***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： | ACM1901 |
| 学 号： | U201915035 |
| 姓 名： | 邹雅 |
| 电 话： | 15058667378 |
| 邮 件： | [1542527211@qq.com](mailto:1542527211@qq.com) |

目 录

[1 课程设计概述 3](#_Toc101460946)

[1.1 课设目的 3](#_Toc101460947)

[1.2 设计任务 3](#_Toc101460948)

[1.3 设计要求 3](#_Toc101460949)

[1.4 技术指标 4](#_Toc101460950)

[2 总体方案设计 6](#_Toc101460951)

[2.1 单周期CPU设计 6](#_Toc101460952)

[2.2 中断机制设计 11](#_Toc101460953)

[2.3 流水CPU设计 13](#_Toc101460954)

[2.4 气泡式流水线设计 15](#_Toc101460955)

[2.5 重定向流水线设计 16](#_Toc101460956)

[3 详细设计与实现 17](#_Toc101460957)

[3.1 单周期CPU 实现 17](#_Toc101460958)

[3.2 中断机制实现 20](#_Toc101460959)

[3.3 流水CPU实现 23](#_Toc101460960)

[3.4 气泡式流水线实现 24](#_Toc101460961)

[3.5 重定向流水线实现 25](#_Toc101460962)

[4 实验过程与调试 28](#_Toc101460963)

[4.1 测试用例和功能测试 28](#_Toc101460964)

[4.2 性能分析 32](#_Toc101460965)

[4.3 主要故障与调试 33](#_Toc101460966)

[4.4 实验进度 33](#_Toc101460967)

[5 团队任务 35](#_Toc101460968)

[5.1 团队任务概述 35](#_Toc101460969)

[5.2 图片显示模块实现 35](#_Toc101460970)

[6 设计总结与心得 37](#_Toc101460971)

[6.1 课设总结 37](#_Toc101460972)

[6.2 课设心得 37](#_Toc101460973)

[参考文献 39](#_Toc101460974)

# 课程设计概述

## 课设目的

计算机组成原理是计算机专业的核心基础课。该课程力图以“培养学生现代计算机系统设计能力”为目标，贯彻“强调软/硬件关联与协同、以CPU设计为核心/层次化系统设计的组织思路，有效地增强对学生的计算机系统设计与实现能力的培养”。课程设计是完成该课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有一定规模的指令系统的简单计算机系统。所设计的系统能在LOGISIM仿真平台和FPGA实验平台上正确运行，通过检查程序结果的正确性来判断所设计计算机系统正确性。

课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行中央处理器底层电路的实现、故障分析与定位、系统调试等环节的综合锻炼，进一步提高学生分析和解决问题的能力。

## 设计任务

本课程设计的总体目标是利用FPGA以及相关外围器件，设计五段流水CPU，要求所设计的流水CPU系统能支持自动和单步运行方式，能正确地执行存放在主存中的程序的功能，对主要的数据流和控制流通过LED、数码管等适时的进行显示，方便监控和调试。尽可能利用EDA软件或仿真软件对模型机系统中各部件进行仿真分析和功能验证。在学有余力的前提下，可进一步扩展相关功能。

## 设计要求

1. 根据课程设计指导书的要求，制定出设计方案；
2. 分析指令系统格式，指令系统功能。
3. 根据指令系统构建基本功能部件，主要数据通路。
4. 根据功能部件及数据通路连接，分析所需要的控制信号以及这些控制信号的有效形式；
5. 设计出实现指令功能的硬布线控制器；
6. 调试、数据分析、验收检查；
7. 课程设计报告和总结。

## 技术指标

1. 支持表 1.1前27条基本32位MIPS指令；
2. 支持教师指定的4条扩展指令；
3. 支持多级嵌套中断，利用中断触发扩展指令集测试程序；
4. 支持5段流水机制，可处理数据冒险，结构冒险，分支冒险；
5. 能运行由自己所设计的指令系统构成的一段测试程序，测试程序应能涵盖所有指令，程序执行功能正确。
6. 能运行教师提供的标准测试程序，并自动统计执行周期数
7. 能自动统计各类分支指令数目，如不同种类指令的条数、冒险冲突次数、插入气泡数目、load-use冲突次数、动态分支预测流水线能自动统计预测成功与失败次数。

表 1.1 指令集

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

# 总体方案设计

## 单周期CPU设计

单周期CPU设计本次我们采用的方案是硬布线控制器。观察总体结构图可以将CPU分为若干模块：取指令寄存器、指令存储器、硬布线控制器、寄存器文件、运算器、数据存储器等。在一个时钟周期内，各个功能模块各司其职。解析指令，产生对应的控制信号并执行。通过 PC 寄存器给出正确的指令地址，从指令存储器中取出数据送到控制器，硬布线控制器解析该指令并给出对应的控制信号。控制信号给出多路选择器的选择信号、寄存器组的写使能信号、内存的写使能信号以及立即数的扩展方式，从而完成指令的正确执行。 在 24 单周期 CPU 的基础上添加 4 条扩展指令分别是:SRAV、LUI、LBU、BGTZ。

总体结构图如图 2.1所示。

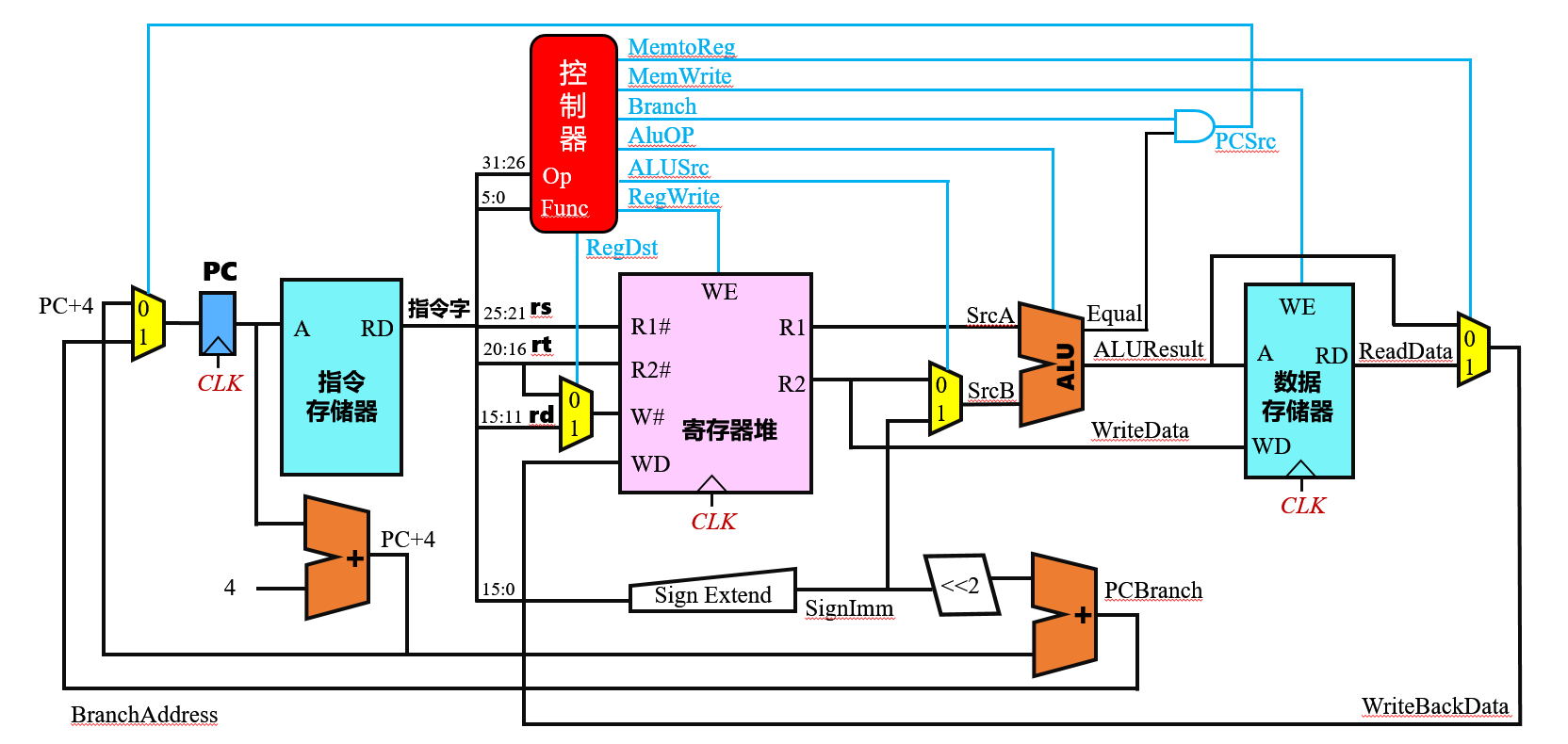


图 2.1 总体结构图

### 主要功能部件

运算器部分，具体设计思路如下。

#### 程序计数器PC

程序计数器中保存着下一条指令在指令存储器中的存储地址，具体硬件用一个寄存器实现。程序寄存器中保存的指令决定程序运行的走向，其中保存的地址可以是当前运行指令增4后的结果，也可以是分支跳转之后需要到达的地址。

#### 指令存储器IM

指令存储在ROM中，数据位宽为32位，以字寻址。取PC的第2到11位输入，将最终取出来的地址送到译码逻辑模块进行译码。

我们将写好的程序生成十六进制加载入ROM中即可执行。

#### 运算器

运算器可以执行13种运算，分别如下表 2.1所示。算术逻辑运算单元的引脚和功能描述如下表 2.2 所示。

表 2.1运算器功能

| ALU\_OP | 十进制 | 运算功能 |
| --- | --- | --- |
| 0000 | 0 | Result = X << Y 逻辑左移 （Y取低五位） Result2=0 |
| 0001 | 1 | Result = X >>>Y 算术右移 （Y取低五位） Result2=0 |
| 0010 | 2 | Result = X >> Y 逻辑右移 （Y取低五位） Result2=0 |
| 0011 | 3 | Result = (X \* Y)[31:0]; Result2 = (X \* Y)[63:32] 无符号乘法 |
| 0100 | 4 | Result = X/Y; Result2 = X%Y 无符号除法 |
| 0101 | 5 | Result = X + Y (Set OF/UOF) |
| 0110 | 6 | Result = X - Y (Set OF/UOF) |
| 0111 | 7 | Result = X & Y 按位与 |
| 1000 | 8 | Result = X | Y 按位或 |
| 1001 | 9 | Result = X⊕Y 按位异或 |
| 1010 | 10 | Result = ~(X |Y) 按位或非 |
| 1011 | 11 | Result = (X < Y) ? 1 : 0 符号比较 |
| 1100 | 12 | Result = (X < Y) ? 1 : 0 无符号比较 |

表 2.2 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 |
| --- | --- | --- | --- |
| A | 输入 | 32 | 操作数A |
| B | 输入 | 32 | 操作数B |
| ALU\_OP | 输入 | 4 | 运算器功能码，具体功能见上表 |
| shamt | 输入 | 32 | 移位数目 |
| Result | 输出 | 32 | ALU运算结果 |
| Equal | 输出 | 1 | Equal=(x==y)?1:0, 对所有操作有效 |

#### 寄存器文件RegFile

MIPS寄存器文件包含32个位宽为32位的寄存器，每个时钟周期可根据寄存器编号同时读两个寄存器的内容，并且修改一个寄存器，写使能控制写数据，输入输出的引脚见表 2.3 。

