***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： | CS2108 |
| 学 号： | U202115563 |
| 姓 名： | 杨道文 |
| 电 话： | 13603786922 |
| 邮 件： | [519561524@qq.com](mailto:519561524@qq.com) |

目 录

[1 课程设计概述 3](#_Toc148276690)

[1.1 课设目的 3](#_Toc148276691)

[1.2 设计任务 3](#_Toc148276692)

[1.3 设计要求 3](#_Toc148276693)

[1.4 技术指标 4](#_Toc148276694)

[2 总体方案设计 6](#_Toc148276695)

[2.1 单周期CPU设计 6](#_Toc148276696)

[2.2 中断机制设计 10](#_Toc148276697)

[2.3 理想流水线设计 12](#_Toc148276698)

[2.4 EX段分支的气泡流水线设计 14](#_Toc148276699)

[2.5 EX段分支的重定向流水线设计 14](#_Toc148276700)

[2.6 动态分支预测机制 15](#_Toc148276701)

[3 详细设计与实现 17](#_Toc148276702)

[3.1 单周期CPU 实现 17](#_Toc148276703)

[3.2 中断处理机制实现 21](#_Toc148276704)

[3.3 理想流水线实现 25](#_Toc148276705)

[3.4 EX段分支的气泡流水线实现 27](#_Toc148276706)

[3.5 EX段分支的重定向流水线实现 29](#_Toc148276707)

[3.6 动态分支预测机制实现 31](#_Toc148276708)

[4 实验过程与调试 33](#_Toc148276709)

[4.1 测试用例和功能测试 33](#_Toc148276710)

[4.2 性能分析 34](#_Toc148276711)

[4.3 主要故障与调试 34](#_Toc148276712)

[4.4 实验进度 36](#_Toc148276713)

[5 团队任务 雷霆战机 37](#_Toc148276714)

[5.1 项目介绍 37](#_Toc148276715)

[5.2 任务分配 37](#_Toc148276716)

[5.3 任务难点 37](#_Toc148276717)

[5.4 项目实现 37](#_Toc148276718)

[6 设计总结与心得 41](#_Toc148276719)

[6.1 课设总结 41](#_Toc148276720)

[6.2 课设心得 41](#_Toc148276721)

[参考文献 43](#_Toc148276722)

# 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。

## 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. 支支持规定的 32 位 RISC-V 指令集（指令集任选），具体见表 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，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，即srl，auipc，lbu，bge。；
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. 团队开发一个具有展示度的演示系统，有输入输出设备，完整的软硬件系统，我们组实现的是雷霆战机。

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 | AND | 与 |
| 4 | ANDI | 立即数与 |
| 5 | SLL | 逻辑左移 |
| 6 | SRA | 算数右移 |
| 7 | SRL | 逻辑右移 |
| 8 | SUB | 减 |
| 9 | OR | 或 |
| 10 | ORI | 立即数或 |
| 11 | XORI | 立即数异或 |
| 12 | LW | 加载字 |
| 13 | SW | 存字 |
| 14 | BEQ | 相等跳转 |
| 15 | BNE | 不相等跳转 |
| 16 | SLT | 小于置数 |
| 17 | STI | 小于立即数置数 |
| 18 | SLTU | 小于无符号数置数 |
| 19 | JAL | 转移并链接 |
| 20 | JALR | 转移到指定寄存器 |
| 21 | ECALL | 系统调用 | If $v0==10 halt(停机指令)  else数码管显示$a0值 |
| 22 | CSRRSI | 访问CSR | 中断相关，可简化为开中断 |
| 23 | CSRRCI | 访问CSR | 中断相关，可简化为关中断 |
| 24 | URET | 中断返回 | 清中断，mEPC 送 PC |
| 25 | Srl | 符号右移 | C-2 |
| 26 | AUIPC | 扩展指令（挺难的） | C-5 |
| 27 | LBU | 加载无符号byte | A-2 |
| 28 | BGE | 大于等于跳转 | B-2 |

# 总体方案设计

## 单周期CPU设计

本阶段使用硬布线控制器原理实现RISC-V单周期CPU，支持中断相关以外的24条基本指令和4条差异性指令。

总体结构图如图 2.1所示。

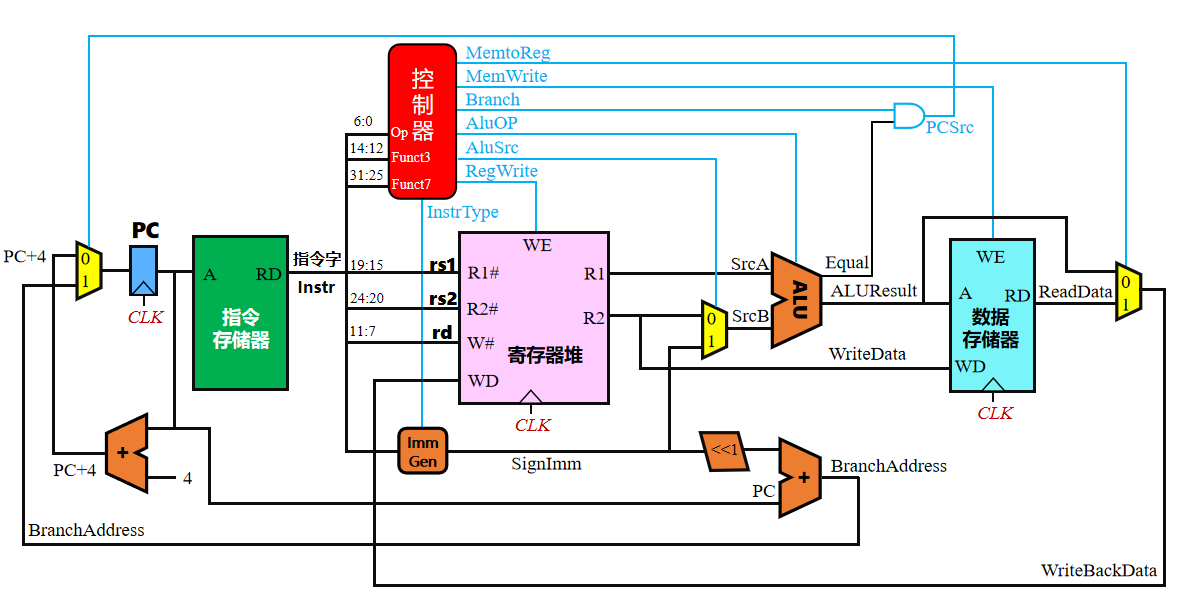


图 . 总体结构图

### 主要功能部件

该结构中的主要功能部件有程序计数器PC，指令存储器IM，运算器ALU，寄存器堆RF，数据存储器DM，下面介绍这几个部件的基本功能和接口作用。

#### 程序计数器PC

程序计数器用于寄存指令地址，同步时钟控制，输入端接下一条指令在存储器中地址，该地址由当前指令类型决定，使能端由计算周期数的计数器决定，输出送至指令存储器，从而取出该地址对应的指令内容。

#### 指令存储器IM

指令存储器存储着该程序地址空间对应的指令，输入为程序计数器输出的地址的2-11位，输出送至后续部件执行该指令的各项具体操作。

#### 运算器ALU

运算器的引脚和功能描述如下表。

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，具体功能表 2.3 |
| 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

寄存器堆的引脚和功能描述如表 2.2。

表 2.2寄存器堆引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| R1# | 输入 | 32 | 寄存器1编号 |
| R2# | 输入 | 32 | 寄存器2编号 |
| W# | 输入 | 5 | 写入寄存器R3 |
| Din# | 输入 | 32 | 写入值 |
| WE | 输入 | 1 | RegWrite，决定是否将写入值写入R3 |
| CLK | 输入 | 1 | 时钟 |
| R1 | 输出 | 32 | 寄存器1数值 |
| R2 | 输出 | 32 | 寄存器2数值 |

#### 数据存储器DM

同步时钟控制，A端口为地址输入端，D为数据输入端，str为MemWrite，决定是否将D写入对应的A地址下的空间，输出为A地址对应的存储内容。

### 数据通路的设计

为了使各基本原件协调工作并实现指令功能，那么我们需要设计数据通路将一系列操作串联并联起来，数据通路的框架如表 2.3所示。

表 2.3指令系统数据通路框架

| 指令 | PC | IM | RF | | | | ALU | | | DM | | Tube |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1# | R2# | W# | Din | A | B | OP | Addr | Din |
| ADD | PC+4 | PC | rs1 | rs2 | rd | ALUresult | R1 | R2 | 5 | -- | -- | -- |
| SUB | PC+4 | PC | rs1 | rs2 | rd | ALUresult | R1 | R2 | 6 | -- | -- | -- |
| AND | PC+4 | PC | rs1 | rs2 | rd | ALUresult | R1 | R2 | 7 | -- | -- | -- |
| OR | PC+4 | PC | rs1 | rs2 | rd | ALUresult | R1 | R2 | 8 | -- | -- | -- |
| SLT | PC+4 | PC | rs1 | rs2 | rd | ALUresult | R1 | R2 | 11 | -- | -- | -- |
| SLTU | PC+4 | PC | rs1 | rs2 | rd | ALUresult | R1 | R2 | 12 | -- | -- | -- |
| ADDI | PC+4 | PC | rs1 | -- | rd | ALUresult | R1 | signed(imm) | 5 | -- | -- | -- |
| ANDI | PC+4 | PC | rs1 | -- | rd | ALUresult | R1 | signed(imm) | 7 | -- | -- | -- |
| ORI | PC+4 | PC | rs1 | -- | rd | ALUresult | R1 | signed(imm) | 8 | -- | -- | -- |
| XORI | PC+4 | PC | rs1 | -- | rd | ALUresult | R1 | signed(imm) | 9 | -- | -- | -- |
| SLTI | PC+4 | PC | rs1 | -- | rd | ALUresult | R1 | signed(imm) | 11 | -- | -- | -- |
| SLLI | PC+4 | PC | rs1 | -- | rd | ALUresult | R1 | zero(imm) | 0 | -- | -- | -- |
| SRLI | PC+4 | PC | rs1 | -- | rd | ALUresult | R1 | zero(imm) | 2 | -- | -- | -- |
| SRAI | PC+4 | PC | rs1 | -- | rd | ALUresult | R1 | signed(imm) | 1 | -- | -- | -- |
| LW | PC+4 | PC | rs1 | -- | rd | ReadData | R1 | signed(imm) | 5 | -- | -- | -- |
| SW | PC+4 | PC | rs1 | rs2 | -- | -- | R1 | signed(imm) | 5 | ALUResult | R2 | -- |
| ECALL | PC+4 | PC | 17 | 10 | -- | -- | -- | -- | -- | -- | -- | -- |
| BEQ | PC+4/PC+E(imm)<<1 | PC | rs1 | rs2 | -- | -- | R1 | R2 | -- | -- | -- | -- |
| BNE | PC+4/PC+E(imm)<<1 | PC | rs1 | rs2 | -- | -- | R1 | R2 | -- | -- | -- | -- |
| JAL | PC+E(imm)<<1 | PC | -- | -- | rd | PC+4 | -- | -- | -- | -- | -- | -- |
| JALR | R1+E(imm) | PC | -- | -- | rd | PC+4 | -- | -- | -- | -- | -- | -- |
| SRL | PC+4 | PC | rs1 | rs2 | rd | ALUresult | R1 | R2 | C | -- | -- | -- |
| AUIPC | PC+4 | PC | -- | -- | rd | ALUresult | PC | U型(imm) | 5 | -- | -- | -- |
| L**B**U | PC+4 | PC | rs1 | -- | rd | ReadData | R1 | unsigned(imm) | 0 | -- | -- | -- |
| BGE | PC+4/PC+E(imm)<<1 | PC | rs1 | rs2 | -- | -- | R1 | R2 | 18 | -- | -- | -- |

### 控制器的设计

完成数据通路后，接下来就需要实现对数据通路以及运算器的控制。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 2.4。

表 2.4主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| AluOP | 0-12 | 运算器操作控制符（4位） |
| RegWrite | 0/1 | 寄存器写使能 |
| MemWrite | 0/1 | 写内存控制信号 |
| MemToReg | 0/1 | 寄存器写入数据来自存储器 |
| 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，用于有条件分支控制 |
| BNE | 0/1 | Bne指令，用于有条件分支控制 |
| ECALL | 0/1 | 根据$a7寄存器的值，决定是停机还是输出 |
| S-Type | 0/1 | S类指令译码信号 |
| CSRRSI | 0/1 | CSRRSI指令信号 |
| CSRRCI | 0/1 | CSRRCI指令信号 |
| BGE | 0/1 | BGE指令信号 |
| LBU | 0/1 | LBU指令信号 |
| AUIPC | 0/1 | AUIPC指令信号 |

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控制信号表所示。



图 2.2控制信号表

## 中断机制设计

### 总体设计

在单周期CPU的基础上加上中断机制，支持3个外部按键中断源，中断实验部分分为单级中断，多级中断和流水单级中断，中断的实现主要是以下几个阶段。

* 1. 中断当前程序并保护断点
  2. 转入中断服务入口
  3. 保护现场
  4. 执行中断服务程序
  5. 恢复现场
  6. 中断返回

1. **单级中断**

单级中断的重点在于如何和何时保存PC、加载中断处理程序的地址、恢复PC。当一个中断成功申请以后，先关闭中断，保存断点，再将中断地址送入PC。引入寄存器EPC来保存断点地址，用一个中断使能寄存器IE保存开关中断情况，用寄存器INT来保存当前中断号。中断返回指令为URET指令，此时开中断，将EPC中存的断点地址送入PC，清空相应中断的请求和INT寄存器内数据。

