***2023***



**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： | CS2103 |
| 学 号： | U202114972 |
| 姓 名： | 刘轩灵 |
| 电 话： | 17843526243 |
| 邮 件： | [1720572012@qq.com](mailto:13456@qq.com) |

目 录

[1 课程设计概述 3](#_Toc60)

[1.1 课设目的 3](#_Toc27898)

[1.2 设计任务 3](#_Toc10052)

[1.3 设计要求 3](#_Toc16458)

[1.4 技术指标 4](#_Toc313)

[2 总体方案设计 6](#_Toc12121)

[2.1 单周期CPU设计 6](#_Toc21341)

[2.2 中断机制设计 9](#_Toc6293)

[2.3 流水CPU设计 10](#_Toc16341)

[2.4 气泡式流水线设计 12](#_Toc1359)

[2.5 重定向流水线设计 12](#_Toc25442)

[2.6 动态分支预测机制 13](#_Toc23830)

[3 详细设计与实现 14](#_Toc5604)

[3.1 单周期CPU 实现 14](#_Toc5293)

[3.2 中断机制实现 17](#_Toc3051)

[3.3 流水CPU实现 21](#_Toc15825)

[3.4 气泡式流水线实现 21](#_Toc18920)

[3.5 重定向流水线实现 22](#_Toc17978)

[3.6 动态分支预测机制实现 24](#_Toc7603)

[4 实验过程与调试 27](#_Toc26475)

[4.1 测试用例和功能测试 27](#_Toc14728)

[4.2 性能分析 28](#_Toc5719)

[4.3 主要故障与调试 29](#_Toc9127)

[4.4 实验进度 30](#_Toc30953)

[5 团队项目设计与实现 31](#_Toc2109)

[5.1 团队选题 31](#_Toc22755)

[5.2 各功能板块连接 31](#_Toc10045)

[5.3 功能模块实现 32](#_Toc1723)

[5.4 团队分工 34](#_Toc9807)

[6 设计总结与心得 35](#_Toc29789)

[6.1 课设总结 35](#_Toc3847)

[6.2 课设心得 35](#_Toc20217)

[参考文献 38](#_Toc17408)

# 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24条基本32位RISC-V指令；
2. 支持表 1.1后4条教师指定的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** | **指令类型** | **简单功能描述** | **备注** |
| --- | --- | --- | --- | --- |
| 1 | ADD | R | 加法 | 指令格式与功能请参考RISC\_V32指令集英文手册，或参考RARS模拟器。 |
| 2 | ADDI | I | 立即数加 |
| 3 | AND | R | 与 |
| 4 | ANDI | I | 立即数与 |
| 5 | SLLI | I | 立即数逻辑左移 |
| 6 | SRAI | I | 立即数算数右移 |
| 7 | SRLI | I | 立即数逻辑右移 |
| 8 | SUB | R | 减 |
| 9 | OR | R | 或 |
| 10 | ORI | I | 立即数或 |
| 11 | XORI | I | 立即数异或 |
| 12 | LW | I | 加载字 |
| 13 | SW | S | 存字 |
| 14 | BEQ | B | 相等跳转 |
| 15 | BNE | B | 不相等跳转 |
| 16 | SLT | R | 小于置数 |
| 17 | SLTI | I | 小于立即数置数 |
| 18 | SLTU | R | 小于无符号数置数 |
| 19 | JAL | J | 转移并链接 |
| 20 | JALR | I | 转移到指定寄存器 |
| 21 | ECALL | I | 系统调用 | If ($a7==34) LED输出$a0的值  else 停机等待Go按键按下 |
| 22 | CSRRSI | I | 访问CSR寄存器 | 中断相关，可简化为开中断 |
| 23 | CSRRCI | I | 访问CSR寄存器 | 中断相关，可简化为关中断 |
| 24 | URET | I | 中断返回 | 清中断，mEPC送PC，开中断 |
| 25 | SRL | R | 逻辑右移 | 扩展指令 |
| 26 | SRA | R | 算术右移 |
| 27 | SB | S | 存字节 |
| 28 | BLT | B | 小于跳转 |

# 总体方案设计

## 单周期CPU设计

单周期CPU的控制器采用硬布线控制器，使得整个电路是严格的同步时序电路。每个周期，CPU控制各功能部件协调工作，使数据信息按当前指令的数据通路流动，实现对数据的加工处理，并放在相应的存储器中。由于取指令和访问数据时均需要使用存储器，因此采用指令存储器和数据存储器相分离的哈佛结构。

将整个CPU划分为四个部分，可以使设计思路更加清晰取。

取指结构：以PC 的内容为地址从指令寄存器中取出指令，并计算后续指令的地址。译码/取操作数结构：通过分线器对指令字进行译码，并将不同的字段分别传入控制器（OP，Funct）和寄存器组（R1#,R2#），得到对应的控制信号，寄存器数据与立即数（Imm）。执行结构：ALU运算器在控制命令的控制下对操作数实现指令相关的加工处理。访存/写回结构：在控制信号的控制下，实现对数据存储器读取与存放和对寄存器组写回操作。

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

1. 程序计数器PC

程序计数器PC是一个寄存器，用于保存将要执行的指令的字节地址。CPU取指令时，将PC的内容作为地址送入指令存储器中。然后根据程序是顺序执行，还是分支跳转，修改PC的值形成下一条指令的地址。另外还需在停机信号到来时停止计数。

1. 指令存储器IM

指令存储器IM存储了将要执行的RISC-V32指令程序的十六进制数据，当前执行指令地址由当前PC的输出决定，指令取出后进行译码。选取ROM只读存储器来实现IM。

1. 运算器

运算器实现指令的算术逻辑运算功能。根据ALUOP对两个输入A、B进行相应的运算，输出运算结果及相关信息。其输入输出引脚与功能描述见任务书。注意其中＜和≥会根据ALU\_OP的不同分为符号比较与无符号比较。

ALUOP和其对应的运算器功能如表 2.1所示：

表 2.1 算术逻辑运算单元 ALUOP 与对应功能描述

| ALUOP | 功能描述 | ALUOP | 功能描述 |
| --- | --- | --- | --- |
| 0 | 逻辑左移 | 7 | 按位与 |
| 1 | 算术右移 | 8 | 按位或 |
| 2 | 逻辑右移 | 9 | 按位异或 |
| 3 | 无符号乘法，高位使用 result2 存储 | 10 | 按位或非 |
| 4 | 无符号除法，result2 存储余数 | 11 | 有符号比较，小于输出 1 |
| 5 | 加法 | 12 | 无符号比较，小于输出 1 |
| 6 | 减法 |  |  |

1. 寄存器堆RF

寄存器堆是32个通用寄存器集合，CPU通过一定的接口对这些寄存器进行读写访问。R1#、R2#两个输入接口给出寄存器编号,寄存器堆内部就会选择对应寄存器的值输出到端口R1、R2，读逻辑是组合逻辑。写逻辑为时序逻辑，当写使能控制信号WE为1，且时钟上跳沿到来时，寄存器堆将WD端的数据写入编号为W#的寄存器中。

1. 数据存储器 DM

数据存储器可以进行数据的读或写操作。在读信号的控制下，把A端口输入地址所对应的数据输出到RD端口；在写信号的控制下且上跳沿到来时，将WD端口的数据写入A端口输入地址。选取RAM来实现 DM。

### 数据通路的设计

数据通路的设计需要我们对CPU上运行的指令集（RISC\_V32）非常熟悉，在此基础上，我们采用简单迭代法，即先固定一些较为简单的情况，测试通过运行后再在这个基础上不断增加新的数据通路。

### 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。

表 2.2主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 | 控制信号 | 取值 | 说明 |
| --- | --- | --- | --- | --- | --- |
| ALU\_OP | 0~12 | 运算器操作控制符（4 位） | BEQ | 0/1 | Beq 指令译码信号 |
| MemToReg | 0/1 | 寄存器写入数据来自数据存储器 | BNE | 0/1 | Bne 指令译码信号 |
| MemWrite | 0/1 | 写数据存储器控制信号 | JAL | 0/1 | Jal 指令译码信号 |
| ALU\_Src | 0/1 | 运算器B输入端选择控制信号 | JALR | 0/1 | Jalr 指令译码信号 |
| RegWrite | 0/1 | 寄存器写使能 | BLT | 0/1 | Blt 指令译码信号 |
| Ecall | 0/1 | ecall 指令译码信号 | SB | 0/1 | Sb 指令译码信号 |
| S\_Type | 0/1 | S 型指令译码信号 |  |  |  |

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。根据控制信号表达式生成表，如图 2.2，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硬布线控制器逻辑线路。

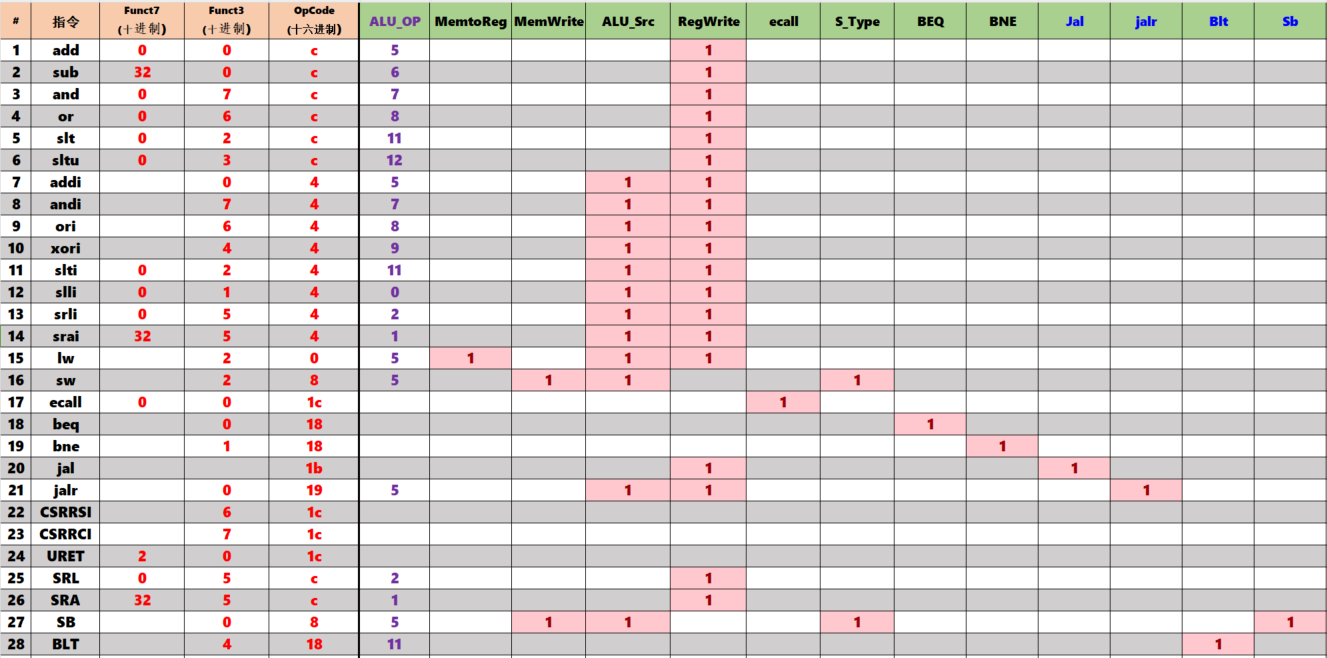


图 2.2 控制信号表达式生成表

## 中断机制设计

### 总体设计

要求设计的中断处理机制仅考虑3个外部按键中断源，分别对应编号为1、2、3三个按钮，中断优先级1<2<3。

首先需要熟悉中断的处理过程：①中断响应：关中断->保存断点->中断识别，这些主要由硬件来实现。②保护现场③中断服务④恢复现场这些主要由软件来实现。⑤中断返回：清除中断请求->开中断->恢复断点，这些由 uret信号控制硬件来完成。