表 2.3 寄存器文件引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| CLK | 输入 | 1 | 时钟信号 |
| R1# | 输入 | 5 | 读寄存器1的编号 |
| R2# | 输入 | 5 | 读寄存器2的编号 |
| W# | 输入 | 5 | 写寄存器的编号 |
| WE | 输入 | 1 | 写使能信号 |
| Din | 输入 | 32 | 输入数据 |
| R1 | 输出 | 32 | 寄存器1的数据 |
| R2 | 输出 | 32 | 寄存器2的数据 |

#### 数据存储器

数据存储器用RAM实现，数据位宽为32位，地址选择位宽为10位。

### 数据通路的设计

对于不同类型的指令，其数据通路也不尽相同，我们可以观察填入其规律，并在有具体指令处理时进行对比修改。

表 2.4指令系统数据通路框架

| 指令 | PC | IM | RegFile | | | | ALU | | DM | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1# | R2# | W# | RDin | A | B | Addr | MDin |
| R型指令 | PC+4 |  | rs | rt | rd | alu | r1 | r2 |  |  |
| syscall | PC+4 |  | 2# | 4# |  |  | r1 | r2 |  |  |
| Store指令 | PC+4 | imm | rs | rt |  |  | r1 | imm | alu | r2 |
| Load指令 | PC+4 | imm | rs | rt | rt | Data | r1 | imm | alu | r2 |
| I型指令 | PC+4||(分支成功  PC+4+(imm<<2)) | imm | rs | rt | rt | ALU | r1 | imm |  |  |
| J型指令 | ||J\_PC  || |  |  |  |  |  |  |  |  |  |

### 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 2.5。

表 2.5主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| AluOP | 0-12 | ALU功能选择信号 |
| MemToReg | 0 | 写数据来源为ALU结果 |
| 1 | 写数据来源为Mem |
| MemWrite | 0 | Mem读 |
| 1 | Mem写 |
| AluSrcB | 0 | ALU输入端B取值为R2 |
| 1 | ALU输入端B取值为imm |
| RegWrite | 0 | 寄存器组读 |
| 1 | 寄存器组写 |
| Syscall | 0/1 | 是否系统调用 |
| SignedExt | 0 | imm使用0扩展 |
| 1 | imm使用符号扩展 |
| RegDst | 0 | 写寄存器编号为rt |
| 1 | 写寄存器编号为rd |
| BEQ | 0/1 | 指令是否为BEQ |
| BNE | 0/1 | 指令是否为BNE |
| JR | 0/1 | 指令是否为JR |
| JMP | 0/1 | 指令是否为JMP |
| JAL | 0/1 | 指令是否为JAL |
| MFC0 | 0/1 | 指令是否为MFC0 |
| MTC0 | 0/1 | 指令是否为MTC0 |
| ERET | 0/1 | 指令是否为ERET |
| LUI | 0/1 | 指令是否为LUI |
| BGTZ | 0/1 | 指令是否为BGTZ |
| LBU | 0/1 | 指令是否为LBU |

对照所有控制信号，依次分析各条指令，分析该指令执行过程中需要哪些控制信号，对于与本条指令无关的控制信号，控制信号的取值一律为0，以简化控制器电路的设计。该控制信号表的框架如图 2.2所示。

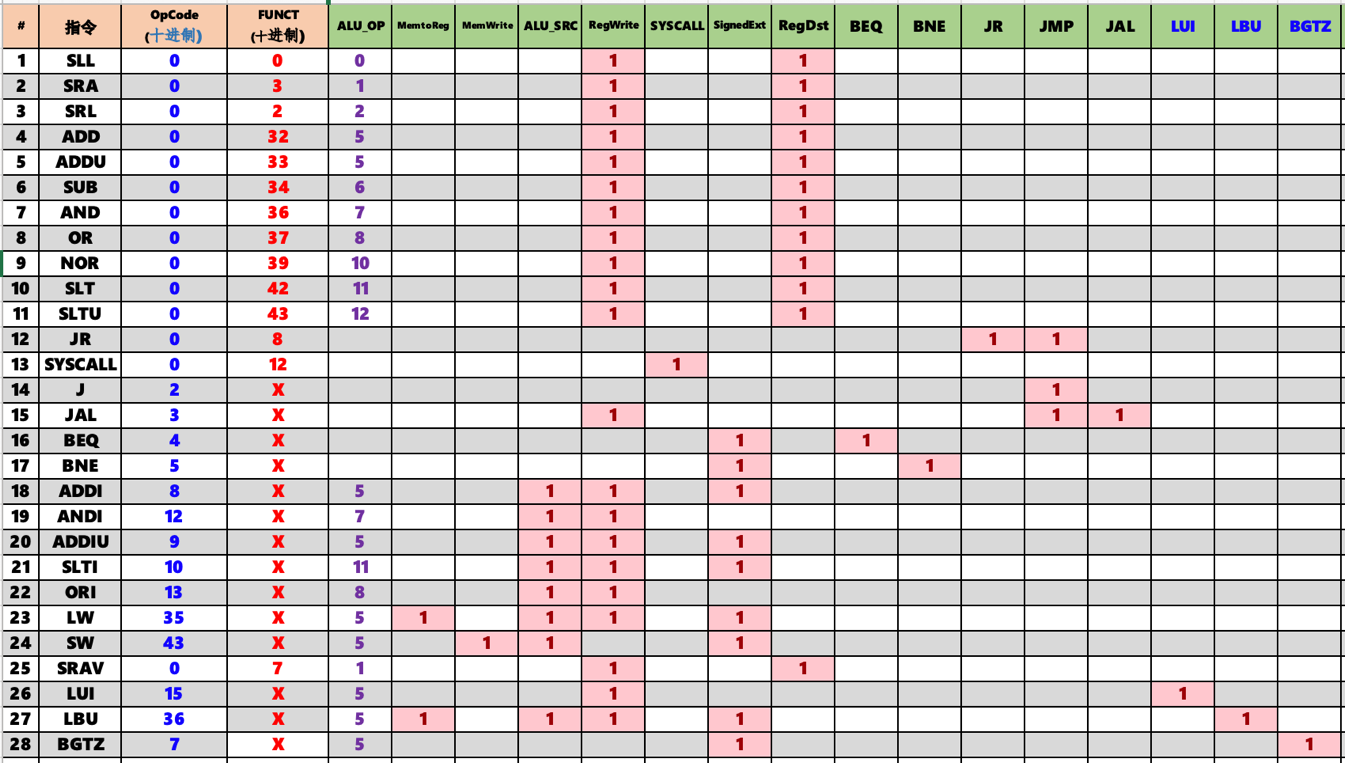


图 2.2 主控制器控制信号框架

## 中断机制设计

### 总体设计

中断分为单级中断和多级中断。

单级中断无需考虑被打断的情况，在遭遇中断时现将断点存入EPC寄存器，再在指令层面保护现场，根据中断仲裁逻辑执行优先度最高的中断服务子程序，假如在执行的过程中遇到其他中断，仍继续执行至结束，之后恢复现场，最后将EPC中存储的地址信息传递给PC，以恢复中断前的指令执行。

多级中断在保护现场后需要开中断以接收其他中断请求，对EPC、IE和中断屏蔽寄存器的操作需要增加ERET、MFC0和MTC0指令并更新控制器和数据通路。

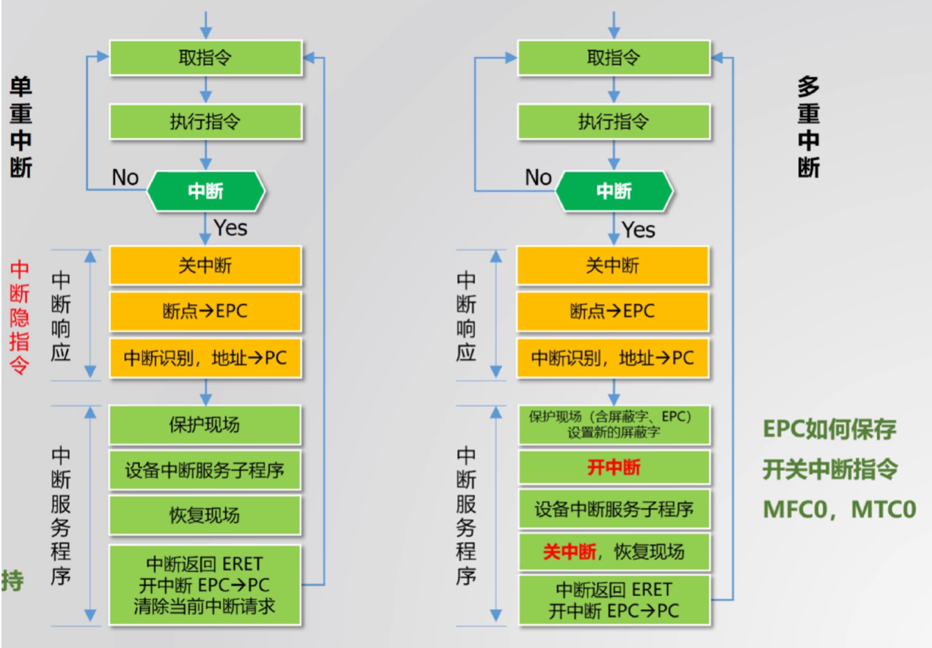


图 2.3 中断处理流程图

### 硬件设计

#### 中断请求电路

用 IR1#, IR2#和IR3# 三个按钮来代替中断请求信号的产生。为了方便测试，可以加入信号模拟电路，自发地产生中断信号。

#### 中断优先编码电路

三个中断信号可能同时出现，而在特定时刻只需要选择其中优先度最高的信号，使用优先编码器对中断请求信号进行编码，输出优先级最高的中断请求信号。

#### 中断使能寄存器

用D触发器实现，输出1时允许中断，反之为0时屏蔽所有中断。

#### 操作控制器修改

修改控制器的逻辑，使其能识别ERET, MFC0和 MTC0指令。MFC0和MTC0指令需要读写通用寄存器和IE、 EPC寄存器，需要将MFC和MTC指令作为寄存器的使能信号。

### 软件设计

软件设计层面需要与硬件协同。

首先需要根据指令是否支持开关中断，分别将中断使能信号IE置为1或0，决定CPU能否接收到外部中断信号。

#### ERET指令

ERET对应中断返回指令，开中断并恢复现场，从EPC中取出保存的指令地址到PC，继续执行之前被打断的指令。现场是可能被随机出现的中断请求破坏的寄存器值，保护现场和恢复现场分别指的是保存寄存器中的值到RAM中，以及从RAM中恢复寄存器的值。

#### MFC0指令

MFC0 指令的用法为 MFC0 rt，rd，其功能是将 CP0 中寄存器组中 rd 的内容送到通用寄存器组中的 rt 里面。

#### MTC0指令

MTC0和MFC0指令类似，用法为MTC0 rt，rd，只不过功能恰好相反，是将通 用寄存器组 rt 中的数据传递到 CP0 中寄存器组中的 rd 里面。

## 流水CPU设计

### 总体设计

流水线技术是指将复杂的任务分解成若干个阶段，每个阶段与其他阶段并行处理，其运行方式和工业流水线十分相似。流水线处理技术从时间上提高了并行性，通过将一个任务分成几个不同的子过程，并且各个子过程在不同的功能部件上并行执行，使得在同一个时钟周期内同时解释多条机器指令，提高程序执行速度。将CPU的数据通路划分为五段流水：取值(IF)、译码(ID)、执行(EX)、 访存(MEM)、写回(WB) 。 其中每个阶段的功能如下所述：

* 取指令(IF)：负责从指令存储器取出指令;
* 指令译码(ID)：操作控制器对指令字进行译码，同时从寄存号
* 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。