单周期中断要支持3个外部按键中断源，且中断优先级 1 < 2 < 3，CPU 执行中断服务程序时不能被其他中断请求中断。我们使用优先编码器选择优先级由高至低的中断信号分步执行。

1. **多级中断**

在单级中断的基础上实现多级中断，差异在于当前中断程序被优先级更高中断信号打断的情况和使用硬件栈来保存返回地址以支持嵌套中断

多级中断增加了csrrsi、csrrci指令数据通路，在本实验中用于实现开中断，关中断。另外，在修改中断服务程序时，相对单级中断，多重中断中mEPC寄存器也必须作为现场进行保护，如采用内存堆栈方式保护mEPC寄存器，则必须通过csrrsi指令读取mEPC寄存器的值到通用寄存器后压栈，恢复现场时从堆栈弹出mEPC寄存器的值，恢复现场完毕URET后开中断。另外多重中断保护现场结束后应开中断，需要执行csrrsi指令设置IE使能位，恢复现场之前也需要关中断、同样需要执行csrrci指令设置IE使能位。

1. **流水单级中断**

流水单级中断在重定向流水电路的基础上进行修改。响应中断时，IF和ID段这两处未进入执行阶段的指令被清空，EX,MEM和WB段继续执行。CPU响应中断，EX段的指令被打断，将正在执行的指令的下一个指令，即ID段指令的地址写入EPC中作为断点保存。恢复现场完毕URET后，由于返回指令是在ID阶段产生URET信号的，故将IF段指令清空防止其进入ID段。

### 硬件设计

1. **单级和流水单级中断**

这两种中断处理机制的硬件设计基本一致。中断信号产生后，若同时有多个中断请求信号，采用优先编码器决定先响应的中断号。根据当前中断号以及URET信号来清除中断请求。EPC由32位寄存器实现，CPU响应中断信号后将断点地址写入EPC。IE由1位寄存器实现，若IE值为0，则开中断，为1，关中断。是否响应当前中断由中断请求和IR值同时决定。响应中断信号时，IE值为1，关中断，执行URET时IE值为0，开中断。用一个2位寄存器INT保存当前正在处理的中断号。

1. **多级中断**

将主程序认为是优先级为0的中断程序，即可将主程序和中断一起考虑。将优先编码器得到的中断号IntNum和现在正在运行的中断号old\_INT进行比较判断当前是否有一个更高优先级的中断。用一个IE控制的多路选择器让IntNum通过，这样IE为0的时候参与比较的IntNum不会变化，从而实现屏蔽中断的功能，且不会受时钟影响。硬件栈存储返回地址，即将原来的EPC寄存器增加至三个作为栈，栈顶存放一个计数器，通过译码器来转换为硬件栈的使能信号加载 NextPC。当 ReadyInt 时栈顶增加，URET信号产生时栈顶减小。此外，用一个IntRunning的控制信号表示当前最高等级的中断程序是否在运行。只有IntRunning为0且两个中断号不一样时，才可产生INT信号。

### 软件设计

使用RARS得到中断处理程序的入口地址。

## 理想流水线设计

### 总体设计

流水线由取指、译码、执行、访存、写回五部分组成。取指是从指令寄存器里取 得指令；译码是将指令转换为控制信号，取得一些立即数或者寄存器值；执行是利用ALU进行运算，或者计算一些分支地址；访存是读写内存；写回是将计算结果或者访存结果写回寄存器堆。

在这五个阶段设置流水接口部件，分别命名为 IF/ID、ID/EX、EX/MEM、MEM/WB，其本质就是流水寄存器，所有流水寄存器，程序计数器PC、寄存器堆、数据存储器均采用统一时钟CLK进行同步，每来一个时钟，就会有一条新的指令进入流水线取指令IF段，同时流水寄存器就会锁存前段加工处理完成的数据和控制信号，为下一段的功能部件提供数据输入，指令流水线各功能段通过流水寄存器完成一次数据传送。

### 流水接口部件设计

流水寄存器，用于锁存本段处理完成的所有数据或结果，以保证本段的执行结果能在下一个时钟周期给下一个阶段使用，采用统一公共时钟进行同步，每来一个时钟，各段组合逻辑功能部件处理完成的数据将会锁存到段尾的流水寄存器中，作为后段的 输入，同时当前段也会接收到前段通过流水寄存器传递过来的新指令或数据。

为简单起见，我们不区分不同阶段流水寄存器的接口差别，统一设计一个包括所有接口的部件。

因此，流水部件要传递指令控制信号、寄存器读取结果、ALU运算结果、内存读取结果、指令中立即数、PC、IR等内容。此外，流水线部件还要支持时钟、使能端、同步清零、暂停功能。

### 硬件设计

由于理想流水线不考虑冲突或冒险问题、不存在资源冲突，且进入流水线的指令相互之间不存在依赖关系或相关关系，在图 2.3五段流水控制信号与传递的基础上按照单周期CPU的实现原理进行简单改动即可。

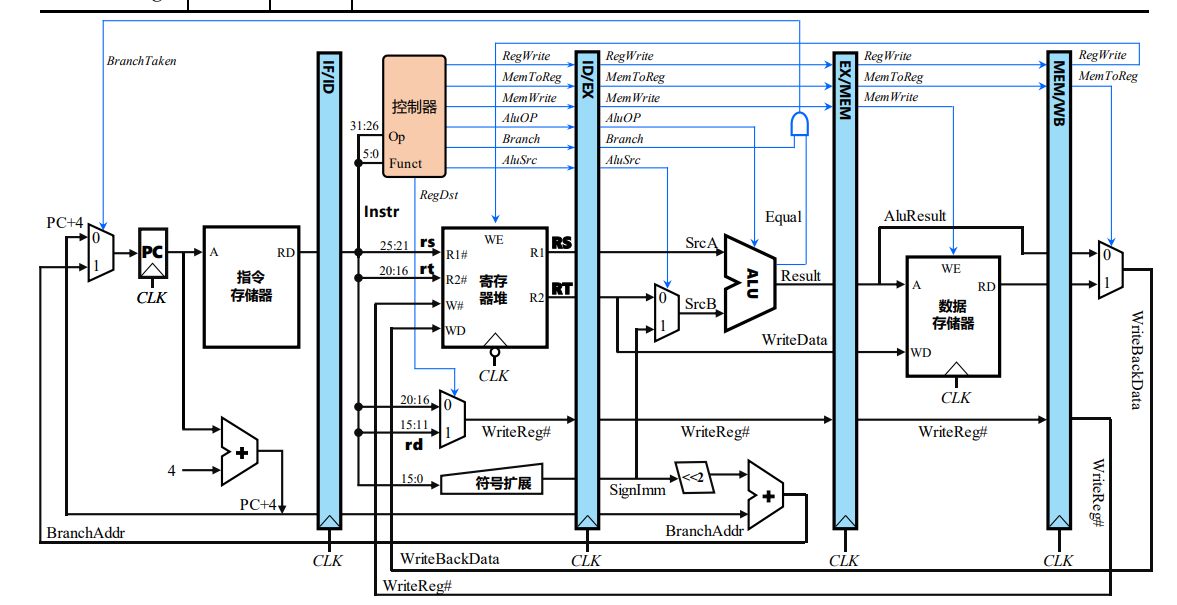


图 2.3五段流水控制信号与传递

## EX段分支的气泡流水线设计

### 总体设计

气泡机制的引入，是为了解决数据相关这个问题。为了解决数据相关问题，最简单的方法就是让后来的指令等待，等到前一条指令完成了写回再继续。这里把寄存器文件从时钟上升沿写入改为时钟下降沿写入，这样一来WB阶段就会提前半个周期执行，后来的指令也可以少停一个周期。另一个关键在于检测数据相关的存在，通过指令类型判断ID段两个寄存器使用情况，再把ID段的源寄存器编号和后面EX、MEM段的目的寄存器进行比较即可。同时，检测必须排除0号寄存器。

### 硬件设计

下一个PC的逻辑是根据在EX段增加的控制信号来计算，随后将算得的PC送到IF段。 需要跳转时，Bubble信号置1，同时PC寄存器将从EX段提供的PC里选择，并将气泡插入相应的部件。通过结合EX与MEM 段的目标寄存器编号、RegWrite信号以及ID段的指令，设计出数据冲突检测器，若发生数据冲突，若产生冲突则发出Stop信号，同时暂停相应部件并插入气泡。

## EX段分支的重定向流水线设计

### 总体设计

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，而是等到指令实际使用这些寄存器操作数时再考虑正确性问题。

为了解决气泡流水线插入气泡过多导致的流水线性能问题，重定向流水线停止ID段的数据冲突判断，并将这一判断置于EX段，若发生数据冲突则需要直接从EX/MEM或MEM/WB段的流水接口中取出正确的数据输入到Alu中进行计算。需要重定向的数据有四种可能性：EX/MEM中的AluResult, MEM/WB中的AluResult以及MemResult (写回RegFile中的数据)。

### 硬件设计

将EX、MEM等段的数据进行重定向选择信号，选择合适的如输入值，带入在ID段并通过流水线部件向后传递。但存在特殊情况：相邻的两条指令数据相关并且前一条指令是访存指令 ，即Load-Use，该情况通常通过在两条指令之间插入气泡来解决。分支成功跳转仍使采用Bubble 信号来进行气泡的插入，Load-Use信号来替代气泡流水线中Stall信号的控制作用。

## 动态分支预测机制

为了解决分支跳转问题引入动态分支预测机制，当IF段为分支指令时分支预测部件给出是否跳转的预测，若预测跳转则同时给出需要跳转的地址并开始预取指令。当分支指令执行到执行阶段时，根据实际是否跳转与预测结果是否一致进行调整。如果预测与实际一致，流水线继续往下执行即可。如果预测与实际不一致，此时就需要清空IF/ID和ID/EX两个流水接口部件，然后将EX段分支指令实际的下一步指令地址送入PC。采用的动态分支预测策略是分支预测缓冲器（BTB），用于存放分支指令的分支跳转历史统计信息。

BTB表会放在IF段，利用PC的值作为关键字进行全相联比较，此过程应与指令存储 器取指令操作并发，并不需要取出指令即可进行分支预测。BTB表命中表示当前指令是分支指令，可以根据BTB表中当前指令的历史预测位决定下条指令的地址是PC+4还是BTB表中的分支目标地址，注意这个分支目标地址不能在 IF 段取指令后计算，而是由BTB 表中的BTB表项提供的。如果BTB表缺失，表明当前指令可能不是分支指令或者是不经常 使用的分支指令，则按照 PC+4 取下条指令。BTB逻辑实现如图 2.4所示。

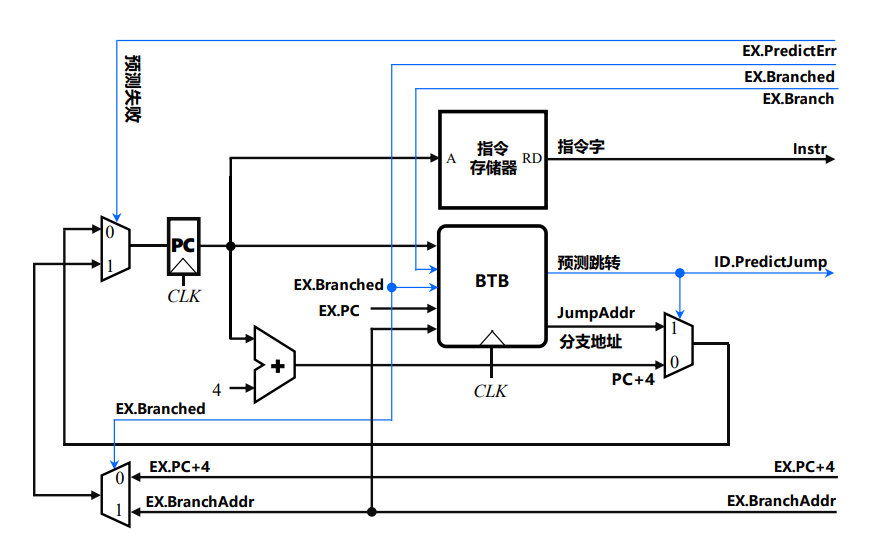


图 2.4动态分支预测BTB逻辑实现

# 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）

程序计数器用于寄存指令地址，使用一个32位寄存器，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。halt为停机信号，将此控制信号通过非门取反后接入使能端，若halt值为1，表示该程序周期数已达最大值，执行完毕，则PC使能端置为0。如图 3.1所示。

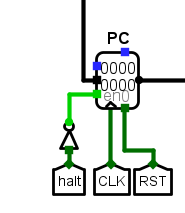


图 3.1程序计数器（PC）

1. 指令存储器（IM）

指令存储器用于存储一个程序的汇编代码，即各地址下的指令内容。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，运行某一程序前只需将该程序的汇编代码存入此存储器中。如图 3.2所示。

