***2019***



**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： | CS1707 |
| 学 号： | U201714785 |
| 姓 名： | 聂鸿勇 |
| 电 话： | 15107122682 |
| 邮 件： | [1902806374@qq.com](mailto:1902806374@qq.com) |

目 录

[1 课程设计概述 3](#_Toc34850764)

[1.1 课设目的 3](#_Toc34850765)

[1.2 设计任务 3](#_Toc34850766)

[1.3 设计要求 3](#_Toc34850767)

[1.4 技术指标 4](#_Toc34850768)

[2 总体方案设计 6](#_Toc34850769)

[2.1 单周期CPU设计 6](#_Toc34850770)

[2.2 理想流水CPU设计 9](#_Toc34850771)

[2.3 气泡式流水线设计 10](#_Toc34850772)

[2.4 重定向流水线设计 11](#_Toc34850773)

[2.5 中断机制设计 12](#_Toc34850774)

[2.6 动态分支预测机制 13](#_Toc34850775)

[3 详细设计与实现 15](#_Toc34850776)

[3.1 单周期CPU 实现 15](#_Toc34850777)

[3.2 理想流水CPU实现 18](#_Toc34850778)

[3.3 气泡式流水线实现 22](#_Toc34850779)

[3.4 重定向流水线实现 23](#_Toc34850780)

[3.5 中断机制实现 25](#_Toc34850781)

[3.6 动态分支预测机制实现 25](#_Toc34850782)

[4 实验过程与调试 26](#_Toc34850783)

[4.1 测试用例和功能测试 26](#_Toc34850784)

[4.2 可自行安排章节 26](#_Toc34850785)

[4.3 性能分析 26](#_Toc34850786)

[4.4 主要故障与调试 27](#_Toc34850787)

[4.5 实验进度 28](#_Toc34850788)

[5 设计总结与心得 29](#_Toc34850789)

[5.1 课设总结 29](#_Toc34850790)

[5.2 课设心得 29](#_Toc34850791)

[参考文献 31](#_Toc34850792)

# 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 $v0==34 halt(停机指令)  else数码管显示$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 | SLLV | 逻辑可变左移 |  |
| 29 | XORI | 异或立即数 |  |
| 30 | LH | 加载半字 |  |
| 31 | BGTZ | 大于 0 转移 |  |

# 总体方案设计

## 单周期CPU设计

在单周期CPU的设计中，使用logisim来完成电路的搭建以及功能正确性的测试，采用硬布线的方案以及程序存储器和数据存储器分离的哈佛结构。支持表 1.1指令集中所列出的所有基础指令、支持扩展指令集中的2条C类运算指令，1条M类存储指令，1条B分支指令。指令syscall的实现方式为，当$v0为34时停机，否则令数码管显示$a0的值。

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

根据改单周期CPU的总体结构图来看，主要功能部件主要分为指令存储器、寄存器组、运算器、控制器、ALU输入控制器，构建数据通路来实现各个部件之间的联系，完成基本指令的执行。

#### 程序计数器PC

利用一个寄存器存储当前执行指令的PC值，根据不同跳转的需求生成各个条件下的下一条可能PC值，通过数据选择器以及控制器产生的跳转控制信号来决定CPU的下一条指令的PC值，并在下一个时钟周期将该值存入寄存器中。

#### 指令存储器IM、数据存储器DM

指令存储器采用rom实现，其输入为取指令的地址，也就是程序计数器的值，而输出则为取出的指令。数据存储器与指令存储器类似，在 logisim 中数据存储器采用 ram 进行实现，同样采用寄存器数组进行实现，与IM不同的是，数据存储器不是只读 的，其输入还包括一个写使能端，除此之外无太大差别。

#### 运算器

运算器用来完成基本的逻辑运算，其输入、输出端位宽以及功能如下表 2.1所示。ALU\_OP的输入值与对应的功能如表 2.2所示。

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 |
| OF | 输出 | 1 | 有符号加减溢出标记，其他操作为零 |
| UOF | 输出 | 1 | 无符号加减溢出标记，其他操作为零 |
| Equal | 输出 | 1 | Equal=(x==y)?1:0, 对所有操作有效 |

表 2.2 ALU\_OP 输入及其对应的功能

| **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 |

#### 寄存器堆RF

在logisim中，Cornell的cs3410已经提供了寄存器文件的实现。

### 控制器的设计

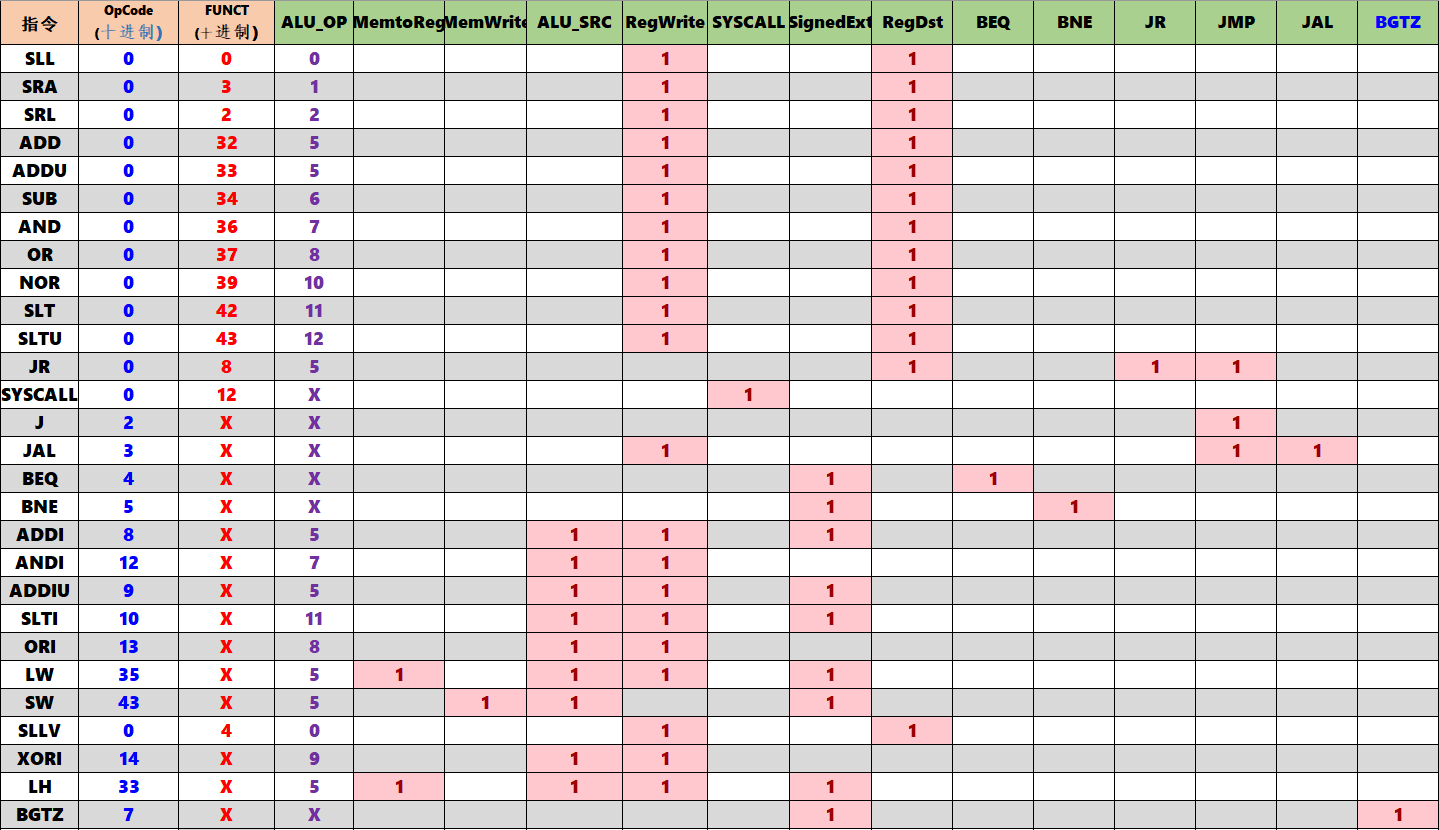
首先对于控制信号进行统计，包括各个主要部件所需要输入的控制信号，以及数据通路合并表中所示的具有多输入的主要部件需要进行输入选择的控制信号，并且对各个统计信号的各种取值情况进行定义，统计得到的控制信号以及说明如表 2.3。

表 2.3主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| 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 |
| RegDst | 写入寄存器编号rt/rd选择 | R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 |
| SignedExt | 立即数符号扩展 | ADDI、ADDIU、SLTI指令 |
| JR | 寄存器跳转指令译码信号 | JR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 |
| JMP | 无条件分支控制信号 | J、JAL、JR指令，选择无条件分支地址 |
| 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 |
| Syscall | Syscall指令译码信号 | 根据$V0寄存器的值，决定是停机还是输出 |

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 2.4所示。

表 2.4主控制器控制信号框架



## 理想流水CPU设计

### 总体设计

理想流水线CPU的设计较为简单，程序无任何数据冲突和分支冒险，所以设计流水线时无需考虑任何冲突冒险的处理。其关键问题在于，阶段数相同：所有加工对象均通过相同的工序；段时延相同：各段传输时延一致，不能有等待现象，取最慢的同步；无资源冲突：不同阶段之间无共享资源，各段完全并发；无段间互锁：进入流水线的对象不受其他阶段的影响，多条指令间存在相关和依赖，仅运行无数据相关，分支相关的程序。对于流水线的实现，从单周期的每一个周期执行一条指令变为了将每一个指令拆分成为5个周期，并在一个时钟周期类同时运行5条指令，而每一条指令所需要的数据直接从流水接口中获得。

### 流水接口部件设计

不同阶段之间增加流水接口部件（锁存器），如图 6.6 所示，在单周期CPU实现基础上需要增加IF/ID、ID/EX、EX/MEM、MEM/WB共四个流水接口部件，四个流水接口均采用公共时钟进行同步，流水接口定义尽可能简化，其内部主要是若干寄存器，用于锁存段间数据。

### 理想流水线设计

流水线通过流水接口部件为后段提供数据信息，控制信息，向前段传递反馈信息，流水线后段对数据的加工处理依赖于前段通过流水接口部件传递过来的信息。ID段译码生成该指令的所有控制信号，控制信号通过锁存器逐段向后传递，后段功能部件所需的控制信号不需要单独生成，直接从锁存器获取。单周期CPU中的控制器可以在ID段直接复用。不同的流水接口部件锁存的数据和控制信号不同，具体可根据前后段之间的交互信息进行考虑，以最为复杂的ID/EX接口部件为例，该锁存器锁存 ID段由控制器产生的所有控制信号，同时还需要锁存由取操作数部件取出的寄存器值或立即数，ID/EX部件设计完成后，其它各段流水接口部件可以直接复制后进行适当精简。