指令流水线逻辑框架见图 2.4。

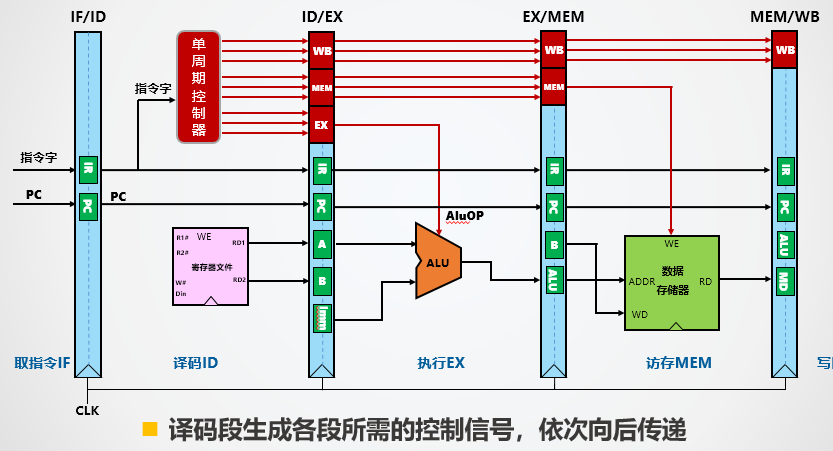


图 2.4 流水CPU框架

### 流水接口部件设计

流水接口部件的作用是将上一段中的数据和控制信号锁存给下一段使用，可以用寄存器组将功能段分开并将结果存储一个时钟周期。所有寄存器共用同一个时钟信号，上升沿触发，使能端接收停机信号，低电平有效；将多路选择器的1端信号接地，正常情况下选择输入信号，清零时选择常数0。

#### IF/ID流水接口

保存和向后传递在IF阶段取出的指令IR和指令对应的PC值。在气泡流水线和重定向流水线中，需要根据冲突判断进行使能端控制和清零。

#### ID/EX流水接口

ID 段获取了 IF 段传递过来的 IR 后对 IR 进行解析，控制器产生的控制信号按照其后阶段的需要进行传递。在 ID 段访问寄存器组之后需要将取出来的数据向后传递，对于 I 型指令，经过位扩展单元后的数据也要向后传递。同样在气泡流水线和重定向流水线中，需要根据冲突判断进行使能端控制和清零。

#### EX/MEM流水接口

EX 段会给出计算的结果，因此计算结果 ALU 需要向后传递。对于 syscall 指令， 为了确保指令能够执行完整，它需要在 WB 段再进行判断，因此 A 和 B 的值还要继 续向后传递。同时，PC、IR、MEM 段和 WB 段的控制信号以及中断相关的信号还需要还需要向后传递。

#### MEM/WB流水接口

访存指令会在 MEM 段给出数据结果，因此需要将访存结果向后传递。除此之外，PC、IR、WB 段的控制信号、中断相关信号、ALU 运算结果等等还需要向后传递。

### 理想流水线设计

理想流水线指的是没有任何数据相关和控制冲突指令的流水线，只需要保证数据正确传递即可。对于单周期MIPS CPU的改进，需要把数据通路用寄存器组划分为五个段，值得注意的是，写回段WB为了防止数据冲突，需要将RegFile设置为下降沿写入，保证下一个上升沿到来在译码段ID也能得到正确的值；其次，停机指令产生的信号要放到WB段，否则会因为在执行段EX停机导致MEM和WB段没有执行完整。寄存器组在 ID 段只可访问，WB 才可写，因此 W#和 Din 都应该由 WB 段给出。

## 气泡式流水线设计

由于理想流水线不能解决数据冲突。数据冲突的含义是当前指令要用到先前指令的操作结果，而这个结果尚未产生或尚未送达指定的位置，会导致当前指令无法继续执行，称为数据冲突。所以我们需要在理想流水线的基础上检测数据相关的逻辑。

插入气泡的含义是，为了避免下一条指令读写数据错误而插入空指令（气泡）来消除数据相关，而下一个时钟仍然正常执行。数据相关检测逻辑中，输入为ID段和EX段的指令信号，以及ID段和MEM段的指令信号/假设任意一个检测到了数据相关，则暂停ID段并插入气泡解决冲突，直到相关的数据成功地写回寄存器内才能继续执行ID段的指令。假设在EX段遇到了分支指令，意味着IF和ID段的指令是无效的，故将其以气泡替换，解决控制冲突。具体实现上需要在IF/ID流水接口和ID/EX流水接口上对使能端和清零端增加信号进行控制。

## 重定向流水线设计

气泡流水线通过延缓 ID 段取操作数的方式解决数据冲突问题，但大量气泡的插入会严重影响指令流水线性能。还有一种思路是先不考虑 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。

假设可以判断指令的冲突情况将MEM或WB阶段的数据交给EX段，就可以不用产生气泡而占用额外的时钟周期了。在ID段检测数据相关分为load-use相关和其他相关，ID段的某一源寄存器是EX段的目标寄存器，且EX段是一条load指令则产生load-use相关，需要在ID/EX接口插入一个气泡，否则流水线的关键时延会变长。修改控制器产生2位宽的输出信号f1和f2，决定ALU的输入端选择何处的数值。

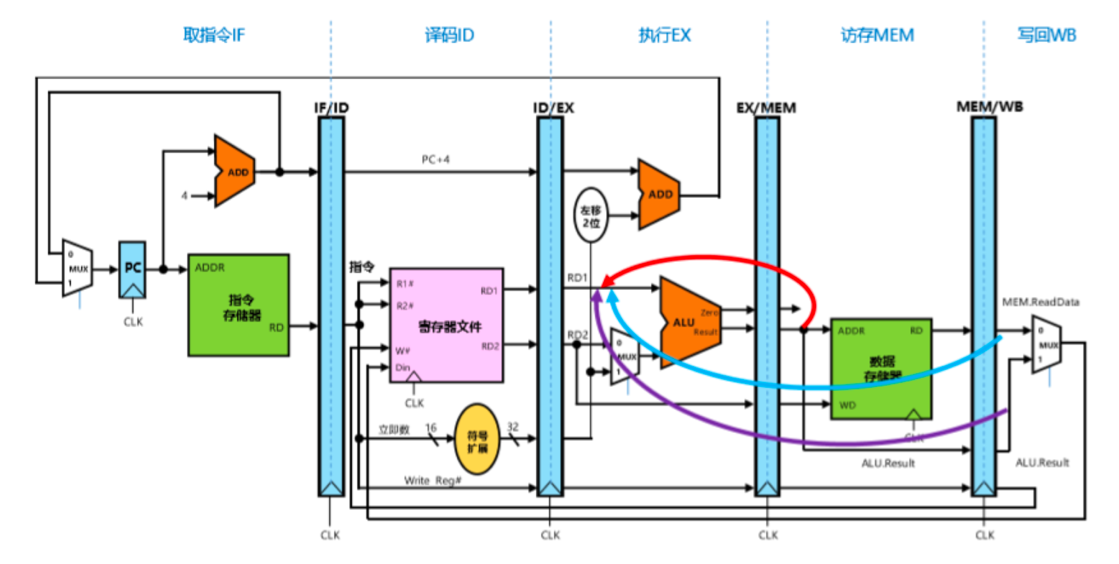


图 2.5 数据重定向示意图

# 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）

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。

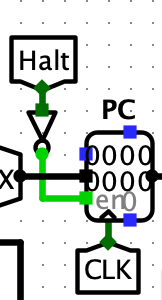


图 3.1程序计数器（PC）

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。

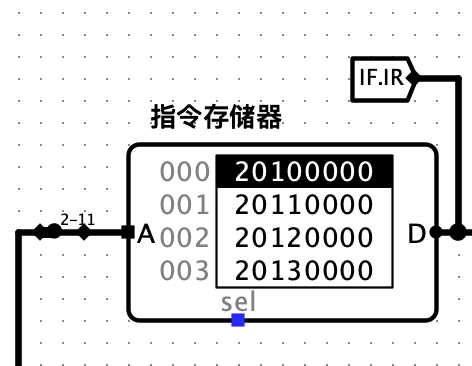


图 3.2指令存储器（IM）

1. 数据存储器

使用数据位宽为32，地址位宽为10的随机存储器（RAM）实现。输入地址32位，取2-11位作为DM的地址。如下图 3.3 所示。

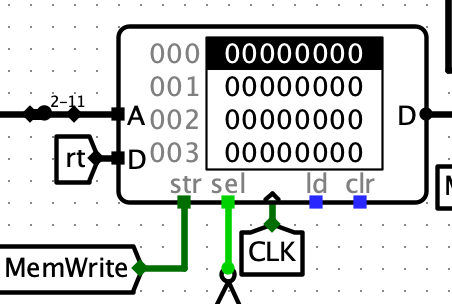


图 3.3 数据存储器（DM）

### 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 | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |  |  |
| SUBU | 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 |  |  |
| SLT | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 1 |  |  |
| SLTU | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 1 |  |  |
| JR | PC+4 | PC | rs |  |  |  |  |  | X |  |  |
| SYSCALL | r1 | PC | v0 |  |  |  | r1 | 10 | X |  |  |
| J | (PC&0XF000) | PC |  |  |  |  |  |  | X |  |  |
| JAL | (PC&0XF000) | PC |  |  | 31 | PC+4 |  |  | X |  |  |
| BEQ | PC+4+offset | PC |  |  |  |  |  |  | X |  |  |
| BNE | PC+4+offset | PC |  |  | 31 | PC+4 |  |  | X |  |  |
| LW | PC+4 | PC | rs |  | rt | DM.Dout | r1 | 立即数 | 5 | ALU |  |
| SW | PC+4 | PC | ra | rt |  |  | r1 | 立即数 | 5 | ALU | RF.D |

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。

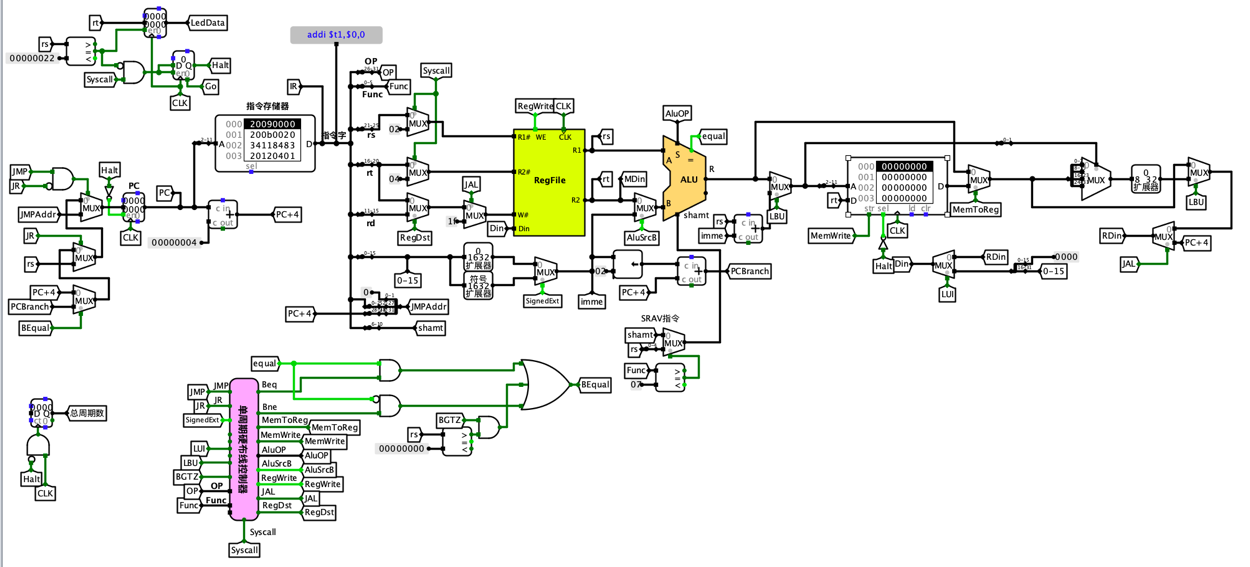


图 3.4 单周期CPU数据通路（Logisim）

### 控制器的实现

根据图 2.2 的主控制器控制信号框架，将对应值填入自动生成真值表达式的excel文件中，将表达式填入logisim的自动生成电路功能，生成运算器控制器、控制信号生成的电路。最后将输出信号引到输出引脚，将控制器封装，控制器实现见图 3.5 。