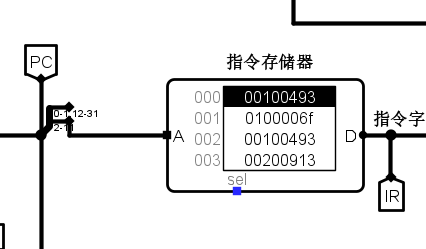


图 3.2指令存储器（IM）

1. 运算器（ALU）

运算器用于完成指令执行过程中的运算环节。该运算器由封装而得，A,B为将要参与运算的32位数，B的取值为寄存器还是立即数由指令性质决定，S指定该运算器的运算类型，B输入端前的二路选择器的选择端和S的输入端均来自于控制器的输出，是指令译码的结果。输出的值对指令的后续执行由影响。如图 3.3所示。

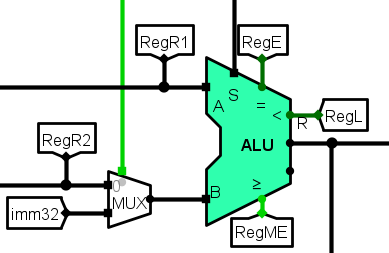


图 .运算器（ALU）

1. 寄存器堆（RF）

寄存器堆用于实现对CPU中寄存器的读取和写入。该原件为实验已给的封装组件，在CS3410 Components.jar中的Register File给出，存储内容为R0-R31共32个32位寄存器，即我们设计的CPU中的所有寄存器，R1#,R2#,W#接5位的寄存器编号，Din接要写入寄存器的数值，WE为写使能信号，CLK为时钟信号。两个输出R1,R2分别为R1#,R2#的编号对应寄存器的值。如图 3.4 所示。

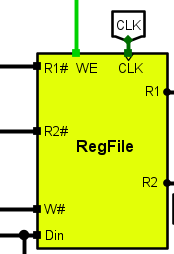


图 .寄存器堆（RF）

1. 数据存储器（DM）

数据存储器用于存储CPU的数据，可实现对指定地址下数据的读取和写入。实现方法与IM类似，这里使用一个随机存储器RAM实现，设置该随机存储器的地址位宽为10位，数据位宽为32位，使用分线器只取32位指令地址的2-11位作为数据存储器的输入地址。MemWrite决定是否将数据写入，由控制器对指令译码得到。如图 3.5所示。

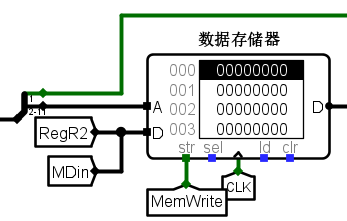


图 .数据存储器（DM）

### 数据通路的实现

在完成指令系统数据通路表 2.3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，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单周期CPU 数据通路如图 3.6所示。

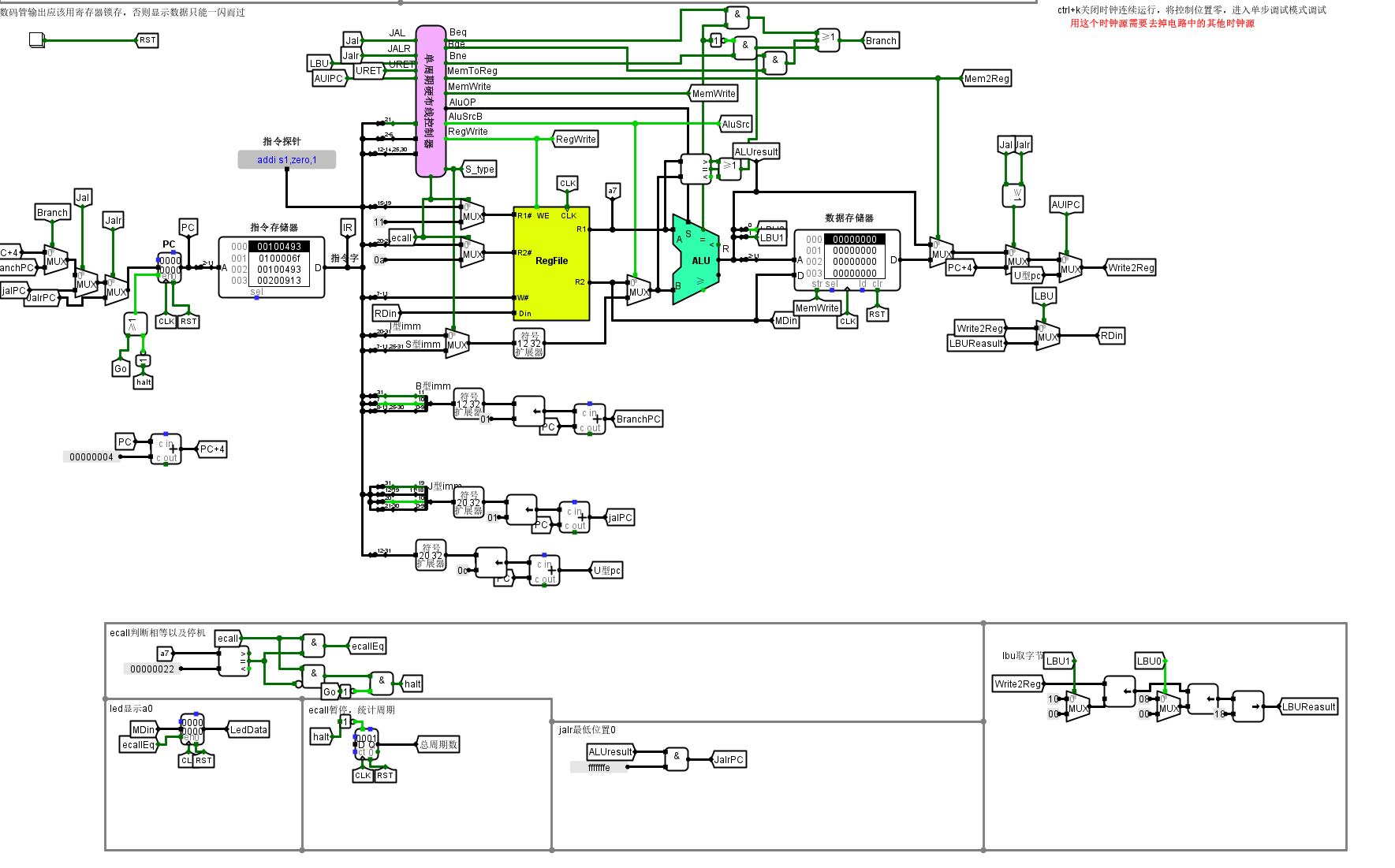


图 .单周期CPU数据通路

### 控制器的实现

根据2.1.3节相关内容，在实验资料中给出的Excel表上填写数据通路上各指令对应的各选择信号的值，自动生成各个输出端的表达式。运算器控制器和信号生成控制器对指令操作码OP\_CODE和Funct进行译码，得到的总体结构即单周期硬布线控制器如图 3.7所示。

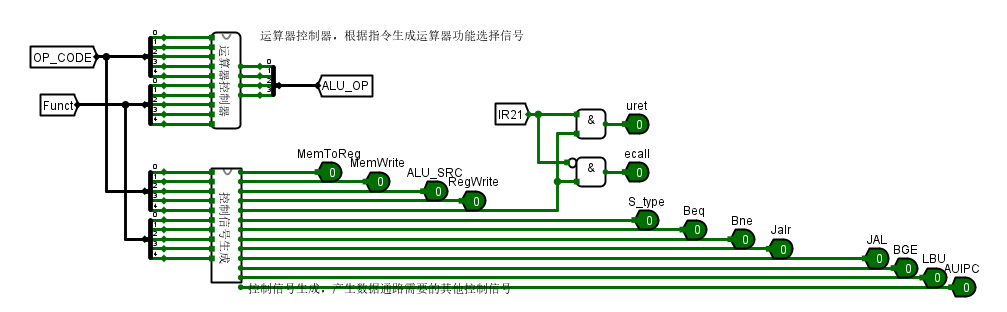


图 .单周期硬布线控制器

### 单条指令数据通路的完整构建

以上为总体框架的构建，下面选取LBU指令的构建过程进行具体描述，其他指令不再赘叙。

由于LHU输入I型指令，I型运算类指令另外一个运算操作数来自12位立即数12位立即数经32位符号扩展器扩展后送入 ALU，二者进行运算得到最终的结果，运算结果写入目的寄存器 rd 中，同样将AluOp设置为不同的值，就可以进行不同的运算，数据通路如图 3.8所示。

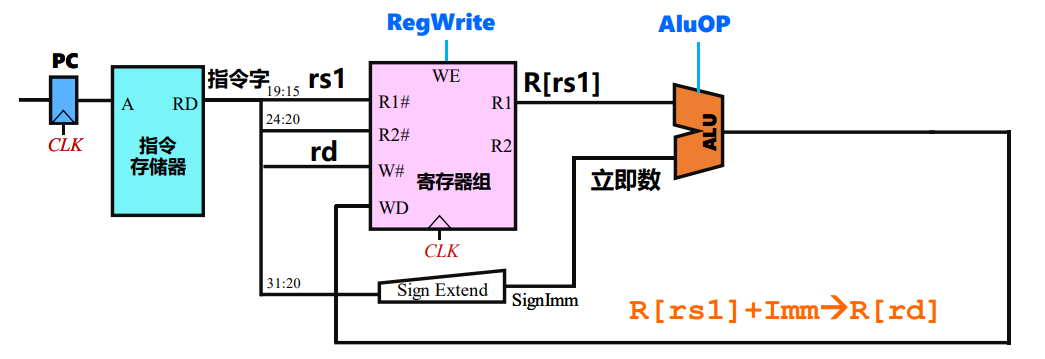


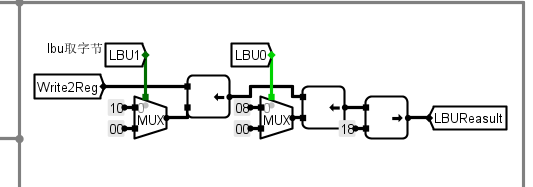
图 3.8 I型运算指令数据通路

LBU，无符号字节加载指令，用法：LBU rd，offset(rs1)，作用：x[rd] = M[x[rs1] + sext(offset)][7:0]。由于要实现寄存器写入，ALU运算时B值为立即数，内存读取，rs1和offset的相加这几个功能，故MemtoReg,ALU\_Src,RegWrite值均置为1，ALU\_OP置为5，同时LBU标志信号也置1。

表 3.1 LHU指令控制信号

| 指令 | ALU\_OP | MemtoReg | MemWrite | ALU\_Src | RegWrite | ecall | S\_Type | BEQ | BNE | Jal | Jalr | LBU | BGE |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lBu | 5 | 1 |  | 1 | 1 |  |  |  |  |  |  | 1 |  |

LBU标志信号的作用在于取出有效地址的数据后进行0扩展并将值写入rd寄存器，故在这里采用一个多路选择器来选择写入寄存器的值。



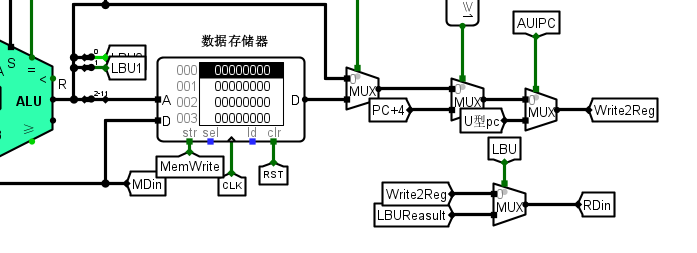


图 3.9 LHU通路部分功能实现

## 中断处理机制实现

### 单级中断机制

在单周期CPU的基础上，稍加改动，即在图 3.10程序计数器PC前后增加对中断响应和结束的判断，用两个2路选择器保证程序的正确执行顺序。NextPC用于判断是否有中断返回，若URET为0，则输出控制信号控制的下一条指令或跳转的地址，否则输出中断的返回地址。当前指令的地址也有两个来源，一个是PC寄存器给出的地址，一个是响应中断后中断处理程序的入口地址。