## 气泡式流水线设计

### 总体设计

在理想流水线的基础上改善，分析分支相关、数据相关的处理方案，设计数据相关检测模块，修改增加跳转指令的逻辑。分支相关处理策略：在EX段出现跳转指令后，采用同步清零的方式将IF、ID段所存储的当前状态清零插入气泡；数据相关处理策略：向后段插入气泡（接口信号清零），向前给出阻塞信号（流水线停顿）避免当前指令被新指令取代。若ID段与WB段数据相关：采用先写后读，寄存器文件下跳沿写入，流水接口上跳沿有效；若ID段与MEM段数据相关：IF段、ID段暂停等待数据写回，EX段插入气泡，下一时刻数据相关变成与WB段相关；若ID段与EX段数据相关：IF段、ID段暂停等待数据写回，EX段插入气泡，下一时刻数据相关变成与MEM段相关，再下一时刻数据相关变成与WB段相关。改逻辑需要在ID段增加数据检测逻辑。

### 数据相关模块设计

，由于 ID 段需要取操作数，所以数据相关检测逻辑应设置在 ID 段。需要设计数据相关检测逻辑，当存在数据相关时，可进行插入气泡的处理。数据相关的依据是比较当前指令是否存在源操作数和后续段的目的操作数是否相同。注意在 MIPS 指令集中不同指令所包含的源操作数是不同的，如 R 型指令涉及两个源操作数 Rs、Rt；I 型指令涉及一个或两个源操作数 Rs、Rt；分支指令 （Beq，Bne）涉及两个源操作数 Rs、Rt；J 型指令无源操作数，但会产生控制冲突。注意比较时可能需要和后续三段的目的操作数均做比较，且需要考虑后续段的三条指令是否存在目的操作数，0号寄存器不考虑相关性。

### 气泡流水线设计

进一步改造理想流水线 MIPS CPU，增加数据相关检测逻辑，增加插入气泡逻辑，增加流水暂停逻辑，增加分支冲突处理逻辑，使得该流水线能处理数据冲突、控制冲突、结构冲突，并能正确运行单周期测试程序。

气泡流水线周期数=单周期执行周期数+（流水充满时间−1+𝐽指令×2+条件分支成功数×误区深度+气泡数

## 重定向流水线设计

### 总体设计

气泡流水线通过延缓 ID 段取操作数动作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性的性能，还有一种思路是先不考虑 ID 段所取的操作数是否正确，而是等到实际需要使用这些操作数时再考虑正确性问题， EX 段的指令可能与 MEM 段，WB 段的两条指令均存在数据相关，此时 EX 段取得的操作数应该是错误的数据，正确的数据分别存放在 EX/MEM 以及 MEM/WB 流水接口部件中，还未写回到寄存器中，此时可以直接将正确数据从其所在位置重定向（Forwarding）到 EX段合适的位置（也称为旁路Bypass）， 这样就可以避免插入气泡引起的流水线性能下降，重定向方式可以解决大部分的数据相关问题，可大大优化流水线性能。

### 重定向数据检测模块设计

在气泡流水线的数据相关模块上改进，在新加入的多路选择器会增加一个forward的选择控制信号，这个信号可以在译码段进行数据相关检测时自动生成，但是如果出现相邻两条指令存在数据相关，且前一条指令是访存指令时（称为Load-Use相关），不能采用重定向方式进行处理。

### 重定向流水线设计

重定向流水线需要修改的部分为气泡发生器的部分以及 EX 段流水线的部分。进一步改造气泡式流水线 MIPS CPU，增加重定向机制，增加Load-Use数据相关检测机制，使得流水线能在不插入气泡的情况下处理大部分数据相关问题，最终能正确运行单周期测试程序，这样，就完成了从气泡流水线到重定向流水线的转换。

重定向周期数=指令条数+（流水冲满时间-1）+分支冲突次数\*预取深度+LOAD-USE数。

## 中断机制设计

### 总体设计

设计中断按键信号产生电路、设计中断识别逻辑、存储区间规划、实现 CP0 寄存器组中与中断相关的寄存器、设计中断隐指令数据通路、增加中断返回指令 eret的支持、编写中断服务程序、系统联调，功能测试。需要修改PC以及控制器的逻辑，使之能够支持eret、mtc0、mfc0指令，并在eret时能够正确的从EPC中读取保存的PC值。在流水中断中实现WB段中断实现eret的逻辑与无条件跳转的逻辑类似，需要清空所有段中的当前指令，避免执行eret之后的指令。

### 硬件设计

在单周期单级中断的实现中，最主要的部分在于控制当前是否能够发生中断，保存中断执行完毕后的PC返回值，该部分可以使用EPC、IE寄存器来实现；而流水线单级中断的实现，需要考虑的是PC的保存以及流水线的清空。多级嵌套中断的屏蔽与识别需要能够正确的处理中断屏蔽寄存器以及中断请求寄存器的关系即可。需要注意的是在实现中断请求寄存器时，需要考虑低级中断尝试打断高级中断时不能直接忽略而是需要将其存入请求寄存器中，否则低级的中断不能在高级中断返回时被正确响应。

### 软件设计

对于中断程序而言，主要需要考虑的是保存的寄存器的值。通过将寄存器压入堆栈，并在中断返回前从堆栈中弹出寄存器的值来保护所有在中断处理程序中需要用到的寄存器。值得注意的是，EPC的值也需要保存，防止被更高级的中断打断而覆盖了EPC的值。EPC的值的保存需要首先使用mfc0读取其值然后压栈，在返回前首先从栈中弹出其值并使用 mtc0 来还原这就需要cpu额外实现对这2条指令的支持。

## 动态分支预测机制

### 总体设计

采用重定向机制后，指令流水线中数据相关基本不需要插入气泡就可解决，只有少数 Load-Use 冲突需要插入一个气泡解决冲突问题，指令流水线性能得到较好的提升。此时指令 流水线中结构冲突（分支冒险）对流水线性能影响很大，分支指令会使指令流水线暂停，因为 分支使得 IF 段以及 ID 段执行的指令（预取指令）被清空，这部分流水线性能损失称为分支延迟。指令预取深度越长，分支延迟越大。为减少分支延迟，应在流水线中尽早判断出分支是否 成功跳转，并尽早计算出分支目标地址，比如将分支指令放在 ID 段完成，这样预取深度就是1，分支延迟也只有一个时钟周期，进一步降低分支延迟。

### 双位预测设计

根据下图有限状态机来设计双位预测机，将其分为成功与失败两种情况，使用两个数据选择器来实现各个状态之间的转换。

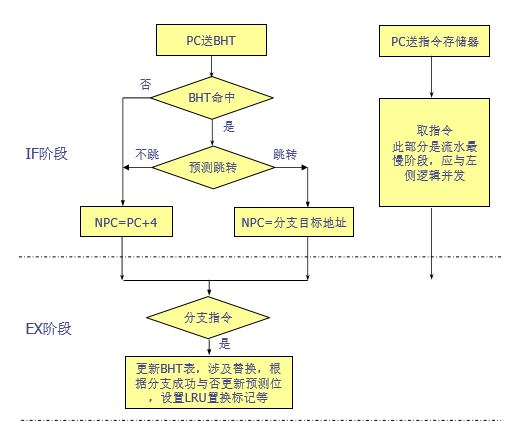


### BHT设计

在这里因为没有想到合适的标志位来进行分组，所以采用全相联的方式实现BHT，具体结构如下表所示。

BHT 表会放在取指令阶段，利用 PC 的值作为关键字进行全相联比较（此过程应与取指 令过程并发），BHT 表命中表示当前指令一定是一条分支指令，此时可以根据 BHT 表中的历 史预测位决定下条指令的地址是 PC+4 还是分支目标地址，如果预测正确，指令流水线不会停 顿，如果预测失败，则分支指令在实际执行阶段还是应该清空预取的指令。当然如果 BHT 表缺失，表明当前指令可能不是一条分支指令或者是不经常使用的分支指令，则按照 PC+4 取下条指令，如 BHT 表已满，需要进行淘汰。如 BHT 表命中，需要根据 当前跳转成功与否情况更新双预测位，同时将 LRU 置换计数信息清零，以提升下次预测的准确性。BHT 表的引入使得取指令 IF 阶段可以在指令并未取出的情况下进行分支预测，由于双位预测的高准确率，可以消除指令流水线中的大多数的分支延迟。





# 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）
2. Logism实现：

使用一个32位寄存器实现程序计数器PC，触发方式为上降沿触发，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。中止为停机信号与PC寄存器使能端相连，当需要进行停机时，中止控制信号为0，，使整个电路停机。在正常情况下，每一个时钟周期 PC+4，若控制器输入跳转指令则将PC修改为对应的跳转目标地址。如图 3.1所示。

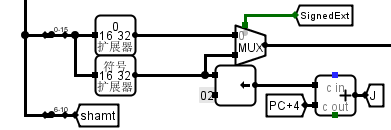
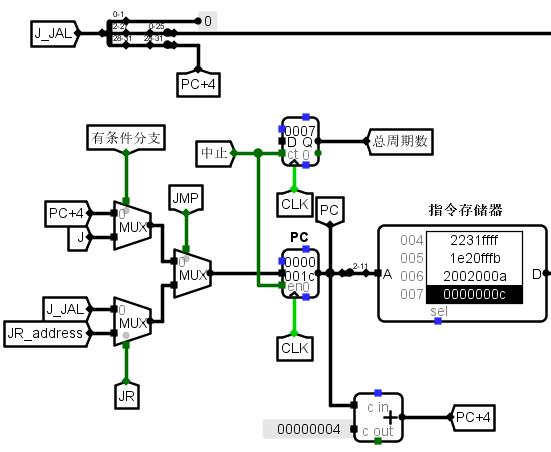


图 3.1程序计数器（PC）

1. 指令存储器（IM）
2. 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。

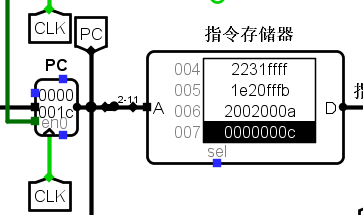


图 3.2指令存储器（IM）

### 数据通路的实现

查找MIPS指令手册，对照24条基本指令加4条拓展指令的功能，利用指令存储器、控制器、寄存器组、运算器、数据存储器，逐条指令分析其功能，记录各个主要部件的输入输入来源输出数据的去向，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，，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。在处理syscall停机指令时，需要处理的是根据寄存器的值来决定是停机还是在数码管上显示数据，若$v0==34 halt(停机指令)，数码管显示$a0值。对于4条拓展指令，主要介绍一下LH指令（加载半字）的实现，在完成24条基本指令的基础上添加这四条指令，LH指令将冲数据存储器中获得的数据分为高16位和低16位，通过判断访问数据的地址第一位是0还是1选择低16位还是高16位经过有符号拓展为32位写入寄存器组中。

