***2022***



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

目 录

[1 课程设计概述 3](#_Toc69950807)

[1.1 课设目的 3](#_Toc69950808)

[1.2 设计任务 3](#_Toc69950809)

[1.3 设计要求 3](#_Toc69950810)

[1.4 技术指标 4](#_Toc69950811)

[2 总体方案设计 6](#_Toc69950812)

[2.1 单周期CPU设计 6](#_Toc69950813)

[2.2 中断机制设计 10](#_Toc69950814)

[2.3 流水CPU设计 12](#_Toc69950815)

[2.4 气泡式流水线设计 13](#_Toc69950816)

[2.5 数据转发流水线设计 14](#_Toc69950817)

[2.6 动态分支预测机制设计 16](#_Toc69950818)

[2.7 数据转发流水线中断设计 16](#_Toc69950818)

[3 详细设计与实现 17](#_Toc69950819)

[3.1 单周期CPU 实现 17](#_Toc69950820)

[3.2 中断机制实现 20](#_Toc69950821)

[3.3 流水CPU实现 23](#_Toc69950822)

[3.4 气泡式流水线实现 26](#_Toc69950823)

[3.5 数据转发流水线实现 27](#_Toc69950824)

[4 实验过程与调试 30](#_Toc69950825)

[4.1 测试用例和功能测试 30](#_Toc69950826)

[4.2 性能分析 33](#_Toc69950827)

[4.3 主要故障与调试 34](#_Toc69950828)

[4.4 实验进度 38](#_Toc69950830)

[5 设计总结与心得 39](#_Toc69950831)

[5.1 课设总结 39](#_Toc69950832)

[5.2 课设心得 39](#_Toc69950833)

[参考文献 41](#_Toc69950834)

[附录 团队任务 42](#_Toc69950834)

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

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 | 算术右移 |
| 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 | 转移并连接 |
| 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等待Go按键暂停 |
| 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 |
| 25 | XOR | 异或 | 扩展指令1 |
| 26 | AUIPC | PC加立即数 | 扩展指令2 |
| 27 | LH | 半字加载 | 扩展指令3 |
| 28 | BGE | 大于等于时分支 | 扩展指令4 |

# 总体方案设计

## 单周期CPU设计

本次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。在本次课程设计，需要使用LOGISIM来创建一个32-位5段流水CPU， 该CPU运行的是标准RISC-V指令集的子集。

总体结构图如图 2.1所示。



图 2.1 总体结构图

### 主要功能部件

#### 程序计数器PC

程序计算器的设计思路是每个时钟周期，计数器都计算相应的pc，每次pc送出后，在时钟上跳沿到来时实现PC自动加4；当停机指令有效时，该计数器停止计数。可以采用寄存器实现，其输出作为指令存储器的地址输入，数据宽度为32。

#### 指令存储器IM

指令存储器用于单独存放程序指令，输入为 32 位的字节地址，输出为 32 位的 RISC-V32指令字。经过程序计数器的pc为所取指令的地址，将该地址传入指令存储器中，经过取址，将得到的指令送出，完成取址过程。

#### 运算器

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 |
| Shamt | 输出 | 5 | 偏移位数 |
| Equal | 输出 | 1 | Equal=(x==y)?1:0, 对所有操作有效 |

#### 寄存器堆RF

寄存器堆是MIPS处理器中32个通用寄存器的集合，CPU 通过一定的接口可以对这些寄存器进行读写访问。在 R1#、 R2# 两个输入端口给出两个 5 位的寄存器编号，其内部两个独立的多路选择器就会并发选择对应寄存器的 32 位值并传输到数据输出端口 R1 和 R2，这里寄存器堆读逻辑是组合逻辑。W# 为 5 位写寄存器编号输入端口， WD 为 32 位的写数据端口。当写使能控制信号 WE 为 1，时钟上跳沿到来时，寄存器堆将 WD 端的数据写入编号为 W# 的寄存器中，写入部分需要时钟配合，是时序逻辑电路。

### 数据通路的设计

数据通路的设计如图 2.2 数据通路图所示，pc寄存器将pc值送到指令存储器，指令存储器将指令取出送入寄存器堆，寄存器堆将寄存器值取出送入ALU操作，计算完成后将结果送入数据存储器，数据存储器写回。

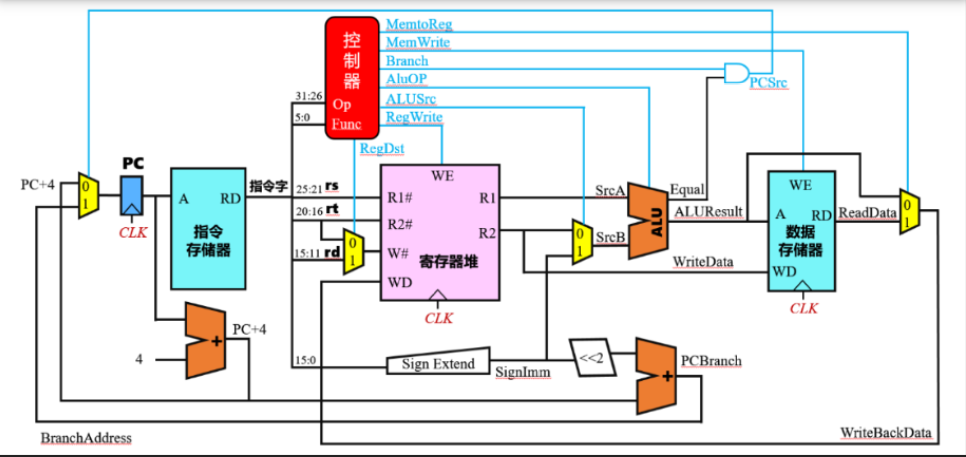


图 2.2 数据通路图

根据图2.3 RISC-V指令可知，有6种指令。在指令送入寄存器堆时，我们主要考虑3种指令，分别为R型、J型I型。根据图 2.3 RISC-V指令的格式，我们可构建出符合不同格式的数据通路，其中6-10为立即数，11-15为rd，16-20为rt，21-25为rs，26-31为OP，0-5为Func。0-15、0-25为立即数。

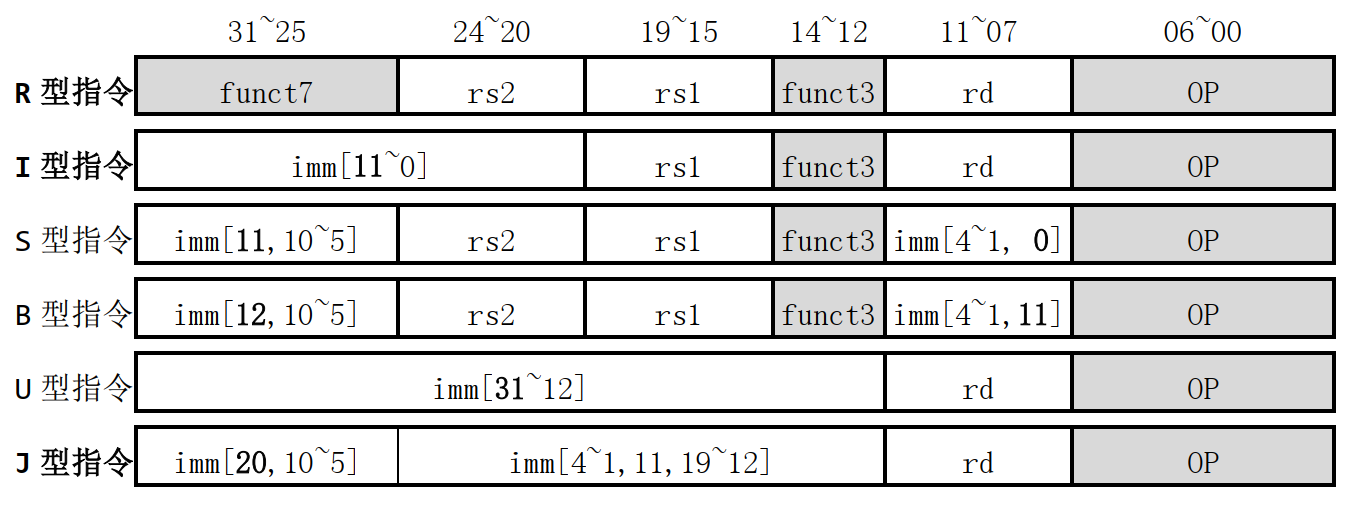


图 2.3 RISC-V指令的格式

### 控制器的设计

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

| 控制信号 | 信号说明 | 产生条件（信号为1） |
| --- | --- | --- |
| RegWrite | 寄存器写使能 | 寄存器写回信号 |
| MemWrite | 写内存控制信号 | sw指令 未单独设置MemRead信号 |
| AluOP | 运算器操作控制符（4位） | R型指令根据Func选择 |
| MemToReg | 寄存器写入数据来自存储器 | lw指令 |
| S\_Type | S型指令译码信号 | S型指令 |
| AluSrcB | 运算器B输入选择 | lw指令，sw指令，立即数运算类指令 |
| JALR | JAL指令译码信号 | JALR指令 |
| JAL | JAL指令译码信号 | JAL指令 选择寄存器写回编号，写回值 |
| Beq | Beq指令译码信号 | Beq指令，用于有条件分支控制 |
| Bne | Bne指令译码信号 | Bne指令，用于有条件分支控制 |
| ecall | ecall指令译码信号 | 根据a7的值，决定是停机还是输出 |

表 2.2主控制器控制信号的作用说明

由于 Logisim 自动生成电路功能输出引脚数目有限，为了方便自动生成电路，这里将 AluOP 输出利用运算器控制器单独输出，其他控制信号则用控制信号生成电路输出，实验需要分别设计实现这两个电路。

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。查看 RISC-V 指令码表（HUST） .xlsx，逐条填写每条指令 OpCode、 Funct7， Funct3 字段以及对应的 AluOp 和其他控制信号值。填写完毕后，打开控制信号表达式生成表，可以得到所有控制信号的逻辑表达式，利用逻辑表达式自动生成运算器控制器和控制信号生成电路即可。该控制信号表的框架如图2.4所示。

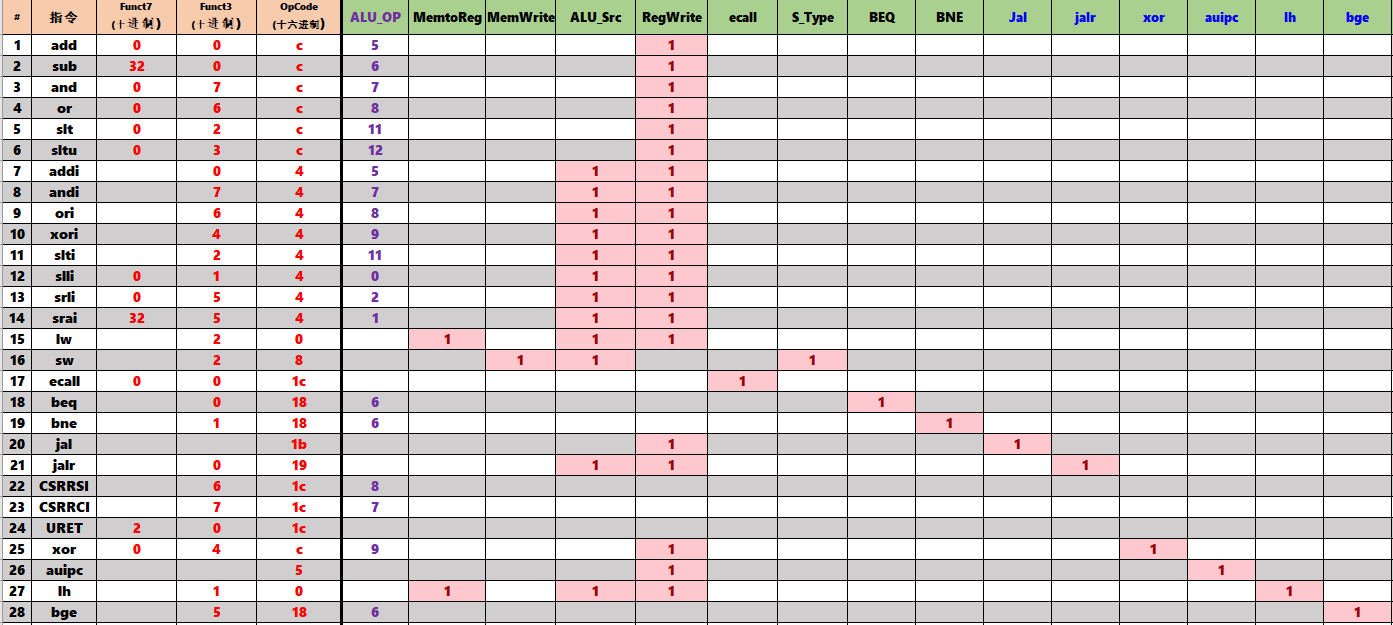


图2.4 控制信号表

## 中断机制设计

### 总体设计

RISC-V中断控制一般通过立即数置位控制状态寄存器CSRRSI，立即数清除控 制状态寄存器CSRRCI，中断使能位IE、中断请求寄存器IR和优先编码器组成。流程图如图2.5所示。

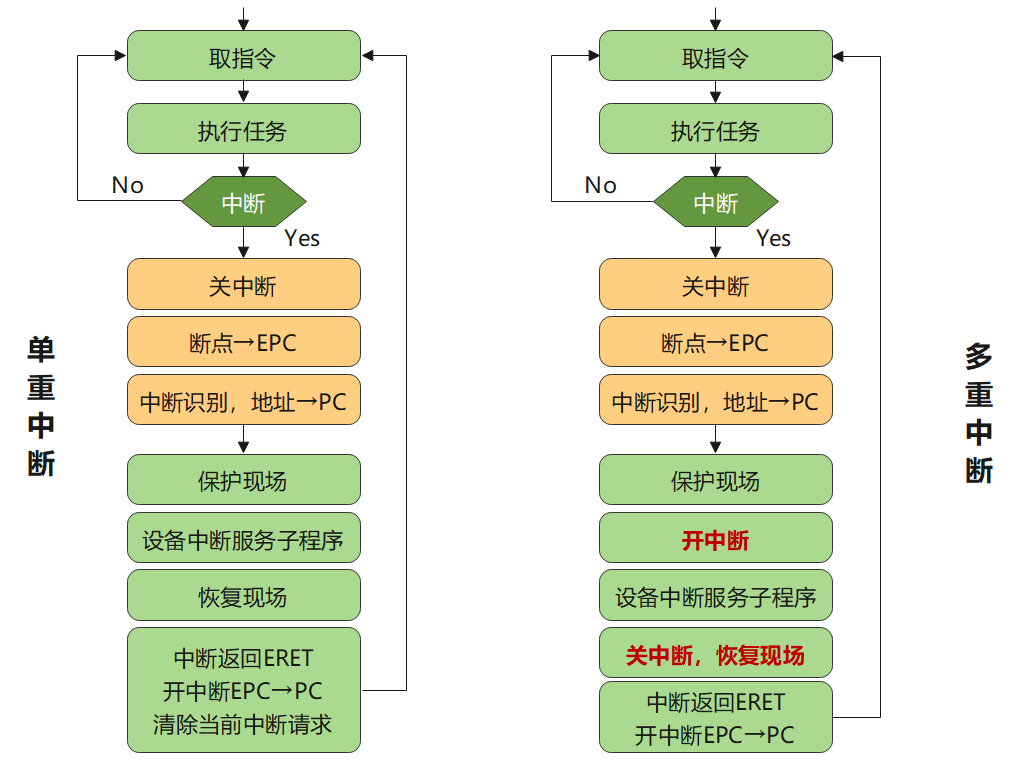


图2.5 中断流程图

### 硬件设计

对于每个中断寄存器，都需要经过按键中断外围电路，该电路已在慕课中给出。经过该电路处理得到的信号，与中断屏蔽字进行与操作，得到的信号送入优先编码器中。优先编码器进行选择之后将得到的信号送入IR寄存器，与IE操作后得到中断信号。中断硬件设计如图2.6所示。