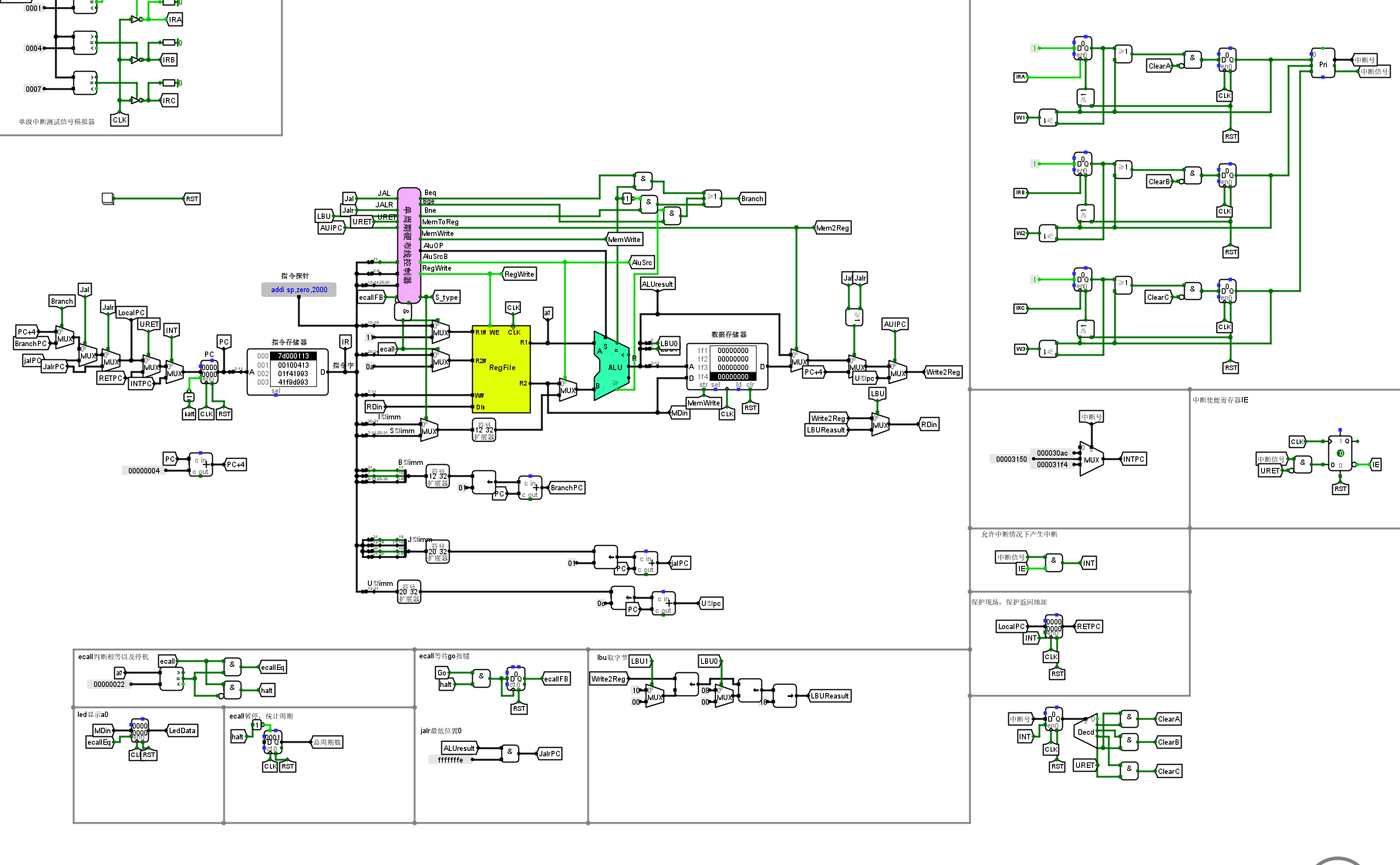


图 3.10单周期中断处理机制数据通路

新增的中断处理机制如图 3.11所示，用一个优先编码器完成对三个中断信号的逻辑选择。优先编码器和中断使能信号IE产生INT中断信号。优先编码器进行中断处理程序入口的选择。返回地址寄存器保存Uret指令后下一条指令的地址。

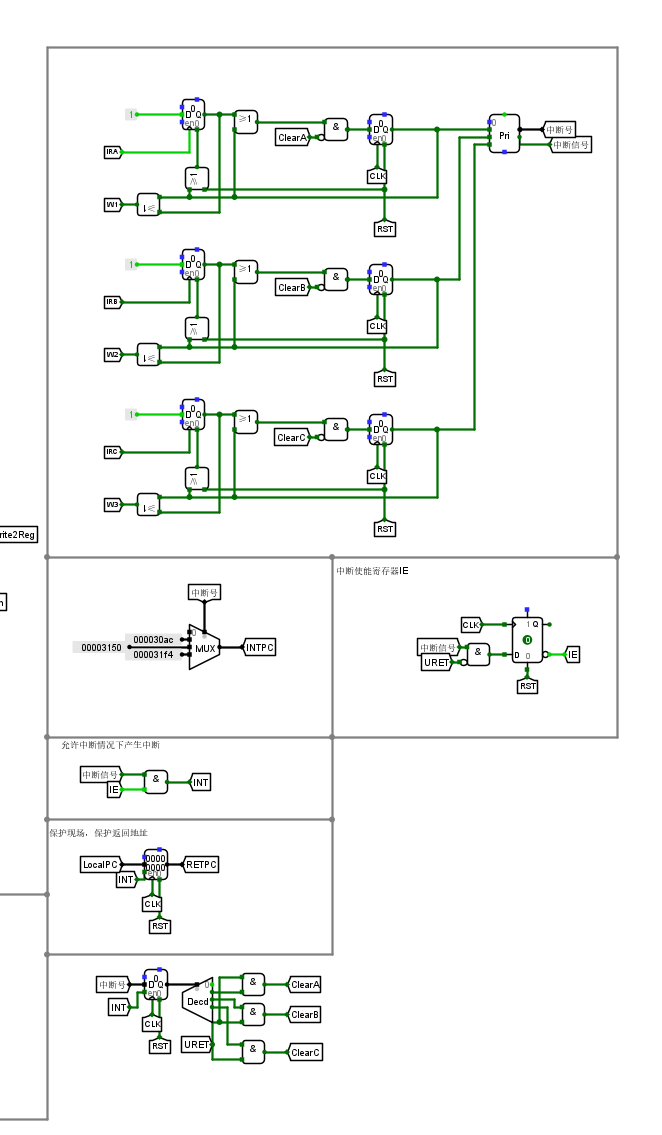


图 3.11单级中断的处理

另外，由于uret和ecall两条指令需要对第21位进行区分，故硬布线控制器做也要做出相应更改，如图 3.12所示。

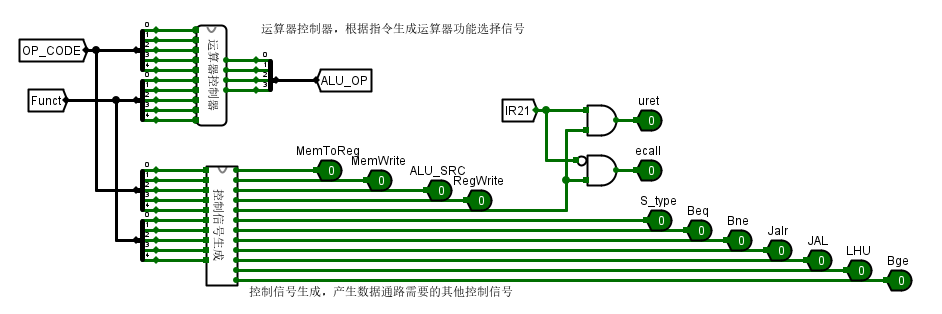


图 3.12单级中断对硬布线控制器的改进

### 多级中断机制

多级中断相对单级中断的实现复杂很多。在数据通路部分，两种中断改进方法相同。主要差别在于处理机制。

如图 3.13所示，将主程序视为中断信号为0的程序，经过优先编码器后选择的结果为IntNum，其作用在于后续判断当前最高级别中断是否在运行，产生的INTNumRunning信号决定INT。在2.1.1提到过对硬件栈和栈顶的实现，图 3.14右上部分即为硬件栈，左侧的寄存器为栈顶计数器。在ReadyInt或uret的时候计数，实现栈顶的加减功能。

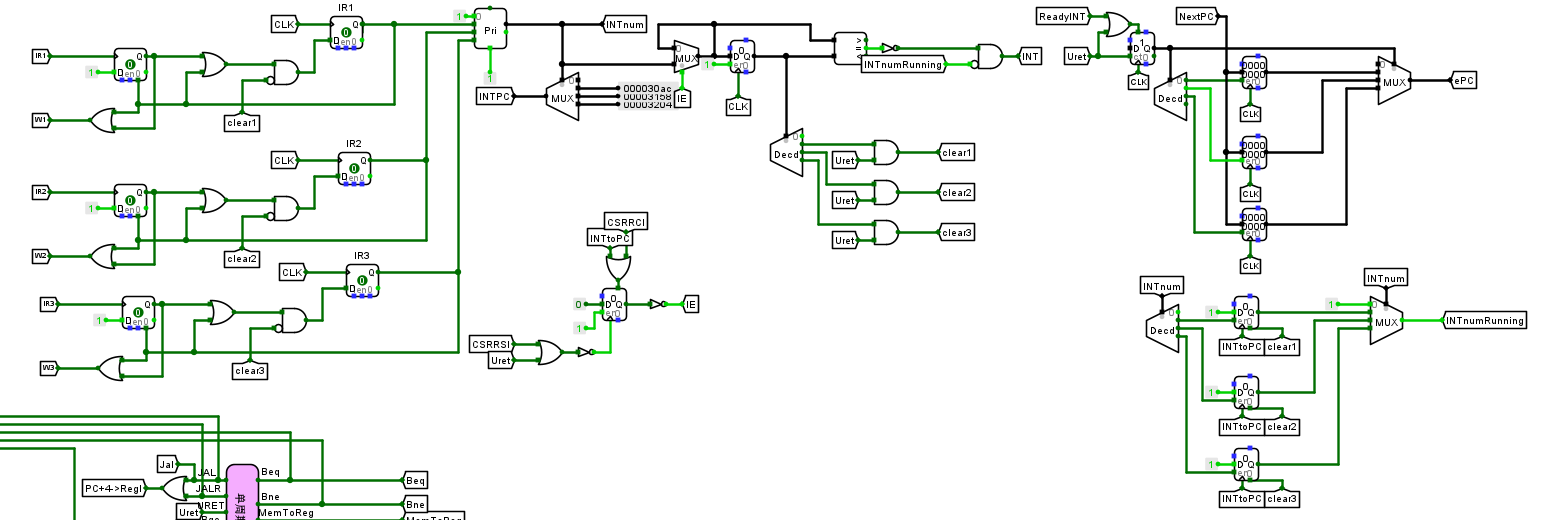


图 3.13多级中断的处理

由于多级中断增加了CSRRCI和CSRRSI这两天开关中断的指令，硬布线控制器也要做出相应的修改，如图 3.14。

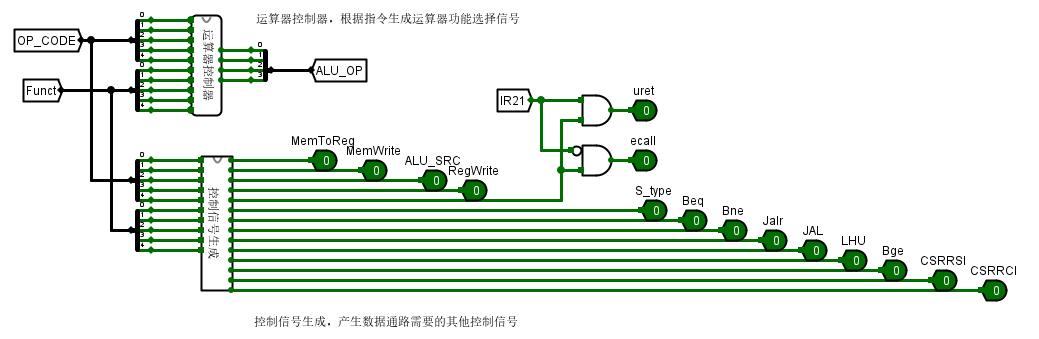


图 3.14多级中断对硬布线控制器的改进

### 流水单级中断机制

流水单级中断是单级中断和重定向流水线的组合。下面介绍几个重要改动部分。

1. 中断响应时，没进入执行阶段IF和ID段的指令被清空，所以需要清空IF/ID以及ID/EX两个流水接口部件。



图 3.15中断响应时流水接口部件清空

1. 中断响应时，EX段的指令被打断，所以正在执行的指令的下一个指令，也就是处于ID段的指令的地址将写入EPC之中作为断点。如图 3.16所示，EPC接入的地址是ID段指令的地址。