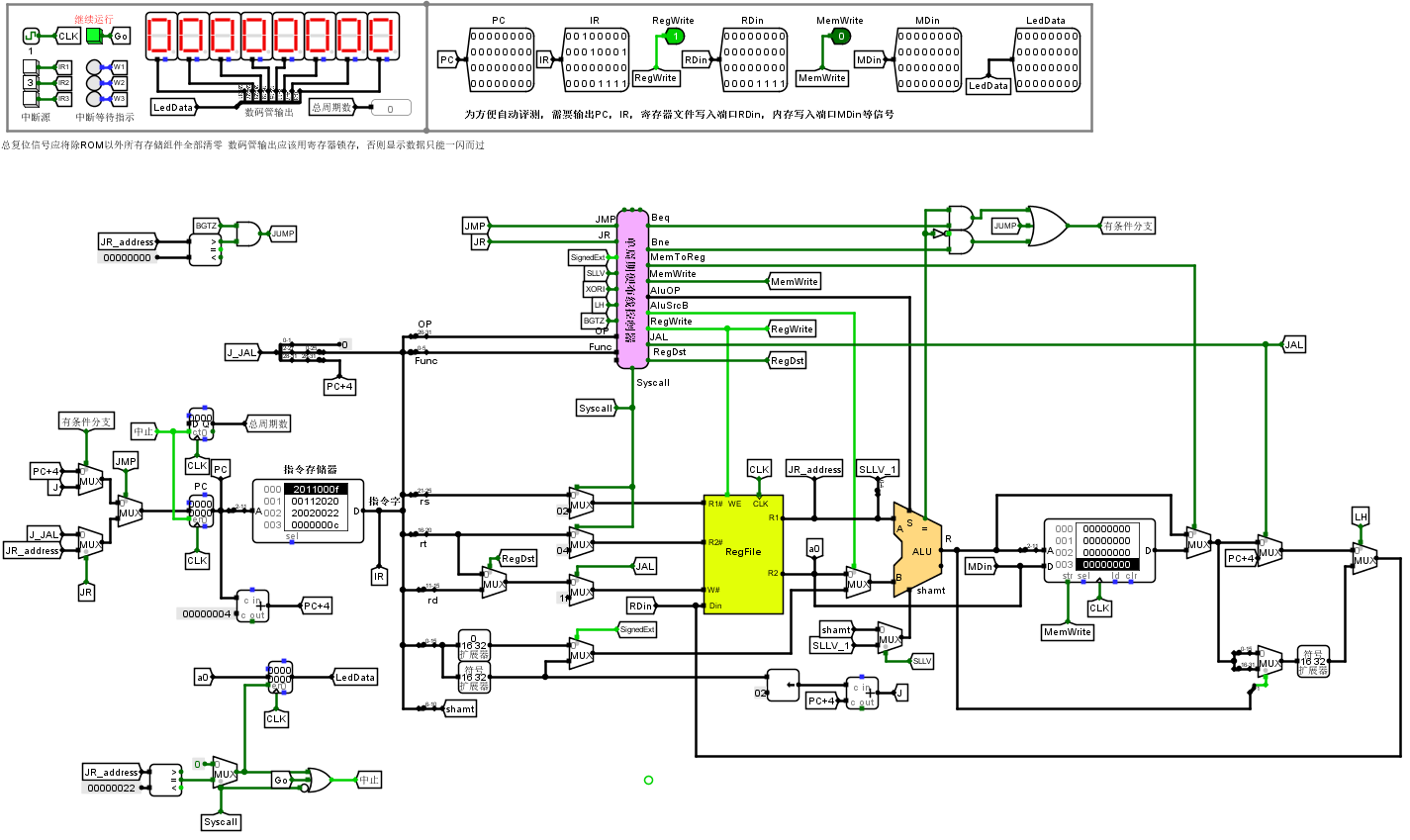
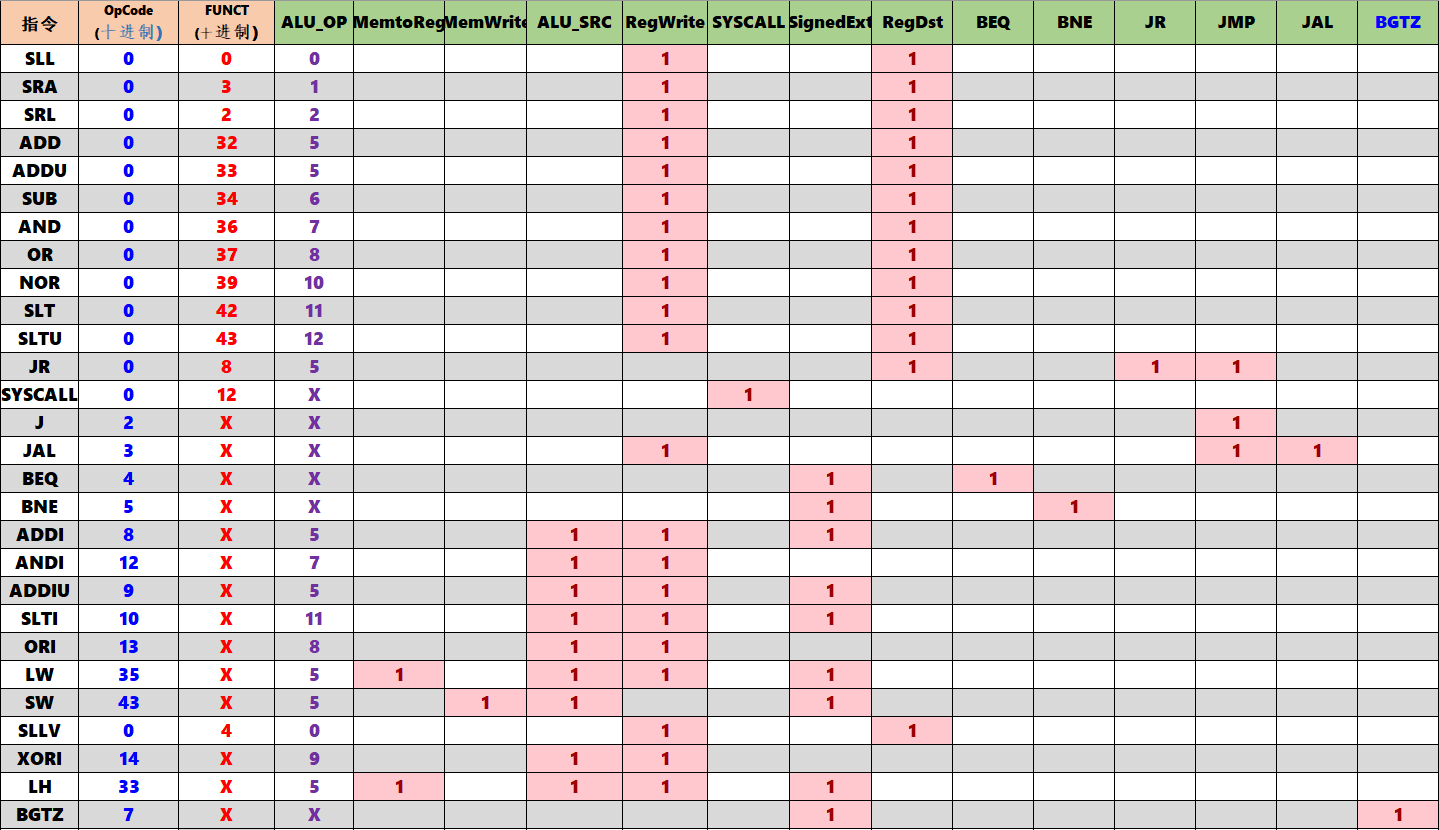


图 3.3 单周期CPU数据通路（Logism）

### 控制器的实现

根据指令功能来判断正确执行该功能时需要使用到那些控制信号，在 logisim 中，控制器使用硬布线的方式进行，根据如表3.2所示的输入指令与控制信号的对应，然后在 logisim 中进行相应的布线。。

表 3.2主控制器控制信号



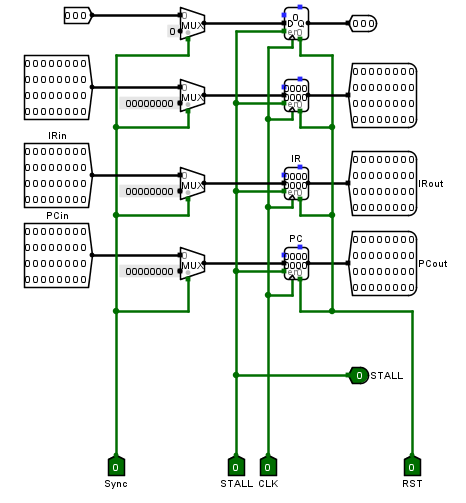
## 理想流水CPU实现

### 流水接口部件实现

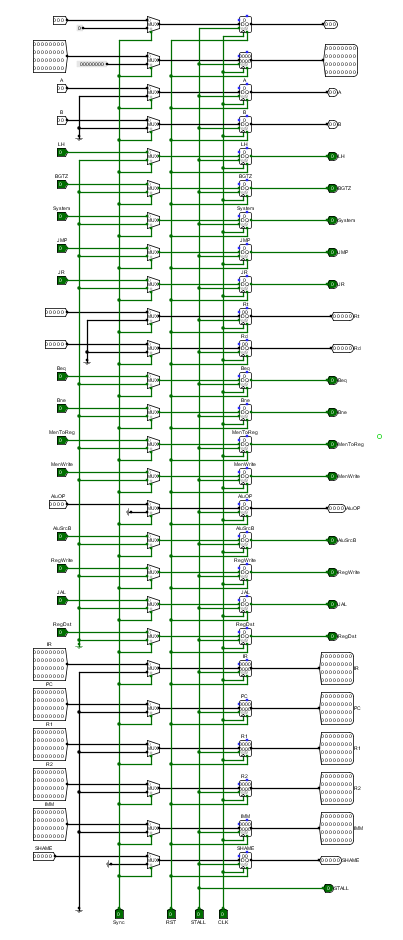
将单周期 CPU 的 实现拆分为 5 个不同的阶段，接口部件分别为：IF/ID、ID/EX、EX/MEM、MEM/WB。(接口部件为最终版本-拓展适用于所有流水线结构)

