***2022***



**硬件综合训练 课程设计报告**

|  |  |
| --- | --- |
| 题 目： | 5段流水CPU设计 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS2002 |
| 学 号： | U202015342 |
| 姓 名： | 徐子路 |
| 电 话： | 18502334612 |
| 邮 件： | [18502334612@qq.com](mailto:18502334612@qq.com) |

目 录

[1 课程设计概述 3](#_Toc117158245)

[1.1 课设目的 3](#_Toc117158246)

[1.2 设计任务 3](#_Toc117158247)

[1.3 设计要求 3](#_Toc117158248)

[1.4 技术指标 4](#_Toc117158249)

[2 总体方案设计 6](#_Toc117158250)

[2.1 单周期CPU设计 6](#_Toc117158251)

[2.2 中断机制设计 10](#_Toc117158252)

[2.3 流水CPU设计 11](#_Toc117158253)

[2.4 气泡式流水线设计 12](#_Toc117158254)

[2.5 重定向流水线设计 13](#_Toc117158255)

[3 详细设计与实现 14](#_Toc117158256)

[3.1 单周期CPU 实现 14](#_Toc117158257)

[3.2 中断机制实现 21](#_Toc117158258)

[3.3 流水CPU实现 26](#_Toc117158259)

[3.4 气泡式流水线实现 27](#_Toc117158260)

[3.5 重定向流水线实现 28](#_Toc117158261)

[4 实验过程与调试 30](#_Toc117158262)

[4.1 测试用例和功能测试 30](#_Toc117158263)

[4.2 性能分析 31](#_Toc117158264)

[4.3 主要故障与调试 32](#_Toc117158265)

[4.4 实验进度 34](#_Toc117158266)

[5 设计总结与心得 35](#_Toc117158267)

[5.1 课设总结 35](#_Toc117158268)

[5.2 课设心得 35](#_Toc117158269)

[参考文献 37](#_Toc117158270)

# 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 指令集前24条基本32位RISC\_V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** | **指令类型** | **简单功能描述** | **备注** |
| --- | --- | --- | --- | --- |
| 1 | ADD | R | 加法 | 指令格式参考RISC-V32指令集，最终功能以MARS模拟器为准。 |
| 2 | ADDI | I | 立即数加 |
| 3 | AND | R | 与 |
| 4 | ANDI | I | 立即数与 |
| 5 | SLLI | I | 逻辑左移 |
| 6 | SRAI | I | 算数右移 |
| 7 | SRLI | I | 逻辑右移 |
| 8 | SUB | R | 减 |
| 9 | OR | R | 或 |
| 10 | ORI | I | 立即数或 |
| 11 | XORI | I | 或非/立即数或非 |
| 12 | LW | I | 加载字 |
| 13 | SW | S | 存字 |
| 14 | BEQ | B | 相等跳转 |
| 15 | BNE | B | 不相等跳转 |
| 16 | SLT | R | 小于置数 |
| 17 | SLTI | I | 小于立即数置数 |
| 18 | SLTU | R | 小于无符号数置数 |
| 19 | JAL | J | 转移并链接 |
| 20 | JALR | J | 转移到指定寄存器 |
| 21 | ECALL | I | 系统调用 | if ($a7==34) LED 输出$a0 的值 else 等待 Go 按键暂停 |
| 22 | CSRRSI | I | 访问CSR寄存器 | 中断相关，可简化为开中断 |
| 23 | CSRRCI | I | 访问CSR寄存器 | 中断相关，可简化为关中断 |
| 24 | URET | I | 中断返回 | 清中断，mEPC 送 PC，开中断 |
| 28 | AUIPC | U | PC和立即数之和置数 |  |
| 29 | SLTIU | I | 小于无符号立即苏置数 |
| 30 | LB | I | 加载字节 |
| 31 | BGE | B | 大于等于跳转 |

# 总体方案设计

## 单周期CPU设计

本次我们采用的方案是微程序控制，且主、控存分开的方案，即采用微程序控制方式，实现主存储器（MM）和微程序控制存储器（CM）不共用一个存储器的方式完成方案的哈佛结构设计。在实施过程中，主要用logisim平台完成开发。在logisim上完成连线并验证仿真后，利用verilog语言重新描述，在vivado平台正确仿真后完成FPGA上板。

总体结构图如图2.1 总体结构图所示。

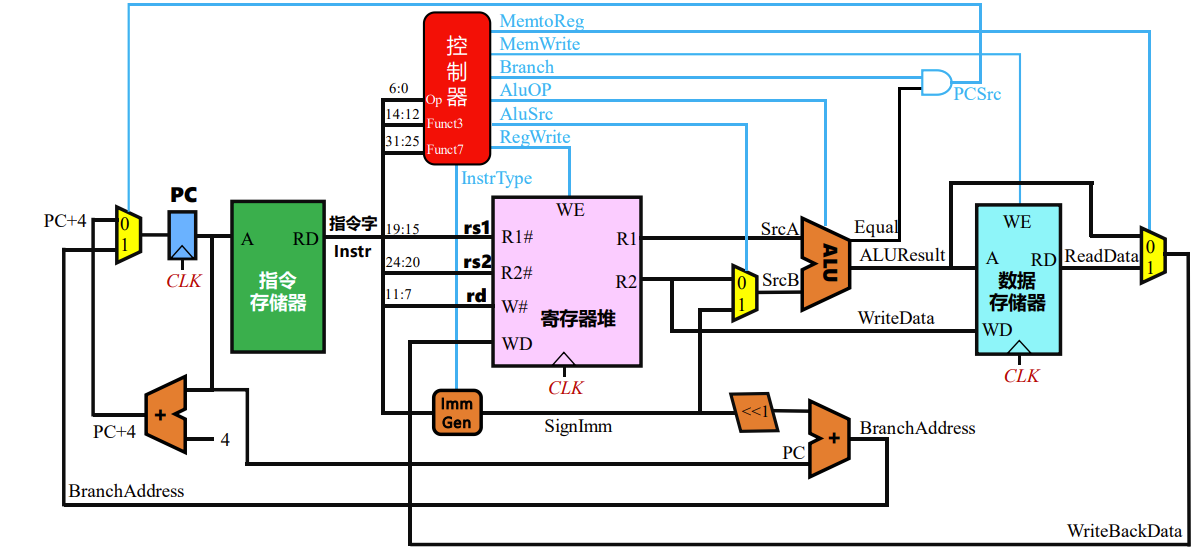


图2.1 总体结构图

### 主要功能部件

#### 程序计数器PC

程序计数器PC部分，具体设计思路如下。 用一个32位寄存器存储PC的值，每一个时钟周期将新的PC的值传入程序计数器。在顺序执行方式下，每一个时钟周期内 CPU 取指令后将 PC 寄存器的值加 4， 形成下一条指令的地址。顺序执行方式下的取指令数据通路如图2.2所示。若有跳转指令，则将需要跳转的地址传入程序计数器。因此使用一个二路选择器，0号和1号端口分别传入PC+4的值和需要跳转的地址，用一个控制信号选择传入PC寄存器输入端口的值。取指令数据通路如图2.2 取指令数据通路所示。

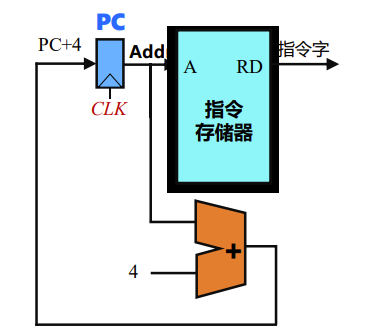


图2.2 取指令数据通路

#### 指令存储器IM

指令存储器IM部分，具体实现思路如下。指令存储器IM用一个只读存储器ROM。ROM有一个地址输入端口和数据输出端口，前者接入当前时钟周期程序计数器输出结果的2到11位，后者为当前周期的指令字IR.

