***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： | CS1601 |
| 学 号： | U201614532 |
| 姓 名： | 吕鹏泽 |
| 电 话： | 18703816020 |
| 邮 件： | [103349015@qq.com](mailto:103349015@qq.com) |

目 录

[1 课程设计概述 3](#_Toc4954006)

[1.1 课设目的 3](#_Toc4954007)

[1.2 设计任务 3](#_Toc4954008)

[1.3 设计要求 3](#_Toc4954009)

[1.4 技术指标 4](#_Toc4954010)

[2 总体方案设计 6](#_Toc4954011)

[2.1 单周期CPU设计 6](#_Toc4954012)

[2.2 中断机制设计 10](#_Toc4954013)

[2.3 流水CPU设计 12](#_Toc4954014)

[2.4 气泡式流水线设计 13](#_Toc4954015)

[2.5 重定向流水线设计 13](#_Toc4954016)

[2.6 动态分支预测机制 14](#_Toc4954017)

[3 详细设计与实现 16](#_Toc4954018)

[3.1 单周期CPU 实现 16](#_Toc4954019)

[3.2 中断机制实现 27](#_Toc4954020)

[3.3 流水CPU实现 30](#_Toc4954021)

[3.4 气泡式流水线实现 31](#_Toc4954022)

[3.5 重定向流水线实现 33](#_Toc4954023)

[3.6 动态分支预测机制实现 36](#_Toc4954024)

[4 实验过程与调试 37](#_Toc4954025)

[4.1 测试用例和功能测试 37](#_Toc4954026)

[4.2 性能分析 40](#_Toc4954027)

[4.3 主要故障与调试 41](#_Toc4954028)

[4.4 实验进度 44](#_Toc4954029)

[5 设计总结与心得 45](#_Toc4954030)

[5.1 课设总结 45](#_Toc4954031)

[5.2 课设心得 45](#_Toc4954032)

[参考文献 47](#_Toc4954033)

# 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 | LUI | 置寄存器高半字 |  |
| 30 | LH | 加载半字 |  |
| 31 | BLTZ | 小于跳转 |  |

# 总体方案设计

## 单周期CPU设计

在单周期设计中，采用的方案是硬布线控制，为避免部件冲突，采用指令存储器和数据存储器分离的哈佛结构。该CPU支持如表 1.1所示的24条基本指令、4条拓展CCMB指令。包括add, addi, addiu, addu, and, andi, sll, sra, srl, sub, or, ori, nor, lw, sw, beq, bne, slt, slti, sltu, j, jal, jr, syscall, xor, lui, lh, bltz。在实施的过程中，采用logism完成电路的硬件搭建，然后在FPGA平台完成28条单周期MIPS CPU的上板验证。

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CPU的组成结构，CPU应当划分为程序计数器PC、指令存储器IM、运算器ALU、寄存器堆RE、数据存储器DM几部分。

1. 程序计数器PC

对于单周期CPU来说，每当时钟周期到来时，PC计数器的值要”+1”，然后以PC为地址访问指令存储器，获得将要执行的指令。因此取指令操作涉及PC、加法器和指令存储器。其中PC可以使用32位寄存器来实现，”+1”操作使用加法器组件实现，考虑到在Logism中32位的存储器是按照字进行编址的，因此PC每次加1（若是按照字节编址，则PC每次加4）。

1. 指令存储器IM

在Logism中，指令存储器采用rom实现，输入为10位的指令地址，输出为32位的指令。在使用verilog进行设计时，指令存储器使用1024个32位的reg实现。

1. 运算器ALU

运算器ALU是CPU的核心部件，完成操作数的算术和逻辑运算，输入与输出端描述如表 2.1所示，X、Y是从寄存器文件或位拓展器读入的数据，Result、Result2是运算结果输出端口，ALU\_OP是运算功能码，其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 |
| Shamt | 输入 | 5 | 移位指令移动的位数 |
| 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功能说明

|  |  |  |
| --- | --- | --- |
| ALU\_OP | 十进制 | 运算功能 |
| 0000 | 0 | Result = Y << shamt 逻辑左移 Result2=0 |
| 0001 | 1 | Result = Y >>>shamt 算术右移 Result2=0 |
| 0010 | 2 | Result = Y >> shamt 逻辑右移 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 |

1. 寄存器堆RF

在Logism平台，寄存器文件直接使用cs3410库提供的寄存器文件。在verilog中，需要使用31个32位的寄存器来替代。需要注意的是，0号寄存器恒为0，因此对于0号寄存器的读写需要特殊处理。

1. 数据存储器DM

与指令存储器类似，在Logism中，数据存储器采用ram实现，输入为10位的数据地址，输出为32位的数据。在使用verilog进行设计时，指令存储器使用1024个32位的reg实现，而且需要增加写使能控制信号。对于需要实现SB指令的组员，还要增加片选信号控制写入。

### 数据通路的设计

逐条分析每一条指令的功能，根据指令功能完成指令系统数据通路框架，分析结果如表 2.3所示。Imm表示由立即数拓展的32位立即数。

表 2‑3指令系统数据通路框架

| 指令 | PC | IM | RF | | | | ALU | | | DM | | Tube |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1# | R2# | W# | Din | A | B | OP | Addr | Din |
| ADD | PC+4 | PC | IM | IM | IM | ALU | R1 | R2 | 0101 | / | / | / |
| ADDI | PC+4 | PC | IM | Imm | IM | ALU | R1 | IM | 0101 | / | / | / |
| ADDIU | PC+4 | PC | IM | Imm | IM | ALU | R1 | IM | 0101 | / | / | / |
| ADDU | PC+4 | PC | IM | IM | IM | ALU | R1 | R2 | 0101 | / | / | / |
| AND | PC+4 | PC | IM | IM | IM | ALU | R1 | R2 | 0111 | / | / | / |
| ANDI | PC+4 | PC | IM | Imm | IM | ALU | R1 | IM | 0111 | / | / | / |
| SLL | PC+4 | PC | IM | IM | IM | ALU | / | IM | 0000 | / | / | / |
| SRA | PC+4 | PC | IM | IM | IM | ALU | / | IM | 0001 | / | / | / |
| SRL | PC+4 | PC | IM | IM | IM | ALU | / | IM | 0010 | / | / | / |
| SUB | PC+4 | PC | IM | IM | IM | ALU | R1 | R2 | 0110 | / | / | / |
| OR | PC+4 | PC | IM | IM | IM | ALU | R1 | R2 | 1000 | / | / | / |
| ORI | PC+4 | PC | IM | Imm | IM | ALU | R1 | IM | 1000 | / | / | / |
| NOR | PC+4 | PC | IM | IM | IM | ALU | R1 | R2 | 1010 | / | / | / |
| LW | PC+4 | PC | IM | Imm | IM | DM | R1 | IM | 0101 | ALU | / | / |
| SW | PC+4 | PC | IM | Imm | IM | / | R1 | IM | 0101 | ALU | R2 | / |
| BEQ | RF | PC | IM | IM | / | ALU | R1 | R2 | / | / | / | / |
| BNE | RF | PC | IM | IM | / | ALU | R1 | R2 | / | / | / | / |
| SLT | PC+4 | PC | IM | IM | IM | ALU | R1 | R2 | 1011 | / | / | / |
| SLTI | PC+4 | PC | IM | Imm | IM | ALU | R1 | IM | 1011 | / | / | / |
| SLTU | PC+4 | PC | IM | IM | IM | ALU | R1 | R2 | 1100 | / | / | / |
| J | IM | PC | / | / | / | / | / | / | / | / | / | / |
| JAL | IM | PC | / | / | $31 | PC+4 | / | / | 0101 | / | / | / |
| JR | RF | PC | IM | / | IM | / | / | / | / | / | / | / |
| SYSCALL | PC+4 | PC | $v0 | $a0 | IM | / | / | / | / | / | / | R1,R2 |
| XOR | PC+4 | PC | IM | IM | IM | ALU | R1 | R2 | 1001 | / | / | / |
| LUI | PC+4 | PC | IM | / | IM | ALU | R1 | / | / | / | / | IM |
| LH | PC+4 | PC | IM | Imm | IM | DM | R1 | IM | 0101 | ALU | / | / |
| BLTZ | RF | PC | IM | IM | / | ALU | R1 | R2 | 1011 | / | / | / |

### 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 | 信号说明 | 产生条件（信号为1） |
| 1 | RegWrite | 寄存器写使能 | 寄存器写回信号 |
| 2 | MemWrite | 写内存控制信号 | sw指令 未单独设置MemRead信号 |
| 3 | AluOP | 运算器操作控制符（4位） | R型指令根据Func选择 |
| 4 | MemToReg | 寄存器写入数据来自存储器 | lw指令 |
| 5 | RegDst | 写入寄存器编号rt/rd选择 | R型指令 |
| 6 | AluSrcB | 运算器B输入选择 | lw指令，sw指令，立即数运算类指令 |
| 7 | SignedExt | 立即数符号扩展 | ADDI、ADDIU、SLTI指令 |
| 8 | JR | 寄存器跳转指令译码信号 | JR指令 |
| 9 | JAL | JAL指令译码信号 | JAL指令 ，选择寄存器写回编号，写回值 |
| 11 | JMP | 无条件分支控制信号 | J、JAL、JR指令，选择无条件分支地址 |
| 12 | Beq | Beq指令译码信号 | Beq指令，用于有条件分支控制 |
| 13 | Bne | Bne指令译码信号 | Bne指令，用于有条件分支控制 |
| 14 | 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 |
| 15 | LUI | LUI指令译码信号 | LUI指令 |
| 16 | LH | LH指令译码信号 | LH指令 |
| 17 | BLTZ | BLTZ指令译码信号 | BLTZ指令，用于有条件分支控制 |

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 2.5所示。其中后三个信号是4条拓展指令的控制信号，其余的是24条基本指令的控制信号。

表 2‑5主控制器控制信号框架



## 中断机制设计

### 总体设计

中断的处理过程可以分为中断识别、中断响应、中断服务、中断返回四部分。首先是中断识别，采用独立请求的方式，来自中断请求寄存器(IR)的多个中断请求信号接入优先编码器，优先编码器根据响应优先级生成中断号，完成中断识别，并将优先编码器所有IR输入信号逻辑或后得到中断请求信号，与中断使能寄存器(IE)逻辑与后送CPU，CPU根据此信号判断是否进入中断处理流程。中断响应部分是当CPU收到中断信号后，暂停当前程序的执行，执行中断隐指令，包括关中断、当前的PC值送EPC保护，中断向量送PC。中断服务由是处理中断的程序，由软件完成。中断返回由eret指令控制，恢复发生中断的那一条指令处继续执行。对于多级中断来说，还需要增加中断MFC0和MTC0指令以及中断嵌套的硬件逻辑电路。

### 硬件设计

硬件需要完成中断识别、中断响应、中断返回三个过程。实验中采取独立请求的方式进行中断识别，来自中断请求寄存器(IR)的多个中断请求信号接入优先编码器，优先编码器根据响应优先级生成中断号，完成中断识别，并将优先编码器所有IR输入信号逻辑或后得到中断请求信号，与中断使能寄存器(IE)逻辑与后送CPU，CPU根据此信号判断是否进入中断处理流程。CPU在收到中断请求信号后进入中断响应阶段，需要执行关中断、保存当前PC值到EPC寄存器，并将中断号对应的中断向量送PC，进入中断服务过程。在收到eret信号后，需要重新开中断，并将EPC的值送PC回到中断点继续执行。

对于多级中断，需要增加中断优先级判断逻辑来确定是否可以进行中断嵌套，可以增加寄存器来保存当前正在执行的中断的优先级，并增加比较器来比较优先级判断是否可以中断嵌套。此外还要增加MFC0和MTC0指令来保护EPC的值。

对于流水中断，为简化设计，选择在WB阶段进行中断处理，逻辑与单级中断相同，不过在中断产生和中断返回时需要清空流水线。

### 软件设计

首先需要确定中断服务程序的入口地址，为简化电路设计，在程序编写完毕后通过MARS获取三个中断服务程序的入口地址，并由中断号来选择中断服务程序。由于中断服务程序会破坏中断点的寄存器的值，因此中断服务程序需要执行保护现场的操作，将程序中使用到的寄存器压栈保护，并且在退出中断服务程序前恢复现场。

需要注意的是，在多级中断中由于会发生中断嵌套，EPC 的值也需要保存，避免更高级的中断破坏低级中断的EPC。EPC 的值的保存由软件完成，首先使用MFC0指令获取EPC的值到某一寄存器中，然后保护该寄存器的值，在退出中断时使用MTC0指令恢复EPC的值。

## 流水CPU设计

### 总体设计

根据指令的执行过程，指令可以分为以下5个阶段：取指令(IF)、译码(ID)、执行(EX)、访存(MEM)、写回(WB)。本次实验实现的流水线CPU在于将这5个阶段分离，每个阶段执行一条指令，实现时间重叠来提高CPU的运算速度。为了保存指令的运算结果，每个阶段后面都需要增加一个流水接口部件，用于锁存本阶段处理完成的数据结果，以保证下一阶段的使用。