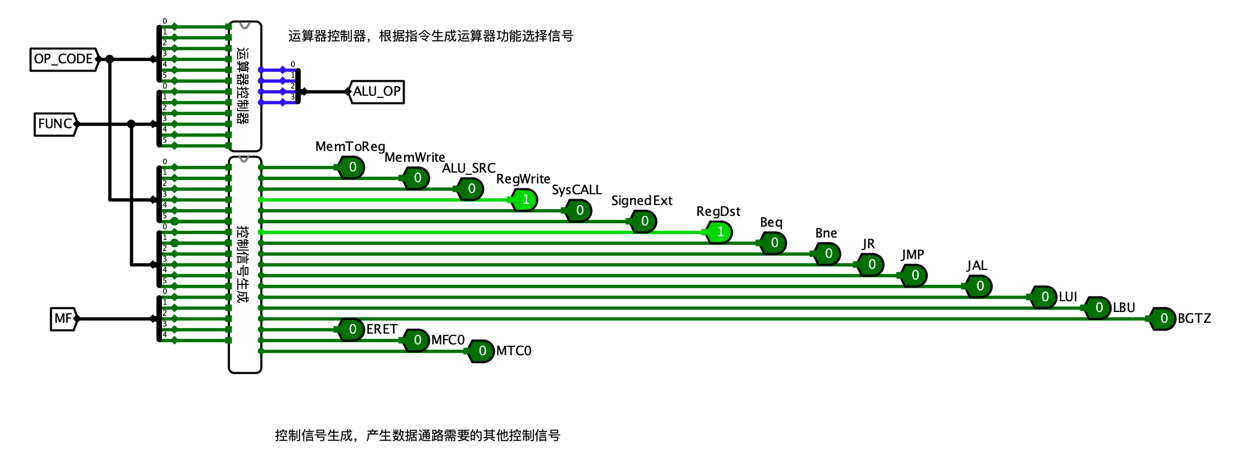


图 3.5 硬布线控制器实现

## 中断机制实现

### 中断信号的产生

中断信号作为D触发器的时钟端输入，假如按下相当于触发器经历了一个上升沿，被锁存在其中，下一个时钟周期D触发器IR会变为1，持续产生中断信号直到同步清零信号将触发器清零。如图 3.6 。

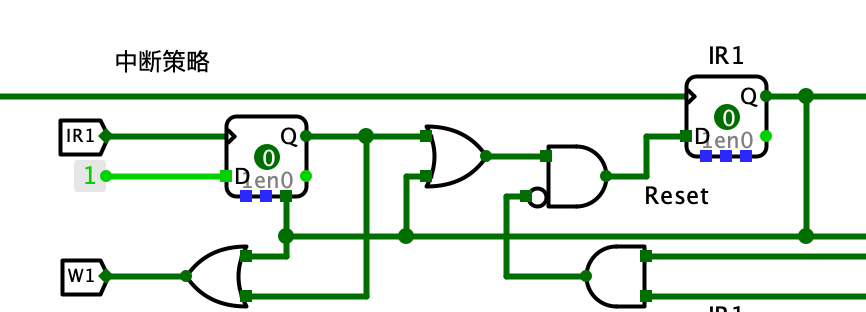


图 3.6 中断信号产生电路

### 中断信号清零

单级中断，当ERET指令到来时，ERET信号与自身中断信号通过与门，求反后作为IR的输入信号，上升沿到来时，将中断信号清零，见图 3.7 。

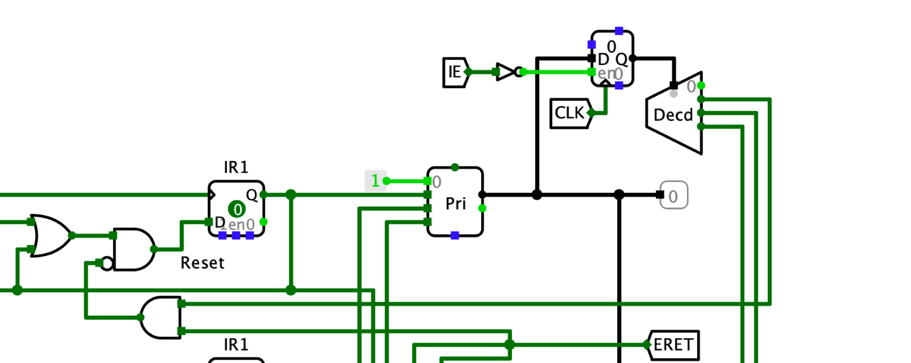


图 3.7 单级中断信号清零

多级中断因为用到了一个2位宽的D寄存器保存当前中断号，因此需要用一个解复用器将ERET的高电平信号输入到对应IR前的与非门里，清空当前控制中断的寄存器，如图 3.8 .

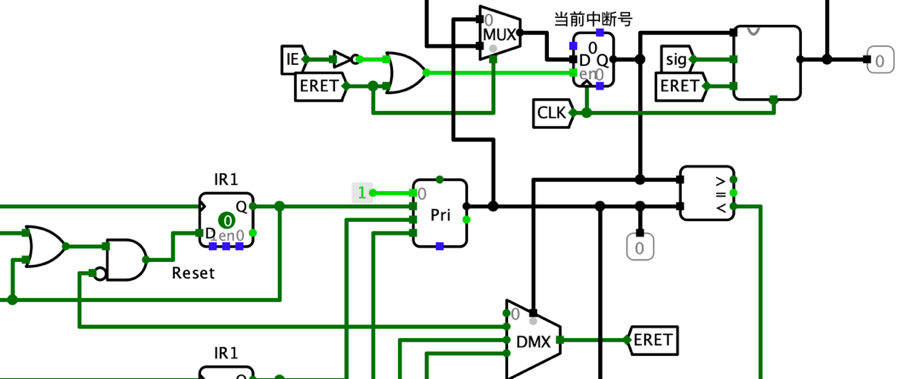


图 3.8 多级中断信号清零

### 中断使能寄存器IE和EPC寄存器

多级中断中需要实现开关中断的指令，MFC0和MTC0分别对应开中断和关中断。ERET为1时，将IE寄存器清零，意味着开中断，关中断信号产生时IE使能端为0，如图 3.9 .

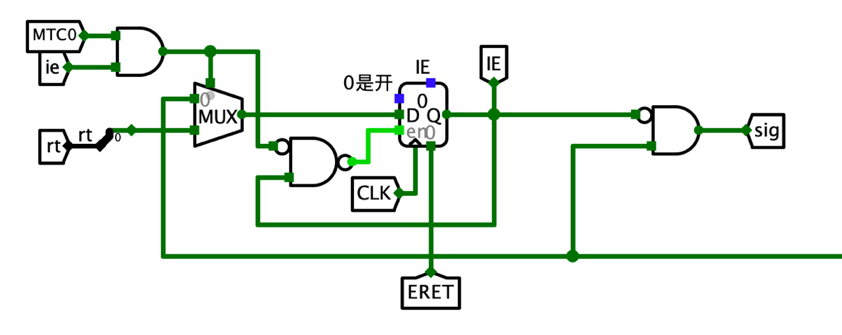


图 3.9 中断使能寄存器IE

### 中断入口地址选择

单级中断中，中断请求信号有优先级，即3>2>1。将3个中断请求信号连上优先编码器，输出为当前优先级最高的中断信号，根据2位宽的中断编号决定最终选用的中断程序入口地址。中断入口地址选择电路如图 3.10。

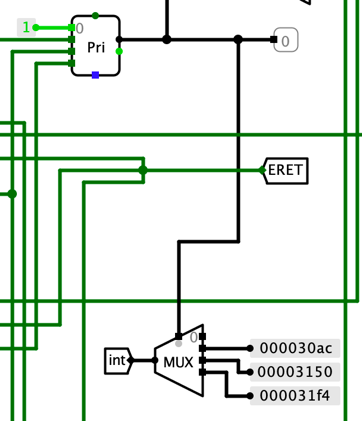


图 3.10 单级中断入口地址选择电路

多级中断的入口选择会更加复杂。因为要记住被打断时当前的中断号，而进入新的中断子程序时可能又被更高优先度的子程序中断，需要记住这一次的中断号，如果使用寄存器那么最开始记住的中断号就会丢失。

为了解决这个问题，设计了一个容量为3的硬件栈，可以将中断时的当前中断号压栈，在遇到ERET后令栈顶元素（上一个中断号）出栈，从而实现中断号的记忆与保存。

输入端ir是当前中断号，sig是中断信号，高电平时计数器正向计数，表示栈元素增加，同时通过解码器令代写寄存器的写使能为1；eret高电平时计数器反向计数，表示有元素出栈，计数器的值是多路选择器的选择输入端，决定栈顶元素是哪个D寄存器的值。电路图见图 3.11。

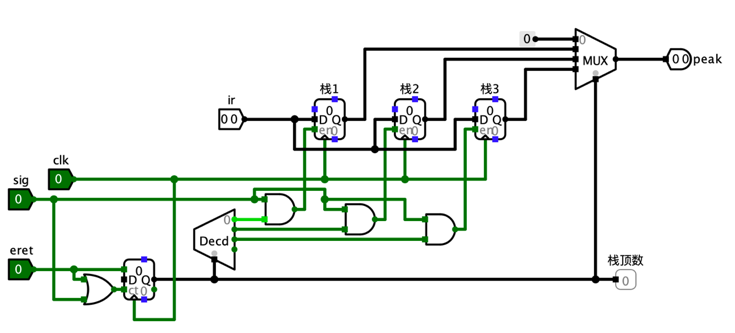


图 3.11中断硬件栈

多级中断决定入口地址同单级中断思路一致。如下图 3.12.