第二就是要注意单级中断与多级中断的区别。单级中断时，一个中断在进行中断服务时，不能被其他外部中断打断。多级中断时，更高优先级的中断请求能打断当前的中断服务，此时需要对当前中断服务进行保存现场。同时有多个中断请求时，会优先响应优先级更高的中断。

### 硬件设计

1. 单级中断

首先考虑中断信号的产生。设计重点是区分好中断源信号，中断请求信号和中断信号的区别。中断源信号来源外部logism按键的按下，其特点是突发且一闪而过，因此需要中断采样电路对中断源的行为进行采样，以产生持续的中断请求信号，直到该中断返回时消除该信号。当且仅当有中断请求信号和允许中断时，才产生中断信号。

是否允许中断涉及到中断使能寄存器IE的设计。需要明确的是开关中断的时机：中断响应时需要关中断，即将IE寄存器置零（中断信号相关）；中断返回时需要开中断（uret指令相关）。

另外，产生中断信号的那一瞬间还会触发三个操作：记录下中断号；将断点保存到 EPC 寄存器中；将对应的中断服务地址传入 PC 寄存器内，这些都是跟中断响应相关的硬件操作。

当中断服务完成，将会出现返回指令uret。uret指令实现了三个功能：开中断，即将IE寄存器置一；将EPC内保存的断点地址返回给PC；产生相应ret信号以消除持续的中断请求信号。

1. 多级中断

多级中断最大的不同就是高优先级的中断服务程序能打断优先级较低的，当这种打断出现时，我们需要保存被打断程序的现场，即其断点地址和中断号（中断返回时有用）。保护现场有软件和硬件两种实现方式，此处采用硬件堆栈。

分别实现栈指针和栈本身。栈指针在中断时进行加一，中断返回时减一，始终指向正在执行的中断。栈本身采用三个寄存器来实现，EPC和中断号各有一个栈，它们使用一个共同的栈指针来指示当前使用的位置。

此外，还需要加入判断中断优先级的逻辑，对比一下当前中断和到来的中断之间优先级的大小关系，来决定是否暂时屏蔽到来的中断。即只有当到来的中断请求优先级高于当前中断，才产生高优先级的中断信号，否则继续执行当前中断。

## 流水CPU设计

### 总体设计

流水CPU的设计采用五段流水架构，框架图如图 2.3所示。程序计数器PC和各个流水寄存器采用统一时钟进行同步。

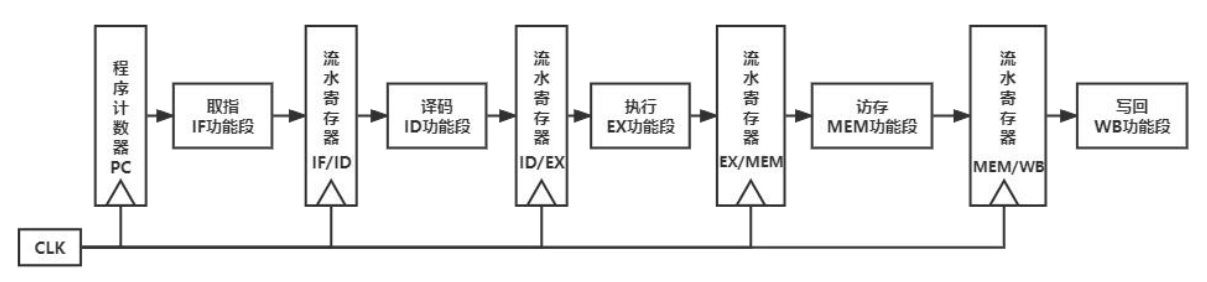


图 2.3 五段流水CPU框架图

五段是指将每条指令的执行过程分为五个阶段，依次为取指（IF）段、译码（ID）段、执行（EX）段、访存（MEM）段、写回（WB）段。流水是指通过各段之间的流水寄存器来锁存各段处理得到的结果并在下一时钟周期到来时传递数据与控制信号给下一个阶段。在理想状态下，这种结构能实现各阶段执行的相对独立性。

但实际的指令流水线中会存在指令相关问题，即某指令的某个阶段必须要等到它前面的某条指令的某个阶段完成后才能开始。如存在各种跳转指令时，取指令不再是按序依次，而要根据跳转指令的结果来进行。或者当前指令需要用先前指令的结果，而这个结果尚未送到指定的位置。上述控制冲突和数据冲突会导致流水线冲突，破坏了流水线各段之间的独立性。

我们将采用两种方案来解决流水线冲突：插入气泡和重定向。此外，我们将采用动态分支预测机制来减少分支延迟的损失，进一步优化流水线性能。

### 流水接口部件设计

四个流水寄存器IF/ID、ID/EX、EX/MEM、MEM/WB用于锁存各段处理得到的结果并在下一时钟周期到来时传递数据与控制信号给下一个阶段。流水接口部件的设计就是根据流水寄存器两端需求和结果给需要传输的数据设计对应的输入接口和输出接口。理想流水线的各流水寄存器锁存数据见表 2.3。

表 2.3 流水寄存器接口设计

| 流水寄存器 | 锁存数据说明 |
| --- | --- |
| IF/ID | PC、IR |
| ID/EX | PC、IR、R1、R2、各类型指令的立即数、W#、控制信号、halt 信号 |
| EX/MEM | PC、IR、Result、MDin、ByteSelect、W#、控制信号、halt 信号 |
| MEM/WB | PC、IR、Result、MDout、W#、控制信号、Halt 信号 |

流水寄存器上升沿触发，有复位端（高电平有效）和使能端（低电平有效）以满足暂停、清空寄存器的需求。

### 理想流水线设计

直接将单周期CPU拆分成五个阶段，并将各输入输出连上流水寄存器。注意每个段所使用的数据和控制信号只能来自前一个流水寄存器的输出，切忌跨段使用。

对于停机指令，在 EX 段处理完停机逻辑后，需要经过两个周期传送到 WB 段再最终执行停机操作，以保证全部指令都执行完毕再停机。

## 气泡式流水线设计

当前理想流水线还存在两种冲突：控制冲突、数据冲突。

对于分支冲突，即当分支指令在EX 段执行时需要跳转，此时IF/ID和ID/EX寄存器内的指令为误取指令。检测到跳转时IF段根据EX段分支目标地址取指令，并将IF/ID和ID/EX寄存器同步清零。

对于数据冲突（专指会造成冲突的RAW），发生在从寄存器堆中取数的阶段与该阶段之后的阶段，即ID段与EX、MEM、WB段。ID 与 WB 的数据冲突只需要将寄存器文件设置为先写后读（下跳沿写入）。ID段与EX、MEM段的冲突要在ID段增加数据相关检测逻辑。一旦检测到数据相关，应该给出PC和ID/EX阻塞信号stall以暂停IF和ID段，同时要将ID/EX寄存器清空，即进行插入气泡。

## 重定向流水线设计

气泡流水线的缺点是插入了太多气泡导致流水线效率极速下降，因此我们换重定向方式来提高效率。与气泡类似，重定向机制在ID段检测数据相关性，但不同的是检出相关性后，我们不再暂停流水线和插入气泡，而是直接将位于后续段的正确的R1、R2的值直接传到用它们的地方。为了实现这一功能，我们只需添加一些直传的数据通路，并在使用处加上多路选择器进行选择。

但是有一种冲突特殊，即Load-Use相关的数据冲突。产生该冲突的是访存指令，如果等待数据存储器读取数据再直传，会将EX段的关键路径延迟变成MEM段访存延迟加上EX段运算延迟，使流水线频率大大降低。对与这种特殊的相关，我们加上LoadUse检测逻辑，出现时暂停IF段和ID段并在EX段插入一个气泡。

## 动态分支预测机制

重定向机制只能提升数据冲突方面的流水线效率。增添动态分支预测机制可以减少“分支”跳转相关的气泡，提高性能。动态分支预测使用一个BTB表来记录分支指令的分支跳转历史，并以此来“预测”下一个PC值。

在重定向流水线的基础上进行动态分支预测的数据通路的修改，主要涉及到PC值的来源选择与流水寄存器的清零条件。PC值的来源选择及其条件如表 2.4。

表 2.4 PC值来源及其条件

| PredictJump(预测跳转) | BranchTaken(实际跳转) | PC来源 |
| --- | --- | --- |
| 0 | 0 | IF.PC+4 |
| 0 | 1 | EX.BranchAddr |
| 1 | 0 | EX.PC+4 |
| 1 | 1 | JumpAddr |

当PredictJump(预测跳转)与BranchTaken(实际跳转)不一致时，生成PredictErr信号，表示动态分支预测错误，此时需要将IF/ID和ID/EX两个流水寄存器清空，通过插入气泡的方式重新执行分支指令。

BTB表采用8路全相联cache来实现。cache行包括有效位Valid、分支指令地址、分支目标地址、分支预测历史位、淘汰计数位。分支预测历史位是一个按有限状态机变化的数据，用于统计历史跳转情况并以此来决定是否跳转;淘汰计数位使用LRU算法在cache槽满后对cache行进行淘汰替换。BTB表的输入输出引脚及其功能说明如图 2.4。

“动态”就是指利用cache行的淘汰替换不断调整可能的预测，且根据分支指令实际的跳转情况不断更新相应cache行的分支预测历史位。主要包括3个过程，IF段进行预测并更新淘汰计数位、EX段命中时更新分支预测历史位，EX段缺失时进行分支信息写入。



图 2.4 BTB表的输入输出引脚及其功能说明

# 详细设计与实现

## 单周期CPU 实现

### 主要功能部件实现

这一部分其实没啥可写的，MEM、RegFile、ALU的logisim电路都是直接有现成的并且就是一些接口而已，而这些功能部件的verilog实现是verilog课的内容，直接搬过来用就行。因此这里仅仅写了PC和IM这两个部件在实现中需要额外注意的点。

1. 程序计数器（PC）

使用一个32位的上升沿触发的寄存器来实现PC，输入即将执行指令的字节地址，输出当前正在执行指令的字节地址。一定要注意的是，送入指令存储器用于取指令的地址是字地址，因此需要将PC后两位扔掉再送入指令寄存器取指令。使用halt信号控制PC的使能，当halt值为1时，非门取反之后将会down掉PC的使能，屏蔽时钟信号，使整个电路停机。