### 流水接口部件设计

每一个流水接口应该包含时钟输入、使能端、同步清零端、数据输入端以及数据输出端。根据流水接口部件位置的不同数据输入端以及数据输出端的数量可能不同。流水接口应该完成如下的功能：在时钟上升沿，如果清零信号为1，则清空流水寄存器。否则如果使能信号为1，则使用输入端口的值更新流水寄存器，如果使能信号为0，则保持流水寄存器的值不变。在实验中为了便于调试，每一个流水接口部件都增加了PC和Instruction接口。

### 理想流水线设计

完成了流水接口部件，为了方便电路的连接，决定在28条指令CPU的基础上重绘数据通路。首先，对较大的部件进行再封装，以减少原理图中的连线。其次，为了方便控制信号的传输，将所有控制信号使用分线器统合为一根线，在使用时再通过分线器分离出来。最后，将对应的信号连接到流水接口部件对应的输入端，在使用信号时从流水接口部件的输出端引出，注意不能从译码阶段引出。此外，Syscall指令需要在WB阶段执行。

## 气泡式流水线设计

### 总体设计

由于指令是并行执行的，因此指令间会存在数据冲突，一种解决冲突的方法就是在指令间插入气泡，直到冲突消失。气泡流水线在理想流水线上增加了逻辑电路进行数据冲突检测电路，如果发现数据冲突则通过硬件插入气泡的方式消除数据冲突，此外，由于ID阶段需要取操作数，所以冲突检测逻辑电路需要设置在ID段，气泡是通过流水接口部件的使能端和重置端来产生的。最后，WB阶段的RAW数据相关冲突可以设置寄存器文件下降沿更新的方式来解决。

### 数据相关检测设计

数据相关逻辑电路的输入信号是ID阶段的指令、ID阶段使用到的寄存器以及EX、MEM阶段的写寄存器编号和写寄存器信号，输出信号是发生数据相关。数据相关的检测逻辑是：1.ID阶段使用的寄存器编号和EX阶段写寄存编号相同并且EX阶段写信号为1，则存在数据相关。2. ID阶段使用的寄存器编号和MEM阶段写寄存编号相同并且MEM阶段写信号为1，则存在数据相关。由于0号寄存器的值恒为0，因此0号寄存器不会产生数据相关冲突，需要特殊处理。

### 气泡流水线设计

完成了数据相关检测电路后，如果数据相关信号为1，则在电路中添加气泡，气泡的产生逻辑是：置IF/ID流水接口部件的使能端为0，同步清空ID/EX流水接口部件。经过一个时钟周期后，EX阶段就会插入一个气泡。由于指令的执行过程中会存在气泡，因此运行的benchmark程序的周期数会有所增加，具体的计算规则是：

总周期数=指令条数+(流水充满时间-1)+(分支跳转次数)\*预取深度+数据相关气泡数。

## 重定向流水线设计

### 总体设计

除了插入气泡，另一种解决冲突的方式是Forward技术，即增加额外的数据通路将MEM、WB阶段的数据重定向到EX阶段。在本实验中，重定向的数据来源有4种：EX的寄存器文件输出、WB阶段的RAM输出、WB阶段的ALU输出、MEM阶段的ALU输出，均有可能重定向到EX阶段的寄存器R1或R2的位置。因此需要增加重定向检测逻辑电路来判断使用哪一路的数据。与数据相关检测类似，重定向检测也要设置在ID阶段，不过由于重定向过程发生在EX阶段，因此需要将重定向信号通过流水接口部件传输到EX阶段。

### 重定向检测电路设计

与气泡流水线的数据相关检测电路类似，重定向检测电路的输入信号是ID阶段的指令、ID阶段使用到的寄存器以及EX、MEM阶段的写寄存器编号和写寄存器信号，输出信号是重定向通路的选择信号。以重定向MEM阶段的ALU输出到EX阶段的R1为例，数据相关的检测逻辑是：ID阶段的指令使用到了寄存器R#，EX阶段的写寄存器编号为R#，写寄存器信号为1，MemToReg信号为0，则在下一个时钟周期需要将MEM阶段的ALU运算结果重定向到EX阶段的R1的位置。

### 重定向流水线设计

在ID/EX流水接口部件的R1、R2处使用4选1的多路选择器的输出替换原来的R1、R2，多路选择器的控制信号有重定向检测电路产生，以R1为例，多路选择器的输入有：ID/EX.R1, WB.Mem, WB.AluR, Mem.AluR。需要特别注意的是，重定向的选择控制信号产生在ID阶段，而使用则发生在EX阶段，因此需要通过流水接口部件进行传输。

## 动态分支预测机制

动态分支预测机制的核心是实现BHT，本质上就是一个Cache，可以利用组成原理实验中实现的Cache。经过分析，得到BHT的格式如表 2‑6所示，有8个表项。其中有效位用于标记表现是否有效，分支指令地址是关键字，用于在BHT种作全相连比较，分支目标地址作为Cache的输出，置换标记用于LRU算法进行替换。

表 2‑6 BHT格式

|  |  |  |  |
| --- | --- | --- | --- |
| Valid | 分支指令地址 | 分支目标地址 | LRU置换标记 |
| 1/0 | xxx | xxx | xxx |

在实现时BHT放在取指令阶段，利用PC的值作为关键字进行全相连比较，如果BHT命中，则表明当前指令是一条分支跳转指令，选择该表项中的分支目标地址作为新的PC值。当该指令执行到EX段时，判断是否预测正确，如果正确则继续执行，如果预测失败，则应该清空预取的指令。动态分支预测的流程图如图 2‑2所示：

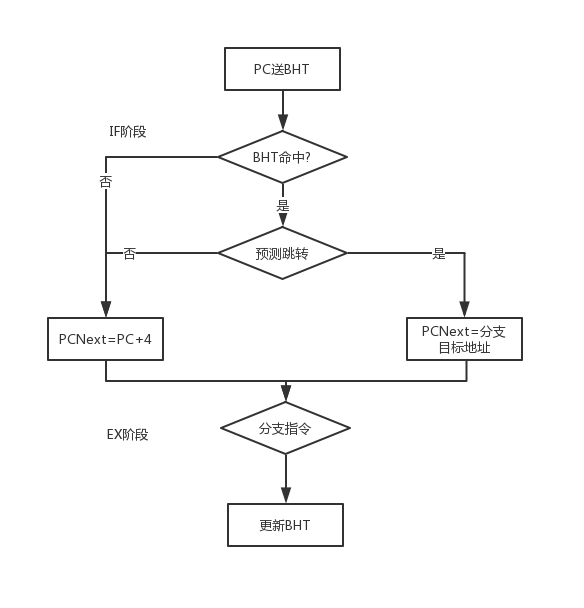


图 2‑2 动态分支预测流程

# 详细设计与实现

指令周期流程图要在此部分出现、微程序流程图、微指令代码表、实验接线图等均需要在适当的位置和模块中表达出来。本章具体实现细节尽量多用图表方式展示，但要做到图文并茂，不能全文都是图。

## 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，接到寄存器的使能端，当Halt为0时表示停机，此时PC寄存器保存原有值不变。Rst为清零信号，用于重置电路。Logism实现如图 3.1所示。

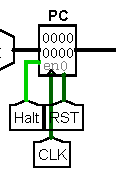


图 3‑1程序计数器（PC）

1. FPGA实现：

程序计数器PC的Verilog代码如下：

module Register(Clk,Rst,Data,En,Out);

parameter N=32;

input [N-1:0] Data;

input En,Clk,Rst;

output reg [N-1:0] Out=0;

always@(posedge Clk, posedge Rst) begin

if(Rst==1) Out=0;

else if(En==1) Out=Data;

end

endmodule

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0-9位作为指令存储器的输入地址。如图 3.2所示。

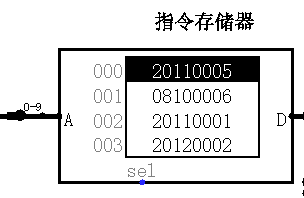


图 3‑2指令存储器（IM）

1. FPGA实现：

在使用verilog语言设计时，指令存储器使用1024个32位的reg实现，并且需要使用Initial语句加载benchmark程序到Rom中。

指令存储器IM的Verilog代码如下：

module RomMxN(Clk,Addr,Data);

//地址线宽度为M，数据宽度为N

parameter N=32;

parameter M=10;

parameter T=2\*\*M; //2的M次方

input Clk; //清空

input [M-1:0] Addr;//地址

output [N-1:0] Data;//数据

reg [N-1:0] mem [T-1:0];//T个N位的存储器

initial begin // 初始化存储器

$readmemh("C:/Users/10334/Desktop/MIPSCPU/benchmark.txt",mem);//加载路径

end

assign Data=mem[Addr];

endmodule

1. 运算器ALU
   1. Logism实现：

在logisim中，运算器ALU通过各种逻辑运算和算术运算部件实现，然后根据AluOp的值通过多路选择器选择运算结果。ALU的实现如图 3‑3所示。其中X和Y是32位的输入数据，shamt是移位指令的移动位数，S是AluOp，结果通过多路选择器选择输出到Result和Result2。

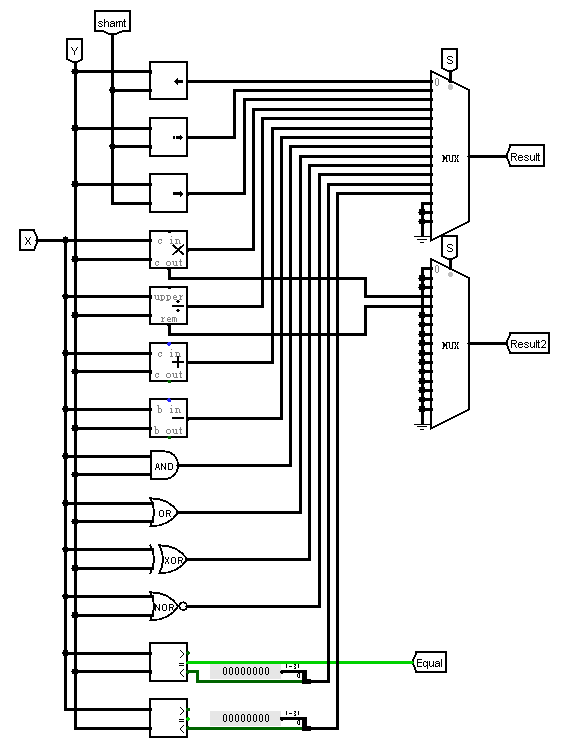


图 3‑3 ALU实现

* 1. FPGA实现：

在FPGA中，首先使用assign语句实现各个运算部件的，以加法器为例，verilog实现的语句是。

assign R=A+B;

然后在always语句中，以AluOp为选择控制信号，选择对应的结果到输出端。

ALU主模块的Verilog代码如下：

module ALU(A,B,S,shamt,Equal,R,R1);

parameter N=32;

input [N-1:0] A,B;//输入端

input [3:0] S;//AluOp

input [4:0] shamt;//移位数

output Equal;//相等比较

output reg [N-1:0] R,R1;//输出端

wire [N-1:0] out[12:0];

wire [N-1:0] out1[1:0];

SLL t0(B,out[0],shamt);//左移

SRA t1(B,out[1],shamt);//算术右移

SRL t2(B,out[2],shamt);//逻辑右移

MulU t3(A,B,out[3],out1[0]);//无符号乘法

DivU t4(A,B,out[4],out1[1]);//无符号除法

Add t5(A,B,out[5]);//加法

Sub t6(A,B,out[6]);//减法

And\_ t7(A,B,out[7]);//按位与

Or\_ t8(A,B,out[8]);//按位或

Xor\_ t9(A,B,out[9]);//按位异或

Nor\_ t10(A,B,out[10]);//按位或非

SignedCmp t11(A,B,out[11]);//有符号比较

UnsignedCmp t12(A,B,out[12]);//无符号比较

initial begin

R<=0;R1<=0;

end

assign Equal=(A==B);

always @(\*)

begin

case(S)

4'b0000:begin

R<=out[0];

R1<=32'bz;

end

4'b0001:begin

R<=out[1];

R1<=32'bz;

end

4'b0010:begin

R<=out[2];

R1<=32'bz;

end

4'b0011:begin

R<=out[3];

R1<=out1[0];

end

4'b0100:begin

R<=out[4];

R1<=out1[1];

end

4'b0101:begin

R<=out[5];

R1<=32'bz;

end

4'b0110:begin

R<=out[6];

R1<=32'bz;

end

4'b0111:begin

R<=out[7];

R1<=32'bz;

end

4'b1000:begin

R<=out[8];

R1<=32'bz;

end

4'b1001:begin

R<=out[9];

R1<=32'bz;

end

4'b1010:begin

R<=out[10];

R1<=32'bz;

end

4'b1011:begin

R<=out[11];

R1<=32'bz;

end

4'b1100:begin

R<=out[12];

R1<=32'bz;

end

default:begin

R<=32'bz;R1<=32'bz;

end

endcase

end

endmodule

1. 寄存器堆RF
   1. Logism实现：

在logism中，直接使用cs3410库中提供的寄存器文件即可，为了简化电路，对原部件进行封装，封装后的结果如图 3‑4所示。R1#、R2#是寄存器的编号，R1、R2是选择的寄存器的值，W#是写入寄存器编号，WE是寄存器写信号，Din是写入寄存器的数据，CLK是时钟端。

