***2021***



**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： | ACM1801 |
| 学 号： | U201814691 |
| 姓 名： | 张鼎元 |
| 电 话： | 15574851676 |
| 邮 件： | [1311435497@qq.com](mailto:1311435497@qq.com) |

目 录

[1 课程设计概述 3](#_Toc69567987)

[1.1 课设目的 3](#_Toc69567988)

[1.2 设计任务 3](#_Toc69567989)

[1.3 设计要求 3](#_Toc69567990)

[1.4 技术指标 4](#_Toc69567991)

[2 总体方案设计 6](#_Toc69567992)

[2.1 单周期CPU设计 6](#_Toc69567993)

[2.2 中断机制设计 10](#_Toc69567994)

[2.3 流水CPU设计 12](#_Toc69567995)

[2.4 气泡式流水线设计 13](#_Toc69567996)

[2.5 数据转发流水线设计 13](#_Toc69567997)

[2.6 动态分支预测机制 14](#_Toc69567998)

[3 详细设计与实现 16](#_Toc69567999)

[3.1 单周期CPU 实现 16](#_Toc69568000)

[3.2 中断机制实现 18](#_Toc69568001)

[3.3 流水CPU实现 20](#_Toc69568002)

[3.4 气泡式流水线实现 21](#_Toc69568003)

[3.5 数据转发流水线实现 23](#_Toc69568004)

[3.6 动态分支预测机制实现 24](#_Toc69568005)

[4 实验过程与调试 28](#_Toc69568006)

[4.1 测试用例和功能测试 28](#_Toc69568007)

[4.2 性能分析 29](#_Toc69568008)

[4.3 主要故障与调试 30](#_Toc69568009)

[4.4 实验进度 31](#_Toc69568010)

[5 团队任务 32](#_Toc69568011)

[5.1 任务主题与设计方案 32](#_Toc69568012)

[5.2 负责部分 32](#_Toc69568013)

[6 设计总结与心得 35](#_Toc69568014)

[6.1 课设总结 35](#_Toc69568015)

[6.2 课设心得 35](#_Toc69568016)

[参考文献 37](#_Toc69568017)

# 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 | XOR | 异或 |  |
| 29 | XORI | 立即数异或 |  |
| 30 | LHU | 加载半字 |  |
| 31 | BGTZ | 大于0跳转 |  |

# 总体方案设计

## 单周期CPU设计

单周期CPU本次我们采用的方案是硬布线控制的方案，且实现指令与数据分开存储的方式完成方案设计。在一个周期内，控制器根据读入的指令给出相应的控制信号，同时其余各功能部件根据控制信号完成相应的功能。在实现时，采用Logisim仿真实现。总体结构图如图 2.1所示。



图 . 总体结构图

### 主要功能部件

单周期硬布线CPU主要的部件有：程序计数器PC，指令存储器IM，运算器ALU，寄存器组RF，数据存储器DM以及硬布线控制器。各个部件的设计如下：

#### 程序计数器PC

程序计数器PC是一个32位的寄存器，用于存储将要执行的指令的地址。在使能信号，复位信号与时钟信号的控制下根据输入的地址改变寄存器中的内容，并将其内容输出作为指令存储器IM的输入。在这次实验中使用上升沿触发的寄存器即可。

#### 指令存储器IM

指令存储器用于存储MIPS指令的二进制指令。根据PC的输出值来取出对应地址的指令并输出作为IR。由于一条MIPS二进制指令为32位，因此IM设计为按4字节对齐的方式读取，因此会忽略PC的低两位来读取指令。在这次实验中使用已提供的Logisim库中的硬件RAM来完成。

#### 运算器ALU

运算器的引脚与功能描述如表 2.1所示。ALUOP与对应功能的关系如表 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 ALUOP与对应功能

|  |  |  |
| --- | --- | --- |
| 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 |

#### 寄存器组RF

寄存器组根据输入的寄存器编号输出对应寄存器的内容，寄存器组会受到时钟信号的控制，在上升沿触发，在本次实验中使用已经提供的库内的寄存器组即可。在流水线的某些电路中会修改为下降沿触发，这一点会在具体电路中说明。

#### 数据存储器DM

数据存储器存储了存储程序的所有数据，构成主存，其组成与指令存储器IM一致，在这次实验中使用已经提供的数据存储器。

#### 控制器

本次实验的控制器采用硬布线控制，通过解析得到的指令给出不同的控制信号，是一个组合逻辑电路。具体的设计见后文。

### 数据通路的设计

指令系统数据通路框架设计如表 2.3所示。

表 2.3指令系统数据通路框架

| 指令 | PC | RF | | | | ALU | | | DM | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1# | R2# | W# | Din | A | B | OP | Addr | Din |
| RR类型 | PC+4 | Rs | Rt | Rd | Alu | R1 | R2 | OP |  |  |
| RI类型 | PC+4 | Rs |  | Rt | Alu | R1 | Imm | OP |  |  |
| 分支指令 | (PC+4)[31:28]||Imm||00 |  |  |  |  |  |  |  |  |  |
| 跳转指令 | (PC+4)[31:28]||index||00 |  |  |  |  |  |  |  |  |  |
| Store类型 | PC+4 | Rs | Rt |  |  | R1 | Imm | OP | Alu | R2 |
| Load类型 | PC+4 | Rs | Rt | Rt | Dout | R1 | Imm | OP | Alu |  |
| Syscall | PC+4 | #2 | #4 |  |  | R1 | R2 |  |  |  |

### 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 2.4。

表 2.4主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| ALUOP | 0~12 | 用于选择ALU功能 |
| MemToReg | 0/1 | 控制寄存器组写数据是否来源于Dout |
| MemWrite | 0/1 | DM写使能 |
| AluSrc | 0/1 | 控制ALU的B输入端的数据来源 |
| RegWrite | 0/1 | 寄存器组写使能 |
| Syacall | 0/1 | 判断是否为Syscall指令 |
| SignedEXT | 0/1 | 立即数有符号拓展信号 |
| RegDst | 0/1 | 写入寄存器编号rt/rd选择 |
| Beq | 0/1 | Beq指令译码信号 |
| Bne | 0/1 | Bne指令译码信号 |
| JR | 0/1 | JR指令译码信号 |
| JMP | 0/1 | 无条件分支控制信号 |
| JAL | 0/1 | JAL指令译码信号 |
| BGTZ | 0/1 | BGTZ指令译码信号 |
| LHU | 0/1 | LHU指令译码信号 |
| R1\_USED | 0/1 | 用于流水线冲突检测 |
| R2\_USED | 0/1 | 用于流水线冲突检测 |

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。

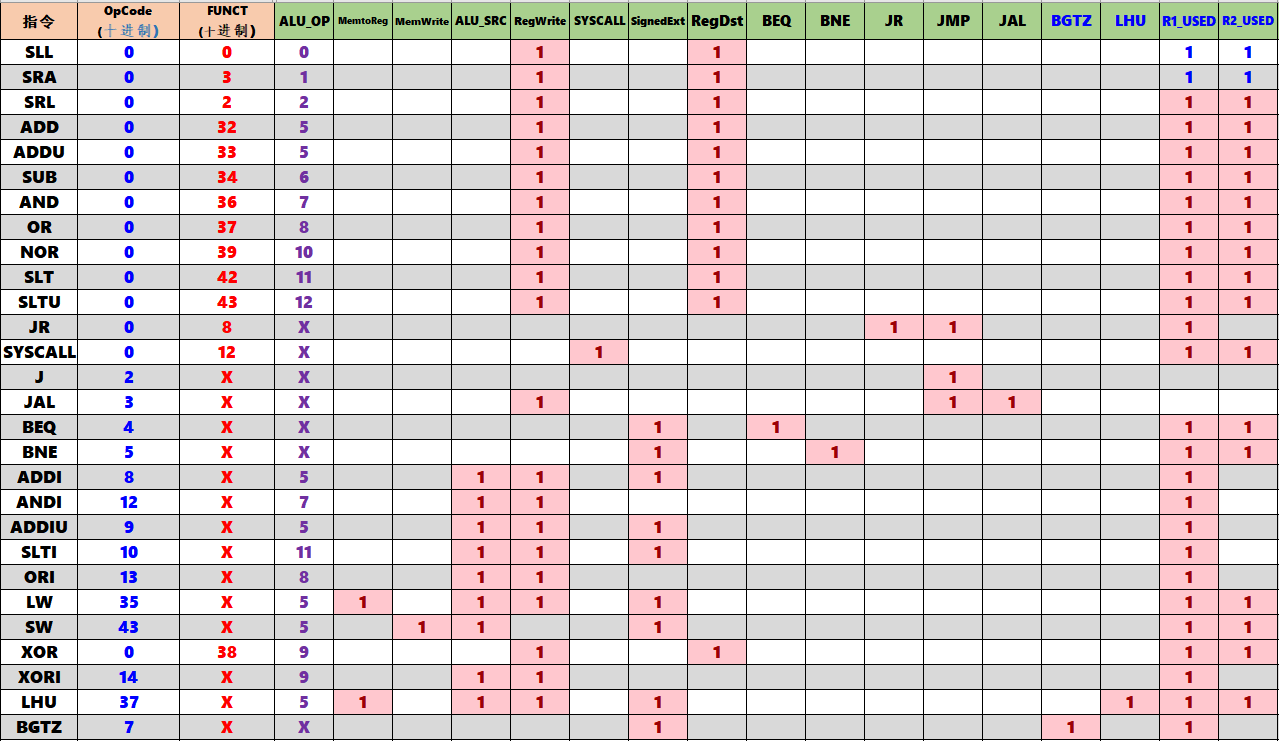


图 2.2 控制信号表框架

## 中断机制设计

### 总体设计

首先考虑单级中断的实现: 由于单级中断发生不能嵌套中断，因此就需要设置中断使能信号，同时由于在执行完中断服务程序之后还要恢复执行原来的程序，因此就需要一个EPC寄存器来保存发生中断时的PC寄存器的值，然后在中断完成后将EPC的值重新赋值给PC寄存器以恢复原来程序的运行。为了能够判断何时恢复PC寄存器的值，就需要增加对Eret指令的判断逻辑。同时还要根据中断号进入相应的中断服务程序，就需要增加中断入口地址的逻辑。

对于多级中断，由于可以嵌套中断，因此就需要设计硬件堆栈来存储每次中断的相关信息，而且考虑到中断优先级，还需要增加优先级判断的逻辑以判断新发生的中断是否能打断正在进行的中断。