#### 运算器

运算器部分，在本次实验中是已经给出且封装好了的一个组件ALU。其相关引脚与功能描述如表2.1 算术逻辑运算单元引脚与功能描述所示。

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2.2 算术逻辑运算单元规格 |
| 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 算术逻辑运算单元规格

| **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 |
| 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 |

#### 寄存器堆RF

寄存器堆RF部分，在本次实验中是已经给出且封装好了的一个组件RegFile。其相关引脚和功能描述如表2.3 寄存器堆引脚与功能描述。

表2.3 寄存器堆引脚与功能描述

| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| R1# | 输入 | 5 | 需要读取的寄存器rs1的对应地址 |
| R2# | 输入 | 5 | 需要读取的寄存器rs2的对应地址 |
| W# | 输入 | 5 | 需要写入的寄存器W的对应地址 |
| Din | 输入 | 32 | 需要写入的地址 |
| WE | 输入 | 32 | 写入使能 |
| CLK | 输入 | 1 | 时钟信号 |
| R1 | 输出 | 32 | 寄存器rs1的输出 |
| R2 | 输出 | 32 | 寄存器rs2的输出 |

### 数据通路的设计

数据通路部分，采用简单迭代法设计。依次完成各个类型指令的数据通路，不断迭代以支持新的指令，直至所有指令都能正常运行。

### 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 主控制器控制信号的作用说明所示。

表2.4 主控制器控制信号的作用说明

| **控制信号** | **说明** |
| --- | --- |
| ALU\_OP | 控制ALU的功能，取值及功能见表2.2 算术逻辑运算单元规格 |
| MemToReg | 取值为1时控制数据存储器Ram的输出写入寄存器组RegFile |
| MemWrite | 取值为1时使能外部输入MDin写入数据存储器Ram |
| ALU\_SRCB | 取值为1时选择I型指令的立即数作为ALU的B输入  取值为0时选择寄存器堆的R2输出作为ALU的B输入 |
| RegWrite | 取值为1时使能外部输入RDin写入寄存器堆RF |
| ecall | 取值为1时，如果寄存器堆RF的a7（rs为17）寄存器值为34，控制LED显示RF的a0（rs为10）寄存器值。  取值为0时，暂停电路直至Go按键 |
| S\_type | 取值为1时，选择U型指令的立即数  取值为0时，选择I型指令的立即数 |
| JALR | JALR指令译码信号，执行JALR指令 |
| JAL | JAL指令译码信号，执行JAL指令，选择寄存器写回编号，写回值 |
| 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 |

## 中断机制设计

### 总体设计

中断程序主要分成单级中断部分和多级中断部分。

单级中断部分相对简单。单级中断不需要开关中断，当中断信号来临时，将当前周期程序计数器PC的输出保存到mEPC中，同时选择对应中断的程序入口作为新的PC，在uret信号出现前不接受新的中断信号，出现uret信号时将mEPC记录的PC传入程序计数器之中。

多级中断相对更加复杂，当中断信号来临时，需要关闭中断，保护现场且将pc值压栈。然后开中断，执行中断程序，此时程序可被更高优先级的中断信号中断。然后关中断，恢复现场，然后开中断。最后中断返回，回到之前记录的断点，执行主程序（或者上一个被打断的中断程序）。

### 硬件设计

采用硬件设计时，多级中断的相应优先级使用优先编码器实现，同一时刻优先级最高的中断请求被 CPU 响应。只有没有更高优先级的中断请求时，电路才会把较低优先级的中断请求 送给 CPU。

多级中断中有三个mEPC寄存器分别保存三种等级中断的pc，用复杂的中断逻辑控制中断状态的转移。硬件设计下CSRRSI和CSRRCI指令分别对应开中断和关中断。

## 流水CPU设计

### 总体设计

将单周期数据通路改造成流水线架构，需要在指令执行的不同阶段加入流水寄存器。流水寄存器用于锁存前段加工处理完毕的数据和控制信号，通常 这些数据和控制信号都会横穿流水寄存器传递到下一段。总共增加了四个流水寄存器，根据其所连接的功能段的名称分别命名为 IF/ID、ID/EX、EX/MEM、MEM/WB，数据通路被被 4个流水寄存器细分为五段流水线。

所有流水寄存器，程序计数器 PC、寄存器堆、数据存储器均采用统一时钟 CLK 进行同步，每来一个时钟，就会有一条新的指令进入流水线取指令 IF 段，同时流水寄存器就会锁存前段加工处理完成的数据和控制信号，为下一段的功能部件提供数据输入，指令流水线各功能段通过流水寄存器完成一次数据传送。

流水CPU的总体结构图如图2.3 流水CPU总体结构图所示。

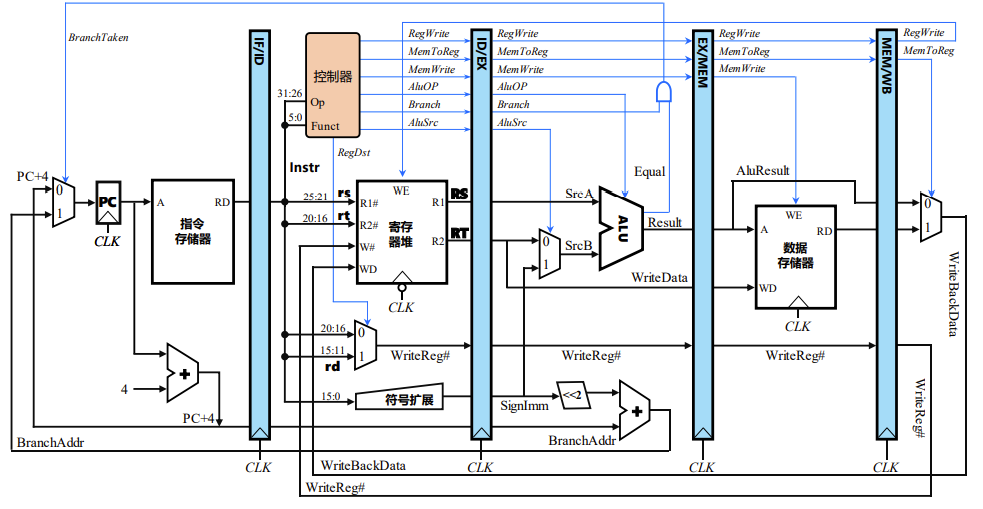


图2.3 流水CPU总体结构图

### 流水接口部件设计

五段流水寄存器共需要4个流水寄存器，每个流水寄存器是作为两个阶段的中转站出现的，因此需要寄存的数据和对应两个阶段需要实现的功能有关。对于一个流水寄存器，其中每个数据都是采用“输入+二路选择器+寄存器+输出”的模式实现的。其中二路选择器的0号输入为数据输入，1号输入为0输入（接地），控制信号为RST信号（实现同步置零）。

### 理想流水线设计

理想流水器由于不用考虑数据冲突的问题，只需要把不同阶段的指令用流水寄存器隔开。由于我在设计时统一在EX段进行指令跳转，因此与指令跳转有关的信号都是使用的ID/EX流水寄存器的输出信号。此外，RegFile的输入需要使用WB阶段的RDin和Rd信号，其他的信号和单周期CPU大致相同。

## 气泡式流水线设计

实验中流水线CPU主要存在控制冲突和数据冲突两种冲突。为了解决控制冲突，在执行程序分支跳转时必须清除流水线中的分支指令后续的若干条误取指令（添加CLR信号）。为了解决数据冲突需要寄存器堆写入控制采用下跳沿触发，而所有流水寄存器采用上跳沿触发，且在检测到数据冲突时阻塞 IF、ID 段指令的执行（添加EN信号），并尝试在时钟到来时在EX段插入气泡。

## 重定向流水线设计

重定向流水线解决控制冲突时和气泡流水线解决控制冲突的方式如出一辙，而在解决数据冲突时使用了性能更优的方法。在ID段时检查当前指令使用的数据是否和EX和MEM段有冲突，若有则将这一状态记录下来用寄存器传到EX段。到下一个时钟周期，所有阶段向后顺移，EX段出现数据冲突，此时直接将正确的操作数从其所在位置重定向到EX段合适的位置参与运算来避免数据冲突（存在Load-Use 相关时仍然需要插入气泡）。

使用重定向流水线后插入气泡的个数明显减少，性能也比气泡流水线更优。

# 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，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 程序计数器（PC）所示。

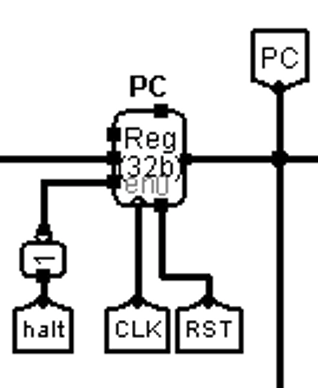


图3.1 程序计数器（PC）

1. FPGA实现：

FPGA中使用一个Register模块实现程序计数器。实现代码如下，其中ActiveLevel是上升沿和下降沿控制。

assign Q = (ActiveLevel) ? s\_state\_reg : s\_state\_reg\_neg\_edge;

    initial begin

       s\_state\_reg <= 0;

       s\_state\_reg\_neg\_edge <= 0;

    end

    always @(posedge Clock or posedge Reset)

    begin

       if (Reset) s\_state\_reg <= 0;

       else if (ClockEnable) s\_state\_reg <= D;

    end

    always @(negedge Clock or posedge Reset)

    begin

       if (Reset) s\_state\_reg\_neg\_edge <= 0;

       else if (ClockEnable) s\_state\_reg\_neg\_edge <= D;

    end

在调用模块时的代码如下，其中wire\_mux21\_out为输入的数据，wire\_PC\_reg\_out为输出的数据。

Register #(.DATA\_WIDTH(32)) PC\_Reg(wire\_Clk\_N,~wire\_halt,wire\_mux21\_out,RST,wire\_PC\_reg\_out);