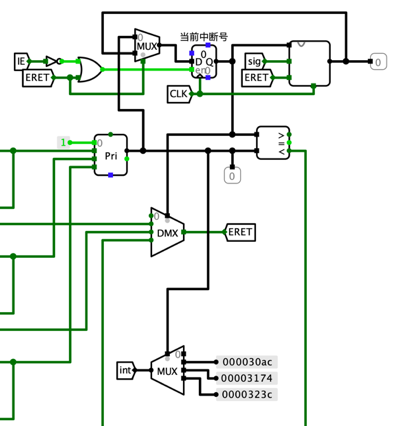


图 3.12 多级中断入口地址选择电路

## 流水CPU实现

### 流水接口部件实现

用寄存器组存放需要的数据和控制信号，采用同一个始终信号，将清零端设为常量0，高电平同步清零，所有输入信号采用对应位宽的寄存器输出，按时钟上升沿触发，使能端低电平有效，高电平忽略时钟输入。如图 3.13所示。

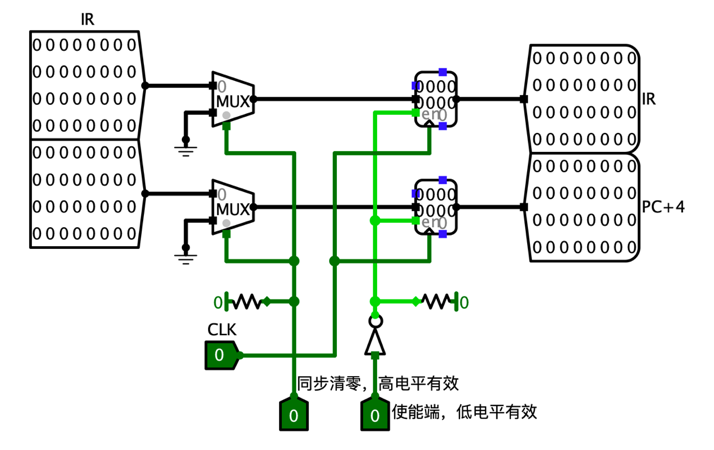


图 3.13 流水接口部件实现

### 理想流水线实现

在单周期 MIPS 的基础上，在指令存储器和寄存器组之间插入 IF/ID 接口，在寄 存器组和 ALU 之间插入 ID/EX 接口，在 ALU 和 MEM 之间插入 EX/MEM 接口，在 MEM 之后放入 MEM/WB 接口将数据通路分为五部分，分别为 IF、ID、EX、MEM 和 WB。每个寄存器组都由图3.13的基本单元构成，可以锁存各种位宽的数值，并在流水线中传递信息，五段流水的理论加速比为5:1。

syscall 的判断在 WB 段执行，寄存器组的写寄存器编号以及待写数据由 WB 段提供。具体实现如图 3.14所示。

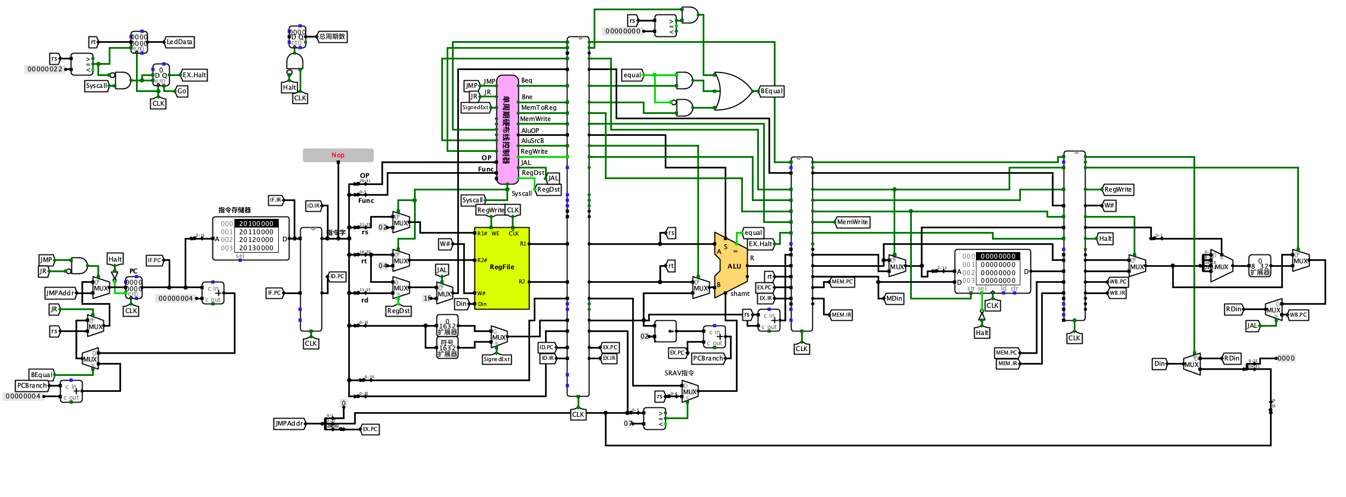
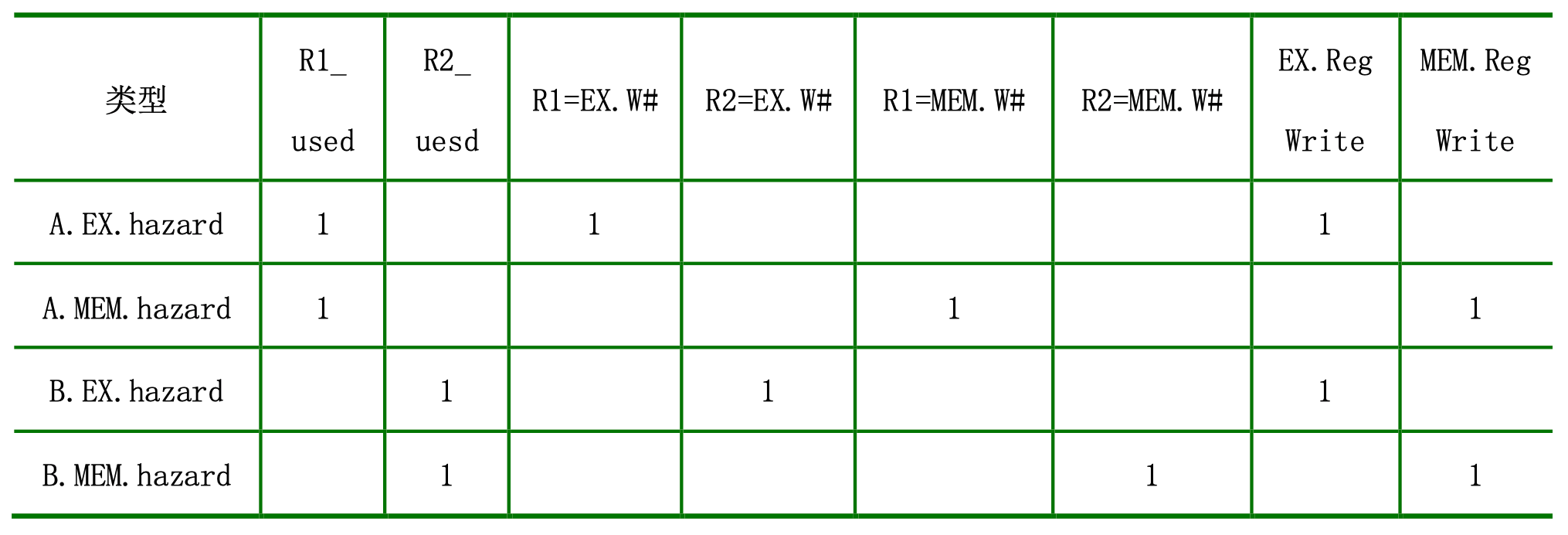


图 3.14 理想流水线实现

## 气泡式流水线实现

在ID段需要对指令进行数据相关检测，对于四种需要检测的数据相关，发生的条件见表 3.2。

表 3.2 数据相关产生条件



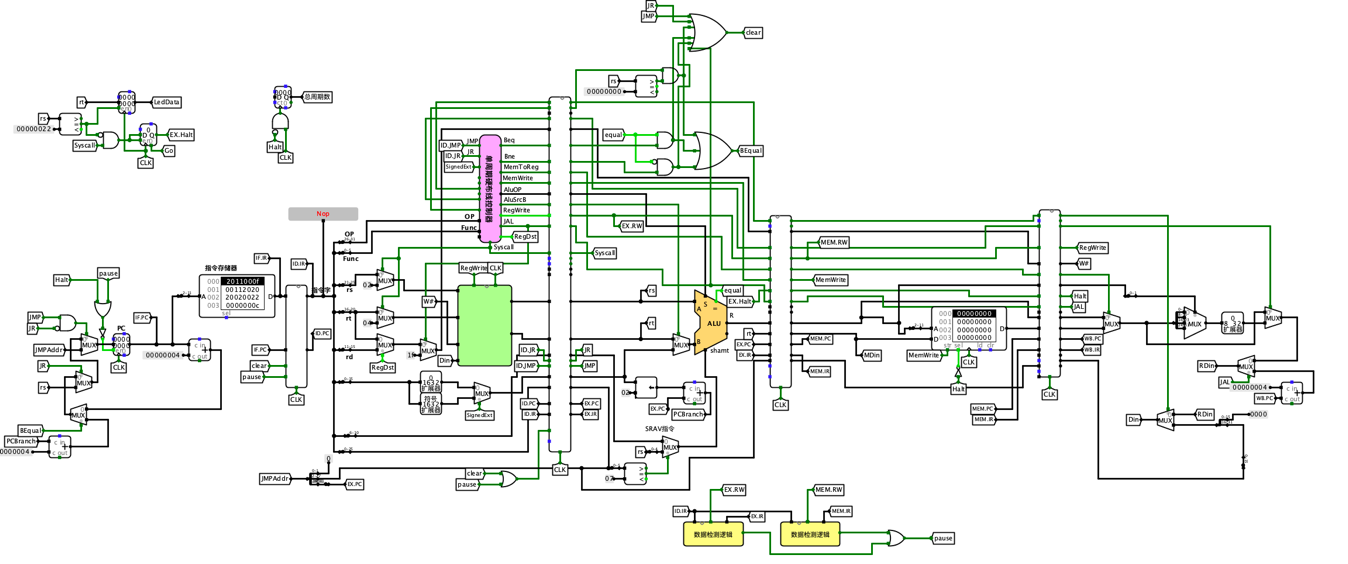


图 3.16 气泡流水线实现

## 重定向流水线实现

重定向流水线中的检测又和气泡流水线中不同。重定向检测的逻辑如下图 3.17所示。

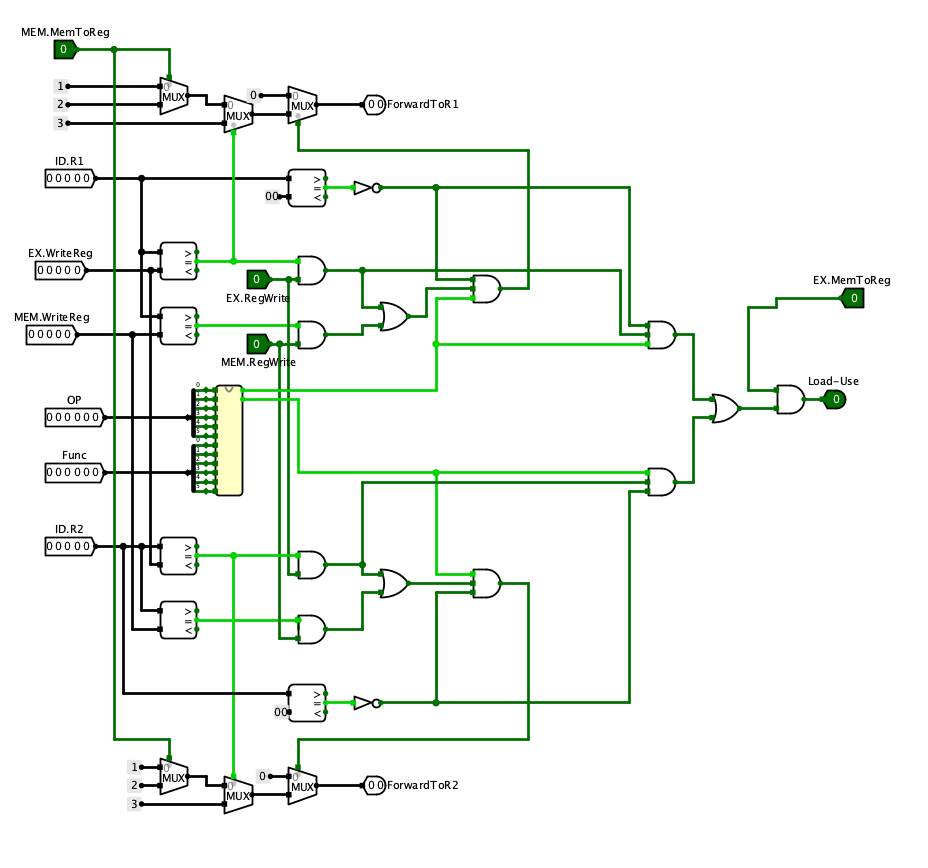


图 3.17 重定向检测

其中黄色的方块代表了寄存器使用情况检测，输入为OP和F，输出为R1\_used和R2\_used，表示R1和R2寄存器的使用情况，是根据每条指令的OP和F的值以及它们指令中寄存器的使用来得到的，把所有用到寄存器的OP和F值都或起来即可。两个输出的表达式如下图 3.18和图 3.19所示。