#### IF/ID

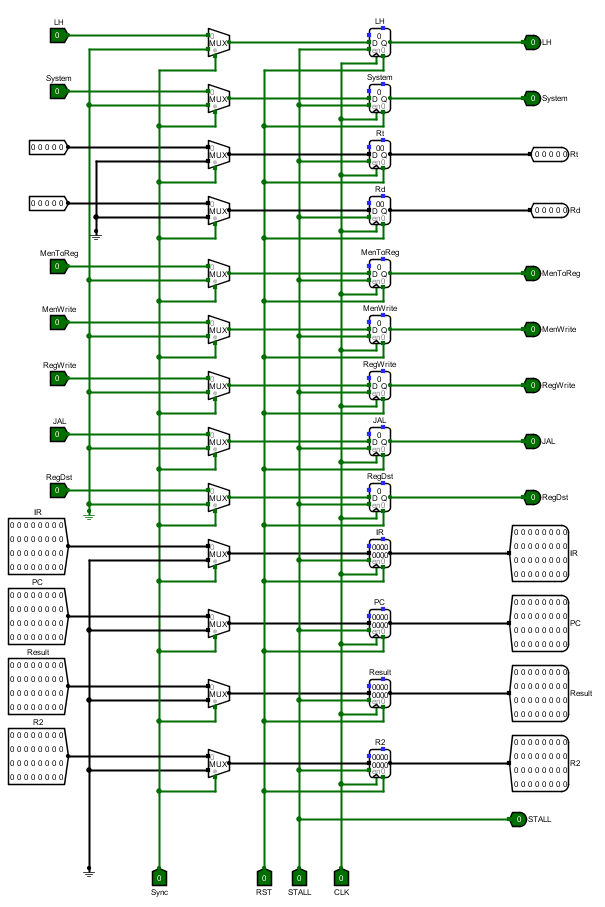
ID段负责取出指令，向后传递IR与PC值。



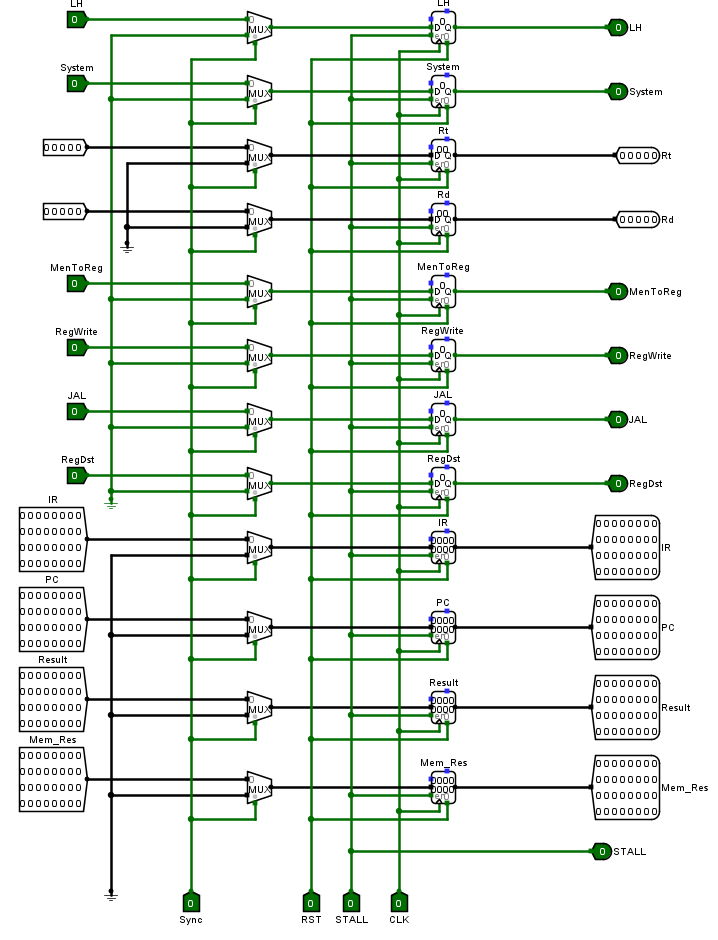
#### ID/EX



#### EX/MEM

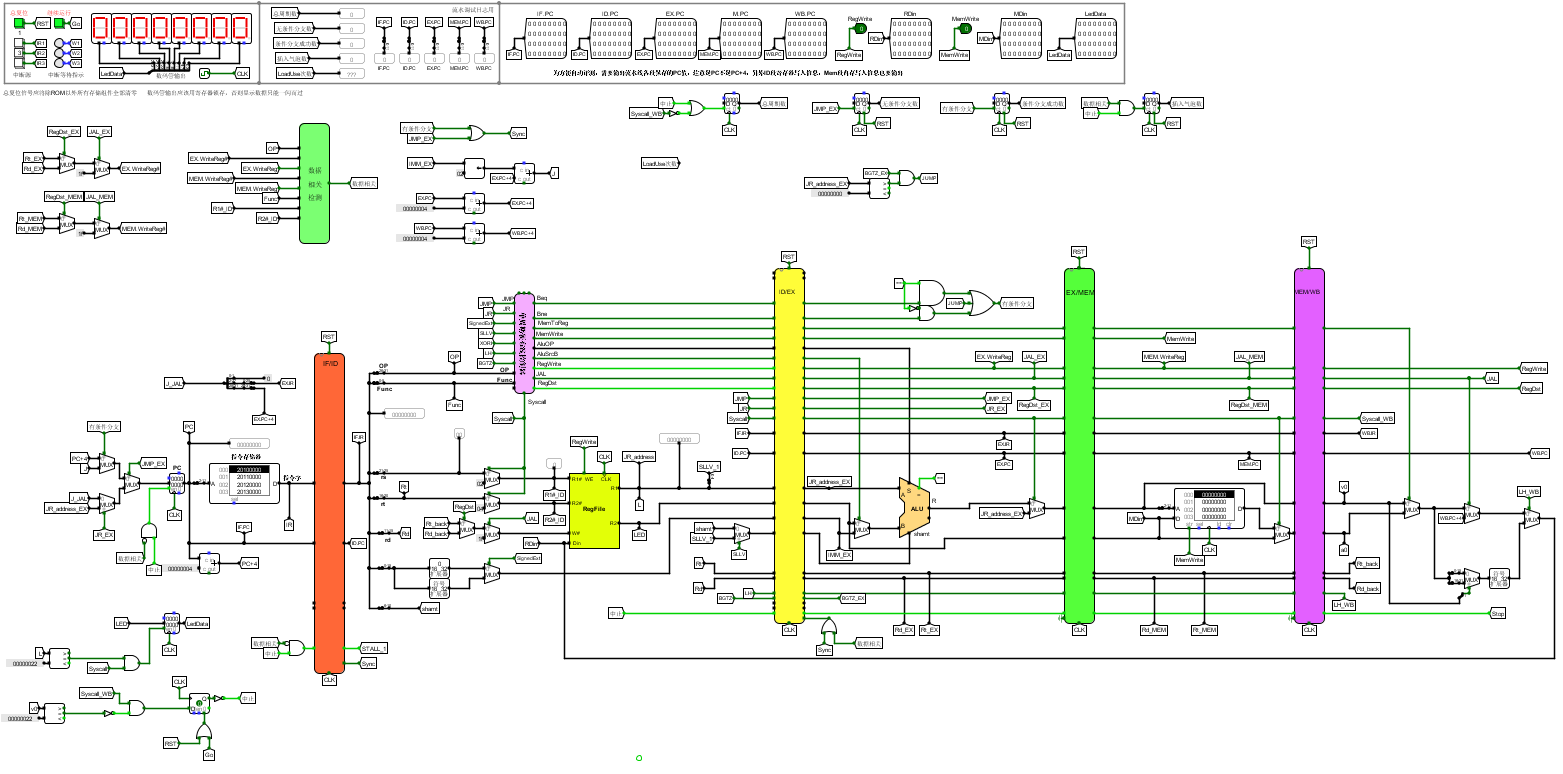


#### MEM/WB



### 理想流水线设计

实现了单周期CPU后，理想流水线的实现则较为容易，只需将单周期CPU的实现拆分为5个不同的阶段即可。理想流水线的实现所示。（气泡流水线是在理想流水线的基础上完善而来）。将对应的控制信号逐级向后传递，在每个段分别执行所需要的操作。



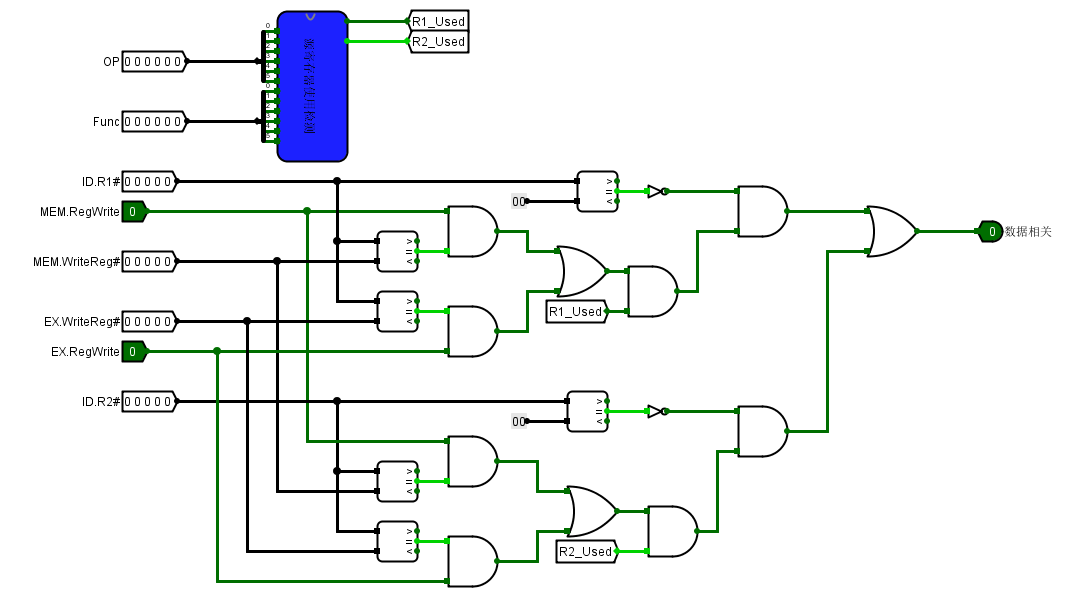
## 气泡式流水线实现

### 源寄存器使用检测设计

ID段包括0~2个源寄存器：R1、R2是否使用标识：R1\_Used、R2\_Used。例如：运算指令含两个源操作数、syscal指令含两个源操作数、JR指令含一个源操作数，以此类推，在控制器生成表格中填写个指令使用源寄存器的情况，根据生产的表达式在logisim中自动生成源寄存器使用检测电路。