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 指令存储器（IM）所示。

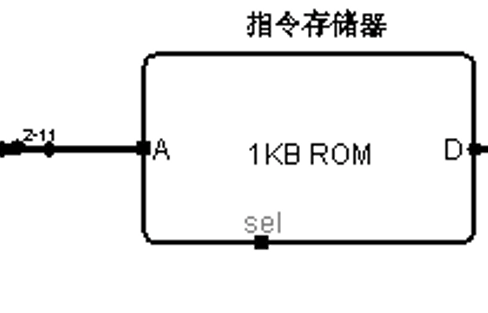


图3.2 指令存储器（IM）

1. FPGA实现：

FPGA中使用一个Rom模块实现指令存储器。选择Rom的数据位宽为32位，因为该ROM的地址位宽为10位，所以选择Rom的大小选择为1024。通过readmemh函数读取指令的benchmark文件，可以完成Rom的初始化。

指令存储器IM的Verilog代码如下：

 reg [31:0] mem [1023:0];

    initial begin

       $readmemh("D:\\pro\\principles\_of\_computer\_composition\\cpu24-riscv\\Single\_Cycle\_RISCV2\\single\_cycle\_cpu\\single\_cycle\_cpu.srcs\\sources\_1\\imports\\verilog\\memory\\ccab.txt",mem);

    end

    always @ (Address)

    begin

      Data <= mem[Address];

    end

直接调用之前设置的ROM作为指令存储器，输入为指令地址的2-11位，输出为该指令。

### 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 指令系统数据通路表所示。

表3.1 指令系统数据通路表

| 指令 | PC | RF | | | | ALU | | | DM | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1# | R2# | W# | Din | A | B | OP | Addr | Din |
| add | pc+4 | rs1 | rs2 | rd | ALU | R1 | R2 | 5 | - | - |
| sub | pc+4 | rs1 | rs2 | rd | ALU | R1 | R2 | 6 | - | - |
| and | pc+4 | rs1 | rs2 | rd | ALU | R1 | R2 | 7 | - | - |
| or | pc+4 | rs1 | rs2 | rd | ALU | R1 | R2 | 8 | - | - |
| slt | pc+4 | rs1 | rs2 | rd | ALU | R1 | R2 | 11 | - | - |
| sltu | pc+4 | rs1 | rs2 | rd | ALU | R1 | R2 | 12 | - | - |
| addi | pc+4 | rs1 | - | rd | ALU | R1 | imm | 5 | - | - |
| andi | pc+4 | rs1 | - | rd | ALU | R1 | imm | 7 | - | - |
| ori | pc+4 | rs1 | - | rd | ALU | R1 | imm | 8 | - | - |
| xori | pc+4 | rs1 | - | rd | ALU | R1 | imm | 9 | - | - |
| slti | pc+4 | rs1 | - | rd | ALU | R1 | imm | 11 | - | - |
| slli | pc+4 | rs1 | - | rd | ALU | R1 | imm | 0 | - | - |
| srli | pc+4 | rs1 | - | rd | ALU | R1 | imm | 2 | - | - |
| srai | pc+4 | rs1 | - | rd | ALU | R1 | imm | 1 | - | - |
| lw | pc+4 | rs1 | - | rd | Mem | R1 | imm | 5 | ALU[11:2] | - |
| sw | pc+4 | rs1 | rs2 | - | - | R1 | imm | 5 | ALU[11:2] | R2 |
| ecall | pc+4 | 17 | 10 | - | - | - | - | - | - | - |
| beq | pc+imm | rs1 | rs2 | - | - | - | - | - | - | - |
| bne | pc+imm | rs1 | rs2 | - | - | - | - | - | - | - |
| jal | pc+imm | - | - | rd | pc+4 | - | - | - | - | - |
| jalr | R1+imm&-1 | rs1 | - | rd | pc+4 | - | - | - | - | - |
| auipc | pc+4 | - | - | rd | pc+imm<<12 | - | - | - | - | - |
| sltiu | pc+4 | rs1 | - | rd | ALU | R1 | imm | 12 | - | - |
| lb | pc+4 | rs1 | - | rd | Mem | R1 | imm | 5 | ALU[11:2] | - |
| bge | pc+imm | rs1 | rs2 | - | - | - | - | - | - | - |

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。单周期CPU数据通路如图3.3 单周期CPU数据通路所示。