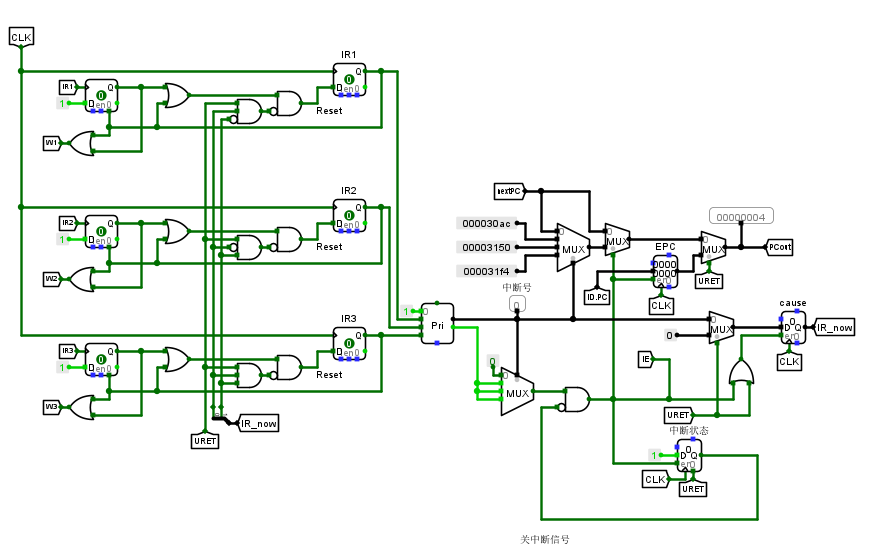


图 3.16重定向流水线中断处理机制

1. 在执行uret指令时，由于返回指令是在ID阶段产生uret信号的，所以我们需要将IF段的指令清空防止其进入ID段。

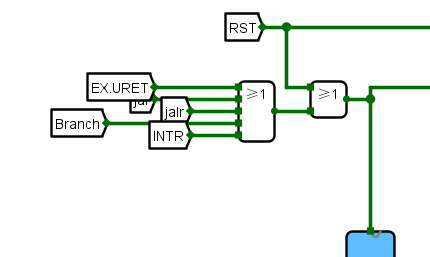


图 3.17 uret信号产生时流水接口部件清空

## 理想流水线实现

### 流水接口部件实现

流水寄存器要对每一个要传递的信号使用一个上升沿的寄存器进行锁存，采用统一公共时钟CLK进行同步，同时为实现同步清零，采用二路选择器，在CLR为1时实现清0。

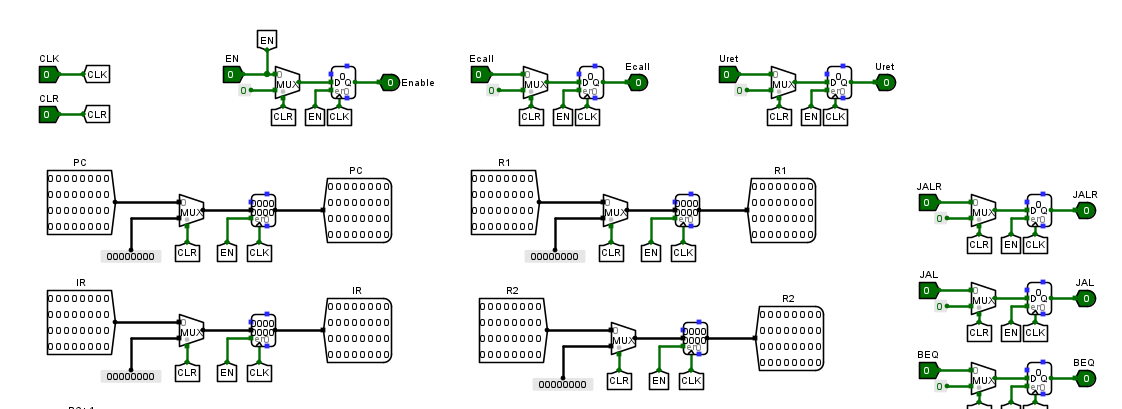


图 3.18流水接口部件内部实现部分截图

### 理想流水线实现

取指、译码和执行部分如图 3.19所示，访存、写回部分如图 3.20所示。

在取指部分的下址计算逻辑，由于理想流水线不需要跳转，故可简化为 PC+4。

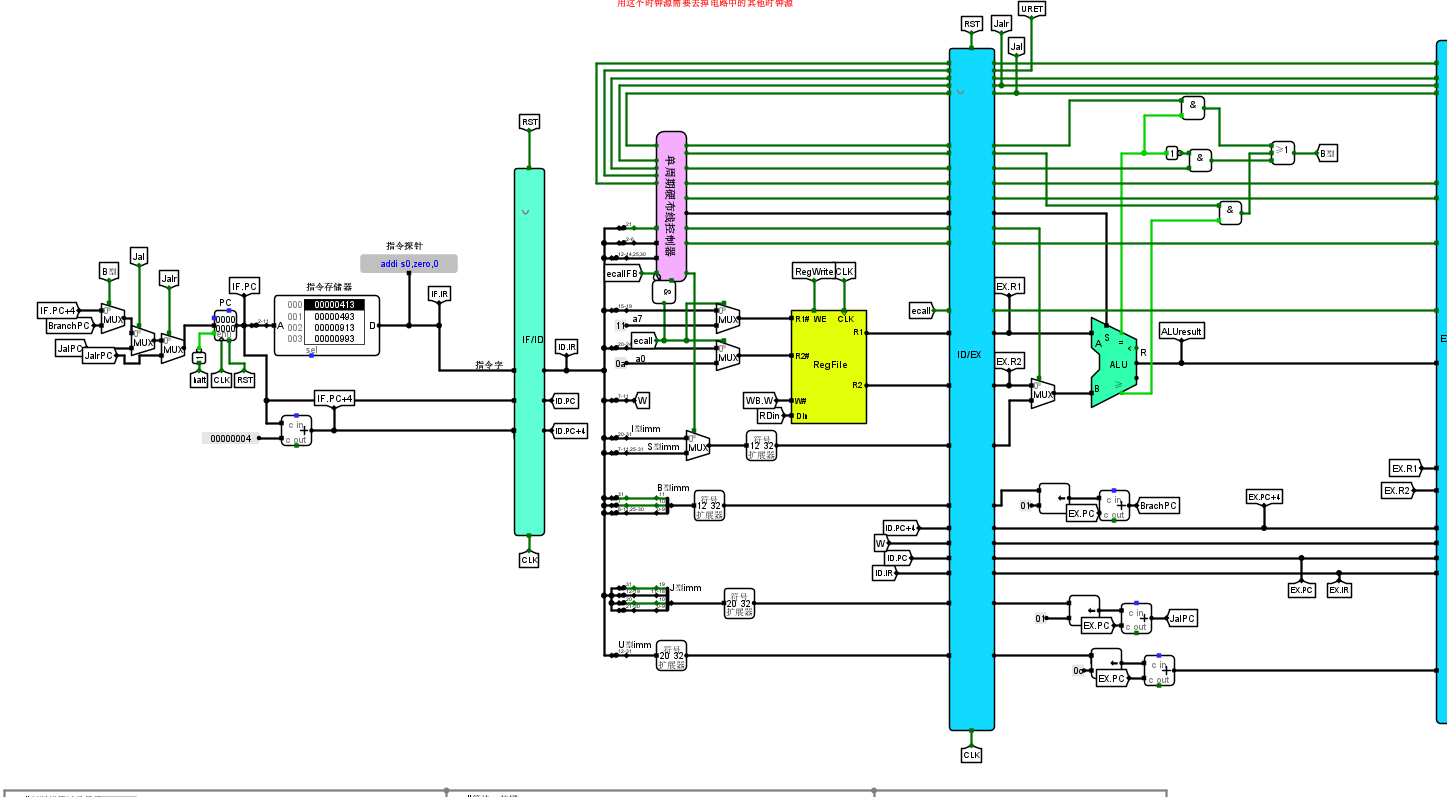


图 3.19理想流水线取指译码执行

寄存器堆的控制信号是在写回部分产生的，为使连线简洁，用隧道来表示。ECALL 的实现是在写回部分，故要一直传到至写回。



图 3.20理想流水线访存写回

## EX段分支的气泡流水线实现

### 流水部件修改

在理想流水线的基础上，气泡流水线要支持气泡的插入，即能实现同步清零，用CLR实现，同时还要支持等待，通过使能端En实现。

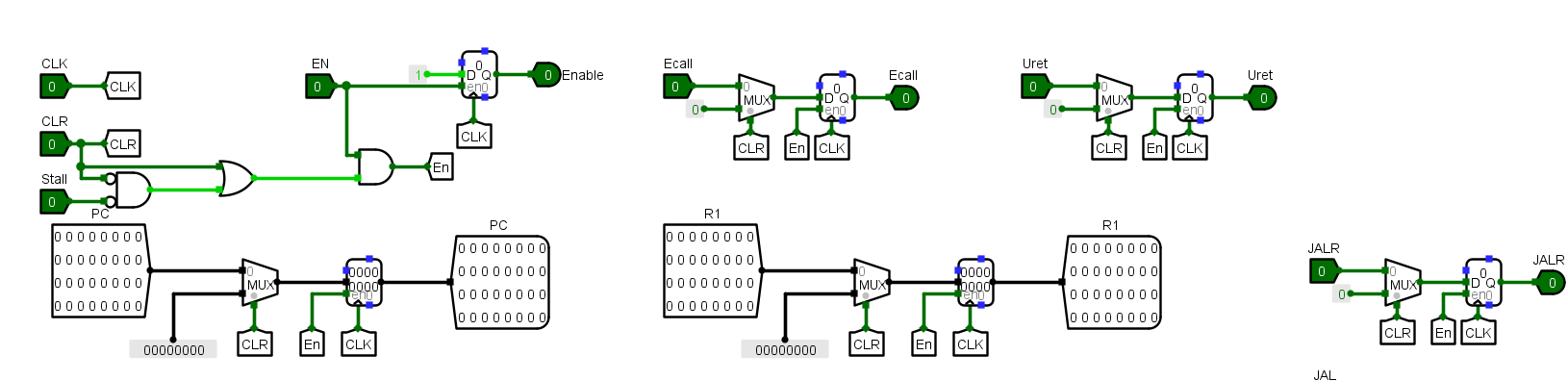


图 3.21气泡流水接口部件内部实现部分截图

### 寄存器冲突判断

为解决对寄存器中的数据冲突，设置一个Stall信号来标志冲突，计算公式如下所示。

DataHazzard = RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#) + RtUsed & (rt≠0) & EX.RegWrite & (rt==EX.WriteReg#) + RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#) + RtUsed & (rt≠0) & MEM.RegWrite & (rt==MEM.WriteReg#)

Stall = DataHazzard # 数据相关时要阻塞暂停 IF、ID段指令的执行

根据公式实现该功能。



图 3.22寄存器使用情况

### 气泡流水线实现

为实现气泡的插入，应满足如下公式。

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

根据公式，绘制气泡流水线图。截取执行访存写回部分如图 3.23所示。在执行阶段，计算取指阶段指令的PC。ID/EX部件在Bubble 或者Stall信号的过程中都表现为清空。

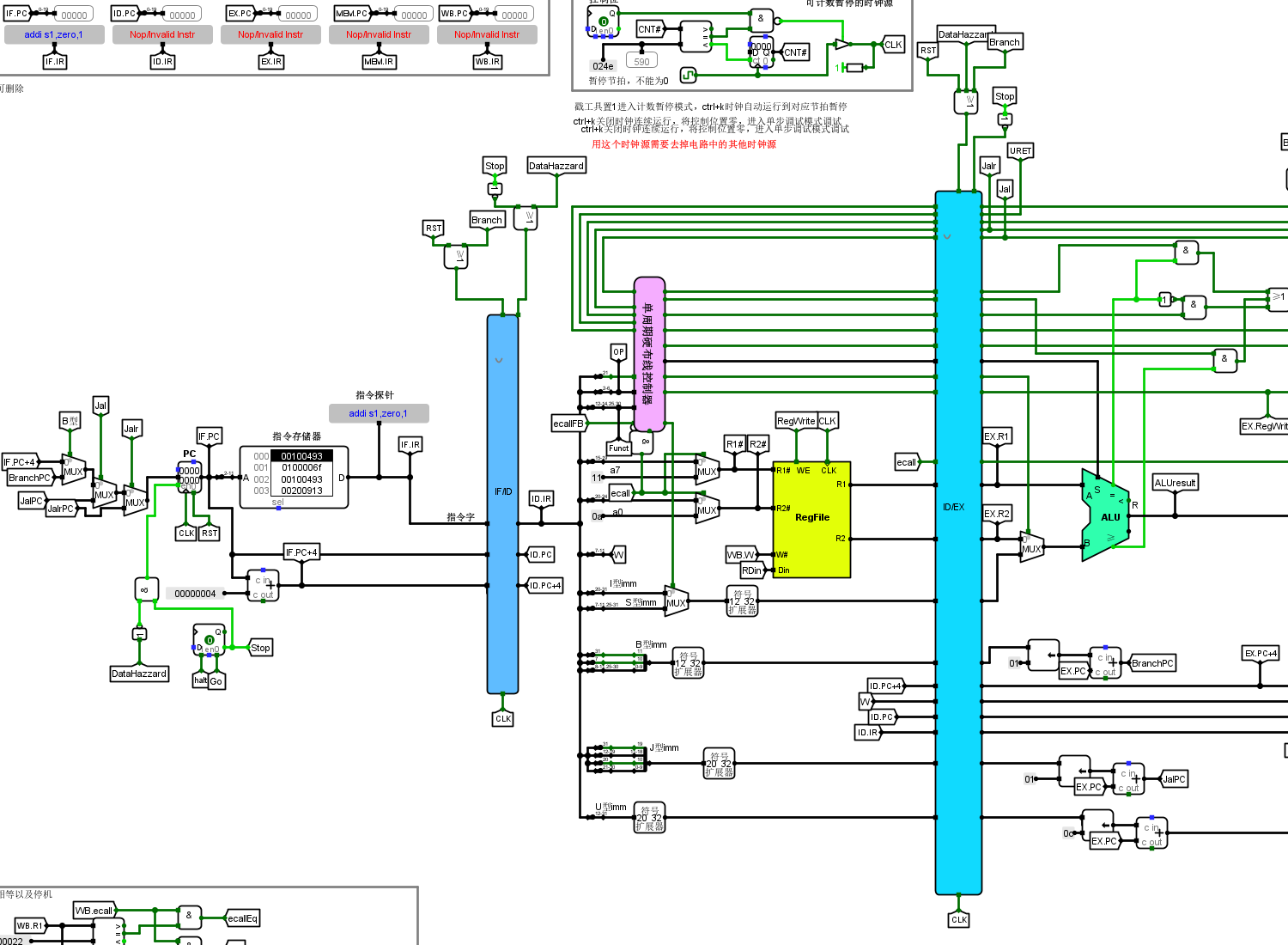


图 3.23气泡流水线实现部分截图

特别注意，RegWrite 信号会受到NOP的影响，因此要修改硬布线控制器，特别处理这个信号。

## EX段分支的重定向流水线实现

### 流水部件修改

重定向流水线先不考虑 ID 段所取的寄存器操作数是否正确，而是等到指令实际使用这些寄存器操作数时再考虑正确性问题，而重定向选择在译码阶段即可计算出来，因此要进行传递。

