***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： | CS2008 |
| 学 号： | U2020215533 |
| 姓 名： | 徐瑞达 |
| 电 话： | 17837353795 |
| 邮 件： | [2014027378@qq.com](mailto:2014027378@qq.com) |

目 录

[1 课程设计概述 3](#_Toc116119676)

[1.1 课设目的 3](#_Toc116119677)

[1.2 设计任务 3](#_Toc116119678)

[1.3 设计要求 3](#_Toc116119679)

[1.4 技术指标 4](#_Toc116119680)

[2 总体方案设计 6](#_Toc116119681)

[2.1 单周期CPU设计 6](#_Toc116119682)

[2.2 中断机制设计 12](#_Toc116119683)

[2.3 理想流水线CPU设计 13](#_Toc116119684)

[2.4 气泡流水线CPU设计 14](#_Toc116119685)

[2.5 重定向流水线CPU设计 14](#_Toc116119686)

[2.6 动态分支预测机制设计 15](#_Toc116119687)

[3 详细设计与实现 16](#_Toc116119688)

[3.1 单周期CPU 实现 16](#_Toc116119689)

[3.2 中断机制实现 19](#_Toc116119690)

[3.3 理想流水线CPU实现 21](#_Toc116119691)

[3.4 气泡流水线CPU实现 23](#_Toc116119692)

[3.5 重定向流水线CPU实现 23](#_Toc116119693)

[3.6 动态分支预测机制实现 25](#_Toc116119694)

[4 实验过程与调试 27](#_Toc116119695)

[4.1 测试用例和功能测试 27](#_Toc116119696)

[4.2 性能分析 29](#_Toc116119697)

[4.3 主要故障与调试 29](#_Toc116119698)

[4.4 实验进度 31](#_Toc116119699)

[5 团队任务 32](#_Toc116119700)

[5.1 选题介绍 32](#_Toc116119701)

[5.2 任务分工 32](#_Toc116119702)

[5.3 问题与解决方案 32](#_Toc116119703)

[5.4 版本提交历史 35](#_Toc116119704)

[6 设计总结与心得 36](#_Toc116119705)

[6.1 课设总结 36](#_Toc116119706)

[6.2 课设心得 36](#_Toc116119707)

[参考文献 38](#_Toc116119708)

# 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Risc-V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. 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RISC-V32指令集，最终功能以RARS模拟器为准。 |
| 2 | ADDI | 立即数加 |
| 3 | AND | 与 |
| 4 | ANDI | 立即数与 |
| 5 | SLLI | 逻辑左移 |
| 6 | SRAI | 算数右移 |
| 7 | SRLI | 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 | SLTI | 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 ($a7==34) LED 输出$a0的值  else 暂停等待Go按键继续运行 |
| 22 | CSRRSI | 访问CSR 寄存器 | 中断相关，可简化为开中断 |
| 23 | CSRRCI | 访问CSR 寄存器 | 中断相关，可简化为关中断 |
| 24 | URET | 中断返回 | 清中断，mEPC送PC，开中断 |
| 25 | AUIPC | 立即数+PC送寄存器 | 指令格式参考RISC-V32指令集，最终功能以RARS模拟器为准。 |
| 26 | SLTIU | 小于无符号立即数置数 |
| 27 | LHU | 加载半字 |
| 28 | BGE | 大于等于跳转 |

# 总体方案设计

## 单周期CPU设计

本次设计单周期CPU采用的方案是硬布线控制器，且采用指令存储器和数据存储器相分离的结构以避免资源冲突。通过填写提供的Excel表格得到各种信号的产生逻辑，由此合并不同指令的数据通路来构造支持表1.1中指令集的单周期CPU。在实现过程中，使用Logisim仿真平台提供的硬件构建电路，完成电路后导入benchmark程序进行联调测试。

总体结构图如图2.1所示。

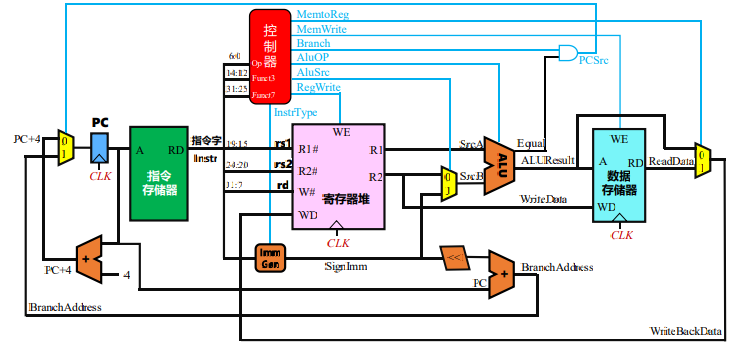


图2.1单周期CPU总体结构图

### 主要功能部件

主要的功能部件有程序计数器PC、指令存储器IM、寄存器堆Regfile、运算器ALU、数据存储器DM。

#### 程序计数器PC

程序计数器PC主要用于提供下一条指令的地址，以从指令存储器取出指令。一般情况下，程序计数器的数据来源是原PC值加4个字节得到的结果，也即当前指令的下一条指令的地址，其他数据来源也可能是条件跳转指令如BEQ、无条件跳转指令JAL。

#### 指令存储器IM

指令存储器IM用于存储需要执行的指令，由于在执行过程中不可进行更改，因此在Logisim仿真平台中采用只读存储器ROM实现。在执行不同的测试程序时，需要加载对应的数据镜像。

#### 运算器ALU

运算器用于进行各种运算，以从Regfile中读取的寄存器内容或者指令中的立即数扩展得到的值作为运算数，输出运算结果或者比较信号供其他电路使用。ALU引脚功能描述如表2.1所示，支持的运算功能及运算码如表2.2所示。

表.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2.2 |
| Result | 输出 | 32 | ALU运算结果 |
| Result2 | 输出 | 32 | 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 |
| Equal | 输出 | 1 | Equal=(x==y)?1:0, 对所有操作有效 |

表.运算码及其运算功能描述

| ALU\_OP | 运算功能 |
| --- | --- |
| 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 |

#### 寄存器堆Regfile

寄存器堆是对CS3410包中Regfile的封装，提供了32个通用寄存器。具体寄存器说明如表2.3所示。

表2.3寄存器说明

| 寄存器 | 别名 | 说明 |
| --- | --- | --- |
| x0 | zero | 恒零寄存器 |
| x1 | ra | 返回地址 |
| x2 | sp | 栈指针 |
| x3 | gp | 全局指针 |
| x4 | tp | 线程指针 |
| x5-x7 | t0-t2 | 临时变量寄存器0到2 |
| x8 | s0/fp | 保存寄存器0/栈帧指针 |
| x9 | s1 | 保存寄存器1 |
| x10-x17 | a0-a7 | 函数参数0到7 |
| x18-x27 | s2-s11 | 保存寄存器2到11 |
| x28-x31 | t3-t6 | 临时变量寄存器3到6 |

#### 数据存储器

数据存储器用来存储程序运行中产生的数据，需要支持读写操作，因此使用RAM来实现。在按字访问、半字访问、字节访问的不同模式下，需要使用多片RAM实现。

### 数据通路的设计

表.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 |
| ADD | PC+4 | PC | RS1 | RS2 | RD | ALU | RS1 | RS2 | 5 | - | - |
| SUB | PC+4 | PC | RS1 | RS2 | RD | ALU | RS1 | RS2 | 6 | - | - |
| AND | PC+4 | PC | RS1 | RS2 | RD | ALU | RS1 | RS2 | 7 | - | - |
| OR | PC+4 | PC | RS1 | RS2 | RD | ALU | RS1 | RS2 | 8 | - | - |
| SLT | PC+4 | PC | RS1 | RS2 | RD | ALU | RS1 | RS2 | 11 | - | - |
| SLTU | PC+4 | PC | RS1 | RS2 | RD | ALU | RS1 | RS2 | 12 | - | - |
| ADDI | PC+4 | PC | RS1 | - | RD | ALU | RS1 | Imm | 5 | - | - |
| ANDI | PC+4 | PC | RS1 | - | RD | ALU | RS1 | Imm | 7 | - | - |
| ORI | PC+4 | PC | RS1 | - | RD | ALU | RS1 | Imm | 8 | - | - |
| XORI | PC+4 | PC | RS1 | - | RD | ALU | RS1 | Imm | 9 | - | - |
| SLTI | PC+4 | PC | RS1 | - | RD | ALU | RS1 | Imm | 11 | - | - |
| SLLI | PC+4 | PC | RS1 | - | RD | ALU | RS1 | Imm | 0 | - | - |
| SRLI | PC+4 | PC | RS1 | - | RD | ALU | RS1 | Imm | 2 | - | - |
| SRAI | PC+4 | PC | RS1 | - | RD | ALU | RS1 | Imm | 1 | - | - |
| LW | PC+4 | PC | RS1 | - | RD | DM | RS1 | Imm | 5 | ALU | - |
| SW | PC+4 | PC | RS1 | RS2 | - | - | RS1 | Imm | 5 | ALU | R2 |
| ECALL | PC+4 | PC | $a7 | $a0 | - | - | RS1 | RS2 | - | - | - |
| BEQ | PC+OFFSET/PC+4 | PC | RS1 | RS2 | - | - | RS1 | RS2 | - | - | - |
| BNE | PC+OFFSET/PC+4 | PC | RS1 | RS2 | - | - | RS1 | RS2 | - | - | - |
| JAL | PC+OFFSET | PC | - | - | RD | PC+4 | - | - | - | - | - |
| JALR | ALU | PC | RS1 | - | RD | PC+4 | RS1 | Imm | 5 | - | - |
| CSRRSI | PC+4 | PC | - | - | - | - | - | - | - | - | - |
| CSRRCI | PC+4 | PC | - | - | - | - | - | - | - | - | - |
| URET | PC+4 | PC | - | - | - | - | - | - | - | - | - |
| AUIPC | PC+4 | PC | - | - | RD | PC+OFFSET | - | - | - | - | - |
| SLTIU | PC+4 | PC | RS1 | - | RD | ALU | RS1 | Imm | 12 | - | - |
| LHU | PC+4 | PC | RS1 | - | RD | DM | RS1 | Imm | 5 | ALU | - |
| BGE | PC+OFFSET/PC+4 | PC | RS1 | RS2 | - | - | RS1 | RS2 | 11 | - | - |

### 控制器的设计

首先对控制信号进行统计，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2.5所示。

表.主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| ALU\_OP | 0-12 | 选择ALU要进行的运算类型，具体运算见表2.2 |
| MemToReg | 0 | 选择ALU运算结果为寄存器写入值 |
| 1 | 选择数据存储器读出值为寄存器写入值，如LW、LHU指令 |
| MemWrite | 0 | **-** |
| 1 | 加载值到数据存储器，如SW指令 |
| ALU\_SrcB | 0 | ALU的输入端送入R2 |
| 1 | ALU的输入端B送入立即数 |
| RegWrite | 0 | **-** |
| 1 | 将RDin写入寄存器Rd |
| ecall | 0/1 | ecall指令译码信号 |
| S\_Type | 0/1 | S型指令译码信号 |
| BEQ | 0/1 | BEQ指令译码信号 |
| BNE | 0/1 | BNE指令译码信号 |
| JAL | 0/1 | JAL指令译码信号 |
| JALR | 0/1 | JALR指令译码信号 |
| BGE | 0/1 | BGE指令译码信号 |
| AUIPC | 0/1 | AUIPC指令译码信号 |
| LHU | 0/1 | LHU指令译码信号 |
| R1Used | 0 | 未使用R1#源寄存器 |
| 1 | 使用R1#源寄存器 |
| R2Used | 0 | 未使用R2#源寄存器 |
| 1 | 使用R2#源寄存器 |

对照所有控制信号，依次分析各条指令，分析该指令执行过程中需要哪些控制信号，对于与本条指令无关的控制信号，控制信号的取值一律为0，以简化控制器电路的设计。该控制信号表的框架如表2.6所示。