对于流水中断只需要在单级中断与流水线电路的基础上结合二者并做出一些简单改进即可。

中断响应的流程图如图 2.3所示

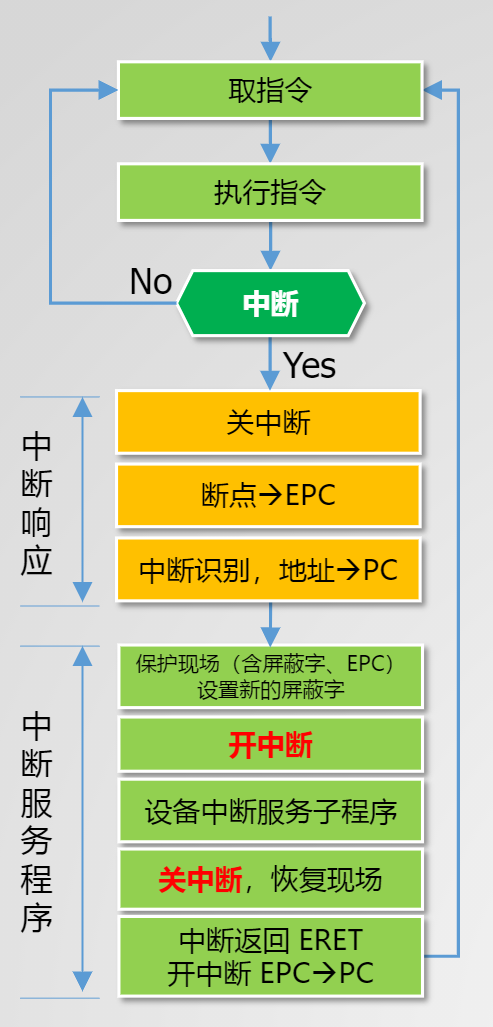


图 2.3 中断响应执行流程图

### 硬件设计

对于单级中断而言，使用中断案件参考电路来输入对应的中断信号，并使用优先编码器来处理同时发生多个中断时的情形，选择优先级最高的中断的入口地址送入到PC寄存器。EPC使用与PC一样的寄存器，在发生中断的时候将此时的PC值保存，在检测到Eret指令时将EPC的值送回到PC寄存器，并将中断信号复位。

对于多级中断，则需要增加硬件部分模拟堆栈，依次保存发生中断的中断号以及对应的EPC寄存器的值，以此完成多个中断的嵌套，相较于单级中断，多级中断在进入中断后不再关中断，因为高优先级的中断能够打断低优先级的中断。关于mfc0与mtc0指令的判断利用比较器来完成即可。

对于流水中断，需要考虑送入EPC寄存器的PC是从哪一阶段来，而且要考虑中断是否要插入气泡等等对于流水线的影响。

### 软件设计

设计中断服务程序时，首先需要保护现场，将寄存器依次压栈以便返回时能够恢复现场。在中断服务程序结束时，要使用Eret指令来表明此时应当返回到被打断的程序中去。在编写好程序后要利用模拟工具来获得每个中断服务程序的入口地址，以便硬件实现。

## 流水CPU设计

### 总体设计

本次课程设计实现的是5段流水，分别对应取指令阶段IF，指令译码阶段ID，指令执行阶段EX，访存阶段MEM以及写回阶段WB。需要利用流水接口部件来将各个阶段分离，完成相应信号的传递与锁存。条件分支与跳转指令在EX段执行，需要在EX段增加相应的逻辑来正确完成跳转功能。

### 流水接口部件设计

流水接口部件锁存了每个阶段所需要的信号，在使能信号，时钟信号等作用下每个周期都从输入读取数据并保存在寄存器中，输出则等于寄存器保存的内容。为了满足气泡流水线以及重定向流水线的实现，还需要增加清零端以将寄存器内容全部清零。

### 理想流水线设计

理想流水线不需要考虑分支与跳转指令，因此只需要在单周期CPU通路上修改即可，将单周期CPU各个部件按照其发生作用的阶段分开，再在相邻两个阶段直接加上对应的流水接口部件，连接相应的信号以及流水接口部件控制信号，总体上按照如图 2.4所示的数据通路实现即可。对于Syscall，则在WB阶段再做具体的处理来判断停机等。

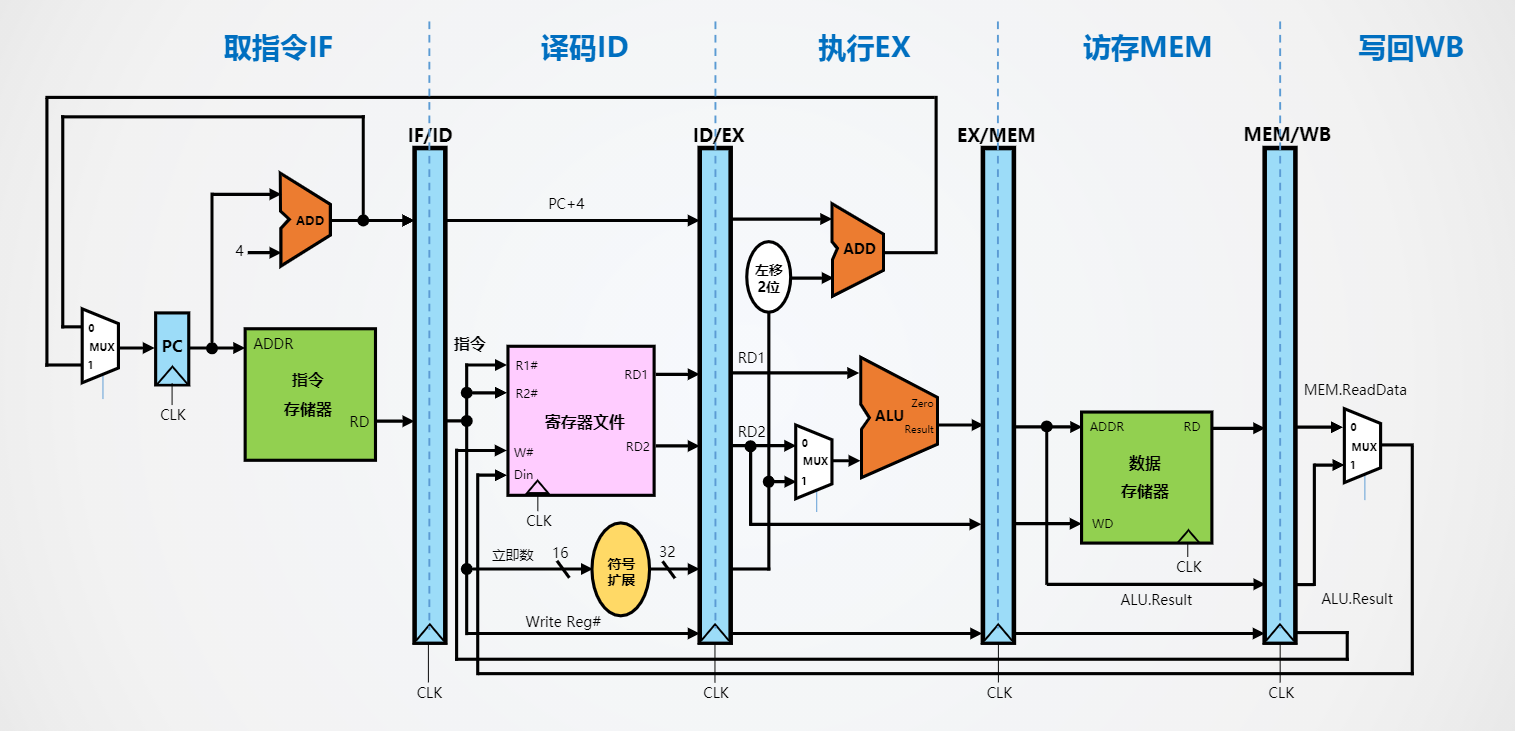


图 2.4 理想流水线数据通路图

## 气泡式流水线设计

气泡流水线则需要处理指令之间形成的冲突，并通过在流水线中插入气泡来解决冲突。对于分支的处理，流水线需要清空误取的指令并插入气泡，因此需要为流水接口部件增加气泡逻辑，在发生了跳转时给出相应的信号给IF/ID以及ID/EX接口部件来解决分支问题。对于数据相关问题的处理，则是分情况分析：对于ID段与WB段相关，只需要将寄存器文件修改为下降沿触发写入即可解决；对于ID段与EX、MEM段数据相关，则需要在ID段增加相应的数据相关检测逻辑，并在检测到数据相关时给出PC与ID/EX的阻塞信号并向ID/EX插入气泡。而数据相关检测逻辑则需要根据ID段的控制信号与EX,MEM段控制信号来判断是否发生了数据相关，具体而言检测ID段当前指令读寄存器是否与后续2条指令写寄存器相同(不考虑0号寄存器)。气泡流水线的数据通路基本与理想流水线数据通路一致。

## 数据转发流水线设计

重定向流水线的思路则是在真正需要数据时直接将后面正确的数据重定向到需要的地方，因此需要增加数据旁路。但是对于Load-Use相关，如果增加相应的数据旁路则会使得流水线性能大幅下降，因此对于Load-Use相关，依然使用插入气泡的方式来解决：给出PC的暂停信号并在IF/ID, ID/EX插入气泡。由于增加了数据旁路，因此就需要增加重定向控制信号来选择正确的数据传递给相应的接口。同时还要增加Load-Use检测逻辑来检测这种相关，重定向流水线的数据通路如图 2.5所示。

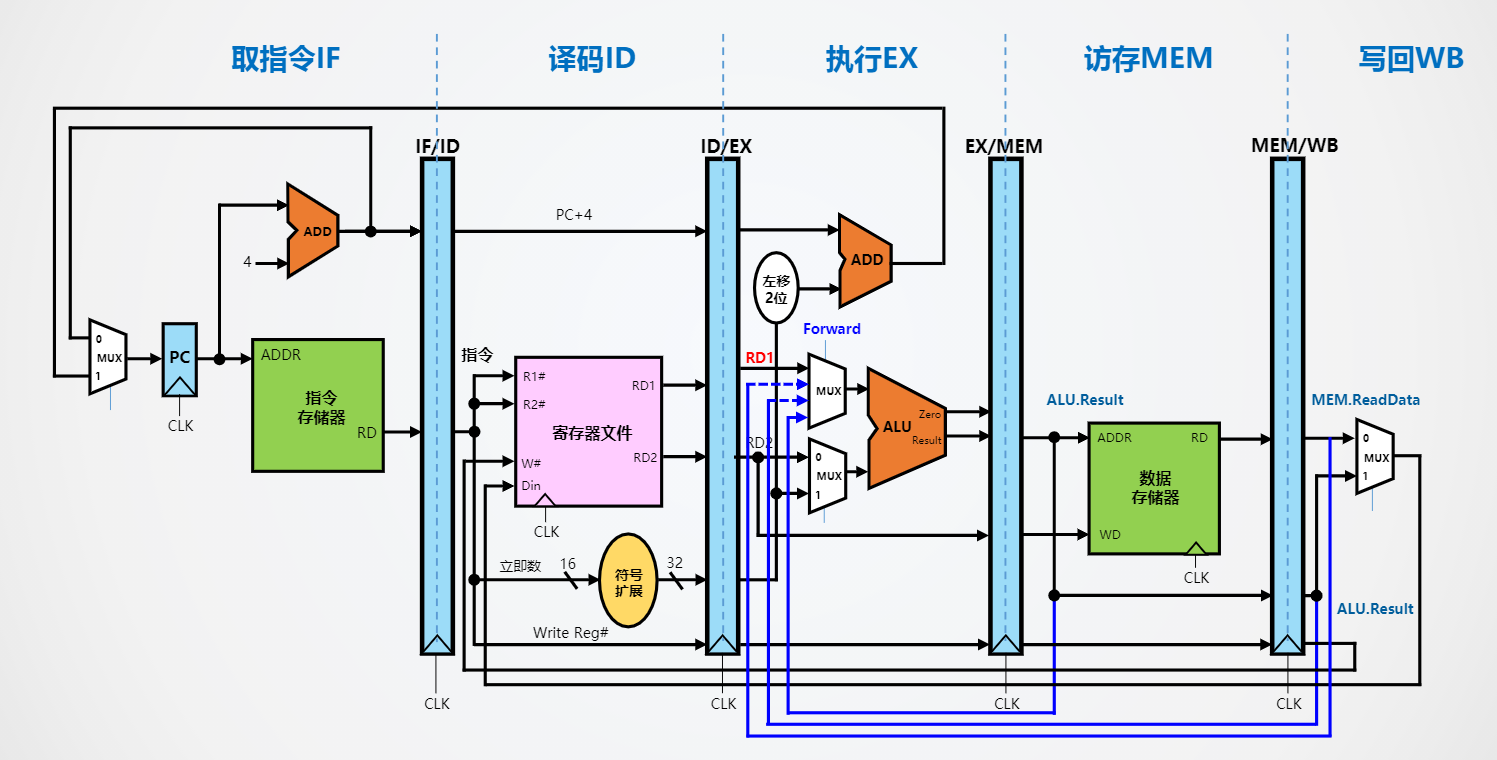


图 2.5 重定向流水线数据通路图

## 动态分支预测机制

由于重定向流水线在处理Load-Use相关时仍然插入了气泡，因此影响流水线性能的关键变为了流水线中的控制冲突，基于加快经常性事件的原理，考虑减少分支指令所引起的性能损失。为此引入动态分支预测，以达到预取正确指令，减少气泡的目的。为了达到这一目的，借鉴Cache实现BTB表来实现分支预测。