### 重定向流水线实现

重定向流水线在气泡流水线的基础上加以改进。气泡流水线的分支冲突处理保留。判断数据冲突逻辑要修改。在重定向流水线阶段，仅仅算出有无数据冲突远远不够，还要知道R1、R2和EX、MEM之间哪个阶段存在冲突。同时，如果R1或者 R2 和EX冲突，并且EX阶段是加载型指令，就说明是 Load-Use情况，就要产生对应的信号。

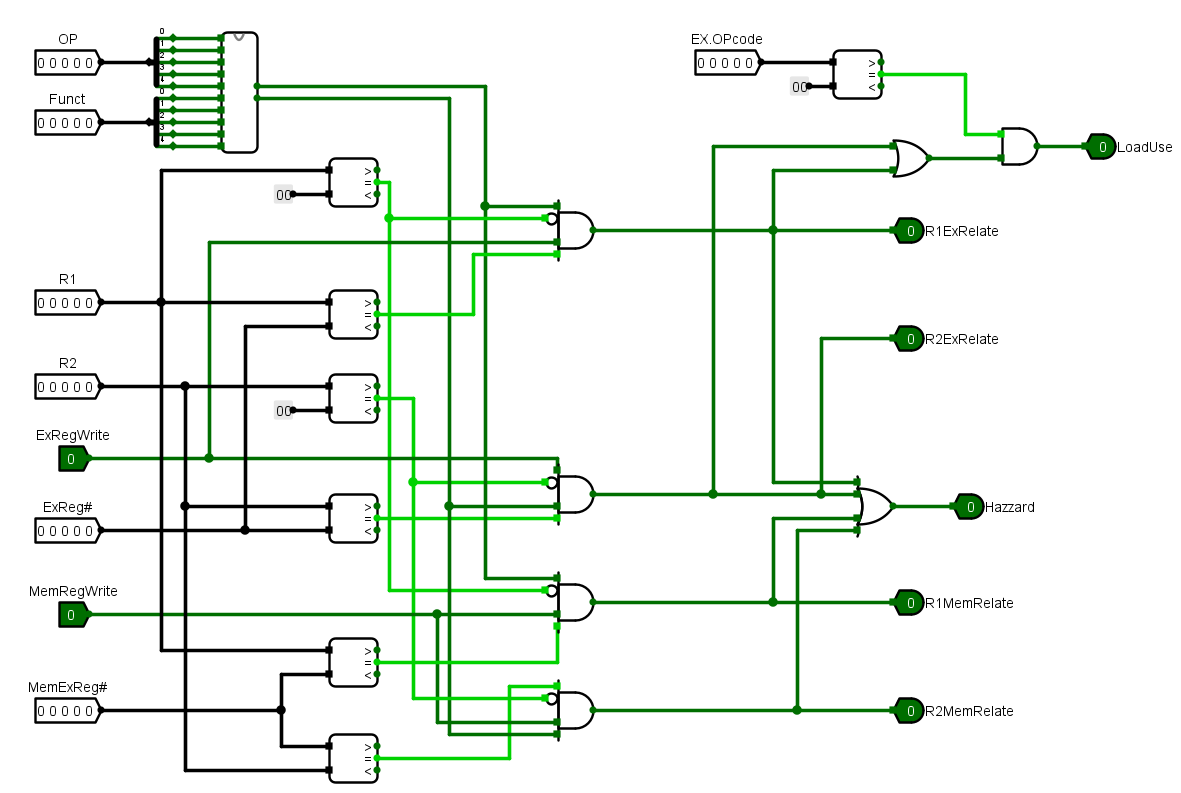


图 3.24重定向流水线寄存器使用情况

重定向流水线的执行访存部分如图 3.25所示，取指阶段传过来的寄存器值被拦 截，在左上角连同访存阶段 ALU 的结果 MemAluResult 和写回阶段的结果RDin 被送入ALU作为运算数。

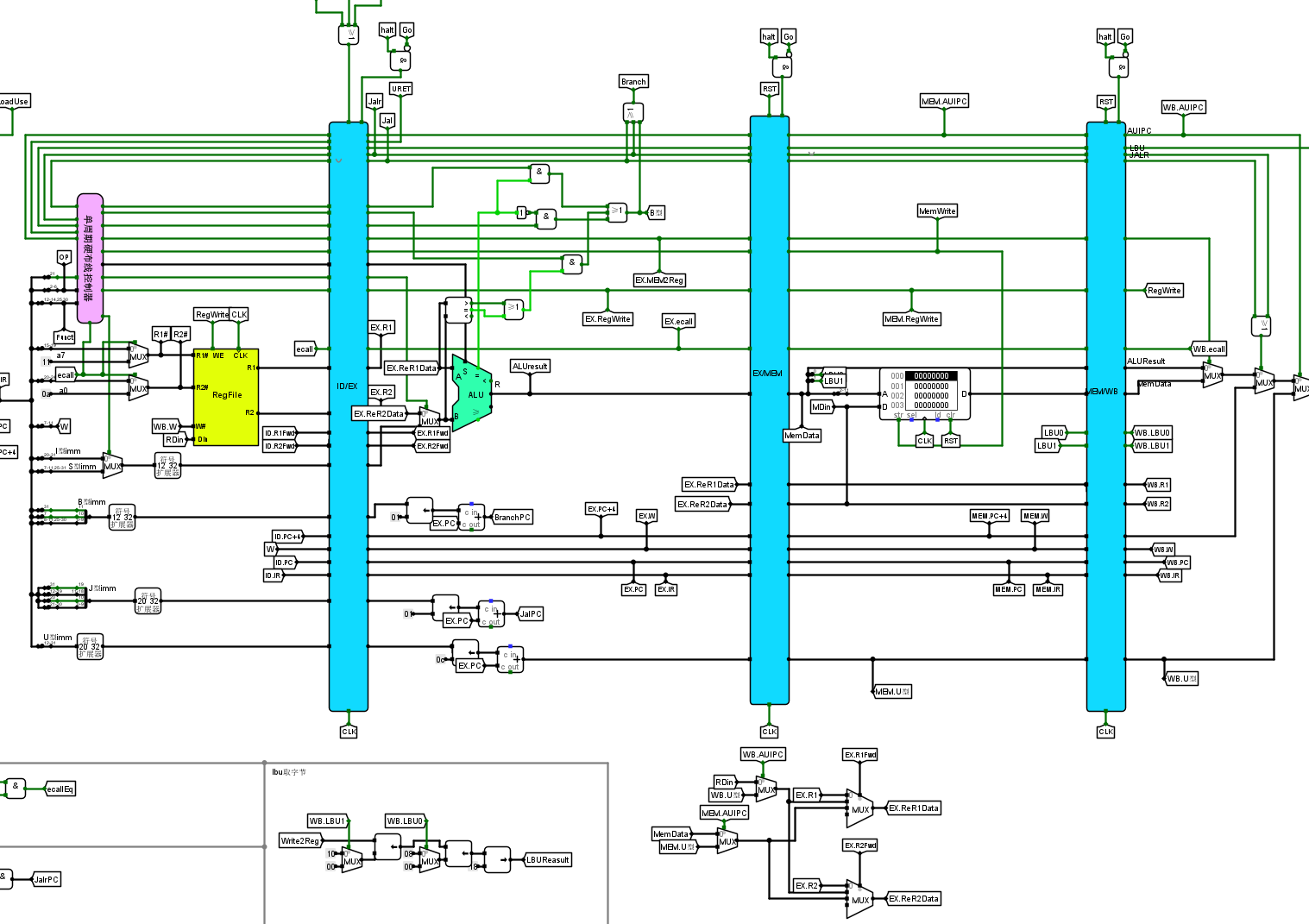


图 3.25重定向流水线实现部分截图

## 动态分支预测机制实现

BTB组件内部是一个全相联的 cache 结构，采用LRU的淘汰策略实现。

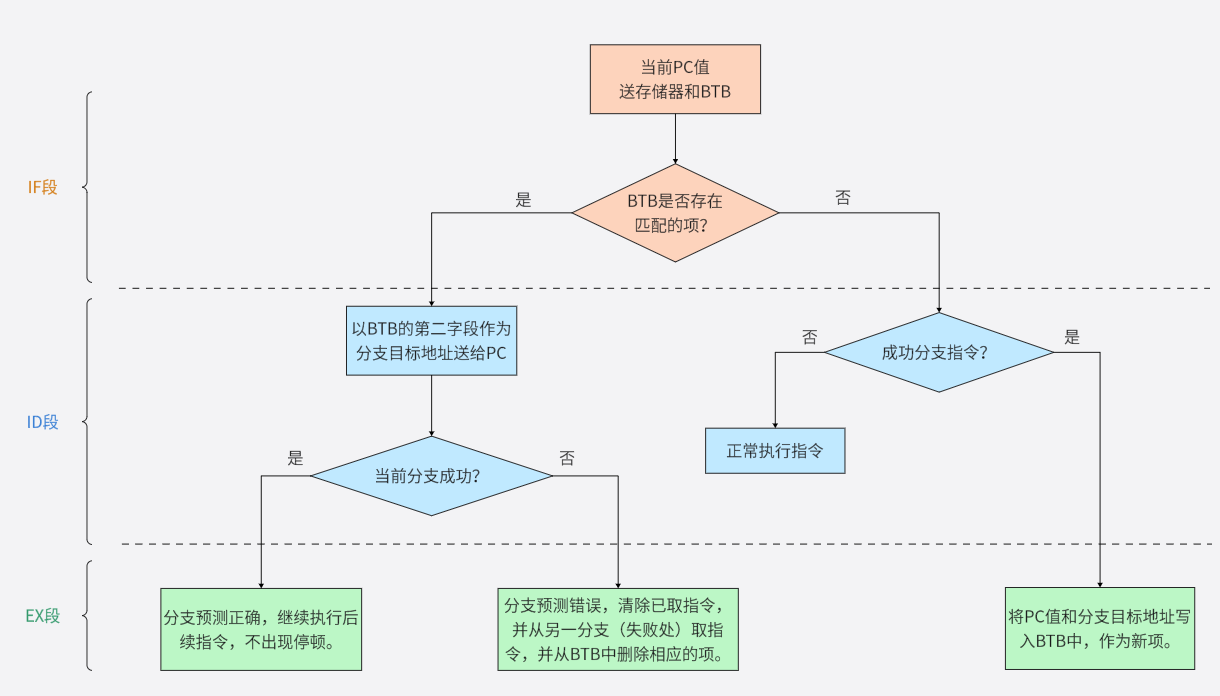


图 3.26 BTB原理

BTB内部结构如图 3.27。

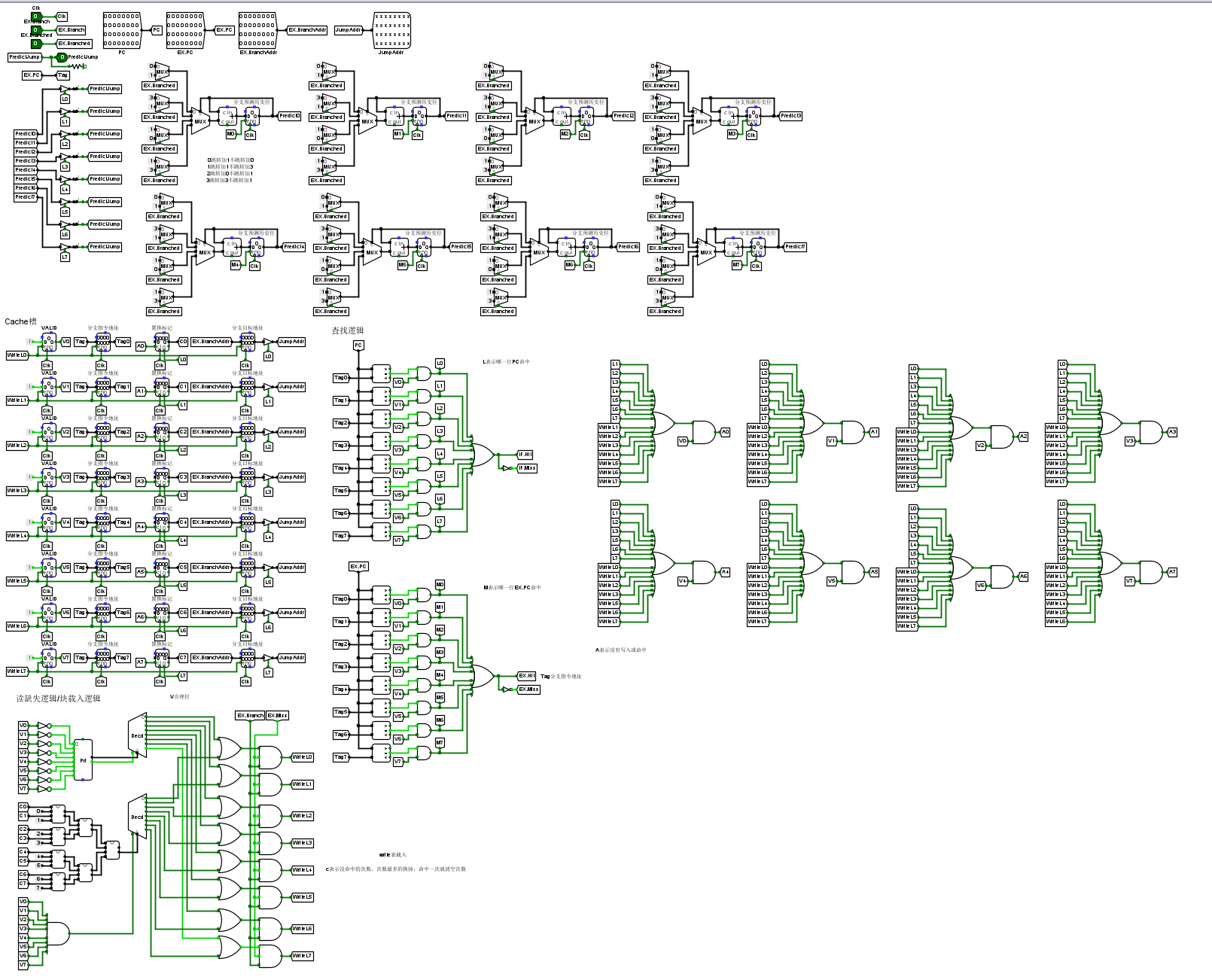


图 3.27 BTB内部结构

在重定向流水线的基础上补充得到动态分支预测，数据通路如图 3.28所示。

当 EX 段从段首的ID/EX流水寄存器接收到从IF段传送来的预测跳转信息 PredictJump时，当实际跳转和预测跳转不符时就会将预测失败信号EX.PredictErr置 1，该信号反馈回IF段控制PC输入端的多路选择器重新取指令，具体取那一个地址的指令由实际跳转情况EX.BranchTaken选择，当 EX.BranchTaken 为时应该选择从EX.BranchAddr 处取指令，否则从EX.PC+4处取指令。