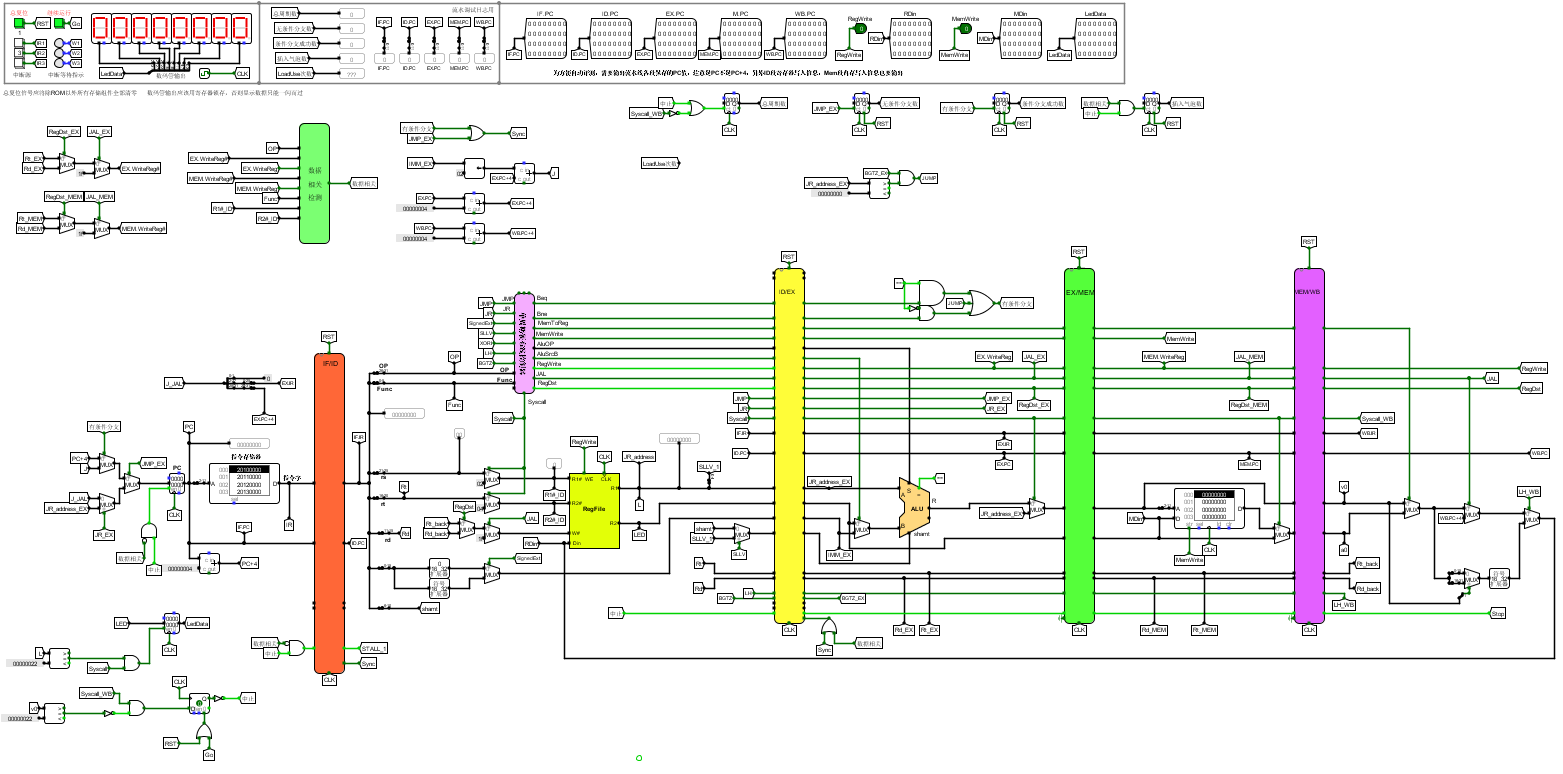
### 数据相关检测设计

MEM.WriteReg#为MEM段所需要使用的源寄存器号、EX.WriteReg#为EX段所需要使用的源寄存器号、ID.R1#、ID.R2#为当前判断指令可能需要使用的2个源寄存号，通过源寄存器使用检测生产的使用标识R1\_Used、R2\_Used，来判断流水线中是否存在与EX段或者MEM段的数据冲突，并将数据相关信号输出。



### 气泡式流水线设计

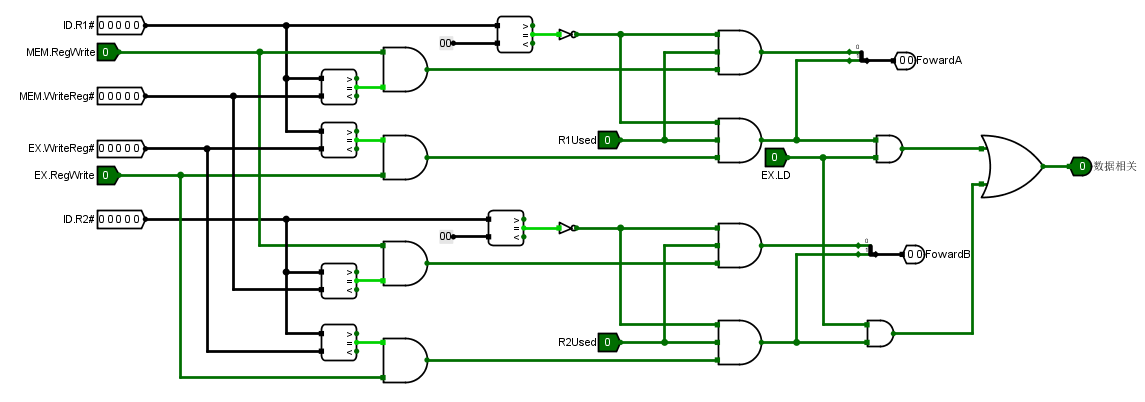
在理想流水线的基础上，向电路中添加数据相关检测模块，输出为数据相关控制信号，将IF/ID段的使能端改为用中止控制信号和数据相关控制信号一起控制，在发生数据相关时，将IF/ID段锁住，添加数据相关控制信号控制ID/EX段的同步清零处，向EX段插入气泡。



## 重定向流水线实现

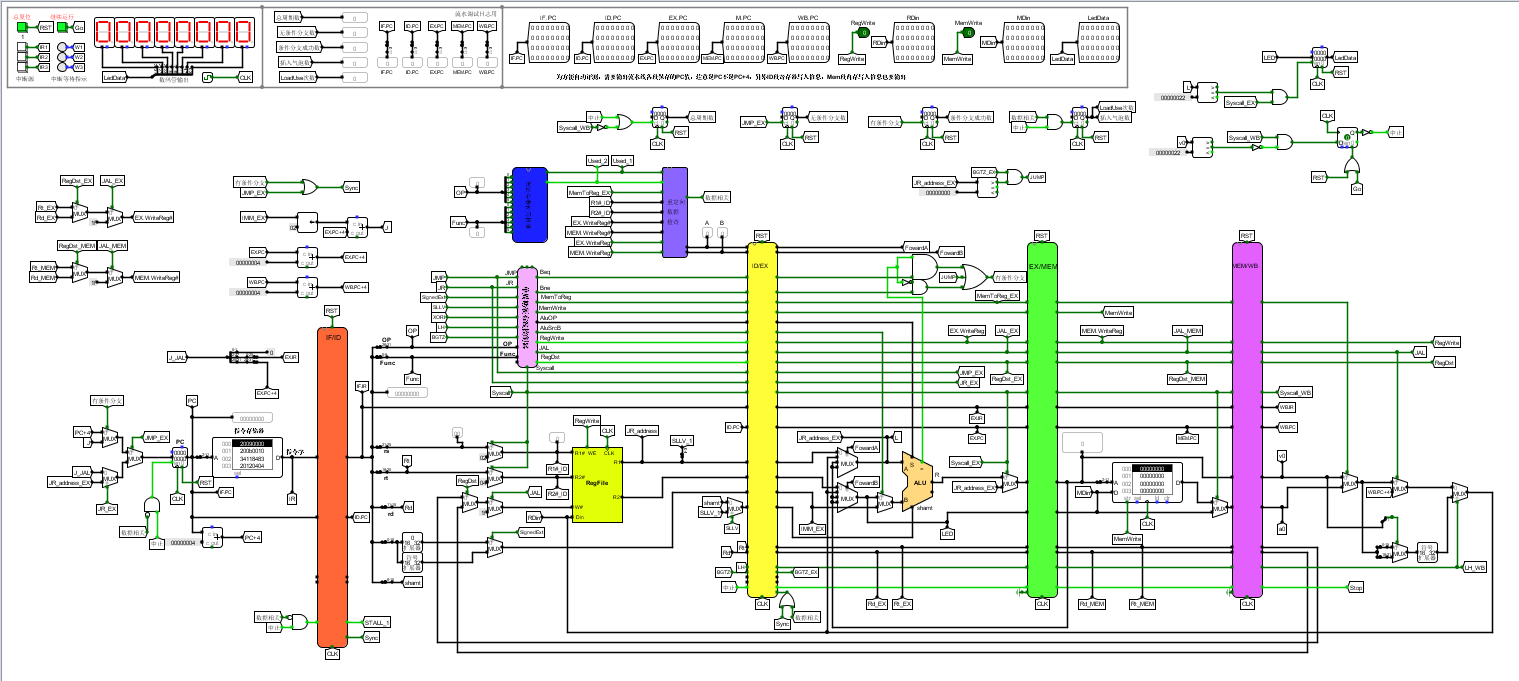
### 重定向数据相关检测设计

为了插入过多的气泡影响流水线的工作效率，采用重定向来解决大部分的数据相关冲突、但是对于load-use情况无法使用重定向的方式来解决，所以在数据相关的基础上改进电路，对于在EX段的数据相关额外添加一项判断，即当前指令是否需要前一条指令的写回寄存器的值，若需要则输出数据相关信号为1，若不需要，则根据改进后的运算器输入端来生成对应情况下的fowardA、fowardB，用这两个信号当作输入端的数据选择器的选择控制相应的输入数据，减少插入气泡带来的效率损失。



### 重定向流水线设计

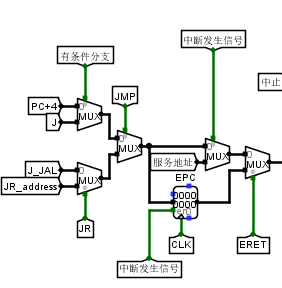
在EX段更改数据通路，将可能出现能够通过改变数据通路解决的数据相关情况下所需要的数据通过数据选择器连接在ALU的输入端口，分别通过fowardA、fowardB来控制进入的数据；在ID段通过源寄存器使用检测和重定向数据检测输出控制信号数据相关、fowardA、fowardB，将fowardA、fowardB通过ID/EX段传入EX段中，实现重定向减少气泡插入。