表2.6主控制器控制信号框架

| 指令 | **ALU\_OP** | **MemToReg** | **MemWrite** | **ALU\_SrcB** | **RegWrite** | **ecall** | **S\_Type** | **BEQ** | **BNE** | **JAL** | **JALR** | **BGE** | **LHU** | **AUIPC** | **R1Used** | **R2Used** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ADD | 5 |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| SUB | 6 |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| AND | 7 |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| OR | 8 |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| SLT | 11 |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| SLTU | 12 |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| ADDI | 5 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| ANDI | 7 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| ORI | 8 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| XORI | 9 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| SLTI | 11 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| SLLI | 0 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| SRLI | 2 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| SRAI | 1 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |
| LW | 5 | 1 |  | 1 | 1 |  |  |  |  |  |  |  |  |  | 1 |  |
| SW | 5 |  | 1 | 1 |  |  | 1 |  |  |  |  |  |  |  | 1 | 1 |
| ECALL |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  | 1 | 1 |
| BEQ |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  | 1 | 1 |
| BNE |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  | 1 | 1 |
| JAL |  |  |  |  | 1 |  |  |  |  | 1 |  |  |  |  |  |  |
| JALR | 5 |  |  | 1 | 1 |  |  |  |  |  | 1 |  |  |  | 1 |  |
| CSRRSI |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| CSRRCI |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| URET |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| AUIPC |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |
| SLTIU | 12 |  |  | 1 | 1 |  |  |  |  |  |  |  |  | 1 | 1 |  |
| LHU | 5 | 1 |  | 1 | 1 |  |  |  |  |  |  |  | 1 |  | 1 |  |
| BGE | 11 |  |  |  |  |  |  |  |  |  |  | 1 |  |  | 1 | 1 |

## 中断机制设计

### 总体设计

中断可以分为外部中断和内部中断，而本次实验中仅考虑可以屏蔽的外部中断源。在设计中断时，应考虑单级中断和多级中断，需要硬件和软件的共同支持。对于单级中断的设计，在转到中断服务程序前，需要在中断断点寄存器EPC中维护断点（PC的值）以确保执行完毕中断服务程序后能够返回进入中断的位置。对于多级中断，由于涉及到不同中断源的处理优先级，需要在硬件中判断新的中断能否打断当前中断，并使用堆栈存储处理多级中断时的不同断点值。

### 硬件设计

单级中断硬件设计：使用中断按键参考电路存储中断请求信号，当同时存在多个中断请求信号时，使用优先编码器输出优先级最高的中断号。当正在处理某个中断时，其他中断请求无法打断当前中断。在进行中断响应前，在中断断点寄存器EPC中维护发生中断时 PC值。当中断服务程序执行完毕时，CPU会根据URET 指令执行中断返回操作，将 EPC中的值送至 PC以跳转到断点处以继续执行主程序。只有当进行中断响应时更新EPC 中的值，并且需要根据中断号向PC送入正确的中断服务程序入口地址。

多级中断硬件设计：与单级中断不同，当正在处理某个中断时，其他比当前中断优先级高的中断请求可以打断当前中断。因此需要存储每次中断响应时的相关信息。首先，与单级中断只需设置一个EPC不同，三级中断需要设置3个EPC以存放至多3个中断同时发生时的PC值。其次，当在处理中断时，如果发生新的中断，需要根据中断优先级判断应该处理哪个中断（先执行优先级高的中断），如果要处理新的中断，需要将新的中断号和被打断的中断号保存在相应寄存器中。此外，控制器还需要支持开关中断指令CSRRSI和CSRRCI，与中断请求信号和ERET指令一起组成是否响应中断的逻辑。

### 软件设计

软件设计方面，主要是支持CSRRSI、CSRRCI、URET指令，配合硬件完成开关中断、中断返回的操作。同时需要从程序中获取中断服务程序入口地址，作为常量连接到硬件相关逻辑电路中以达到模拟中断向量表的目的。

## 理想流水线CPU设计

### 总体设计

RISC-V指令执行过程可以分为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。通过在不同阶段之间插入流水寄存器锁存当前段的数据，使得不同阶段能够处理不同的指令，提高执行效率。理想流水线并不考虑分支冲突、数据冲突等冒险问题，是后续气泡流水线和重定向流水线的基础，因此需要在理想流水线设计中完成流水线寄存器的设计及流水线基本结构。

### 流水接口部件设计

在设计流水接口部件时，参考了设计运算器时使用到的乘法流水线接口部件。每个流水接口部件包括时钟端CLK，同步清零端，使能端以及数据输入输出端口。由于不同阶段需使用的数据不同，因此不同阶段间的流水接口部件的数据端口不尽相同，需要根据实际情况修改流水寄存器需要所存的数据。

### 理想流水线设计

完成流水接口部件的设计后，只需要将单周期CPU设计中的不同阶段需要用到的部件分离，并在不同阶段间插入流水接口部件。在设计时需要注意细微的逻辑变化，如在写入寄存器时，Rd#、RDin、RegWrite应该均来自WB段，否则会导致错误逻辑。

## 气泡流水线CPU设计

### 总体设计

气泡流水线对于分支冲突的处理方法：当在EX段需要进行分支跳转时，清除IF/ID流水寄存器和ID/EX流水寄存器的内容以清除误取指令。

气泡流水线对于数据冲突的处理方法：在译码ID段判断是否与EX段或MEM段发生数据相关，如果发生数据相关，则暂停ID/EX段流水寄存器的执行（使能端赋高电平）以插入空气泡，等待数据不再相关后继续推进流水线的执行。

### 气泡逻辑设计

* 数据相关模块：通过源寄存器的使用情况、EX段或MEM段是否写入寄存器（RegWrite信号）、ID段寄存器R1#/R2#与EX段WriteReg#或MEM段WriteReg#是否相等组成的逻辑来检测是否发生数据相关。
* 由数据相关信号、分支跳转信号、halt信号输出PC寄存器的使能端信号、各个流水线寄存器的使能端和清零端信号，以送入各个部件中。

## 重定向流水线CPU设计

### 总体设计

重定向流水线对于分支冲突的处理方法与气泡流水线相同。

重定向流水线对于数据冲突的处理方法：为了解决气泡流水线中插入气泡过多的问题，当发生数据相关时，不再插入气泡，而是在下一个时钟周期将正确的尚未写回到寄存器中的数据重定向到EX段。

LoadUse相关：如果相邻两条指令存在数据相关，且前一条指令是访存指令时，这种数据相关不能用重定向进行处理。如果采用重定向方式，需要将数据存储器的输出重定向到EX段，使得关键路径包含访存阶段，从而加长关键路径影响CPU工作效率。因此，需要额外判断LoadUse相关，并使用插入气泡的方式解决冲突。

### 重定向逻辑设计

* 重定向选择信号模块：通过对气泡流水线中数据相关模块稍加修改，输出Fwd1和Fwd2信号，由Fwd1和Fwd2信号作为多路选择器的选择端以选择EX段需要重定向的正确数据。
* LoadUse相关模块：通过源寄存器的使用情况、EX段是否访存（MemToReg信号）、ID段寄存器R1#/R2#与EX段WriteReg#是否相等组成的逻辑来检测是否发生LoadUse相关。
* 由LoadUse相关信号、分支跳转信号、halt信号输出PC寄存器的使能端信号、各个流水线寄存器的使能端和清零端信号，以送入各个部件中。

## 动态分支预测机制设计

### 总体设计

动态分支预测基于重定向流水线，通过优化对分支冲突的处理以减少插入气泡数。

### BTB表设计

* 分支预测缓冲器：在BTB表中用于存放分支指令的分支跳转历史统计信息，通过八路并发比较进行全相联查找，其包含的表项为Valid位、分支指令地址、分支跳转地址、分支预测历史位。
* 淘汰算法：在BTB表已满时，使用LRU淘汰算法进行淘汰置换。
* 双位状态预测机：以输入的现态和是否进行分支跳转的信号得到次态输出，其现态即为BTB表中的分支预测历史位，根据是否进行分支跳转来预测是否进行跳转。双位预测状态转换如图2.2所示。

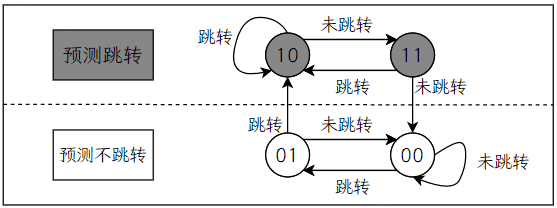


图2.2双位预测状态转换图

# 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。halt为停机信号，将此控制信号通过非门取反之后送入使能端。当需要进行停机时，halt控制信号为1，经过非门之后为0，从而忽略时钟输入，使整个电路停机。如图3.1所示。

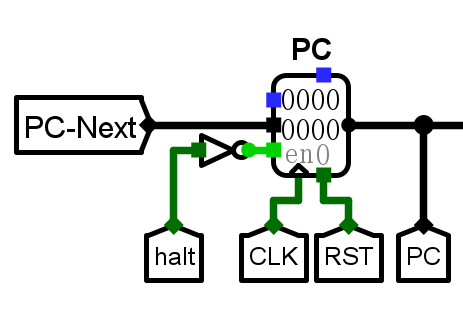


图3.1程序计数器（PC）

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），并设置指令存储器的地址位宽为10位，数据位宽为32位。而PC中存储的指令地址有32位，因此需要将32位指令地址高位部分和字节偏移部分屏蔽后送入ROM作为指令地址得到指令字IR。实现时，分线器取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。

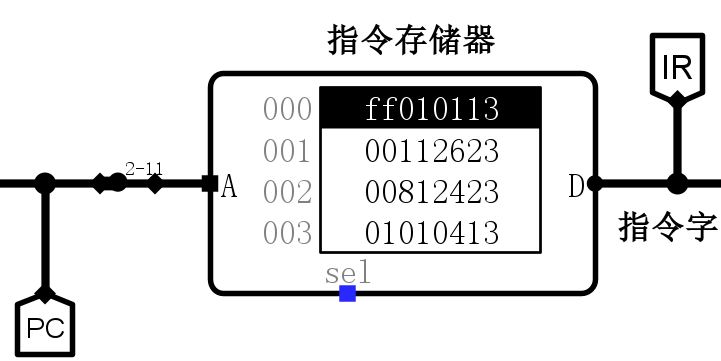


图3.2指令存储器（IM）

1. 运算器ALU

使用cs3410.jar 包中 MIPS ALU 部件即可，输入为运算数A、运算数B、运算码Alu\_OP，输出为运算结果Result、是否相等Equal等。

1. 数据存储器

由于存在需要按半字访问数据存储器的指令，因此使用4片随机存储器RAM实现数据存储器。每片RAM的地址位宽为10位，数据位宽为8位。在按字节访问时使用低两位作为片选信号来片选4片RAM，按半字访问时使用第1位作为片选信号。如图3.3所示。

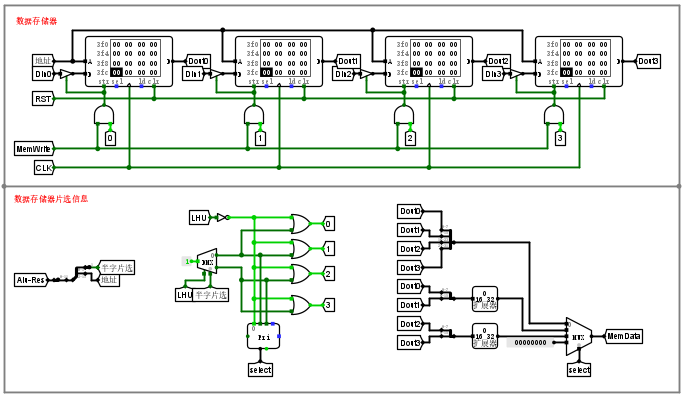


图.数据存储器

### 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。根据总体方案设计中数据通路设计的内容，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2.4所示。