### BTB设计

分支预测缓冲器BTB用于存放分支指令的分支跳转历史统计信息，BTB表的每个表项主要包括valid位、分支指令地址、分支目标地址、分支预测历史位、置换标记。BTB表本质上是一个全相联的cache，因此在设计与实现时直接在cache基础上修改。BTB表以分支指令地址作为关键字进行全相联并发比较，如果命中则根据本次分支是否跳转调整相应的表项的分支预测历史位；否则将该分支指令的相关信息载入，并设置合适的分支预测历史位；最终输出预测结果。淘汰算法使用LRU。此外BTB同样需要时钟信号等信号的控制。

### 动态分支预测流水线设计

动态分支预测流水线在重定向流水线的基础上修改，主要的修改是添加BTB以及相应的数据通路。在IF段识别到分支指令时分别对三种情况进行处理：对于未命中的指令，流水线与原来行为一致；对于命中的指令，BTB预测错误，则执行Load-Use相关处理一样的操作；对于预测正确的指令则流水线继续执行而无需插入气泡，从而提升了性能。动态分支预测部分设计如图 2.6所示。

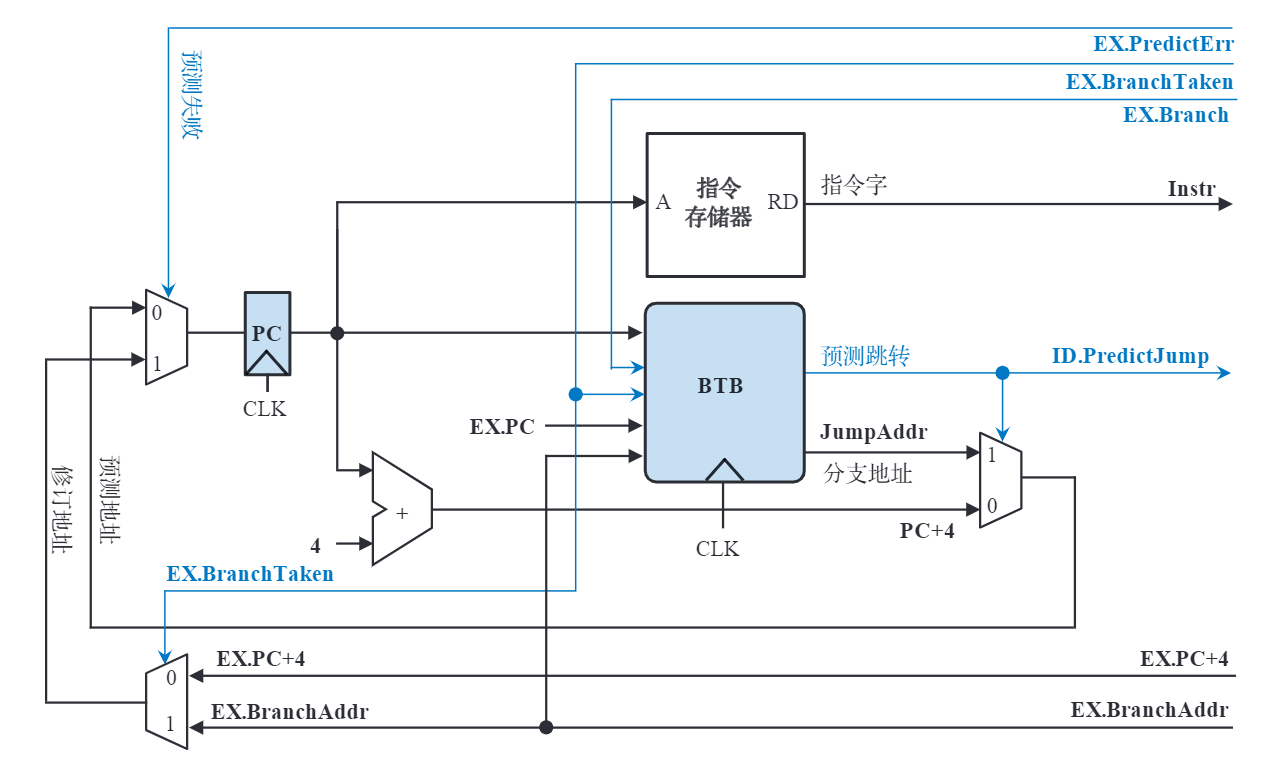


图 2.6 动态分支预测BTB部分原理图

# 详细设计与实现

## 单周期CPU 实现

### 主要功能部件实现

1. 程序计数器（PC）

使用一个32位寄存器实现程序计数器PC，触发方式为上升沿触发，输入为下一条将要执行的指令的地址，输出为当前执行指令的地址。halt为停机信号，当需要进行停机时，halt控制信号为1，多路选择器选择Go信号作为PC段的使能端，直到Go信号为1前PC都会忽略时钟输入，使整个电路停机。如图 3.1所示。

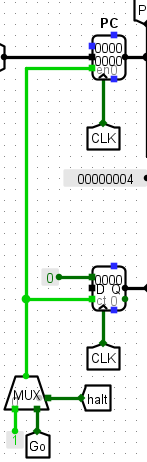


图 3.1程序计数器（PC）

1. 指令存储器（IM）

使用一个只读存储器ROM实现指令存储器（IM）。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。

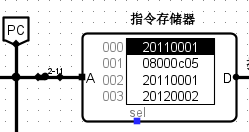


图 3.2指令存储器（IM）

1. 运算器ALU

本次实验中直接使用cs3410.jar中的ALU部件。

1. 寄存器组RF

本次实验中直接使用cs3410.jar中的MIPS RegFile部件。

1. 数据存储器DM

由于本质上数据存储器与指令存储器是类似的，因此使用RAM来实现DM。设置该只读存储器的地址位宽为10位，数据位宽为32位。因为计算得到的地址有32位，而RAM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位地址的2-11位作为指令存储器的输入地址。如图 3.3所示。

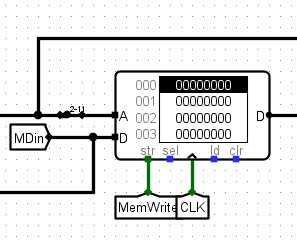


图 3.3 数据存储器(DM)

### 数据通路的实现

在完成主要功能部件的设计与实现后，将各个主要功能部件进行连接，对于所有的多输入部件使用多路选择器进行输入选择。最终便可以完成数据通路的搭建，数据通路如图 3.4所示。

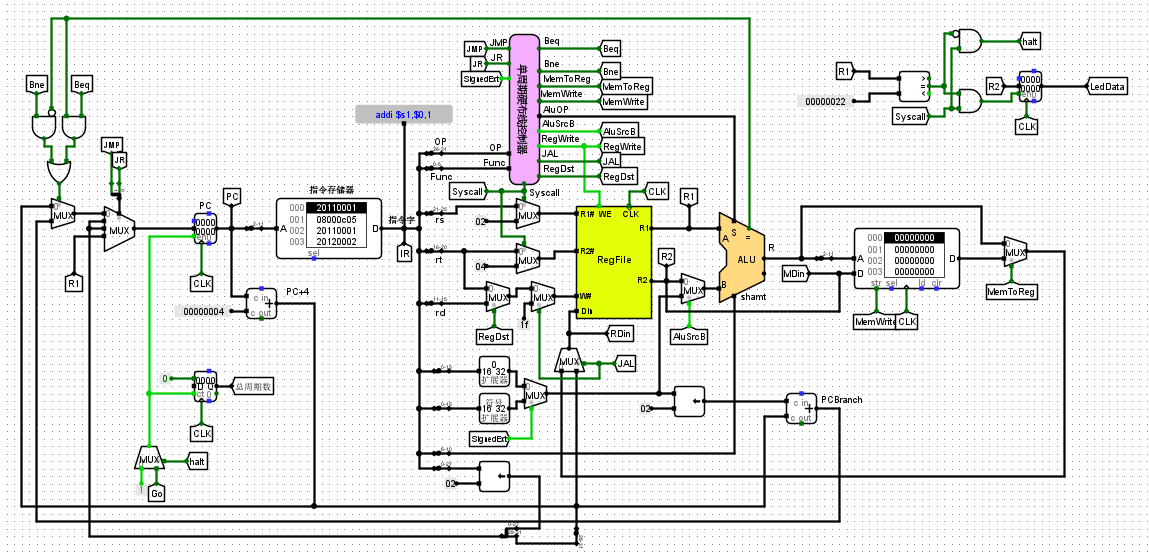


图 3.4 单周期CPU数据通路

### 控制器的实现

根据图 2.2所示填写给定的Excel表格并自动获得控制信号的表达式，利用Logisim的自动生成电路功能生成运算控制器子电路与控制信号生成子电路，然后连接相应部件与输入即可得到控制器，实现如图 3.5所示。

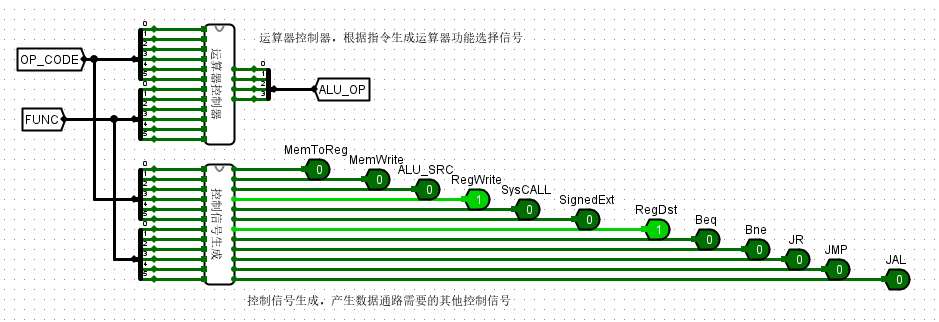


图 3.5 单周期CPU控制器

## 中断机制实现

### 单级中断

首先考虑中断使能IE，由于IE受到时钟信号的控制，且在发生中断以及Eret指令执行时发生变化，因此可以使用一个寄存器来保存；然后考虑发生中断时的断点保存EPC，由于其也受到时钟信号的控制，因此也使用一个寄存器来保存，寄存器输入为发生中断时下一条指令的地址值，且使能端要接入中断标志；然后是Eret之后根据中断号将对应的中断信号归位，使用一个解复用器即可；最后考虑中断信号产生，使用已提供的中断信号产生电路即可，若同时有多个中断产生，则利用优先编码器选择优先级最高的即可。中断部分的数据通路如图 3.6所示，主体电路的数据通路如图 3.7所示。