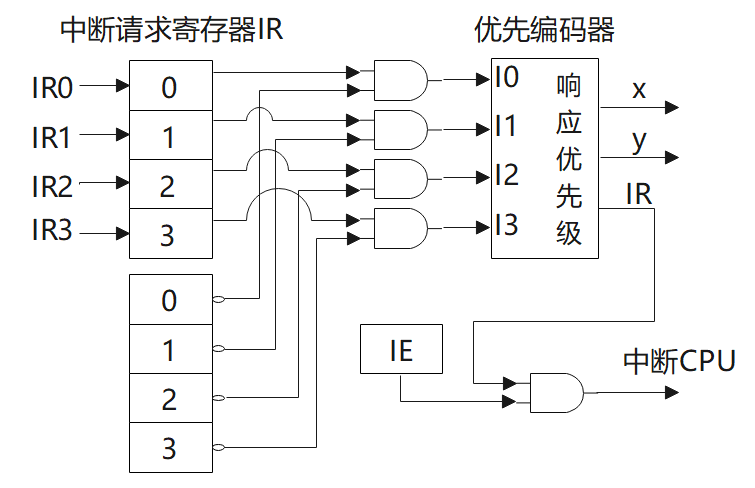


图2.6 中断硬件设计

在测试程序中找出中断地址，这些地址为对应的中断信号需要跳转的中断地址，需要通过优先编码器产生的编码号进行选择。

对于中断返回信号，设计一个寄存器，当中断信号有效时，若遇上了中断返回信号，就像对应的中断信号清零，程序返回原来的跳转点，该点的pc存储在寄存器EPC中。

### 软件设计

在软件设计时，采取指令支持，如图2.7所示。尤其在多级中断时要加入开中断和关中断信号。除此之外，还需要设计堆栈寄存器，来对应指令中多级中断需要使用到的堆栈操作。其余设计与单级中断一致。

对于堆栈的设计，每一层采用多路选择器和寄存器实现。多路选择器的0对于上一层数据，1对于下一层数据，一次完成堆栈的存取操作。

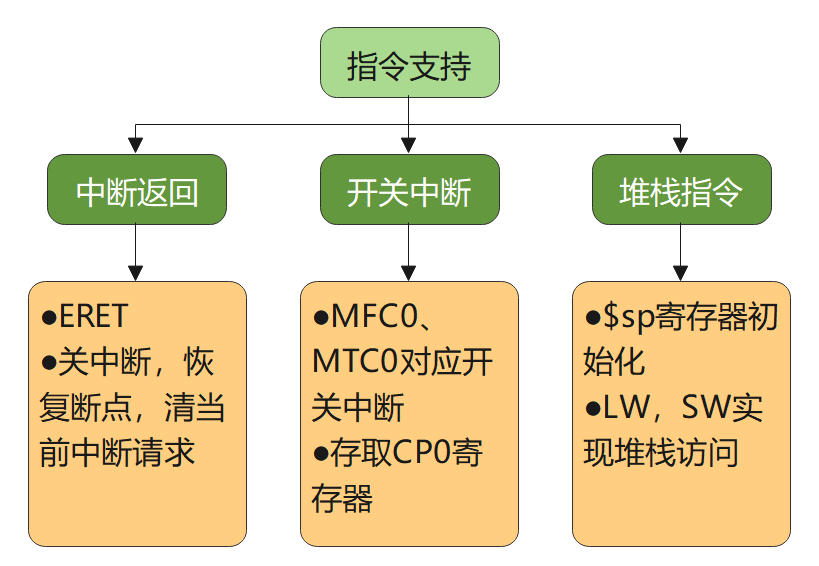


图2.7 中断软件设计

## 流水CPU设计

### 总体设计

将指令过程分成5个阶段IF、ID、EX、MEM、WB，取指令（IF）：负责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进行译码，同时从寄存器堆中取操作数。指令执行（EX）：执行运算操作或计算地址。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。不同阶段之间设置缓冲接口部件，构建各阶段之间的接口部件寄存器，后续部件对数据的加工处理依赖于前阶段传递过来的信息。

计算机指令流水线存在大量的指令相关和数据依赖，通常会引起流水线的阻塞 /暂停（Stall）。所谓指令相关，是指在指令流水线中，如果某指令的某个阶段必须等到它前面的某条指令的某个阶段完成后才能开始。

设计流水线时需要消除这些流水线冲突，包括数据相关、结构相关、控制相关。

### 流水接口部件设计

流水线接口处通过将上一阶段送进了的信号存储起来，在下一个时钟周期送入下一阶段去。通过流水寄存器传递的不仅是当前指令、当前阶段待加工的数据，还需要向后段传递数据加工所需的操作控制信号。这些控制信号应与各功能段处理的指令同步，每一个操作控制信号均只在某一功能段内使用一次，操作控制信号使用完毕后就不再向后段继续传递，因此ID 段产生的控制信号在从前到后传递的过程中数目逐渐减少。

为了简便设计，将ID阶段硬布线产生的信号在此后每一个接口都留了传递的位置，以便调试和设计。

### 理想流水线设计

理想流水线在单周期CPU的基础上改造，不需要考虑流水线中的数据冲突问题，只需要将指令分为五段操作即可，在每两个阶段之间加上接口，并判断每个信号应该在哪部分使用。因为理想的流水线是所有待加工对象均需要通过相同的阶段，不同阶段之间无共享资源，且各段传输时间延迟一致，进入流水线的对象也不应受其他功能段的影响。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。其中分支指令在EX段完成。通路的逻辑如图2.8所示。

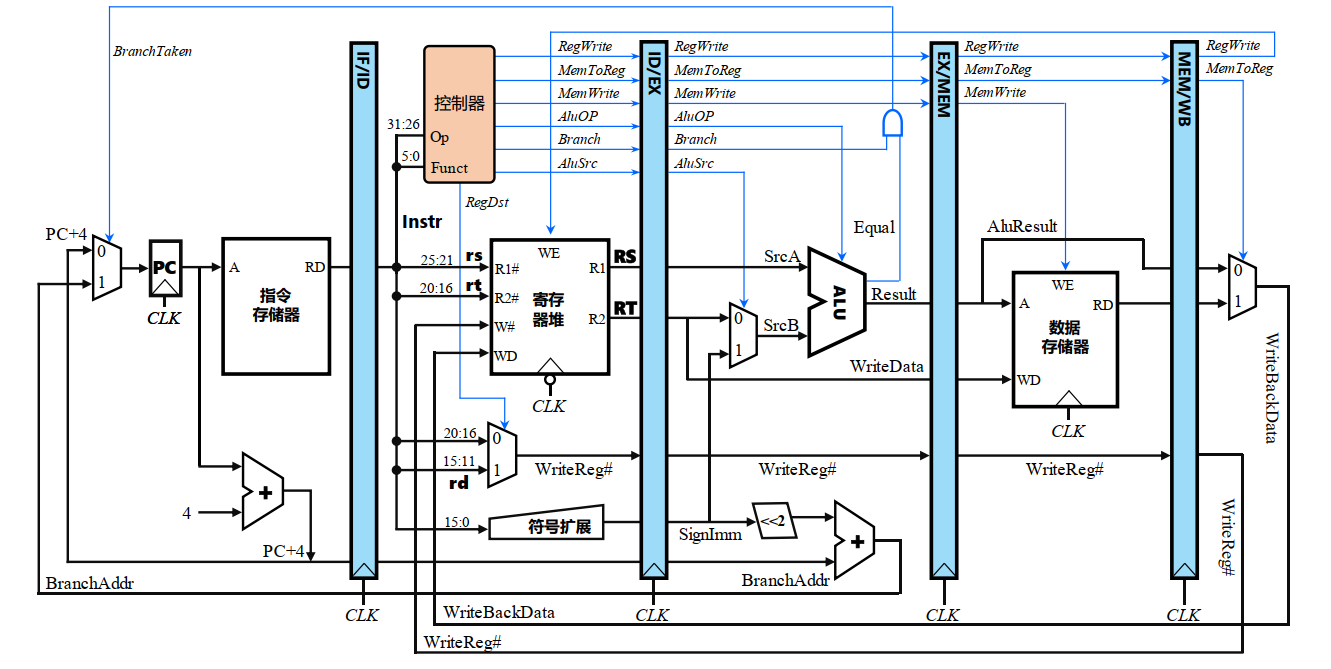


图2.8 理想流水线通路

## 气泡式流水线设计

消除控制冲突：分支指令会引起控制冲突，要解决控制冲突，在执行程序分支跳转时必须清除流水线中分支指令后续的若干条误取指令。消除控制冲突只需要在实际分支跳转时，将分支指令所在功能段左侧所有即将存放误取指令的流水寄存器同步清零即可；因为在EX段执行分支指令，需要清除 IF/ID、ID/EX 流水寄存器。

消除数据冲突：ID 段与WB 段冲突，但由于 RISC-V 寄存器堆的读写逻辑是完全独立的逻辑，读写地址和数据均通过不同的端口进入，读写逻辑可以并发操作，所以只需要将寄存器堆改成下降沿写入即可；ID段和MEM段冲突，IF、ID暂停等待数据写回、EX段插入气泡；ID段和EX段冲突，IF、ID暂停等待数据写回、EX段插入气泡。

通路图如图2.9所示，根据相关数据处理逻辑和通路图完成对应的硬件设计。

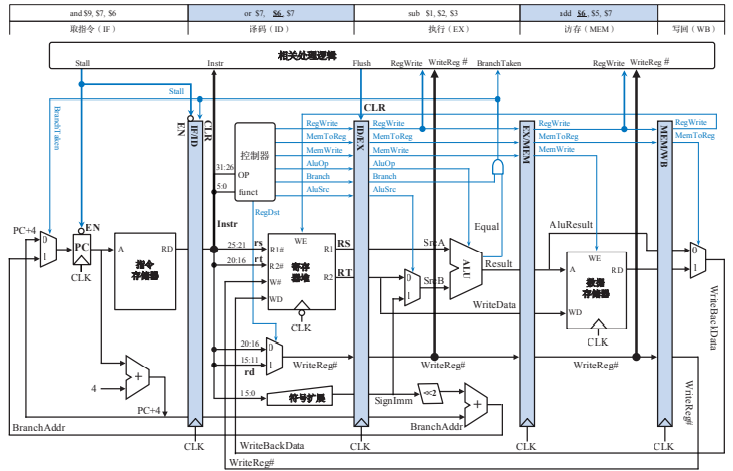


图2.9 气泡流水线通路

## 数据转发流水线设计

在EX段对数据进行重定向，加入多路选择器进行判断。选择信号设为forward1和forward2。其逻辑为如图2.10所示，forward2同理。

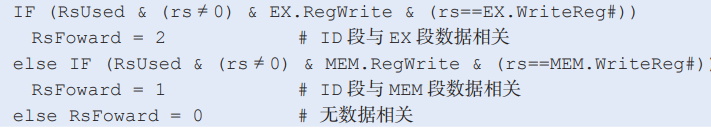


图2.10 重定向选择forward1逻辑

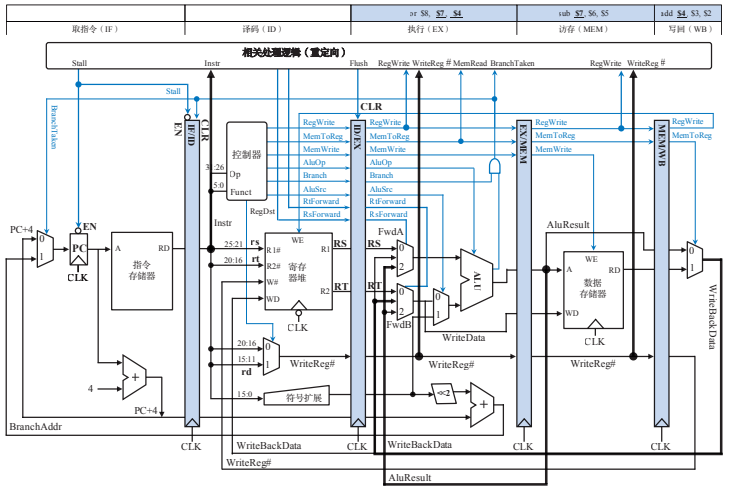


图2.11 重定向流水线通路

以对于 Load-Use 相关，不能采用重定向方式解决数据冲突，必须在发生 Load-Use 相关的两条相邻指令之间强制插入一个气泡以消除这种相关，插入气泡在 ID 段实现。Load-Use检测：LoadUse = RsUsed & (rs ≠ 0) & EX.MemRead & (rs==EX.WriteReg#)+ RtUsed & (rt ≠ 0) & EX.MemRead & (rt==EX.WriteReg#)。

当发生 Load-Use 相关时，需要暂停 IF、ID 段指令的执行，并在 EX 段中插入气泡，需要控 制 PC 使能端 EN、IF/ID 使能端 EN、ID/EX 清零端 CLR；而 EX 段执行分支指令时会清空 ID 段、 EX 段中的误取指令，会使用 IF/ID 清零端 CLR、ID/EX 清零端 CLR。

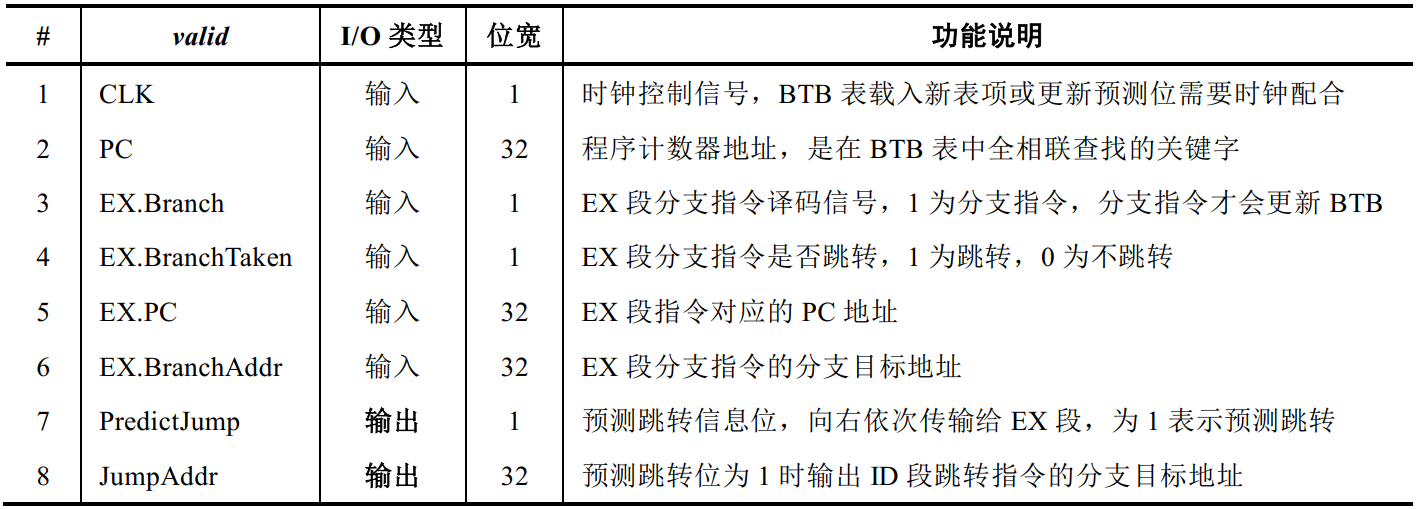
## 动态分支预测机制设计

动态分支预测策略采用分支预测缓冲器， 用于存放分支指令的分支跳转历史统计信息。BTB表每个表项主要包括valid位、分支指令地址，分支目标地址，历史跳转信息描述位（预测状态位）、置换标记五项。