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，将各个主要功能部件进行连接，对于所有的多输入部件使用多路选择器进行输入选择，最终完成数据通路的搭建。具体数据通路如图3.4所示。

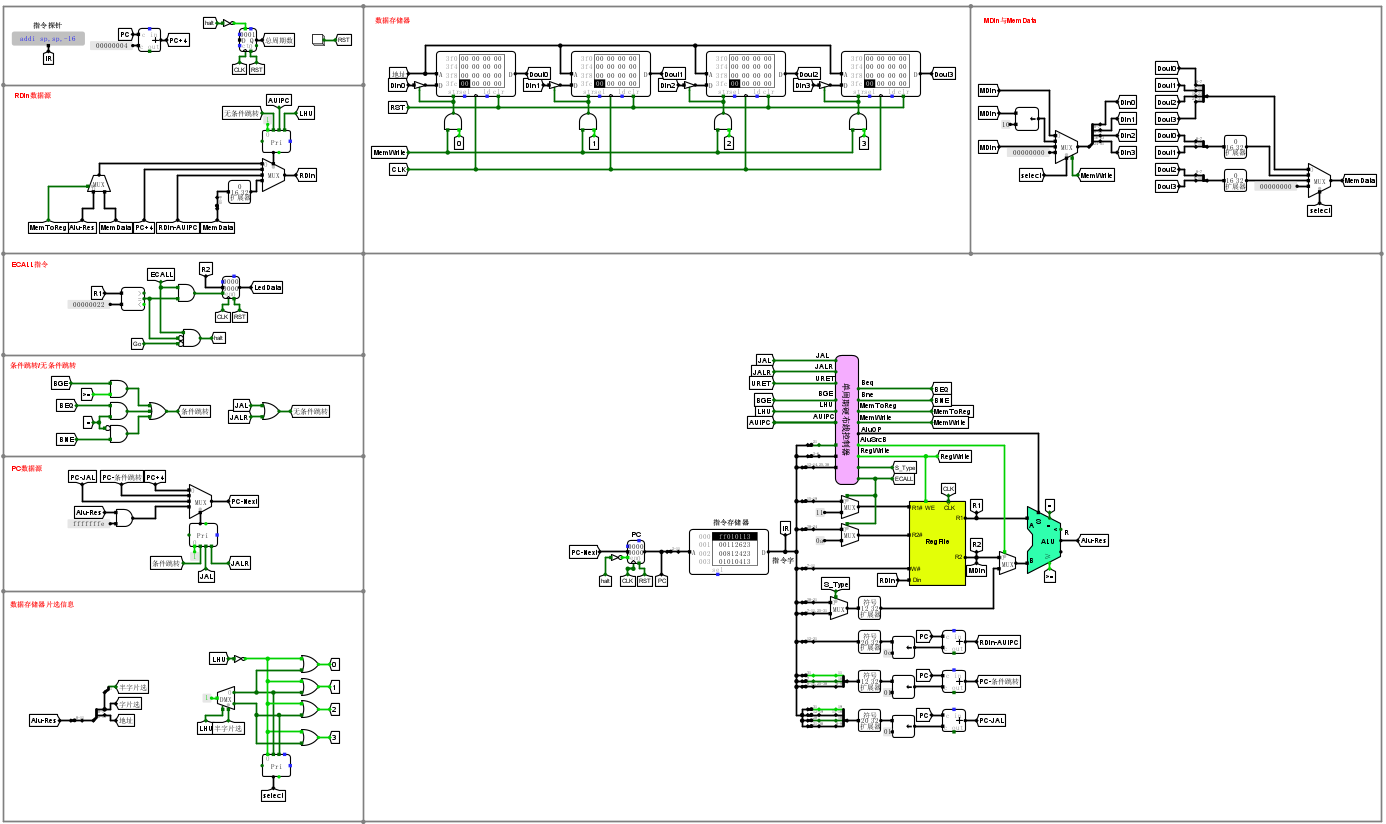


图3.4单周期CPU数据通路（Logisim）

### 控制器的实现

1. 主控制器

主控制器控制信号如表2.6所示，主控制器的具体实现如图3.5所示。

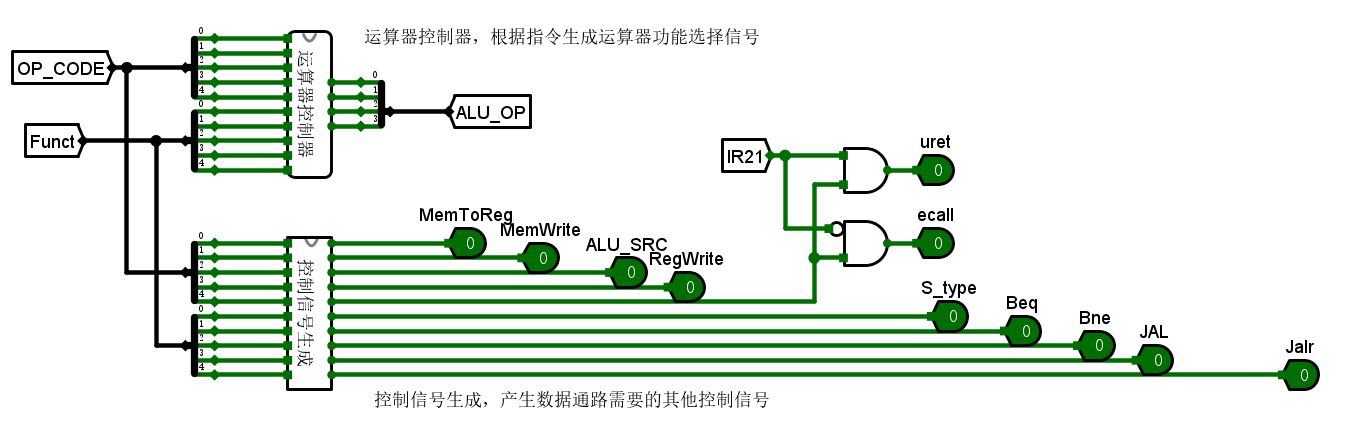


图.主控制器的实现

1. Branch控制器

Branch控制器主要负责根据条件分支和无条件分支决定下一条指令的地址，主要使用多路选择器和优先编码器实现，具体实现如图3.6所示。

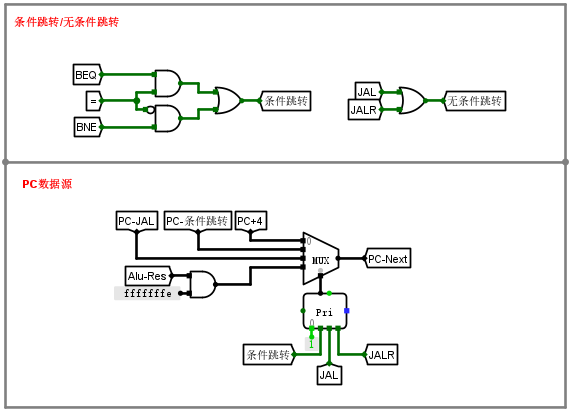


图3.6 Branch控制器实现

1. ECALL控制器

ECALL控制器主要负责当出现ECALL指令时，根据寄存器$a7是否等于34决定是否停机，具体实现如图3.7所示。

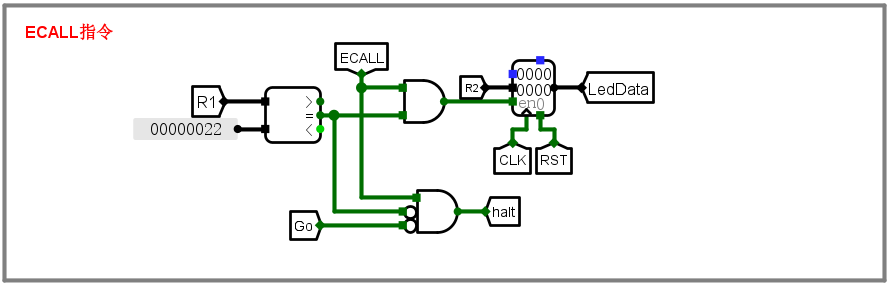


图. ECALL控制器实现

## 中断机制实现

### 单级中断

1. 中断按键信号采样电路

使用提供的中断按键信号采样电路存储中断请求，并使用优先编码器输出优先级最高的中断号以及是否有中断请求，具体实现如图3.8所示。

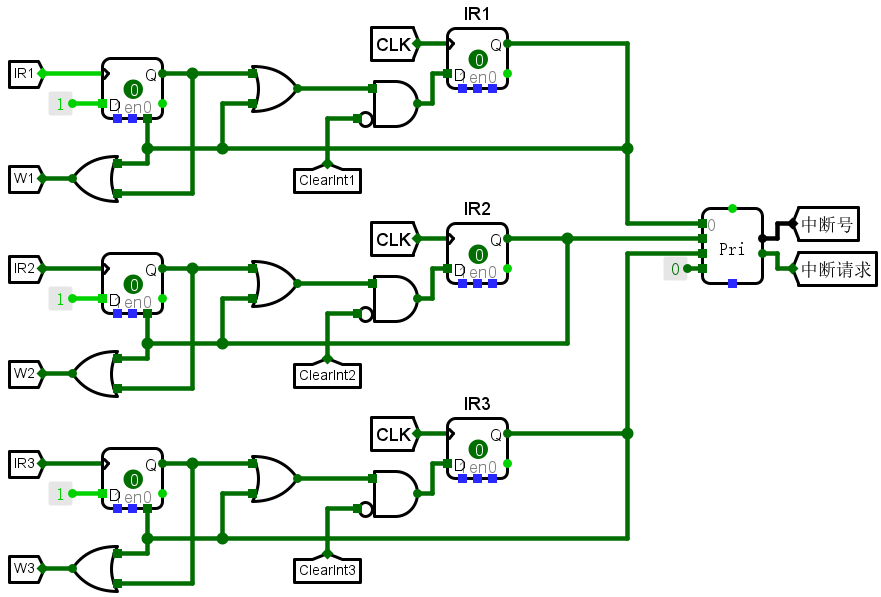


图3.8单级中断按键信号采样电路

1. 中断相关寄存器/硬件支持

为了实现单级中断，需要实现中断程序返回寄存器mEPC、中断使能寄存器IE、中断响应信号INT、中断服务程序入口表。具体实现如下：当发生中断请求且中断使能时将PC送入mEPC寄存；当发生中断请求时置IE为1，当执行URET指令时置IE为0；当发生中断请求且中断使能时置中断响应信号INT为1；使用多路选择器输出中断服务程序入口。具体实现如图3.9所示。