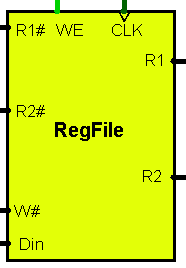


图 3‑4 Logism的RF实现

在FPGA平台，使用31个32位的寄存器来替代，此外0号寄存器需要特殊处理。

* 1. FPGA实现：

在FPGA平台，使用31个32位的寄存器来替代，此外0号寄存器需要特殊处理。RF模块的verilog代码如下：

module RegFile(R1In,R2In,WE,WIn,Din,Clk,R1,R2);

input WE,Clk;

input [4:0]WIn,R1In,R2In;

input [31:0]Din;

output [31:0]R1,R2;

reg [31:0]regFile[0:31];

assign R1=(R1In==0) ? 0 : regFile[R1In];

assign R2=(R2In==0) ? 0 : regFile[R2In];

always @(posedge Clk)

begin

if(WE==1)

regFile[WIn]=Din; //0号寄存器只读

end

endmodule

1. 数据存储器DM
   1. Logism实现：

与指令存储器类似，在Logism中，数据存储器采用ram实现，输入为10位的数据地址，输出为32位的数据。实现的电路图如图 3‑5所示：

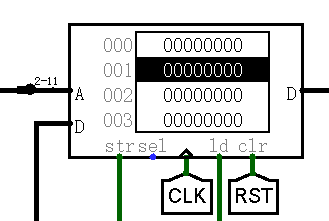


图 3‑5 Logism的DM实现

* 1. FPGA实现：

在使用verilog进行设计时，指令存储器使用1024个32位的reg实现，而且需要增加写使能控制信号。DM模块的verilog代码如下：

module RamMxN(Rst,Clk,Str,Ld,Addr,DataIn,DataOut);

parameter N=32;

parameter M=10;

parameter T=2\*\*M; //2的M次方

input Clk; //时钟

input Rst; //清空

input Str; //写信号

input Ld; //读信号

input [M-1:0] Addr; //地址

input [N-1:0] DataIn;//输入数据

output [N-1:0] DataOut;//输出数据

reg [N-1:0] mem[T-1:0];

integer i;

assign DataOut=mem[Addr];

always @(posedge Clk, posedge Rst)

begin

if (Rst) //清空

begin

for(i=0;i<=T-1;i=i+1) //reset, 按字操作

mem[i] <= 32'b0;

end

else if (Str) //写入

begin

mem[Addr] <= DataIn;

end

end

endmodule

### 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，详细内容见表 2‑3。

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。完成后的数据通路如图 3‑6。

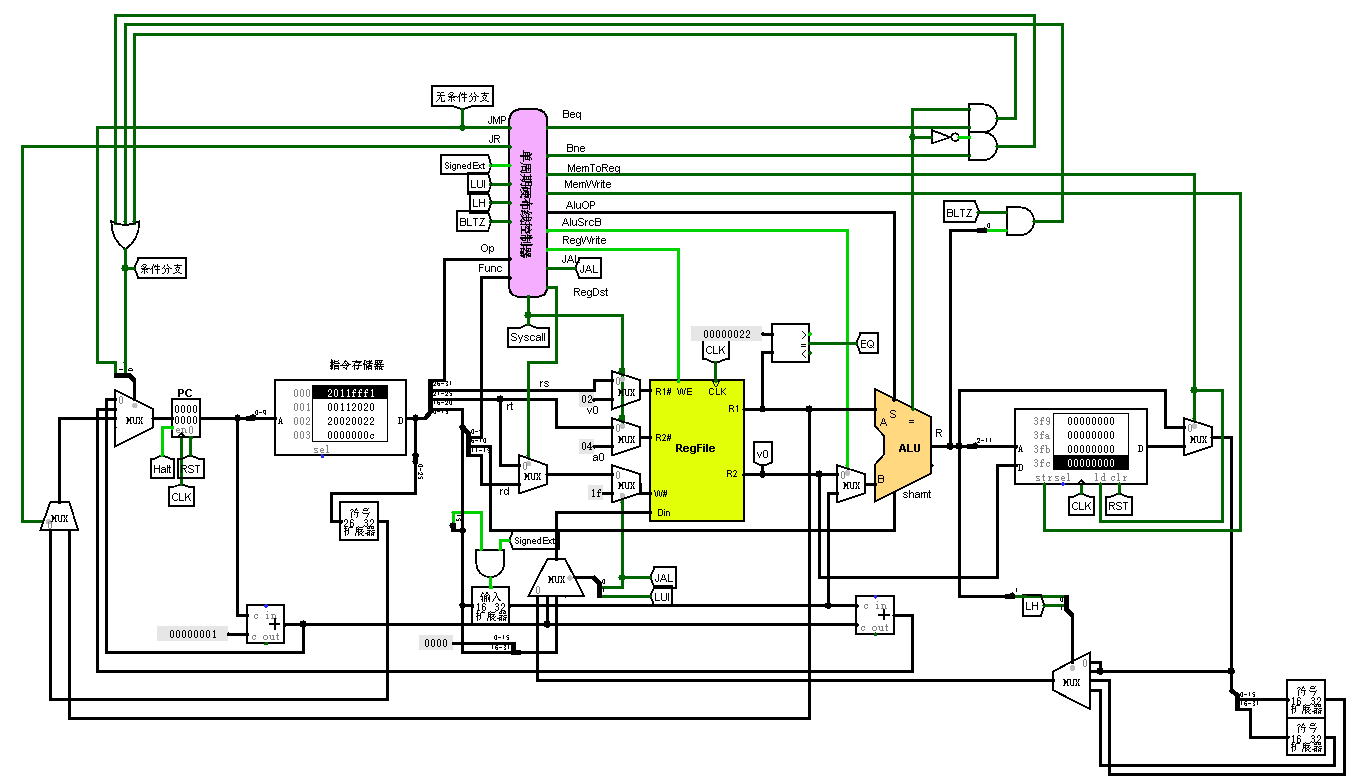


图 3‑6 单周期CPU数据通路（Logism）

在Vivado中使用Verilog语言搭建的数据通路的原理图如图 3‑7所示。

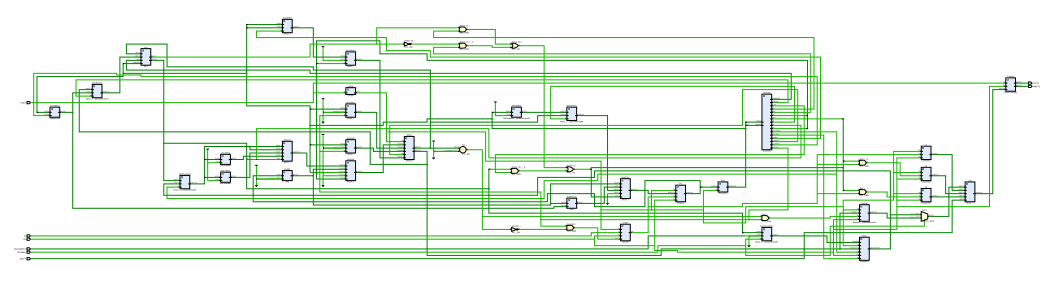


图 3‑7单周期CPU数据通路（FPGA）

### 控制器的实现

1. logism实现

在logism中，控制器使用硬布线的方式进行，根据如表 3‑1所示的输入指令与控制信号的对应，然后在logism中采用表达式自动生成的方式产生控制器。

表 3‑1 主控制器控制信号



实现后的logism的控制器模块如图 3‑8所示。

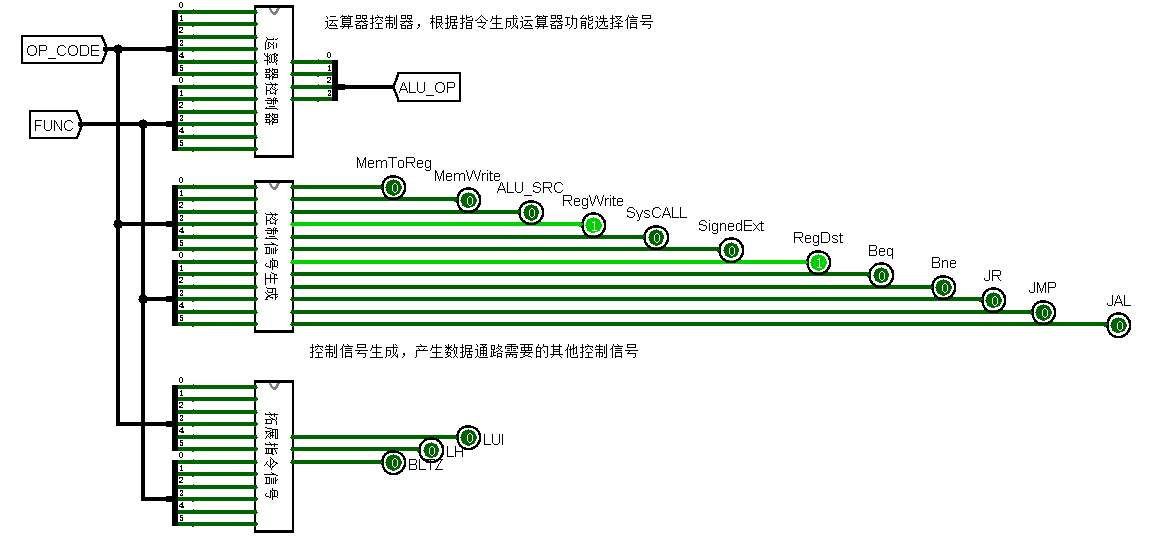


图 3‑8 logism控制器实现

1. FPGA实现

根据在Logism实现中得到的各个一位控制信号的表达式，直接使用数据流建模，使用assign分的Verilog代码过于冗长，故只取对于控制信号MemToReg的生成代码举例如下：

assign

MemToReg=(OP5)&(~OP4)&(~OP3)&(~OP2)&(OP1)&(OP0)|(OP5)&(~OP4)&(~OP3)&(~OP2)&(~OP1)&(OP0);

以此类推，最终便可以实现整个主控制器中所有控制信号的生成。在Vivado中使用Verilog语言构成的主控制器原理图如图 3‑9所示。

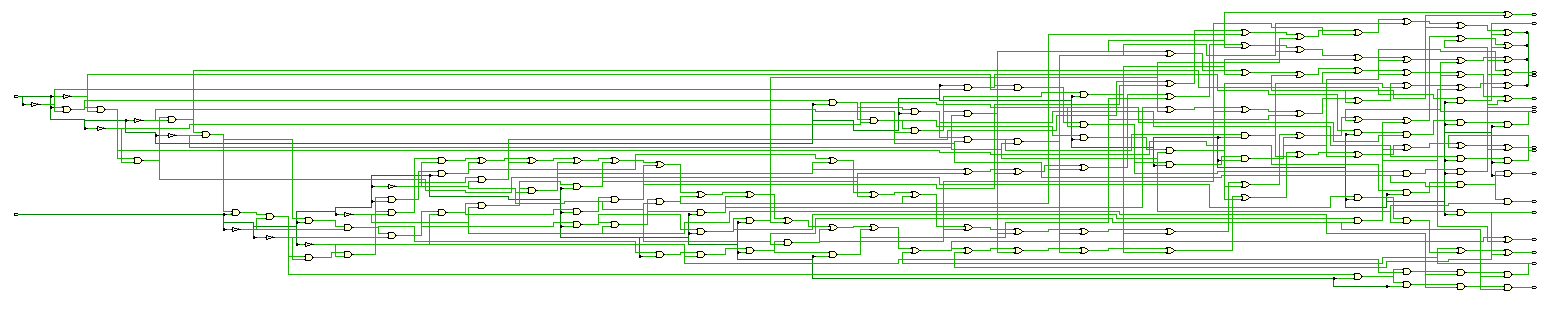


图 3‑9主控制器原理图

## 中断机制实现

### 单级中断实现

1. 中断采样电路实现

中断采样电路的实现如图 3‑10所示，IR1是中断请求寄存器，其中同步清零信号用于清楚中断请求信号。

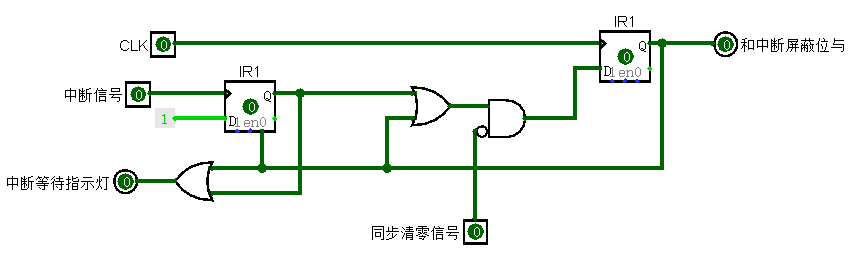


图 3‑10 中断采样电路实现

1. 中断逻辑电路实现

硬件需要完成中断识别、中断响应、中断返回三个过程。实验中采取独立请求的方式进行中断识别，来自中断请求寄存器(IR)的多个中断请求信号接入优先编码器，优先编码器根据响应优先级生成中断号，完成中断识别，并将优先编码器所有IR输入信号逻辑或后得到中断请求信号，与中断使能寄存器(IE)逻辑与后送CPU，CPU根据此信号判断是否进入中断处理流程。CPU在收到中断请求信号后进入中断响应阶段，需要执行关中断、保存当前PC值到EPC寄存器，并将中断号对应的中断向量送PC，进入中断服务过程。在收到eret信号后，需要重新开中断，并将EPC的值送PC回到中断点继续执行。中断逻辑电路的实现如图 3‑11所示。此外，PC寄存器的输入端口需要增加一个多路选择器在中断返回时将EPC的值送入PC，电路实现如图 3‑12所示。