动态分支预测逻辑采用硬件实现，内部是一个全相联的 cache 结构，其主要输入输

出引脚及功能说明如表2.3所示。

表 2.3 分支历史表BTB逻辑引脚说明



将响应的硬件逻辑加入RISC-V指令流水线中就得到图2.12所示的支持动态分支预测的五段流水线，新的流水线数据通路中 IF 段增加了一个 PredictJump 信号。由于 EX 段只有检测到分支预测错误才会清空流水线中的误取指令，所以将预测失败信号 PredictErr 接入相关处理逻辑中原 BranchTaken 引脚即可复用原电路，其他逻辑并没有任何变化。

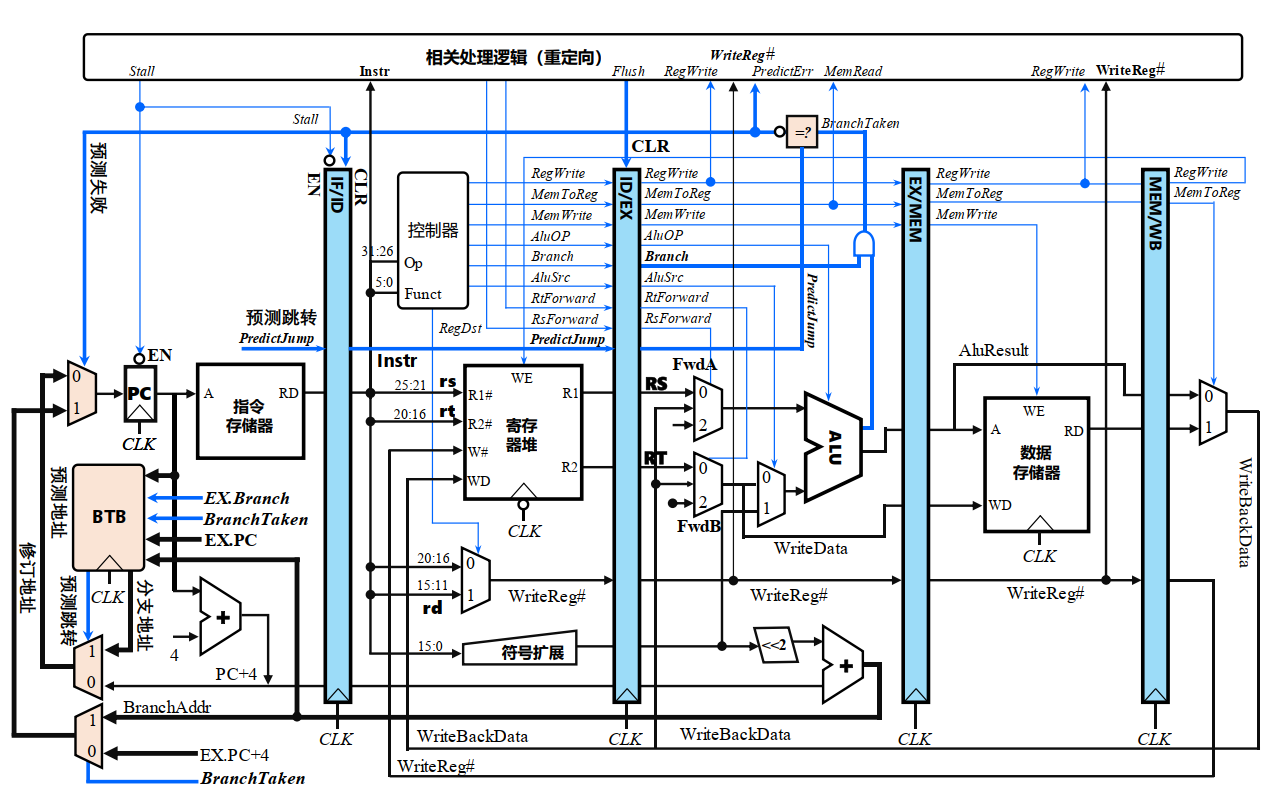


图2.12 动态分支预测通路

## 数据转发流水线中断设计

该电路只需要将重定向流水线和单级中断的部分电路结合，然后将重定向电路中对应的插入气泡和清零的部分根据中断信号和中断返回信号进行修改即可。当然，对用的PC前面的选择也应加上相应的部分。

# 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）

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。

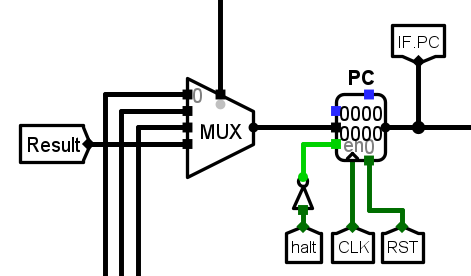


图 3.1程序计数器（PC）

1. 指令存储器（IM）

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。

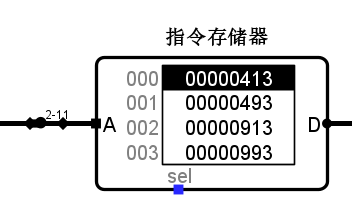


图 3.2指令存储器（IM）

### 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 | | Tube |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 立即数 | 5 |  |  |  |
| ADDIU | PC+4 | PC | rs |  | rt | alu | r1 | 立即数 | 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 | 立即数 | 7 |  |  |  |
| SLL | PC+4 | PC |  | rt | rd | alu | r2 | 立即数 | 0 |  |  |  |
| SRA | PC+4 | PC |  | rt | rd | alu | r2 | 立即数 | 1 |  |  |  |
| SRL | PC+4 | PC |  | rt | rd | alu | r2 | 立即数 | 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 | 立即数 | 8 |  |  |  |
| NOR | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 10 |  |  |  |

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，连接完成后的电路图如图 3.3所示。

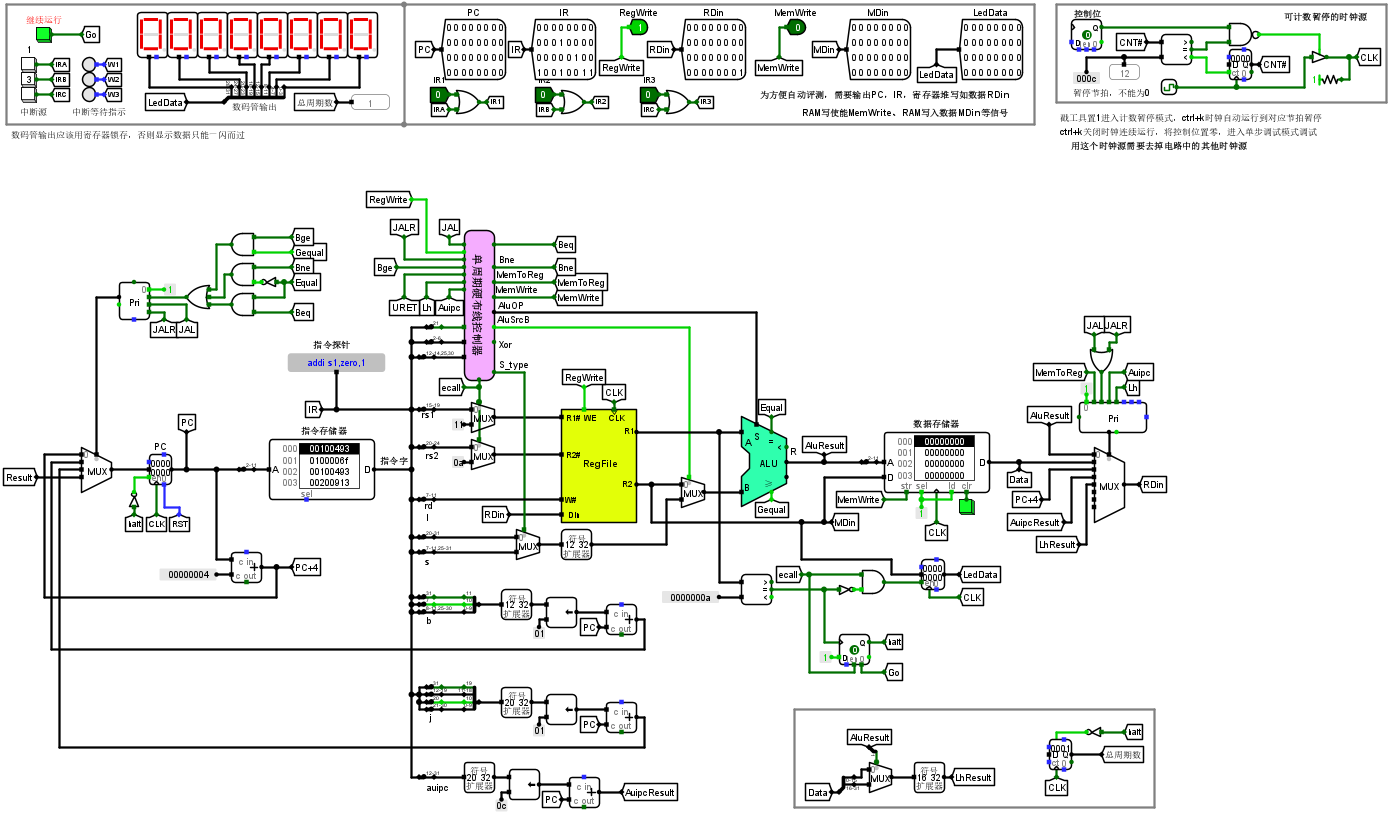


图 3.3 单总线CPU电路图

### 控制器的实现

根据总体方案设计中控制器的设计那一小节的相关内容，分别在Logism上进行主控制器、Branch控制器、SYSCALL控制器的具体实现。主控制器信号如表 3.2所示。

表 3.2主控制器控制信号

| 指令 | R | RW | WE | X | EXT | Y | ALUop | MemWrite | MemRead | Din | Branch | SYSCALL |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|
| ADD | 00 | 00 | 1 | 0 | 0 | 00 | 0101 | 0 | 0 | 00 | 00 | 0 |
| ADDI | 00 | 10 | 1 | 0 | 0 | 10 | 0101 | 0 | 0 | 00 | 00 | 0 |
| ADDIU | 00 | 10 | 1 | 0 | 0 | 10 | 0101 | 0 | 0 | 00 | 00 | 0 |
| ADDU | 00 | 00 | 1 | 0 | 0 | 00 | 0101 | 0 | 0 | 00 | 00 | 0 |
| AND | 00 | 00 | 1 | 0 | 0 | 00 | 0111 | 0 | 0 | 00 | 00 | 0 |
| ANDI | 00 | 10 | 1 | 0 | 1 | 10 | 0111 | 0 | 0 | 00 | 00 | 0 |
| SLL | 00 | 00 | 1 | 1 | 0 | 01 | 0000 | 0 | 0 | 00 | 00 | 0 |
| SRA | 00 | 00 | 1 | 1 | 0 | 01 | 0001 | 0 | 0 | 00 | 00 | 0 |
| SRL | 00 | 00 | 1 | 1 | 0 | 01 | 0010 | 0 | 0 | 00 | 00 | 0 |
| SUB | 00 | 00 | 1 | 0 | 0 | 00 | 0110 | 0 | 0 | 00 | 00 | 0 |
| OR | 00 | 00 | 1 | 0 | 0 | 00 | 1000 | 0 | 0 | 00 | 00 | 0 |
| ORI | 00 | 10 | 1 | 0 | 1 | 10 | 1000 | 0 | 0 | 00 | 00 | 0 |
| NOR | 00 | 00 | 1 | 0 | 0 | 00 | 1010 | 0 | 0 | 00 | 00 | 0 |
| LW | 00 | 10 | 1 | 0 | 0 | 10 | 0000 | 0 | 1 | 10 | 00 | 0 |
| SW | 00 | 00 | 0 | 0 | 0 | 10 | 0000 | 1 | 0 | 00 | 00 | 0 |
| BEQ | 00 | 00 | 0 | 0 | 0 | 00 | 0000 | 0 | 0 | 00 | 01 | 0 |
| BNE | 00 | 00 | 0 | 0 | 0 | 00 | 0000 | 0 | 0 | 00 | 01 | 0 |
| SLT | 00 | 00 | 1 | 0 | 0 | 00 | 1011 | 0 | 0 | 00 | 00 | 0 |
| SLTI | 00 | 10 | 1 | 0 | 0 | 10 | 1011 | 0 | 0 | 00 | 00 | 0 |
| SLTU | 00 | 00 | 1 | 0 | 0 | 00 | 1100 | 0 | 0 | 00 | 00 | 0 |
| J | 00 | 00 | 0 | 0 | 0 | 00 | 0000 | 0 | 0 | 00 | 10 | 0 |
| JL | 00 | 01 | 1 | 0 | 0 | 00 | 0000 | 0 | 0 | 01 | 10 | 0 |
| JR | 00 | 00 | 0 | 0 | 0 | 00 | 0000 | 0 | 0 | 00 | 11 | 0 |
| SYSCALL | 11 | 00 | 0 | 0 | 0 | 11 | 0000 | 0 | 0 | 00 | 00 | 1 |

## 中断机制实现

### 中断按键电路

对于每个中断寄存器，都需要经过按键中断外围电路，该电路的设计在mooc中有讲解，完成后的电路图如图 3.4所示。

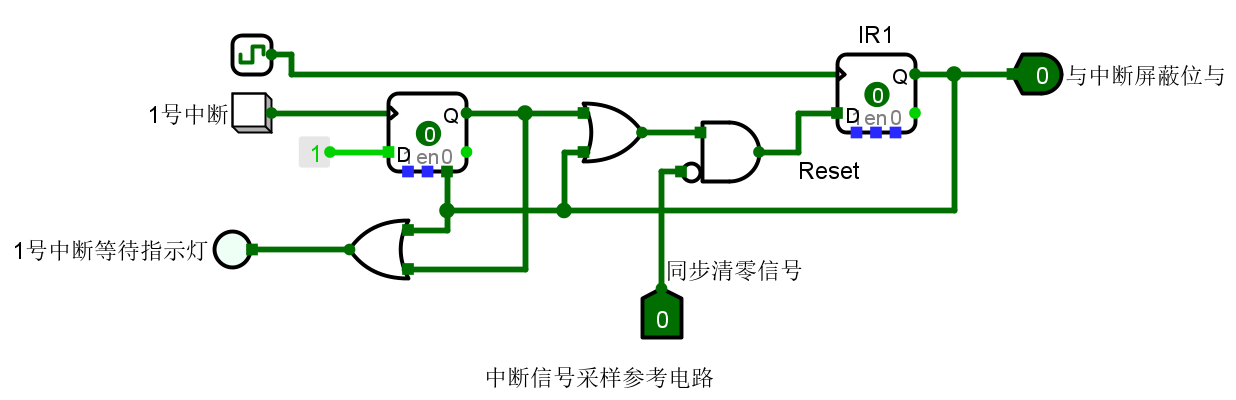


图 3.4 中断按键电路

### 单级中断

通过RARS运行老师所给的测试程序，找到三个中断程序入口地址，得到的结果如图 3.5所示。

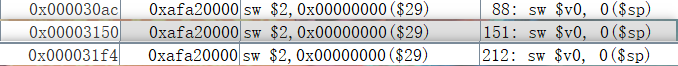


图 3.5 中断程序入口地址

根据上一节的图绘制相关的电路图， 对于中断返回信号，设计一个寄存器，当中断信号有效时，若遇上了中断返回信号，就像对应的中断信号清零，程序返回原来的跳转点，该点的pc存储在寄存器EPC中。得到的电路图如图 3.6所示。