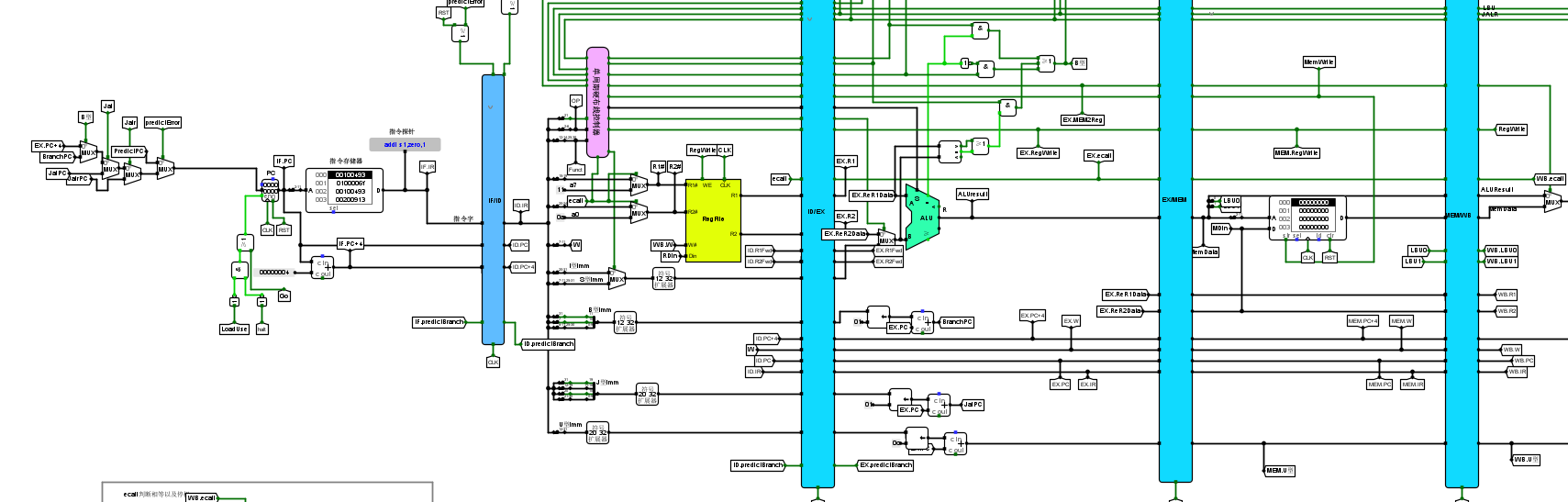


图 3.28 动态分支预测数据通路

# 实验过程与调试

## 测试用例和功能测试

本次课设我完成了单周期CPU、理想流水线、气泡流水线、重定向流水线、单级中断、多级中断、流水中断、分支预测八个模块，已通过Educoder线上测试和线下CCAB检查，这里只展示部分测试。

### 测试用例1

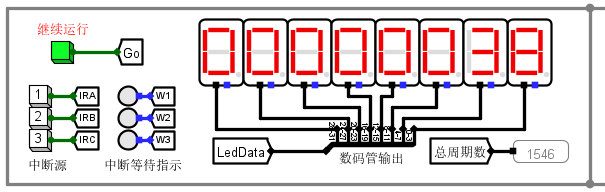


图 4.1单周期CPU运行结果

### 测试用例2

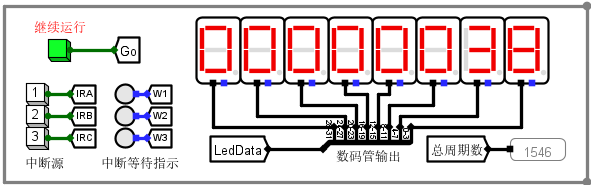


图 4.2理想流水器运行结果

### 测试用例

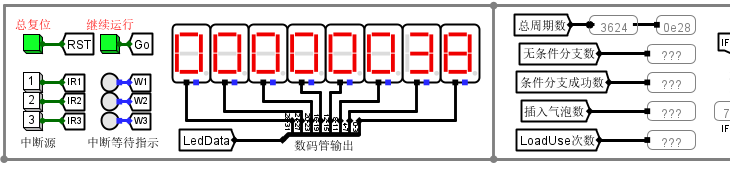


图 4.3气泡流水器运行结果

### 测试用例3

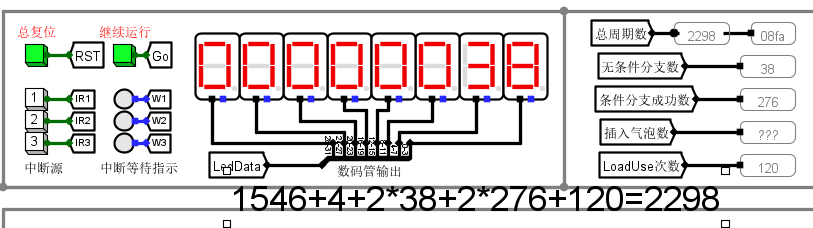


图 4.4重定向流水器运行结果

### 测试用例4

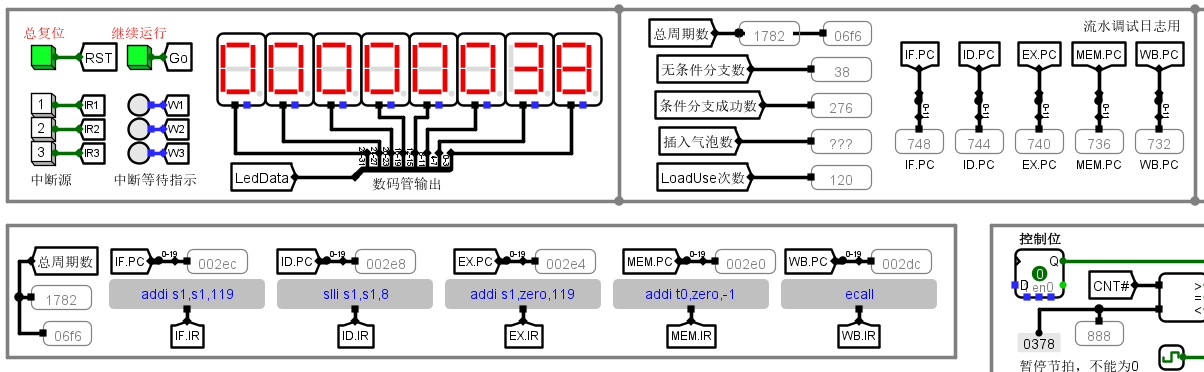


图 4.5重定向加分支预测运行结果

## 性能分析

单周期CPU符合周期数1546的要求

气泡流水线的总周期数、无条件分支数、有条件分支数、插入气泡数（实际上是 数据冲突的次数而不是气泡数）如图 4.3所示，符合总周期数=1546+4+气泡数目+分 支误取深度\*分支数-1=3623的要求。

重定向流水线的相关情况如图 4.4所示，也符合总周期数=1546+4+分支误取深度 \*分支数+load-Use 次数=2297的要求。

重定向流水线加分支预测总周期数为1782，满足要求。重定向减少了数据冲突时插入气泡数目，分支预测减少了分支指令引起的分支延迟损失，因此该流水线运行时间最短。

## 主要故障与调试

### 数据写入错误

故障现象：寄存器堆和数据存储器内数据不正确

原因分析：忽略lw,sw,lhu中对地址的计算，填表时没有设置Alu\_OP值，导致rd2没有与Imm相加，从而没有存入正确数据或没有写入正确寄存器。

解决方案：将这几条指令的Alu\_OP设为5，重新生成硬布线控制器。

### 程序半路终止

故障现象：气泡流水线运行到总周期数为30时程序终止，且寄存器写入出错。

原因分析：仔细阅读教材，发现寄存器堆应设为下降沿触发，ID段和WB段的数据相关可以采用先写后读的方式解决，寄存器堆写入控制采用下跳沿触发，而所有流水寄存器采用上跳沿触发(假设完整时钟周期从1开始，0结束，中间是下跳沿)，这样在一个时钟节拍的中间时刻(下跳沿)可以完成寄存器堆的数据写入，时钟节拍的后半段就可以利用组合逻辑读取寄存器正确的值。解决了ID段和WB段之间的数据相关，就只需要考虑连续三条指令的数据相关性，在流水线实现时只需要考虑ID段与EX、MEM两段之间的数据相关。

解决方案：将RegFile触发方式改为下降沿触发。

### 死循环

故障现象：气泡流水线运行过程中总周期数没有上界，程序不会结束，进入死循环。

原因分析：可能原因有跳转指令译码出错，气泡插入出错，由于气泡流水线是在理想流水线和单周期CPU的基础上完成的，前两个实验已通过，故不可能是在硬布线控制器处出错，经过单步调试，发现一旦出现气泡，程序出错，则说明Bubble信号输出有错，经过排查，发现输出Bubble前的比较器设置错误。

解决方案：将比较器的比较类型从2的补码改为无符号。

### 分支次数统计错误

故障现象:无条件、有条件分支次数不对。

原因分析:若采用CLK上升沿触发，那时各信号可能还未就位。

解决方案:将次数统计的计数器的触发方式改为下降沿触发。

## 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 | 复习关于指令流水线的知识点 |
| 第四天 | 实现理想流水线设计并通过测试 |
| 第五天 | 完成气泡流水线设计并通过测试 |
| 第六天 | 完成重定向流水线设计并通过测试 |
| 第七天 | 复习关于中断相关的知识点并完成单级中断 |
| 第八天 | 实现多级中断并并通过测试 |
| 第九天 | 查看分支预测相关资料，初步开始设计分支预测流水线 |
| 第十天 | 实现分支预测流水线并再次基础上加上单级中断 |

# 团队任务 雷霆战机

## 项目介绍

本组实验主要是利用logism平台设计了一个简易的游戏雷霆战机：游戏主体可分为两部分：玩家和敌机，玩家操控飞机进行上下移动，期间，飞机发射子弹，打中敌机得分，同时要避免被敌机的子弹打中。