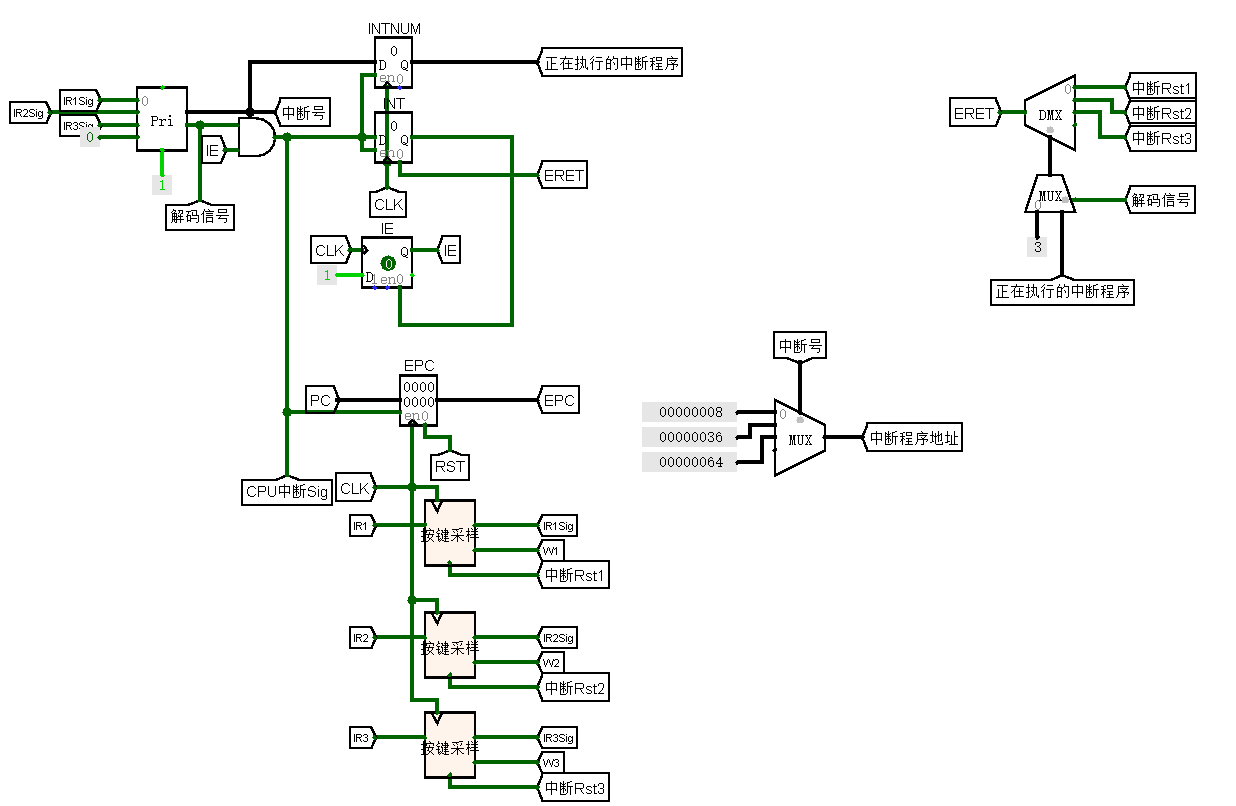


图 3‑11中断逻辑电路实现

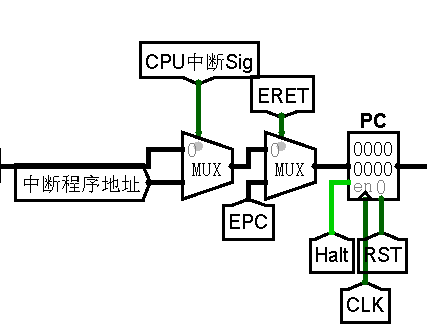


图 3‑12中断返回电路设计

### 多级中断实现

对于多级中断，需要在单级中断的基础上增加中断优先级判断逻辑来确定是否可以进行中断嵌套，可以增加寄存器来保存当前正在执行的中断的优先级，并增加比较器来比较优先级判断是否可以中断嵌套。此外还要增加MFC0和MTC0指令来保护EPC的值。多级中断的电路实现如图 3‑13所示。

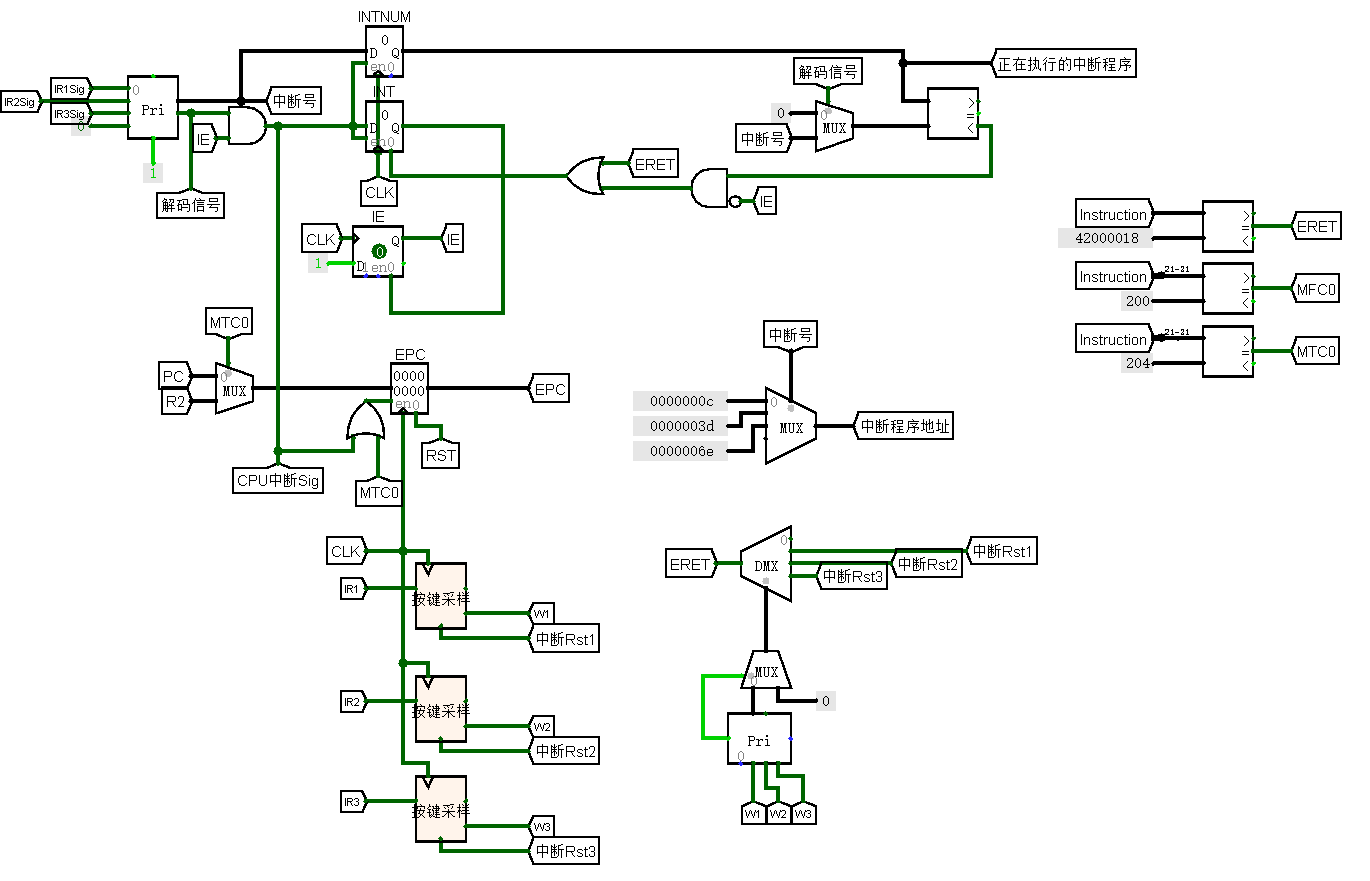


图 3‑13 多级中断电路实现

### 流水中断实现

对于流水中断，为简化设计，选择在WB阶段进行中断处理，中断逻辑电路与单级中断相同，在此基础上增加了中断产生和中断返回时清空流水线的操作。电路实现如图 3‑14所示，其中流水清空逻辑如图 3‑15所示。