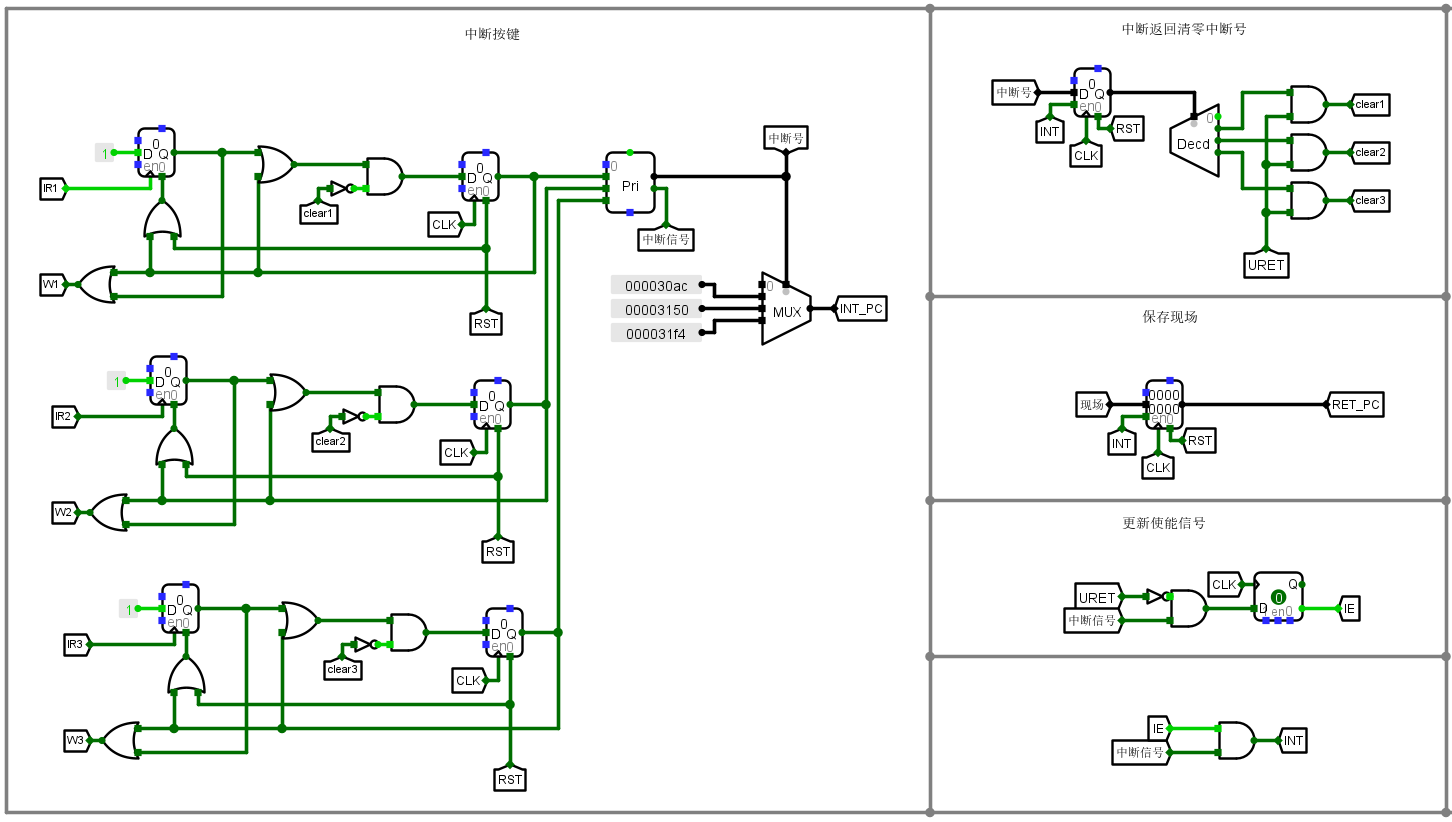


图 3.6 单级中断电路实现

### 堆栈实现

对于多级中断来说，若当前中断的优先度小于即将进来的中断的优先度，当前中断需要立即停止让路给进来的中断，这就意味着要将当前中断跳转时的地址和编号存起来，这时便需要将两者分别压入堆栈中。

对于堆栈，输入为push，输出为pop，同时应该兼具清零和使能信号，以便于多级中断实现。实现完成后电路图如图 3.7所示。

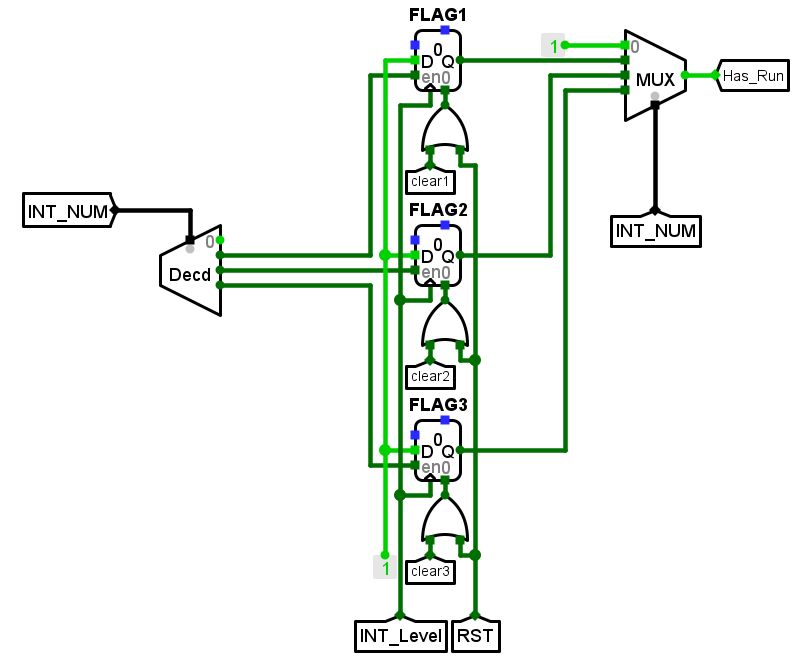


图 3.7 堆栈实现电路

### 多级中断

主电路中使用堆栈将跳转点的pc的和当前中断编号压入堆栈的电路如图 3.8所示。

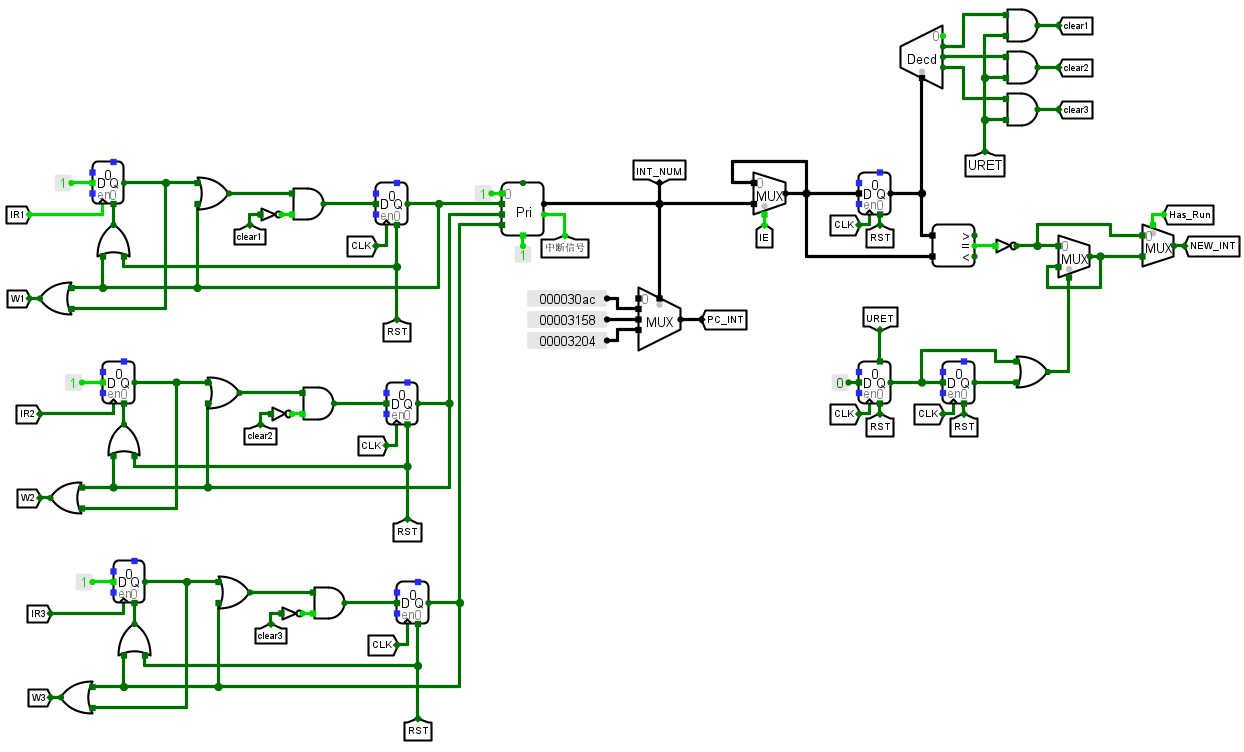


图 3.8 pc和中断编号压入堆栈

对于多级中断的入口地址，与之前的单级中断不一样，而且还需要加上开关中断信号的产生，直接用IR指令进行比较判断即可。

## 流水CPU实现

### 流水接口部件实现

流水线接口处通过多路选择器和寄存器实现。接口处每个传入的信号会先经过二路选择器，二路选择器0对于传入的数据，1对于常量0，选择信号为同步清零信号。经过多路选择器的选择，将信号送入寄存器，寄存器连接统一的使能信号，和统一的时钟信号，再将寄存器的输出值送入输出端。

ID/IF、ID/EX、EX/MEM、MEM/WB寄存器分别如图3.9、图3.10、图3.11、图3.12所示。

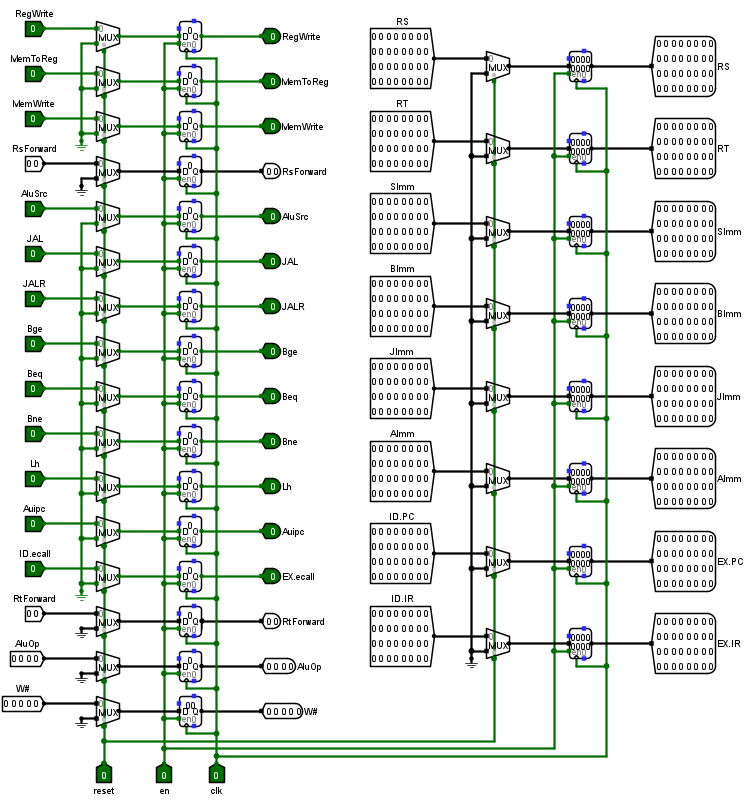


图3.9 ID/EX寄存器

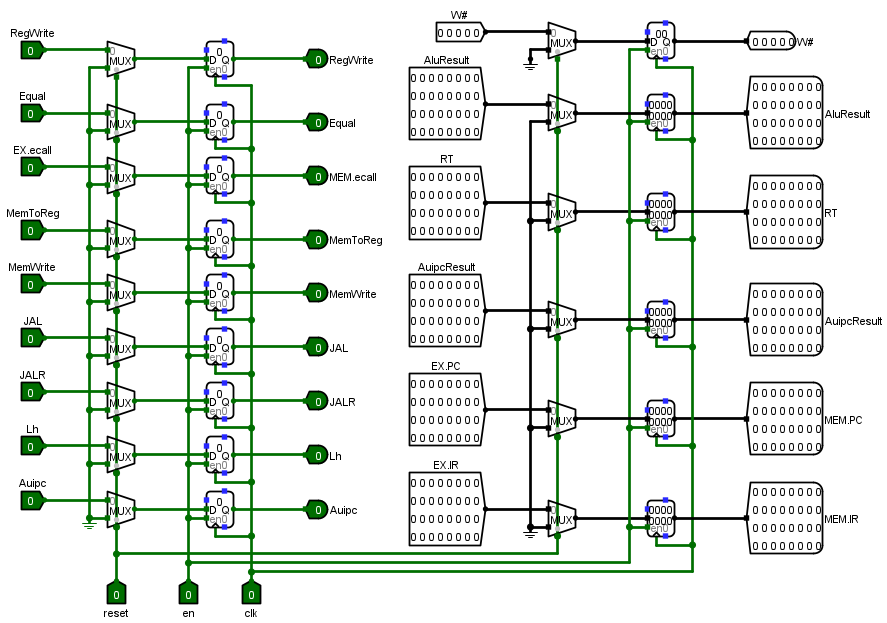


图3.10 EX/MEM寄存器

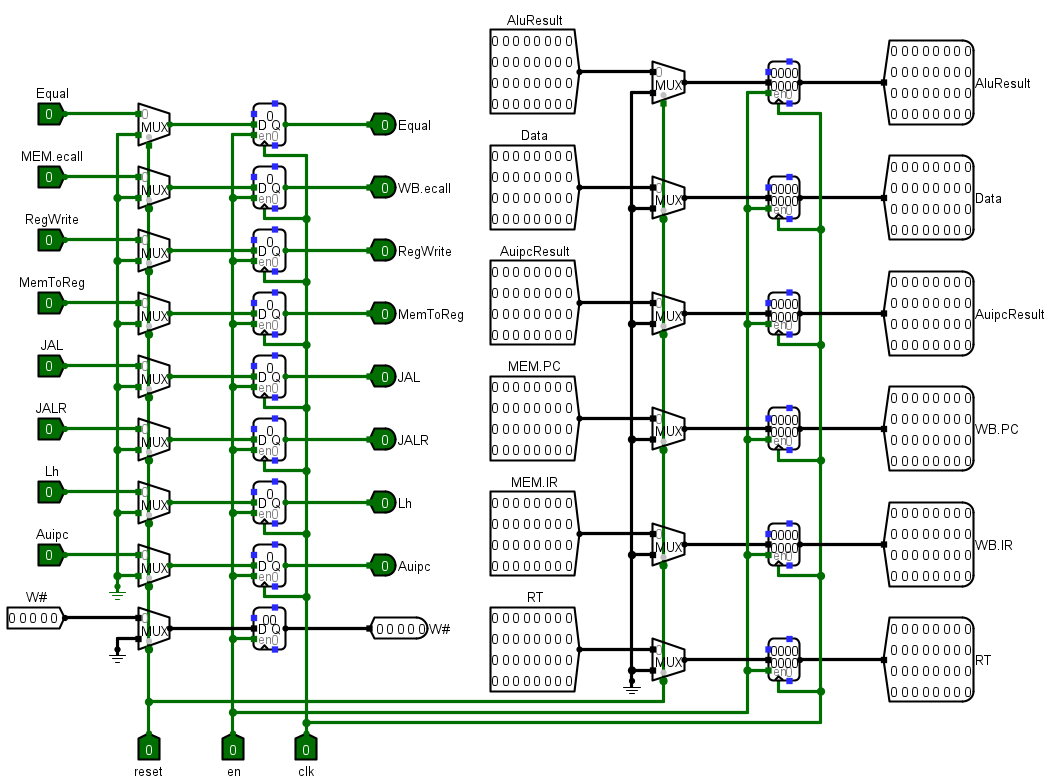


图3.11 MEM/WB寄存器

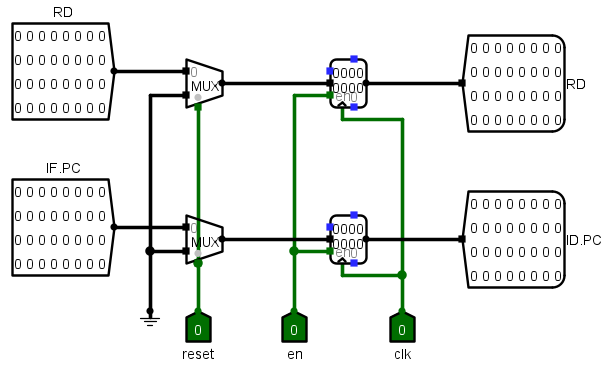


图3.12 IF/ID寄存器

### 理想流水线实现

5 段指令流水线中 ID 段负责指令译码生成操作控制信号，所以操作控制器应该设置在该段中。操作控制器的输入为 IF/ID 流水寄存器锁存指令字中的 OP 和funct 字段，内部为组合逻辑电路；输出为控制信号，其中RegDst信号为 ID 段使用，其他后段使用的控制信号输出到 ID/EX 流水寄存器中， 并依次向后传递。RegWrite信号必须传递至 WB 段后才能反馈到 ID 段的寄存器堆的写入控制端 WE；条件分支译码信号 Branch 也需要传递到 EX 段，与 ALU 运算的标志 equal 信号进行逻辑与 操作后反馈到IF段控制多路选择器进行分支处理。