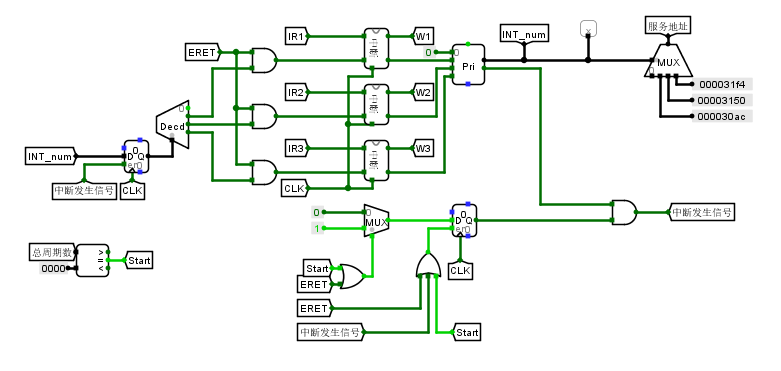
## 中断机制实现

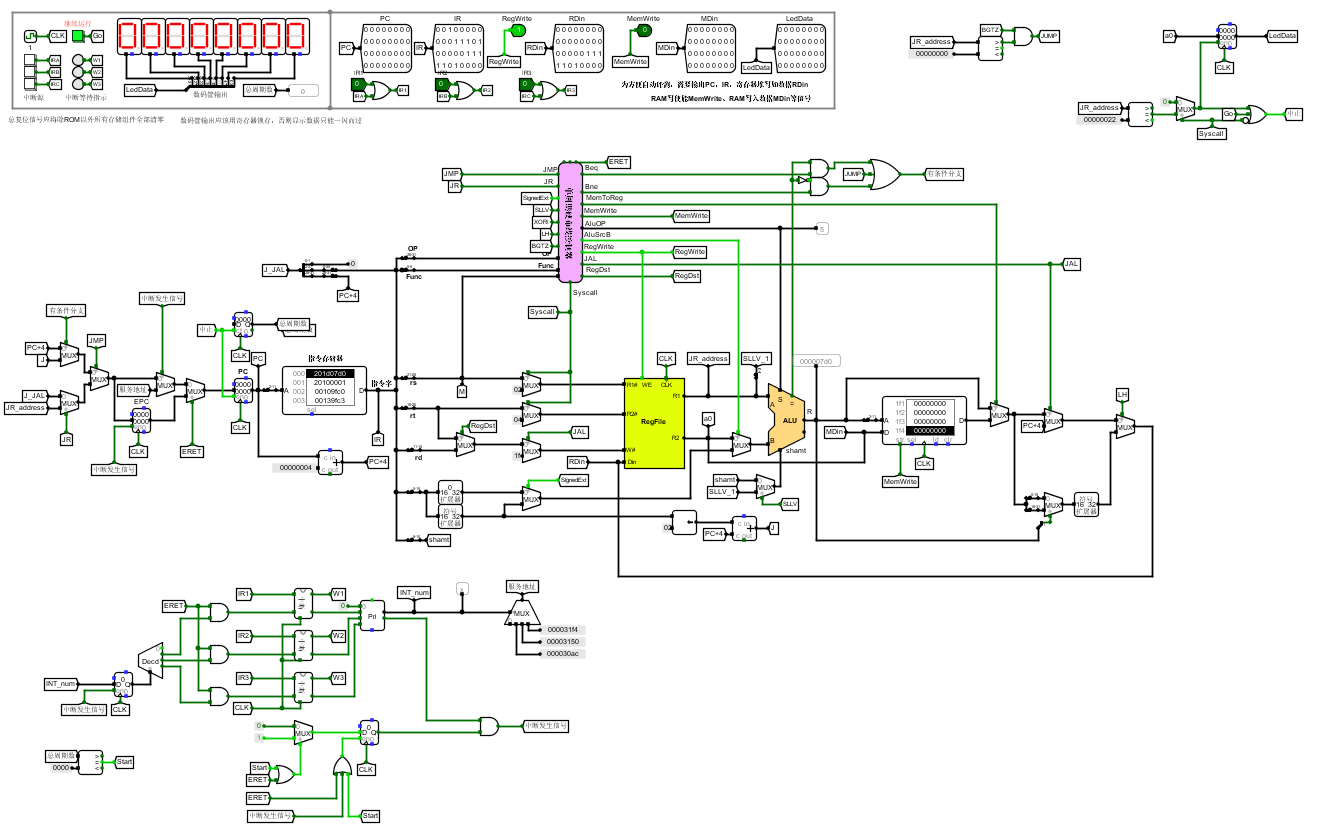
### 单周期MIPS CPU单级中断设计

首先改进程序计数器PC，添加一个EPC寄存器存储中断地址的返回值，在中断发生后，利用数据选择器将PC跳转到指定中断号的服务地址，如下图所示。



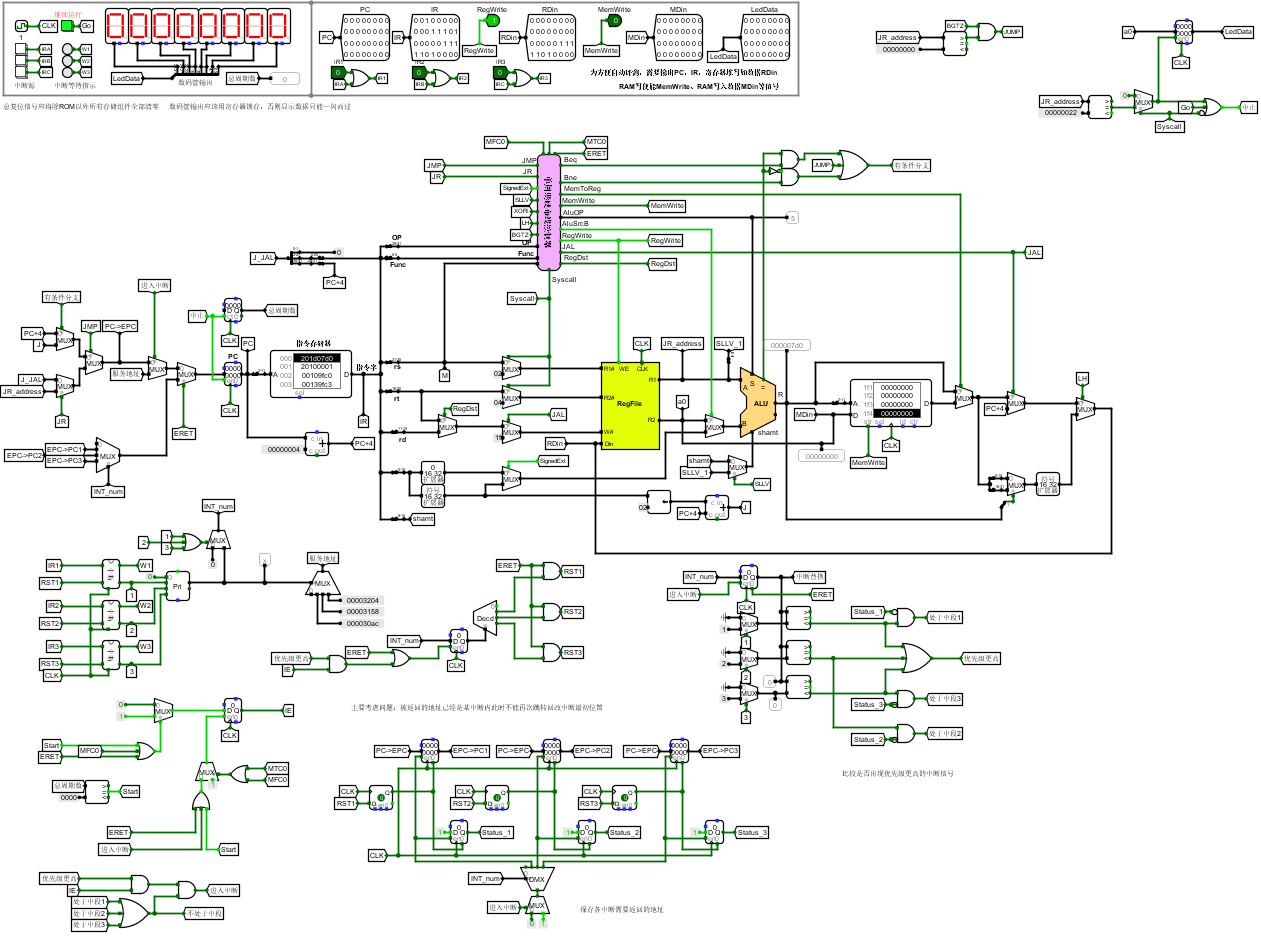
添加中断处理机制，在cpu刚开始运行时，将存储中断能否发生的寄存器置为1，在该寄存器值为1时，可以接受中断并跳转到中断服务程序，当进入中断服务程序后该寄存器置为0，中断服务程序结束后执行eret指令再次将该寄存器值置为1，利用寄存器来控制中断是否发生；在同时有多个中断发生时，使用优先编码器来决定优先度大的中断号先执行。如下图所示。





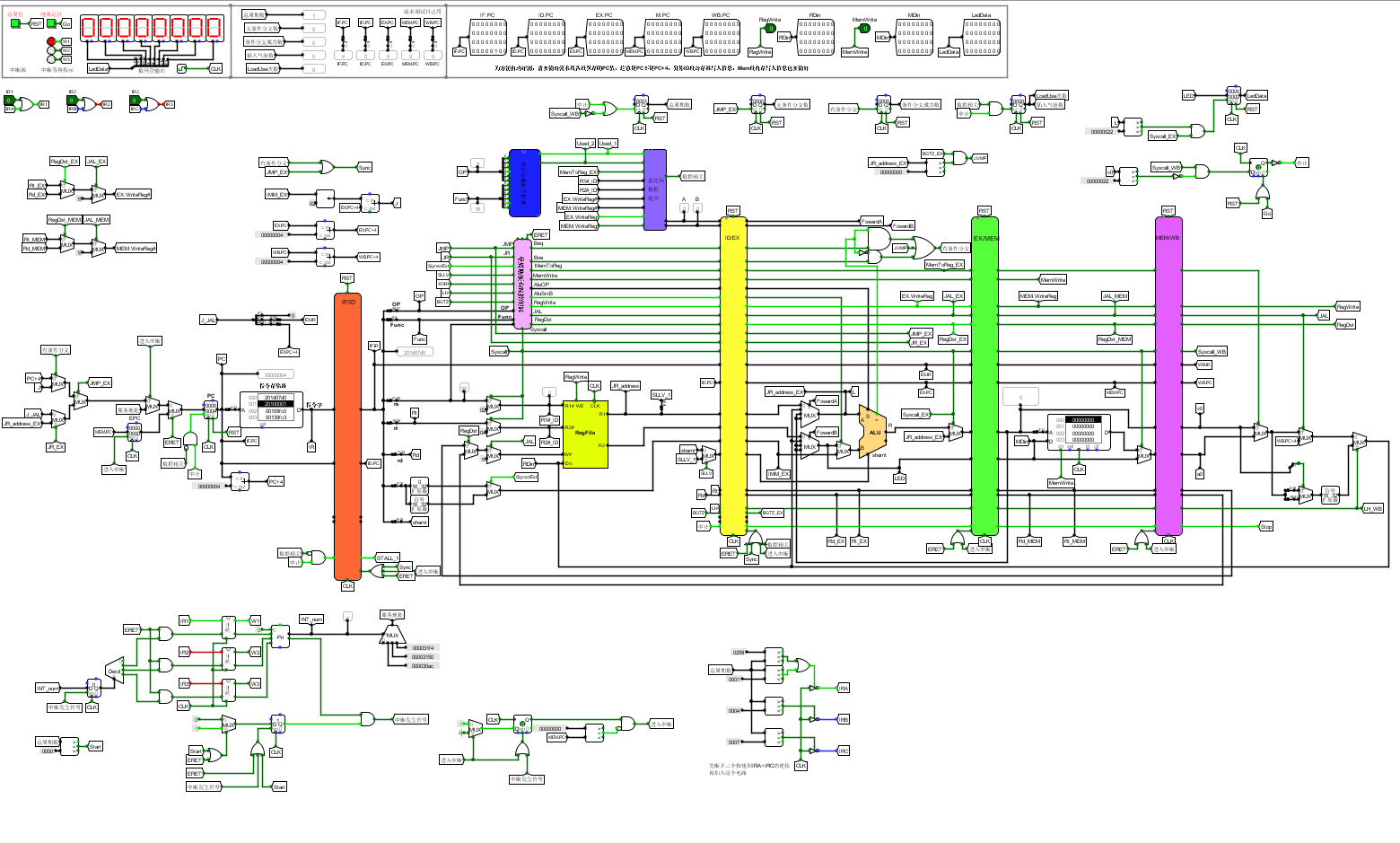
### 单周期MIPS CPU多级中断设计

多级中断与单级中断的不同之处在与可以在中断中执行中断。因此，需要使用三个不同的寄存器来存储3个中断号发生后结束时需要返回的地址，同时，需要判断在中断服务程序中是否存在优先级更高的中断号出现。