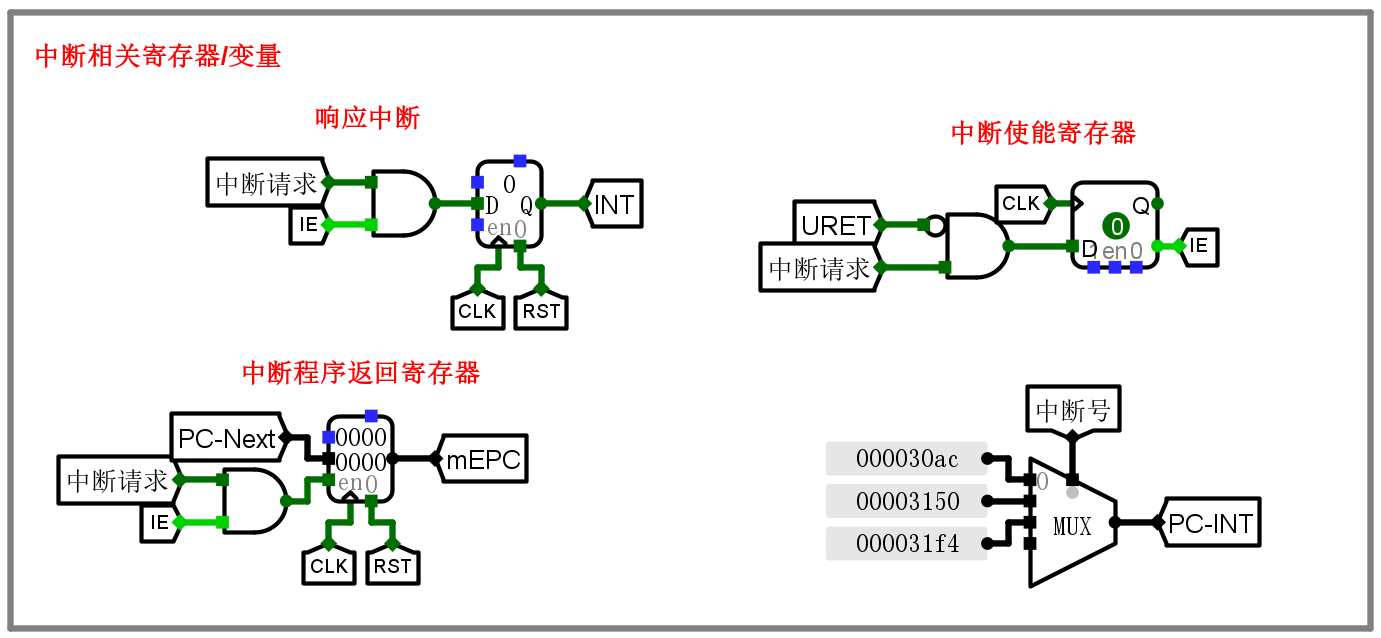


图3.9单级中断相关硬件支持

1. 中断相关软件支持

为了实现单级中断，需要实现中断返回指令的支持。当执行URET指令时，将mEPC寄存器保存的断点地址送入PC寄存器，具体实现如图3.10所示。

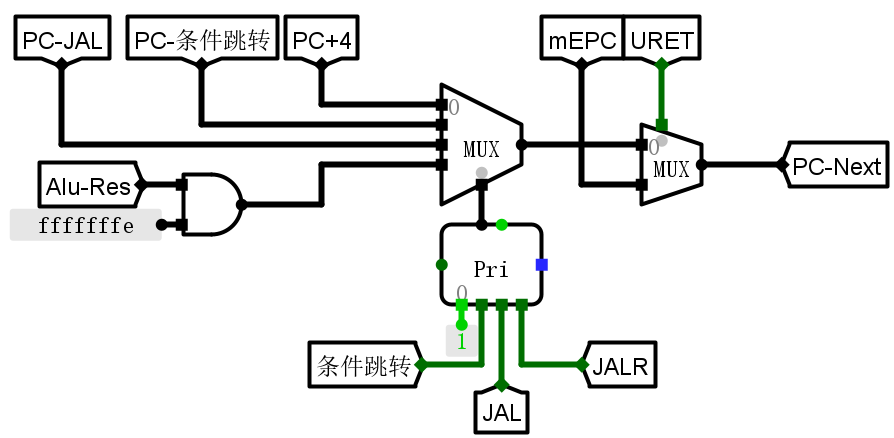


图3.10单级中断相关软件支持

1. 单级中断数据通路

根据中断信号采样电路、中断相关寄存器、URET指令支持，即可在单周期CPU的基础上实现单级中断，具体实现如图3.11所示。



图.单级中断数据通路

### 多级中断

1. 中断按键信号采样电路

多级中断与单级中断相比，增加了寄存当前中断号的电路，如图3.12所示。

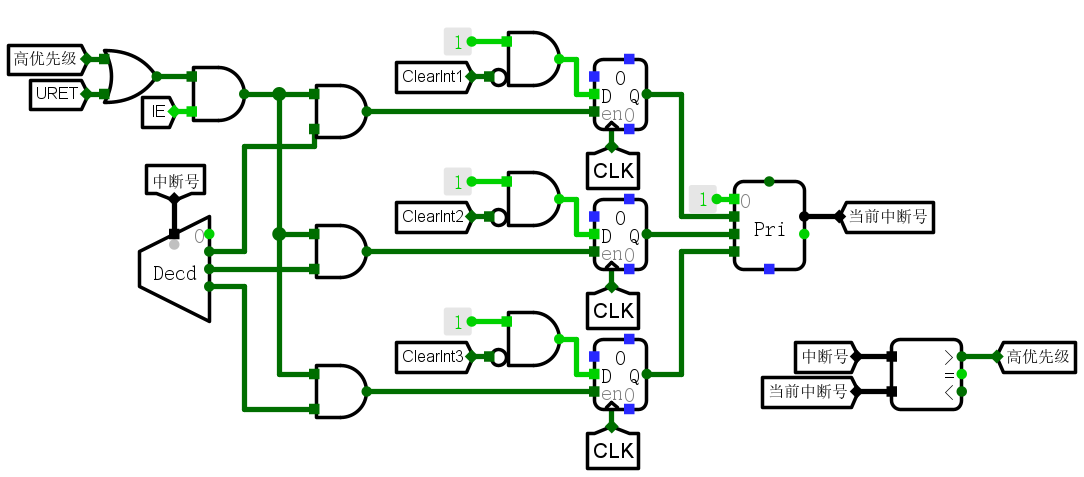


图3.12多级中断当前中断号寄存电路

1. 中断相关寄存器/硬件支持

为了实现多级中断，需要设置多个终端程序返回寄存器mEPC。同时，使用CSRRSI、CSRRSI指令模拟开关中断，因此需要更改中断使能寄存器IE的相关逻辑。其余硬件支持与单级中断大致相同。具体实现如图3.13所示。