理想流水线的电路图如图3.13所示。

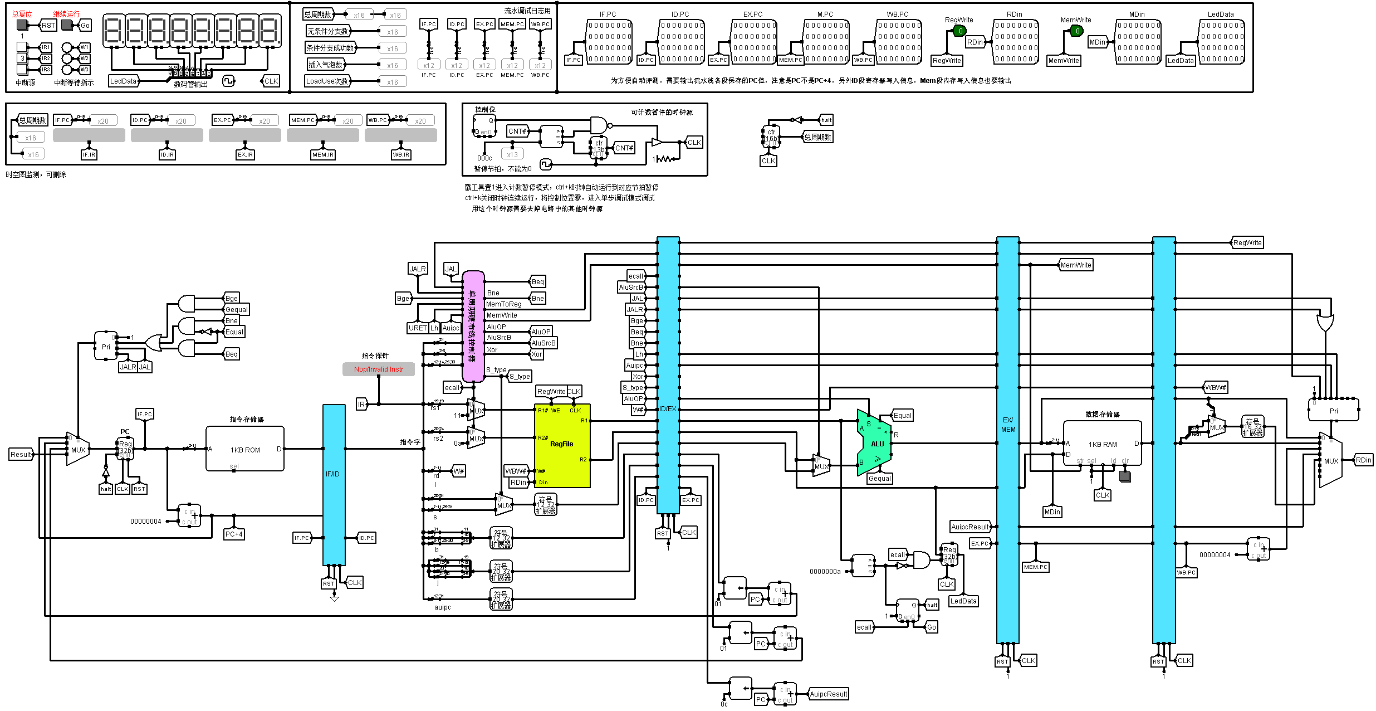


图3.13 理想流水线电路图

进入EX的信号有ALUOP、Memtoreg、MemWrite、AluScrb、RegWrite、Syscall、Beq、Beq、JR、JMP、JAL、WriteReg、IR、PC、R1、R2、shamt、立即数。

进入MEM的信号有Memtoreg、MemWrite、RegWrite、Syscall、JR、JAL、WriteReg、IR、PC、R1、R2、aluout。

进入WB的信号有Memtoreg、RegWrite、JAL、WriteReg、IR、PC、aluout、data。

## 气泡式流水线实现

气泡流水线是在理想流水线的基础上进行改进，加入了数据相关检测的子电路。该子电路的设计逻辑见2.4所述。根据逻辑完成设计后的电路图见图3.14所示。

对于气泡流水线的消除控制冲突在EX段执行分支指令，需要清除 IF/ID、ID/EX 流水寄存器以及消除数据冲突： ID段和MEM段冲突，IF、ID暂停等待数据写回、EX段插入气泡；ID段和EX段冲突，IF、ID暂停等待数据写回、EX段插入气泡。

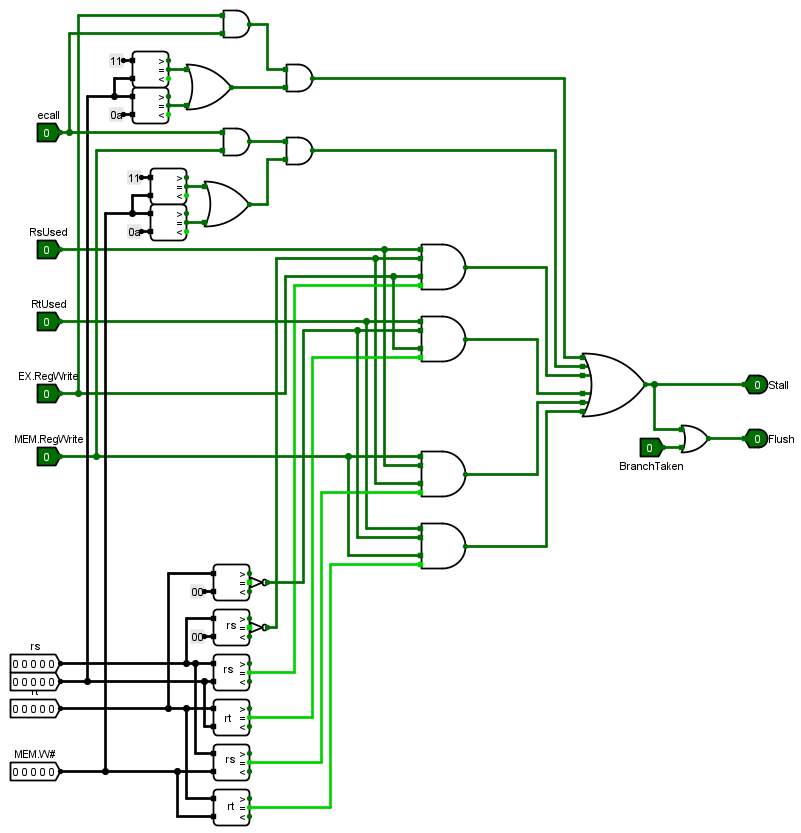


图3.14 数据相关检测电路

采用插入气泡的方式处理数据相关的处理后，流水线数据通路如图3.15所示。图中相关处理逻辑的输入信号除 EX、 MEM 段的 RegWrite、 WriteReg#信号外，还包括 ID 段的指令字 Instr、 EX 段的分支跳转 BranchTaken 信号，输出则为阻塞暂停信号 Stall、流水清空信号 ID/EX.Flush，具体可根据前面的逻辑表达式设计生成组合逻辑电路。

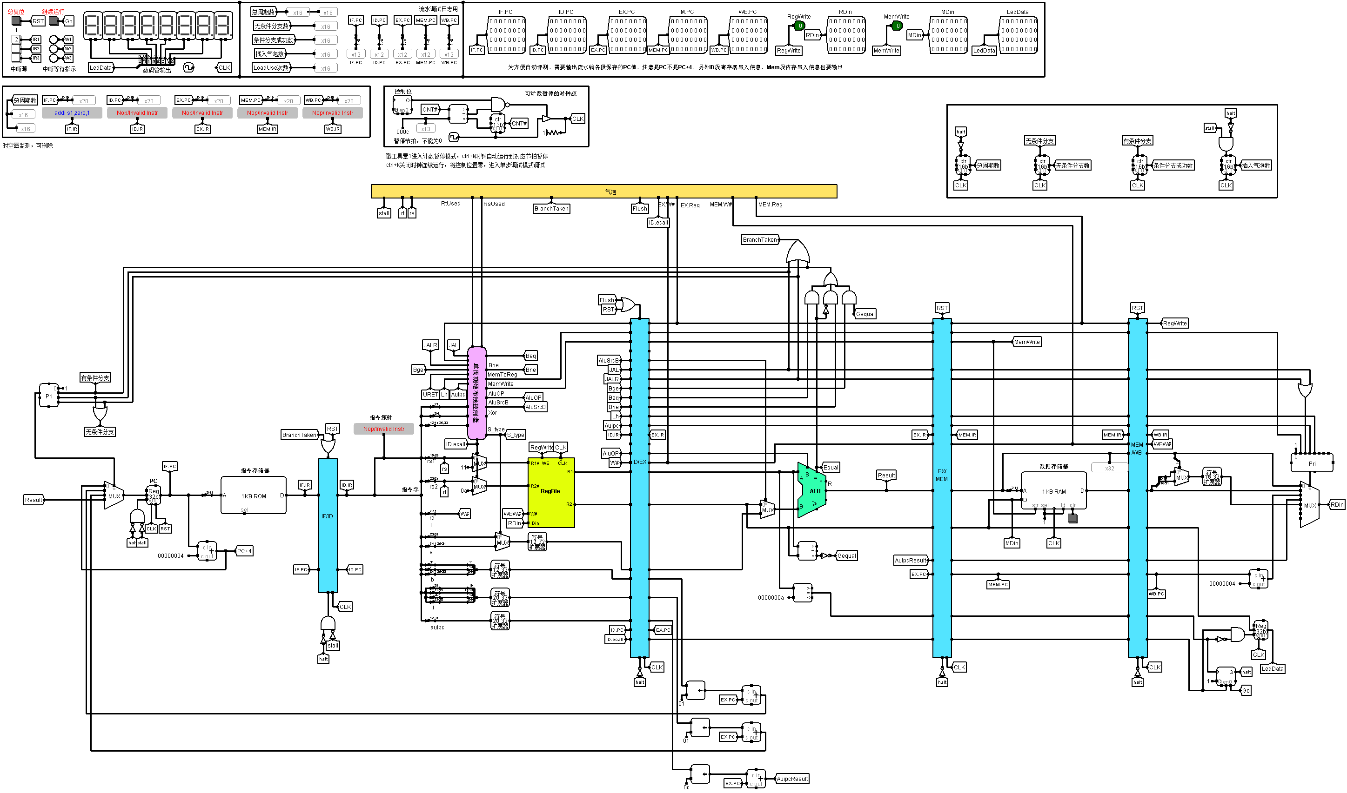


图3.15 气泡流水线电路

## 数据转发流水线实现

数据相关检测与气泡流水线有些不一样，若R1产生冲突，不论是哪里，R1所传入的值都为EX.R1。若只有EX.R1有冲突，MEM没有，则传入的值为ALUIN;若EX和MEM两个阶段的R1都和ID段没有冲突，则以MEM.Memtoreg为准，若MEM.Memtoreg为1，传入MEM.readData，若为0，传入ALUOUT。其真值表如表 3.3所示，forward2与forward1同理。

表 3.3 forward1真值表

|  |  |  |  |
| --- | --- | --- | --- |
| MEM.Memtoreg | EX.R1 | R1 | FORWARD1 |
| 0 | 0 | 0 | 00 |
| 0 | 0 | 1 | 01 |
| 0 | 1 | 0 | 00 |
| 0 | 1 | 1 | 11 |
| 1 | 0 | 0 | 00 |
| 1 | 0 | 1 | 10 |
| 1 | 1 | 0 | 00 |
| 1 | 1 | 1 | 11 |

重定向的数据相关检测电路图可在气泡的数据检测基础上根据以上修改得出，绘制的电路图见图3.16。

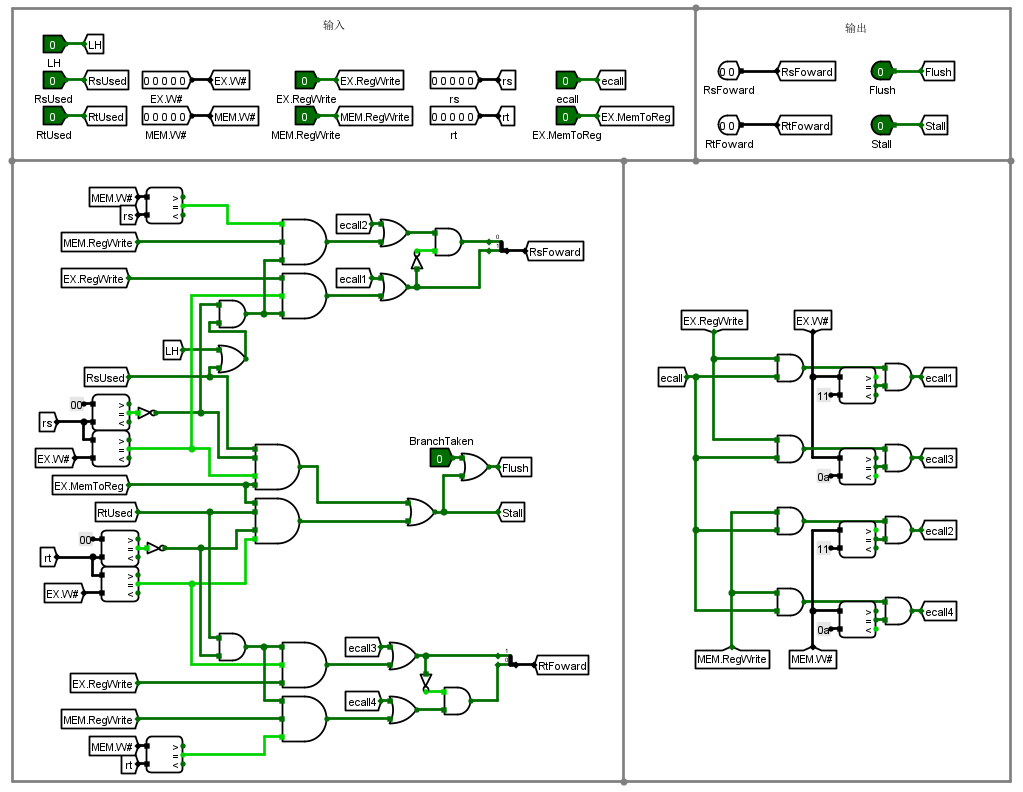


图3.16 重定向数据相关检测

采用插入重定向方式处理数据相关的处理后，流水线完整数据通路如图3.17所示，图中相关处理逻辑输入除 EX、 MEM 段的 RegWrite、 WriteReg#信号外，还包括 ID 段的指令字 Instr、EX 段的分支跳转 BranchTaken 信号，输出为暂停信号 Stall、ID/EX.Flush、RsForward、RtForward，整体逻辑为组合逻辑电路。