1. 指令存储器（IM）

使用一个只读存储器ROM实现IM。将指令存储器的地址位宽设置为10位，且因为机器指令是32位所以将数据位宽为32位。因为ROM地址仅为10位宽，故所以使用分线器只取PC送入地址的其中十位，且如前所述需要丢掉低两位，所以取2-11位作为指令存储器的输入地址。

### 数据通路的实现

1. Logism实现：

下面的叙述将使用到指令字中的源寄存器号rs1、rs2，目的寄存器号rd，立即数I，扩展立即数sext(I)，数码管输出数据LedData，CPU停机信号halt。其中R型和I型指令的数据通路任务书上面有，就不再赘述。

* lw指令数据通路（特殊的I型指令）

lw指令是一种特殊的I型指令，实现操作R[rd] = M[R[rs1]+sext(I)]。lw指令的数据通路还包含读数据存储器的通路，即将ALU的运算结果作为地址送入数据存储器取出对应存储区的值并写回到目标寄存器中。

* S型指令数据通路

S型指令为存储指令，实现操作M[R[rs1]+sext(I)] = R[rs2]。指令从存储器取出后经过分线器译码得到源寄存器号rs1、rs2，将其输入寄存器文件并读取源寄存器的值，同时将立即数I送入立即数扩展单元得到扩展的立即数sext(I)。之后sext(I)和rs1的值进入ALU进行加法运算得到目标位置的数据存储器地址，最后将rs2的值写入该位置。

* J型指令数据通路

J型指令为无条件跳转指令，实现操作R[rd] = PC+4,PC = PC+sext(I)。指令从存储器取出后经过分线器译码得到目的寄存器号rd，将其输入寄存器文件，同时将立即数I送入立即数扩展单元得到sext(I)。加法器将当前PC值+4后写入到目标寄存器中。另一个加法器将当前PC值+sext(I)，得到目标地址并传递给程序计数器。

* B型指令数据通路

B型指令为条件跳转指令，实现操作if(R[rs1] 条件 R[rs2]) PC = PC+sext(I)。指令从存储器取出后经过分线器译码得到源寄存器号rs1、rs2，将其输入寄存器文件并读取源寄存器的值。将这两个源寄存器的值经过比较器进行比较，比较结果决定了是否将由加法器得到的目标地址（当前PC值+sext(I)）传递给程序计数器。

* ecall 指令数据通路

ecall指令为环境调用指令，实现操作if($a7==34) LedData = $a0 else halt = 1。指令将a7寄存器号17，a0寄存器号10当作源寄存器号rs1、rs2输入寄存器文件并读取源寄存器的值。然后将rs1和rs2的值送入比较器，其比较结果送入条件判断逻辑，控制LED 显示或CPU停机。

* sb指令数据通路

sb指令为S型指令的一种，实现操作M[R[rs1]+sext(I)] = R[rs2][7:0]。该指令需要数据存储器支持字节访问，而数据存储器的输入地址为字地址，因此要增添片选信号的生成逻辑并配合输入数据移位。该数据存储器RAM组件的选择控制端有定义“0001表示D端口的0-7位被选中，0010表示D端口的8-15位被选中...”，将这些选项写为16进制数放在多路选择器的选择端，使用输入地址的末两位作为多路选择器选择端来选择究竟写到哪个字节上。片选信号的生成逻辑如图 3.1。

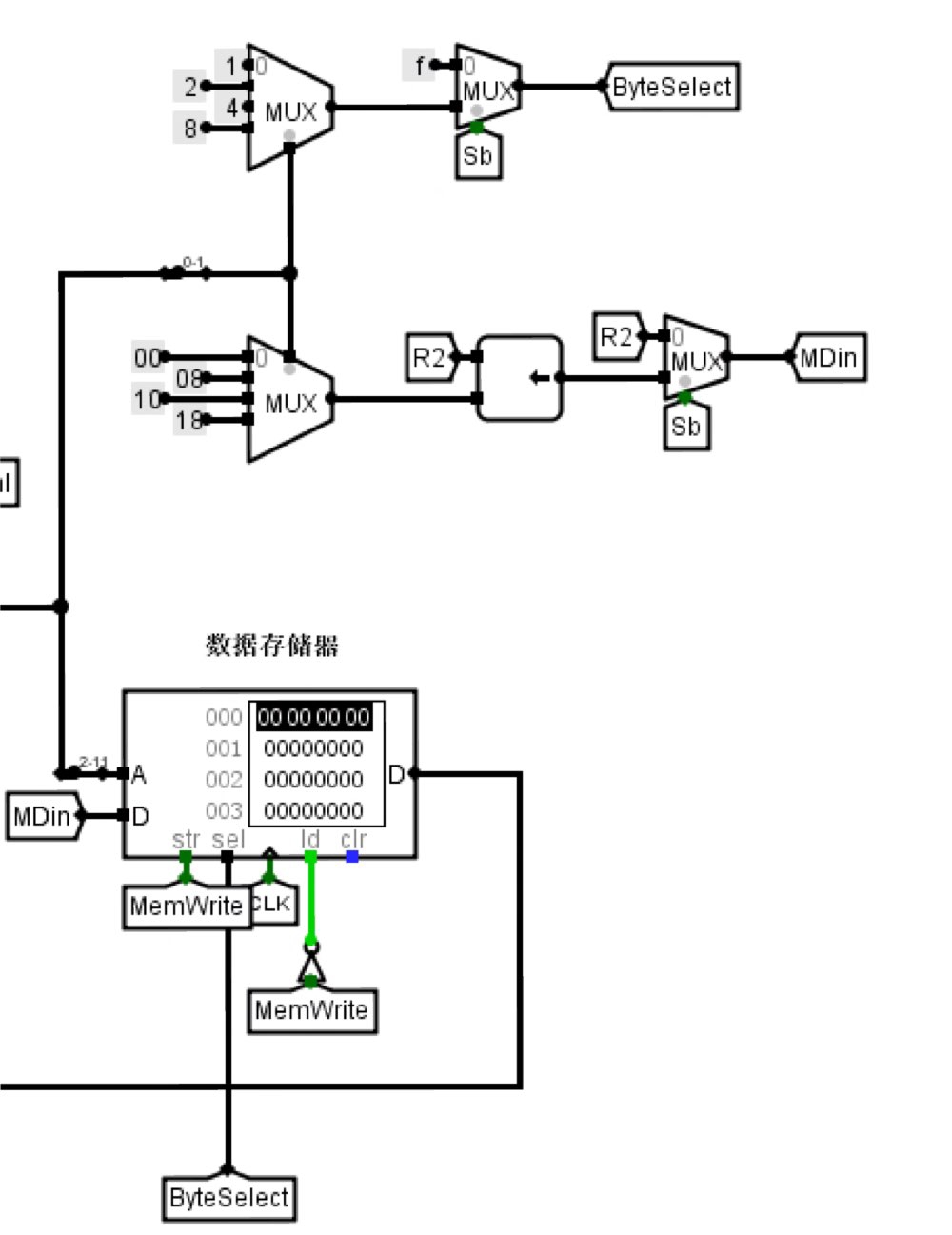


图 3.1 片选信号的生成逻辑

因为这个RAM组件的定义很奇怪，所以我们还要将输入数据中要写入的字节移动到我们片选的位置，选择0-7位不左移，选择8-15位需要左移8位...实现如图 3.2。

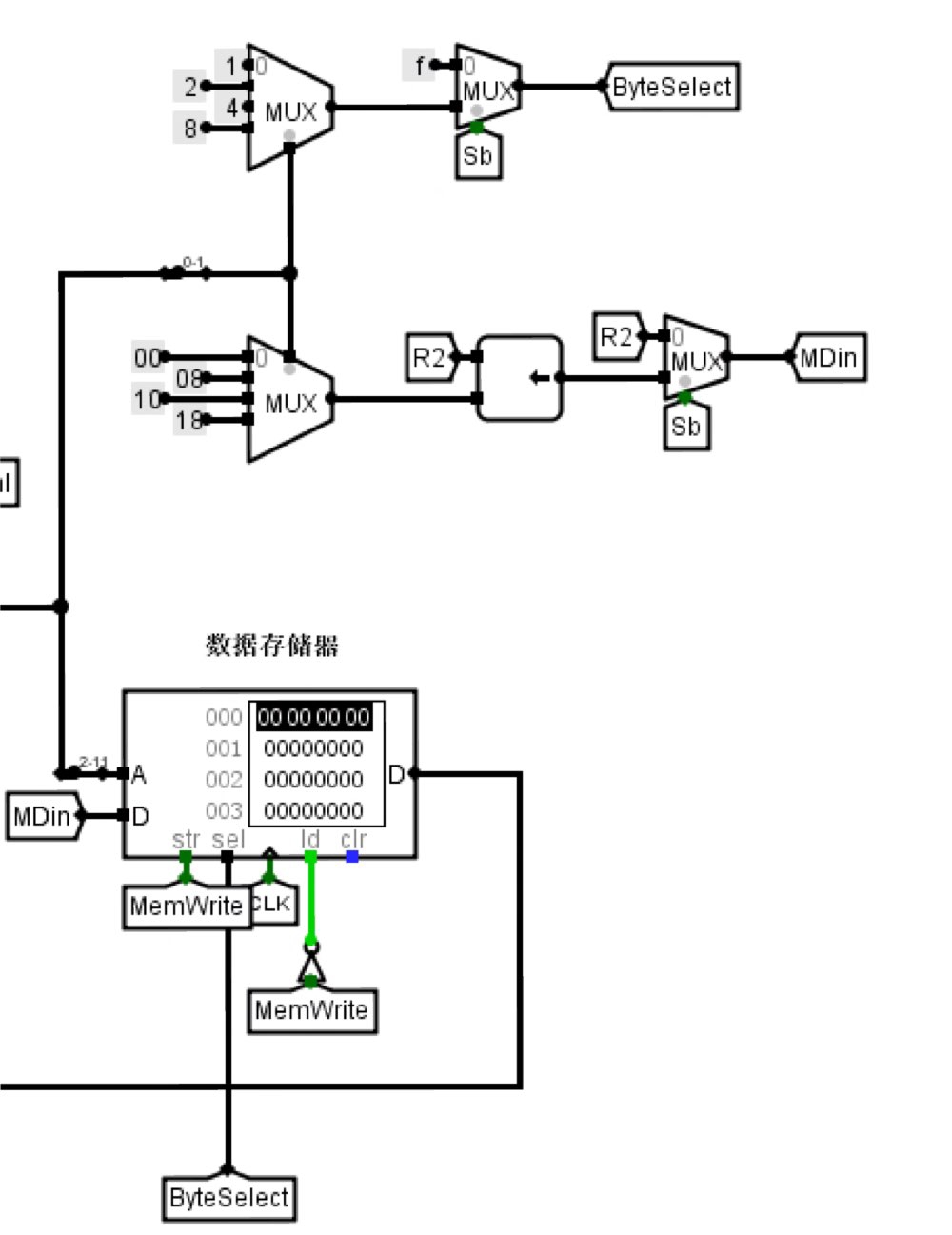


图 3.2 数据配合移位

1. FPGA实现

数据通路的FPGA实现非常繁杂，要理清楚CPU的线路关系，给每根线提前取好名字，然后按照电路图进行编程。

首先定义好整个CPU的输入输出端口，如下。

module CPU\_RISCV(CLK,CLK\_Sel,Display\_Sel,Go,RST,SEG,AN,halt);