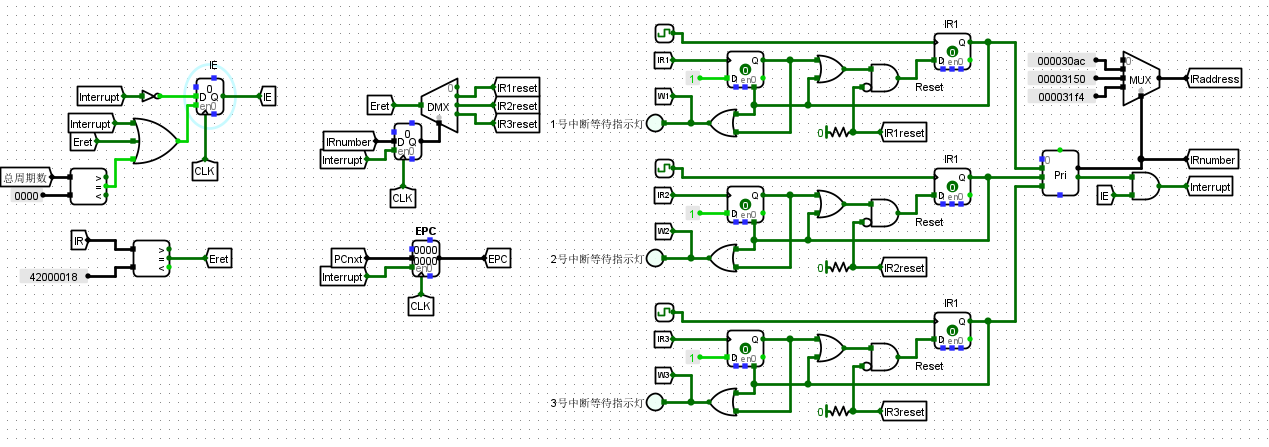


图 3.6 单级中断中断部分数据通路

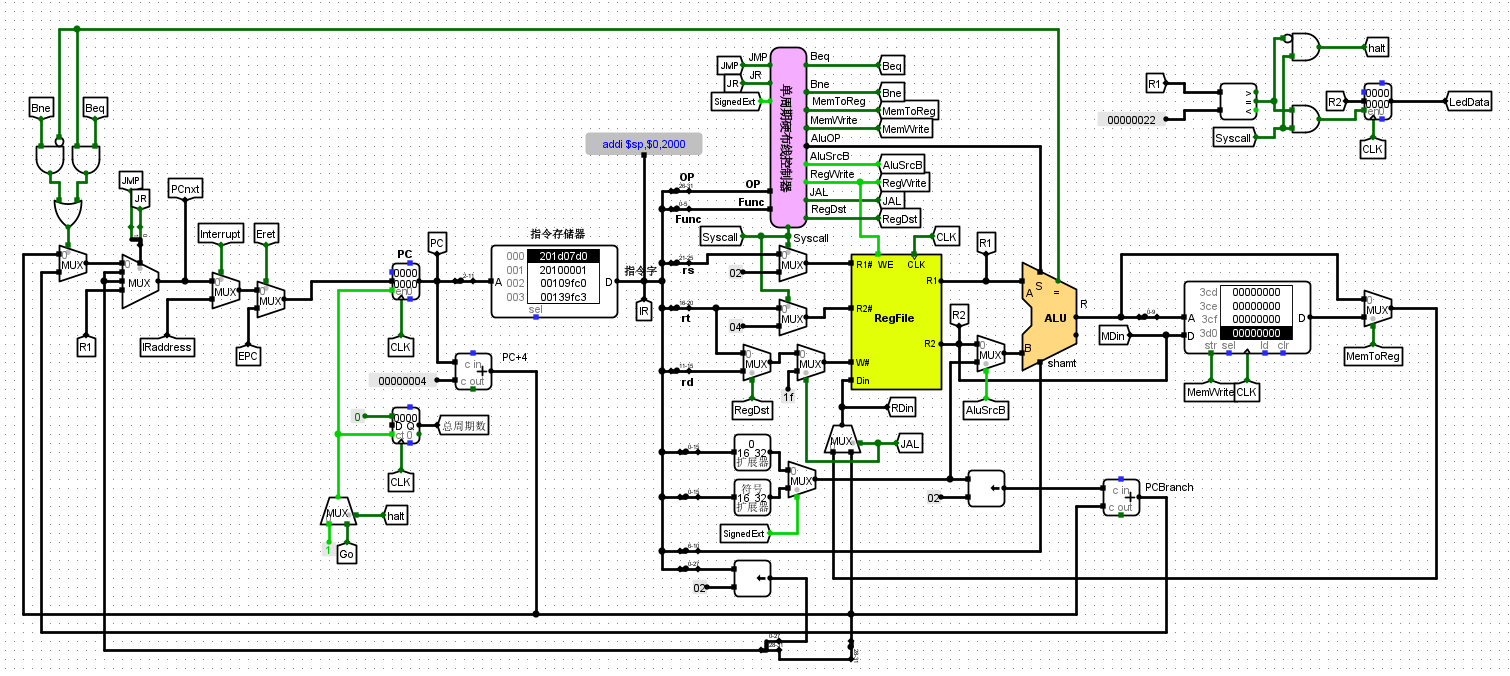


图 3.7 单级中断主体电路数据通路

### 多级中断

首先考虑中断使能IE，相较于单级中断，IE还会受到mtc0与mfc0的影响，因此可以在单级中断相关的逻辑上加上mtc0与mfc0的影响；然后考虑EPC，由于同一个中断不会同时发生两次，但不同的中断可能会同时处理，因此对于每一个中断，分别设置一个EPC保存对应中断的返回地址，在返回时根据当前中断号选择其中相应的EPC即可；中断信号归位部分则是根据当前中断号来将对应中断信号归位，使用解复用器即可；中断信号产生部分仍然与单级中断一致，唯一的区别是要满足优先级高的中断才能打断当前中断；对于mtc0与mfc0的判断则直接取IR的OP与Rs段利用比较器完成；硬件堆栈使用3个寄存器来完成，同时用硬件维护表示栈顶的StackTop信号，为此可以利用两位加法器，在来了新的中断时，将栈顶+1，在弹出中断时则将栈顶+3来利用加法器溢出达到栈顶-1的效果；硬件堆栈部分根据StackTop信号的解码结果来决定将中断号写入哪个寄存器之中。中断部分数据通路如图 3.8所示。主体电路部分数据通路与单机中断基本一致故不再赘述。

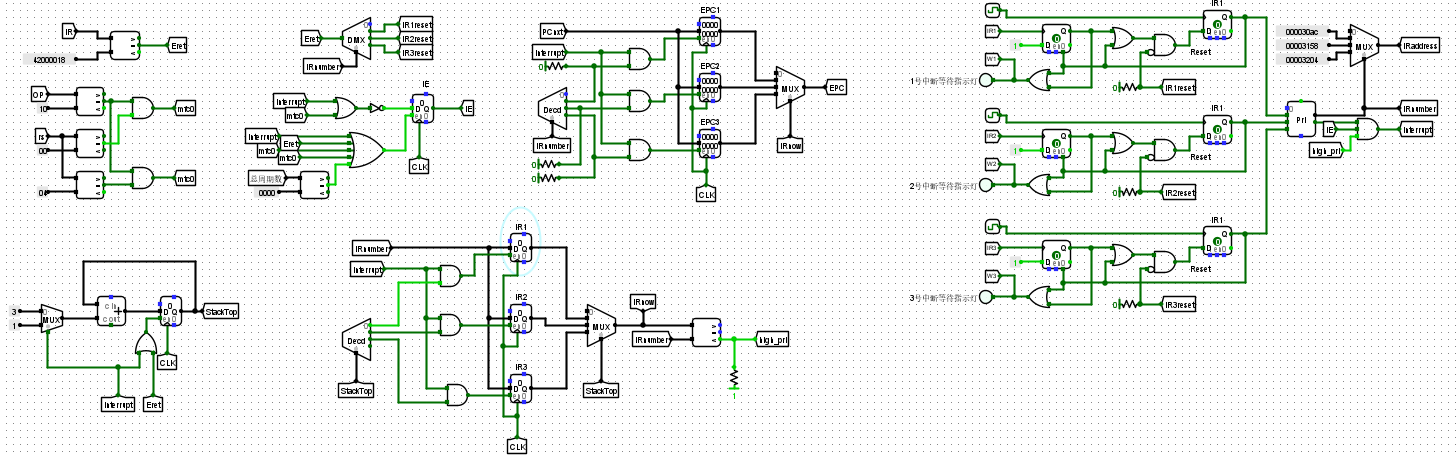


图 3.8 多级中断中断部分数据通路

## 流水CPU实现

### 流水接口部件实现

流水线接口部件受到时钟信号，使能信号，异步清空以及气泡信号的控制，为了达到所存的目的，需要使用寄存器来实现，气泡信号控制气泡的插入，可以使用多路选择器来确定使用输入还是插入气泡，由于各个流水接口部件内部结构基本一致，差别仅仅在于接口名字与个数不同，因此这里仅展示最简单的IF/ID接口，其他接口不再赘述。IF/ID接口实现如图 3.9所示。