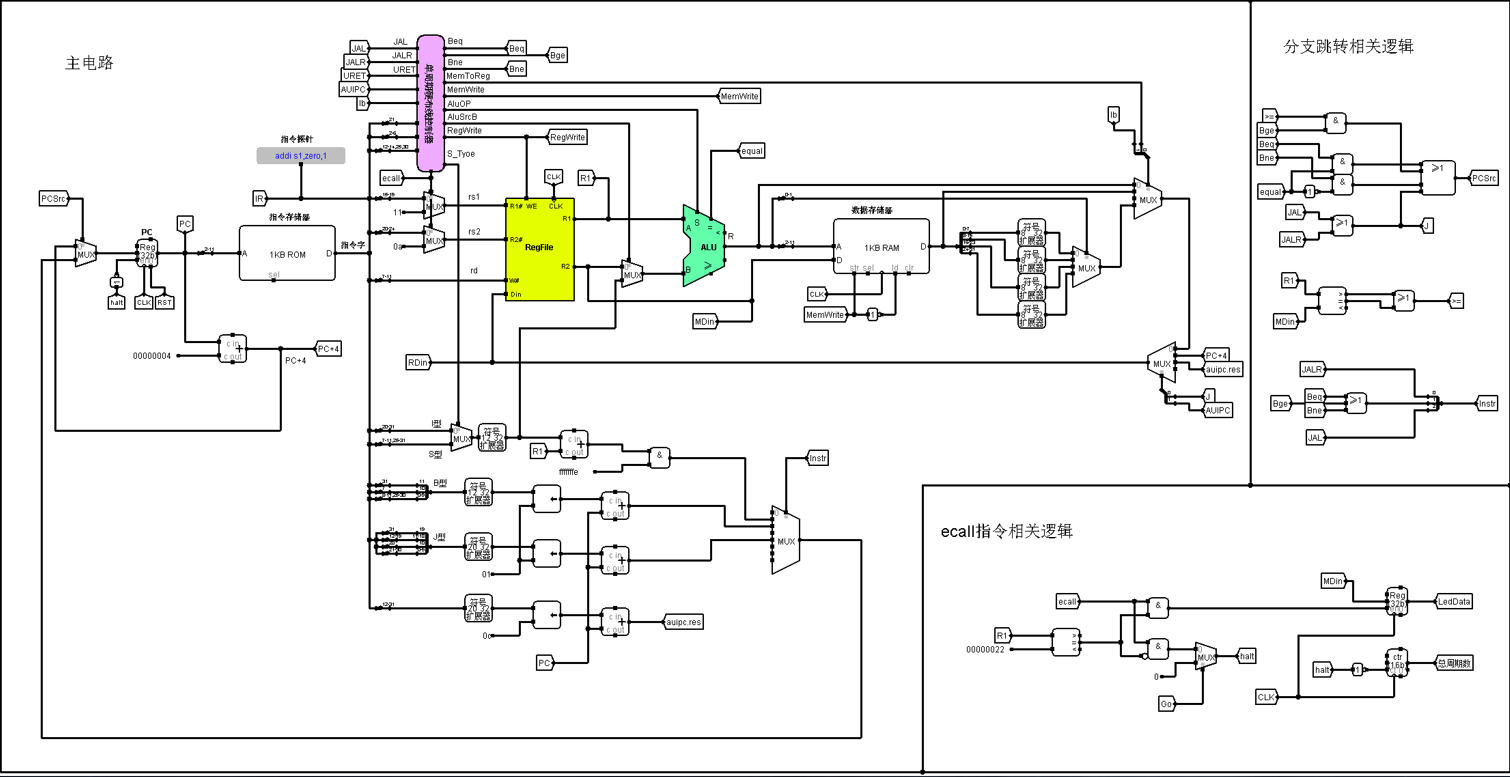


图3.3 单周期CPU数据通路（logisim）

在Vivado中使用Verilog语言搭建的数据通路的原理图如图3.4 单周期CPU数据通路（FPGA）所示。

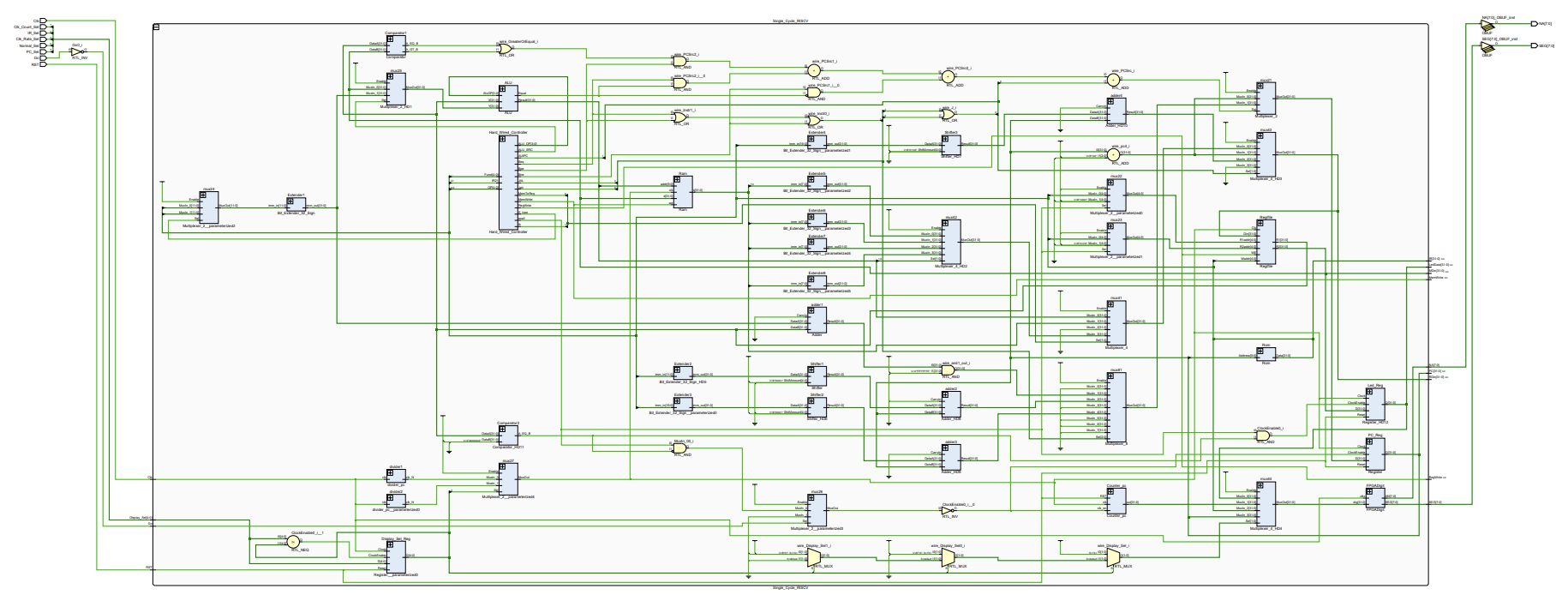


图3.4 单周期CPU数据通路（FPGA）

### 控制器的实现

硬布线控制器通过填写RISC-V单周期硬布线控制器表达式自动生成表.excel得到每个信号相应的逻辑。在logisim平台中可以用“分析组合逻辑电路”功能直接得到运算器自动生成和控制信号自动生成图。在FPGA平台用verilog编写时也可以将excel表中各个信号生成的逻辑稍作修改（所有“加”改成“或”）直接使用。

真值表如图3.5 单周期硬布线控制器表达式真值表所示。

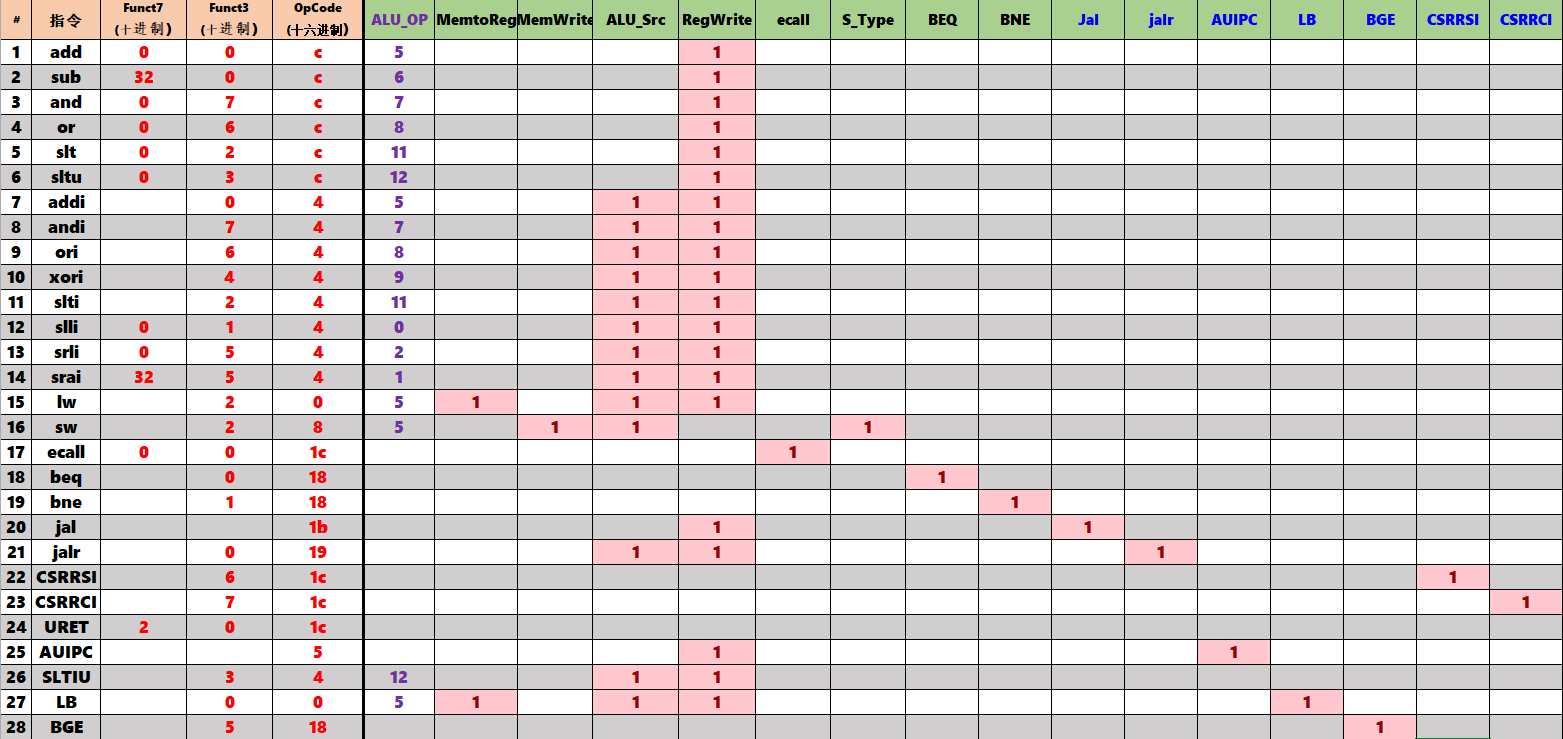


图3.5 单周期硬布线控制器表达式真值表

1. FPGA实现

根据在excel表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。此处仅示例控制信号自动生成模块中部分信号的赋值代码。

assign MemToReg = (~F14& F13&~F12&~OP6&~OP5&~OP4&~OP3&~OP2)|(~F14&~F13&~F12&~OP6&~OP5&~OP4&~OP3&~OP2);

    assign MemWrite = ~F14& F13&~F12&~OP6& OP5&~OP4&~OP3&~OP2;

    assign AUIPC = ~OP6&~OP5& OP4&~OP3& OP2;

    assign LB = ~F14&~F13&~F12&~OP6&~OP5&~OP4&~OP3&~OP2;

    assign BGE =  F14&~F13& F12& OP6& OP5&~OP4&~OP3&~OP2;