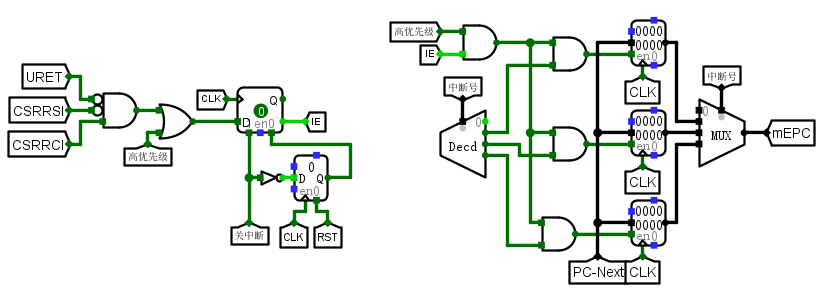


图3.13多级中断相关硬件支持

1. 多级中断数据通路

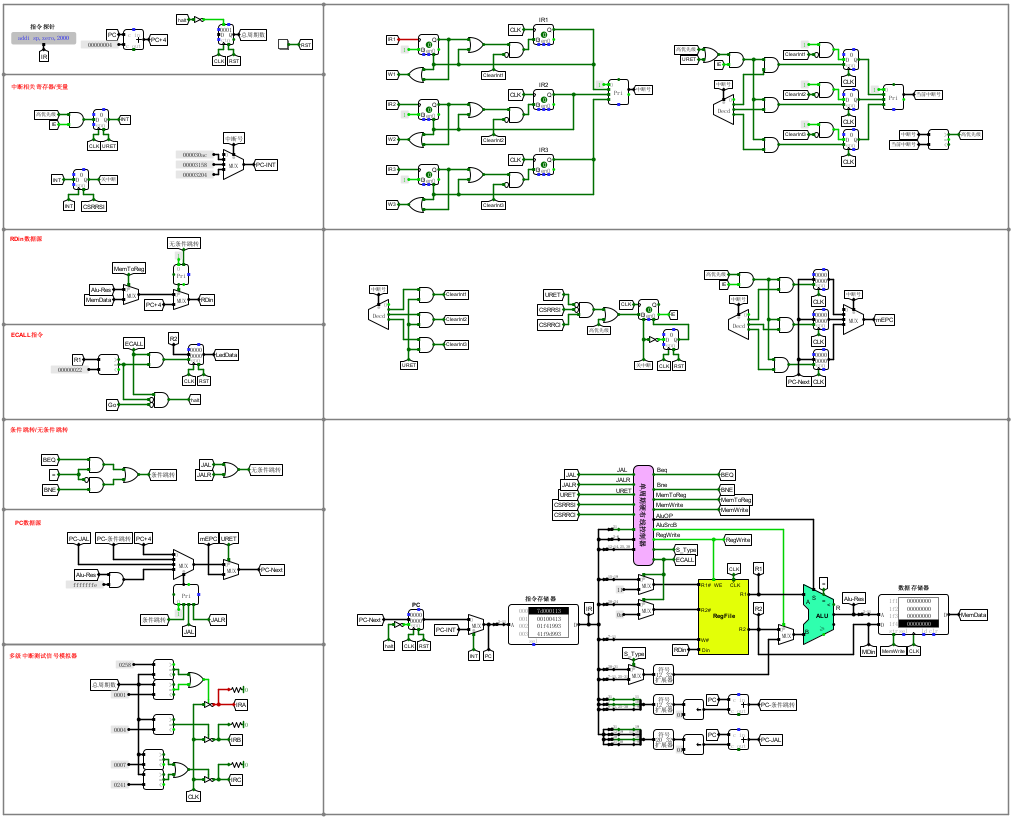


图.多级中断数据通路

## 理想流水线CPU实现

### 流水接口部件实现

流水接口部件内部采用寄存器进行锁存，寄存器均采用上升沿触发，同时需要实现同步清零端、使能端。以ID/EX段流水接口部件为例，具体实现如图3.15所示。

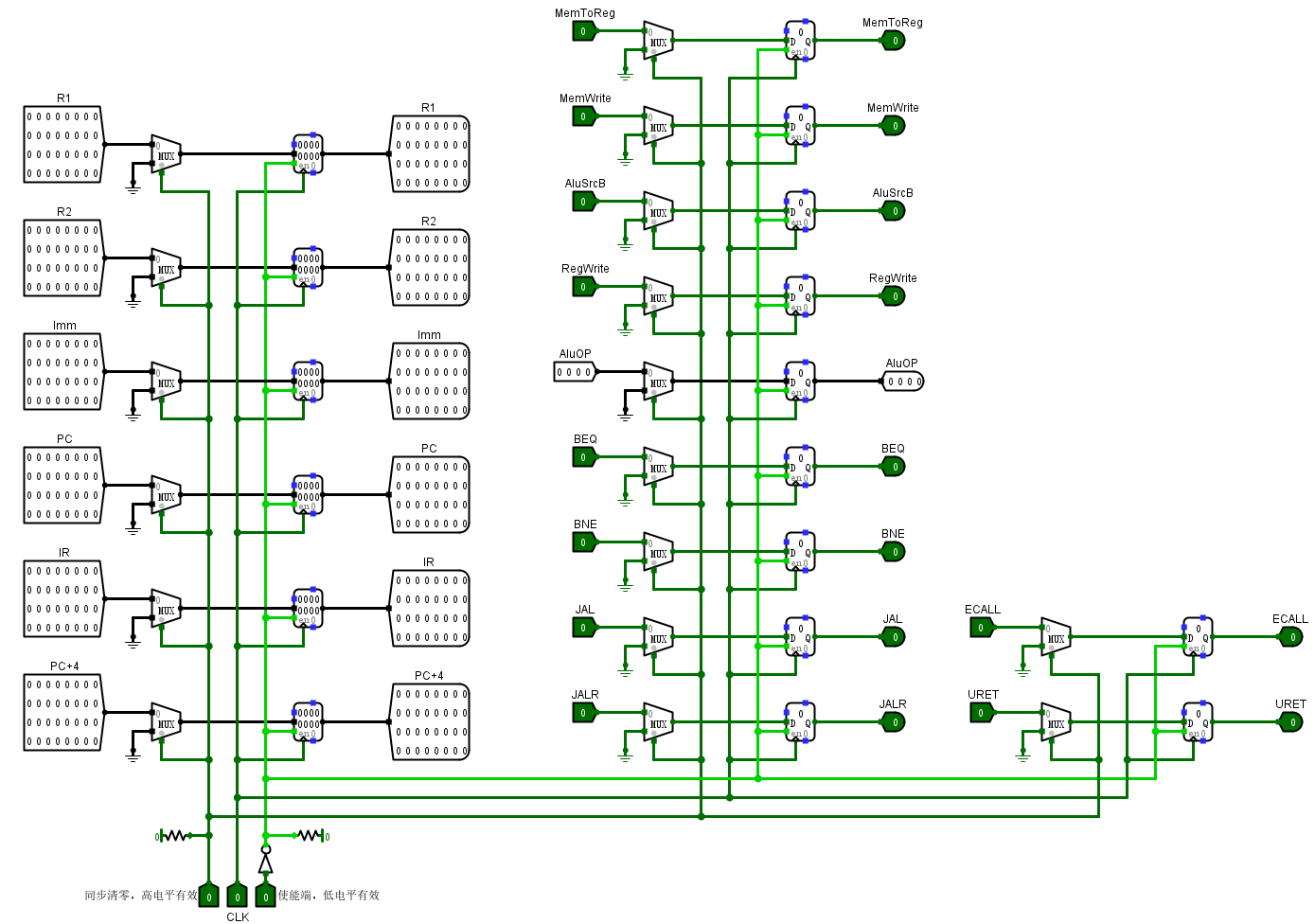


图3.15流水接口部件实现

封装流水接口部件时，将输入端布置在左侧，输出端布置在右侧，使能端、清零端、时钟端布置在上下侧。以ID/EX段流水接口部件为例，具体封装如图3.16所示。

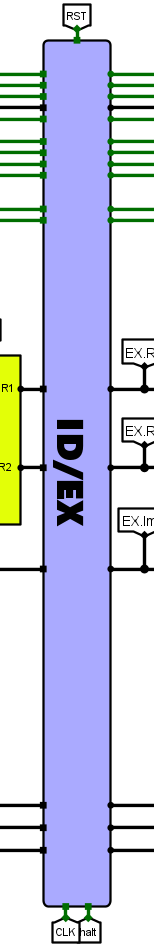


图3.16流水接口部件封装

### 理想流水线实现

理想流水线不涉及对分支冲突和数据冲突的处理，因此只需完成各控制信号的传递.在布线时，需要实时调整流水接口部件的封装以保证布线的美观性，同时需要考虑不同段数据的正确性，如寄存器写入时的相关信号与数据。具体实现如图3.17所示。

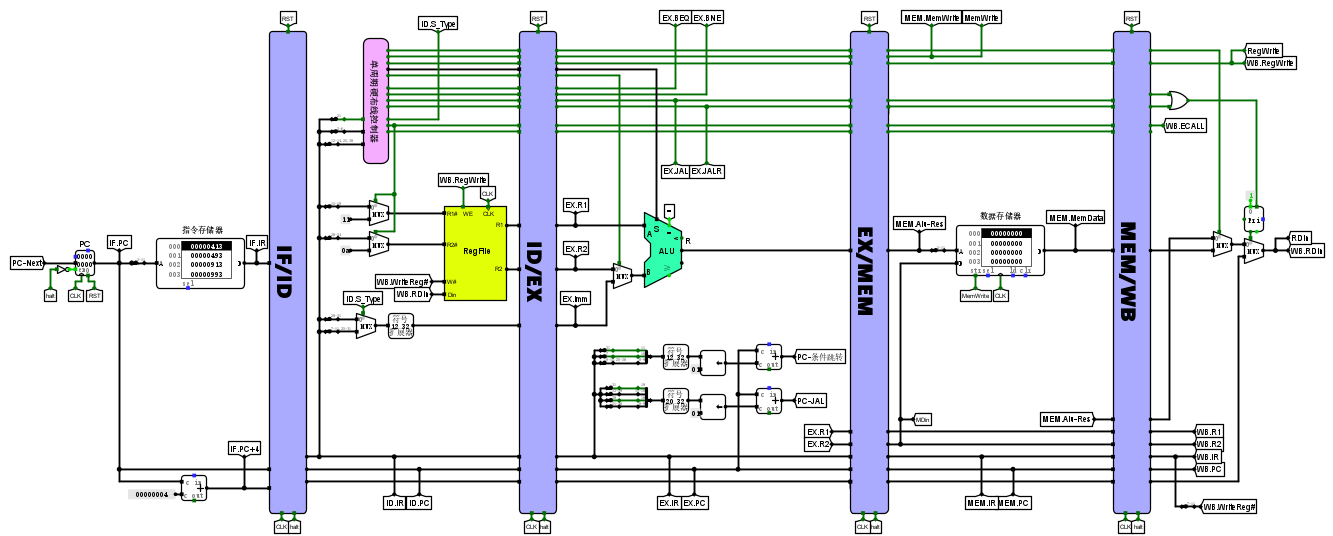


图3.17理想流水线实现

## 气泡流水线CPU实现

### 气泡逻辑实现

在实现插入气泡的逻辑时，需要分别实现数据相关检测逻辑、产生为了完成插入气泡操作需要送入相关部件的信号，如IF.RST、ID.RST、Stall、~Stall分别送入IF/ID流水寄存器的清零端、ID/EX流水寄存器的清零端、PC寄存器的使能端、IF/ID流水寄存器的使能端。具体实现如图3.18所示。