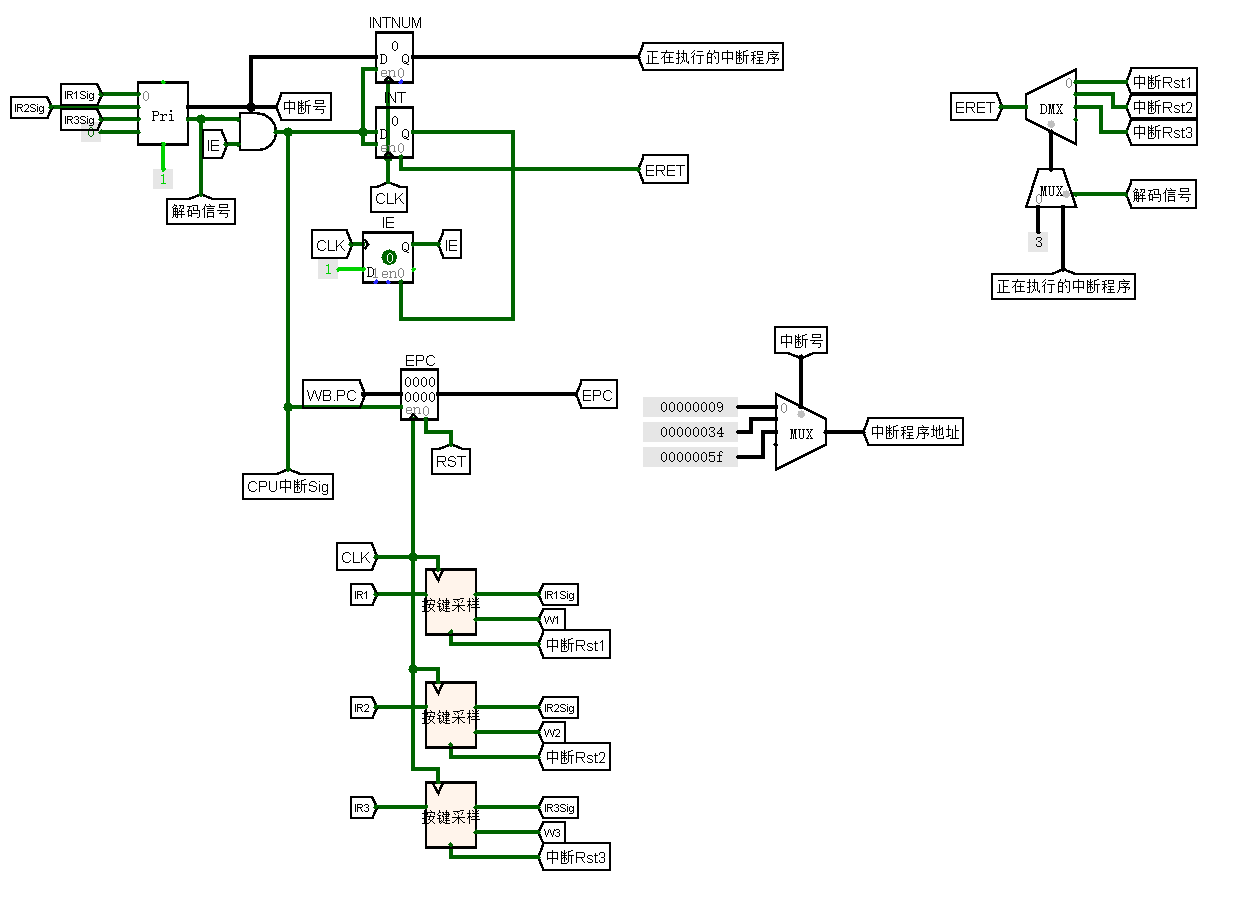


图 3‑14流水中断实现

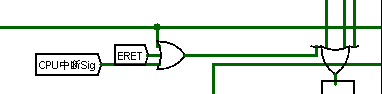


图 3‑15流水清空逻辑

## 流水CPU实现

### 流水接口部件实现

每一个流水接口应该包含时钟输入、使能端、同步清零端、数据输入端以及数据输出端。以IF/ID流水接口部件为例，其logism实现如图 3‑16所示。

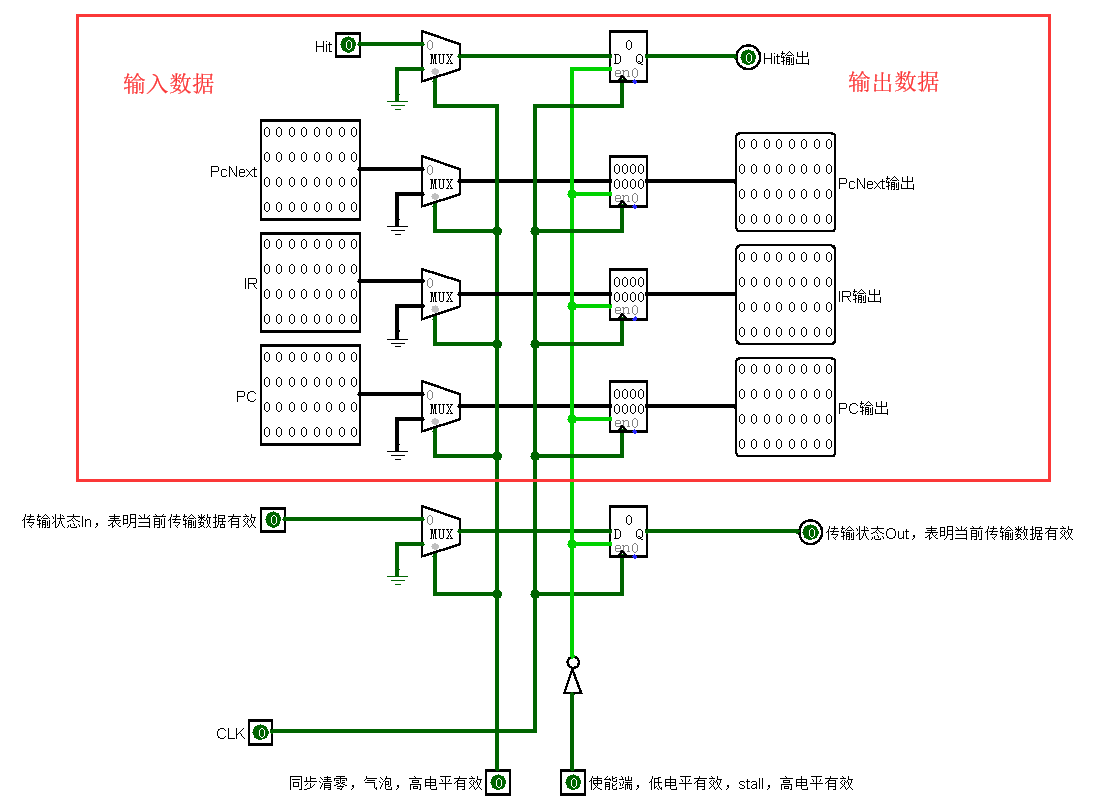


图 3‑16 IF/ID流水接口部件实现

### 理想流水线实现

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。理想流水线的实现如图 3‑17所示。

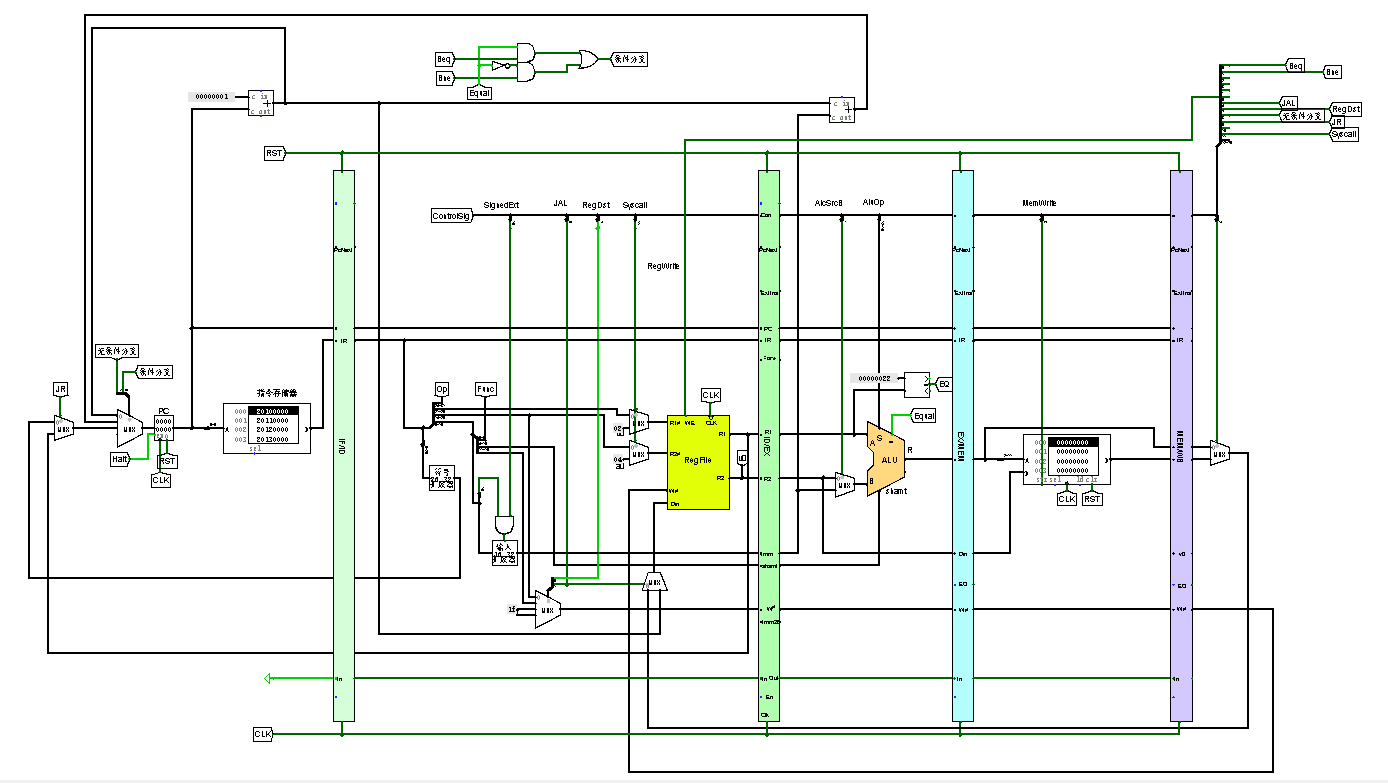


图 3‑17 理想流水线实现

## 气泡式流水线实现

1. 数据相关检测实现

气泡流水线在理想流水线上增加了逻辑电路进行数据冲突检测电路，如果发现数据冲突则通过硬件插入气泡的方式消除数据冲突，数据冲突的检测原理已经在设计中阐述过，其电路实现如图 3‑18所示。ID.X、EX.X、MEM.X分别表示ID阶段、EX阶段、MEM阶段的信号，需要特别注意，电路中对0号寄存器进行了特殊判断的处理。

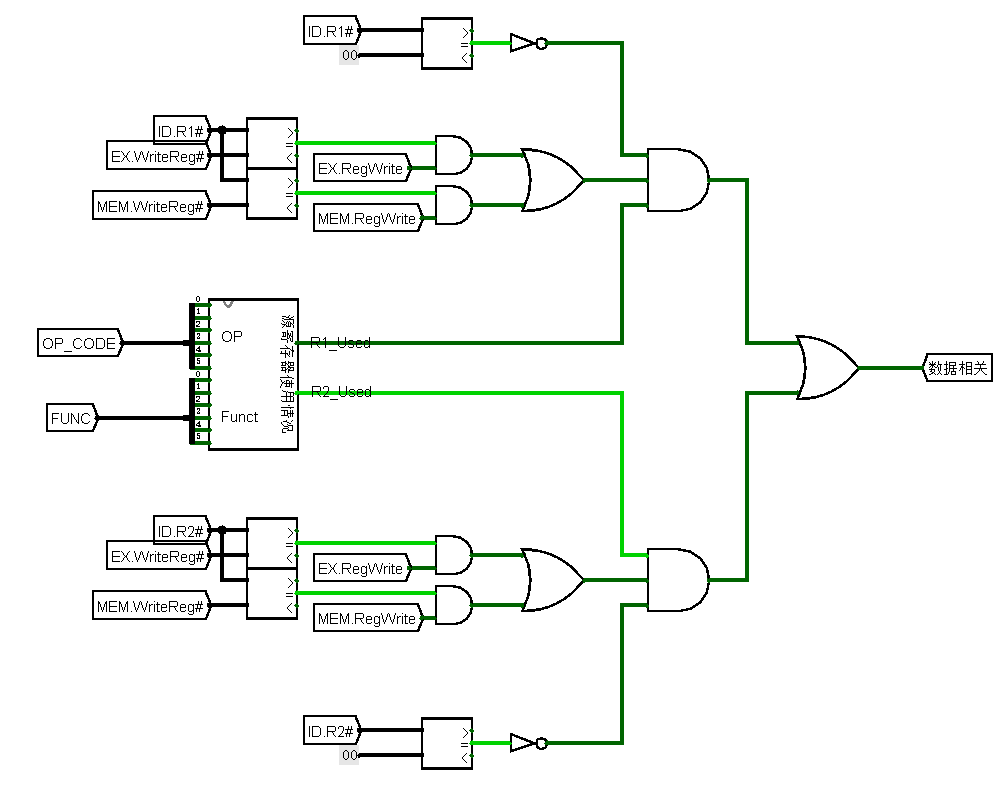


图 3‑18 数据相关检测实现

1. PC寄存器改造

当产生数据相关需要插入气泡时，PC寄存器需要暂停一个时钟周期，因此需要改造PC寄存器的使能端信号，即将数据相关信号取反后接入到PC的使能端，实现的电路如图 3‑19所示。

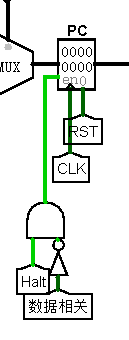


图 3‑19 PC寄存器改造

1. 流水接口部间改造

发生数据相关产生气泡时，需要置IF/ID流水接口部件的使能端为0，同步清空ID/EX流水接口部件。数据相关信号需要成为IF/ID的流水接口部件使能信号以及ID/EX流水接口部件的清空信号，电路实现如图 3‑20和图 3‑21所示。

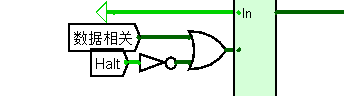


图 3‑20 IF/ID流水接口部件改造

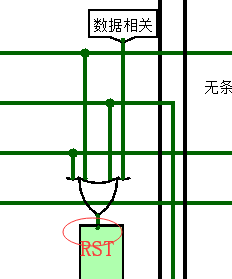


图 3‑21 ID/EX流水接口部件改造

1. 气泡流水线实现

在完成气泡发生器的构造，控制器以及程序计数器的修改之后，需要将其整合进入理想流水线中，形成气泡流水线。最终气泡流水线的实现如图 3‑22所示。

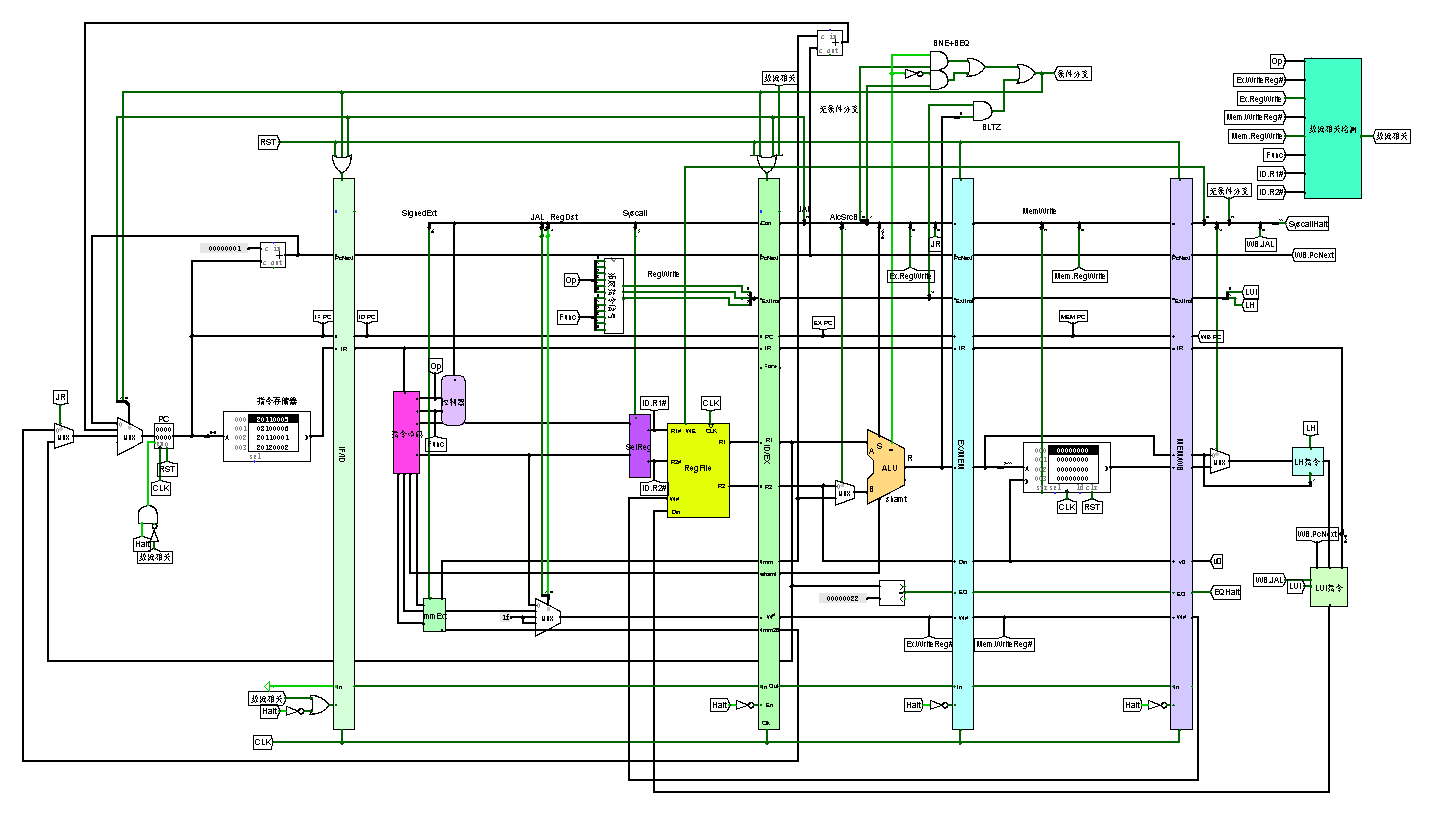


图 3‑22 气泡流水线实现

## 重定向流水线实现

1. 重定向相关检测实现

气泡流水线在理想流水线上增加了逻辑电路进行重定向相关检测电路，如果发现数据冲突则根据冲突类型生成forward数据，其电路实现如图 3‑23所示。ID.X、EX.X、MEM.X分别表示ID阶段、EX阶段、MEM阶段的信号，需要特别注意，电路中对0号寄存器进行了特殊判断的处理。

