteReg#, 符号拓展, 无符号拓展 |
| EX/MEM | PC, PC+4, IR, RegWrite, MemToReg, MemWrite, JAL, Syscall, 写寄存器编号WriteReg#, 数据存储器待写入数据Writedata, v0, AluResult |
| MEM/WB | PC, PC+4, IR, RegWrite, MemToReg, JAL, Syscall, 写寄存器编号WriteReg#, 数据存储器待写入数据Writedata, v0, AluResult, 数据存储器读出数据RD |

### 理想流水线设计

由于进入理想流水线的对象不受其他阶段的影响，仅仅运行无数据相关，没有分支指令，我们可以将流水寄存器中和控制信号产生部件中有关跳转指令的端口悬空，并在单周期MIPS的基础上删除有关分支跳转和无条件转移的电路，然后将对应的信号和数据连接对应的流水寄存器接口即可。

## 气泡式流水线设计

实际流水线存在很多指令相关和数据依赖，从而引起流水线的停顿。指令相关会导致结构冲突和控制冲突，数据相关则会带来数据冲突。

结构冲突往往是因为多条指令在同一周期使用同一个操作部件，这可以通过增设这个冲突部件来解决。

控制冲突主要体现在分支指令，当流水线遇到分支指令或其他会改变 PC 值的指令时，在分支指令之后载入流水线的相邻指令可能因为分支跳转不能进入执行阶段。为了解决控制冲突，需要在执行程序分支跳转的时候清除已经进入流水寄存器中的分支指令后续的若干条误取指令。当选定在EX阶段执行分支指令，此时有两条指令分别进入了IF/ID, ID/EX流水寄存器，需要将分支成功跳转的信号BranchTaken输送到这两个流水寄存器的清零端进行清空操作。同时在此阶段将计算出来的跳转地址送入到PC的选择器中。

数据冲突主要体现在：后续指令I2 要用到前面指令 I1 的结果，如果指令

I2 在指令 I1 将结果写寄存器之前就在 ID 段读取了该寄存器的旧值，则读取数据就是错误的。为了解决这样的数据冲突，我们在存在冲突的指令之间插入空指令直到冲突消失。为此，我们首先需要设计在理想流水线的基础上增加一个数据冲突检测子电路，该子电路在ID段确认指令使用的源寄存器是否在前两条指令中被写入。由于MIPS指令只包括0~2个源操作数，分别是rs,rt字段对应的寄存器，我们只要检查EX、MEM段的寄存器堆写入信号RegWrite是否为1且写寄存器编号WriteReg#和源寄存器编号是否相同即可。当数据冲突检测子电路检测到冲突产生冲突信号Stall时，将该信号取反送入到IF/ID、PC寄存器的使能端就可以实现IF、ID段指令的停顿。

完整时钟周期从1开始，0结束，中间时刻为下跳沿，因此为了解决ID段和WB阶段的数据冲突，可以让寄存器堆写入控制采用下跳沿触发，而其他流水寄存器使用上跳沿触发，这样在一个时钟节拍的中间时刻（下跳沿）可以完成寄存器数据的写入而后半段可以正确读取寄存器堆中的数据值。气泡流水线的数据通路如图2.4所示。

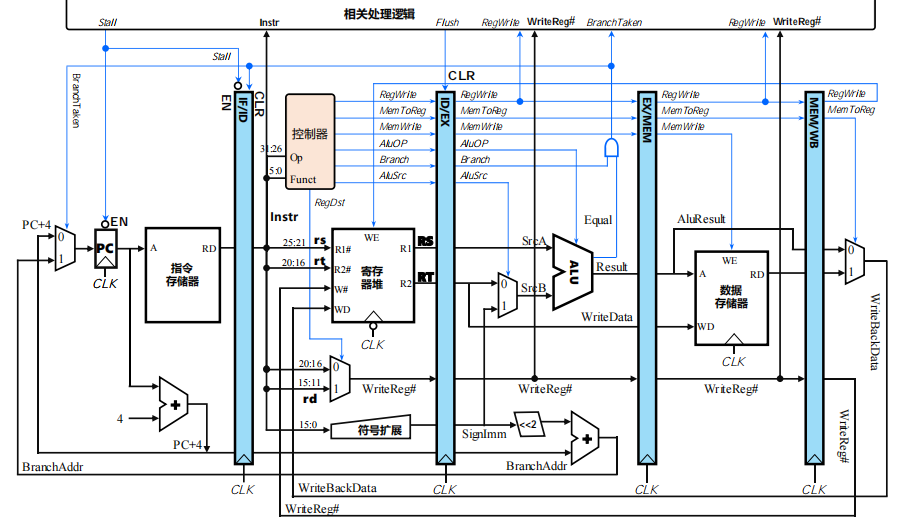


图2.4 气泡流水线数据通路

## 重定向流水线设计

实际上的程序存在大量的数据冲突和控制冲突，采用气泡流水线将带来大量气泡和停顿严重影响流水线的性能。我们知道当ID段检测到数据冲突时，此时EX阶段取到的操作数一定是错误的，但此时的正确的操作数其实已经在MEM或WB阶段产生。所以我们可以直接将正确的操作数从其所在的位置重定向到EX段的合适位置而不需要进行数据检测从而避免产生气泡导致流水线的停顿。

EX阶段源操作数有两个RS,RT。为了选择正确的操作数，我们需要设计相关处理电路生成两个重定向选择信号RsForward和RtForward。如果产生冲突的两条指令的前一条为Load指令且位于EX段时，此时无法通过重定向解决，因为理论上虽然可以但此时关键延迟将变长而使流水线的性能变差。因此我们还是要检测由Load带来的数据相关。重定向依然还需要解决由分支指令带来的控制冲突，基本原理和2.4节中气泡流水线解决方案一致。

最终重定向电路将在遇到Load指令冲突时使PC寄存器产生停顿，在遇到分支跳转时对IF/ID和ID/EX流水寄存器进行清空。

## 动态分支预测机制

重定向流水虽然很大程度上解决了数据冲突带来的停顿，但还是存在由分支指令带来的控制冲突，因此我们可以采用动态分支预测的方法，设计一个分支预测缓冲器BTB表，根据分支指令的分支跳转历史不断对预测策略进行动态调整，当预测成功时不需要停顿，以此提升性能。

为了实现对分支的预测，我们设计一个双预测位的状态转换机。该状态机主要操作为两步：① 分支预测；② 状态修改，能够根据分支跳转实际执行情况和当前状态修改下一个状态，其状态转换图如图2.5所示。从图中可以看出，只有连续两次预测错误才会改变对分支去向的预测。

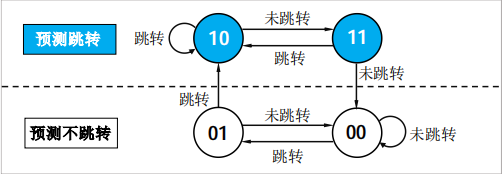


图2.5 双预测位状态转换图

由状态图不难看出，当双预测位的高位为1时，BTB表预测“分支成功”，为0时预测“分支失败”。

BTB表本质上是一个全相联的cache结构。其每个cache槽主要包括 valid 位、执行过的成功分支指令地址（标识），预测的分支目标地址，历史跳转信息描述位（预测状态位）、淘汰计数五项。采用BTB的处理过程如图2.6所示。其数据通路如图2.7所示。