//输入输出

input CLK;//开发板上的时钟源100MHz

input RST;//复位信号

...//其他

接着按照事先的命名定义好线路，如下。

//必要线路

wire [31:0] PC\_in,PC\_out,PC\_add\_4,IR;

wire [4:0] Funct,OpCode;//输入硬布线控制器中的信号

wire [3:0] ALUOP;//运算信号

...//其他

线路中的一些逻辑使用assign语句来实现，包括与门或门分线器等等，如下。

//线路逻辑

assign PC\_en=Go|(~halt);//生成PC使能信号

assign PC\_add\_4=PC\_out+4;//生成PC+4

//输入硬布线控制器中的信号

assign Funct={IR[30],IR[25],IR[14:12]};

...//其他

实例化功能部件，将已经定义好的各种线路连接上去，如下。

//实例化模块

divider #(50000000) divider1(.clk(CLK),.clk\_N(CLK\_1));//生成1Hz的时钟信号

counter\_32 counter1(.clk(CLK\_N), .out(PeriodNum),.halt(halt),.RST(RST));

BinToBCD BinaryToBCD(.Bin(PeriodNum),.BCD(BCD));

...//其他

另外，数据通路还涉及数码管显示逻辑，这一部分直接照搬verilog课的就行。

### 控制器的实现

1. Logisim实现

根据控制器的设计思路，将逻辑表达式直接喂给logisim并自动生成逻辑电路。

1. FPGA实现

控制器的实现只需要将信号控制生成器和运算控制器弄到一起并封装起来，而这两个控制器直接用if-else语句一条一条罗列即可。以生成信号控制器为例，如图 3.3。

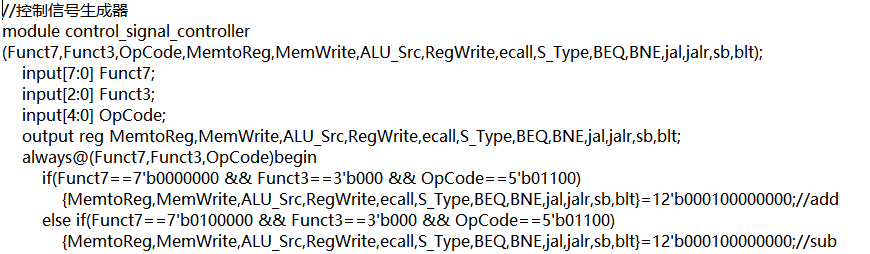


图 3.3 生产信号控制器verilog

## 中断机制实现

根据单级中断机制设计，需要实现的部件有中断信号采样实现电路、中断响应实现电路、中断使能状态电路、EPC 断点保存电路以及中断返回实现电路。

### 单级中断实现

### 中断响应实现

中断信号产生时，优先编码器选出的优先级最高的中断号被送入中断号寄存器。该中断号作为多路选择器的sel端将选出对应中断的跳转地址（跳转地址是使用RARS查看的）并送入PC。具体实现如图 3.4。

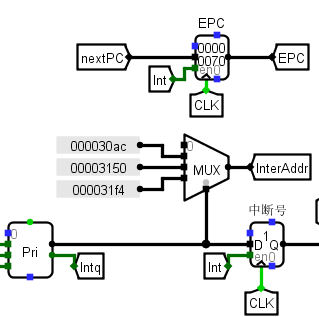


图 3.4 中断号相关硬件

### 中断使能状态实现

保存中断使能状态只需要设置一个IE寄存器，其值为1时表示开中断，反之为关，初始状态为1。中断响应或者中断返回时，寄存器器状态才会被改变，因此将Int和uret信号相或后作为IE的使能端，IE的具体值。具体实现如图 3.5。

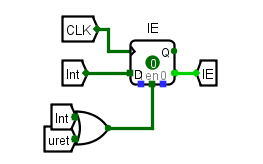


图 3.5 中断使能寄存器实现

### EPC断点保存实现

使用EPC寄存器来保护断点。中断信号到来时，PC本来的下一跳地址将被存入EPC寄存器，如图 3.6。

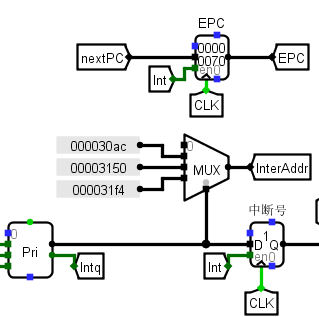


图 3.6 EPC寄存器实现

### 中断返回实现

中断返回时，uret信号会变1，将其与中断号寄存器的译码信号相与得到当前中断的返回信号。此外，uret还有开中断，送回断点地址，清除中断请求信号的功能，具体实现如图 3.7。

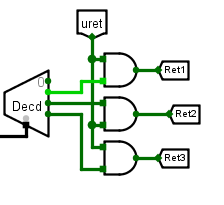


图 3.7 中断返回实现

### 多级中断实现

与单级中断相比，多级中断需要添加屏蔽字判断能否被打断，其使能状态条件更也多且需要硬件堆栈保护多级中断的断点。

1. 中断屏蔽字实现

低优先级中断不能打断高优先级中断，因此需要设置中断屏蔽字，由中断请求信号、IE状态信号和shield信号协同产生中断信号。将当前正在执行的中断号放在栈中，通过比较当前中断号和刚到来的中断号来决定是否对其进行屏蔽。若刚到来的中断请求信号优先级更高，那么将该中断号入栈，同时响应该中断的服务。每次响应中断请求时，总是用栈顶的中断号来进行服务。具体实现如图 3.8。

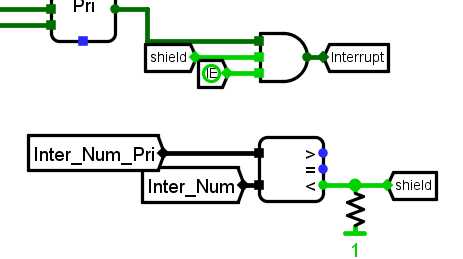


图 3.8 中断屏蔽字实现

1. 中断使能状态条件调整

如今有4个能影响寄存器器状态的信号，增加了csrrci和csrrsi这两个信号。需要注意的是csrrci、csrrsi、uret这三个信号互斥出现。通过列表发现，IE寄存器的值可以用interrupt|csrrci来表示。具体实现如图 3.9。

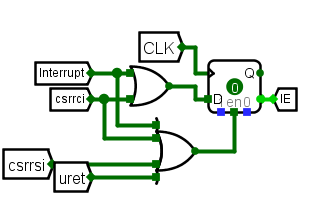


图 3.9 中断使能状态条件调整

1. 硬件堆栈实现

硬件堆栈由3 个寄存器来实现。其中栈指针由计数器来实现。中断响应时，栈指针指向的寄存器使能端置1，数据入栈。中断返回时，栈指针减一。栈指针始终指向正在响应的中断的相关数据。要注意的是入栈与读栈时，栈指针差一。具体实现如图 3.10。

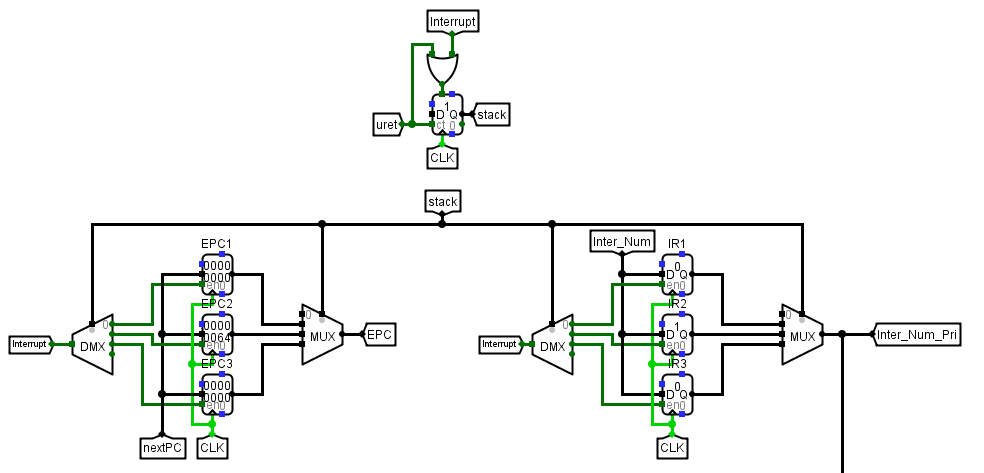
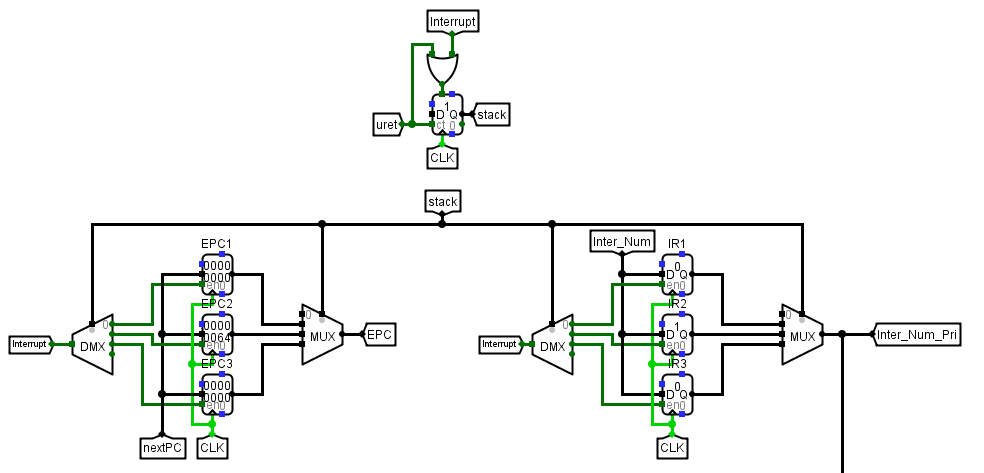


图 3.10 硬件堆栈实现

### 流水线中断实现

我选择在EX段实现单级中断。关键之处在于采用与处理分支指令类似的方式，即只要产生中断信号Int,便清空IF/ID和ID/EX这两个流水寄存器，保证产生中断的瞬间IF、ID和EX段的指令不执行。保存这一瞬间的最后一段有效指令（即不为气泡）地址作为断点。有效指令识别的实现如图 3.11。