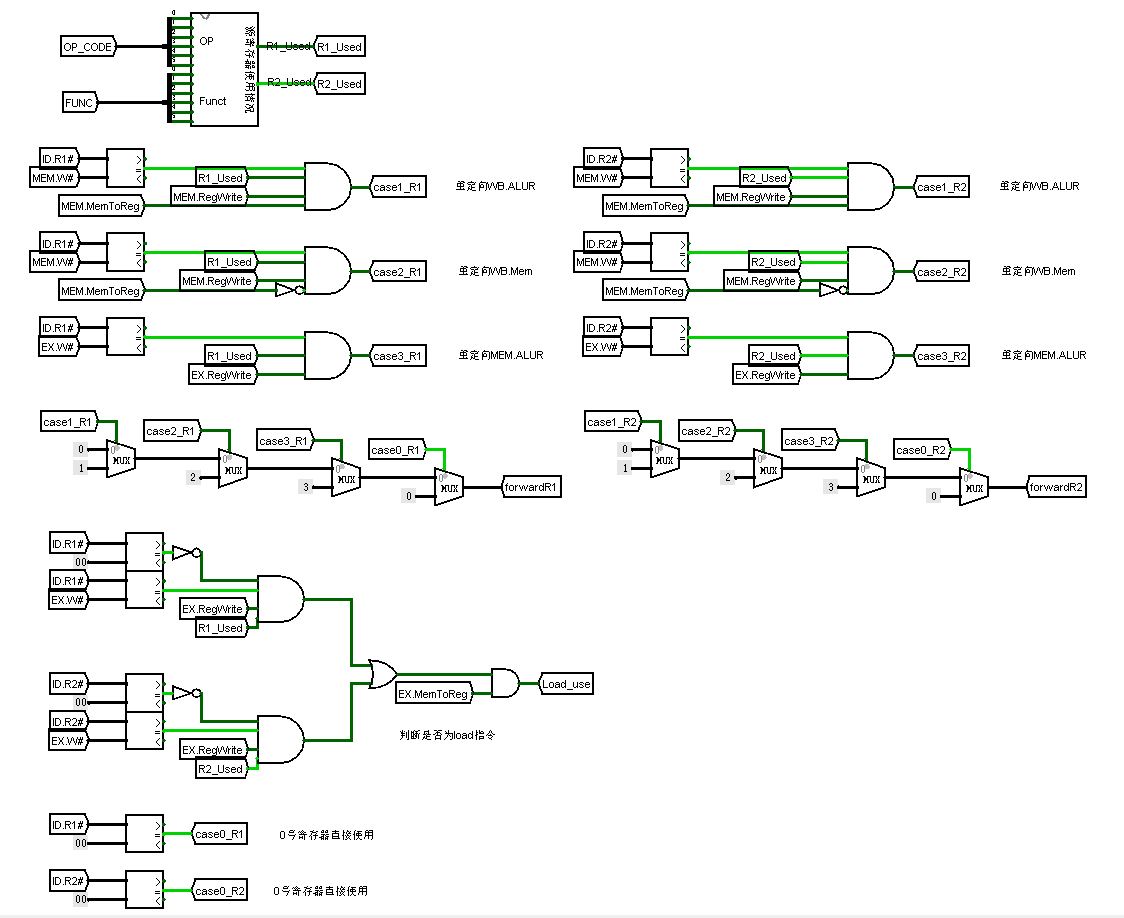


图 3‑23重定向相关检测实现

1. PC寄存器改造

当产生load\_use数据相关时需要插入气泡时，PC寄存器需要暂停一个时钟周期，因此需要改造PC寄存器的使能端信号，即将数据相关信号取反后接入到PC的使能端，实现的电路如图 3‑24所示。除此之外，与气泡流水线类似，IF/ID的流水部件要暂停一个周期，ID/EX流水部件要产生一个气泡。

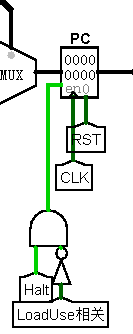


图 3‑24 PC寄存器改造

1. EX阶段改造

在ID/EX流水接口部件的R1、R2处使用4选1的多路选择器的输出替换原来的R1、R2，多路选择器的控制信号有重定向检测电路产生。电路实现如图 3‑25所示。

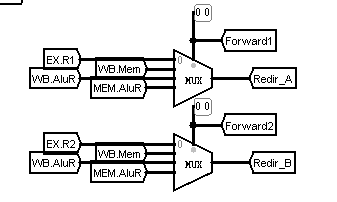


图 3‑25 数据重定向

1. 气泡流水线实现

在完成气泡发生器的构造，控制器以及程序计数器的修改之后，需要将其整合进入理想流水线中，形成重定向流水线。最终重定向流水线的实现如图 3‑26所示。

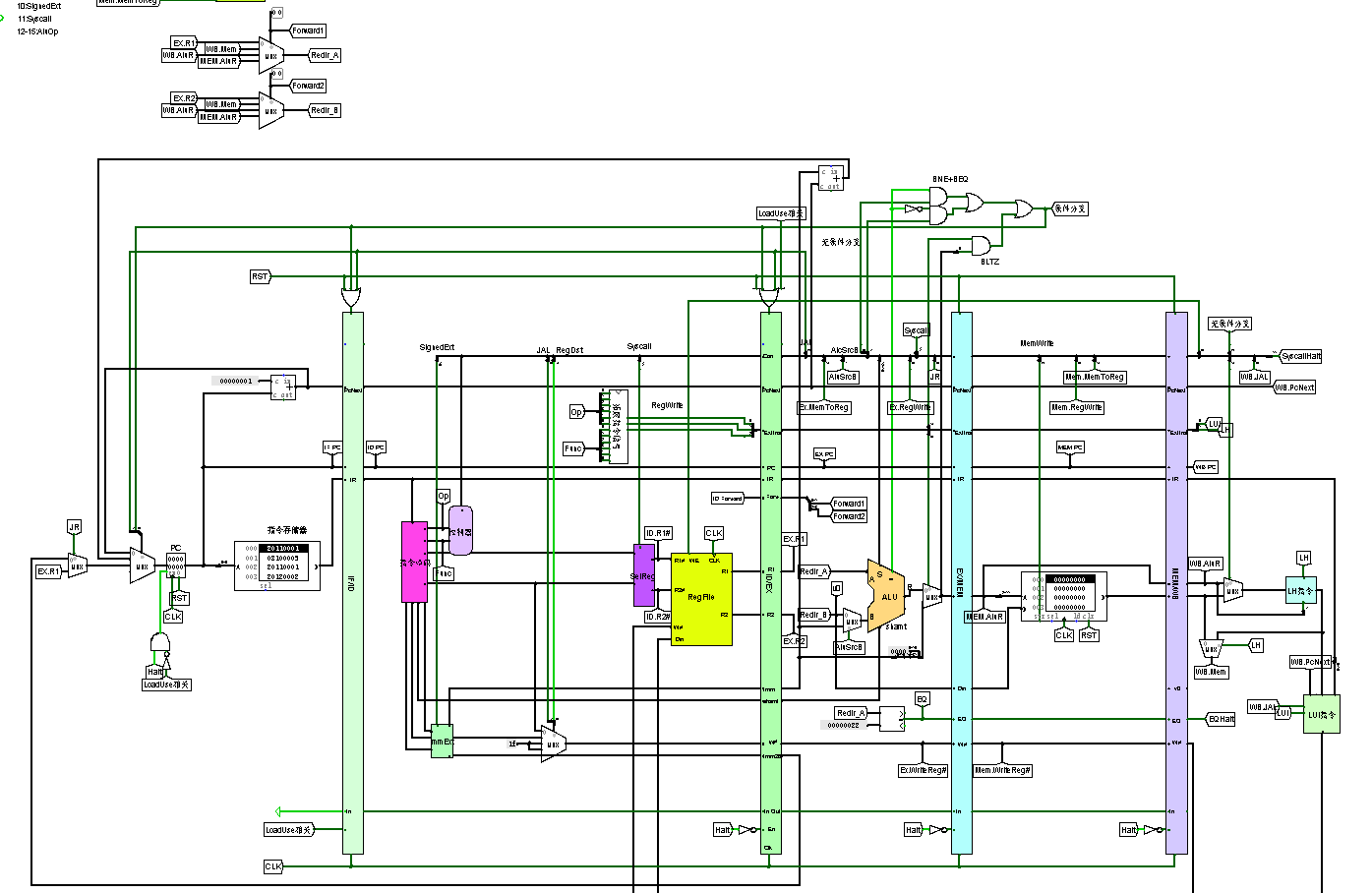


图 3‑26 重定向流水线实现

## 动态分支预测机制实现

要实现分支预测，最重要的是 BHT 表的实现以及分支预测和替换规则的实现，其中BHT可以利用组成原理实验中完成的Cache组件，其电路图如图 3‑27所示：

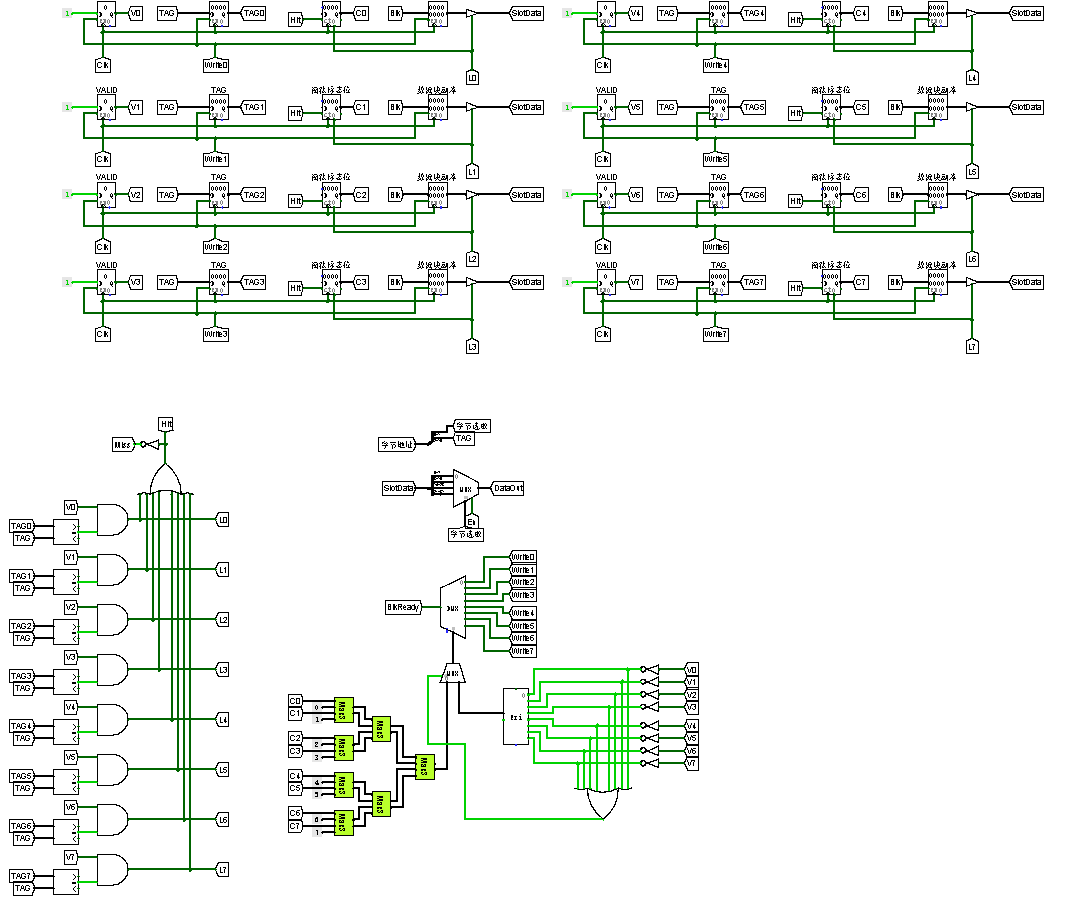


图 3‑27 BHT实现

在实现时BHT放在取指令阶段，利用PC的值作为关键字进行全相连比较，如果BHT命中，则表明当前指令是一条分支跳转指令，选择该表项中的分支目标地址作为新的PC值。当该指令执行到EX段时，判断是否预测正确，如果正确则继续执行，如果预测失败，则应该清空预取的指令。分支预测和替换规则的实现如图 3‑28所示