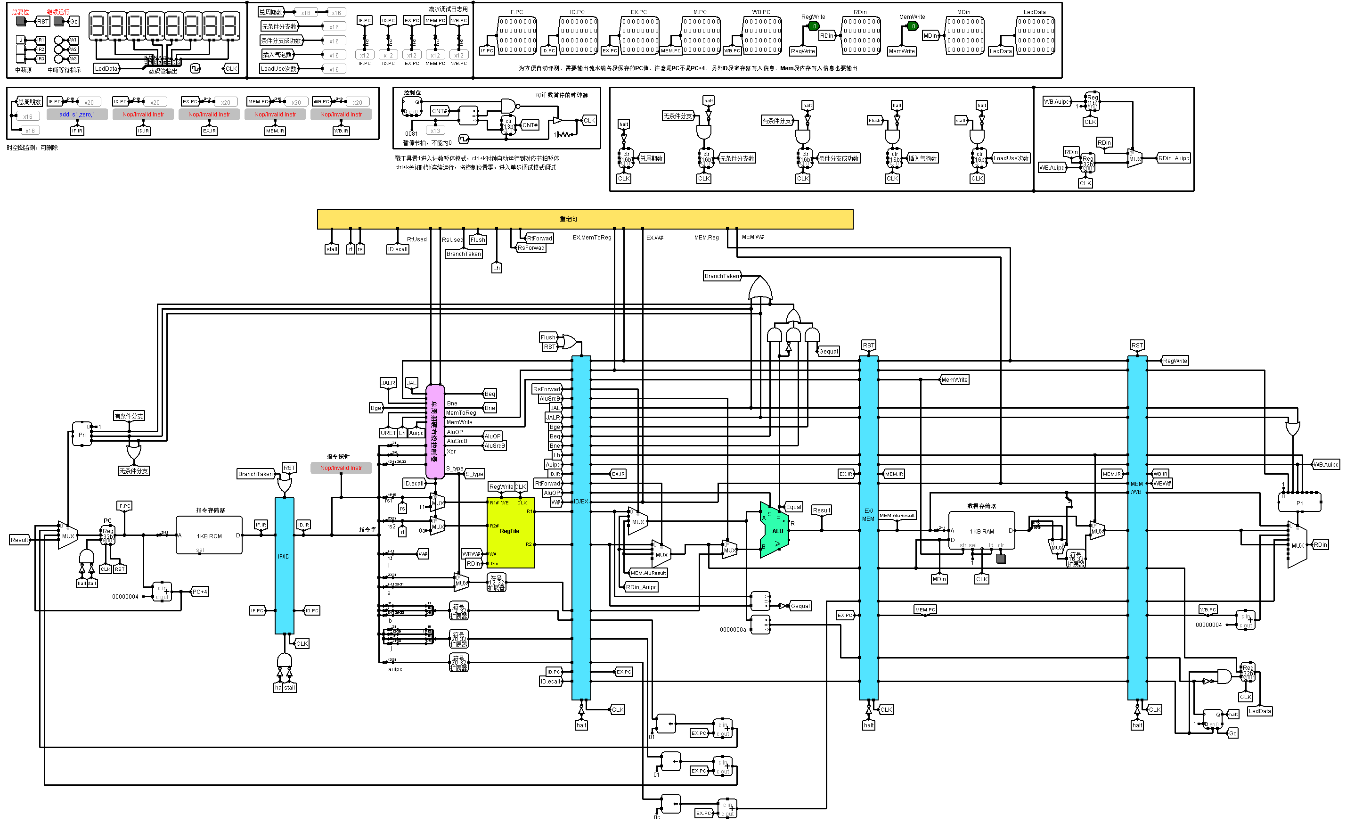


图3.17 重定向流水线电路

# 实验过程与调试

## 测试用例和功能测试

根据老师所给的ARM文件，用RARS打开，根据任务书中所给的步骤生成hex测试文件，对已经实现的电路进行相应的检测。

### 单周期CPU

总周期数为1546，数码管停在38，符合预期结果。结果如图 4.1。

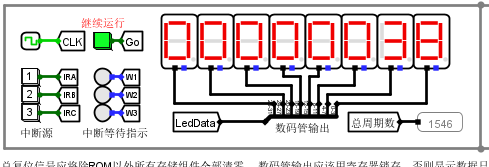


图 4.1 单周期CPU测试

### 单周期+单级中断

1、2、3中断信号灯的亮暗顺序为：1亮2亮3亮1灭3灭2灭1亮1灭，优先度3>2>1，灯灭顺序符合预期结果。

### 单周期+多级中断

依次点击1，2，3号中断按键，先后进入1→2→3→2→1→CPU；依次点击2，3，1号中断按键，先后进入2→3→2→1→CPU；优先度3>2>1，灯灭顺序符合预期结果。

### 理想流水线

总周期数20，内存数据写入正常，符合预期结果。测试结果见图 4.2。

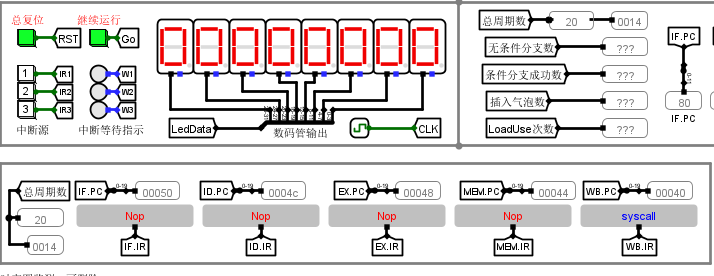


图 4.2 理想流水线测试

### 气泡流水线

能正确运行benchmark+CCAB指令；能统计气泡数目、分支跳转次数；总周期数3624，数码管停止38，符合预期结果。测试结果见图 4.3。

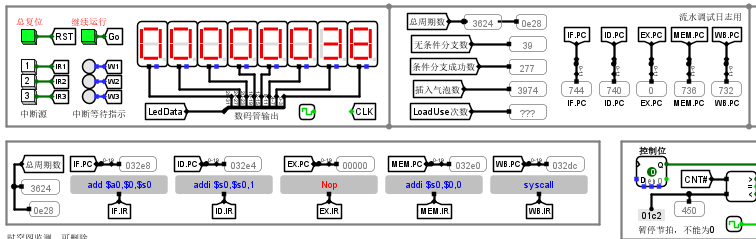


图 4.3 气泡流水线测试

### 重定向流水线

能正确运行benchmark+CCAB指令；能统计Load-Use次数、分支数；总周期数2299，数码管停在38，符合预期结果。测试结果见图 4.4。

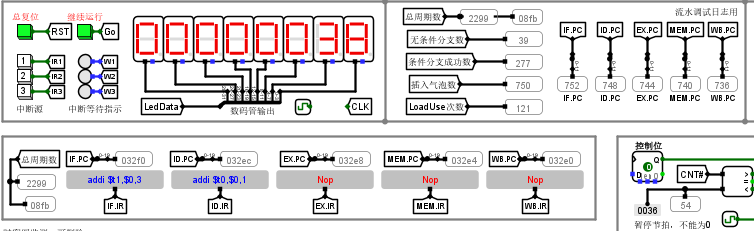


图 4.4 重定向流水线测试

### 重定向流水线+单级中断

1、2、3中断信号灯的亮暗顺序为：1亮 2亮 3亮 1灭 3灭 1亮 2灭1灭

优先度3>2>1，灯灭顺序符合预期结果。

### 动态分支预测

能正确运行benchmark+CCAB指令；总周期数为1801，符合预期结果。

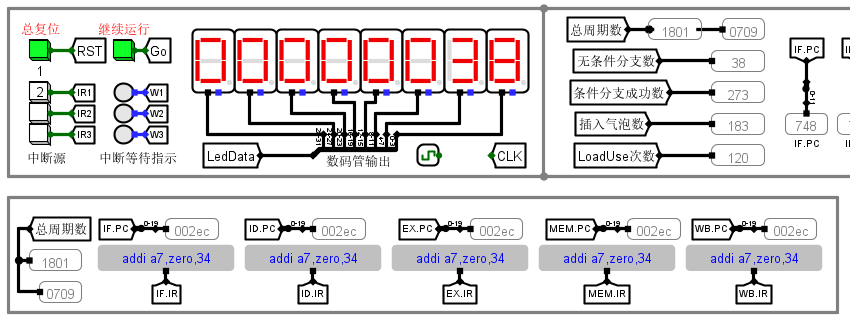


图 4.5 重定向流水线测试

### 差异化指令

1. XOR：

数码管依次输出：0x00007777 0xffff8888 0x00007777 0xffff8888 0x00007777 0xffff8888 0x00007777 …… 0xffff8888 0x00007777，符合预期结果。

测试结果见图 4.6。

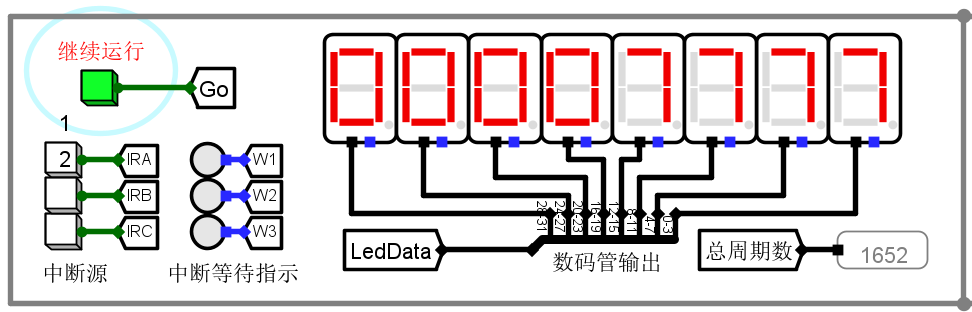


图 4.6 SLLV测试结果

1. AUIPC：

数码管依次输出：0x00430004 0x00430014 0x00430024 0x00430034 0x00430044 0x00430054 0x00430064 0x00430074，符合预期结果。

测试结果见图 4.7。

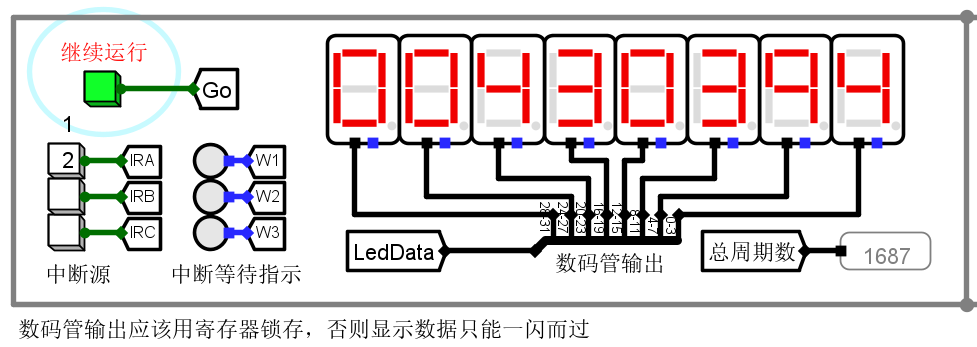


图 4.7 XOR测试结果

1. LH：

数码管依次输出： 0xffff8281 0xffff8483 0xffff8685 0xffff8887 …… 0xffffbebd 0xffffc0bf，符合预期结果。

测试结果见图 4.8。

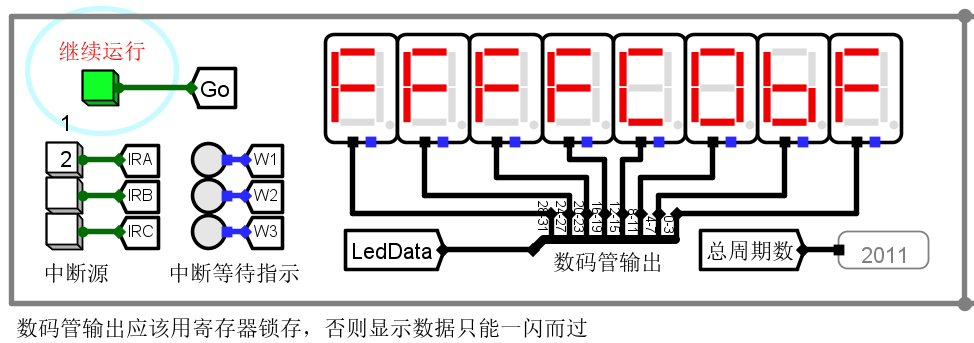


图 4.8 SH测试结果

1. BGE：

依次输出：0x0000000f 0x0000000e 0x0000000d 0x0000000c 0x0000000b …… 0x00000005 0x00000004 0x00000003 0x00000002 0x00000001，符合预期结果。

测试结果见图 4.9。

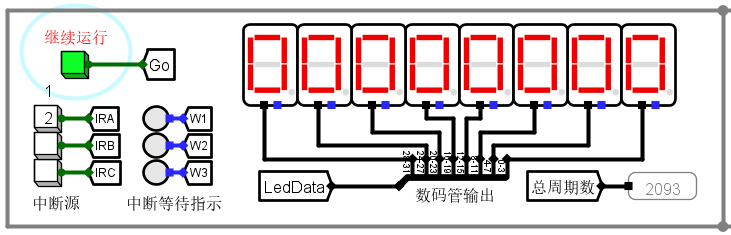


图 4.9 BLTZ测试结果

## 性能分析

根据表 4.1可得，虽然单周期CPU的时钟周期数少，但是实际情况中单周期的每个时钟周期的时间要比流水线的一个时钟周期时间要大。

假设流水线的一条指令从头到尾的五个小时钟周期的时间与单周期一条指令的时钟周期的时间相等（因为是五段流水线），则单周期需要7730个小时钟周期时间，气泡流水线需要3624个时钟周期时间，重定向流水线需要2299个时钟周期时间。

可见重定向流水线的效率>气泡流水线>单周期CPU。

表 4.1分析不同方案时钟周期数差异

|  |  |
| --- | --- |
| CPU | 总时钟周期 |
| 单周期CPU | 1546 |
| 理想流水线 | 20（不能跑benchmark） |
| 气泡流水线 | 3624 |
| 重定向流水线 | 2299 |