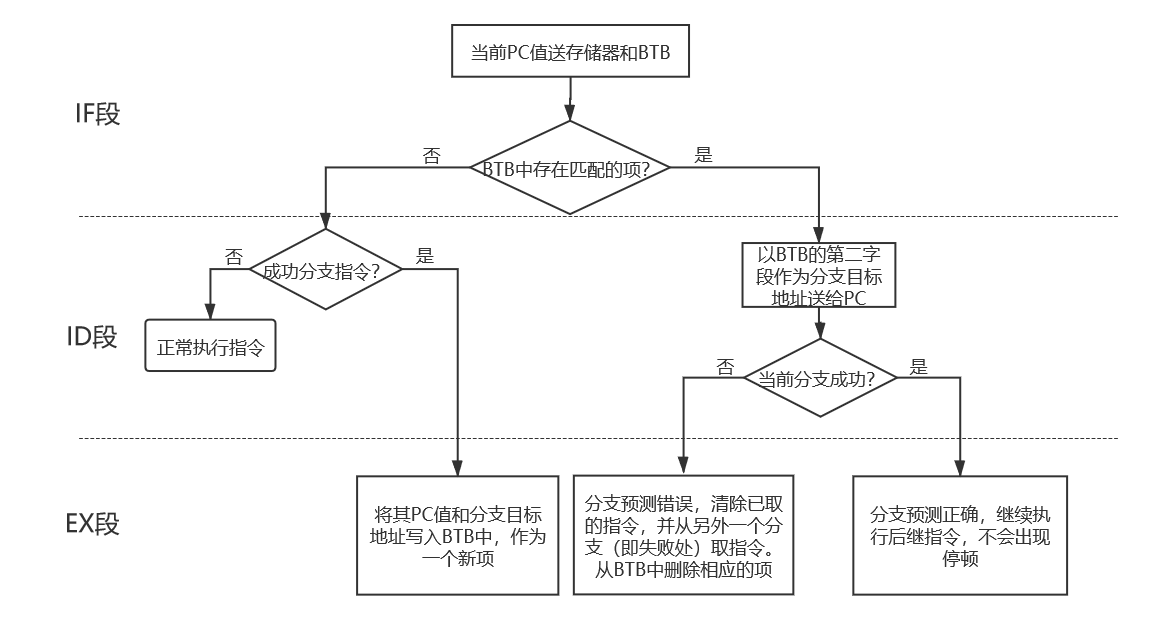


图2.6 采用BTB时的处理步骤

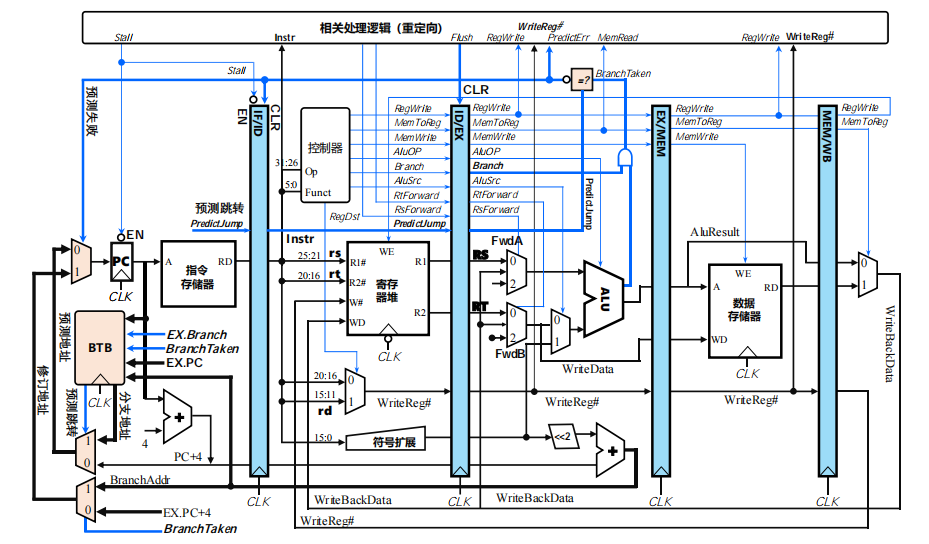


图2.7 使用动态分支预测的五段流水线数据通路

# 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）

使用一个32位寄存器实现程序计数器PC，触发方式为上升 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。v0!=34为停机信号，将此控制信号通过非门取反之后和Go相或，当需要进行停机时，v0!=34控制信号为1，经过非门之后为0，Go为0，相或后为0，屏蔽时钟信号，使整个电路停机。若此时按下Go按钮，或门输出为高脉冲，能够触发时钟使电路继续执行。因为程序中涉及到跳转指令，下一条执行的指令与当前的指令有关，所以输入还需要通过一个选择器选择正确的PC地址。如图 3.1所示。

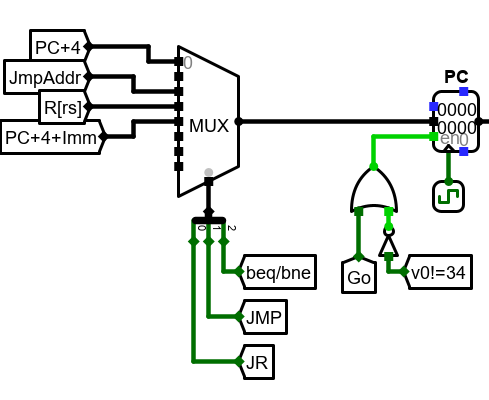


图 3.1程序计数器（PC）

1. 指令存储器（IM）

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。

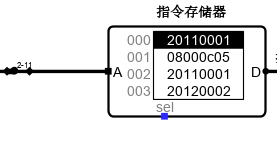


图 3.2指令存储器（IM）

1. 运算器（ALU）

运算器采用Logisim所带的库cs3410.jar中的运算器电路。运算器的X输入端来自寄存器堆的R1输出，Y输入端取决于AluSrcB和SignedExt信号，当二位宽数据{AluSrcB,SignedExt}=2b’0|2b’1时，选择寄存器堆的R2输出端输入，为2b’2时选择有符号拓展立即数，为2b’3时选择无符号拓展的立即数。如图3.3所示。

1. 寄存器堆（RF）

寄存器堆器件也采用cs3410.jar中自带的子电路。输入端口R1#和R2#取决于syscall指令，如果此时为syscall指令则R1#选择2号寄存器，否则选择rs字段对应的寄存器；R2#选择4号寄存器，否则选择rd字段对应的寄存器。输入端口W#取决于RegDst和JAL两个控制信号，JAL为1选择0x1f号寄存器，为0时如果RegDst为1则选择rd字段对应的寄存器，否则选择rt字段对应的寄存器。输入端口Din取决于JA控制信号，当JAL为1时将PC+4的值写入，否则选择数据存储器中的内容。如图3.3所示。

1. 数据存储器(DM)

使用一个随机存储器RAM实现数据存储器（DM)。和指令存储器一样，输入的地址运算结果只取2-11位输入。RAM的输入端来自于寄存器堆的R2输出，并在MemWrite信号的控制下进行向输入的地址所在的地方写入数据，并在MemToReg信号为1时将存储器中的数据写回到寄存器堆中。如图3.3所示。对于差异化指令LHU（读取半字），读的是高半字还是低半字取决于存储器输入的32位地址的第1位，第1位为1代表为高半字，此时选取存储器的16~31位按照0拓展为32位输出，否则选择0~15位0拓展输出。其修改如图3.4所示。

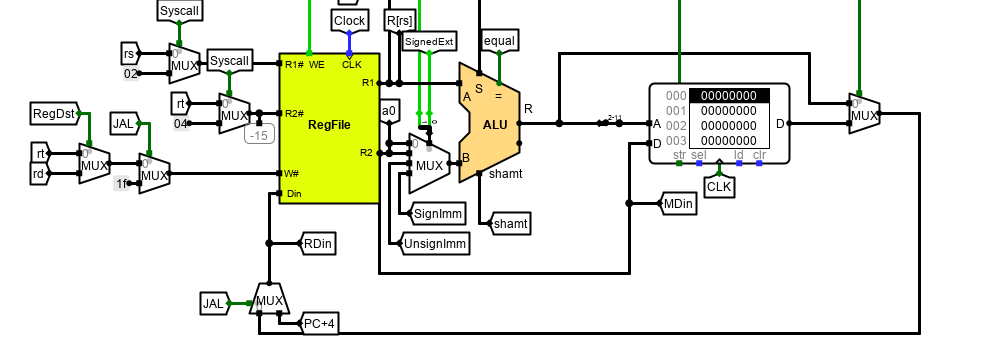


图 3.3 RF、ALU、DM电路结构图

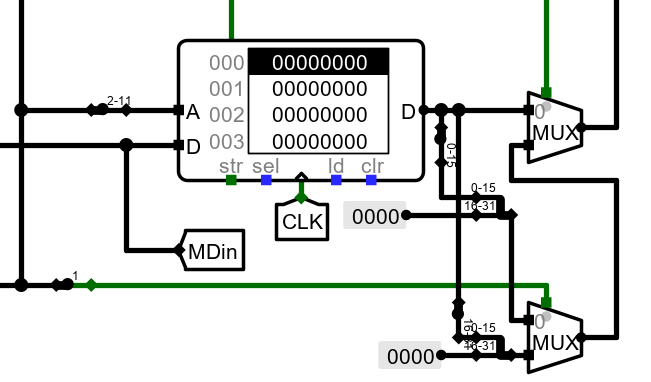


图 3.4 基于LHU修改DM设计

### 数据通路的实现

在完成主要功能部件后，根据表2.4的数据通路表进行多指令数据通路的合并，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如图3.5所示。