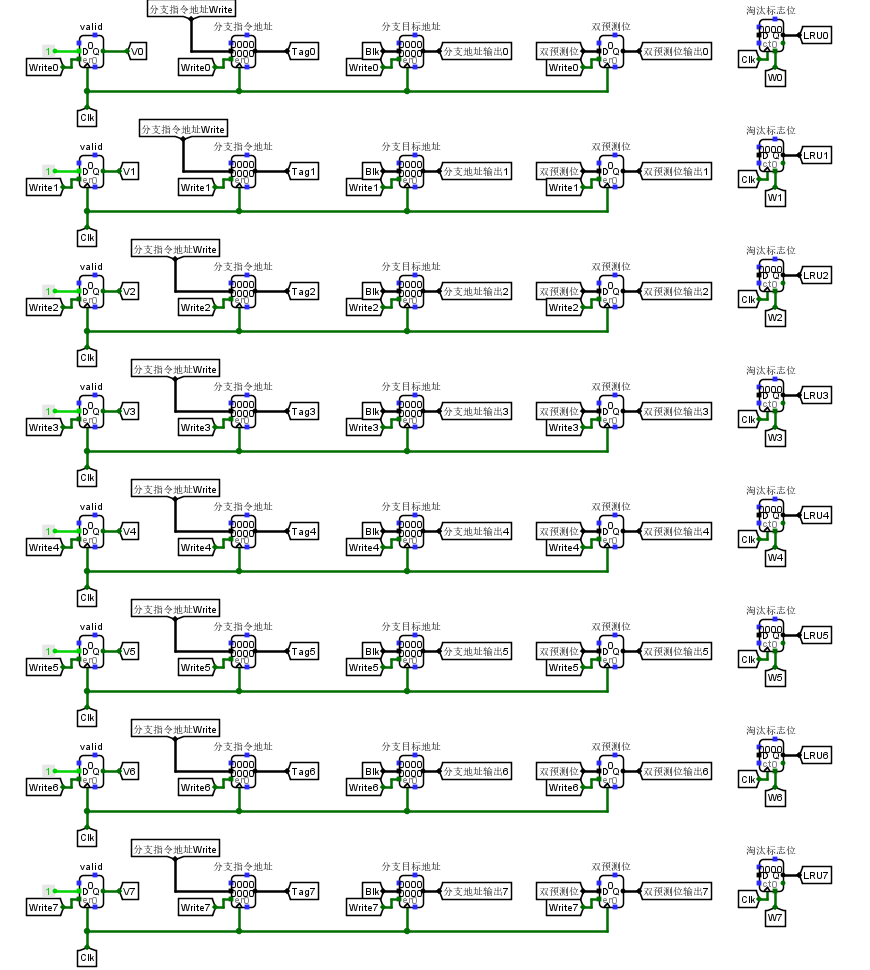
### 重定向流水线CPU单级中断设计

该部分同单周期MIPS CPU单级中断设计，不同的是在执行eret指令后，所有段所存储的数据要被清空。而且在进入中断时需要判断当前存储的返回值是否是气泡值，若是气泡值则需要等待下一个时钟周期，知道有效的PC返回值出现。

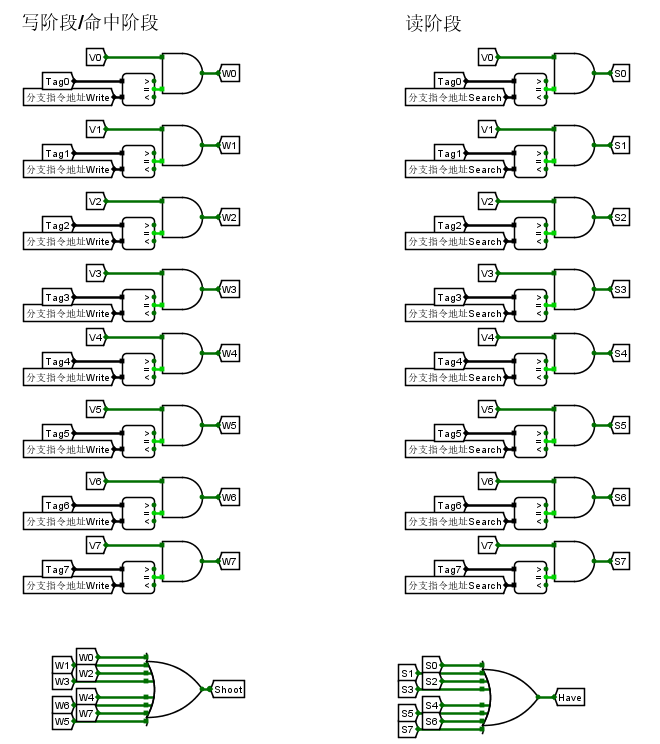


## 动态分支预测机制实现

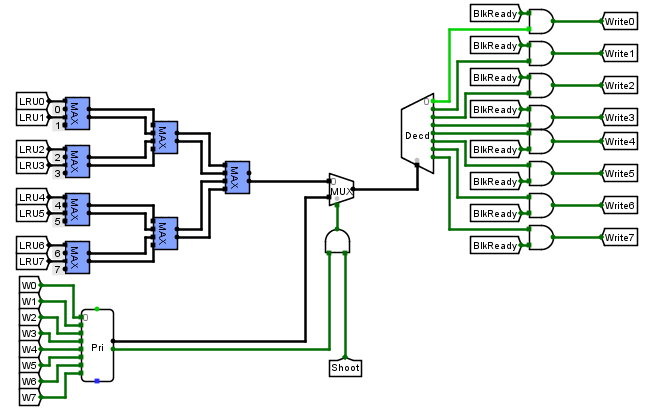
### BHT设计



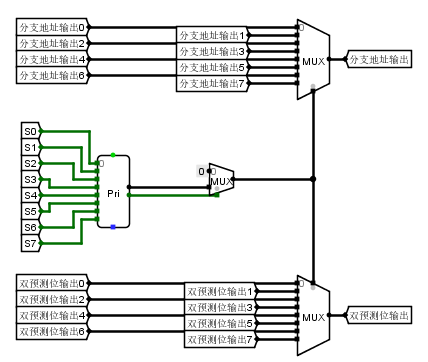
下图分别表示在读阶段以及在写阶段判断当前指令是否存在BHT中，根据相应的结果进行下一步的判断。



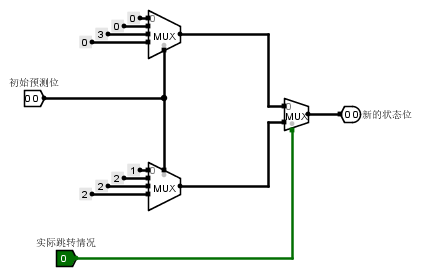
若在写入阶段，在BHT中命中指令，则根据相应的写入地址更新对应块的数据，若BHT没有命中或BHT已满则根据LRU原则判断写入地址。