…

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3.6 主控制器原理图所示。

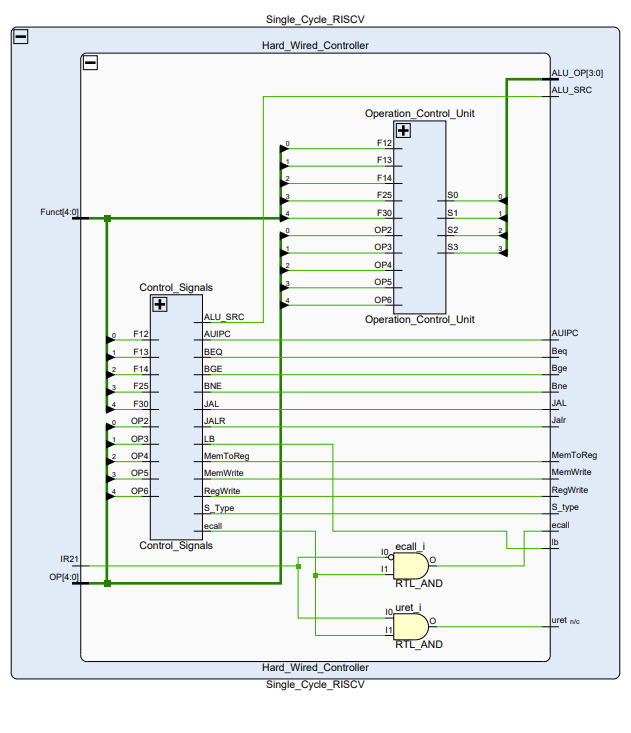


图3.6 主控制器原理图

## 中断机制实现

### 单级中断实现

单机中断的实现过程大致按照任务书中单级中断部分的实验步骤进行。

1.增加中断按键信号采样电路

中断按键信号采样电路在任务书中给了实现的例子，在本次实验中由于存在3种级别的中断，因此需要3个中断按键信号采样电路，分别采样1，2，3级的中断信号，并将中断请求状态反映到中断指示灯上。如图3.7 中断按键信号采样电路所示。

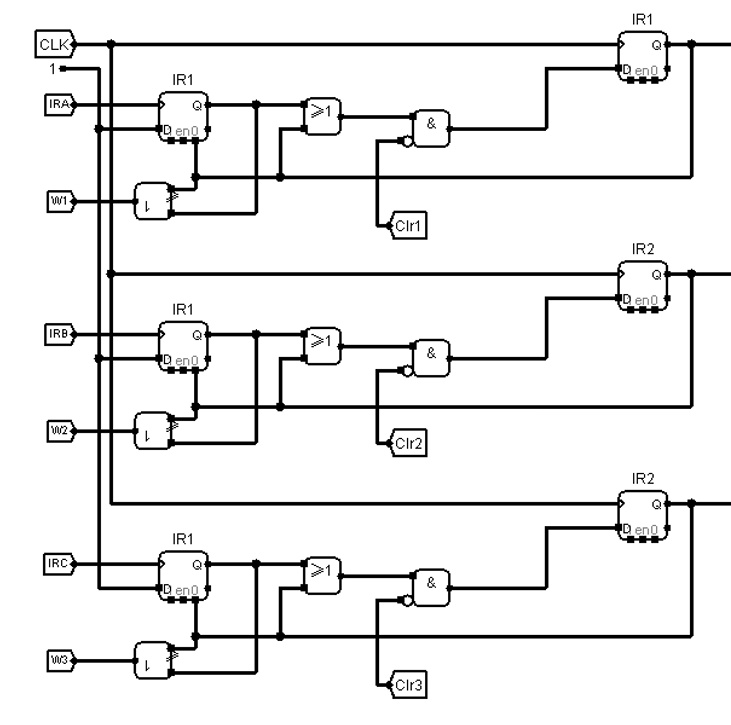


图3.7 中断按键信号采样电路

2.设计中断识别逻辑

根据任务书上的提示，我在实验中使用优先编码器实现，详细的做法是将3个中断采样电路输出的中断请求作为输入 送至优先编码器，让优先编码器输出优先度最高的信号。优先编码器有两个输出，一个是输入中优先度最高的信号的端口值，一个记录优先编码器输出中是否有高电平信号。前者作为中断入口信号的选择信号以及中断清楚信号的选择信号，后者即Intout作为中断使能信号的组成部分。电路图如图3.8 中断识别逻辑电路所示。

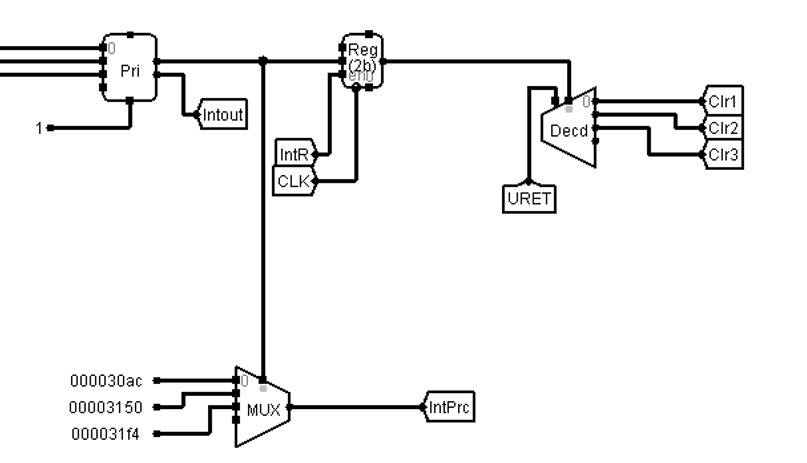


图3.8 中断识别逻辑电路

3.设计中断相关寄存器即相关数据通路

这部分思路相对比较前面的部分更复杂。设计需求是需要Intout信号来到时，将电路中正在执行的指令的下一条指令存到一个寄存器mEPC中，而到了下一个周期该寄存器会被锁存起来直到出现URET信号。我的实现思路是使用3个寄存器，第一个是中断使能寄存器，用来保存中断的使能信号；第二个是mEPC用于保存返回的指令，最后一个寄存器是生成中断使能信号的部件。设计图如下所示，在这种设计下，最开始IntRe信号为1，不会影响到中断信号Intout的传入。当Intout为1时，IntR为1，mEPC接受CLK信号，将下一条指令存入mEPC中；与此同时，第三个寄存器被强制置为1，IntRe信号归0，因此中断使能器之后的输入一直为0.直到URET信号来临前，mEPC中的值被锁存。电路如图3.9 中断使能寄存器数据通路所示。

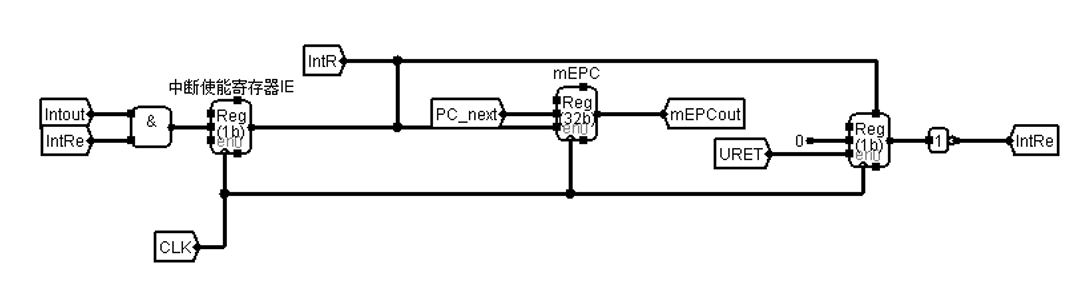


图3.9 中断使能寄存器数据通路

### 多级中断实现

多级中断相对单级中断，需要实现高优先级中断正确打断低优先级中断，实现起来也会变得复杂很多。

我实现多级中断思路主要是四个问题：如何判断进入了中断程序？如何判断新的中断请求等级和已经进入的中断程序等级大小？中断程序结束时如何返回？如何实现开关中断？

为了解决这三个问题，我将多级中断相关逻辑分成了中断按键采样逻辑、中断使能控制逻辑、中断状态逻辑、中断出口逻辑和开关中断逻辑几个部分。这些逻辑不具有严格的先后关系，而是彼此联系成一个整体。