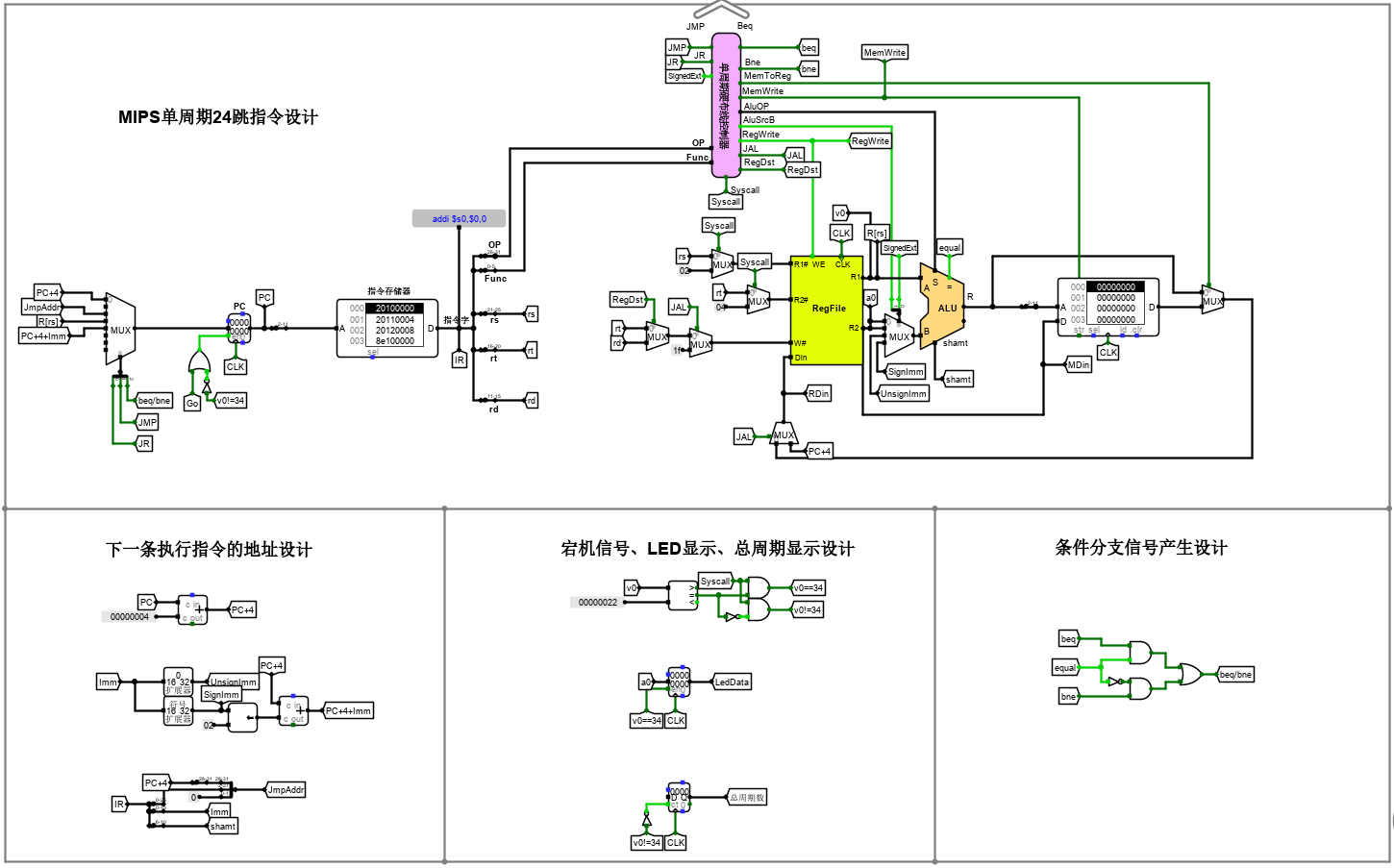


图 3.5 单周期CPU数据通路（Logisim）

### 控制器的实现

根据图2.2所示的控制信号表，利用Logisim自动生成组合逻辑的功能实现控制信号部件和运算器控制器部件。电路如图3.6所示。

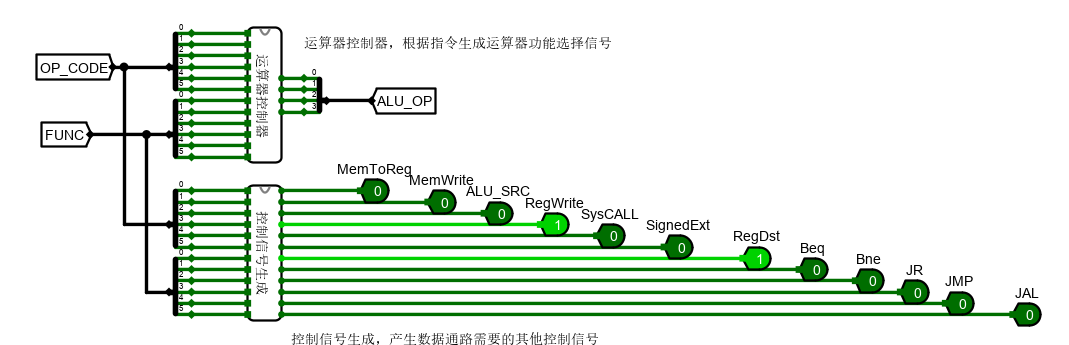


图 3.6 单周期控制器部件

## 中断机制实现

### 单级中断

1. 按键中断外围电路

中断信号采样电路如图3.7所示，左侧的D触发器在当按下中断按键时被触发，该信号通过右侧的D触发器锁存，同时Clr信号控制该触发器实现同步清零。

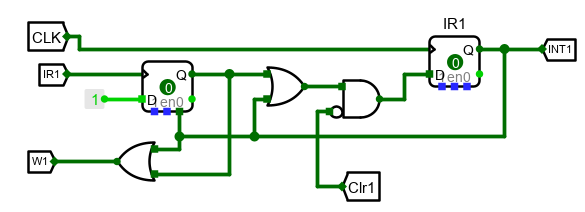


图 3.7 中断信号采样电路

1. 中断优先仲裁电路和中断入口识别电路

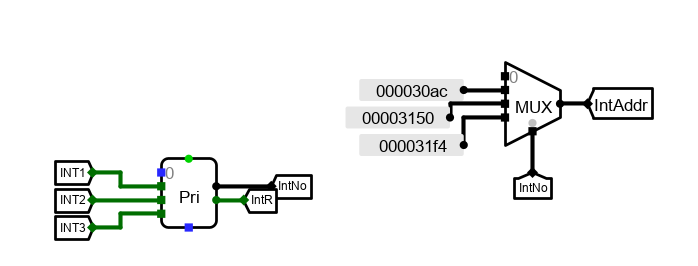


图 3.8中断优先仲裁和入口识别电路

1. 异常程序EPC寄存器和IE寄存器

在单级中断中存在ERET指令，该指令信号由指令译码给出，中断程序将在该指令执行时执行开中断操作，因此在IE寄存器中，中断请求信号IntR和ERET取非后相与一起送到D触发器的输入端，这样在当有按键中断请求产生且ERET为0时，触发器输出为1，表示关中断，且一旦进入一个中断程序之后，即使有新的优先级更高的中断请求产生，因为当前中断程序尚未执行完毕，ERET信号为0，D触发器的输出始终为1，始终为关中断从而不会进入其他中断程序。只有在中断服务程序执行完毕后，中断返回ERET为1，D触发器输出为0，才开中断允许响应其他中断请求。IE寄存器的输出取反后与IntR信号相与可以得到真正的被允许的中断请求信号。

EPC寄存器的输入端为当前正在执行指令的下一条指令的PC地址，在当中断请求信号为1时触发时钟，为0时屏蔽，使得程序PC地址正确保存到寄存器中。

在当ERET信号为1时，执行中断返回，PC寄存器将获取EPC中保存的程序断点作为下一条执行的指令；当中断请求产生时, PC寄存器将获取当前的中断入口程序地址作为下一条执行的指令地址，否则按照正常程序执行取址。