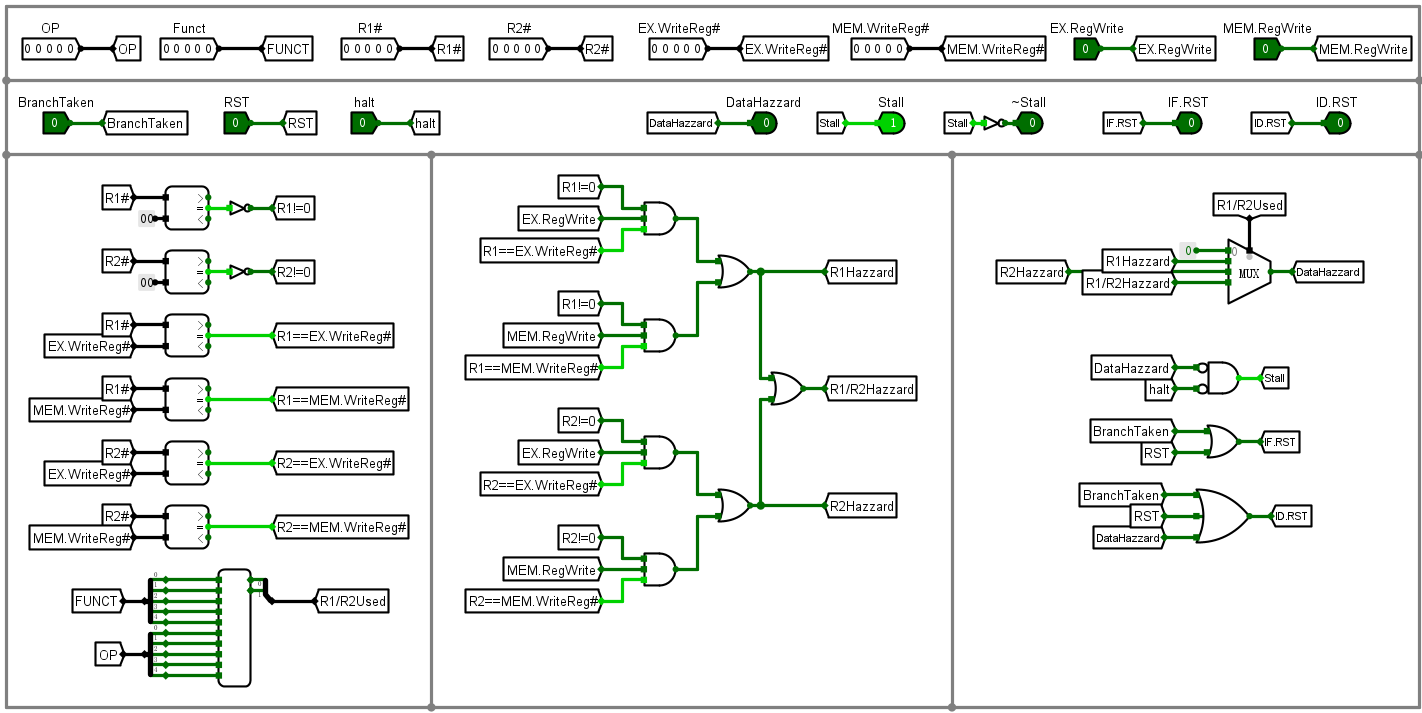


图3.18气泡逻辑实现

将气泡逻辑的相关实现封装为气泡逻辑部件，如图3.19所示。



图3.19气泡逻辑封装

### 气泡流水线实现

将气泡逻辑部件添加到原有的理想流水线中，并将各输入输出连接正确后，即可得到气泡流水线的具体实现，如图3.20所示。



图3.20气泡流水线实现

## 重定向流水线CPU实现

### 重定向逻辑实现

在实现重定向逻辑时，只需要在气泡逻辑的基础上添加LoadUse相关的检测、Fwd1和Fwd2信号的生成，并更改相关信号的逻辑。具体实现如图3.21所示。

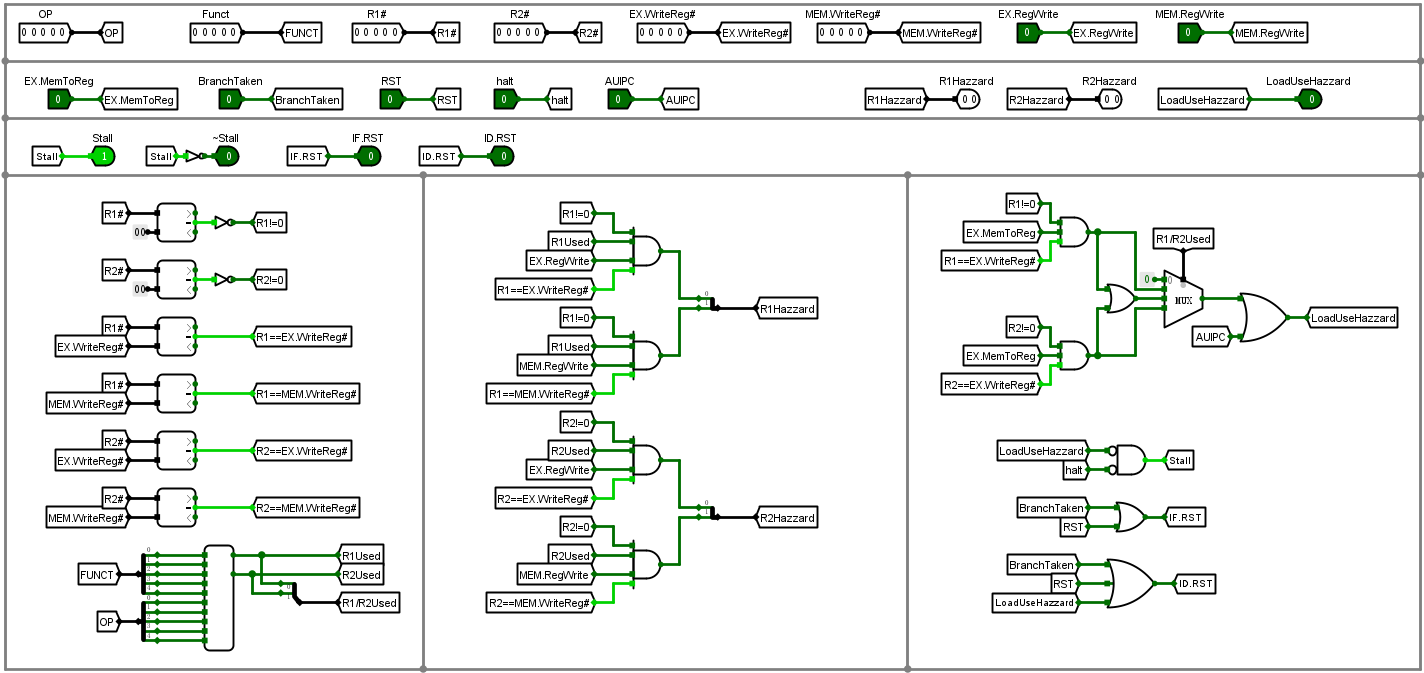


图3.21重定向逻辑实现

将气泡逻辑的相关实现封装为气泡逻辑部件，如图3.22所示。

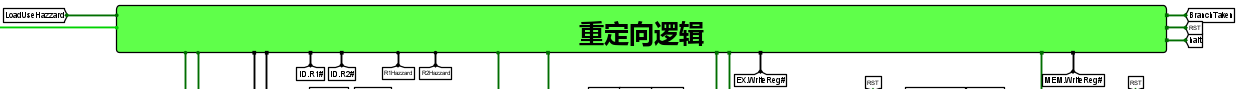


图3.22重定向逻辑封装

### 数据重定向实现

在EX段，使用Fwd1和Fwd2选择信号将正确的数据重定向到运算器的输入端，具体实现如图3.23所示。

### 重定向流水线实现

使用重定向逻辑部件替换气泡逻辑部件，并将各输入输出连接正确，添加数据重定向逻辑后，即可得到重定向流水线的具体实现，如图3.24所示。

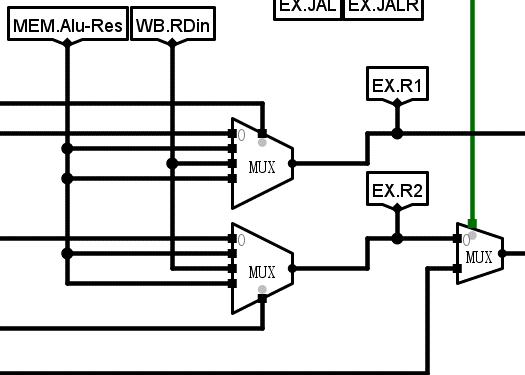


图3.23数据重定向实现

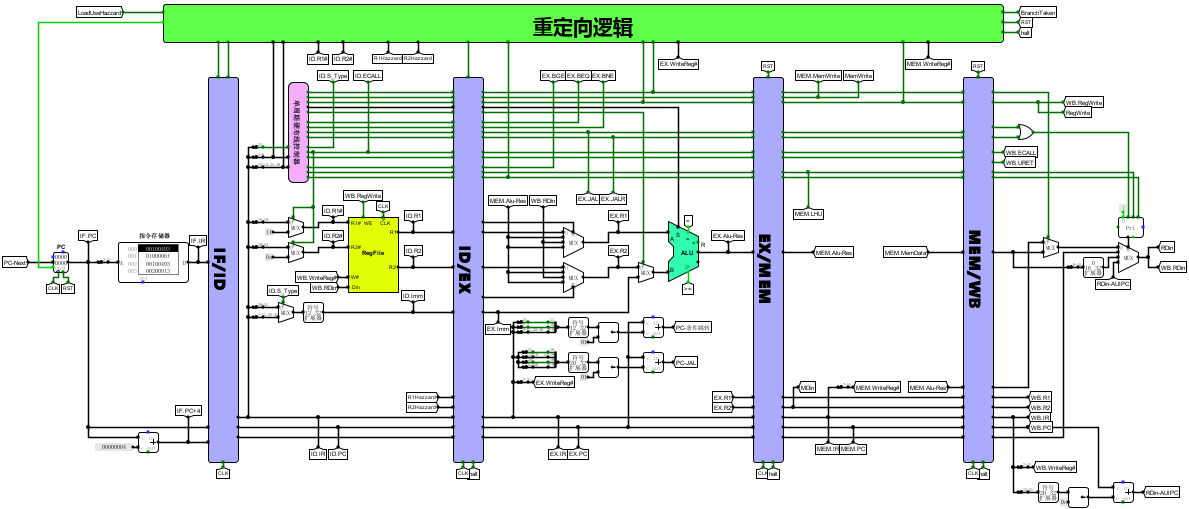


图3.24重定向流水线实现

## 动态分支预测机制实现

### 双位预测状态机实现

根据双位预测状态转换图（图2.2），使用Logisim软件的分析组合逻辑电路功能，得到双位预测状态机如图3.25所示。

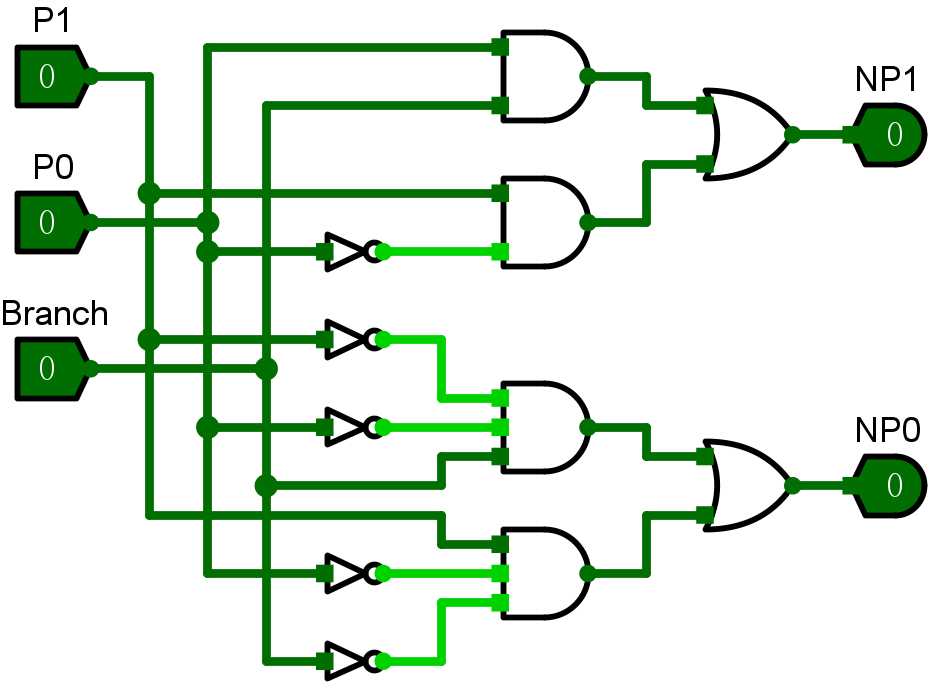


图3.25双位预测状态机

### BTB表实现

BTB表的存储设计与全相联Cache基本一致，有关八路并发比较、LRU淘汰算法等实现不再赘述，这里着重说明分支历史预测位的相关逻辑。

在存储分支历史预测位时，寄存器使用双位预测状态机输出的次态作为输入，并将现态作为状态机的输入。由于分支预测初始值也会影响预测结果，这里选择初始值为11，将分支历史预测位的高位作为预测结果输出。总体实现如图3.26所示。