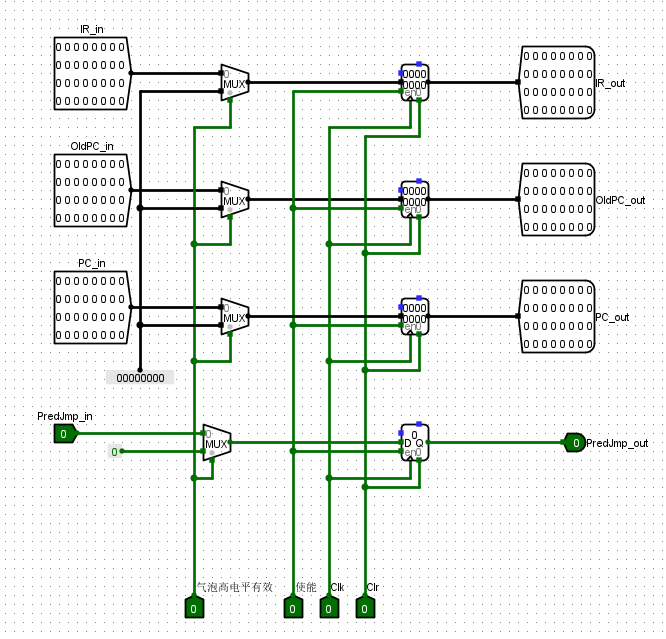


图 3.9 IF/ID流水接口部件

### 理想流水线实现

利用已经实现的流水接口部件按照图 2.4所示连接电路，即可得到最终的理想流水线，具体的数据通路实现如图 3.10所示。

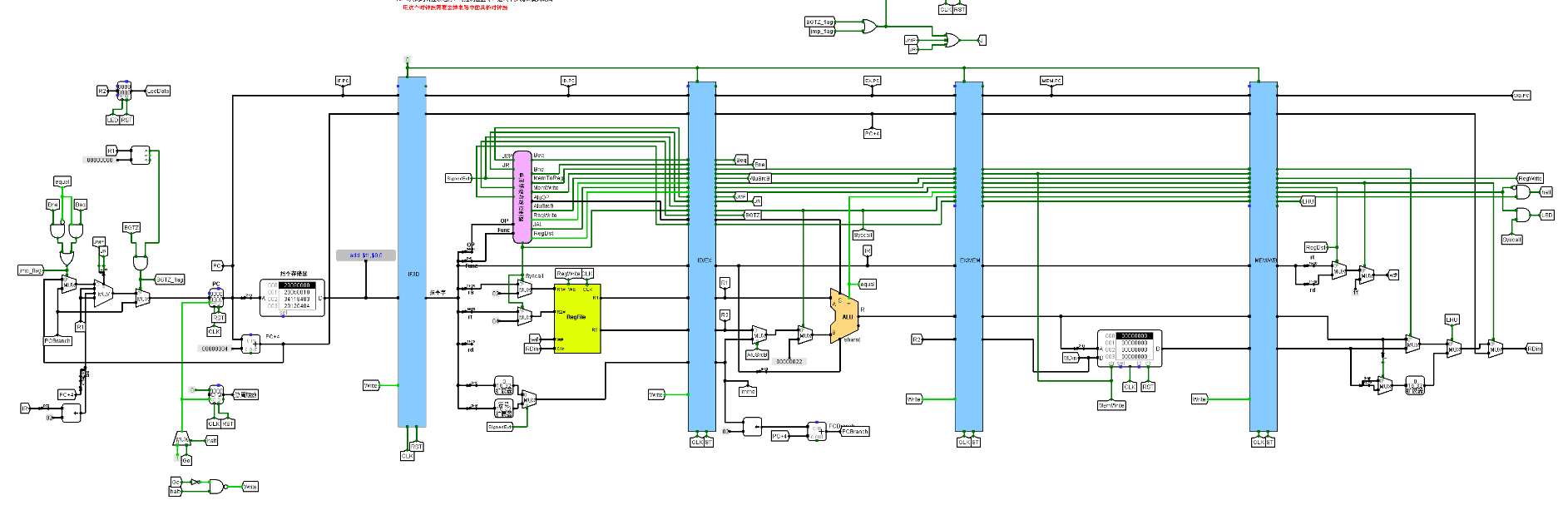


图 3.10 理想流水线数据通路

## 气泡式流水线实现

### 源寄存器使用情况子电路

由于在图 2.2中已经填写了各个指令的源寄存器使用情况，因此可以利用Logisim的自动生成电路功能生成该子电路，由于电路过于庞大，不在此展示。

### 数据冲突检测子电路

使用实现好的源寄存器使用情况子电路可以得到关于源寄存器的使用情况，然后利用比较器直接比较ID段的两个读取寄存器编号是否与EX段、MEM段写寄存器编号一致即可，要排除0号寄存器的影响，具体实现如图 3.11所示。

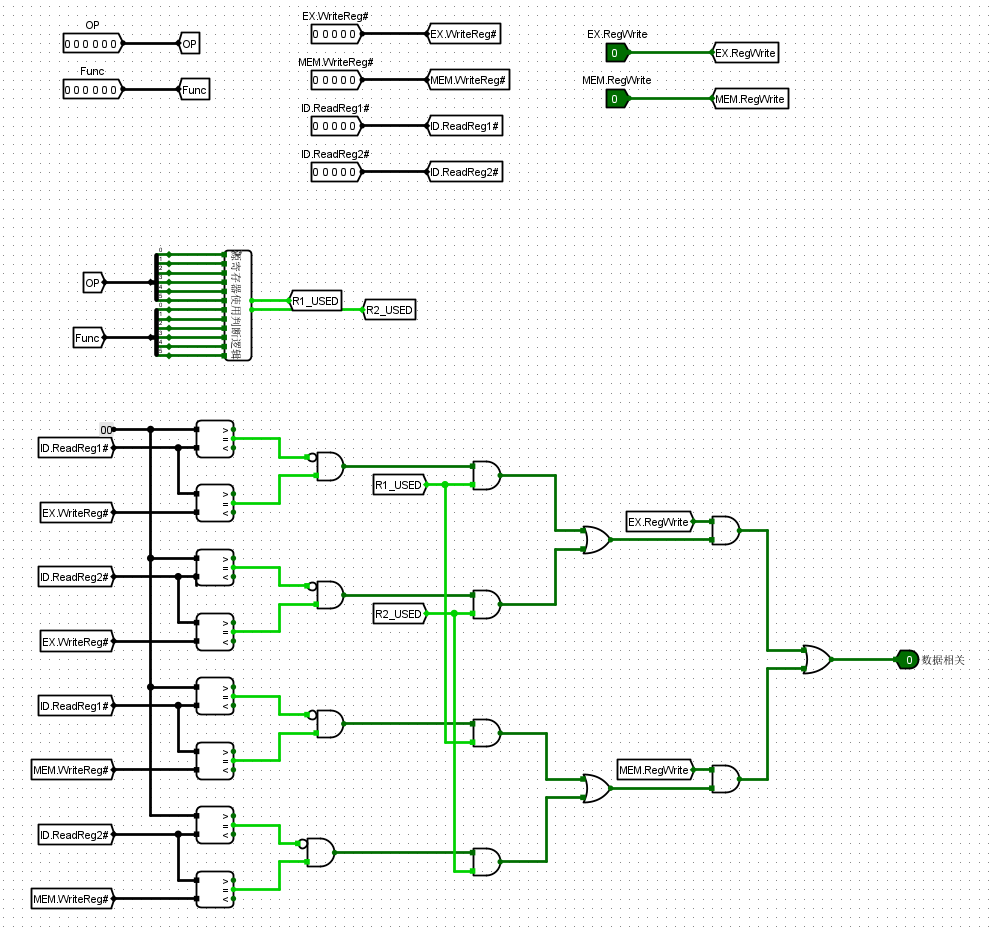


图 3.11 数据冲突子电路

### 气泡流水线

利用已经实现好的数据冲突检测电路判断是否发生了数据冲突，在发生了数据相关时要插入气泡并发送暂停信号，同时在发生跳转时要插入气泡，主要修改部分的数据通路如图 3.12所示。

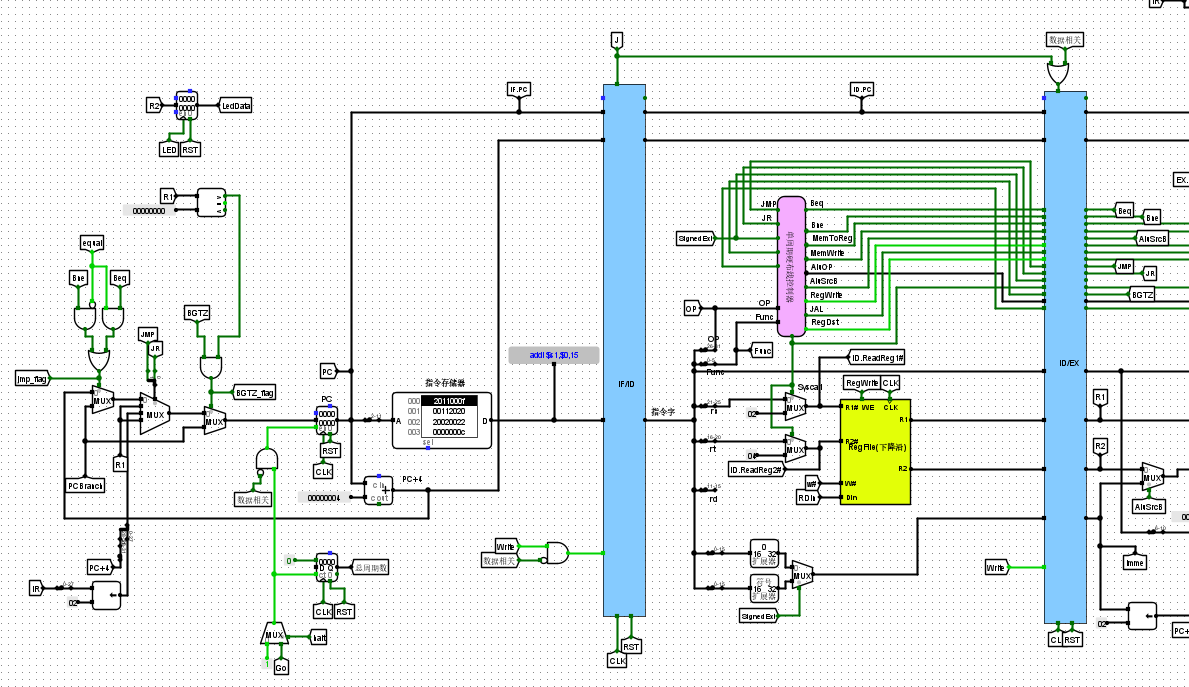


图 3.12 气泡流水线主要修改部分数据通路

## 数据转发流水线实现

重定向流水线需要在气泡流水线数据通路的基础上增加数据旁路以便数据能够重定向到合适的位置，但这种机制不能解决Load-Use冲突，因此对于load型指令需要增加Load-Use冲突检测电路。同时由于数据旁路的增加就使得重定向的位置需要多路选择信号来完成数据的选择。

### 重定向控制逻辑实现

重定向控制逻辑通过读取相应的信号作为输入来分析得到Load-Use冲突信号以及重定向多路选择信号。利用前面实现的寄存器使用情况电路可以得到寄存器的使用情况，然后再根据EX与MEM段的写寄存器信号以及ID段的R2#信号并结合EX.OP信号来判断是否有Load-Use冲突产生，并同时产生用于重定向的多路选择信号。具体实现如图 3.13所示。

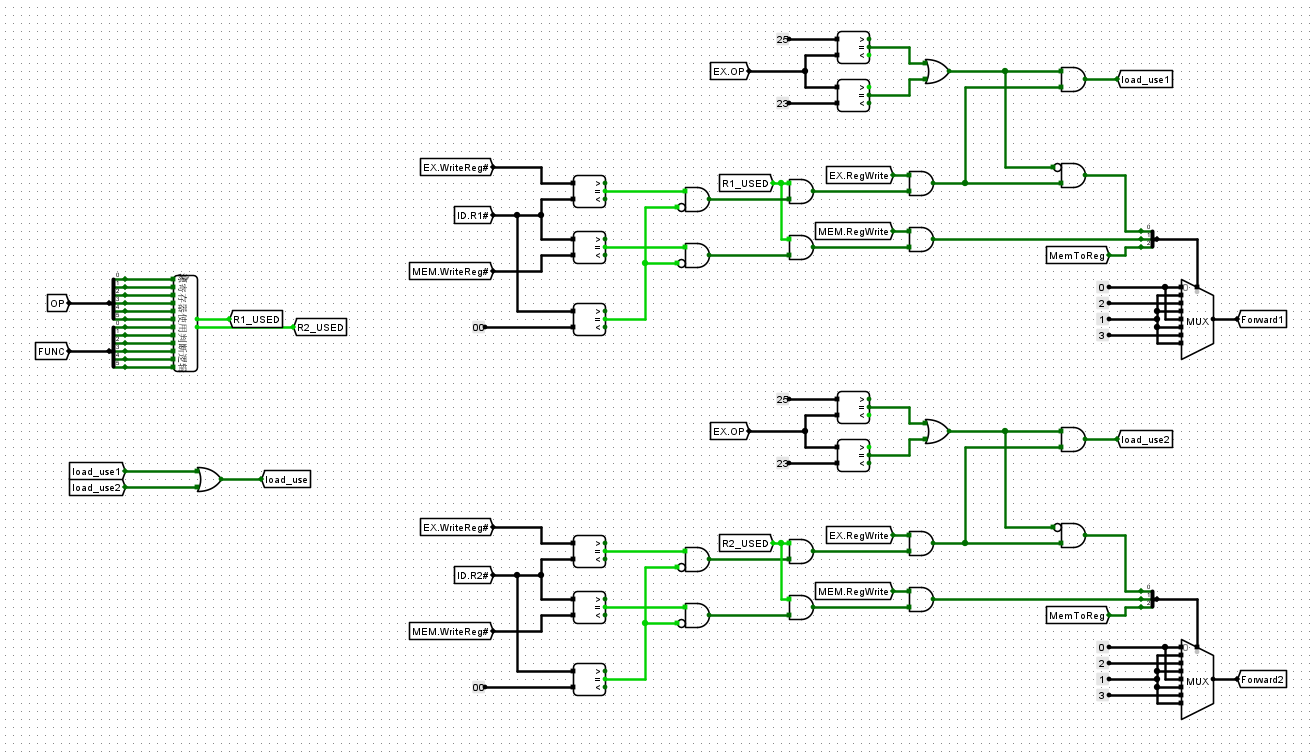


图 3.13 重定向控制逻辑电路

### 数据转发(重定向)流水线

数据通路基本与气泡流水线数据通路一致，主要不同之处在于用重定向控制逻辑替换了气泡流水线中的数据相关检测，并使用重定向多路选择信号选择输入ALU两端的数据，主要修改部分如图 3.14所示。其中NewR1与NewR2分别输入到ALU的两个数据端。