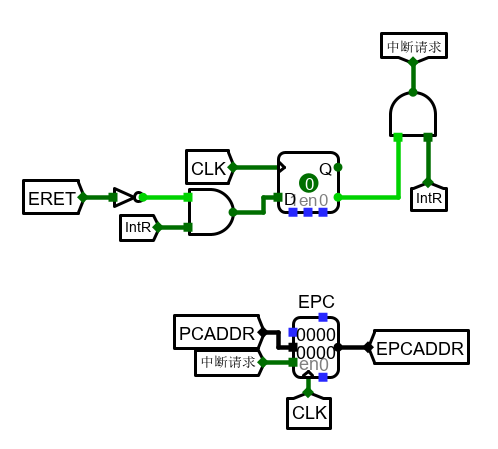
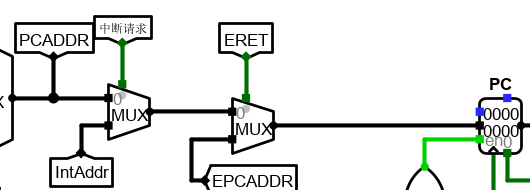
 

图 3.9 EPC寄存器和IE寄存器 图 3.10 添加中断的数据通路

### 多级中断

1. 异常程序计数器EPC堆栈

依据栈先进后出的思想，连续使用三个寄存器实现一个堆栈。当有多个中断请求信号先后到来时，最先执行的低优先级中断程序的地址将被触发存入后面的寄存器中，而在当一个中断服务程序执行完后，ERET为1，寄存器被触发选择后一个寄存器的内容作为输入从而实现恢复断点。中断服务号同理。

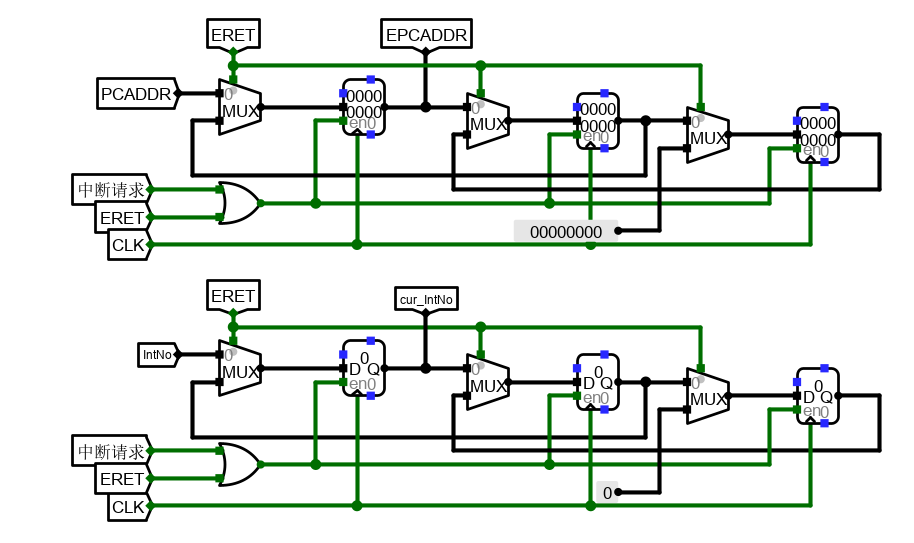


图 3.11 多级中断寄存器堆栈（上为EPC寄存器，下为中断服务号寄存器）

1. 开关中断信号产生逻辑

rs为指令译码信号中寄存器编号对应的字段，OP为指令译码信号中的操作码字段。OP字段与0x10通过一个比较器如果输出等于为1，表明当前为MFC0或MTC0指令，该输出信号分别通过和rs与0x00，rs与0x04的比较器输出结果相与，在当rs等于0x00时与门输出1表示为MFC0指令，当rs等于0x04时表示为MTC0指令。

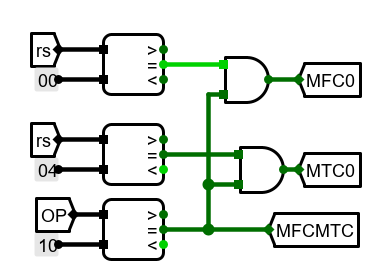


图 3.12 MFC0和MTC0信号产生逻辑

1. IE使能寄存器和中断请求产生逻辑

IE使能通过1位宽的寄存器实现。关中断信号MTC0和中断请求信号相或作为寄存器的输入，中断请求信号、关中断、开中断信号MFC0和ERET信号相或作为寄存器的使能端，其中有一个为1时就触发时钟更改IE的值。这是因为在当有允许的中断请求产生时，程序执行MTC0指令，MTC0信号为1，寄存器被触发，从而IE=1关中断，而当程序执行完毕后，执行MFC0开中断指令，MFC0信号为1再次触发寄存器，此时输入端为0，所以IE=0，实现开中断。根据优先级实现中断嵌套体现在当按键产生的中断如果比当前正在执行的中断程序的优先级高,满足中断嵌套的条件，IntR与cur\_IntNo比较结果输出为1，再与IE寄存器的输出取反相与后，会在IE=1关中断时产生被允许的中断请求信号。电路如图3.13所示。

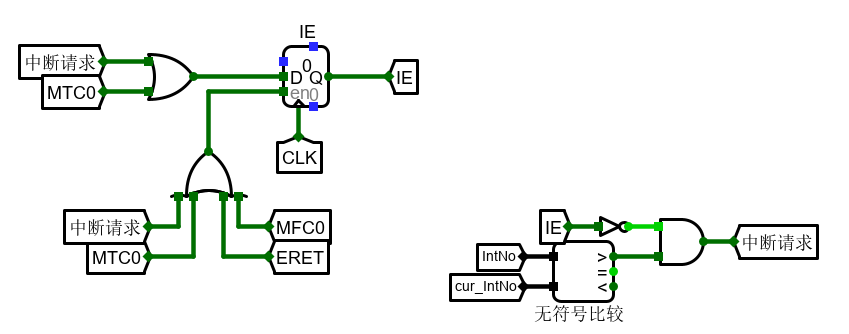


图 3.13 IE中断使能寄存器和中断请求信号产生逻辑

### 流水中断

流水中断是基于气泡流水线和单中断实现的。由于异步中断不一定要立即响应，可以在处理器方便的时候进行处理，设定当流水线检测到异步中断时，在WB阶段进行中断响应。如果响应中断时，位于WB阶段的指令是分支指令，且分支跳转成功，或者如果当前由于数据冲突WB阶段的指令为空指令，则EPC寄存器保存的WB.PC+4就是错误的断点地址；所以我们认为此时暂时不响应该中断，所以在单周期中断逻辑部分，我们修改中断请求信号的产生逻辑和EPC寄存器，仅在当前无数据冲突且WB阶段不是空指令且关中断且有中断请求时产生请求信号，信号产生电路和EPC寄存器的实现分别如图3.14和图3.15所示。

当中断请求产生或者ERET信号为1时需要清除IF/ID、ID/EX，EX/MEM流水寄存器的误取指令。

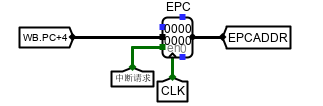
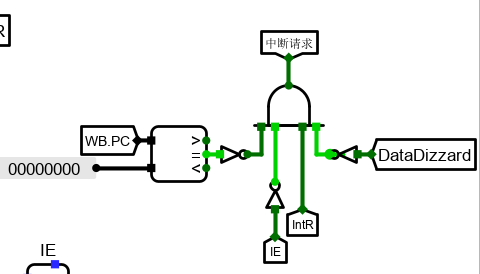


图 3.14 流水中断中断请求信号产生逻辑 图3.15 流水中断EPC寄存器设计

## 流水CPU实现

### 流水接口部件实现

图3.16给出气泡流水线IF/ID部分。其他流水线接口设计除了输出引脚的数量和代表的含义不同以外，电路连接方式与图3.16一致，具体的引脚信息由第二节表2.6给出。