1.中断按键采样逻辑

中断按键采样逻辑和单级中断完全一致，这里不多做赘述。电路如图3.10 中断按键采样逻辑电路所示。

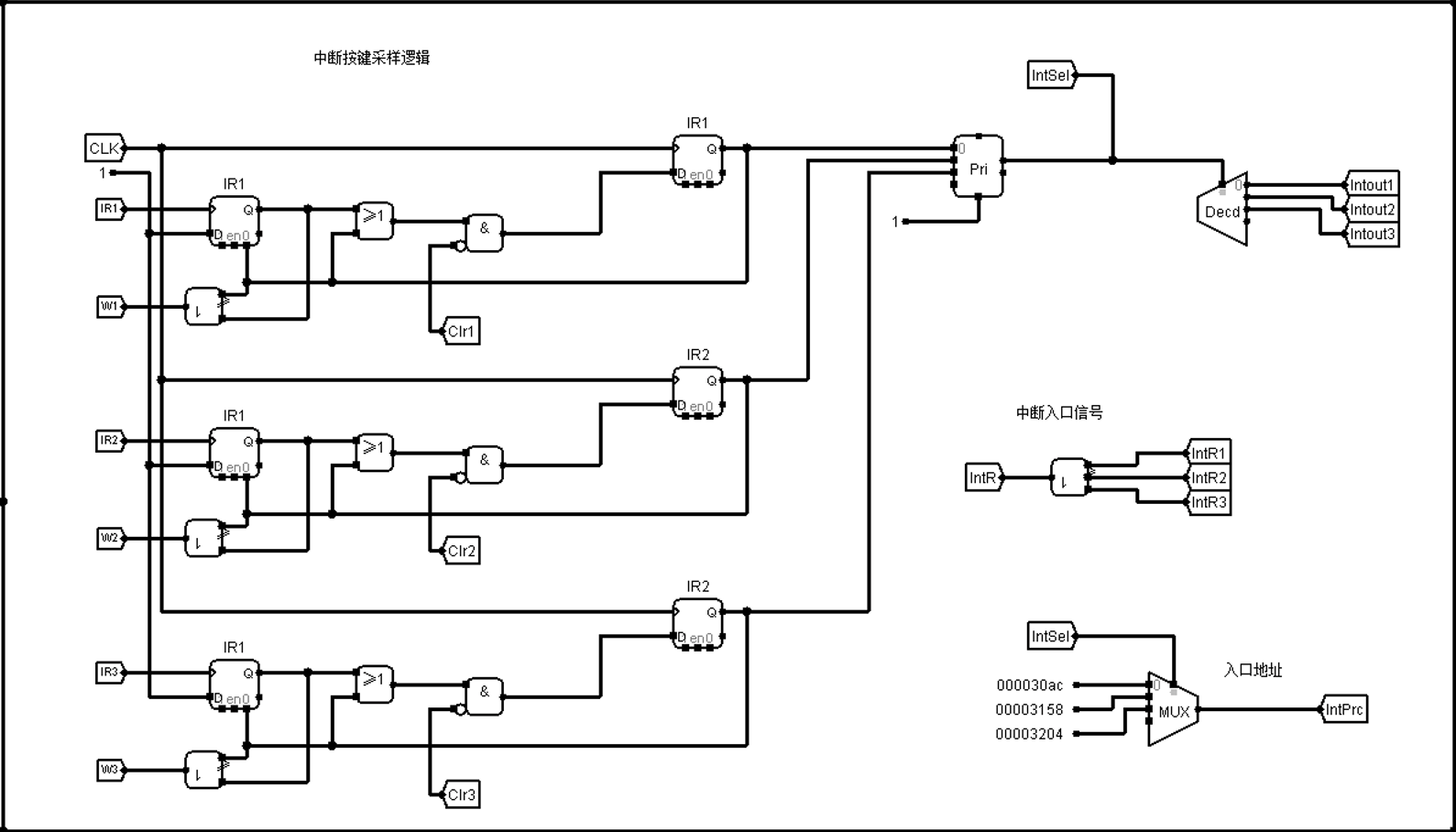


图3.10 中断按键采样逻辑电路

2.中断使能控制逻辑

中断使能控制逻辑同样和单周期类似，不同之处在于3个中断信号分别对应一组中断使能逻辑。IntSuper信号取值为1时表示当前程序可以被输入的中断信号打断，当程序目前不处于中断状态(IntStatus为0)，或者程序处于中断状态（IntStatus为1）、开中断且输入的中断比当前中断等级高（IntSel > IntCur）时取值为1。这样就解决了如何判断新的中断请求等级和已经进入的中断程序等级大小的问题。大体电路如图3.11 中断使能控制逻辑电路所示。

其中IntStatus、IntCur信号来源于中断状态逻辑，IntSel信号来源于中断按键采样逻辑，开中断信号来源于开关中断逻辑。电路如图3.11 中断使能控制逻辑电路所示。IntSuper信号的生成逻辑如图3.12 IntSuper信号生成逻辑所示。

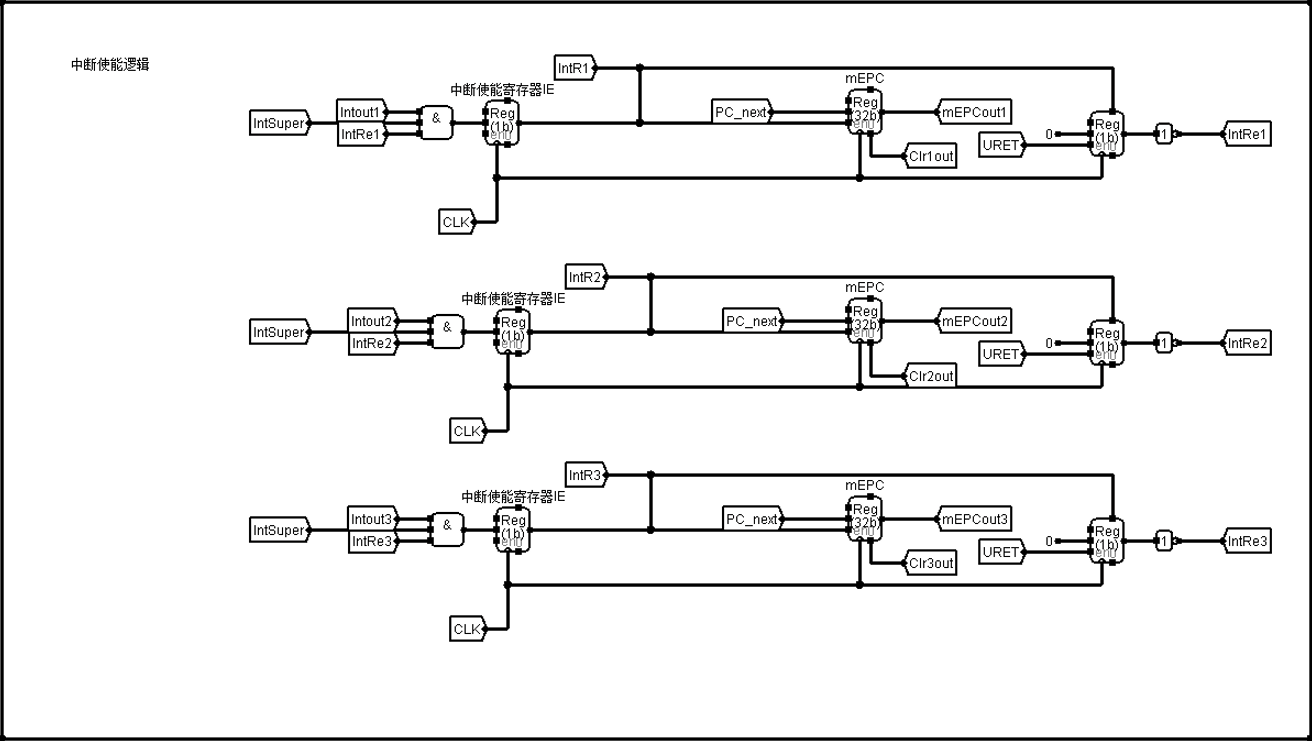


图3.11 中断使能控制逻辑电路

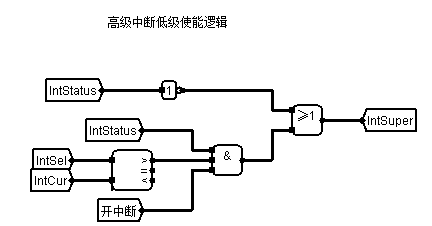


图3.12 IntSuper信号生成逻辑

3.中断状态逻辑

中断状态逻辑第一部分是解决之前提出的第一个问题：判断程序是否已经进入中断。我用IntStatus\_x(x=1,2,3)记录是否进入中断x，判断方法是：如果已经进入了中断x，那么其对应的mEPC寄存器的值一定不为0。相应的，为了维护这一关系的正确性，每次中断结束后都需要把对应的mEPC寄存器清0。这部分逻辑如图3.13 中断状态逻辑第一部分所示。

中断状态逻辑的第二部分的作用是给出当前信号的中断等级IntCur（方便与传入的新的中断的中断等级IntSel比较）、清空信号的逻辑以及中断信号的控制逻辑。其中清空信号需要滞留一个周期，目的是避免mEPC中断点的地址还未送入PC就被清空。这样第三个问题也解决了。这部分逻辑如图3.14 中断状态逻辑第二部分所示。