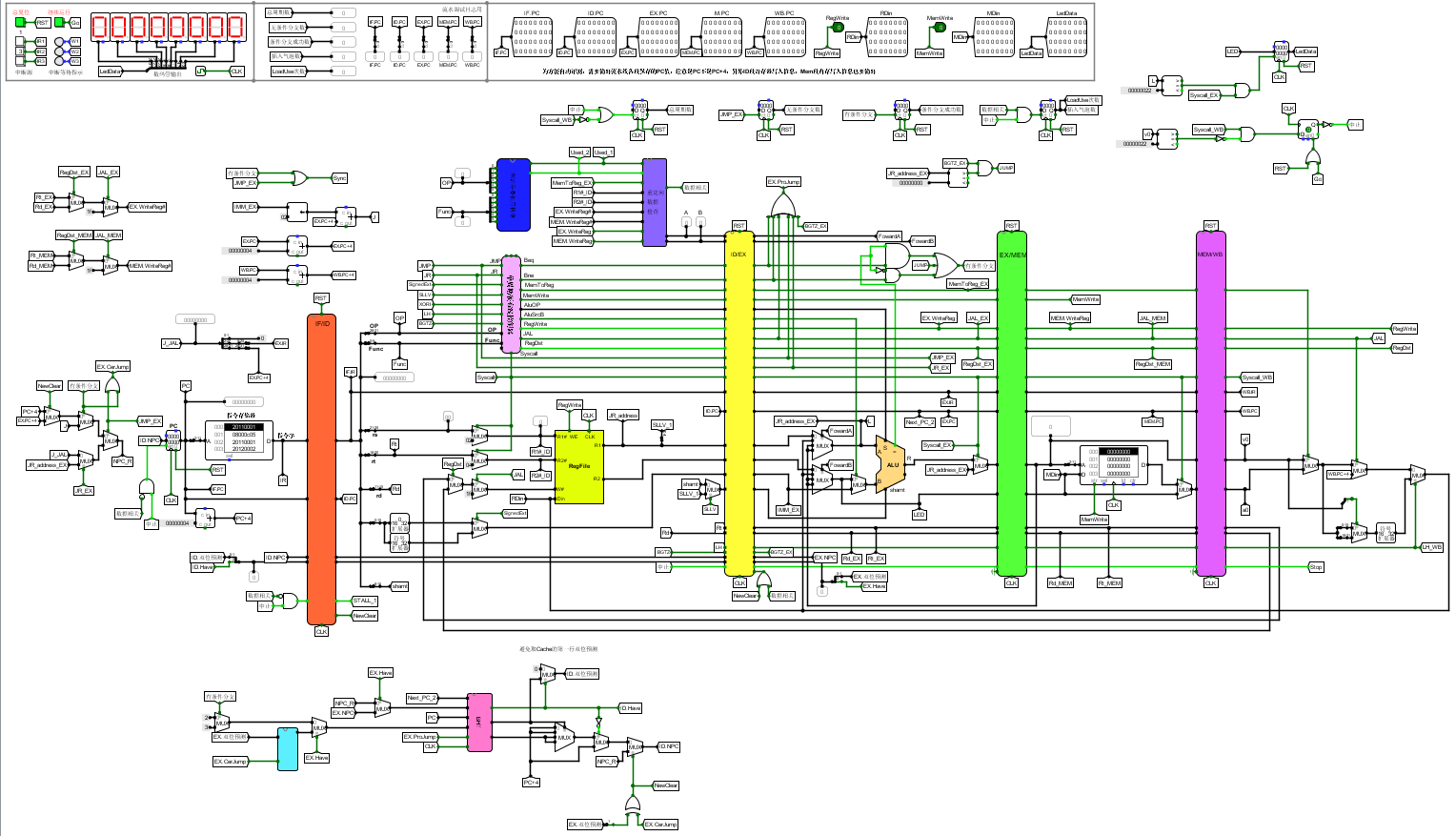
在读阶段，若查找到对应信息，则输出对应的BHT表数据信息。



### 双位预测设计



### 数据通路设计

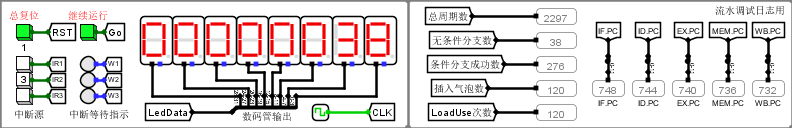


# 实验过程与调试

## 测试用例和功能测试

### benchmark以及ccmb指令测试

使用提供的benchmark文件，并在其后添加SLLV、XORI、LH、BGTZ指令，编译后导入logisim中对于重定向流水线进行测试，并观察程序的表现。在benchmark阶段，程序正确的通过了移位测试、走马灯测试以及排序测试。在排序测试结束后，打开内存查看器查看ram中的内容，如图所示，与预期的降序排序结果一致。





#SLLV测试：依次输出 0x00000876 0x00008760 0x00087600 0x00876000 0x08760000 0x87600000 0x76000000 0x60000000 0x00000000

#XORI测试：依次输出0x00007777 0x00008888 0x00007777 0x00008888 0x00007777 … 0x00008888 0x00007777

#LH测试：依次输出0xffff8281 0xffff8483 0xffff8685 0xffff8887 … 0xffffbebd 0xffffc0bf

#BGTZ测试：依次输出0x0000000f 0x0000000e 0x0000000d 0x0000000c 0x0000000b …

… 0x00000005 0x00000004 0x00000003 0x00000002 0x00000001

程序最后的周期数为2297，无条件分支指令数为 8，无条件分支数38，有条件分支成功跳转数为276，如图所示，均与统计结果一致。这些结果说明benchmark 部分测试通过，基本指令实现正确。对比文档中拓展指令SLLV、XORI、LH、BGTZ的执行结果看，拓展指令功能执行正确。

### 单级中断测试

使用单级中断测试程序文件，加载文件后，在Educoder上使用平台自动测试，对比每一时钟的各数据值是否正确，电路成功通过测试，单级中断功能测试正确。

### 多级中断测试

使用多级中断测试程序文件，加载文件后，在Educoder上使用平台自动测试，对比每一时钟的各数据值是否正确，电路成功通过测试，多级中断功能测试正确。

## 性能分析

程序benchmark是比较基础的测试程序，但也能够考虑到各条指令的执行次数以及功能完成度，通过程序benchmark的测试结果来看单周期MIPS CPU执行benchmark需要时钟周期……

## 主要故障与调试

### 流水线控制信号故障

流水线电路：部分功能实现不正确。

**故障现象：** 流水线上指令的执行完全混乱。

**原因分析：** 在单周期MIPS CPU的基础上改进得到的流水线电路，只是单一的传递了各段所需要的控制信号，在使用这些控制信号的时候依旧保留着部分单周期MIPS CPU的数据通路，导致对器件的各段控制信号的使用不正确。

**解决方案：** 将各段需要控制信号的器件所使用的控制信号该为正确值，解决问题。

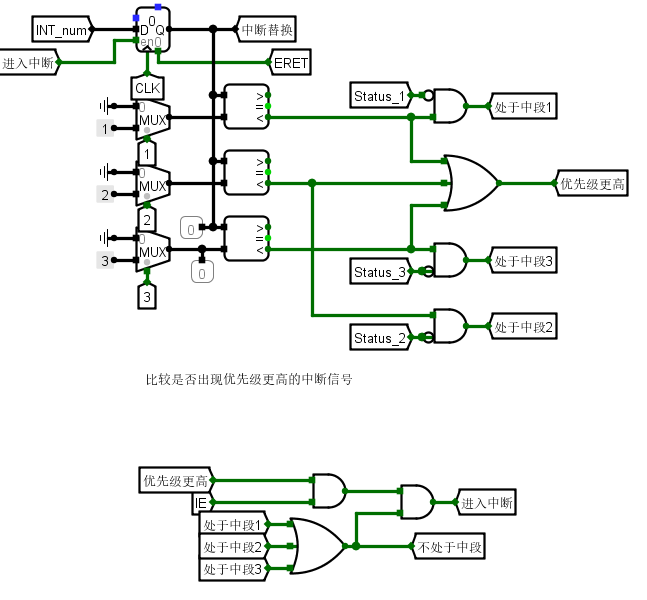
### 单周期MIPS CPU+多级中断判断是否出现优先级更高故障

单周期MIPS CPU多级中断：中断嵌套时出现问题。

**故障现象：** 当已经处于中断1服务程序中时，再出现优先级高的中断号，程序会进一步进入优先级更高的中断服务程序，但是当该服务程序结束执行eret指令后，会重新进入中断1服务程序，没有正确实现中断嵌套的功能。

**原因分析：** 在优先级的判断上出现了问题，当在中断1中嵌套更高优先级的中断返回后，在优先级的判断上，会误判为返回后处于无中断的情况而中断1正在等待，所以程序会重新进入中断1的服务程序，导致故障的发生。

**解决方案：** 在电路中添加判断逻辑，判断当前电路是否存在在某一中断号服务程序当中，必须在出现更高优先级的中断号且并没有处于该中断号服务程序当中，才能够进入该中断号的服务程序。



## 实验进度

表 4.1 课程设计进度表

| 时间 | 进度 |
| --- | --- |
| 寒假 | 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，填写单周期MIPS CPU-24控制器生成表格，并完成数据通路的基本构建。并通过Educoder的检测。 |
| 第一天 | 想控制器生成表中添加自己的四条拓展指令，并在24指令的数据通路上完善，加载拓展指令的测试程序，验证结果的正确性。完成基本的理想流水接口部件，将24+4单周期数据通路分割为5部分，插入接口部件，实现理想流水线。 |
| 第二天 | 在理想流水线的基础上实现分支相关、数据相关处理，加载测试程序，不断调试，分析问题，最终完成气泡流水线。 |
| 第三天 | 在气泡流水线的数据相关模块上加以改进，完成重定向数据检测模块，更改数据通路，完成重定向流水线。 |
| 第四天 | 在24+4单周期MIPS-CPU中加入单级中断。 |
| 第五天 | 在重定向流水线中加入单级中断。 |
| 第六天 | 在24+4单周期MIPS-CPU中加入多级中断。 |
| 第七天 | 查看动态预测资料。 |
| 第八天 | 实现动态预测电路 |

# 设计总结与心得

## 课设总结

本次课程设计通过循序渐进的方式完成了从单周期 CPU 到功能完善的多级 CPU 的实 现，在这个过程中具体做了如下的工作：

1） 完成了对于单周期CPU、理想流水线CPU、气泡流水线CPU、重定向流水线CPU、支持中断以及分支预测的重定向流水线CPU的设计与logisim实现，并完成了单周期CPU、重定向流水线CPU以及支持中断和分支预测的，通过Educoder实训平台进行了电路功能性正确的验证。

2） 单周期CPU支持包括24条基本指令+3条中断指令+4条拓展指令。在中断的实现当中，完成了单级中断以及用硬件堆栈实现的多级中断。

3） 在重定向流水线CPU中实现了单级中断以及动态分支预测。

## 课设心得

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。两个星期从早到晚的不懈努力以及国庆节假期的辛苦加班才终于完成了整个课程设计的设计任务。现在再来回顾整个课程设计的整个过程，满满的成就感自是不用说，但是其中也有不少的细节值得我去深思与体会。

课程设计刚刚开始的时候，第一个任务是使用Logism设计单周期CPU，该任务和上学期实验课的任务类似，所以整个过程还算比较迅速。然而，在将该电路使用Verilog语言进行描述时，麻烦接踵而至。因为Logism对于电路是完全可视化的，连接过程清晰明了，但是使用Verilog语言进行数据通路时，因为各个部件的接口在定义时起名不是很规范，又因为只能使用各个wire型变量对于关键部件进行连接，而这些变量的名字在定义时同样没有做到一目了然，从而导致了我在完成了数据通路的连接后，使用了大量的时间对于整个电路进行排错和检查，最终还是将所有的接口和连线规范化的起了名字，才终于解决了问题。这也就说明了工程化、规范化对于电路设计或是程序设计的重要性。

紧接着，理想流水线CPU的设计并没有什么难度，但是使用插入气泡、数据重定向技术对于流水线CPU进行冒险处理时，因为这些方法书本上并没有，老师提供的PPT上也只有简单的一些描述，这就要求我不断地在网上搜索相关的知识内容，和小组内的成员进行相关探讨。随后的动态分支预测的设计难度更高，老师并没有提供任何相关的内容，这又使得我不断地去网上搜寻资料文献，阅读全英文的学术论文。从这一切也可以看出团队在一个工作中的重要性，以及自我学习能力的必要性。

然而对于本次课程设计，我还有一些小小的建议和改进。本次课程设计按照分组的机制，但是因为所有人所需要完成的工作相同，分组也只是更便于组内的同学在设计过程中进行互相交流，分组的作用并不是非常明显。分组内的同学们进度完全不同，且有些同学因为对于课程内容的掌握不是很好，逐渐边缘化，分组并不能调动所有同学加入到课程设计任务中。本课程设计的难度很大，但是时间又有一点不够充分，可以适当简化一下任务（并不是降低最终目标），使得在有限的时间内，在完成全部课程设计目标的基础上，适当减少任务中其中重复的繁杂的工作，以提高整个课程设计的效率。

最后在这里也感谢三位老师对于我在本次课程设计中无数问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。我相信组成原理课程设计必将成为我整个大学生涯中一段无比难忘的回忆。

# 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. 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.
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:** |