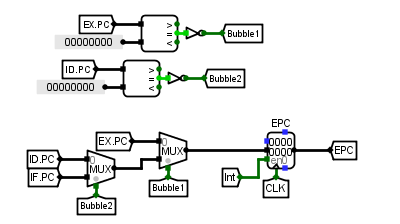


图 3.11 有效指令识别逻辑

## 流水CPU实现

### 流水接口部件实现

流水寄存器是将若干个寄存器封装在一起，共用CLK时钟控制端、RST清零端和EN使能端这三个控制信号。每个寄存器在流水寄存器上有一个成对的端口，用于输入或输出其存储数据。。

### 理想流水线实现

将单周期CPU根据功能段进行分块，然后将每块使用流水线接口部件进行拼接。

## 气泡式流水线实现

3.4.1**控制冲突解决**

需要在EX段检测是否出现分支指令跳转，无条件分支指令出现即跳转，条件分支指令是否跳转还依赖于ALU的信号。条件分支跳转检测逻辑如图 3.12。

将条件跳转信号B与无条件跳转信号JALR、JAL相与过后连接到IF/ID、ID/EX的复位端实现清零功能。

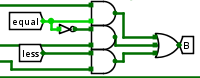


图 3.12 条件分支跳转检测

3.4.2数据冲突解决

ID 与 WB 的数据冲突只需要将寄存器文件设置为先写后读（下跳沿写入）。

ID段与EX、MEM段的冲突要在ID段增加数据相关检测逻辑。发生数据冲突一定涉及到从寄存器中取数，那么判断是否出现数据冲突需要先判断ID段指令译码后的R1、R2寄存器是否有效（有些指令不涉及寄存器），填表后让logism自动生成该判断逻辑。当ID段寄存器有效且非零，且EX段或MEM 段要进行写寄存器（RegWrite为1），且被写的这个目的寄存器的编号与ID段指令译码出来的某一个寄存器编号相同，则产生冲突，输出数据相关。具体实现如图 3.13。

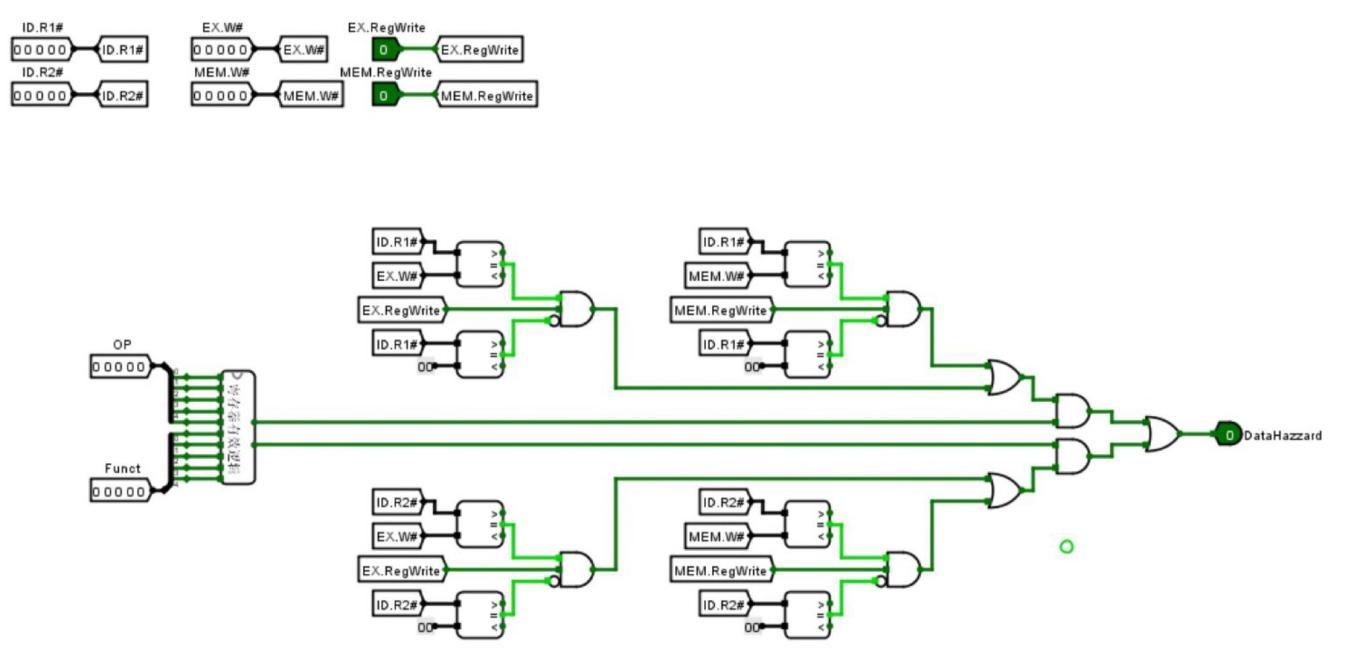


图 3.13 数据冲突检测逻辑

将DataHazzard信号接到ID/EX的复位端实现清空ID/EX段的功能，将DataHazzard信号取反接到PC和IF/ID的使能端，实现阻塞功能。

## 重定向流水线实现

### 重定向数据通路增加

### 在ALU的输入前进行对R1和R2的值的重定向，Forward信号为0时，传递原数值；为1时，将WB段的写回数据重定向到输入端；为2时，将MEM段的写回数据重定向到输入端。具体实现如图 3.14。

### 20231022231906

图 3.14 重定向数据通路

3.5.2 Forword信号产生

改造数据相关检测逻辑，使得其能具体地指明ID段到底是和后面哪一段产生了数据冲突。用两位的Forward记录3种冲突状态：0为不冲突，1为与MEM段冲突，2为与EX段冲突。要是同时与两段均发生冲突，那么EX段的优先级更高，即始终要用最新的那个数据。具体实现如图 3.15。

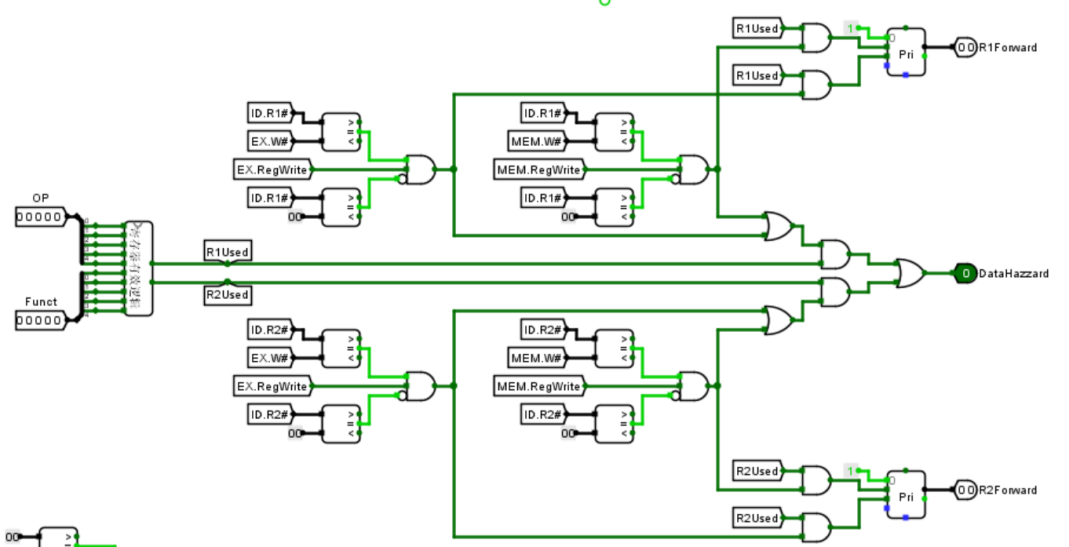


图 3.15 Forword信号产生

3.5.3 Load-Use相关的数据冲突检测

Load-Use是ID段与EX段数据相关的一种特例，即涉及访存，因此可以将其和MemToReg信号相与来生成Load\_Use信号。Load\_Use信号接到ID/EX的复位端实现清空ID/EX段的功能，取反接到PC和IF/ID的使能端，实现阻塞功能。具体实现如图 3.16。

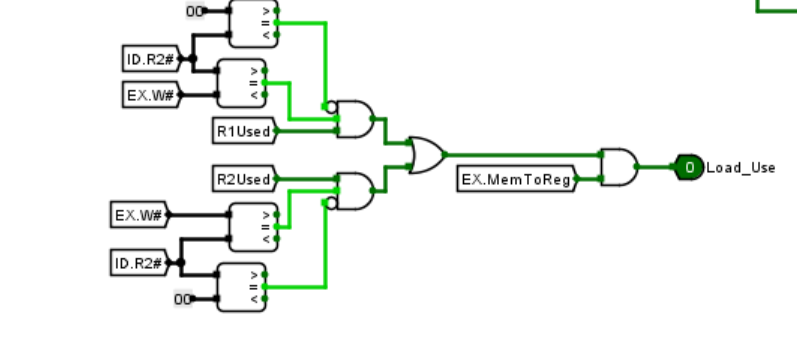


图 3.16 Load-Use相关的数据冲突检测

## 动态分支预测机制实现

### PC来源通路调整

根据设计时总结的表来实现出PC值的来源选择的通路，具体实现如图 3.17。

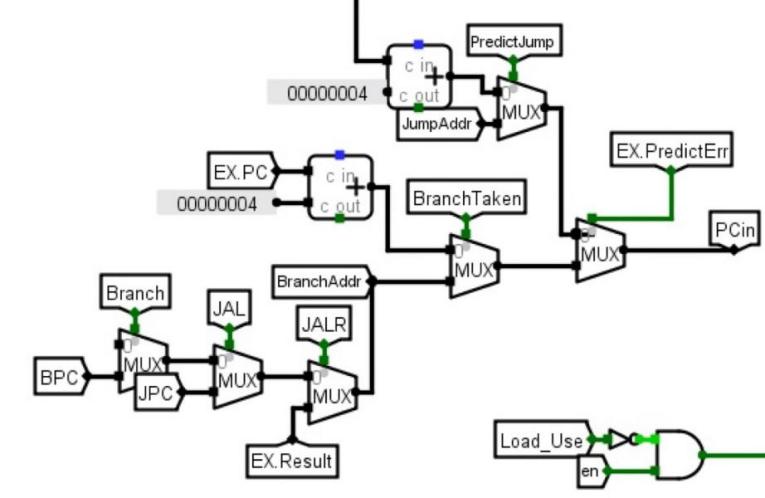


图 3.17 PC来源通路调整

### BTB表cache行实现

根据设计时的cache行组成，使用寄存器来实现cache行，其中的write和L信号的接法将在后面阐述。清零处采用D寄存器以防止毛刺。具体实现如图 3.18。

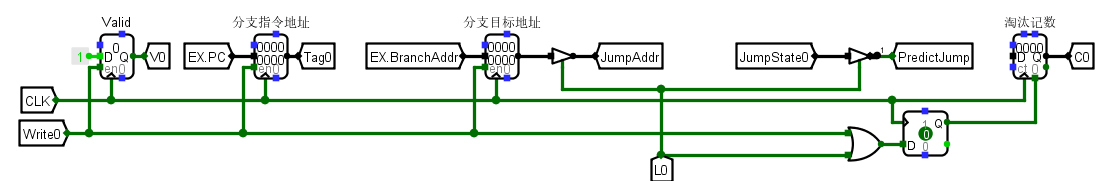


图 3.18 cache行实现

### BTB表读逻辑实现（支持IF段预测并更新淘汰计数位）

用IF.PC与各个cache行的Tag进行全相联比较，相等且有效位为1时生成读信号。读信号打开相应cache行的三态门传递分支目标地址、预测跳转信息位，并将淘汰计数位清零。具体实现如图 3.19，共8行，这里只截取了两行。