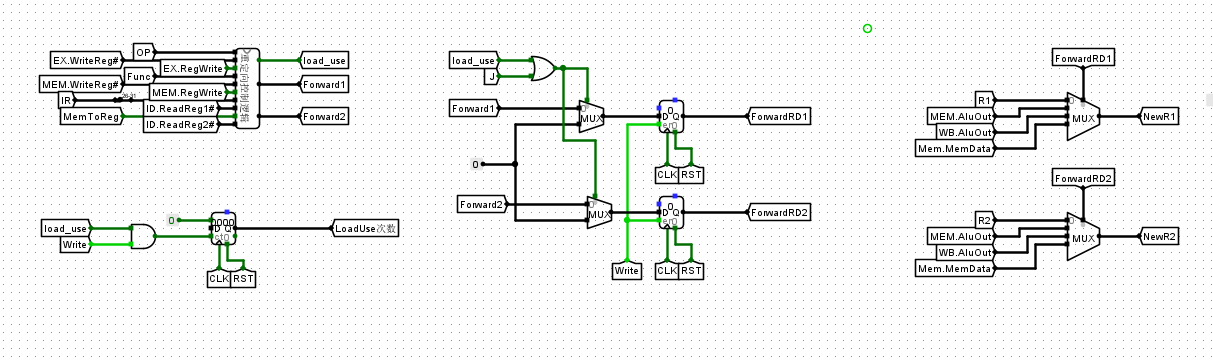


图 3.14 重定向流水线主要修改部分

## 动态分支预测机制实现

### BTB电路实现

BTB类似于全相联cache，包含有效位valid, 标记位tag, 置换标记c，分支目标地址与分支预测历史位。其淘汰算法LRU的实现电路以及命中判断电路与cache一致，不再赘述。其存储信息部分电路如图 3.15所示，这部分电路存储着上面描述的大部分信息。

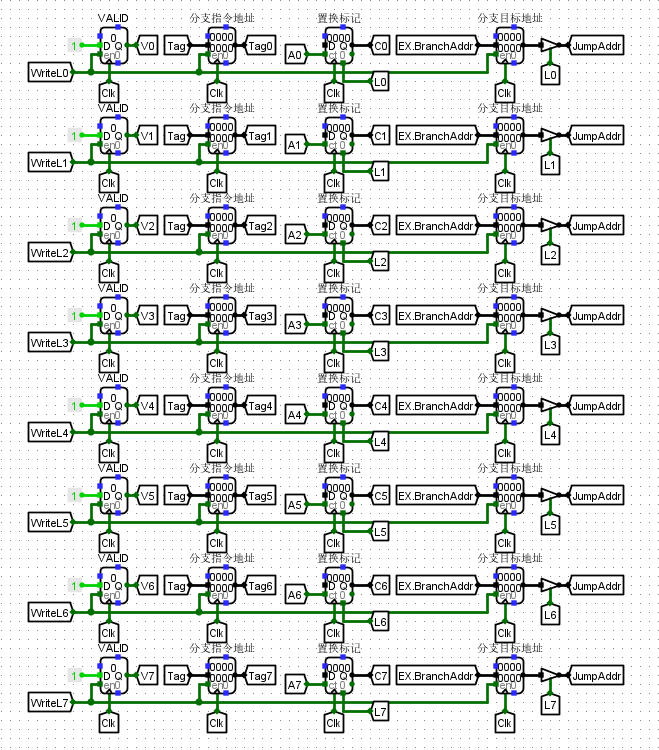


图 3.15 BTB存储信息部分电路

分支预测历史位部分电路如图 3.16所示。分支预测历史位使用双位预测，其预测状态转换图如图 3.17所示。同样利用加法器溢出来用+3代替-1的操作。

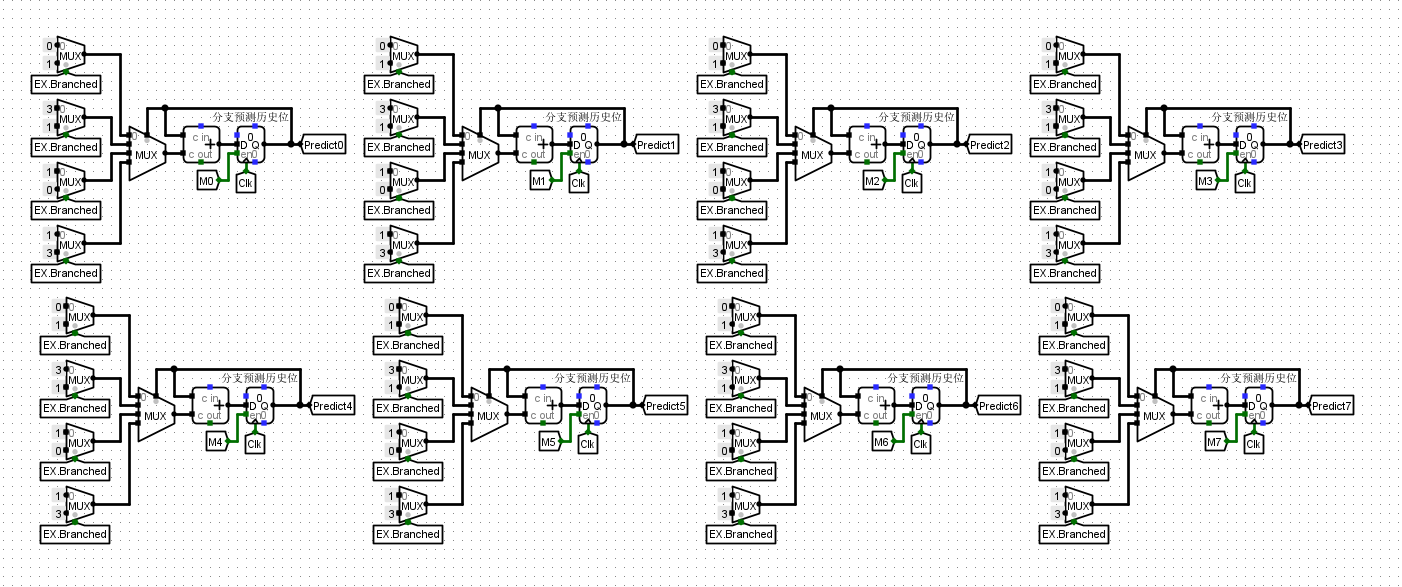


图 3.16 分支预测历史位电路

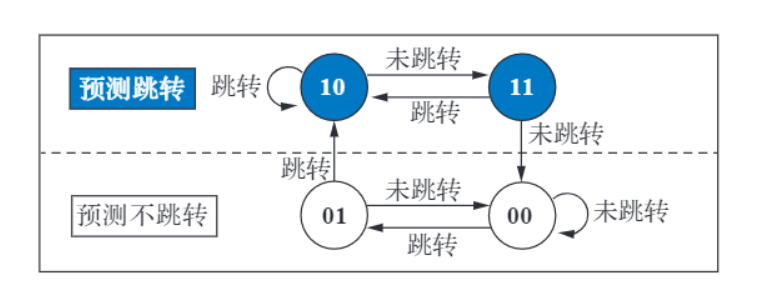


图 3.17 双位预测状态转换图

### 动态分支预测

动态分支预测在重定向流水线的基础上实现，为此只需要在重定向流水线电路的基础上加上上述实现的BTB结构并增加相应的逻辑即可。依照图 2.6所示连接BTB与数据通路，可以得到电路如图 3.18所示。

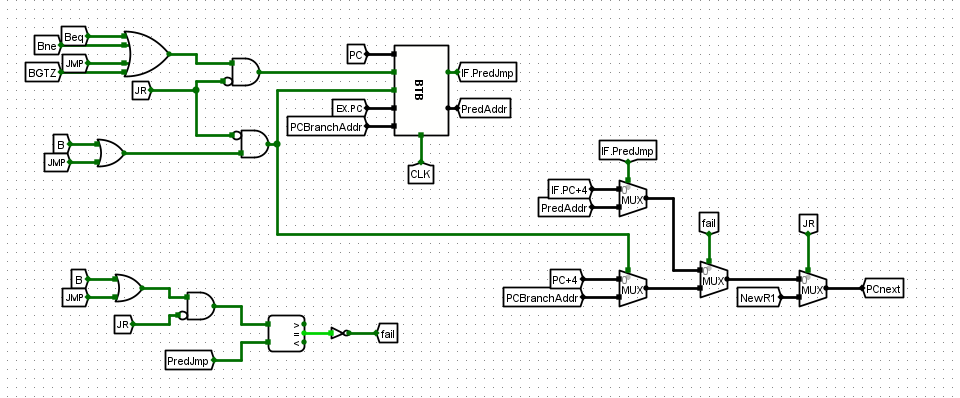


图 3.18 动态分支预测BTB相关电路

当实际跳转情况与预测跳转信号不符合时说明预测失败，此时需要在IF/ID与ID/EX中插入气泡，为此只需要在相应部件的气泡控制端用上图的fail信号替换原有的跳转信号即可，同时还需要在流水功能部件上加上PredJmp信号的结构，主要修改部分如图 3.19。