## 任务分配

杨道文： 项目构建以及任务分配，玩家飞机的实现，bgm实现，编写代码并调整电路。

黄朝杨： 敌机的实现，编写代码并调整电路。  
宋试翼： 点阵显示，交互界面的实现，ppt制作，伤害机制的实现

## 任务难点

1.对于在敌机不能移动的基础上，能够让敌机进行上下移动。

2.软件中断的实现

3.硬件飞机图形的实现

4.代码发射逻辑的实现

5.玩家移动逻辑的实现以及衍生问题的解决

6.点阵显示界面太过繁琐

7.对持续伤害的处理

8.如何使用尽量少的寄存器，实现飞机移动与子弹发射

9.飞机移动的同时，子弹也要跟随飞机移动发射

## 项目实现

游戏界面如下图

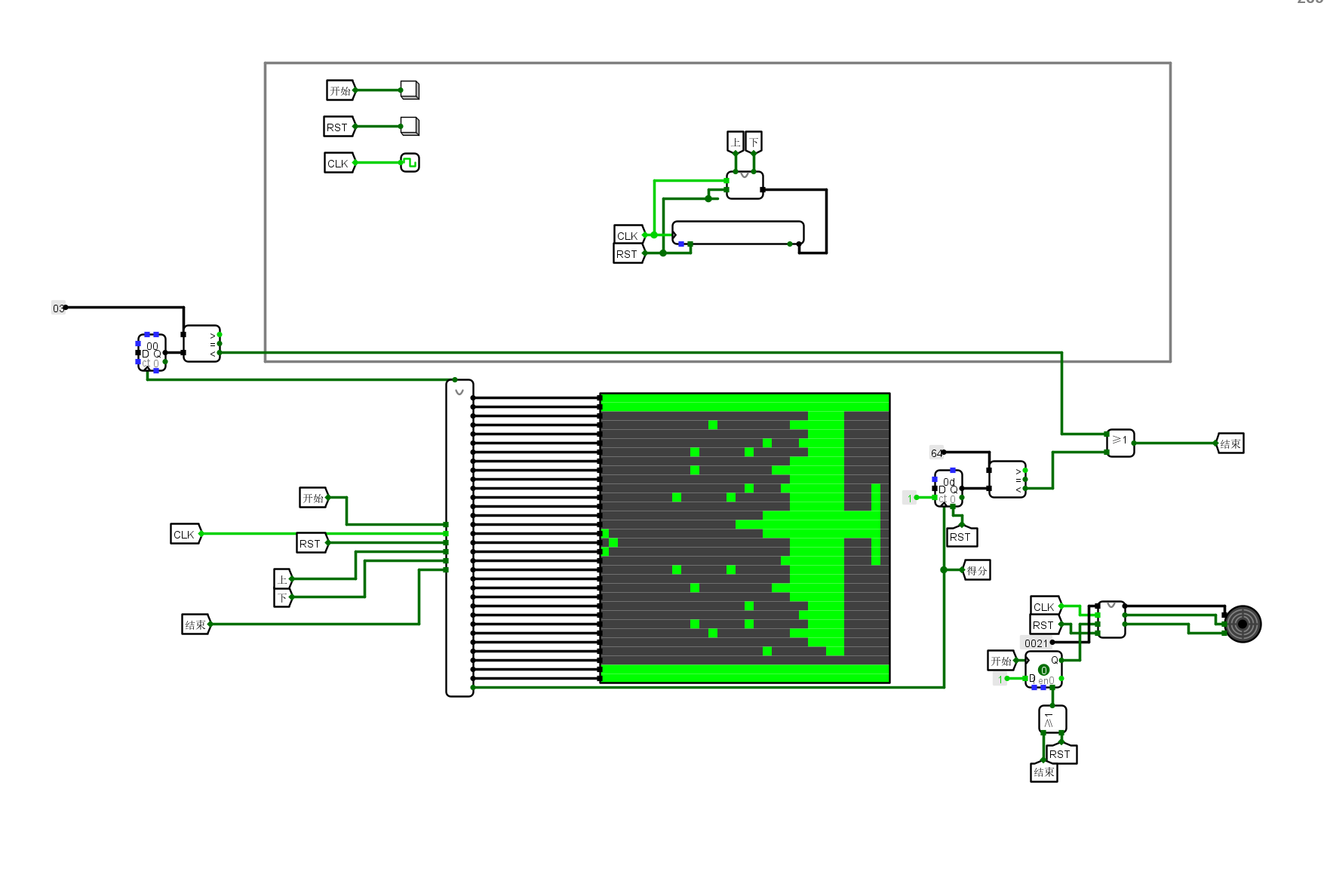


图 5.1 游戏界面

玩家可以通过键盘上的“w”和“s”键进行上下移动，敌方飞机会不断发射子弹攻击我方飞机。我方要不断躲避子弹，同时发射子弹攻击对方。知道我方血量为0，或敌方血量为0。

我的主要任务是负责我方飞机的移动逻辑以及发射子弹逻辑，主要通过软件实现，用的cpu是单周期cpu，因为我发现重定向流水线cpu有bug，同时运行速度比单周期cpu慢，按理说重定向是快的，但可能因为电脑算力固定，流水线消耗的资源更多，才导致单周期速度比流水线快。负责模块部分如下图：

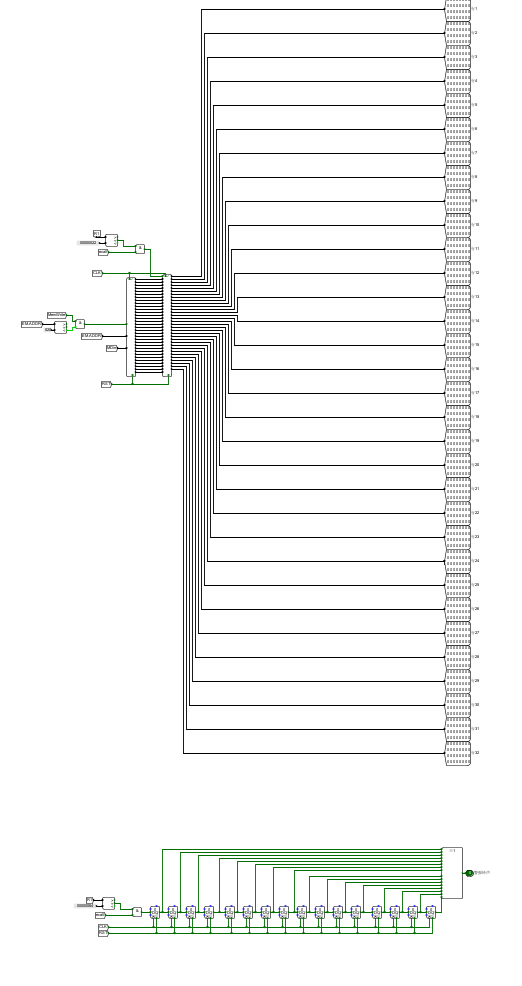
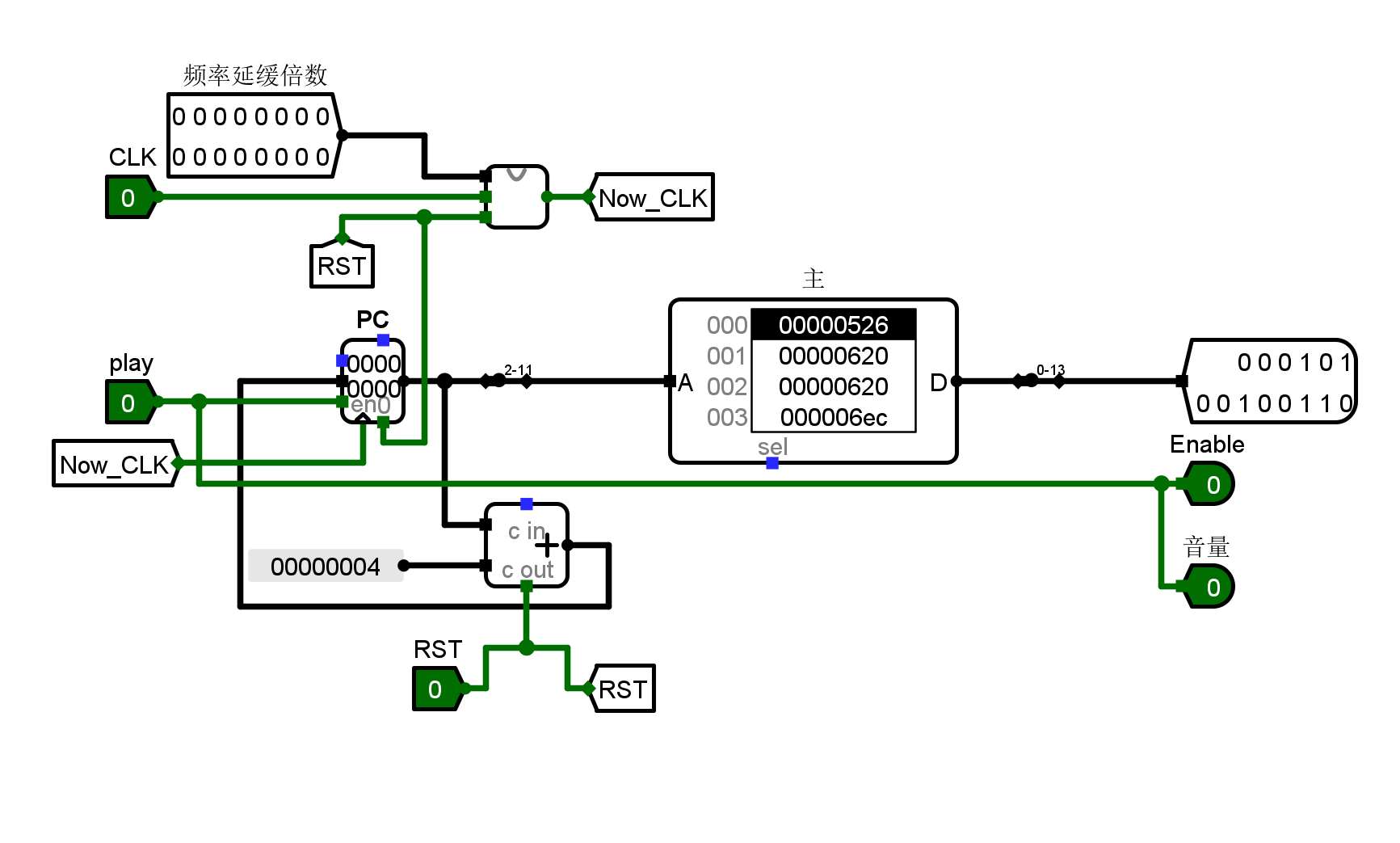


图 5.2 负责模块

主要难点其实是对riscv语言不熟悉，还是靠同学和老师的帮助才逐渐了解了riscv语言的语法，这个过程花了2-3天，之后就步入正轨。我的任务主要是通过软件实现，这是个困难的挑战，要根据自己的cpu将各种指令和寄存器运用恰当。

以及值得一提的是bgm，当时还不知道bgm怎么产生，上网查了各种资料，明白了用的是一个蜂鸣器的设备，将特定频率转成声音。之后我精挑细选，使用了蜜雪冰城主题曲。首先获得乐谱，之后用python把他们转化成16进制频率。最后设计bgm组件，如图。



事实上，我感觉效果还是不错的。

# 设计总结与心得

## 课设总结

本次课程设计我做了如下几点工作：

1. 用Logisim实现了单周期 CPU 的数据通路，完成了单周期 CPU 24条基础指令和 CCAB 4条差异性指令的电路；
2. 在可接收、存储、传递信号的简单流水线部件的基础上，设计了不考虑逻辑跳转的理想流水线；
3. 解决理想流水线不考虑逻辑跳转的问题，流水线中插入气泡，克服结构冲突和数据冲突，完成气泡流水线；
4. 解决气泡流水线插入气泡过多导致的性能问题，修改气泡流水线逻辑，实现重定向流水线；
5. 在单周期CPU上完成支持3个中断源的单级中断，设计了处理中断的基本逻辑，进而在重定向流水线上实现单级中断；
6. 在单级中断基础上实现了保存返回地址的硬件栈，从而完成多级中断实验；
7. 自主学习分支预测方法，在重定向流水线上实现成功。

## 课设心得

耗时两周时间，终于完成课设Logisim部分实验。本次课程设计难度较大，挑战性高，后续上板子任务虽然没有继续完成，留有遗憾，但还是学到了很多知识，能力得到了锻炼。

我认为此次课设最大的困难就是难上手，刚开始写时无从着手。虽然本次课设建立在上学期的理论课基础上，但是和上学期的实验不同，这次是要从0开始完完全全自己搭建一个CPU，而且已经过了一个假期，基础更加不牢固，所以刚开始我非常焦虑，反反复复看任务书，也摸不着头脑，不知道从哪里开始写，只有在我仔细阅读一遍教材相关内容之后，真正理解CPU原理和程序执行过程，我才开始正式进入电路设计环节。

当时上课的时候我对中断就一直没有理解清楚，到现在自己设计的阶段，更是头痛，所以我在这个阶段耗费了大量时间，看了很多资料，而且中断本身难度也大，要考虑的东西非常多，不仅需要考虑硬件部分，还要考虑软件部分，幸亏有了老师在任务书中提出的问题，在一定程度上帮助了我理解这个概念。

我觉得最有成就感的是做理想流水线设计，因为理解最简单，但是连线巨复杂，我花了一整天时间才连完，但连完后看着密密麻麻的电路图，成就感巨大。同时在气泡流水线和重定向流水线，这两个实验中，由于信号很多，稍微少考虑一点就会出错，找故障更是个麻烦。

电路设计过程中遇到过各种各样的bug，上面主要故障与调试列出来的只是冰山一角，有时候甚至会花费一个晚上来解决一个问题，很多故障都是我自己的粗心，还有对概念理解不透彻造成的，但是在bug查出的时候的喜悦可以抵去所有疲惫。

同时团队任务中，将自己的cpu投入实践中，做成一个可以交互的游戏是一件非常有成就感的事情。期间和队员沟通，协调，最后做出一个经典游戏，也算完成儿时玩游戏机的梦想了。只是可惜上板子实在不会，不然做出一个实物，应该更开心。所以希望老师给下届没有上板子经验的同学，讲一讲fpga开发，这样应该会有更多人喜欢上这门课。

同时很感谢老师和同学对我的不吝赐教，由于我自身基础不牢固，很多问题问出来可能会有些愚蠢，如果没有这些帮助，我也完成不好整个实验。同时，最后没有参与团队任务我也感到很遗憾，因为我能认识到我在这方面的欠缺，害怕任务做到半途而废，所以在今后的学习当中，我会更加踏踏实实地打好基础。

连好电路里一根根线后，我发现我更能沉下心来去看资料，去完成各项任务，更能理解踏踏实实学习做事的重要性。

# 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: 杨道文** |