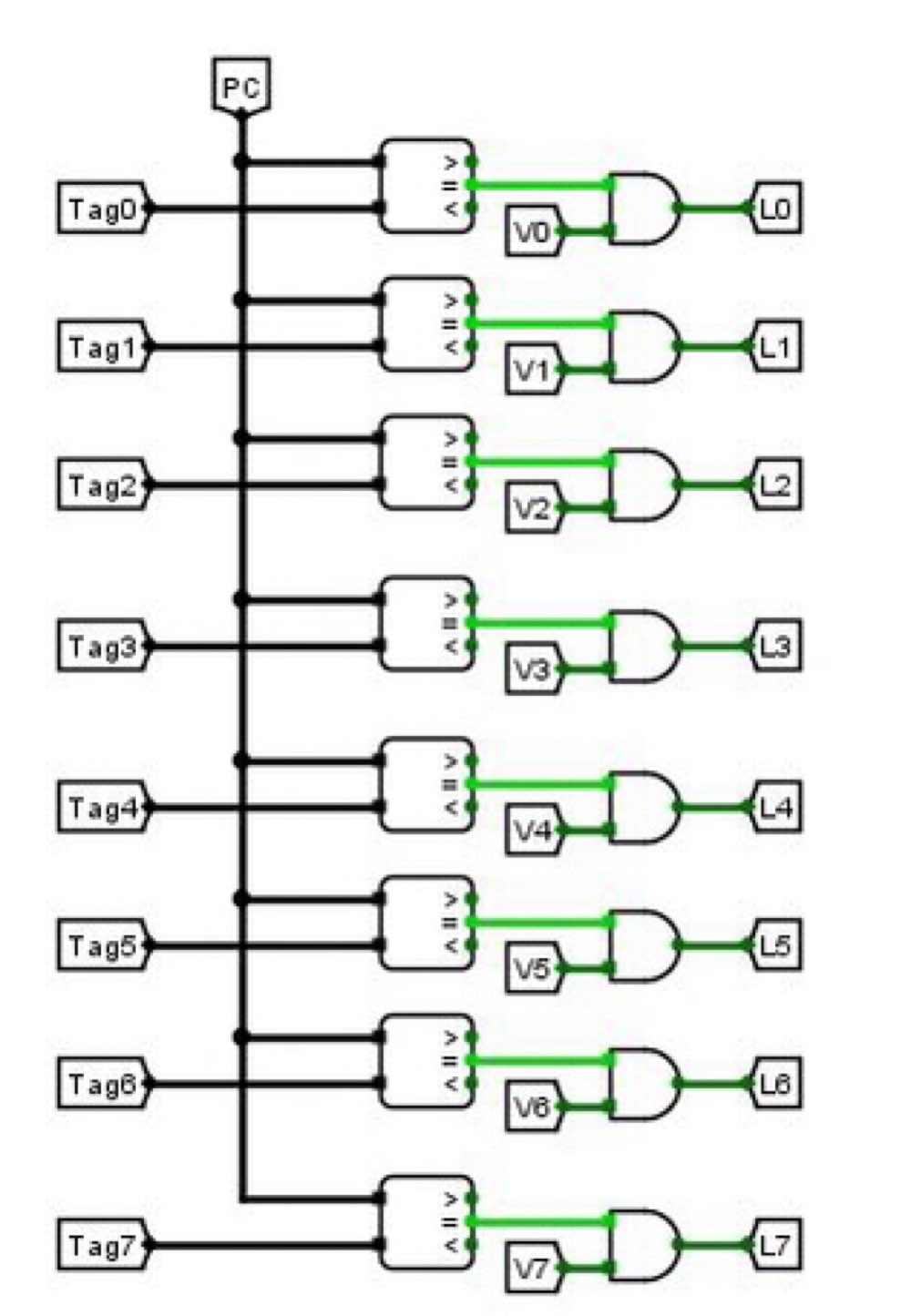


图 3.19 BTB表读逻辑

### BTB表写逻辑实现（支持EX段缺失时进行分支信息写入）

每一条分支指令在EXE段执行是（即Branch信号为1），将EX.PC与各个cache行的Tag进行全相联比较，相等且有效位为1时生成Hit信号，否则生成Miss信号。一旦出现Miss信号，则需要向cache行中载入该分支的相关信息。至于用哪个cache行则要看槽满没满。淘汰计数位使用LRU算法在cache槽满后对cache行进行淘汰替换。具体实现如图 3.20。

### 36A72F7898169F8A6C8D4760ADE68F63 6295DBFFE14140FF32CB3DF7588D1355

图 3.20 BTB表写逻辑

### BTB表JumpState逻辑实现（EX段命中时更新分支预测历史位）

分支预测历史位的状态转移如图 3.21，研究表明双预测位可在较低的成本下实现很高的预测准确率。每当cache中存储的分支指令执行到EX阶段且命中时就会对分支预测历史位进行一次更新。确定下一个状态需要知道现态和分支是否真的跳转，该状态机的逻辑实现如。其中多路选择器根据BranchTaken和JumpState来选择最终的次态。注意每个cache行都带有一个分支预测历史位。具体实现如图 3.22。

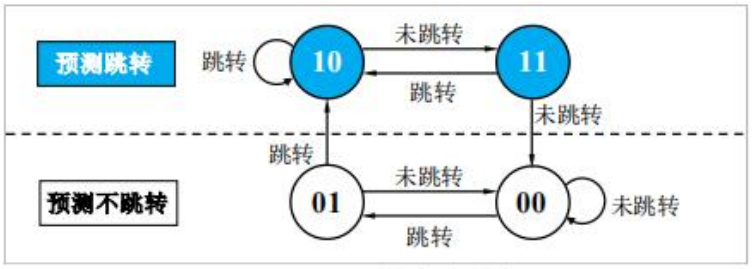


图 3.21 分支预测历史位状态转移

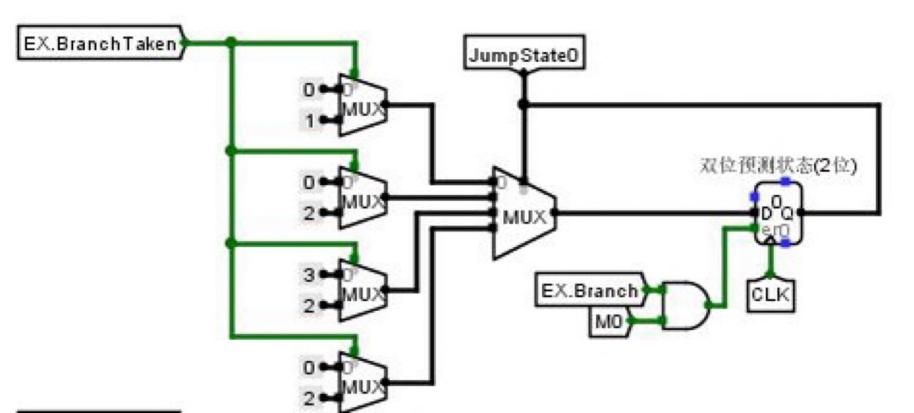


图 3.22 BTB表JumpState逻辑实现

# 实验过程与调试

## 测试用例和功能测试

用测试用例risc-v-benchmark\_ccab来测试各种CPU（单周期、气泡、重定向、动态分支预测）的指令实现情况。用测试用例risc-v中断测试程序（走马灯）来测试单级和多级中断，流水线中断实现为单级中断，测试省略。理想流水线为一个基本架构，在测试气泡流水线与重定向流水线时其实就包含了对理想流水线的测试，故省略。

### 测试用例risc-v-benchmark\_ccab

将带risc-v-benchmark\_ccab.hex载入指令存储器。基础命令测试完毕后会运行到第一个停机的ecall指令，点击继续运行按钮，可以依次测试4条扩展指令，LED 显示符合预测值，测试成功。

以下为risc-v-benchmark\_ccab.hex在单周期（图 4.1）、气泡（图 4.2）、重定向（图 4.3）、动态分支预测（图 4.4）的运行结果。扩展指令都已经线下检测过了，也没什么好写的，省略。

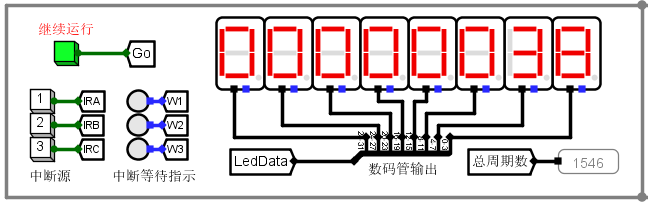
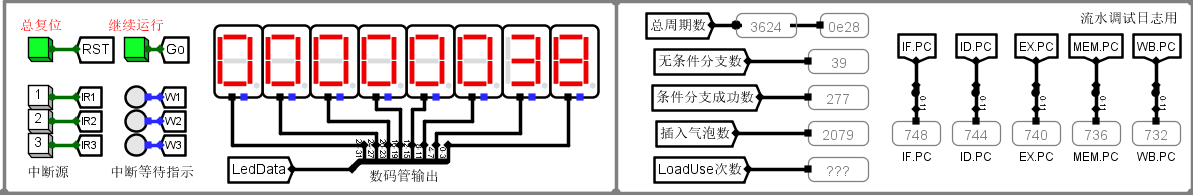
 

图 4.1 单周期CPU测试 图 4.2 气泡流水线CPU测试

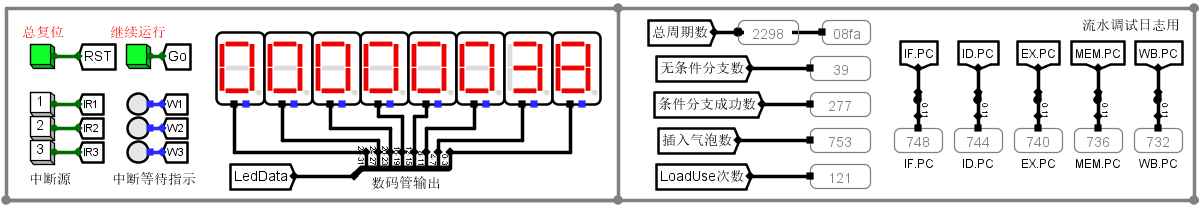
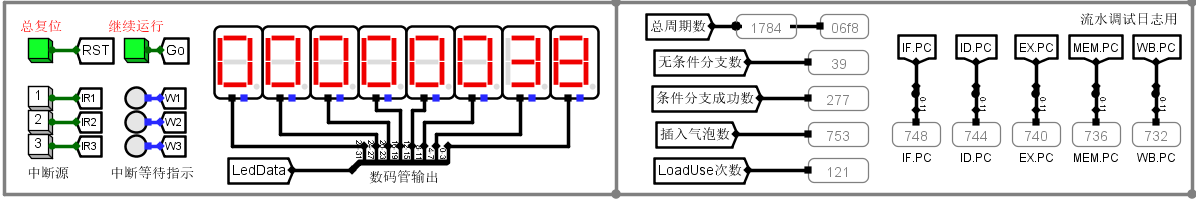
 