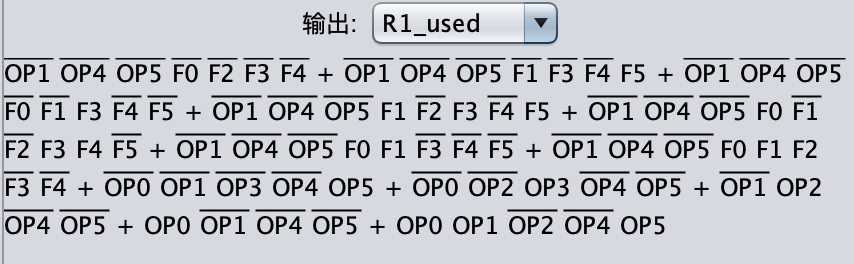


图 3.18 R1\_used输出表达式

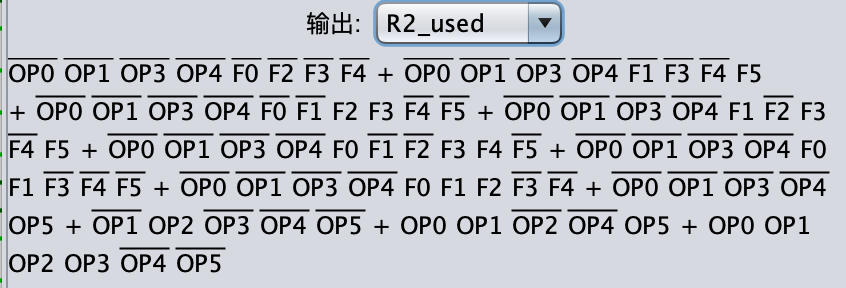


图 3.19 R2\_used输出表达式

除上所述之外，气泡流水线和重定向流水线中都是用了下降沿的寄存器文件。重定向流水线如下图 3.20所示。

  
图 3.20 重定向流水线实现

# 实验过程与调试

## 测试用例和功能测试

对各个任务用测试文件分别单独测试，除了多级中断执行中断服务程序，理想流水线执行简化后的测试程序，其他任务都用benchmark标准程序进行测试。最后还要测试自己的扩展指令：SRAV、LUI、LBU和BGTZ。

### 单周期MIPS CPU

加载benchmark.hex，开启时钟连续进行测试。运行结果如下图 4.1所示。

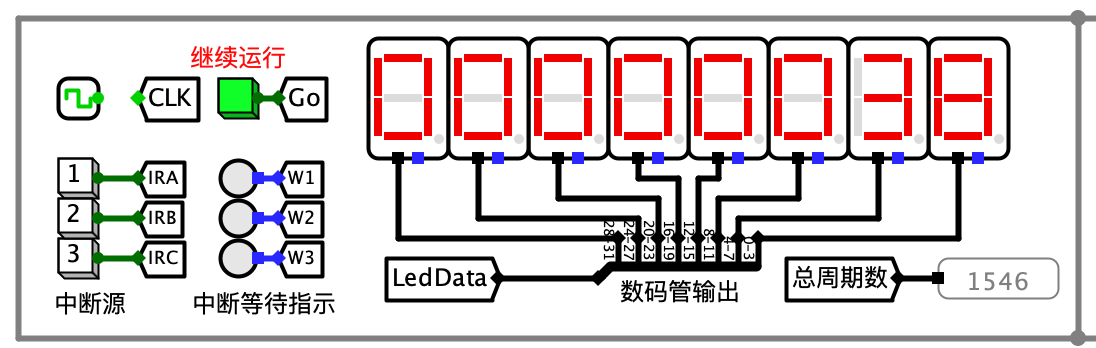


图 4.1 单周期benchmark运行结果

### 单周期MIPS+单级中断

主程序的输出内容为1-F左右循环的跑马灯，第一号中断的输出内容为倒数1…3, 1…2, 1…1，第二号中断的输出内容为2…3, 2…2, 2…1，以此类推。

开始时钟连续仿真测试，依次点击产生1号、2号、3号中断，于是先进入1号中断子程序，执行完成后进入优先度更高的3号中断。

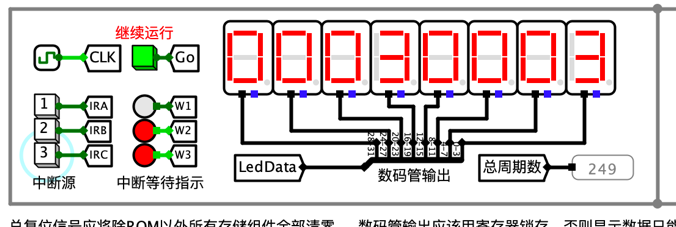
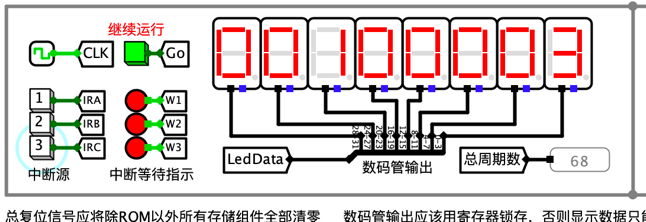


图 4.2 1号中断执行完后进入3号中断

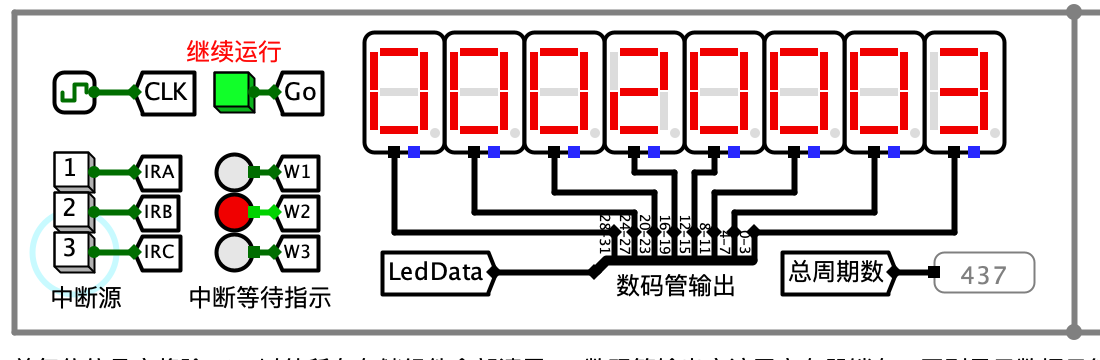


图 4.3 3号中断执行完后进入2号中断

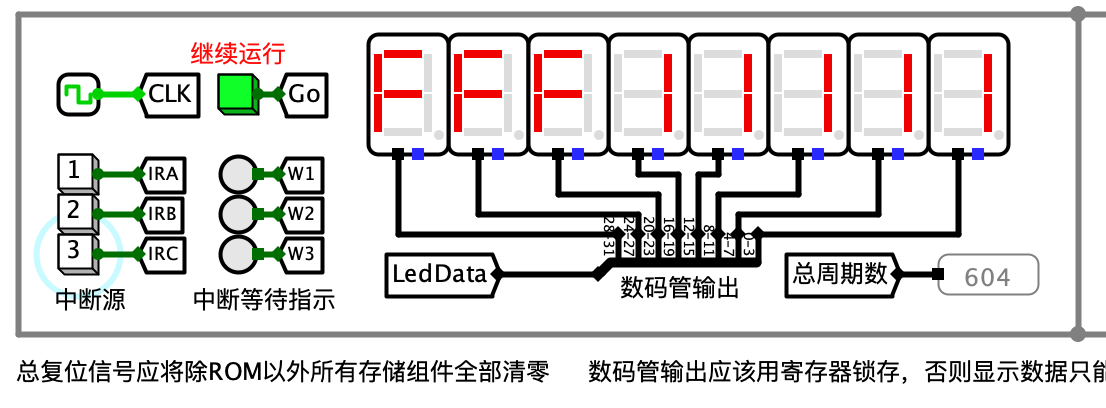


图 4.4 2号中断执行完后进入主程序

### 单周期MIPS+多级中断

多级中断测试在开始时也是顺序接收1, 2, 3号中断，但是多级中断允许打断，故进入优先度最高的3号中断子程序，执行完后进入2号中断子程序。如下图 4.5所示。

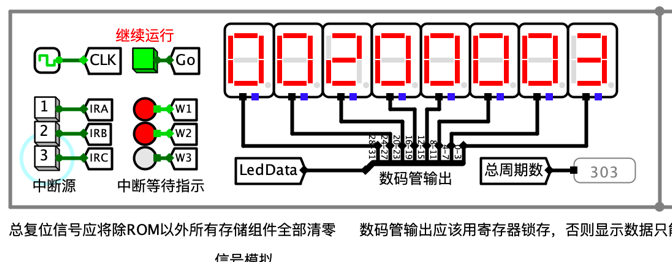
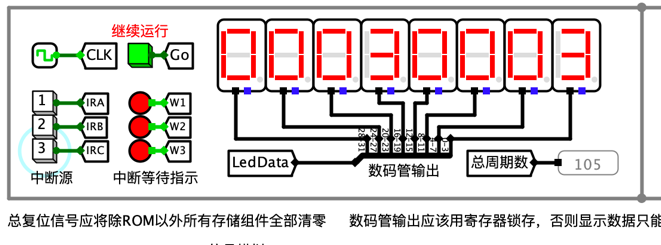