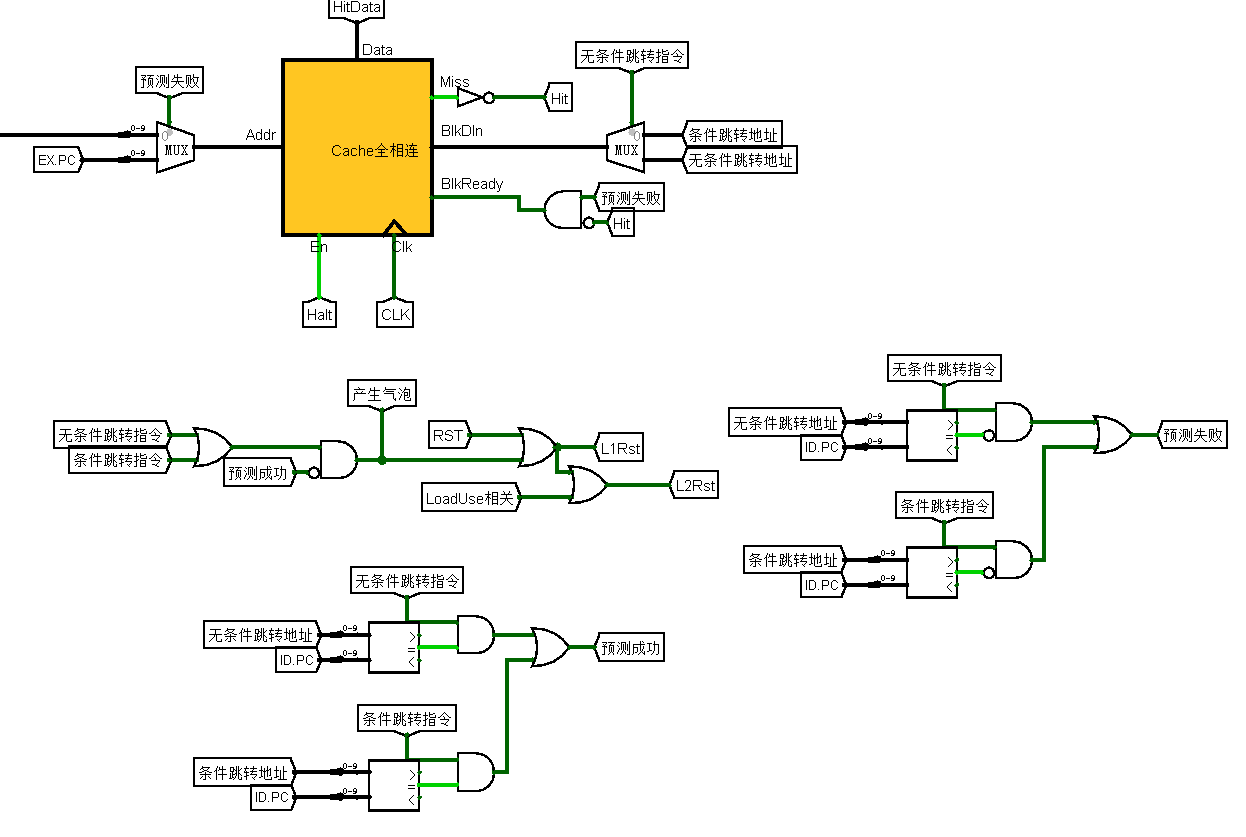


图 3‑28 分支预测实现

# 实验过程与调试

## 测试用例和功能测试

### benchmark及ccmb指令测试

对重定向流水线CPU进行测试，在benchmark尾部添加ccmb指令的测试代码，然有使用Mars程序生成十六进制机器码，加载到指令存储器中进行执行。在测试阶段，程序正确通过了跑马灯、移位、排序等测试，测试完毕后，打开Ram查看排序结果，如图 4‑1所示，与理论降序排列结果一致。

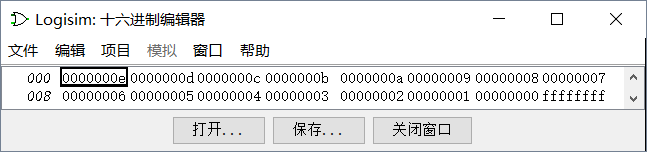


图 4‑1 Ram排序结果

程序运行完毕后，统计运行的周期数如图 4‑2所示，总周期数为2297，无条件分支数为38，条件分支数位276，LoadUse次数为120，与理论结果一致。

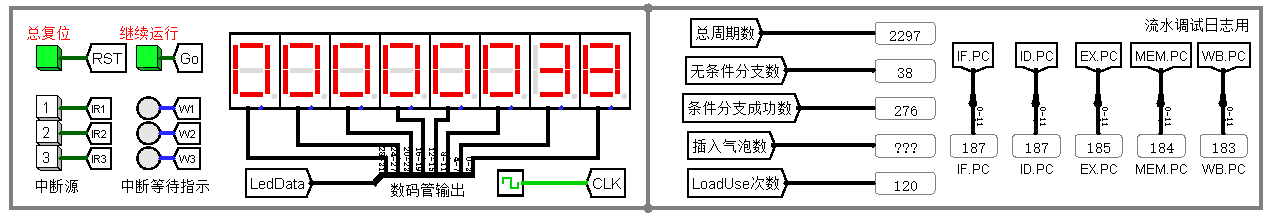


图 4‑2 程序运行周期数

接下来按下Go，依次观察XOR指令、LUI指令、LH指令、BLTZ指令运行时数码管的数据，与预期相符，说明CCMB指令也通过了测试。

### 28条单周期MIPS CPU仿真测试

在vivado中进行单周期MIPS CPU的仿真测试。仿真结果如图 4‑3所示，到达syscall暂停后，总周期数为1546，有条件分支数为276，无条件分支数为38，与理论结果一致。

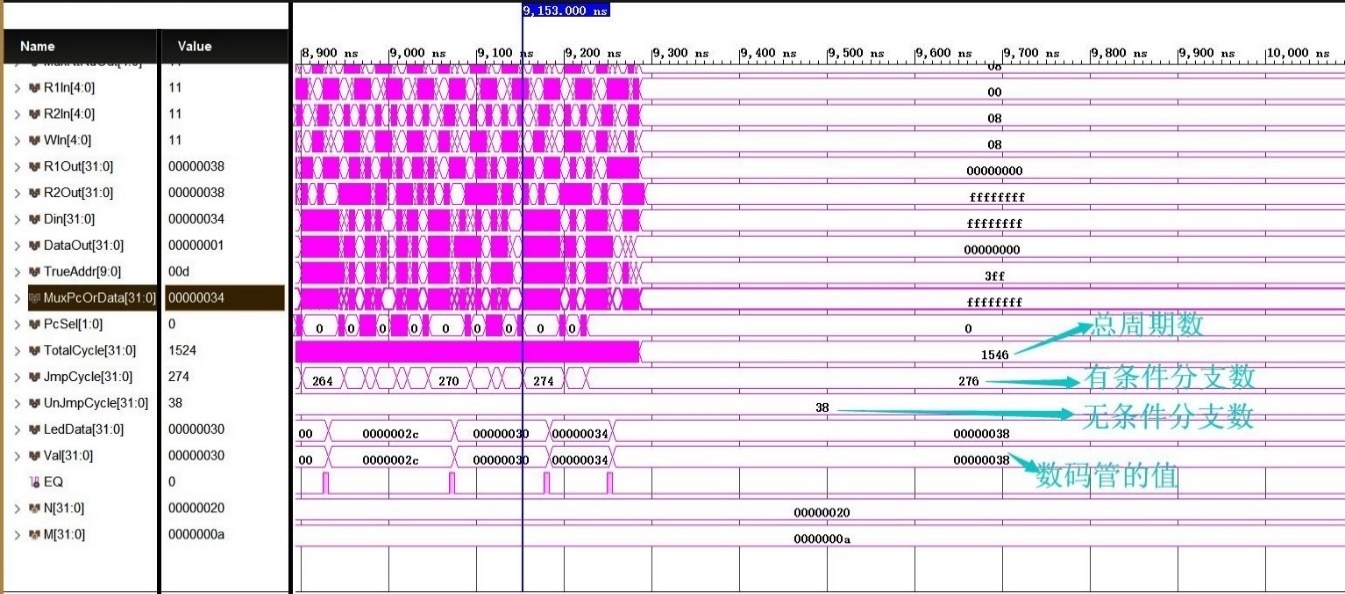
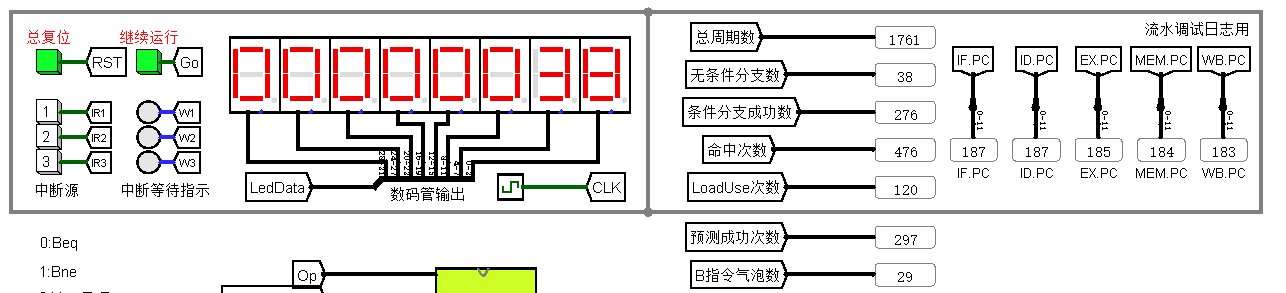


图 4‑3 仿真结果

查看RAM的数据，结果如图所示，为降序排列，说明排序结果正确。

### 动态分支预测CPU测试

在动态分支预测CPU中载入benchmark程序，得到的运行结果如图所示。总周期数位1761，相对于重定向流水线2297减少了536个周期。跳转指令设置在EX段执行，因此每命中成功一次就会减少2个周期的气泡，通过计数器可知预测成功了297次，除去B指令跳出时的29次，总共减少了2\*(297-29)=536个周期.