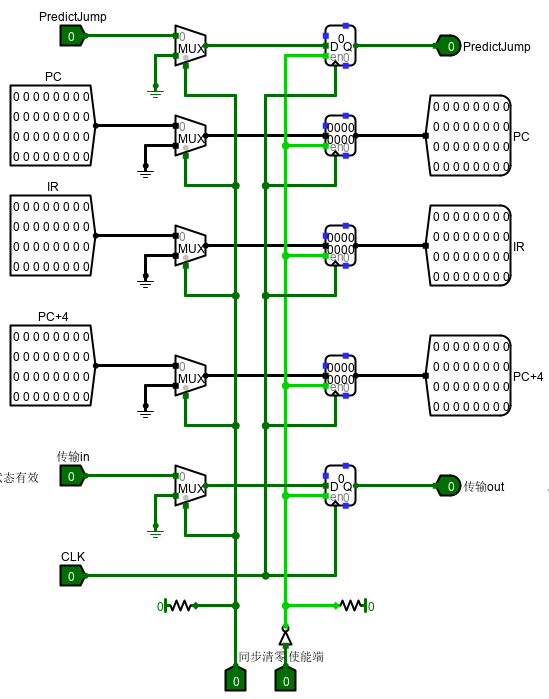


图 3.16 IF/ID流水接口部件电路

### 理想流水线实现

在单周期的基础上，将指令执行的操作分为五个阶段，并简化跳转指令，删除有关跳转逻辑的部分，最后得到的电路如图3.17所示。

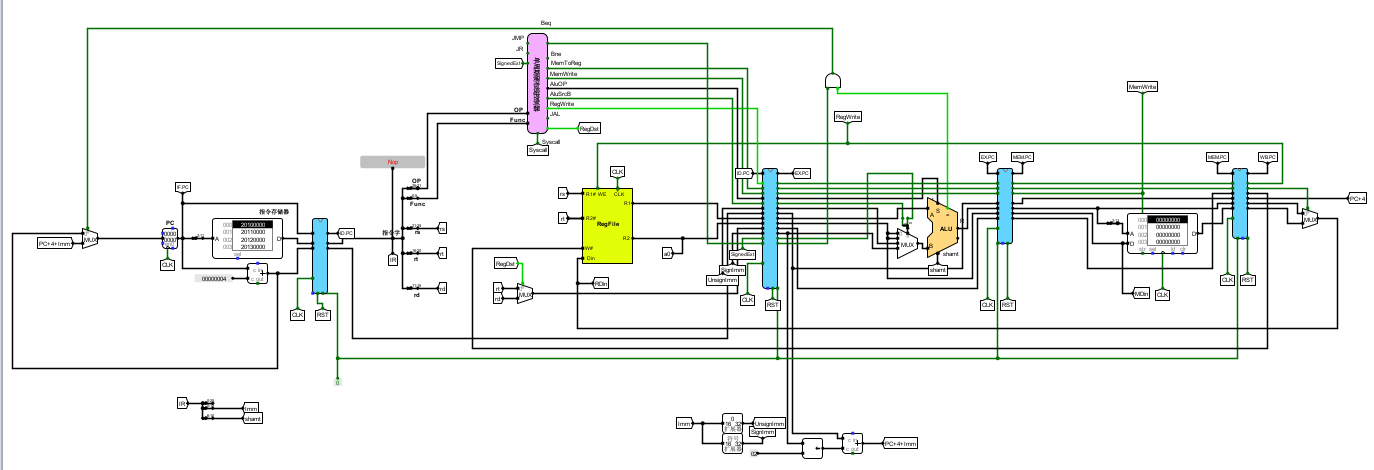


图 3.17 理想流水线通路

## 气泡式流水线实现

1. 在理想流水线的基础上补充跳转指令的逻辑并完善通路。
2. 实现流水线分支跳转逻辑
   1. 完成EX段分支地址与IF段PC输入多路选择器选择信号的连接。在EX段计算出分支跳转的地址，并作为PC寄存器的选择端输入。
   2. 实现分支相关的气泡逻辑。当EX段为bne指令且equal为0或为beq指令equal为1时或者为无条件转移指令JMP信号为1或JR信号为1时产生分支清零的信号。
3. 实现数据相关检测逻辑。
   1. 构建源寄存器使用情况子电路。输入为OP,Funct，输出为R1\_USED,R2\_USED。填写控制器真值表利用Logisim自动生成电路。表如图3.18所示。电路如图3.19所示

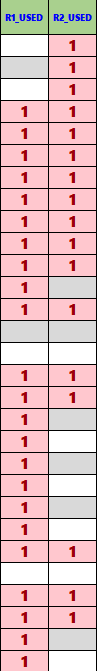
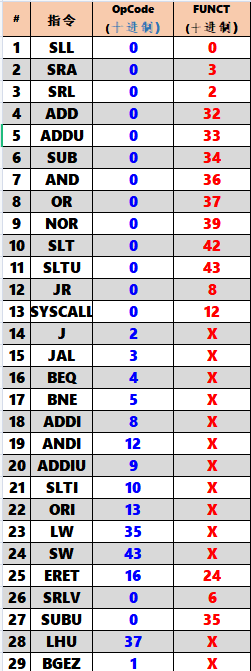
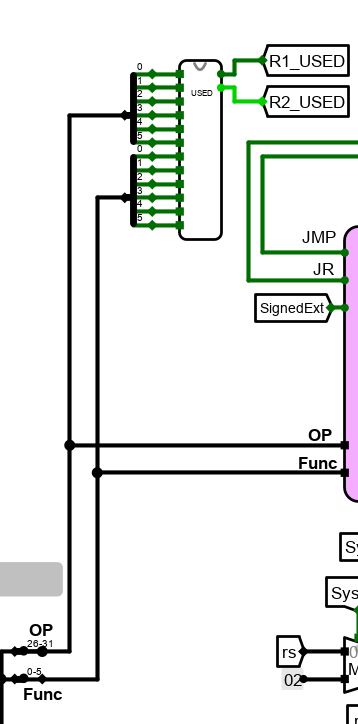
 

图 3.18 源寄存器使用情况真值表 图 3.19 源寄存器使用信号产生电路

* 1. 构建数据相关检测逻辑子电路。输入为R1\_USED, R2\_USED, R1#, R2#, EX.WriteReg#, MEM.WriteReg#, RegWrite，输出为数据相关信号DataDizzard、EX相关信号EX\_CONFLICT和MEM相关信号MEM\_CONFILCT。其实现的逻辑表达式如下：
* rs,rt 分别表示指令字中的rs,rt字段，分别对应指令字中的25~21,20~16位
* RsUsed, RtUsed分别表示ID段指令需要读rs, rt字段对应的寄存器
* EX.RegWrite表示EX段的写寄存器堆写使能控制信号RegWrite，锁存在 ID/EX流水寄存器中
* MEM.WriteReg#表示MEM段的写寄存器编号WriteReg#，锁存在EX/MEM 流水寄存器中

对应的电路实现如图3.20所示。

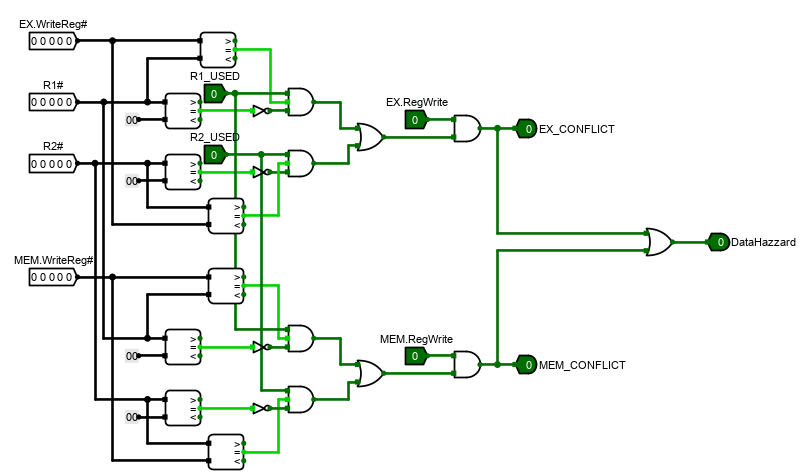


图 3.20 数据相关检测逻辑电路

1. 实现ID，ID段暂停逻辑和EX段气泡逻辑

各寄存器的暂停和清除信号表达式如下：

* Stall = DataHazzard # 数据相关时要阻塞暂停 IF、ID 段指令的执行
* PC.EN = ~Stall # 程序计数器 PC 使能端输入
* IF/ID.EN = ~Stall # IF/ID 寄存器使能端输入
* IF/ID.CLR = BranchTaken # 出现分支跳转时要清空 IF/ID
* ID/EX.CLR = Flush = BranchTaken + DataHazzard # 出现分支或数据相关时要清空 ID/EX