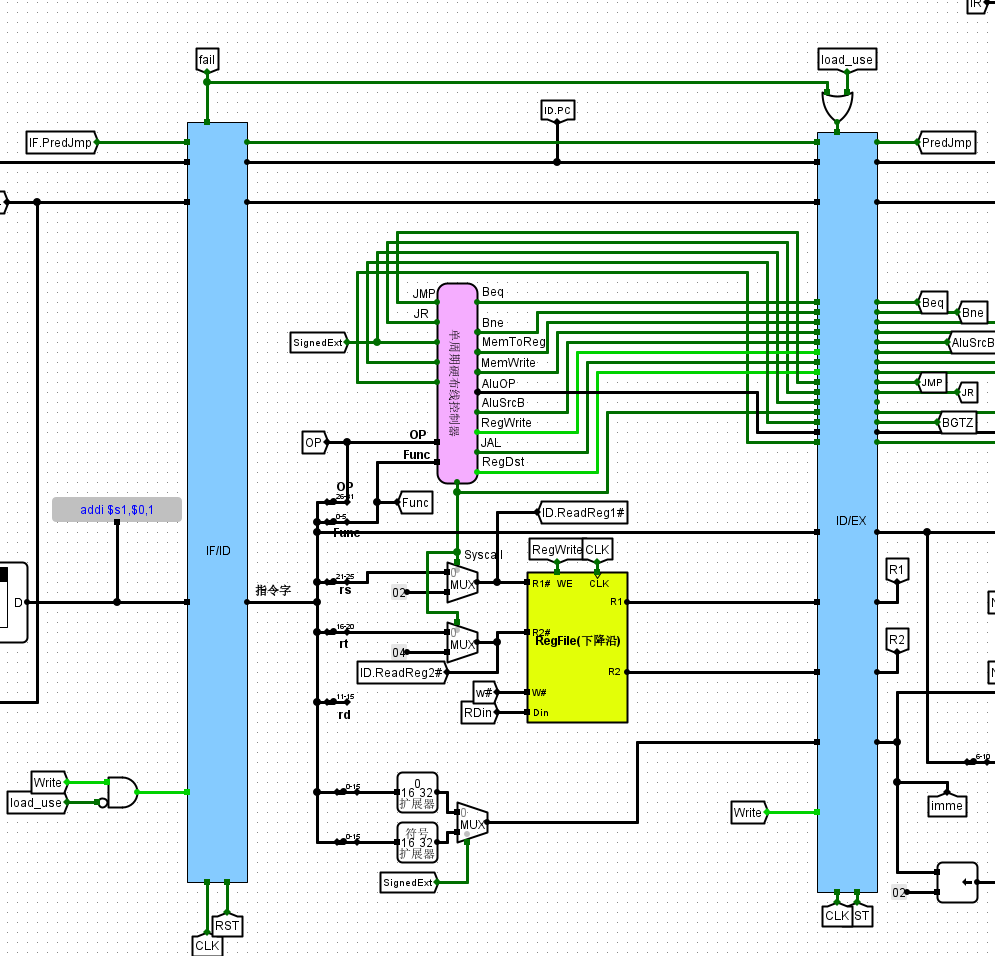


图 3.19 分支预测主要数据通路修改部分电路

# 实验过程与调试

## 测试用例和功能测试

对于单周期CPU、单级中断、多级中断以及流水线电路，使用educoder自动评测电路功能正确性；对于流水中断、动态分支预测以及差异化指令，加载给定的测试样例并根据LED显示的输出进行功能测试。

### 单周期CPU、单级中断、多级中断、流水线

使用educoder自动测评电路，全部结果如图 4.1所示。说明电路功能正确。



图 4.1 educoder自动测评结果

### 动态分支预测、流水中断、差异化指令

对于动态分支预测，加载benchmark.hex测试样例，运行电路，最终的结果如图 4.2所示，可以看到LED显示的结果与没有加入动态分支预测时一致，符合预期。而且最终的时钟周期数是1781，小于1800，说明动态分支预测起到了优化作用，且优化力度符合预期，说明电路功能正常。

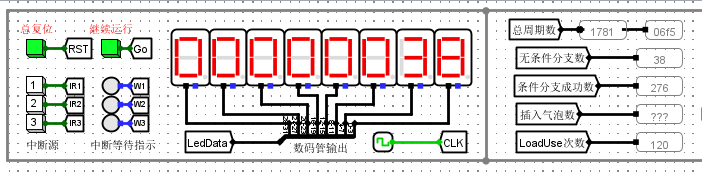


图 4.2 动态分支预测电路最终结果图

对于流水中断，加载单级中断测试程序.hex，在第140周期时分别按下1，2，3号中断源按钮，以1号中断为例，进入中断前与从中断返回后LED的显示结果如 图 4.3所示。说明电路功能正确。

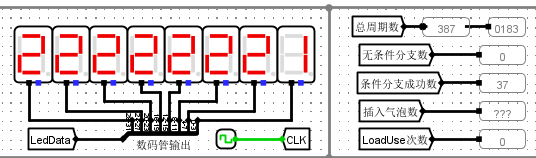
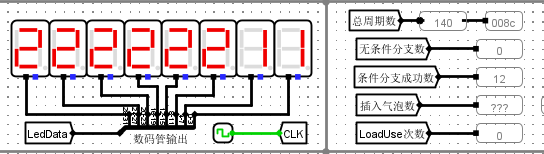


图 4.3 流水中断测试结果

对于差异化指令，运行benchmark\_ccmb.hex，当完成前面的指令停机后，每次按下go按钮便继续测试每一条差异化指令，以XOR指令为例，测试结果如图 4.4所示，LED在交叉显示00007777，ffff8888后最终显示00007777，符合预期，其他三条指令同样符合测试预期，说明电路功能正确。

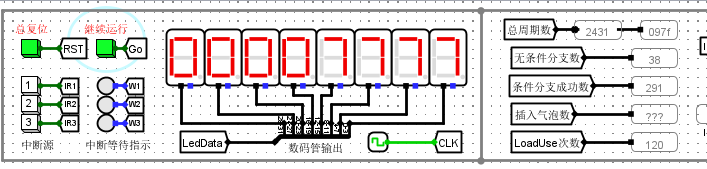


图 4.4 XOR指令测试最终结果图

## 性能分析

这一节比较MIPS单周期CPU、气泡流水线、重定向流水线以及加入动态分支预测的重定向流水线的性能。用benchmark.hex分别在这四个电路上运行，最终运行的时钟周期数分别为1545，3623，2297，1781。流水线技术在理想状态下可以每个时钟周期均完成一条指令的执行，然而由于各种冲突的存在，因此流水线不可能以理想状态运行，气泡流水线在遇到冲突时需要插入气泡，因此时钟周期数较大；重定向流水线由于使用了重定向，使得只有在遇到load型指令时才插入气泡，因此插入的气泡数大幅减少，因此时钟周期数相较于气泡流水线大幅减少；动态分支预测由于加入了对分支的预测因而使得分支导致的冲突减少，因此时钟周期数进一步减少。因此这些电路中动态分支预测电路的性能是最高的。

## 主要故障与调试

### 指令寄存器取指错误故障

MIPS单周期电路： 取值错误问题。

**故障现象：**对应PC值取出的指令不正确

**原因分析：**PC寄存器传给指令寄存器的地址错误，误将PC寄存器的第0~9位传给了指令寄存器

**解决方案：**将PC寄存器的第2~11位传给指令寄存器

### 中断优先级判断故障

多级中断(EPC硬件堆栈)：中断优先级错误问题

**故障现象：**在3号中断与1号中断同时发生时，进入了1号中断程序而不是优先级更高的3号中断程序

**原因分析：**在进行中断优先级比较的时候，使用的比较器是按照2的补码来进行的，因此3在比较时实际上按-1来解释，因此会比1小

**解决方案：**将比较器修改为按照无符号进行比较

### 停机故障

理想流水线：提前停机终止。

**故障现象：**第19个节拍出错，提前终止。

**原因分析：**有关停机部分的判断逻辑没有移动到流水线的最后一段，因此导致了提前停机

**解决方案：**将停机部分的判断逻辑移至流水线的最后一段。

## 实验进度

表 . 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，实现了单周期CPU并且通过了测试。 |
| 第三天 | 完成Logism单周期CPU的故障报告，开始设计单级中断与多级中断电路。 |
| 第四天 | 完成了单级中断电路的实现，并通过了测试。完成了部分多级中断电路的实现 |
| 第五天 | 完成了多级中断电路的实现并通过了测试。完成了理想流水线的实现并通过测试。 |
| 第六天 | 学习气泡流水线与重定向流水线的知识，开始设计气泡流水线与重定向流水线 |
| 第七天 | 完成气泡流水线与重定向流水线的实现并通过测试 |
| 第八天 | 复习cache相关知识，学习动态分支预测相关知识，开始设计BTB表结构 |
| 第九天 | 完成BTB表的实现，并改造重定向流水线加入动态分支预测并通过测试 |
| 第十天 | 设计并实现了流水线中断，通过测试。 |

# 团队任务

## 任务主题与设计方案

我们团队任务选题是实现一个五子棋游戏，需要在带有中断机制的单周期CPU电路上实现。其中输入通过按键实现，触发中断以实现棋盘的显示以及胜负的判断。

整个电路采用如图 5.1的方式进行显示，每个显示屏显示’+’ 或者’×’来表示两种棋子，上方用绘制的字母表示当前电路的运行状态，其中”DRAW”表示等待玩家下子；”WAIT”表示电路正在进行计算，请玩家等待计算的结果；”WIN+”表示’+’一方获胜，”WIN×”表示’×’一方获胜。右侧表示当前是哪一方的回合。玩家下子通过点按对应位置屏幕旁边的按钮来完成，点按按钮后触发中断，然后由程序绘制棋子并判断胜负情况。左侧是重新开始游戏的按钮。

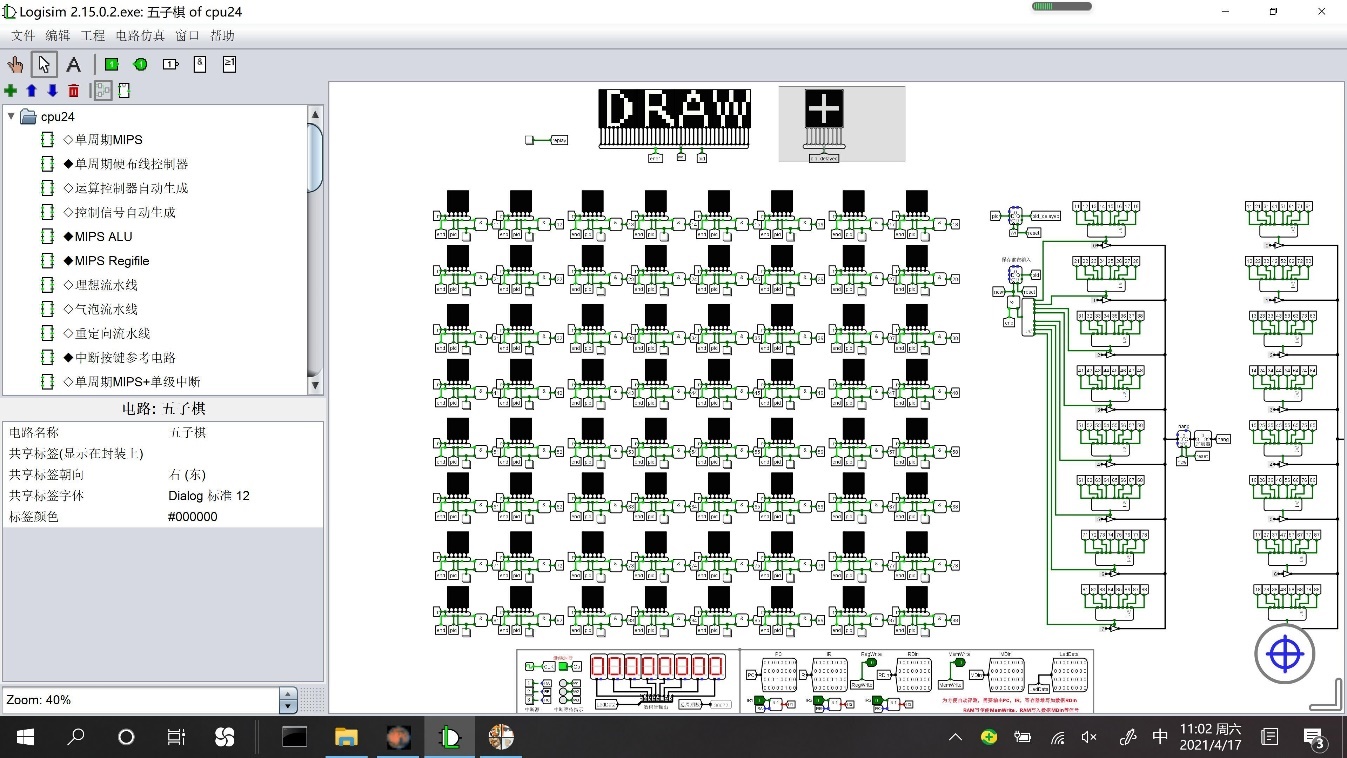


图 5.1 五子棋显示电路图

## 负责部分

我在团队任务中负责完成判断胜负C语言程序的编写以及汇编代码并修改代码等代码移植工作，同时还负责完成判断胜负程序的优化工作。

由于我认为项目的重点不在于程序的编写，因此在网络上找到了一个简易版本的判断胜负的程序作为最初版本的C语言代码，最初版本的C语言代码如图 5.2所示，使用准备好的mips gcc转变为汇编代码然后使用Mars汇编后，放到电路的指令寄存器中，发现每下一个子，都需要至少30s中才能完成判定，这就使得整个游戏根本没有可玩性。