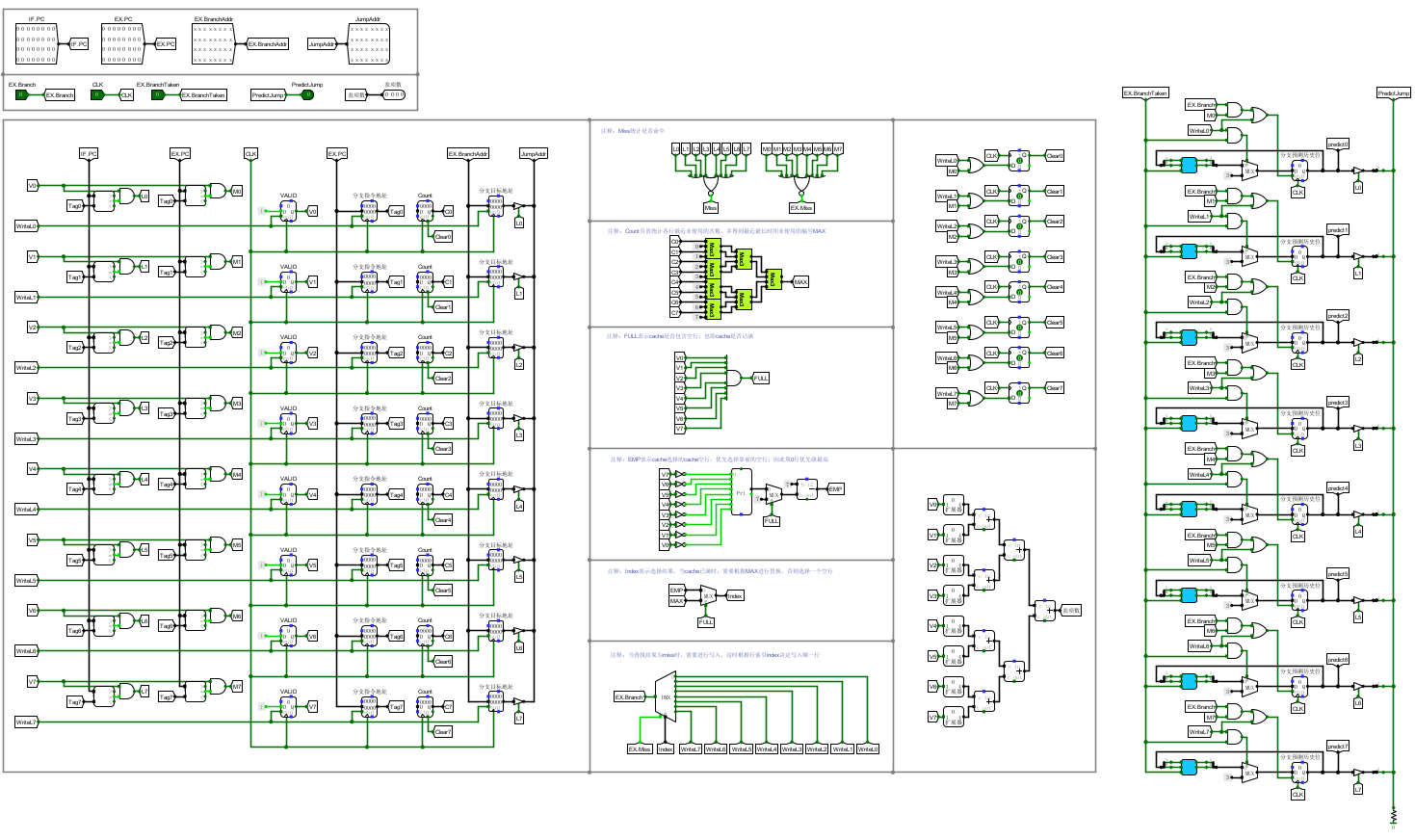


图3.26 BTB表总体实现

### 基于重定向流水线的动态分支预测机制实现

在重定向流水线的基础上，添加图3.27结构，使用PredictError代替BranchTaken送入重定向逻辑，即当分支预测失败时插入气泡。

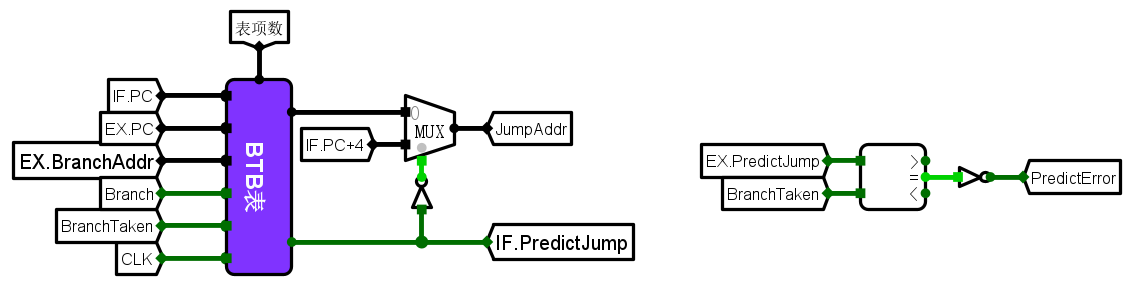


图3.27 BTB表的使用

在添加在流水接口上增加PredictJump接口、修改总周期数统计逻辑等细节后，可以得到动态分支预测的具体实现。

# 实验过程与调试

## 测试用例和功能测试

### CCAB指令测试

1. AUIPC指令

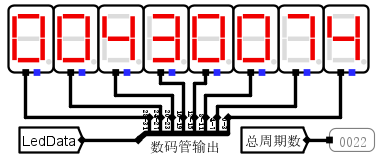


图4.1 AUIPC指令测试结果

1. SLTIU指令

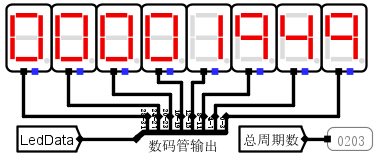


图4.2 SLTIU指令测试结果

1. LHU指令

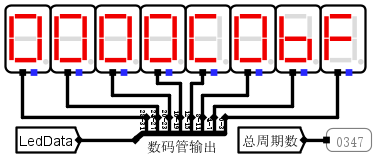


图4.3 LHU指令测试结果

1. BGE指令

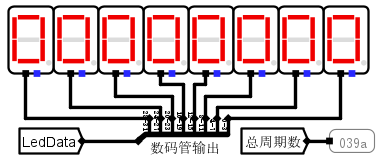


图4.4 BGE指令测试结果

### 流水线测试

1. 气泡流水线

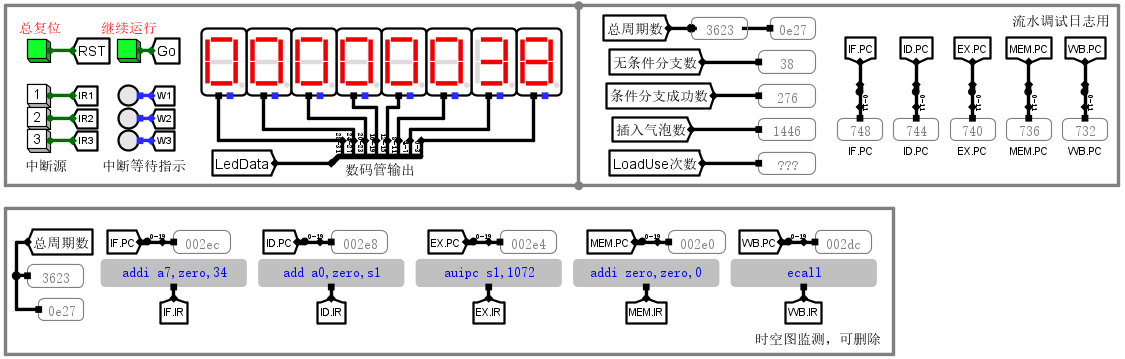


图4.5气泡流水线测试结果

1. 重定向流水线

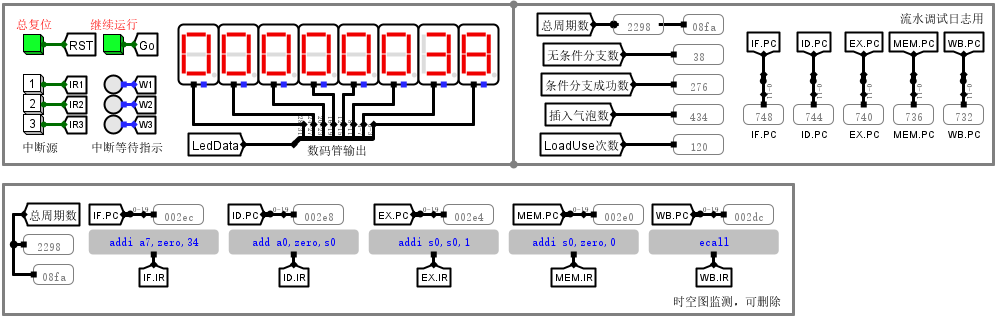


图4.6重定向流水线测试结果

1. 动态分支预测

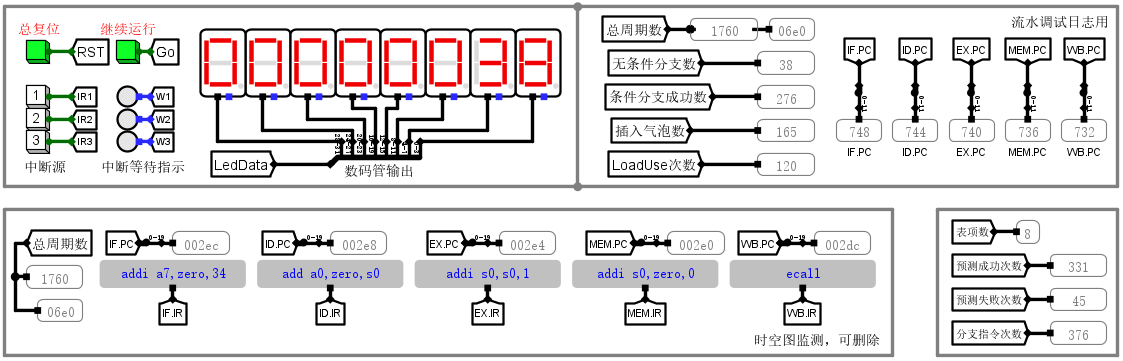


图4.7 动态分支预测测试结果

## 性能分析

表 4.1 不同方案时钟周期数对比

| 方案 | 时钟周期数 |
| --- | --- |
| 单周期CPU | 1545 |
| 气泡流水线 | 3624 |
| 重定向流水线 | 2297 |
| 动态分支预测 | 1759 |

单周期硬布线的时钟周期最短，但是每条指令的平均执行时间最长，效率最差；

气泡流水线时钟周期最多，主要是由于处理数据冲突和分支冲突时均采用插入气泡的方式，从而导致插入了大量气泡；

重定向流水线采用重定向方式处理数据冲突，减少了插入气泡数；

动态分支预测则在重定向的基础上优化了分支冲突的处理方式，效率最高。

## 主要故障与调试

### 多级中断中中断返回后无法响应其他中断请求

多级中断中，当某个中断服务程序执行完毕返回时，无法继续响应其他中断请求。

**故障现象：**当执行中断号为3的中断服务程序的过程中，发生了中断号为1的中断请求，由于中断优先级：3>1，因此继续执行当前中断服务程序。当当前中断服务程序返回时，却无法响应中断号为1的中断请求。

**原因分析：**在多级中断中，新中断发生时未能将中断号正确保存在相应寄存器中，导致当前中断执行结束时无法继续响应新中断。

**解决方案：**使用三个寄存器将每次的中断请求保存下来，利用 IE寄存器的状态和是否发生新中断信号作为使能端，更新历史记录，根据最近历史记录和当前中断号进行判断决定应该进行的中断号。具体实现如图4.8所示。

### 动态分支预测优化失败

动态分支预测优化失败，减少的气泡数未达到预期。

**故障现象：**正确完成动态分支预测时，理想的气泡数为1800左右，而当我完成动态分支预测时，经过屡次调整，插入气泡数也未达到理想水平。

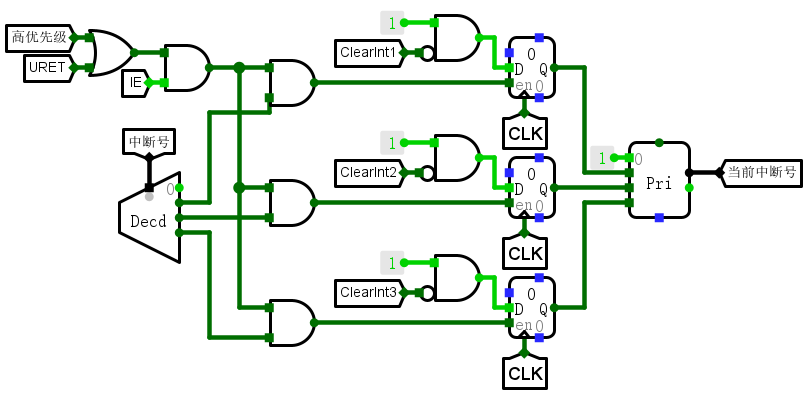


图.多级中断中断请求信号寄存器图

**原因分析：**由于初期布线设计BTB表时，写入BTB表的逻辑有误，分支预测历史位的更新逻辑有误，将BTB表预测结果与重定向流水线整合时有误，导致屡次修改调整，减少的气泡数未达到预期。

**解决方案：**

对于写入BTB表，应当在EX段中，EX.PC未命中且EX.Branch为1时写入；

对于分支预测历史位的更新逻辑，应当在EX段进行更新，因此需要使用EX.Branch信号及EX.PC的命中情况作为寄存器的使能端；

在与重定向流水线整合时，需要注意当预测失败时，送入PC寄存器的内容应为EX.PC+4，而非IF.PC+4。

在经过以上调整，并且置分支预测历史位初始值为11后，最终气泡数达到预期。

### 数据存储器按半字访问错误

数据存储器按半字访问得到的数据错误。

**故障现象：**在测试CCAB指令中的LHU指令时，理想的LED输出为0x00008281 0x00008483 0x00008685 0x00008887 ……序列，而实际输出为0x00008281 0x00008685 ……序列，产生数据丢失。

**原因分析：**在布线时，未考虑到按半字或按字节访问数据存储器的情况，而仅仅使用了一个数据存储器，默认为按字访问。

**解决方案：**利用之前在计算机组成原理实验中完成的可以按字、半字、字节访问的存储器兼容LHU指令，具体实现可见图3.3。

## 实验进度

表.课程设计进度表

| 时间 | 进度 |
| --- | --- |
| 第一天 | 阅读课程设计任务书和RISC-V指令手册。 |
| 第二天 | 完成数据通路，实现单周期CPU并且通过测试。 |
| 第三天 | 阅读指令流水线内容，开始尝试搭建理想流水线CPU。 |
| 第四天 | 重写流水接口部件，调试理想流水线CPU并通过测试。 |
| 第五天 | 阅读气泡流水线对于冲突的处理方法，尝试气泡流水线CPU。 |
| 第六天 | 重写气泡逻辑部件，完成气泡流水线并通过测试。 |
| 第七天 | 构建重定向逻辑，完成重定向流水线，通过测试。 |
| 第八天 | 完成单级中断并通过测试，尝试多级中断。 |
| 第九天 | 调试多级中断的bug，通过测试。开始构建BTB表。 |
| 第十天 | 解决分支预测中的诸多bug，完成分支预测功能。 |

# 团队任务

## 选题介绍

本团队以Risc-V CPU为基础，结合LCD Video组件，实现简易的数独游戏，其中包含数独游戏的显示、填写功能。在查看往届团队的选题时，发现选题多有重复，而我们又曾在程序综合课程设计中设计过数独游戏，因而确定了数独游戏的选题。

## 任务分工

我主要负责电路逻辑，刘鉴之主要负责代码编写。具体分工如表5.1所示。

表.团队任务分工

| 徐瑞达 | 刘鉴之 |
| --- | --- |
| * 完成主要电路部分 * 录制Logisim演示视频 * 完成PPT制作 | * 编写数独C语言代码并汇编为Risc-V代码 * 修改Risc-V CPU以支持上述代码所需要的指令 * 编写代码生成数字字库文件、数据存储器文件 * 录制成果汇报视频 |

## 问题与解决方案

### 如何产生数独

**问题说明：**当使用程序综合课程设计中的Las Vegas算法+挖洞法生成数独时，由于存在大量递归过程，转换为Risc-V代码运行后会发生堆栈溢出，不得不弃用。

**解决方案：**最终不得不放弃挖洞法生成数独，转而采用生成随机数选取已有数独的方案。

如代码块5.1所示，rand函数根据seed函数产生的随机数种子产生随机数，确保每次产生随机数的随机性。

在Risc-V中，可以使用rdtime指令取time寄存器作为随机数种子，然而rdtime指令将被汇编为csrrs指令，需要访问CSR寄存器，因此只能采用硬件支持的方法。