其实现电路如图3.21所示。

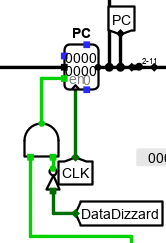
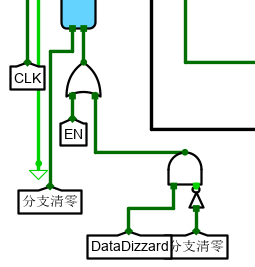
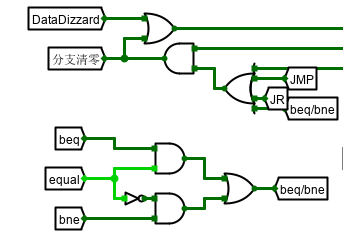
  

图 3.21 暂停和气泡逻辑实现电路

## 重定向流水线实现

重定向流水线将在气泡流水线的基础上进行一定的修改。主要分为四个步骤：

1. 构建重定向通路。需要在第一次使用寄存器你的位置增加多路选择器并连接可能的重定向通路。通路图3.23如所示
2. 构建重定向逻辑。在ID段根据数据相关情况产生对应的重定向控制信号。以RsForward例，其表达式如下：

* IF (RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#))

RsFoward = 2 # ID 段与 EX 段数据相关

else IF (RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#))

RsFoward = 1 # ID 段与 MEM 段数据相关

else RsFoward = 0 # 无数据相关

1. 构建Load-Use检测器。Load-Used的逻辑表达式如下：

* LoadUse = RsUsed & (rs≠0) & EX.MemRead & (rs==EX.WriteReg#)

+ RtUsed & (rt≠0) & EX.MemRead & (rt==EX.WriteReg#)

* 注意单周期 CPU 实现中为了简化电路，只实现了 MemWrite 写信号，没有实现 MemRead 信号，但由于该信号和 MemToReg 信号是同步的，所以可以用 MemToReg 信号代替 MemRead 信号

最终重定向选择信号和数据冲突检测信号合并的子电路设计如下：

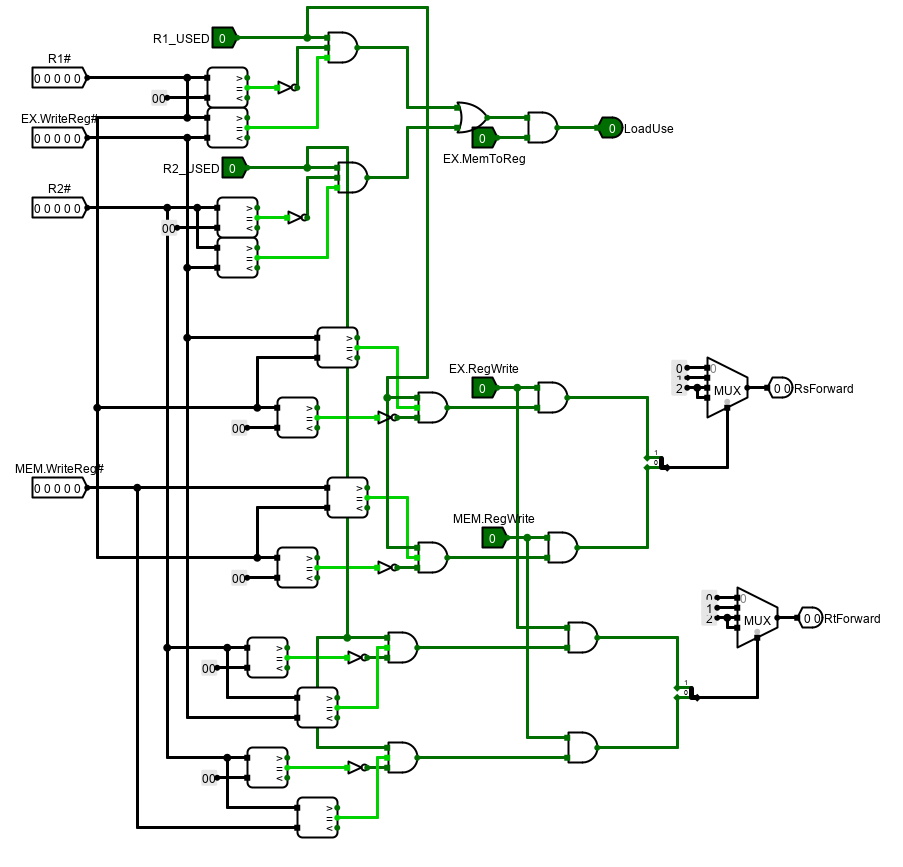


图 3.22重定向选择信号和数据冲突检测信号实现电路

1. 在ID段增加Load-Use插入气泡逻辑。在气泡流水的基础上将DataDizzard信号全部换为Load-Used信号即可。

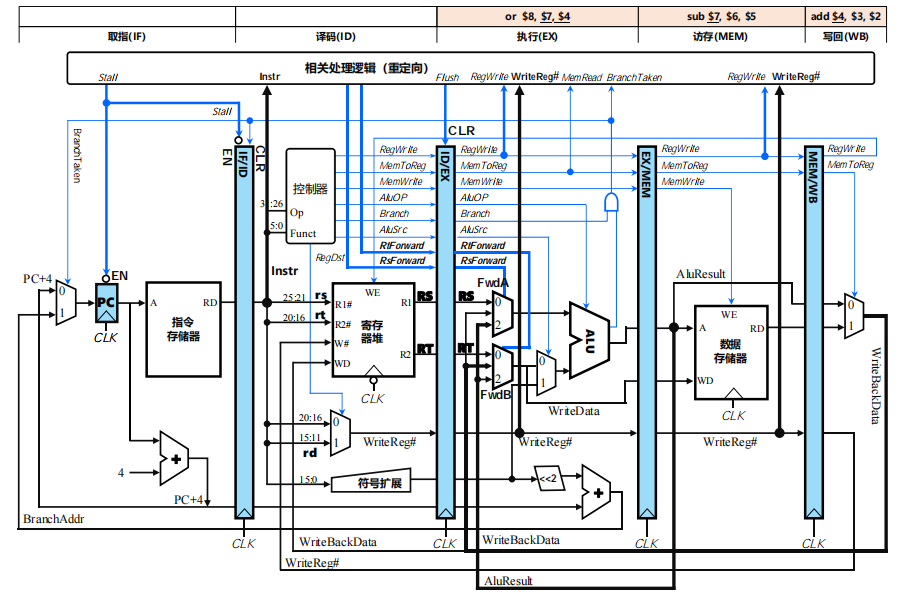


图 3.23 重定向流水线数据通路

## 动态分支预测机制实现

1. 历史预测位状态转换机的实现。

根据图2.5所示的双预测位历史状态转换图，在Logism中填写对应的真值表即可自动生成状态机电路。如图3.24所示。

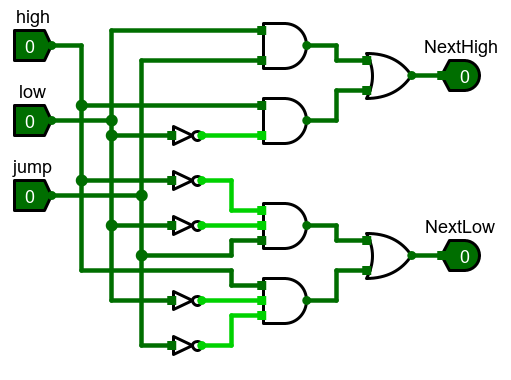
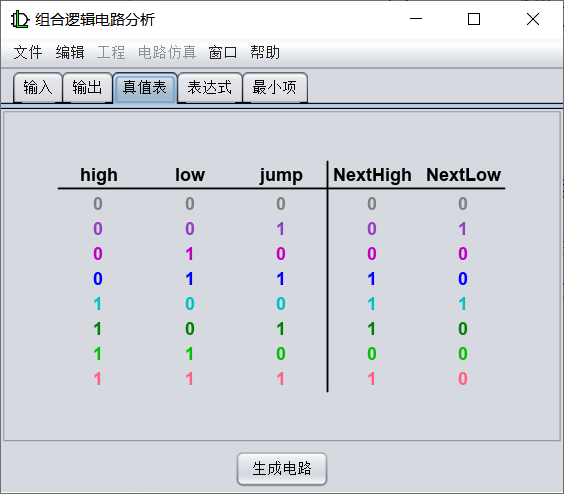