图 4.5 3号中断执行完进入2号中断

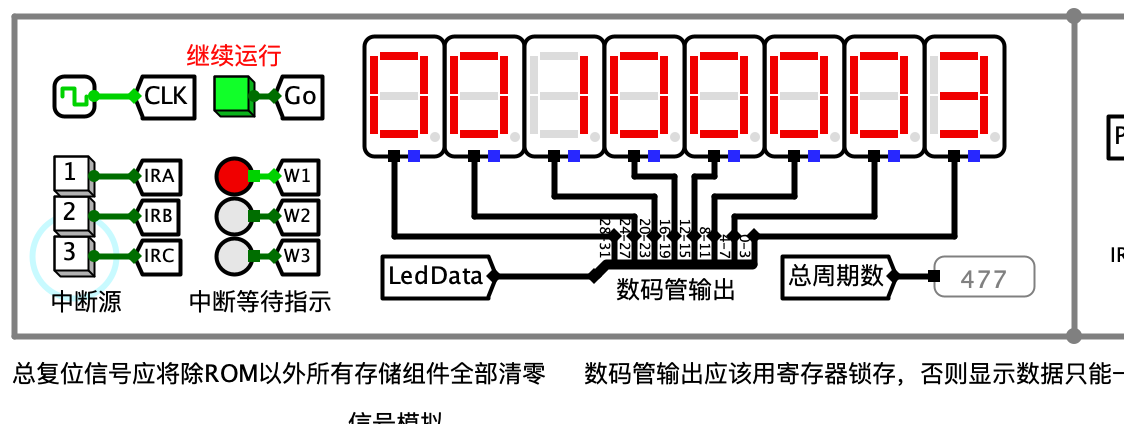


图 4.6 2号中断执行完进入1号中断

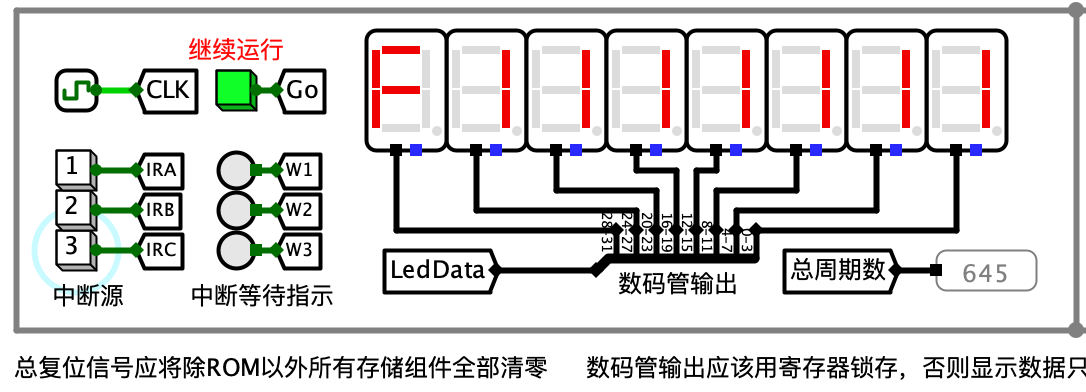


图 4.7 1号中断执行完进入主程序

### 理想流水线

加载理想流水线的程序，结果如下图 4.8所示。



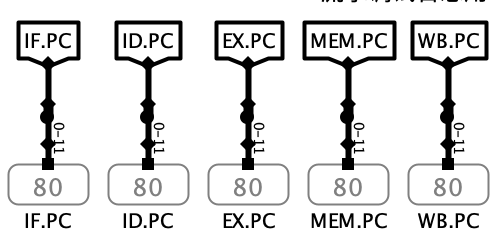


图 4.8 理想流水线运行结果

查看存储器中保存的情况，如下图 4.9所示。



图 4.9 理想流水线运行存储结果

### 气泡流水线

加载benchmark，运行结果如下图 4.10。

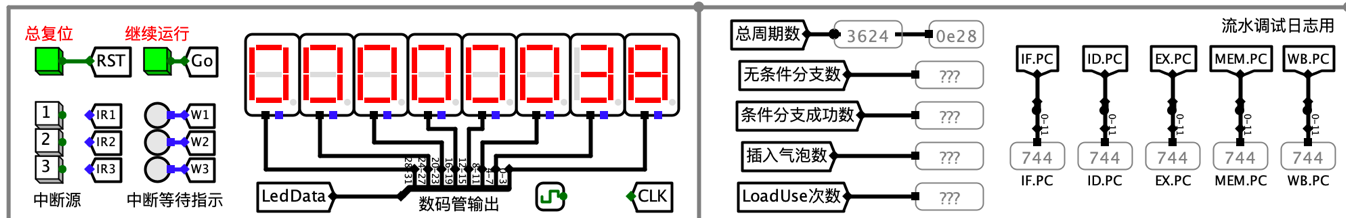


图 4.10 气泡流水线运行结果

### 重定向流水线

由于采用EX段的重定向，总周期数为2298，运行结果如下图4.11所示。



图 4.11 重定向流水线运行结果

### 扩展指令

#### SRAV

依次输出0x87600000 0xf8760000 0xff876000 0xfff87600 0xffff8760 0xfffff876 0xffffff87 0xfffffff8 0xffffffff

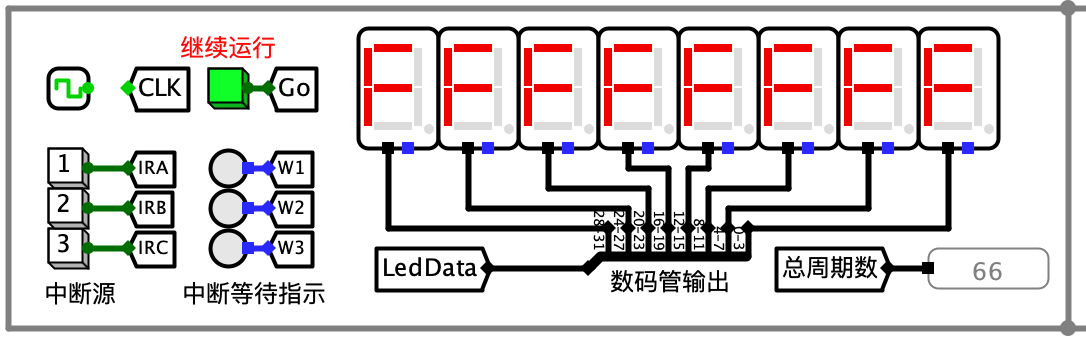
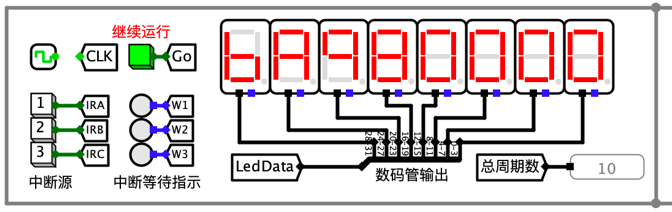
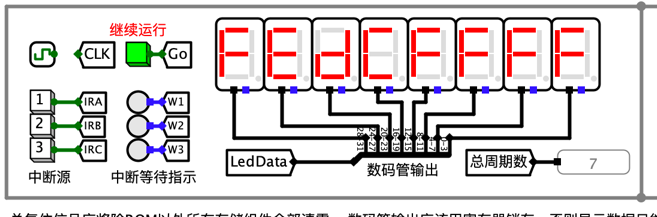


图 4.12 SRAV运行最终结果

#### LUI

依次输出 0xfedcffff 0xba980000 0x76540000 0x32100000 …… 0xfedcffff 0xba980000 0x76540000 0x32100000



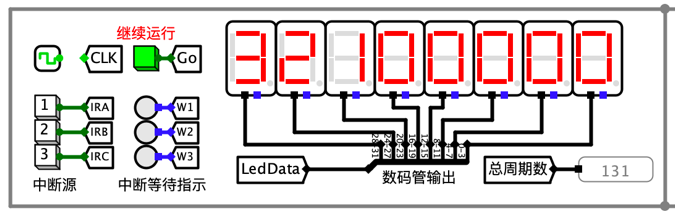
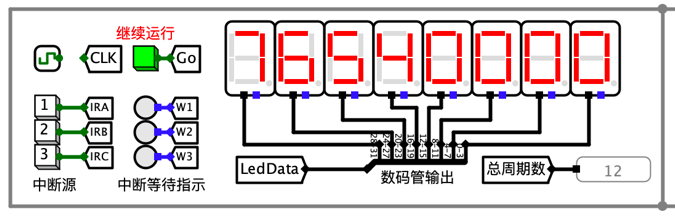


图 4.13 LUI运行结果

#### LBU

依次输出 0x00000081 0x00000082 0x00000083 …… 0x0000009e 0x0000009f **0x000000a0**



图 4.14 LBU最终运行结果

#### BGTZ

依次输出0x0000000f 0x0000000e 0x0000000d 0x0000000c 0x0000000b …… 0x00000005 0x00000004 0x00000003 0x00000002 **0x00000001**

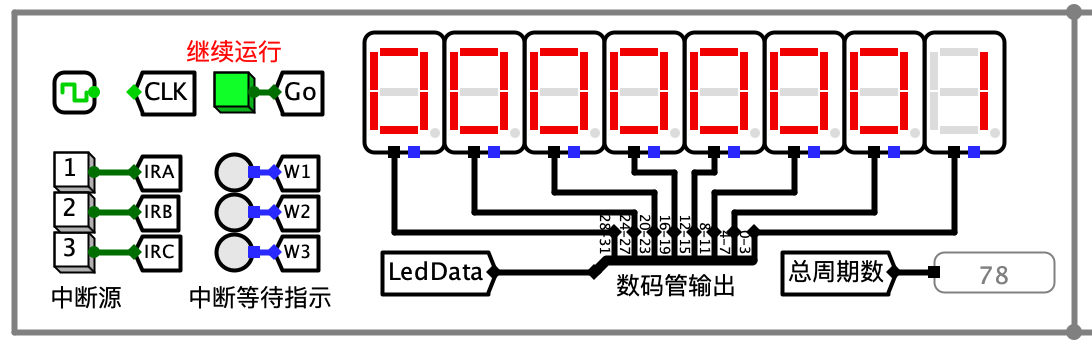
****

图 4.15 BGTZ最终运行结果

## 性能分析

(1) 单周期CPU的总周期数最少，CPI为1，而流水线CPU的CPI高很多。但实际上单周期CPU的关键路径时延高得多，时钟频率无法提高很多，导致执行时间严重被关键路径长的指令约束。

(2)看似气泡流水线是一种较为方便的解决冲突的妥协方式，但是实际上数据相关的指令和跳转指令在整个程序指令中的占比很高，因此会造成流水线资源极大地浪费，大量时钟周期CPU在空转，性能不好。而重定向流水线则性能更好。

(3) 优秀的分支预测策略能很好地解决数据相关和控制冲突，将其损失降低。

## 主要故障与调试

### 气泡流水线故障

气泡流水线：停在了第31个周期。

**故障现象：**如图4.16所示，流水线过早地接收到了停机信号。

**原因分析：**遇到syscall指令时，首先应该看rs寄存器的值是否为34，是表示打印rt的内容到led灯，否则执行停机，观察到syscall的前一条指令是addi $v0, $0, 34，说明rs的写入有问题才发生的误将打印变成停机。最后观察气泡流水线的特点，发生数据相关的时候，清除之前段相关的指令，发现是结构冲突，寄存器组同时读写会有先后问题。

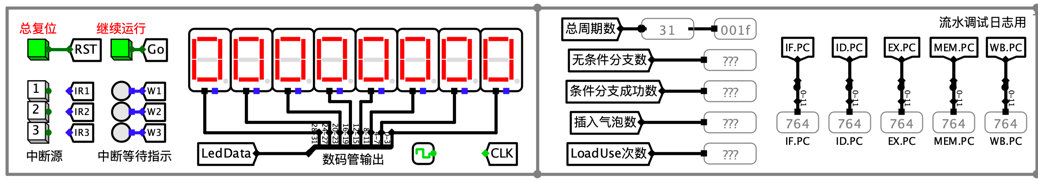


图 4.16 气泡流水线故障

**解决方案：**将RegFile改上升沿的触发方式为下降沿，使得在读写寄存器前半个周期时数据就被成功写入了寄存器**。**

### 扩展指令LBU故障

重定向流水线中的扩展指令LBU故障。

**故障现象：**在完成其他电路的扩展指令设计之后，重定向流水线中的LBU指令一直错误，LED显示乱跳。

**原因分析：**在源寄存器使用情况检测的电路中，未加上LBU指令的考虑，导致在执行时没有正确检测和处理冲突。

**解决方案：** 考虑上LBU的源寄存器使用，修改其R1\_used和R2\_used输出的逻辑表达式。

## 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。 |
| 第三天 | 实现了单周期CPU并且通过了测试。 |
| 第四天 | 查看流水线CPU的思想和知识，并开始理想流水线CPU设计。 |
| 第五天 | 完成流水线接口部件的设计。 |
| 第六天 | 不断调试理想流水线CPU并完成，通过测试。开始准备气泡流水线。 |
| 第七天 | 画数据相关检测逻辑，调试气泡流水线并通过测试， |
| 第八天 | 修改数据检测逻辑，完成重定向流水线。 |
| 第九天 | 看慕课学习中断相关的知识，完成单级中断。 |
| 第十天 | 在单级中断的基础上，修改调试完成多级中断。 |