图 4.3 重定向流水线CPU测试  图 4.4 动态分支预测测试

### 测试用例risc-v中断测试程序

1. 多级中断测试

1号中断进行时，依次按下按钮2与按钮3，1号中断被打断，程序进入3号中断，如图 4.5。3号中断返回后，进入2号中断，如图 4.6。2号中断返回后，进入1号中断，回到被打断的位置继续运行。

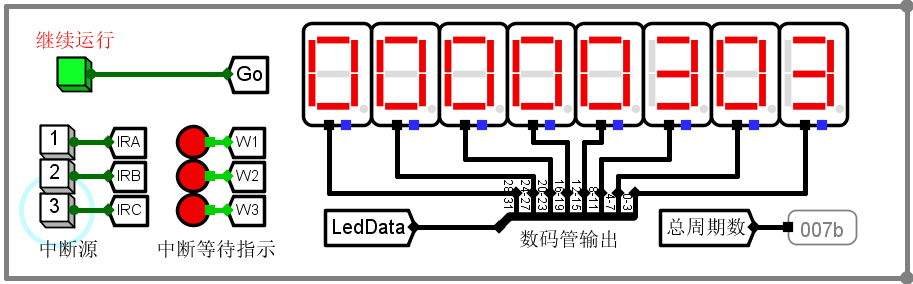


图 4.5 多级中断测试1

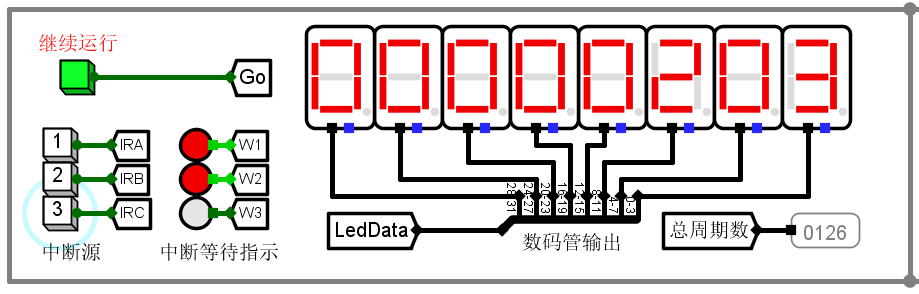


图 4.6 多级中断测试2

## 性能分析

各种CPU运行不带ccab的benchmark程序的总周期数如表 4.1。

表 4.1 benchmark运行总周期数统计

| CPU | benchmark程序运行总周期数 | CPU | benchmark程序运行总周期数 |
| --- | --- | --- | --- |
| 单周期 | 1546 | 重定向流水线 | 2298 |
| 气泡流水线 | 3624 | 动态分支预测 | 1784 |

单周期CPU总周期数为1546，但是每周期长度较长，估算可认为其时钟周期约为5段流水CPU时钟周期的5倍，所以单周期CPU性能远远差于5段流水CPU。

对比各种流水线CPU：气泡流水线性能较差，运行总周期数为3624。重定向流水线优化了数据冲突时延，运行总周期数为2298，比气泡流水线减少1/3。动态分支预测则优化了分支冲突时延，运行总周期数为1782，比重定向流水线又减少了1/4。

## 主要故障与调试

### ecall指令的停机问题

**故障现象：**无CCAB的常规程序执行到最后那个本该暂停的ecall指令时停不下来。在头哥上进行测评得到结果：PCerr 11、Werr 01000。经查发现PC期望为2dc，但实测是2e0；RegW期望是0，但实测为1。

**原因分析：**分析头哥报错中PC期望值与实测值的差别，发现PC继续向后面取了一个，那么应该是halt信号出了问题。halt信号的产生如图 4.7，当R2（$a7）的值等于0x22时，halt信号寄存器的输入端便为1，等到下一个时钟周期来临，halt信号便被置一，问题就出在这个下一个时钟周期。在这里放一个寄存器是为了实现Go功能，寄存器默认是上升沿有效。上升沿有效的一个问题就是，当前时钟周期内执行ecall指令，halt寄存器的输入端被置1，但是halt信号没被置1，当下一个时钟上升沿来临时，PC的值会和halt信号的值同步被更新，所以导致本该暂停的PC又往下去了一个。

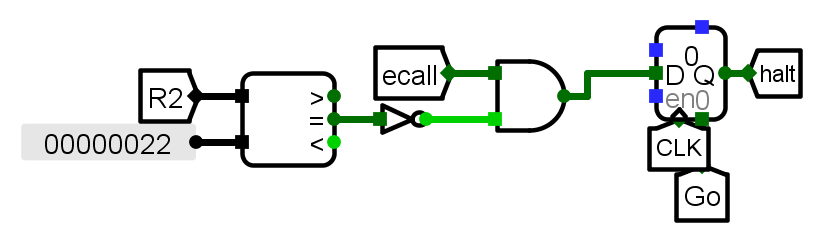


图 4.7 halt信号的产生

**解决方案：**将存储halt信号的寄存器设为下降沿有效。

### halt信号生效问题

**故障现象：**部分指令无法完成，停留在流水线中。

**原因分析：**ID由ecall信号产生halt信号，如果在EX段直接执行停机，则会导致后续**MEM**段和**WB**段的指令停留在流水线中，没有执行完毕。

**解决方案：将**halt信号通过流水线一直向后传递到WB段，再反馈到PC和流水线寄存器执行停机**。**

### 头哥上面的一个奇怪故障

**故障现象：**在头哥上面测试气泡流水线时，头哥报错说第16拍分支指令跳转的PC值不对，然而去看汇编却发现就应该是报错的这个值。

**原因分析：**在交流群看见有人跟我有一样的错误，我们推测是因为hex文件中加了ccab的缘故，这导致指令存储器里面某些指令的PC就变了。

**解决方案：**删掉ccab。

## 实验进度

表 4.2 课程设计进度表

| 时间 | 进度 |
| --- | --- |
| 第一天 | 复习单周期CPU的相关理论知识，阅读课设ppt，根据课设任务书上面的步骤，完成了部分指令数据通路的构建。 |
| 第二天 | 查阅RISC\_V指令手册，进一步熟悉各种指令的类型及功能，完善了任务书上没有提到的指令类型的数据通路。另外，填好了单周期CPU的控制信号表，弄好了控制器。 |
| 第三天 | 对CPU进行联调并最终通过了头哥测试。完成了单周期CPU部分的实验报告。 |
| 第四天 | 学习5段流水CPU的相关理论知识，完成流水寄存器的设计与部分实现。 |
| 第五天 | 实现了四个流水寄存器的搭建，完成了理想流水线的数据通路，调试并排除故障，通过头哥测试。完成了实验报告关于理想流水线的部分。 |
| 第六天 | 完成气泡流水线的搭建，调试并通过头哥。 |
| 第七天 | 休息。 |
| 第八天 | 完成重定向流水线的搭建，调试并通过头哥。 |
| 第九天 | 完成流水线部分的实验报告。复习跟中断相关的理论知识，熟悉中断过程。 |
| 第十天 | 完成单级中断机制的实现。 |
| 第十一天 | 完成多级中断机制的实现。 |
| 第十二天 | 完成流水线中断并完成关于中断的实验报告。 |
| 第十三天 | 复习verilog并完成CPU各个功能部件的编写，对各个部件进行了仿真测试。 |
| 第十四天 | 使用Verilog完成单周期CPU数据通路的连接。最终CPU通过Vivado的模拟仿真，并成功将生成bit流烧入FPGA板内实现预计功能。 |

# 团队项目设计与实现

## 团队选题

团队实验任务选题为"坦克战争"，如图 5.1：在32\*32的网格屏幕当中，由两名选手分别使用键盘上的不同按键控制左右的两个坦克上下移动和彼此攻击，双方坦克最开始都有5条生命值，坦克在屏幕中显示为高3宽2的三角形（用三个方块代表车声，一个突出的方块代表炮管），当坦克侧体被攻击时会受到1点伤害，当坦克主体被攻击时会受到2点的暴击伤害。

为了增强游戏的可玩性，我们还设置了5颗炮弹的上限限制，如果消耗了炮弹，炮弹不会瞬间恢复，而是每隔一段时间增加一颗。双方的炮弹如果在行进过程中与对方的炮弹碰撞则会抵消消失。

图形用户界面

描述已自动生成

图 5.1 游戏界面最终效果（游戏过程见视频）

## 各功能板块连接

首先将键盘输入连到键盘逻辑当中，对于特定的按键解码返回不同的信号，再将与坦克移动相关的信号输入到单周期CPU当中去，在这里我们可以实现对于坦克的上下移动操作，而与子弹发射相关的键盘输入信号则会被输入到子弹数处理逻辑元件当中去，这将根据目前坦克的剩余子弹数决定是否产生一个发射子弹的信号，之后将双方坦克目前所处的位置，先前子弹状态和是否开火信号传入子弹处理逻辑元件当中，这个元件负责每一个clk中子弹的移动以及新发射的子弹的产生，因为要维护每颗子弹的移动，所以我们在每一排使用了两个32位的bit，分别表示在这一排当中左边坦克发射的子弹和右边坦克发射的子弹的位置。同时上述信号可以接入碰撞处理逻辑元件当中去，碰撞处理元件将根据目前所有子弹的状态和坦克目前的位置判断坦克是否受到攻击，并更新坦克的剩余血量。最后将每一排的子弹和坦克所在的行数传入显示逻辑即可。

## 功能模块实现

对于显示逻辑处理，如图 5.2。我们只需要知道坦克所处的位置，就可以在其对应的行将其对应的bit设置为1，对于受到伤害时的闪烁效果，我们额外设置了一个全局的计数器，并令其上限为5且达到上限时不再计数，每当受到伤害时，将计数器清零，而我们令坦克只有在计数器计数值为奇数时才会显示，这样一来我们就实现了受到伤害后闪烁3次的逻辑。