### 中断测试

以多级中断为例，将中断测试程序载入到Logism的Rom中，其中中断处理程序1的测试代码如下：

addi $sp, $sp,-4 #入口地址1

sw $v0, 0($sp)

addi $sp, $sp,-4

sw $a0, 0($sp)

addi $sp, $sp,-4

sw $s0, 0($sp)

addi $sp, $sp,-4

sw $s3, 0($sp)

addi $sp, $sp,-4

sw $s4, 0($sp)

addi $sp, $sp,-4

sw $s5, 0($sp)

addi $sp, $sp,-4

sw $s6, 0($sp)

mfc0 $s2,$0 #保护EPC

addi $sp, $sp,-4

sw $s2, 0($sp)

addi $s6,$zero,1 #中断号1,2,3 不同中断号显示值不一样

addi $s4,$zero,6 #循环次数初始值

addi $s5,$zero,1 #计数器累加值

IntLoop1:

add $s0,$zero,$s6

IntLeftShift1:

sll $s0, $s0, 4

or $s3,$s0,$s4

add $a0,$0,$s3 #display $s0

addi $v0,$0,34 # display hex

syscall # we are out of here.

bne $s0, $zero, IntLeftShift1

sub $s4,$s4,$s5 #循环次数递减

bne $s4, $zero, IntLoop1

addi $v0,$zero,10 # system call for exit

lw $s2, 0($sp)

addiu $sp, $sp,4

mtc0 $s2,$0#恢复EPC

lw $s6,0($sp)

addiu $sp,$sp,4

lw $s5,0($sp)

addiu $sp,$sp,4

lw $s4,0($sp)

addiu $sp,$sp,4

lw $s3,0($sp)

addiu $sp,$sp,4

lw $s0,0($sp)

addiu $sp,$sp,4

lw $a0,0($sp)

addiu $sp,$sp,4

lw $v0,0($sp)

addiu $sp,$sp,4

eret # 中断返回

依次按下1号键、2号键、3号键，程序依次进入中断处理程序1、2、3，当中断处理程序3执行完毕后，再次按下3号键，此时中断处理程序2会被中断处理程序3中断。多级中断的执行状态如图 4‑4所示：

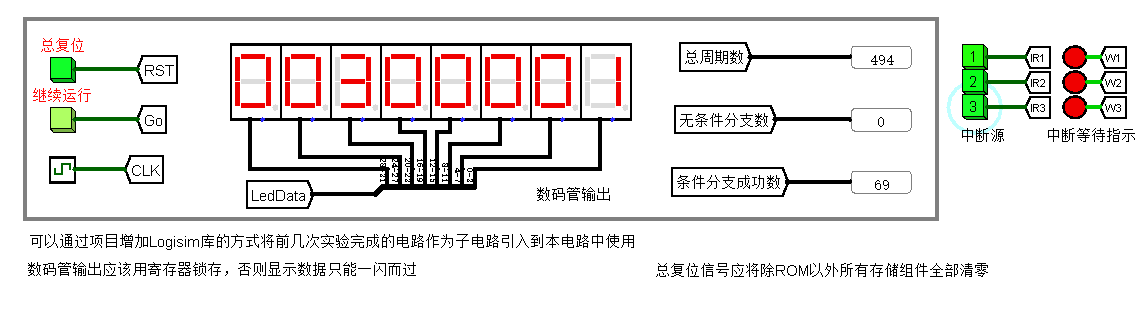


图 4‑4 多级中断测试

## 性能分析

单周期CPU的时钟周期数最短，是1546，但是时钟频率较低，经过验证可以上板的最高频率是25MHz。为了提高时钟频率，设计了气泡流水线CPU，由于数据冲突需要插入气泡，时钟周期数为3624。为了缩短时钟周期，设计了重定向流水线CPU，在保证了较高的时钟频率下时钟周期数缩短为2298。然后在重定向流水线CPU的基础上增加了动态分支预测部件，使时钟周期数缩短为1762。

## 主要故障与调试

### 数码管显示错误

单周期CPU上板： 数码管显示问题。

**故障现象：**数码管显示乱码

**原因分析：** 通过查阅Nexysddr的文档，如图图 4‑5，数码管的显示和选择均是低电平有效，在之前设计时默认了高电平有效，导致了显示乱码。

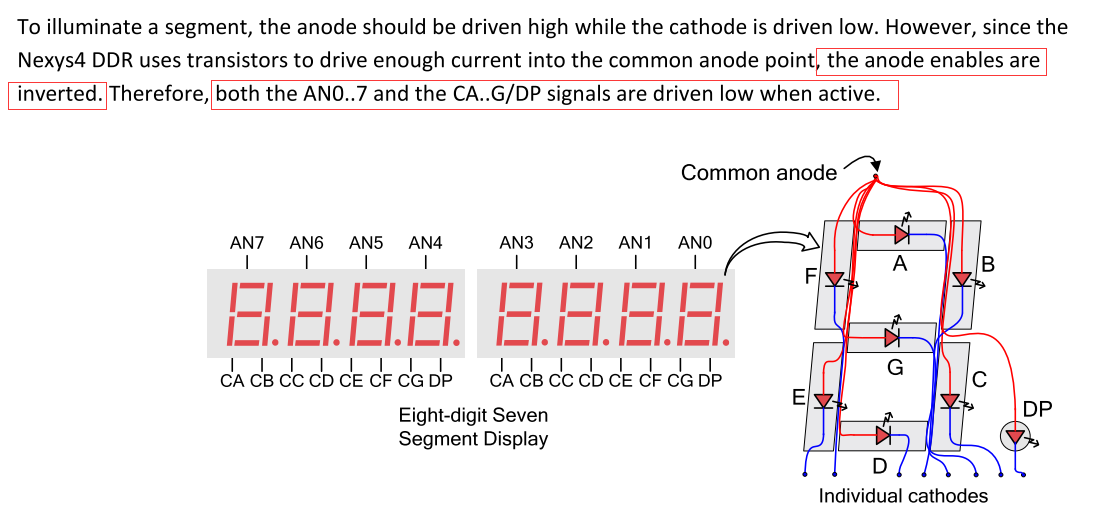


图 4‑5 Nexysddr文档

**解决方案：**对An和Seg的驱动取反。以An[0]和Seg[0]为例，修改后的代码为：

//Pattern模块

4'b0000:Patt=8'b11000000;//Seg[0]

//Decoder3\_8模块

3'b000:Sel=~8'b00000001;//An[0]

### Rom模块错误

单周期CPU上板：Rom模块取出的数据延时一个周期。

**故障现象：**如图 4‑6所示， 在输入地址是1的时候取出的是地址为0的指令，在输入地址是2的时候取出的是地址为1的指令。指令被延时了一个周期才取出。

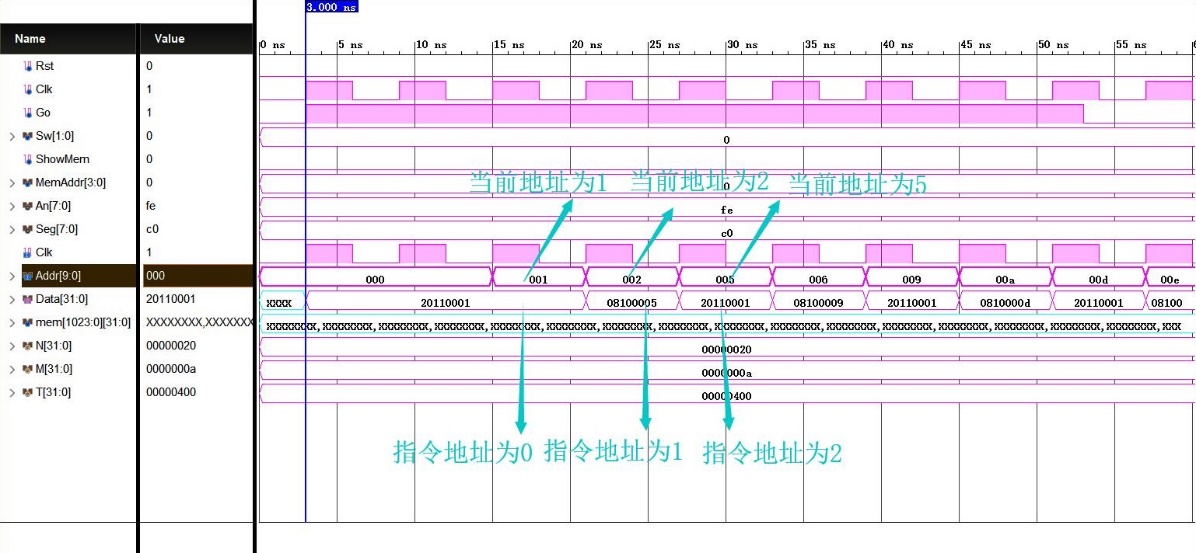


图 4‑6 Rom模块程序故障图

**原因分析：**在使用verilog实现Rom模块时，采用了always和clk触发的方式，这将导致数据被延时一个周期取出。错误的代码如下：

always @(posedge Clk)

begin

Data=mem[Addr];

end

**解决方案：**采用assign语句实现Rom，修改后的代码如下：

assign Data=mem[Addr];

### 连线错误

单周期上板：主模块连线错误。

**故障现象：**如图 4‑7，无条件跳转指令的目标地址是有26位有符号数拓展得到的，在连接模块时将26位有符号数错写成了16位有符号数，导致了无条件跳转指令错误。

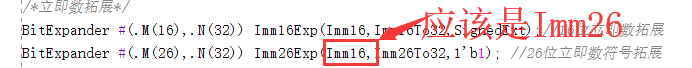


图 4‑7 连线故障

**解决方案：**将Imm16修改为Imm26即可。

### 重定向错误

重定向流水线(logism)：重定向数据错误。

**故障现象：**程序运行错误。

**原因分析：** 如图 4‑8所示，在执行到某一条指令时应该重定向WB.AluR的数据，但是却重定向了MEM.AluR的数据，猜测选择信号错误。复查重定向检测模块，发现检测发生在ID阶段，而执行发生在EX阶段，延后了一个周期。

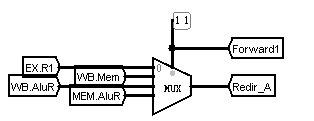


图 4‑8 重定向故障

**解决方案：**将重定向信号经过流水由ID阶段传输到EX阶段使用，修改后的电路图如图 4‑9所示。

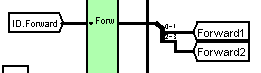


图 4‑9 重定向故障解决方案

### 动态分支预测错误

动态分支预测(logism)：数据无法命中。

**故障现象：**写入到BHT中的数据无法命中。

**原因分析：** 如图 4‑8所示，由于PC寄存器的宽度为10位，因此在比较PC值时只需比较低10位，而不应该比较32位。

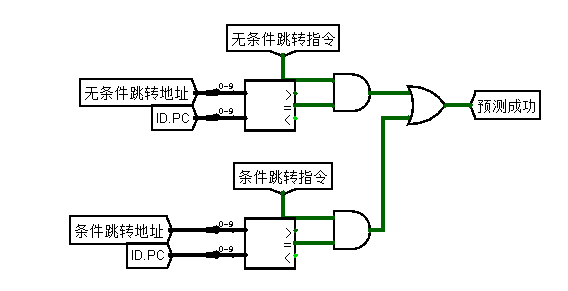


图 4‑8 重定向故障

**解决方案：**如图 4‑8所示，将比较器的位数修改为10位，只比较低10位的数据即可。

## 实验进度

表 4‑1 课程设计进度表

| 时间 | 进度 |
| --- | --- |
| 第一天 | 阅读课设任务书，阅读MIPS指令手册，开会商议了24条指令上板方案设计，进行了模块分工以及基本原件的verilog代码封装。 |
| 第二天 | 完成基本模块的数据通路的联通，调试数据通路存在的bug，修改通路消除warning使得数据通路更加健壮。 |
| 第三天 | 完成24条指令单周期CPU上板，在此基础上修改数据通路完成28条指令CPU上板。 |
| 第四天 | 查阅流水线的相关知识，在logism平台上完成了理想流水线数据通路的构建，并通过测试代码。 |
| 第五天 | 查阅分支流水线的相关知识，在logism平台上完成了分支流水线数据通路的构建，并通过测试代码。 |
| 第六天 | 查阅气泡流水线的相关知识，在logism平台上完成了气泡流水线数据通路的构建，并通过测试代码。 |
| 第七天 | 查阅动态分支预测的相关知识，在logism平台上完成了动态分支通路的构建。 |
| 第八天 | 调试动态分支预测电路，并通过测试代码。 |
| 第九天 | 查阅单级中断的相关知识，在logism平台上完成了气泡流水线数据通路的构建，并通过测试代码。 |
| 第十天 | 查阅多级中断和流水中断的相关知识，在logism平台上完成了多级中断和流水中断的数据通路，并通过测试代码。 |

# 设计总结与心得

## 课设总结

本次课程设计从单周期MIPS CPU开始，采取循序渐进的方式实现了从单周期CPU到5段流水CPU的实现，在这个过程中具体做了如下的工作：：

1. 实现了Logism平台的单周期 CPU、理想流水线 CPU、气泡流水线 CPU、重定向流水线 CPU、支持单级中断和多级中断的单周期CPU、支持分支预测的重定向流水线 CPU。
2. 设计并完成了单周期CPU的上板验证。
3. 实现了支持24条基本指令和4条CCMB拓展指令的5段流水线CPU。

## 课设心得

组成原理课程设计是所有课程设计中较难的也是最有意思的课程设计，回想起来，我用了10天的时间，完成了从单周期CPU到5段流水线CPU的实现，让我颇有成就感，在实验的过程中我也学到了许多东西，也极大的增强了我的团队协作能力。

课程设计的第一个任务是在logism平台上搭建28条单周期CPU，此任务是在寒假布置的，时间充足，再加上上学期的实验中已经实现过单周期的CPU，因此实现过程较为顺利。紧接着，单周期上板是小组合作完成，在开始实现之前我们进行了组内讨论，规划设计方案，进行任务分配，使用驼峰命名法的编程风格，并对子模块的接口进行了设计。在每个人实现并测试了负责的子模块之后，均交给组内的另外一个人再次进行测试，以保证子模块的正确性。总体来说，组内成员每个人各司其职，相互配合，花费了3天完成了单周期CPU的上板。接下来是流水CPU，结合Mooc的实验介绍实现中也没有遇到特别的困难。其次是动态分支预测功能，由于主要模块BHT可以直接利用上学期实验中的Cache模块，因此实现过程较为顺利。最后是中断功能，一开始测试程序忘记添加保存寄存器的代码，导致了程序运行错误，修改之后通过了测试。

对于本次实验，我有两点建议：第一，实验可以和汇编课程结合起来，让同学们编写自己的程序，在自己设计的CPU上运行，即可加深对两个课程的理解，也可以增强同学们的成就感。第二，在实现单周期CPU上板时组员相互协作很紧密，但是在后续的实验中组员的交流越来越少，希望可以增加后续实验的协作性，提升团队协作能力。

# 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:** |