# 团队任务

## 团队任务概述

我们的团队任务是一个简易的小游戏，就是用一个T型的篮子来接2\*2的方块。每隔固定的时间屏幕上方会随机生成方块，然后匀速下落。如果接到方块玩家的分数会加1。如果漏掉一个方块玩家的生命值会减去1，玩家的初始生命值为3，漏掉3个方块后游戏结束。

在游戏中，玩家可以控制T型篮子向左移或者向右移，也可以选择随时重新开始游戏。

而我完成的部分是其中硬件部分的图片显示模块。我们小组使用的是20\*32的LED显示屏，由于LOGISIM中的LED是随着连接数据的变化即时更新的，而我们想要的是更新完整块屏幕的数据后屏幕再读取数据，因此需要添加一个图片显示模块，暂存每一列的更新数据。

图片显示模块有4个输入，column输入指定了哪一列将更新数据，writeEnable是写使能端，为高电平会更新模块内部保存值。OutEnable显示使能端为高电平会更新20列的所有数据。

## 图片显示模块实现

实现图片数据的保存需要两组寄存器，一组寄存器伴随着writeEnable写使能端为高电平时更新数据，但是这组数据并不实时显示，实际显示需要伴随OutEnable显示使能端为高电平时更新20列的所有数据同时显示。这部分实现可以如下示例图 5.1所示。

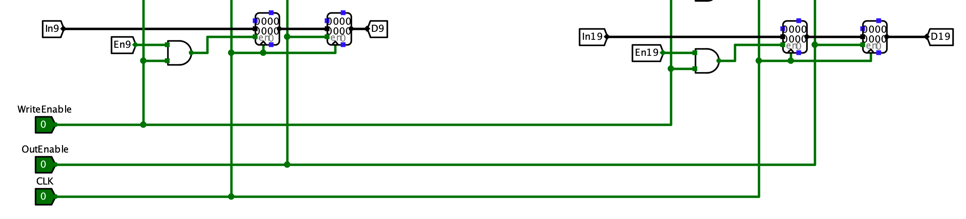


图 5.1 图片显示模块寄存器组

其中In+数字的隧道内容来自输入的data和列值进行选择（解复用）之后得到的值。具体实现如下图 5.2所示。

而En+数字的隧道内容来自于列选择，因为更新数据是针对某一列值而言的，所以需要进行列选择，具体实现如下所示图 5.3所示。

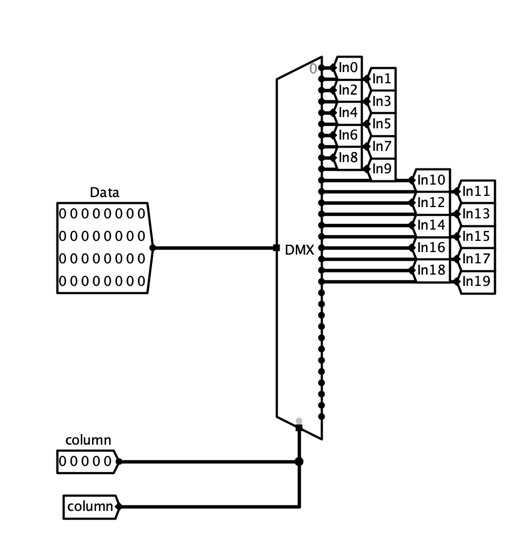
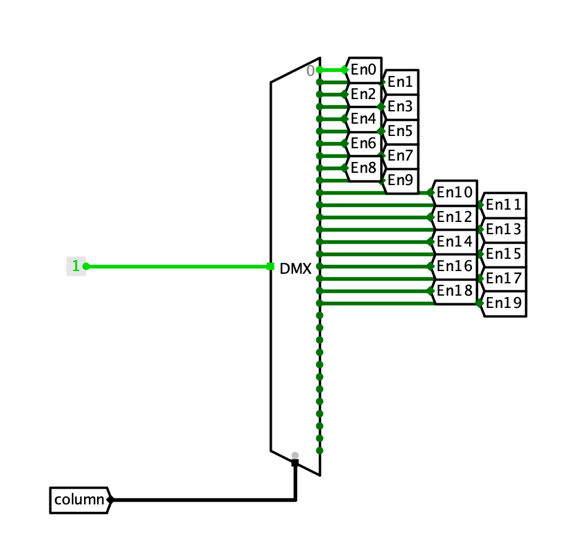
 

图 5.2 数据列选择 图 5.3 更新使能端列选择

最后将每一列内容输出即可。如下图 5.4所示。

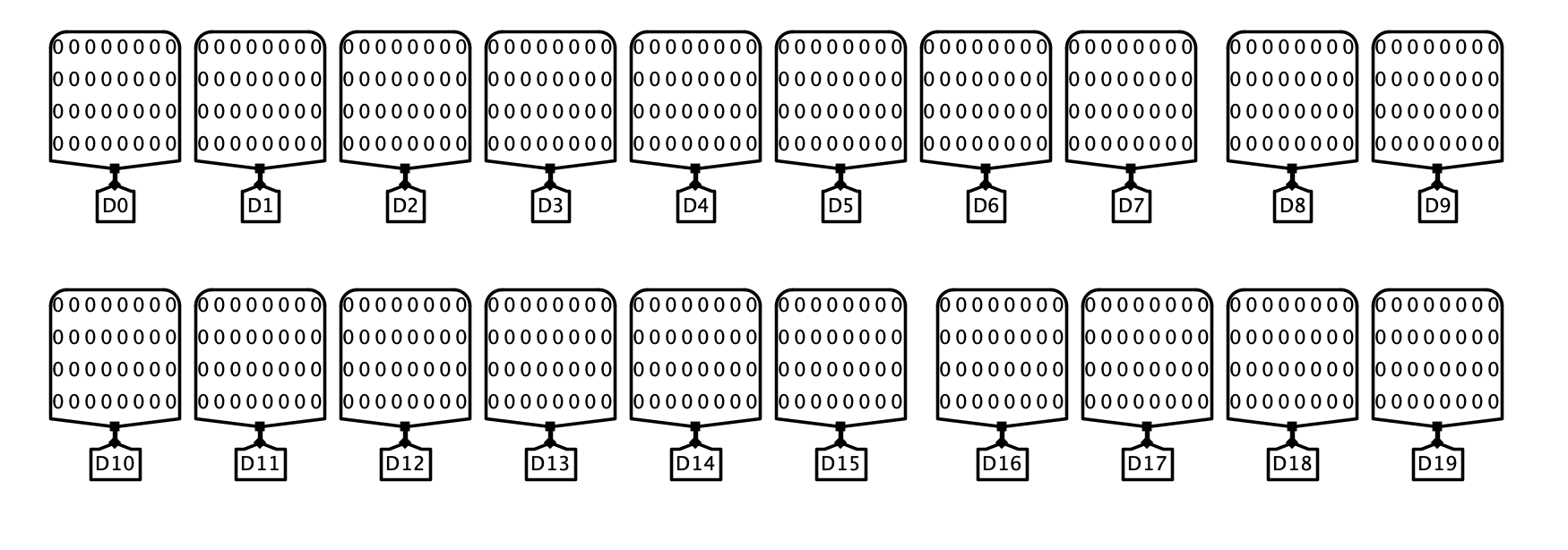


图 5.4 图片显示模块输出

完成后展示结果如下图5.5 所示。

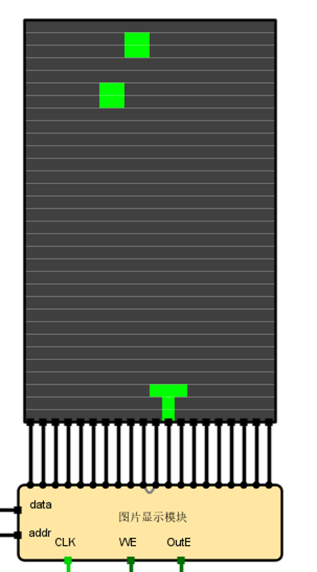


图 5.5 游戏最终图片显示

# 设计总结与心得

## 课设总结

本次课设加深了我对MIPS单周期和流水线CPU的理解。主要作了如下几点工作：

1. 单周期 CPU 实现了 24 条基本指令 + 4 条扩展指令分别为SRAV、LUI、LBU、BGTZ。
2. 在单周期CPU的基础上加入中断逻辑，实现了单级中断和多级嵌套中断。
3. 对单周期 CPU 指令执行阶段分成五个阶段:取指令 IF、指令译码 ID、指 令执行 EX、访存 MEM、写回 WB，正确设计实现接口部件，得到理想流水线。在理想流水线基础上添加数据相关检测和控制冲突处理逻辑实 现气泡流水线。对气泡流水线进行优化，实现重定向流水线。
4. 完成团队任务中图片显示模块的Logisim设计和视频录制。

## 课设心得

受疫情影响，本次组原课设时我还未返校，所有和老师同学的沟通都是在网络上完成的，一定程度上增加了学习成本，同时也在此感谢谭老师线上检查实验的时间。本次课程设计可以说是迄今为止所有实验以及课程设计中难度最大的一门。一开始看到实验要求时，就感觉这是一门很有系统完整性、任务量很大的课设。做完课设之后真的能够体会到老师们对课设的用心设计，也对自己曾经整体坐在桌前钻研最终完成的结果有成就感。

一开始画单周期CPU时，我先按照整体框架图搭建，把每个功能部件想清楚并且画出一个大概的框架。可是等到真的对着指令来看执行流程进行完善的时候，不得不感叹自己想得太简单了。比如对PC取值我一开始就没考虑到会有那么多种情况。但随着一次次调试，电路慢慢完善，我看到自己搭建起来的CPU能正常运行那一刻的心情就如久旱逢甘霖，也许这就是学习的趣味。

课设过程中第二个遇到的难点就是在单周期改成流水线的时候，一开始只考虑眼前图里的组件摆放，画得很密集，等到画流水线的时候，满世界拖组件，线总是交叉重叠需要调整，磕磕绊绊才完成，导致画出来的图也不太好看。画到重定向流水线的时候更是卡了两三天，对指令使用寄存器情况和检测逻辑总是这漏一点那少一点，最后基本指令完成了，扩展指令又卡了很久。

此外，在中断那一块的实现，对我来说也是很大的挑战。即使我对中断的原理和概念十分清楚，但是动手又是另外一回事。加上可查阅资料大多没有很详细的指导，只能一边摸索一边询问同学。

在团队任务画图片显示模块的时候，一开始没有明白队友想要的表现方式，所以画出来两三个版本都有些事与愿违，好在一步步地调整得到了一个好的结果。

对于本次课程设计，在内容上我觉得设计得非常好，和组原实验一样，都近一步加深了我对理论知识的理解。可能在实验形式上我会有一些简单的建议。第一个是，相对来说实验资料比较散，在组原群里找到一部分，在gitee上找一部分，对于我这样远程实验沟通成本比较大的，一开始实验的时候就有点摸不着头脑。第二个是，组原群长期活跃，大家畅所欲言问问题，这是很好的氛围。但对于不太习惯查看群消息的人来说，打开看到大量消息不知道从哪开始看起，又怕错过重要通知。建议可以把通知放到更明显的地方。而且大家问得比较多的问题也可以开一个FAQ文档，减少沟通成本，尽量避免一个问题问很多很多次但是还是不太知道回答在哪或者一些踩坑的经验分享过了后面的人还是继续跌倒的情况。

最后，在这里感谢谭老师的耐心解答和对组原的用心教学。同时也感谢所有组员的热心回答以及包容我因为未返校而造成的麻烦。我相信这段组成原理课程设计的旅程将在未来成为我珍贵的回忆，组成原理这门课程的知识也会一直在我未来的学习生涯里影响和指导我。

# 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: 邹雅** |