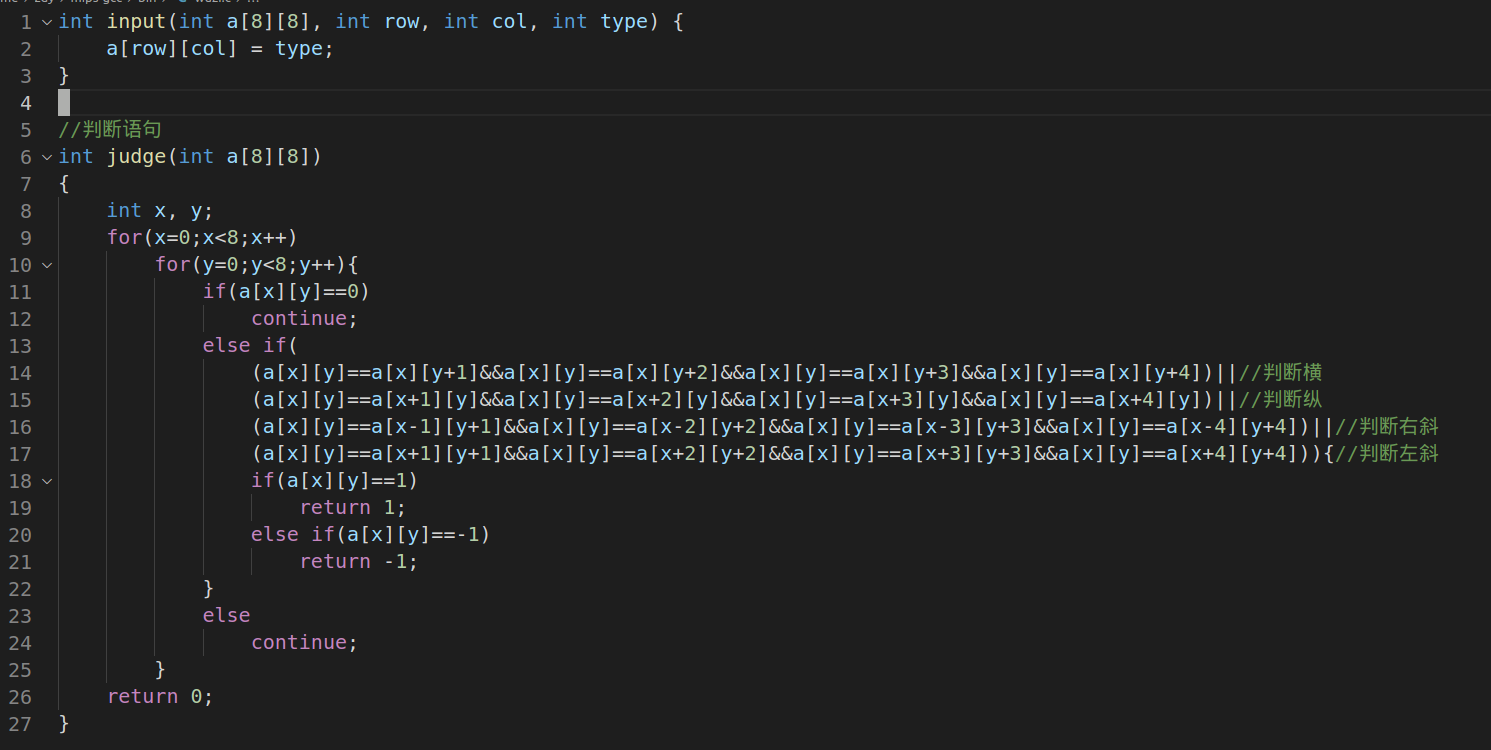


图 5.2 最初版本胜负判断C程序

为此需要显著优化判断胜负的C程序，得到了最终版本的判断胜负的C程序如图 5.3所示。但是仍然需要几秒的时间才能完成判定。

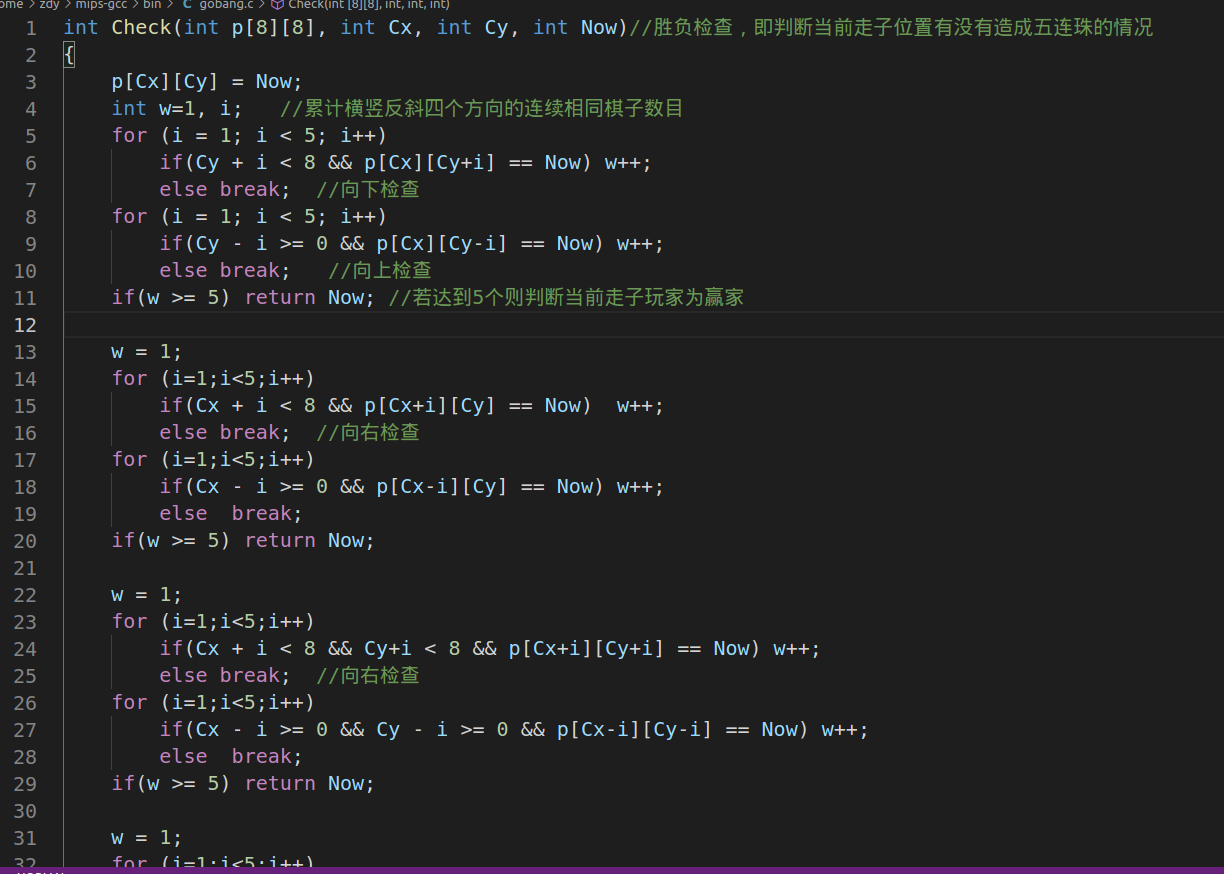


图 5.3 最终版本胜负判断C程序

考虑到C语言层面已经难以优化，为此我们选择在最后的C程序的基础上在编译时加上O2优化以期望得到一个更优的汇编程序。经过这样的优化后，每次下子都能在1s内完成判定，这样游戏的可玩性大大提高。但是我们发现经过O2优化得到的代码在电路中出现了错误，判断胜负出错。

于是我们开始一条条检查得到的汇编代码，并最终发现mips gcc的O2优化利用了mips的延迟槽技术，从而使得某些跳转指令与其前后指令的位置不符合逻辑，因为我们的电路没有使用延迟槽技术，从而导致了错误的产生。为此我们将所有不符合逻辑的指令调整了顺序，最终得到了足够高效且正确的汇编代码。在这份代码的加持下我们的五子棋电路完成最终的实现，而且具有了可玩性。

# 设计总结与心得

## 课设总结

在这次课设中我完成了如下几点工作：

1. 设计了MIPS单周期24条指令CPU电路、单级中断电路、多级中断电路、理想流水线电路、气泡流水线电路、重定向流水线电路、流水中断电路以及动态分支预测电路。
2. 实现了MIPS单周期24条指令CPU电路控制器以及数据通路逻辑电路
3. 实现了单级中断、多级中断的逻辑电路
4. 实现了单周期CPU电路向理想流水线的改造
5. 实现了数据相关检测逻辑电路以及Load-Use检测逻辑电路，实现了气泡流水线以及重定向流水线逻辑电路。
6. 实现了BTB表逻辑电路，实现了重定向流水线上的动态分支预测。
7. 实现了重定向流水线上的流水中断逻辑电路。
8. 设计了在MIPS单周期CPU电路上的五子棋电路，实现了五子棋电路基础上的胜负判断程序的实现，实现了代码的移植与优化

## 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。从寒假开始着手设计到最后完成整个课设的实现，战线比较长，但是完成课设的成就感是不言而喻的。

课程设计刚刚开始的时候，设计单周期CPU以及单级中断，该任务和上学期实验课的任务类似，因此完成比较轻松，出现了一点小问题但是也很快解决了。对于多级中断则设计了如何用Logisim硬件实现堆栈，一番思考之后成功完成了硬件堆栈的实现，多级中断迎刃而解。

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，由于之前没有学习过这部分知识，因此去网络上寻找了许多资料学习相关的原理，而后才设计出了相关的冲突检测以及处理电路并最终完成了流水线的实现。

最后是动态分支预测，同样在网络上寻找资料学习原理，结合之前cache的相关知识实现了BTB，最终完成了动态分支预测电路的设计。从这一切也可以看出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。本课程设计的难度很大，对于团队任务而言，相关的指导以及资料较少，短时间内比较难以着手解决团队任务，希望可以针对一些团队任务常用的部分给出一些指导性的意见以及相关的资料等等，能够让我们更好地去构思如何设计并实现团队任务。

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. 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.

|  |
| --- |
| 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: 张鼎元** |