## 主要故障与调试

### 气泡流水线电路故障

**故障现象：**电路中的Branchtaken信号在不应该亮时变亮，测试程序结果不对。

**原因分析：**在判断是否清空分支时使用了IF阶段的信号，而电路整体采用EX阶段分支，应该使用EX阶段的信号判断，如图 4.10。

**解决方案：**使用EX阶段的分支信号判断Branchtaken。

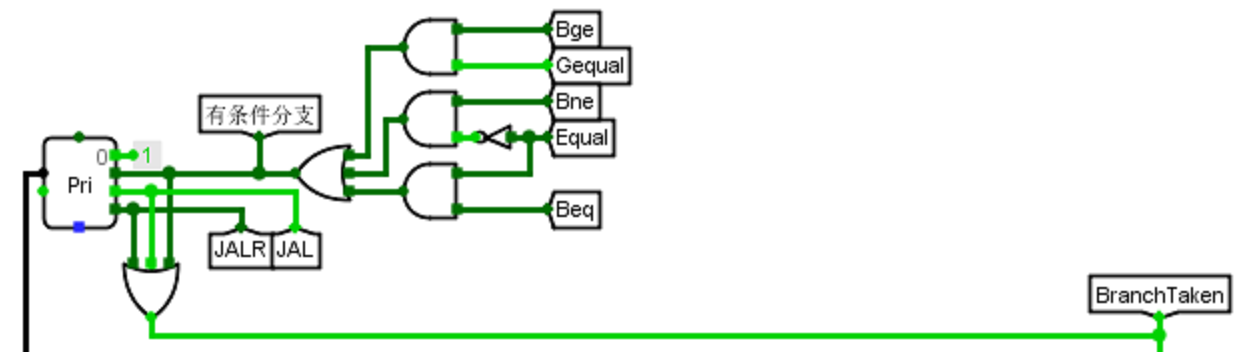


图 4.10气泡流水线故障

### 中断信号清零故障

**故障现象：** 测试时中断信号的顺序不正确，3直接中断了1，比1 先执行完。

**原因分析：**中断信号的清零机制可能出了问题，在清零时，使能端应该是中断有效。 即多个中断一起来时，因为是单级中断，不会打断当前的中断。

**解决方案：**在清零寄存器的使能端添加中断信号，为1时忽略时钟。如图 4.11所示。

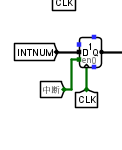


图 4.11中断信号清零故障

### 周期计数电路故障

**故障现象：** 计算周期的电路计数错误。

**原因分析：** 由于logisim不同版本的存储器封装不同，导致切换不同版本时周期计数的电路连线出错，如图 4.12所示。

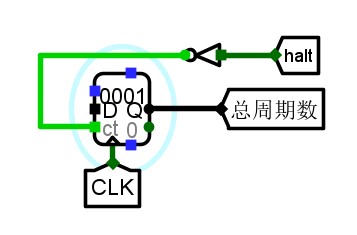


图 4.12周期计数电路故障

**解决方案：** 修改此处的连线，将halt接在存储器使能端，在整个实验过程中尽量不要切换版本，避免此类错误。

### 中断表格填写故障

**故障现象：**测试中断电路时，CSRRSI信号为高电平时，RegWrite信号也为高电平，如图 4.13所示。

**原因分析：**Excel表格填写有误，CSRRSI一行中RegWrite应不填。

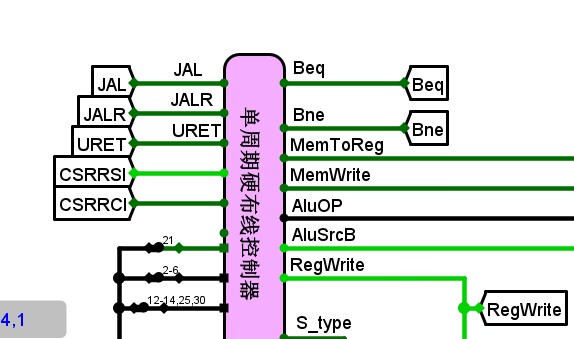


图 4.13表格填写错误

**解决方案：** 修改表格。

### 重定向流水线封装故障

**故障现象：** 检测重定向流水线时，电路错误停止。

**原因分析：** 封装WEM/WB时引脚位置偏移，导致电路中使用时连了空引脚，如图4.14所示。

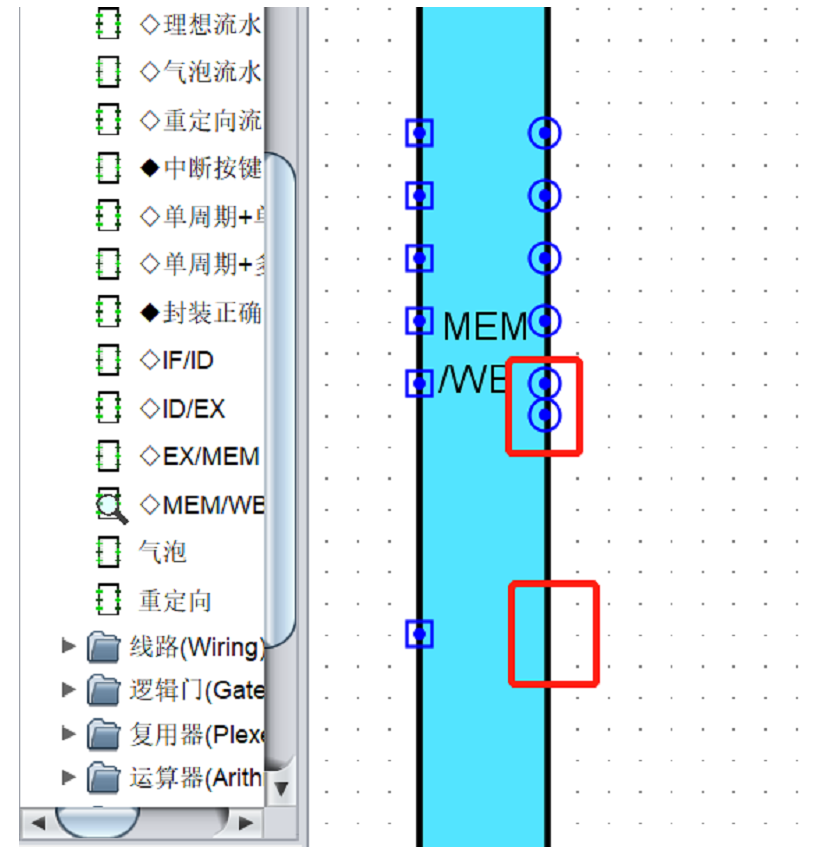


图4.14 重定向流水线封装故障

**解决方案：** 修改封装的引脚位置。

### 气泡流水线逻辑故障

**故障现象：** 运行气泡流水线的测试程序，在执行ecall指令时，EX.PC应该为0，但电路显示不为0。

**原因分析：** 气泡处理逻辑除了数据相关判断有阻塞，还要避免RAW的冲突，也需要插入气泡，所以需要判断ecall指令在EX阶段写入的寄存器是不是a0和a7，但电路中没有考虑。

**解决方案：** 修改电路，在数据相关检测逻辑中加上ecall指令相关的判断，如图4.15。

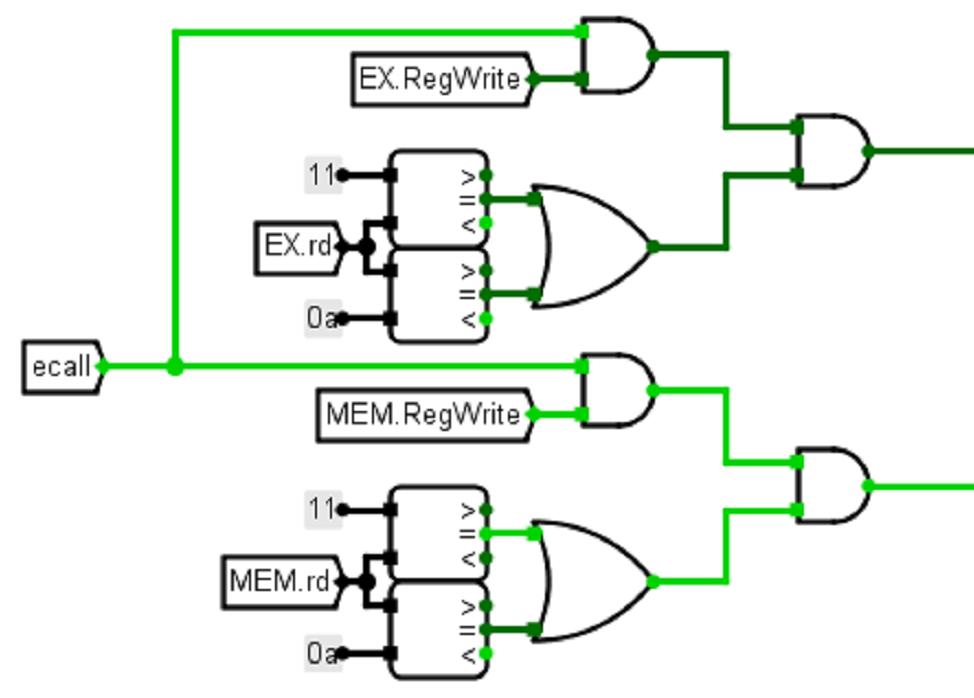


图4.15 气泡流水线逻辑故障

### AUIPC指令故障

**故障现象：**测试单周期AUIPC指令时，LED显示和PC值错误。

**原因分析：**在保存ALU计算结果时使用的标签名为“Result”，但在计算PC值时使用的标签为“AluResult”，导致PC值错误，如图4.15所示。

**解决方案：**将标签名改为一致。

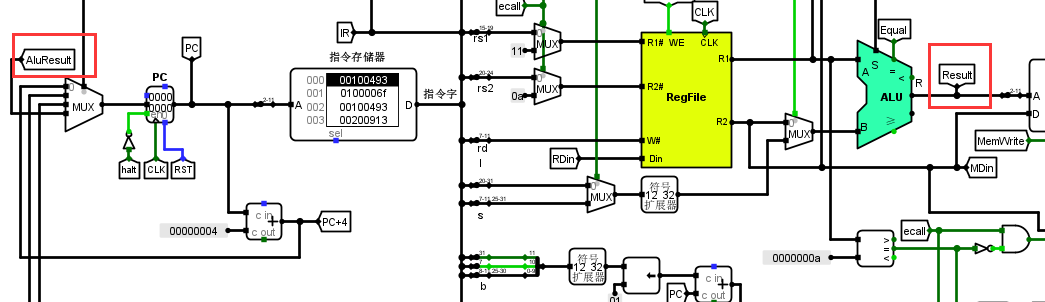


图 4.16 AUIPC指令故障

### 气泡流水线差异化故障

**故障现象：**测试气泡流水线电路的差异化指令时，按Go不能正确的跳转显示下一个指令。

**原因分析：**对流水线的理解有误，halt及LedData应在最后的WB阶段输出，但原电路将其在EX阶段输出了。

**解决方案：** 修改该转换前的电路，将halt及LedData移至WB阶段输出。修改之后如图 4.17所示。

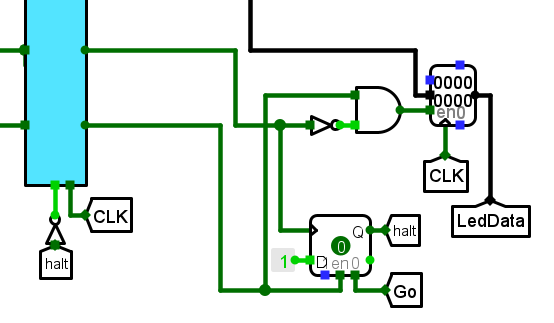


图 4.17 气泡流水线差异化故障

## 实验进度

表 4.2 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RISC-V指令手册，并列出CPU各部件的数据通路表，并完成数据通路的基本构建。 |
| 第二天 | 完成单周期CPU的控制信号表，使用Logisim搭建控制器，实现了单周期CPU并且通过了测试。完成部分Logism单周期CPU故障报告。 |
| 第三天 | 完成Logism单周期CPU的故障报告，并且通过了单周期CPU的检查。 |
| 第四天 | 在Logisim平台上设计流水段间接口部件，完成理想流水线电路的故障报告，并且通过了相关检查。 |
| 第五天 | 完成气泡流水线的数据冲突检测，处理分支冲突，并通过了Logisim气泡流水线的检查。 |
| 第六天 | 为流水线添加重定向机制和控制器改造，并通过了Logisim平台上重定向流水线的检查。 |
| 第七天 | 设计单周期CPU中断硬件，正在调试。 |
| 第八天 | 完成单级中断的调制，并为单周期CPU添加多级嵌套中断，通过了Logisim上的单级中断、多级中断的检查。 |
| 第九天 | 为流水线添加单级中断机制，并通过了Logisim上流水线中断的测试。 |
| 第十天 | 成功完成相联存储器设计，用LRU算法实现动态分支预测，预测成功率显著提高。 |

# 设计总结与心得

## 课设总结

在当前对计算机组成原理学习的基础上，为了加深对其各种CPU的理解，使CPU的各个流程更加具体化，开始对CPU进行设计，对于CPU的设计作了如下几点工作：

1. 制定出设计方案；分析RISC-V指令系统格式，指令系统功能。
2. 根据指令系统构建基本功能部件，主要数据通路。
3. 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。
4. 设计出实现指令功能的硬布线控制器。
5. 根据通路设计出单周期CPU，并完成差异化指令。
6. 在单周期CPU的基础上设计单级中断和多级中断。
7. 在单周期CPU的基础上进行分段设计，设计出理想流水线。
8. 在完成理想流水线的基础上根据流水线冲突原理设计出气泡流水线。
9. 了解气泡流水线和重定向流水线的区别，根据其设计重定向流水线。
10. 在重定向流水线的CPU上完成自己的差异化指令。
11. 将重定向流水线和单级中断结合起来，使重定向流水线具有单级中断的功能。
12. 设计相联存储器完成动态分支预测机制。
13. 课程设计报告和总结。

## 课设心得

本次课设设计对于我来说极具挑战性，尤其是暑假刚开始回忆复习CPU的相关知识的时候，对于CPU的相关知识已经部分遗忘，刚开始画单周期CPU和研究课程说明书时，花费了较多时间。不过暑假画完的单周期CPU是整个课设的基础，因为前面了解的比较透彻，所以在开学后做后面的实验时，稍显轻松。

在我看来，本次课设中最难的部分在于多级中断和重定向流水线。做多级中断之前先做了单级中断，在一定程度上减少了多级中断的难度，其实主要是中断部分画IE寄存器的时候的逻辑的难度比较大，经常因为输入的信号和关系不对而中间错了某个时钟周期。对于重定向流水线，该流水线是在气泡流水线的基础上完成的。气泡流水线的数据相关检测的逻辑调试了很久，不过按照教材上给的逻辑，理解起来非常清楚。而完成重定向流水线时，还是因为数据相关逻辑比较难。

此外，本次课设电路出现次数最多的故障均和引脚、隧道标签有关，这也提醒我在使用Logisim时要格外注意电路的封装和隧道的使用。在今后的学习、生活中，也要更加细心一些。

课设的难度个人觉得刚好，指导书上可以给出大部分提示，但也留下很多自己思考的空间，既有趣味性又有挑战性。并且，本次课设与教材、慕课的内容非常的贴切，在做实验时会有很强的学以致用的感觉，同时对于CPU的相关理解也会更加透彻，完成之后让人印象深刻。

但是在完成课设的过程中，也踩了许多不必要的坑。例如Hust改良版的Logisim虽然方便分析多输入电路，但是和原版的Logisim在寄存器等部件上的封装略有不同，实验调试时起初未发现，导致在使用不同版本时产生了一些电路、引脚连接的错误。此外，在将Logisim转Verilog时也常常出现一些说明书中没有的bug，个人觉得可以将这些常见错误都收集起来，进一步完善经验文档。