图 3.24双预测位状态转换机真值表和实现电路

1. 实现BTB8项全相联cache结构
2. cache槽。

包括有效位、标志、淘汰计数器和分支跳转地址。淘汰计数器采用LRU的思想，淘汰最近不常使用的槽，即淘汰时，计数器计数值最大的cache槽，而在该cache槽被淘汰或被写write信号为1或者命中L信号为1时将计数置零重新开始表示最近被使用。Tag寄存器中保存着曾经跳转执行成功过的分支指令地址。有效位为1表示当前cache槽内有数据。EX的分支目标地址在命中且写信号为1时触发寄存器写入输出为BTB的预测分支目标地址。

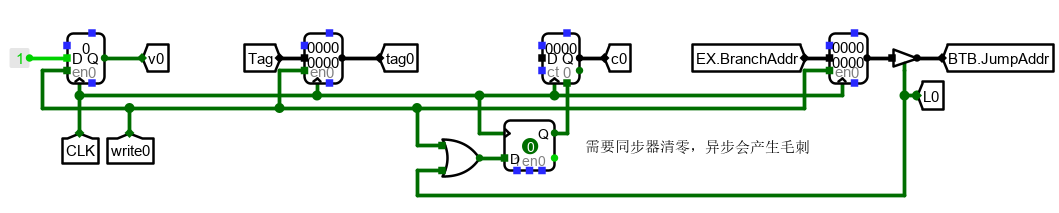


图 3.25 cache槽

1. 查找、写入和更新预测位逻辑

根据图 BTB的处理流程，当前IF段的PC值和BTB中的分支指令地址进行比较查找。如果比较结果相等且当前有效位为1则产生L命中信号，并且根据分支预测实力位的值输出预测跳转信息位PredictJump。当EX阶段执行分支指令，即EX.Branch为1时，与BTB中的分支指令地址进行比较查找，如果找到产生M命中信号，此时需要根据EX段的实际跳转情况EX.Branched更新预测位状态机中的信息。如果BTB中所有的槽的M信号都为0表示这个分支地址缺失，需要写入，且如果此时BTB是满的，采用LRU算法淘汰掉最近最不常使用的槽。

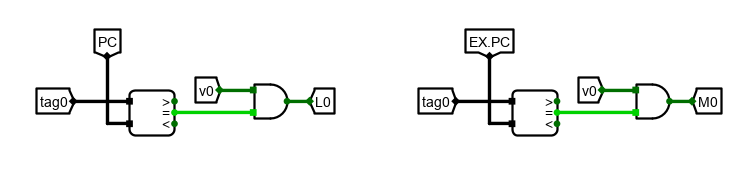


图 3.26 查找逻辑和EX写入逻辑

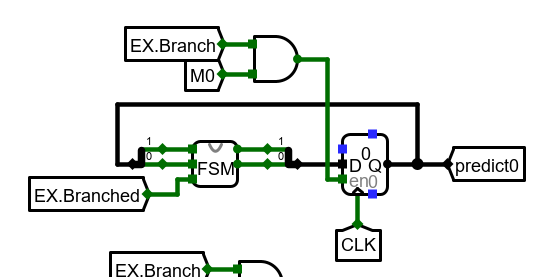


图 3.27 预测位状态机更新逻辑

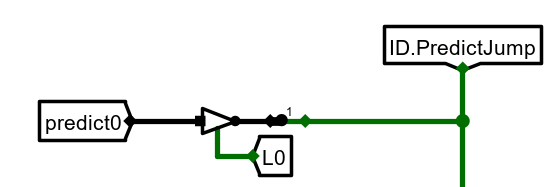


图 3.28 预测跳转信息逻辑

图3.27中FSM为3.6节中第1步实现的状态转换机封装器件，其初始状态为00，输入为当前预测位状态，输出为预测后的下一个状态，该状态保存在寄存器中并由两位状态字的高位判断预测是否正确，高位为1则预测正确，为0为预测失败，在当前行命中的前提下输出预测结果为PredictJump信号。

在具体的BTB实现中，由于设计了8行cache槽，图3.25，3.26，3.27，3.28的设计都要重复8遍，并修改tag0, v0, L0, M0,predicte0为所在行的标签。

LRU算法的实现如图3.29所示。由8行cache的命中信号M互相或产生命中信号HIT，当有一个为1时，即为命中。由8行cache的有效标志位v互相与形成判断cache是否满的信号FULL，当所有行的标志为都为1时代表为满，FULL输出为1。当发生写入需要分两种情况，一种是cache已满需要利用LRU算法淘汰，一种是还没满选择一个空行写入，所以设计一个选择器，选择端为FULL信号，当FULL为0时代表未满，此时8行的标志位取非后送入优先编码器，这样当v为0（未写）取非后为1，通过优先编码器输出编号小的行，而当FULL为1时，将8行的淘汰计数器c通过MAX3比较器两两比较大小，最终输出最大计数的行编号。选择器的输出端为需要写入的行的编号，通过一个带有使能端的译码器，在当前指令为分支指令且缺失的前提下译码为所在行的写信号write。

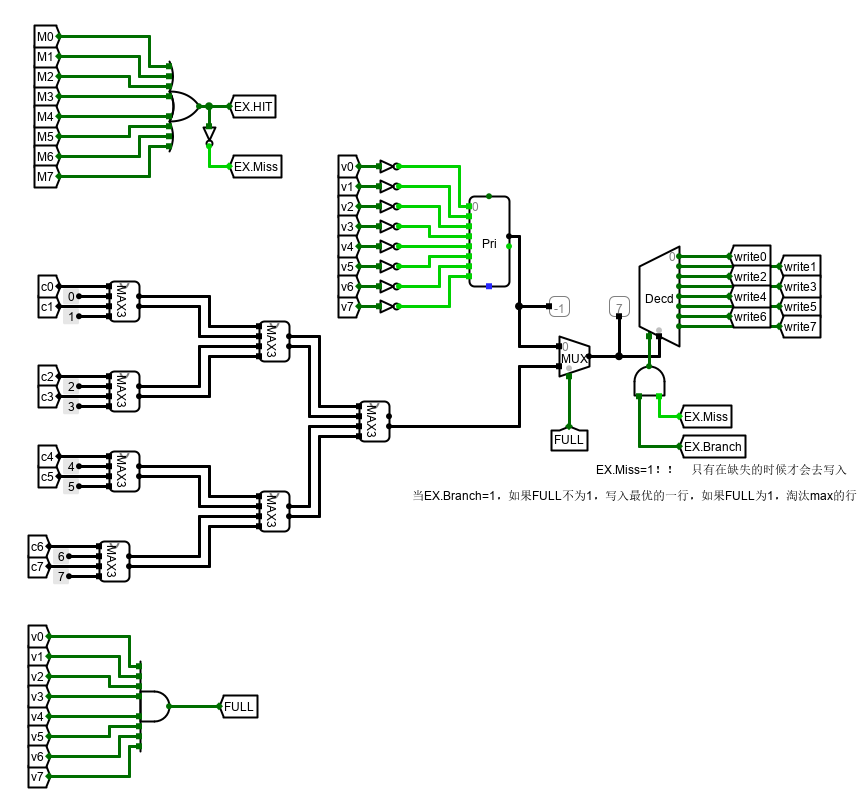


图 3.29 cache槽写信号产生和选择淘汰逻辑

1. 根据图2.7给出的动态分支预测通路完成连线。包括BTB连接和气泡产生部分。气泡产生的实现是当BTB预测失败PredictErr信号为1时，分别输送到IF/ID和ID/EX的清零端。其表达式如下：

* PredictErr = ~(BranchTaken==PredictJump)
* IF/ID.CLR = PredictErr
* ID/EX.CLR = LoadUse + PredictErr