在Logisim平台实现CPU时，当执行至csrrs指令时，将Logisim中的随机数寄存器产生的随机数送入a0寄存器作为随机数种子。

代码块5.1 产生数独的Risc-V代码



### 如何绘制数字

**问题说明：**使用LED数码管显示每个宫格内的数字时体验性差，且不具有挑战性；如果使用LCD Video显示每位数字时，如何送入数据？

**解决方案：** 如图5.1所示，使用9个LCD Video组件分别绘制每个宫格内的数字，每个LCD Video绘制9个格子和分隔线；初盘中的数字将被绘制为白色，而用户输入的数字将被绘制为蓝色。通过在ROM中分别存储每个数字和分隔线的点阵信息，使用到时依次输出至对应LCD Video控件即可。

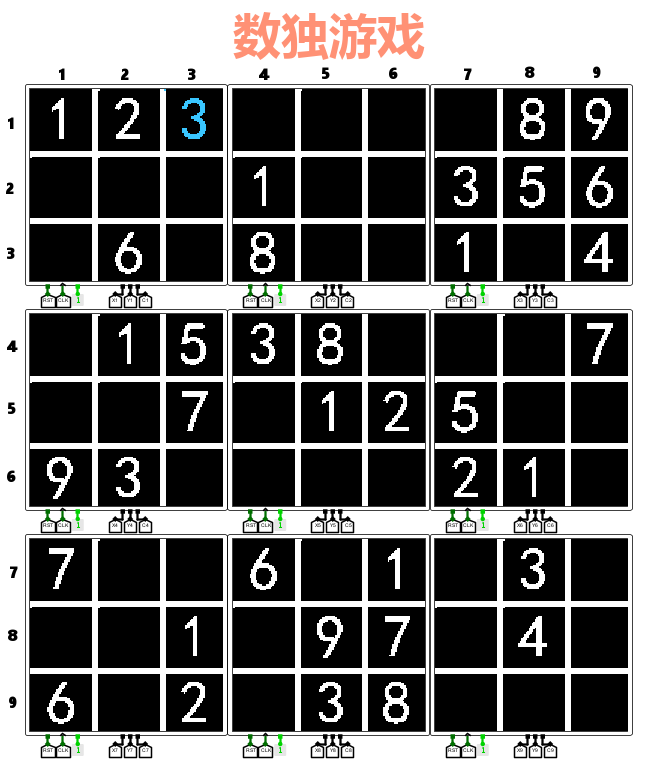


图5.1绘制数独的解决方案

### 如何向LCD Video送入数据

**问题说明：**如何配合时钟向LCD Video送入数据？如何将数独初盘和用户输入的数字绘制为不同颜色？如何在用户修改数字时刷新显示？

**解决方案：**经过团队商讨后，决定将“数字点阵库”封装为单个部件，并封装“LCD数据源”部件。在LCD数据源部件内，通过逐个输出9个格子内的每个格子对应的点阵序列，达成在LCD Video内逐个绘制格子的目的。同时，由是否选中的信号决定“数字点阵库”输出的颜色常量值。在用户修改数字时，通过使用覆盖整个格子的点阵库并送入黑色以达到清空的目的，然后送入用户填入的数字以刷新显示。

### 如何判别坐标

**问题说明：**如何在游戏初始化时，聚焦于首个空格？如何根据当前坐标，处理上下左右移动事件，聚焦于合适的空格？

**解决方案：**图5.2是部分坐标判别逻辑的电路实现图。以NextLeft为例，在初盘寄存器电路中，将当前X坐标与该行中所有空格的坐标进行比较，并送入优先编码器中生成符合要求空格的坐标；在其余电路中，还实现了当前坐标的左侧无其他空格时，跳转到上一行末空的功能。依照NextLeft信号的生成逻辑，可以得到NextRight、NextUp、NextDown以及首个空格坐标信号的生成逻辑。

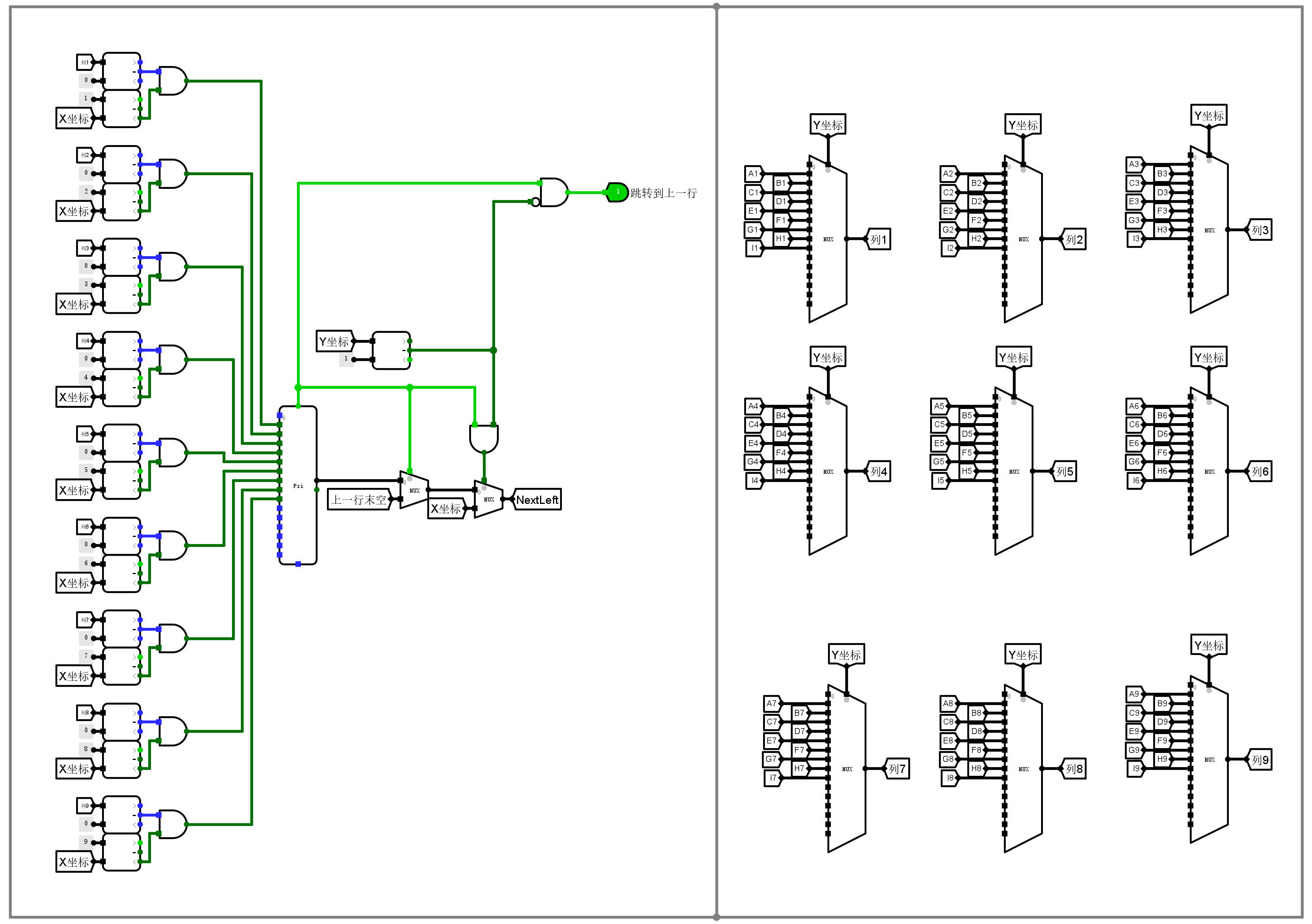


图5.2部分坐标判别逻辑实现

## 版本提交历史

表5.2版本提交历史

| 提交时间 | 提交者 | 提交内容 |
| --- | --- | --- |
| 2022-09-17 00:07 | 徐瑞达 | First Commit |
| 2022-09-17 15:49 | 刘鉴之 | 完成选择数独代码部分 |
| 2022-09-17 18:06 | 徐瑞达 | 完成数独初盘、终盘寄存器 |
| 2022-09-17 21:22 | 刘鉴之 | 完成对Risc-V CPU的修改 |
| 2022-09-17 22:00 | 徐瑞达 | 完成LCD数据源体系构建 |
| 2022-09-18 12:13 | 刘鉴之 | 完成字库文件生成代码部分 |
| 2022-09-19 00:03 | 徐瑞达 | 初步达成显示数独功能，构建当前格局寄存器 |
| 2022-09-19 23:39 | 徐瑞达 | 完成基本功能，可填写及判断是否填写正确 |
| 2022-09-20 12:22 | 徐瑞达 | 完成更加方便的坐标选择逻辑 |
| 2022-09-20 16:30 | 团队 | 联调测试，修改若干小bug，完成项目 |

# 设计总结与心得

## 课设总结

本次课程设计主要作了如下几点工作：

1. 完成方案总结：设计了单周期CPU、理想流水线CPU、气泡流水线CPU、重定向流水线CPU、支持单级中断的单周期CPU、支持多级中断的单周期CPU、支持动态分支预测机制的重定向流水线CPU。
2. 功能总结：单周期CPU、气泡流水线CPU、重定向流水线CPU、支持动态分支预测机制的重定向流水线CPU都实现了包括CCAB指令的指令集；气泡流水线通过插入气泡的方式实现了处理数据冲突和分支冲突；重定向流水线通过重定向数据的方式减少了插入气泡数；支持动态分支预测机制的重定向流水线通过预测分支跳转的方式进一步减少了插入气泡数；支持多级中断的单周期CPU实现了三级嵌套外部中断。
3. 团队任务总结：实现了使用随机数随机选择数独游戏的功能，实现了在LCD Video中绘制数独游戏的功能，实现了通过键盘输入选择空格子的功能，实现了动态绘制用户输入数字的功能，实现了数独填写正确时响起提示音的功能。

## 课设心得

本次课程设计是对上学期组成原理实验的拔高，从个人任务的完成到团队任务的选题、设计与完成，总共经历了大约半个月的时间。

课程设计刚开始时，由于对整个实验及其难度没有一个大致的把握，有种畏怯心理，导致看了一天的任务书和相关文档，也没有头绪。在对照着数据通路进行布线连接后，才开始慢慢进入状态，第二天便完成了单周期CPU的设计。

接着设计理想流水线的过程也不是很顺利，在使用了上学期组成原理实验的乘法流水线接口部件并且仔细阅读课本掌握流水线的原理后，才在第四天完成理想流水线的设计。然后是气泡流水线的设计，由于先前对数据冲突和分支冲突的概念了解不透彻，只能迷迷糊糊地按照课本上讲述的逻辑设计。在设计重定向流水线时，我方才认识到，理想流水线没有处理冲突，气泡流水线采用插入空气泡的方式处理冲突，而重定向流水线则是在气泡流水线的基础上改进了数据冲突的处理方式。对于最后具有一定难度的动态分支预测机制，则是在重定向流水线的基础上改进了分支冲突的处理方式。从理想流水线到气泡流水线，从重定向流水线再到支持动态分支预测机制的流水线，逐步提高CPU的运行效率，在逐步提高难度的过程中也获得了更强烈的自豪感。

在团队任务中，我的体悟也更为深刻。从确定选题后的毫无头绪到数独格局的成功读取，从LCD成功显示第一个数字到9个LCD组成的阵列成功显示整个数独，从成功完成键盘事件的处理到最终数独游戏的顺利完成，我都无法想象那些复杂部件的设计是我们在几天内完成的。虽然受限于LCD的绘制速度，数独游戏的绘制需要漫长的等待，但是当我想到我可以使用自己画出的电路玩一盘数独游戏时，心情是无比激动的，澎湃的自豪感淹没了这半个月来遇到的所有崎岖坎坷。

最后十分感谢课程组老师们的帮助，也感谢团队伙伴刘鉴之同学的支持与帮助，我相信这是我大学生涯中最有成就感的事情之一。

# 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: 徐瑞达** |