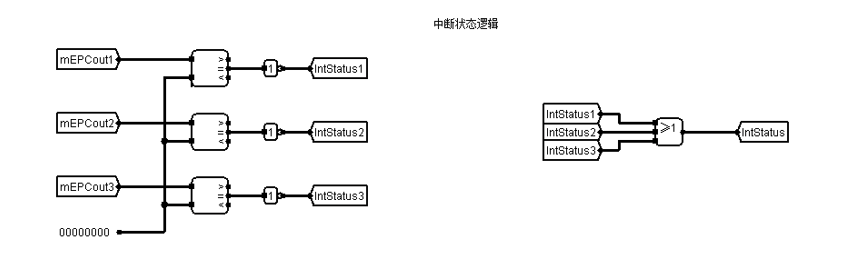


图3.13 中断状态逻辑第一部分

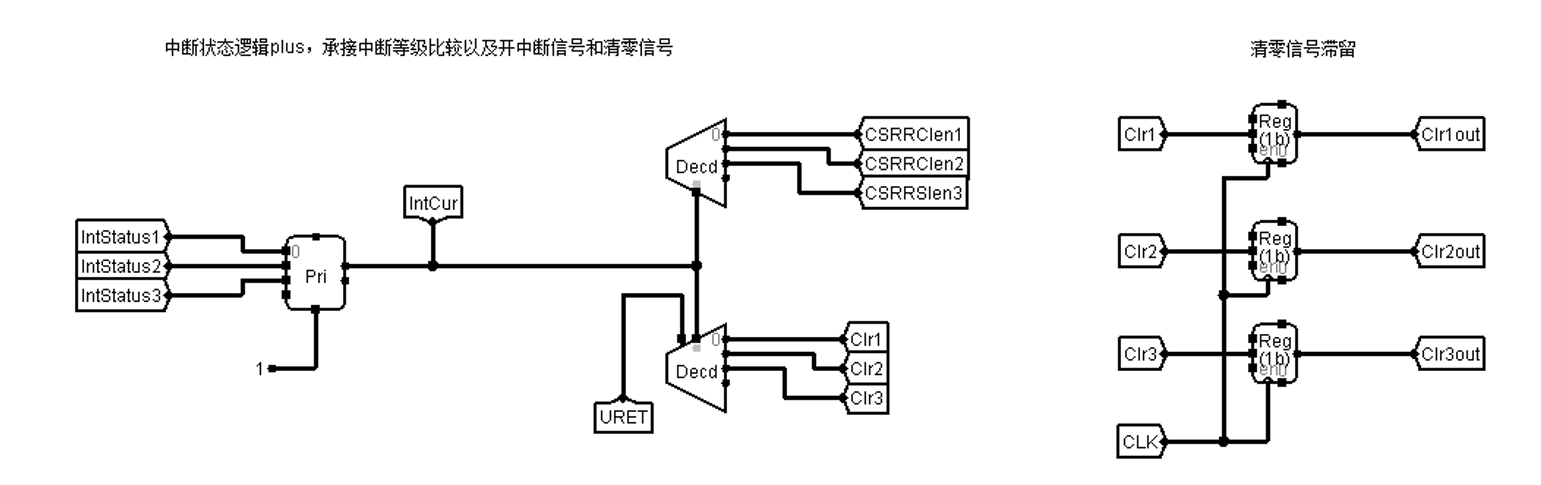


图3.14 中断状态逻辑第二部分

4.中断出口逻辑

中断出口逻辑相对简单，就是将3个mEPC寄存器的输出值通过一个多路选择器，在IntCur信号的选择下将当前运行中断程序对应的断点值返回PC之中。

5.开关中断逻辑

这部分逻辑主要解决之前提出的第四个问题。开关中断逻辑大致的思路是让3个中断信号对应的开关中断分别保存在寄存器中。对于每一个中断信号的开关中断逻辑，当已经进入该中断(IntStatus\_x为1)且传入开中断信号时将对应寄存器置为1；若该中断对应中断服务程序正在运行（CSRRCIen1为1）且传入关中断信号，将对应寄存器置0。这部分电路如图3.15 开关中断逻辑所示。程序在运行时，开关中断通过一个多路选择器得到程序的开关中断信号，如图3.16 程序的开关中断信号所示。

这里CSRRClen信号不能替换成IntStatus信号，原因是关中断只能关闭当前运行的中断程序，因此只需要当前运行中断程序对应的信号（也就是中断状态逻辑中IntCur译码的信号），而IntStatus信号只要进入中断了就会置为1，如果替换可能出现运行中断程序3时，关中断信号将中断1和2关中断的现象。

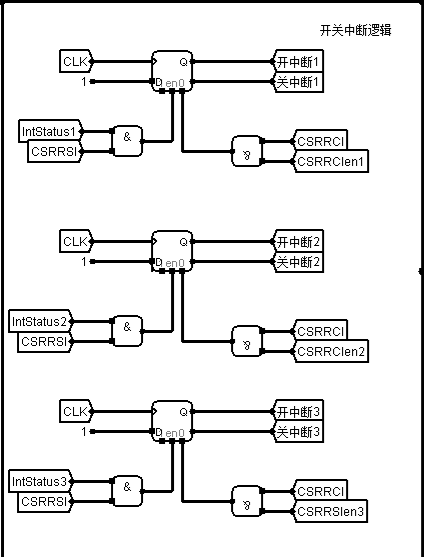


图3.15 开关中断逻辑

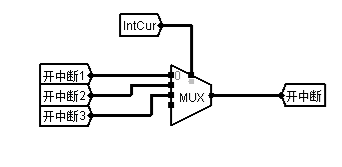


图3.16 程序的开关中断信号

## 流水CPU实现

### 流水接口部件实现

流水接口部件中每个数据都是采用“输入+二路选择器+寄存器+输出”的模式实现的。其中二路选择器的0号输入为数据输入，1号输入为0输入（接地），控制信号为RST信号（实现同步置零）。IF/ID流水寄存器logisim连线如图3.17 IF/ID流水寄存器logisim连线所示。

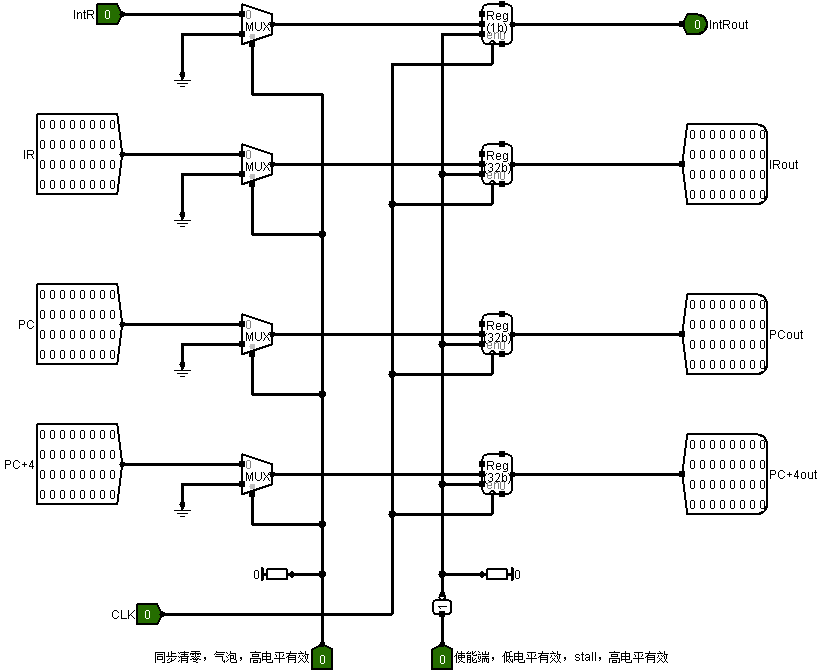


图3.17 IF/ID流水寄存器logisim连线

### 理想流水线实现

理想流水线的实现思路同设计思路，篇幅原因这里不做赘述。

## 气泡式流水线实现

实现气泡流水线，主要解决的是如何处理数据冲突的问题。需要给出正确的stall逻辑，然后按照设计思路为IF/ID流水寄存器和ID/EX流水寄存器添加CLR和EN逻辑。参考资料上给出了数据冲突的逻辑，如图3.18 气泡流水线数据冲突逻辑所示。

没有给出的是RsUsed和RtUsed逻辑。通过观察真值表，我们可以得到产生了JAL信号的JAL指令未使用Rs1；产生了ecall信号的ecall指令、产生了JAL信号的JAL指令和产生了AluSrcB信号的I型指令均未使用Rs2。因此可以得到逻辑：

Rs1Used = ~JAL;

Rs2Used = ~(JAL + ecall + AluSrcB);