# 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.

# 附录 团队任务

## 项目介绍

在本次团队任务中，我们使用实验设计的单周期 CPU，搭配 Logisim 中的显示组件，完成了一个视频播放系统，成果如图1所示。

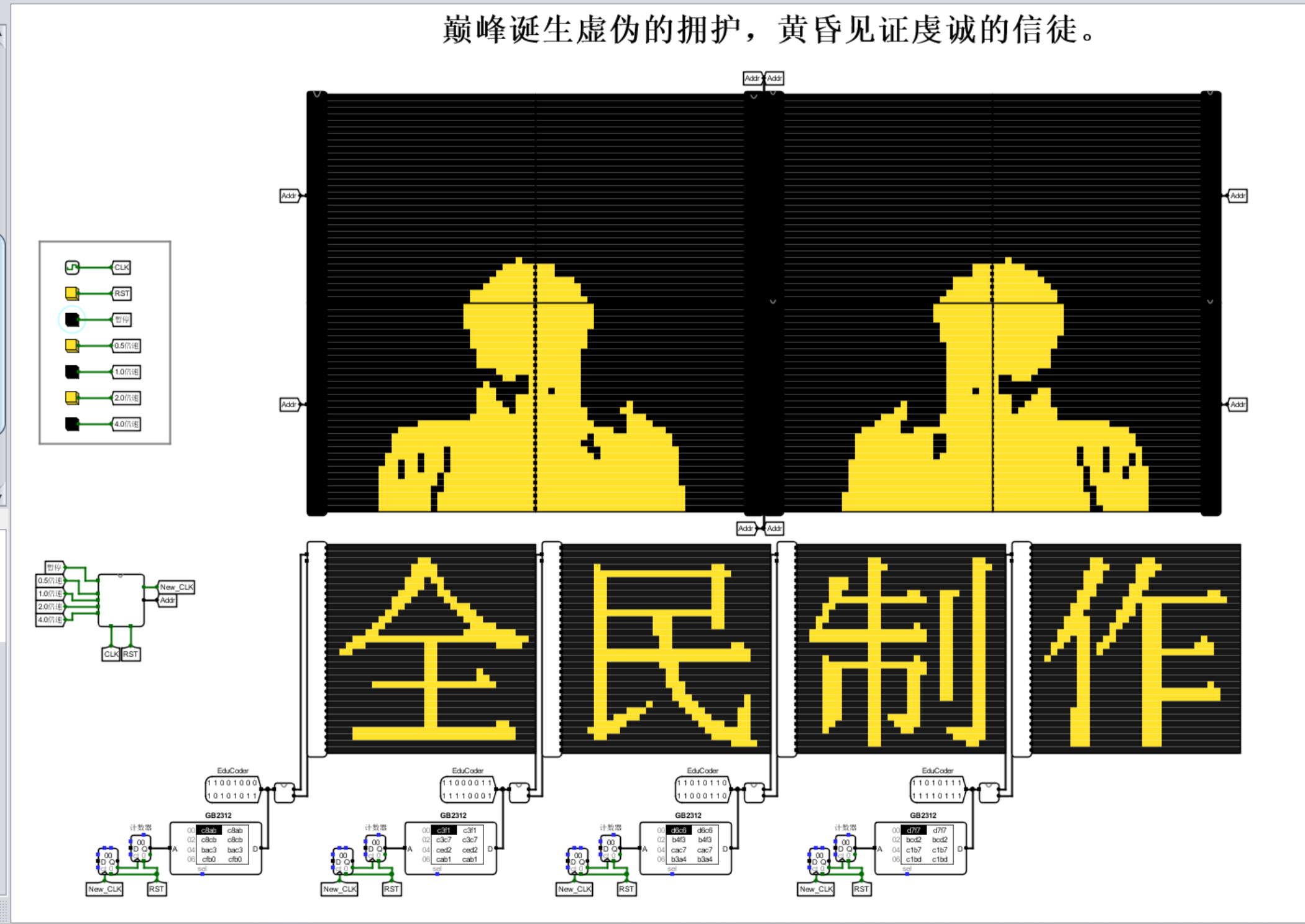


图1 成果电路图

项目的主要亮点有：

1) 功能齐全：该视频播放系统实现了暂停、重置、倍速等播放功能。

2）形式创新：不同于一般的播放器，我们添加了镜像播放，使视频的形式更加有趣。

3）工作量丰富：我们结合了上学期的组原课设，为视频播放器添加字幕功能，丰富工作量和成果电路。

## 团队分工

* （视频抽帧）

负责选定合适的抽帧频率、视频分辨率宽、视频分辨率高；通过opencv -python实现多线程视频抽取关键帧。

* （视频取模）

负责使用取模软件选定合适的阈值转换成点阵信息；通过python实现16进制转换成ROM可读的形式。

* （显示电路）

负责执行指令进行编写；分频器实现不同倍速；控制器实现暂停播放；存储器组读入数据；顶层显示器实现展示。

* （字幕电路）

负责根据区位码原理字幕转换为GB2312格式；通过计数器实现对时钟分频、调整ROM中汉字重复个数以保持与视频同步；剪辑演示视频。

## 项目实现

* **视频抽帧**

通过Python自带的OpenCv模块实现抽帧，主要使用了cv2.Videocapture( )和camera.read( )函数，主要代码如图2所示。

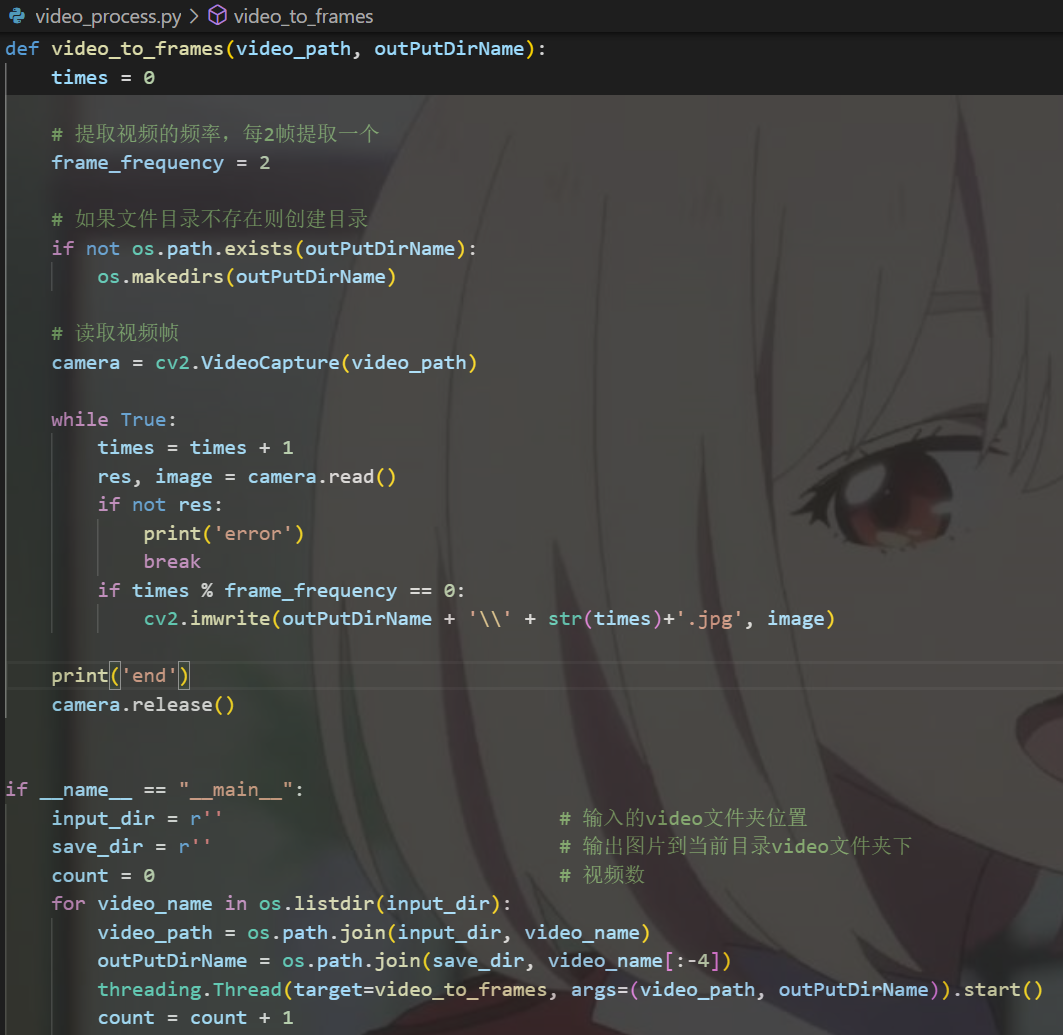


图2 抽桢代码

由于单线程单帧抽取实现速度较慢，所以我们考虑多线程抽取，最终抽取频率确定为2帧。取桢后得到的图片如图3所示。



图3 取桢后图像

* **视频取模**

使用 PS 对抽取的关键帧进行裁剪，进一步使用取模软件对初步处理的照片进行取模，选定色彩阈值为 59%，最后将照片转换为点阵信息，取模后的效果如图3所示。

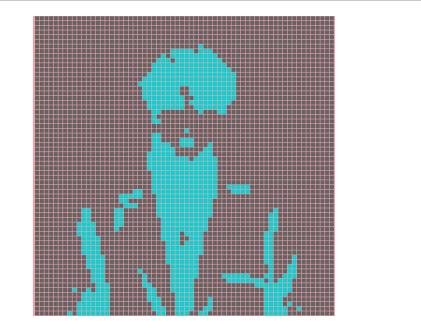


图4 取模后效果

点阵数值信息需要转化为特定的文件格式才能被 Logisim 读取，这里我们使用Python 编写文件处理程序，将上述 16 进制信息转为 ROM 可读取的形式。

* **显示电路**

单周期CPU改造：输入为CLK和RST，输出为Addr，利用IR中的OP和FUNCT对LW指令进行识别，每到来一条LW指令。电路如图5所示。

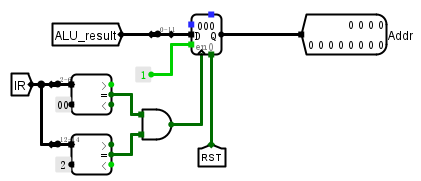


图5 单周期CPU改造

执行指令编写：总帧数为1220，利用循环完成地址从0到1219的遍历，lw读取的初始地址为0，l执行完后地址加1，直到等于1220退出循环。

分频器实现：我们将当前时钟频率的1/8作为基准频率，则8分频的输出结果为1倍速；4分频的输出结果为2倍速；2分频的输出结果为4倍速；原时钟为8倍速。通过多路选择器进行选择输出。主要电路部件如图6所示。

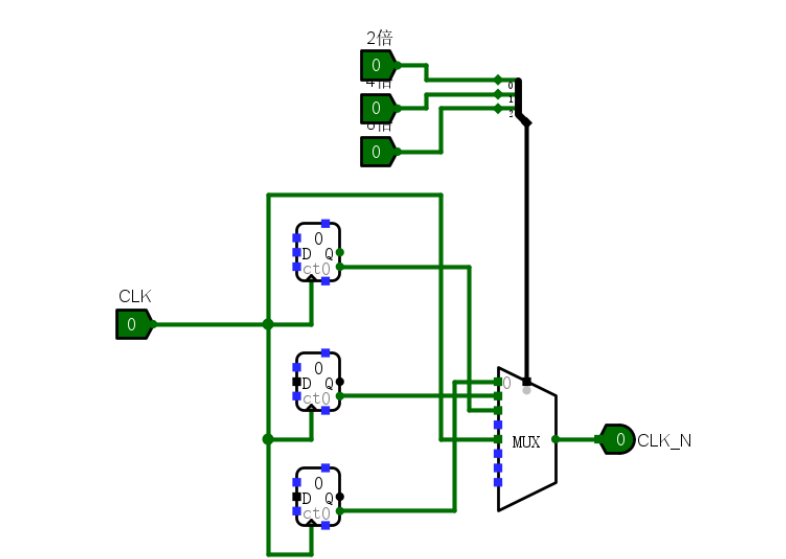


图6 分频器实现

控制器实现：利用D触发器的置0和置1操作，可以让控制器外围控制倍速的输入改为按钮；同样是利用D触发器，输入外围输入的暂停信号，将当前D触发器输出的暂停信号取反后输入D触发器，就可以利用按钮来实现暂停与继续的切换。

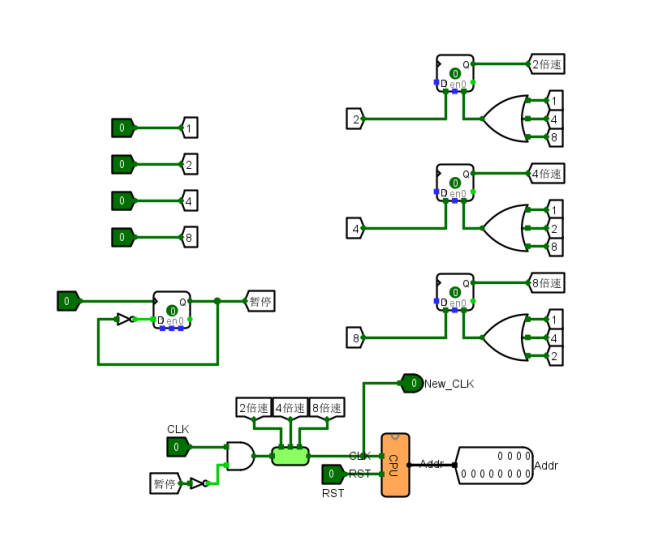


图7 控制器实现

存储器组实现：每一块32x32显示屏对应一个ROM存储器组，一共8组，一组中有32个ROM，每个ROM对应1行的所有数据。

顶层显示屏实现：输入为CLK、RST、暂停信号、倍速控制信号，利用控制器输出访存地址Addr,将Addr分别输入8个存储器组，每个存储器组的输出连接显示屏的输入。

* **字幕电路**

设计原理：区位码=GB2312+dfe0，我们将字幕转化为GB2312，再由电路输出显示。

主要难点：字幕和视频的速度不匹配。我们通过计数器实现对时钟分频、调整ROM中汉字的重复个数，使字幕与视频基本同步；并且能满足倍速调整的要求。

主要电路部件如图8所示。

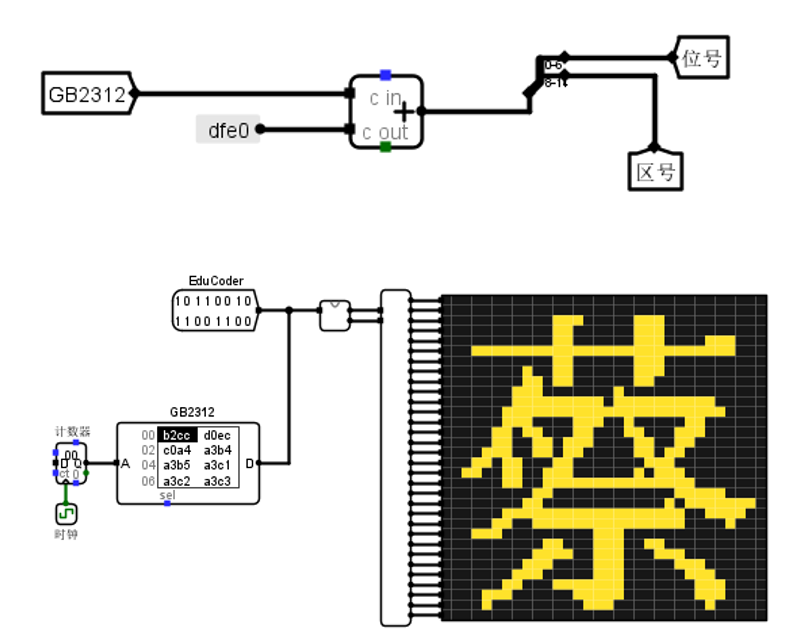


图8 字幕电路部件

|  |
| --- |
| 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] |