PredictErr的实现如图3.30所示，BTB的主要实现部分如图3.31所示。

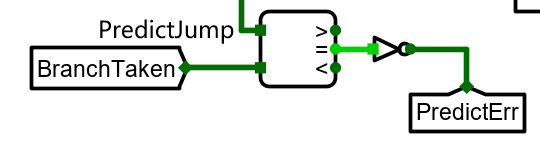


图 3.30 PredictErr控制信号设计

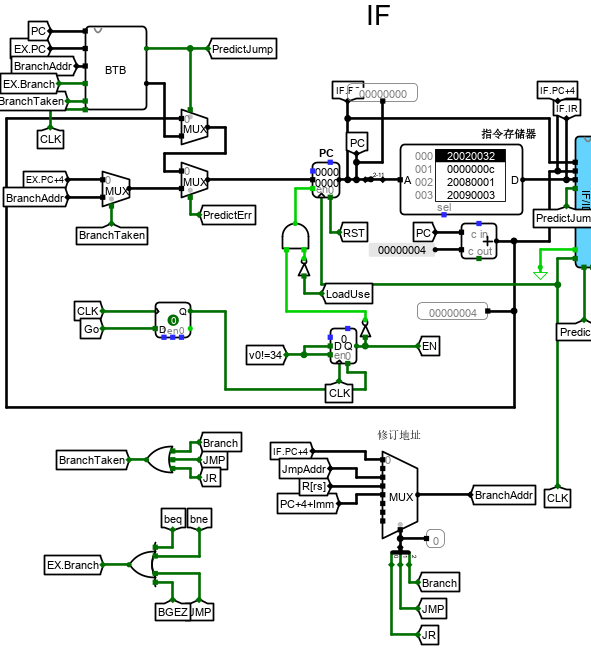


图 3.30 BTB主要实现部分

# 实验过程与调试

## 性能分析

运行同一段程序bechmark，单周期电路周期数为1546，气泡流水线的周期数为3624，重定向流水线的周期数为2298，使用动态分支预测后的重定向流水线为1782。流水线的时钟周期数虽然比单周期多，但是在单位时间内处理的指令数却大于单周期。遇到分支指令和数据相关较多的程序时，气泡流水线会产生较多的气泡和停顿，而重定向流水由于很大程度上解决了数据冲突，仅在Load指令时需要进行停顿，性能上相比气泡流水要改善很多，性能提高约1.53倍。由于程序的空间局部性和时间局部性，程序会执行重复的操作，动态分支预测基于这一点采用合理的算法提前预测下一次的分支，能够减少由于分支指令带来的气泡，性能提高了0.78倍。

## 主要故障与调试

### BTB淘汰计数故障

**故障现象：**测试BTB子电路，当cache槽全部满时，需要根据淘汰计数器的值选择最大的进行淘汰将计数器，被选中淘汰的cache槽的计数器的将在write信号下清零，但是调试发现，尽管确实只选中了一行进行淘汰，也确实只有一行的write信号为1，但最后所有行的计数器都被清零。

**原因分析：**在计数器的清零是通过write信号和L命中信号相或直接输入到计数器的清零端的，但是该清零端是异步清零，会产生毛刺导致电路出现异常。

**解决方案：**将write信号和L命中信号或的结果通过一个D触发器在输送到清零端就可以实现同步清零。和这个同类型的问题还有点击Go按钮实现继续执行程序但是显示异常，因为Go按钮也是异步清零需要改为同步清零。

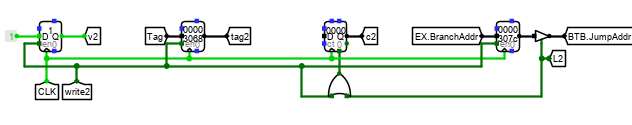


图 4.1异步清零计数器

### 气泡流水执行故障

**故障现象：**如图 4.2所示，在Educoder平台上测试气泡流水线的电路时发现第4个节拍指错字PcErr值为11100。IFPC，IDPC，EXPC值均是错误的。调试时发现第四个节拍的PC值应该为指令PC值但我的却为空指令，第三个节拍我的电路检测到数据冲突但是分析发现此时不应该存在冲突。



图 4.2程序故障图

**原因分析：** 因为第三个节拍产生了错误的数据冲突信号导致第四个节拍的指令被清空，所以问题在于数据检测子电路。但是经过仔细分析数据检测子电路的逻辑和连线后发现没有什么问题，那么肯定是产生R1\_USED和R2\_USED信号电路的问题，应该是填写表格3.17时出现错误。

**解决方案：**阅读资料发现R1\_USED和R2\_USED信号是在ID进行检测的，而不是说如果指令在实现的过程中用到了rs，rt寄存器就应该在表格中填入1，应该对照表2.4中RF列填写的情况完成表 。

## 实验进度

表 4.1 课程设计进度表

| 时间 | 进度 |
| --- | --- |
| 寒假期间 | 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，完成了单周期MIPS和单中断的设计。 |
| 上课第一天 | 阅读有关多级中断的资料，大致完成了多级中断的通路。 |
| 第三天 | 完善设计并调试多级中断的线路，并通过了Logisim多级中断的检查。 |
| 第四天 | 复习关于指令流水线的知识点，完成理想流水线的电路设计，并通过了Educoder平台上的理想流水线检查，观看了有关气泡流水线的慕课并大致完成了流水寄存器的部件设计。 |
| 第五天 | 完成了气泡流水线和重定向流水线的电路设计并通过了Educoder上的测试。 |
| 第六天 | 阅读有关流水中断的资料，尝试在单级中断的基础上修改为流水中断 |
| 第七天 | 大致完成了流水中断的通路设计，并在24条指令的基础上添加了自己的4条差异化指令。 |
| 第八天 | 完成了流水中断的设计并通过了自己的测试。 |
| 第九天 | 阅读有关动态分之预测的资料，并设计了一部分BTB的全相联cache结构 |
| 第十天 | 大致完成了BTB全相联存储器的设计和BTB的通路连接 |
| 第十一天 | 完成了动态分支预测电路的设计，最终结果程序的运行周期为1782 |

# 团队任务

## 团队任务设计

本次团队设计的是一个简单的计算器，能够实现十以内的加减乘法计算。使用Logisim自带的keyboard器件和文本显示器件，点击keyboard之后可以输入算术表达式，表达式结果会通过LED显示。

本人在这次实验中主要负责程序代码的实现，键盘输入的字符表达式包括两个操作数和一个运算符，这三个字符输入后会保存在数据存储器DM中，程序代码设计时就通过lw指令获取两个操作数和字符并分别放到s0,s2,s1寄存器中，通过beq指令依次比较s1寄存器和加法符号’+’、减法符号’-’、乘法符号’\*’的十六进制值，当分支成功时跳到相应的运算程序部分。运算部分的结果获得仅需要通过相应的运算指令，如add, sub, mul。结果的显示通过syscall指令完成，在每次结果计算完毕后使用j无条件跳转到显示程序部分。由于除法指令div经过译码后包含了诸如break、mfl0等指令，设计起来比较复杂所以最后舍弃了除法的实现。

## 团队分工

本次实验的分工如下：

* 数据通路的搭建和构造——刘汉鹏
* 程序代码的实现——刘美
* github资料的查阅及整理——吴志远
* 实验日志的记录——凯撒·衣马艾木山

# 设计总结与心得

## 课设总结

本次实验完成了MIPS单周期和5段流水线的设计，并尝试了不同的设计方案来解决MIPS指令流水线中的各类冒险冲突，最终设计完成的CPU能够运行基本的24条指令和差异化指令，并进一步增加了中断异常处理机制。作了如下几点工作：

1. 单周期MIPS设计。主要难点在于数据通路的设计。
2. 单级中断和多级中断的设计。主要在于理解异步中断的执行过程，设计为了实现中断要满足的硬件和软件方面的部分。多级中断重点设计一个EPC堆栈实现嵌套。
3. 5段流水线设计。主要在于数据通路以及堆冒险冲突的处理设计和改进。

## 课设心得

本次课设的整体难度较大。前两个实验单周期MIPS和单级中断主要是对上学习课程实验的一个过渡和提升，细心完成控制信号真值表的填写和数据通路的连线以及理解中断执行过程的话还是很好实现的。后面的气泡流水以及重定向流水虽然是新的知识但是有详细的样章讲解实现起来也不是很难。主要难点在于多级中断、流水中断和动态分支预测。这两个内容都是书上和慕课都提的比较少的，多级中断难在EPC堆栈的构建，流水中断难在中断响应的阶段选择，动态分支预测则是难在BTBcache设计上。通过本次实验我对如何实现中断嵌套处理和如何设计流水线并做出改善有了更深的理解。

关于意见，首先是上板，感觉老师提供的资料有点少，让对于Verilog和Vivado不熟悉的我很头疼，遇到的一些问题以及仿真上板的步骤都不是很清楚。最后还是依赖同学的野生资料（虽然上板还是失败了）。其次就是团队任务这一部分，感觉对团队的合作要求不高，整体的工作量一个人也可以完成。

最后非常感谢课程老师在本次实验中提供的许多帮助！

# 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. 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011年.
5. 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.
6. 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.
7. 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: 刘美 IMG_0249** |