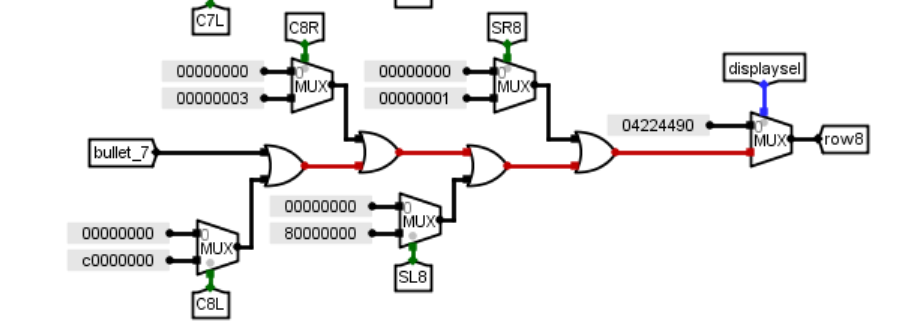


图 5.2 显示逻辑电路

对于碰撞检测逻辑，如图 5.3，只需要将坦克的显示bit与对手子弹的显示bit进行与操作，如果结果不为0则判定坦克与子弹碰撞，其部分电路如图所示。检测到碰撞后，将被碰撞坦克的生命值减1，并输出碰撞信号使其产生闪烁效果。

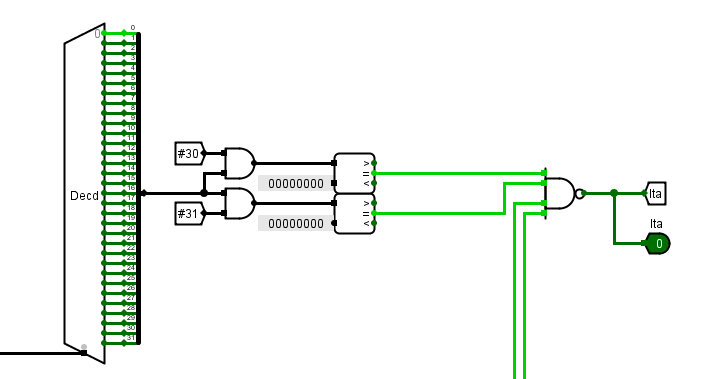


图 5.3 碰撞检测逻辑电路

对于子弹处理逻辑，如图 5.4。我们令x为一个32bit数，每个bit为1代表一个向右移动的子弹，同理令y代表这行向左移动的所有子弹，再令x’为下一刻向右移动的子弹状态，那么我们可以使用来实现整排子弹的向右移动以及子弹之间的相互抵消。对于y我们也可以使用同样的公式维护其左移和抵消，而新子弹的产生也很简单，只需要将对应得位置bit设置为1再或上这排子弹先前的状态即可。

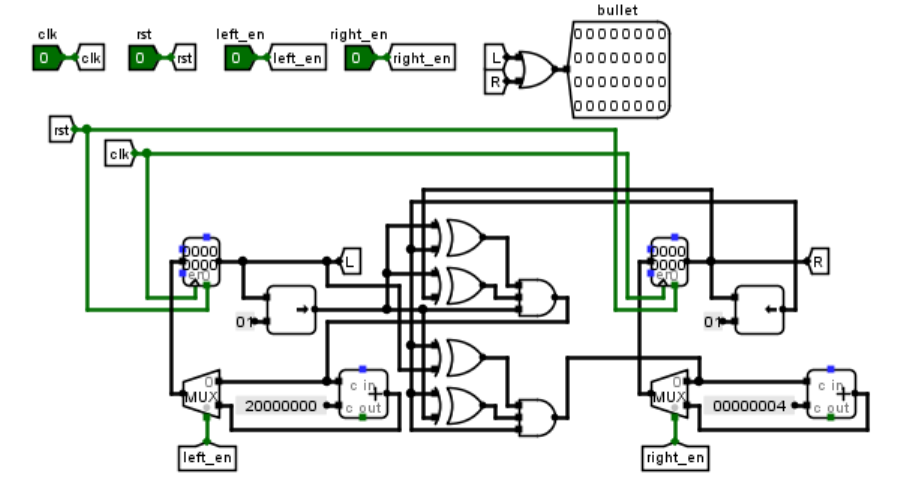
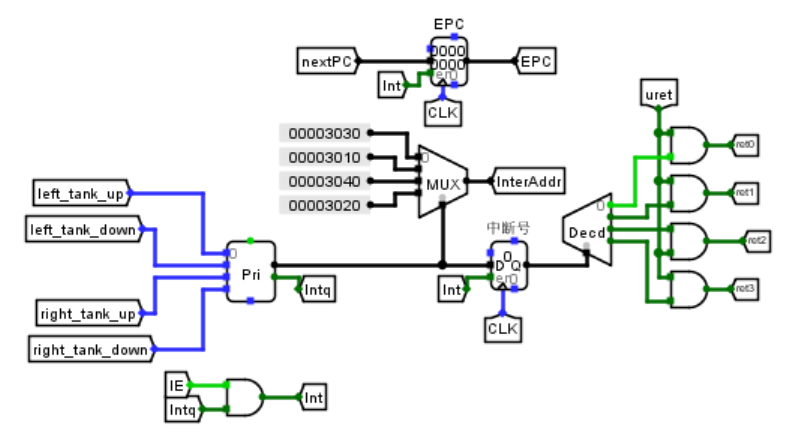


图 5.4 子弹处理逻辑电路

坦克移动的实现如图 5.5，图 5.6，我们令a0和a7分别存储左右坦克的初始行数，然后分别定义了左右坦克上下移动的4个中断处理程序，注意使用blt特判坦克的位置不能移出边界。

文本

描述已自动生成 图 5.5 单周期CPU中断处理 图 5.6 单周期CPU执行程序

## 团队分工

表 5.1 团队分工

| 姓名（按姓氏排序） | 任务 |
| --- | --- |
| 付余 | 碰撞检测逻辑，生命值记录，音效 |
| 胡伟江 | 子弹发射逻辑，子弹对冲逻辑，系统时钟分频 |
| 刘轩灵 | 键盘与中断，汇编代码，综合联调 |
| 庄景豪 | 坦克和子弹显示逻辑，击中闪光 |

# 设计总结与心得

## 课设总结

在本次硬件综合训练中，我完成了如下几点工作：

1. 设计实现了支持24条基础指令和4条扩展指令的单周期CPU。
2. 设计实现了理想流水线、气泡流水线和重定向流水线这三种5段流水CPU。
3. 设计实现了单周期CPU的单级中断、多级中断和5段流水CPU的单级中断。
4. 设计实现了包含动态分支预测机制的重定向流水线CPU。
5. 完成了单周期 CPU的verilog上板。
6. 完成了educoder平台的测试与线下的检查。
7. 设计实现了团队任务“坦克大战”，完成了PPT与答辩视频的制作。

## 课设心得

### 课设体会

1. 熟悉掌握了CPU的设计全流程

从学习各种指令的格式及功能、设计单周期CPU的数据通路、实现硬布线控制器，到后面的构建5段流水CPU、实现中断机制，我对组成原理知识的理解越来越深刻，对CPU的设计也越来越熟练。甚至感觉现在随便给我一个指令集，我就能快速搞出一个玩具CPU。

1. 深刻理解了CPU设计背后的迭代思想

无论是构建5段流水CPU还是实现中断机制，背后都是由浅入深，层层递进的思想。理想流水线漏洞百出，气泡流水线有用但低效，重定向流水线优化数据冲突时延，最后动态分支预测机制优化分支冲突时延。多级中断在单级中断的基础上增添了屏蔽字判断能否被打断，增添了更多的使能状态条件，增添了硬件堆栈保护多级中断的断点。

1. 学会了给自己制造“黑箱”

当我试图使用Verilog“翻译”我的Logism电路时，陷入了茫然无措的境地：线路过多且没有名字，隧道标签东一个西一个。而Verilog里面要用wire型变量代替Logism中可视化的连线。最开始我一边定义wire变量，一边思考CPU 的各个连线，导致写出的程序漏洞百出，重复定义了很多变量或者遗漏了不少连线。这时候我想起来上学期学Verilog语言时，我都没去搞懂那个电路有什么意义，光凭老师画的连线图就能轻松写出程序。这背后的原因就是Verilog老师给我们制造了一个“黑箱”：你不用管电路为什么这么连，有什么作用，你只需要按照这些线路的名称和连接方式把程序写出来就行。制造“黑箱”能让我们的思维单线程化，能使我们的行为更加清晰而不用四处纠结。因此，我仿造老师的方法，给每条连线挨个命名，并且优化掉部分结构，弄出新的电路模样，然后忘掉它是一个CPU，直接根据这个图写Verilog程序，很快便实现完成。

### 课设建议

1. 给同学们树立信心

我觉得本次课设并不像学长们说的那样是地狱难度，它反而是我在华科做过的最顺利，体验最好的一个课设。课设的前五十分完全是上学期的实验内容，这使得我们稍稍复习便能迅速上手，课设的其他内容也有大量的资料可供参考查阅。所以，我认为在课设开始前，就要给大家建立一个这样的心态：这次课设并不难，只要耐下心来，认认真真阅读任务书，踏踏实实连线，就肯定能做完大部分内容。学长们很多时候过分渲染了这个课设的难度，这让同学在课设刚开始会产生畏难心理，总觉得自己不会做，所以要去抄抄其他人的，这无疑浪费了这么好的实验资源。

1. 关于课程时间安排

首先课程总时长两周，其实时间对大多数同学来说是有一点紧张的。我认为这个课设的目的是让绝大多数同学都能通过实践熟悉和掌握设计CPU的一个全流程，而不是为了拉开同学们的区分度。所以，尽可能地让更多同学更好地完成课设才是最终的一个目的。因此我认为实验的总时间可以适当延长到四周左右，这样也能避免同学们为了赶进度而敷衍了事。事实上，本实验的检查也一拖再拖，好像到第四五周才检查完。

在时间分配上也可以做适当的调整。刚入手进行单周期和流水线的设计时，大家几乎都是摸着石头过河，相对来说经验不足，可以多给一点时间。当实验进行到中断和动态分支预测等电路，因为已经有了前面的经验和基础，速度可以稍稍加快。

1. 可以要求提交周总结报告

我发现很多同学都没有边做实验边写报告的习惯，包括我自己有时候也会有些偷懒，所以可以要求一下大家提交周总结报告。这个报告仅仅是督促一下大家一定要边做实验边写报告。这样不仅有利于老师观察大家的实验进度，也能提高大家的效率，将报告拆分成很多部分，最后拼接一下就成最后的完整报告了。这个周总结报告没有任何要求，就交一些做实验的过程记录（包括设计思路，问题解决还有错误记录等等）。

1. 收集疑问汇总

很多时候做实验卡住了或者出现一些莫名其妙的问题时，组原群里面总是能看见跟我遇到一样问题的同学。我觉得可以将有些莫名其妙的问题慢慢收集起来整理成一个小册子备查，这样不仅能帮助同学们迅速找到问题点，也能减轻老师的答疑负担，减少重复提问。

1. 可以给一个已经完成了大部分接口的流水线寄存器

流水线接口的设计其实非常简单，没什么值得思考的地方，但是复制粘贴寄存器，给接口改名，调整接口位置非常耗费时间。我认为可以给一个已经完成了大部分接口的流水线寄存器，然后同学们只需要在这个基础上增添自己需要的部分接口，这样能减少任务中重复繁杂的工作，以提高整个课程设计的效率。

# 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，周军龙，肖亮. 计算机组成原理实验指导与习题解析.北京：人民邮电出版社，2022.
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: 刘轩灵** |