这些信号均为id段产生的信号，因为判断数据冲突是判断id段的指令是否与ex和mem段的指令冲突。

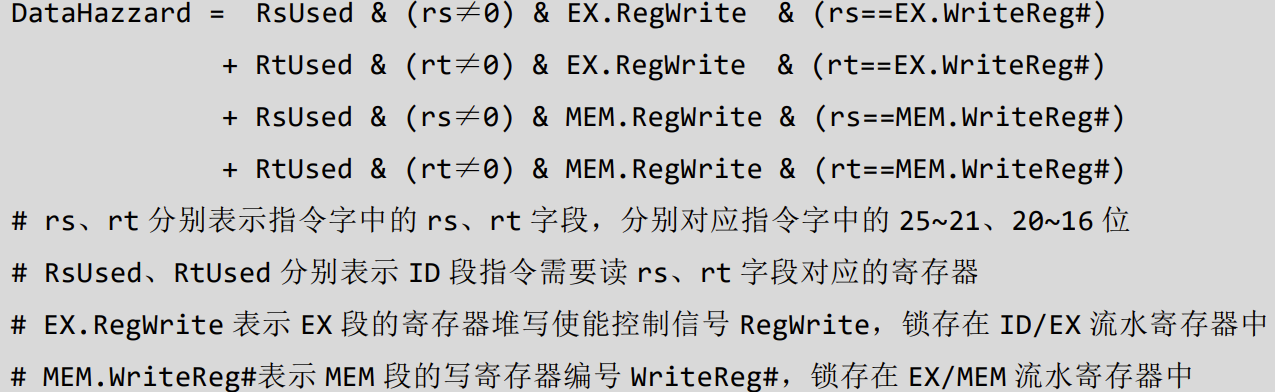


图3.18 气泡流水线数据冲突逻辑

气泡流水线中，stall信号等价与DataHazzard信号。stall信号为1时，将IF/ID流水寄存器的使能端置1，将IF/ID流水寄存器和ID/EX流水寄存器的清空端置1。这样就实现了气泡的插入机制。

## 重定向流水线实现

重定向流水线的数据冲突逻辑相对气泡流水线更复杂，主要需要实现两个逻辑：第一个是ID段指令使用的数据是否与EX段和MEM段的出现冲突，第二个是Load\_Use逻辑。两个逻辑在参考资料中均已给出，如图3.19 重定向流水线数据冲突逻辑、图3.20 重定向流水线LoadUse逻辑所示。

其中RsUsed和RtUsed信号直接复用气泡流水线中的信号即可。

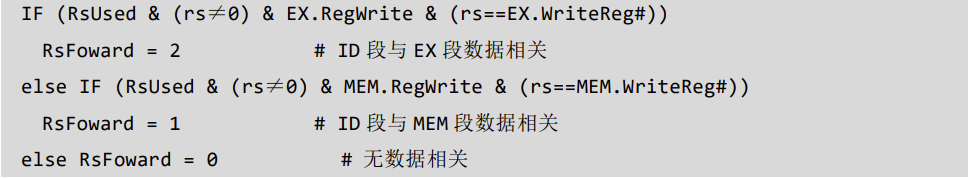


图3.19 重定向流水线数据冲突逻辑

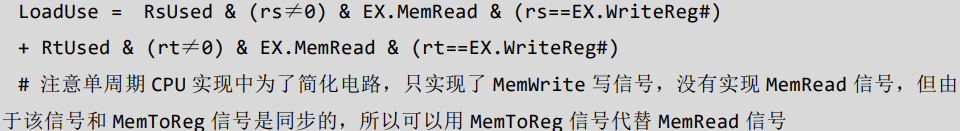


图3.20 重定向流水线LoadUse逻辑

完成上述两个逻辑后，可以得到R1选择信号R1Src和R2选择信号R2Src。然后需要在EX段添加多路选择器，选择ALU的两个输入，如图3.21 ALU输入修改所示。最后还需要添加和气泡流水线一样的CLR逻辑和EN逻辑，这里不再赘述。

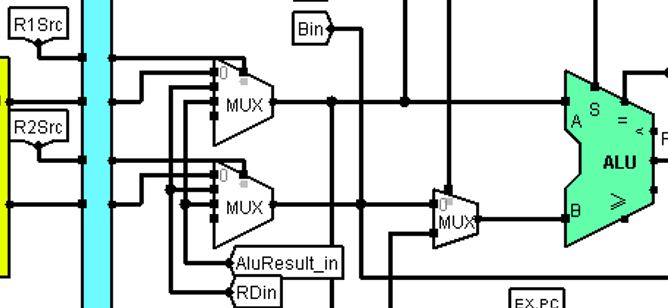


图3.21 ALU输入修改

# 实验过程与调试

## 测试用例和功能测试

在实验过程中，针对不同的设计使用了多种样例。其中，单周期CPU、气泡流水线、重定向流水线均使用risc-v-benchmark\_ccab作为测试程序。单级中断使用risc-v单级中断测试程序，多级中断使用risc-v多级中断测试(EPC硬件堆栈保护)程序。

由于educoder上已经上传了通用程序的评测记录，本节仅仅记录ccab的评测结果。（因此不记录中断相关样例）

### 测试用例risc-v-benchmark\_ccab

单周期CPU、气泡流水线以及重定向流水线3个电路由于测试用例相同，最后的输出结果（总周期数厨外）也相同。

AUIPC指令运行结束后Led显示如图所示。该值与标准答案不符合，分析原因发现AUIPC输出的值与PC的值有关系，由于AUIPC测试前已经运行了其他指令，导致PC初始值不为0.将ccab测试程序单独分离后可以得到正确的结果。如图4.1 AUIPC指令的输出结果所示。

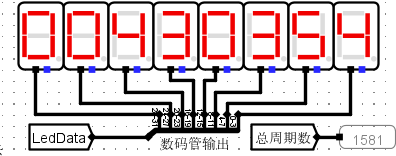


图4.1 AUIPC指令的输出结果

SLTIU指令运行结束后LED显示如图4.2 SLTIU指令的输出结果所示，与预期相符。

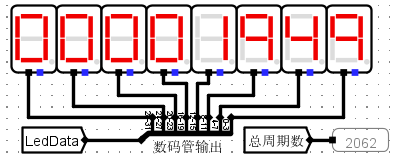


图4.2 SLTIU指令的输出结果

LB指令运行结束后LED显示如图4.3 LB指令的输出结果所示，与预期相符。

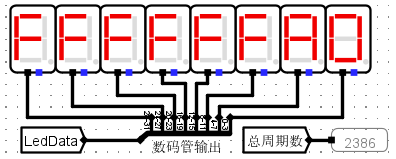


图4.3 LB指令的输出结果

Bge指令运行结束后LED如图4.4 Bge指令的输出结果所示，与预期相符。

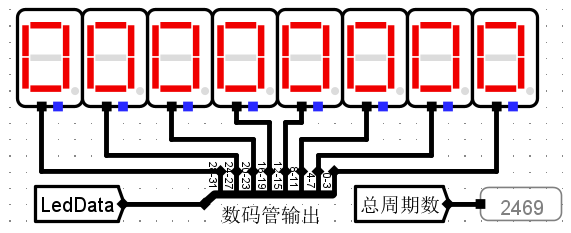


图4.4 Bge指令的输出结果

## 性能分析

本节主要分析气泡流水线和重定向流水线的性能差异。

气泡流水线中，总周期数=1546+4+气泡数目+分支误取深度\*分支数-1。重定向流水线中，总周期数=1546+4+分支误取深度\*分支数+load-use次数-1。（两个公式的分支误取深度均为2）由于重定向流水线的load-use次数实际上是气泡数目的一部分，所以重定向流水线的性能是肯定优于气泡流水线的。

如图4.5 气泡流水线运行结束后数据结果、图4.6 重定向流水线运行结束后数据结果所示，气泡流水线和重定向流水线运行完测试程序risc-v-benchmark\_ccab后的数据运算均符合公式。二者不同之处在于重定向流水线的LoadUse次数远小于气泡流水线的插入气泡数，因此总的时钟周期要比气泡流水线更少。

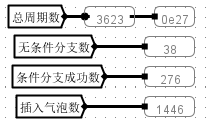


图4.5 气泡流水线运行结束后数据结果

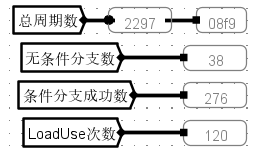


图4.6 重定向流水线运行结束后数据结果

## 主要故障与调试

### 重定向流水线数据冲突逻辑故障

重定向流水线：无法正确处理数据冲突

**故障现象：**执行sw指令时数据冲突逻辑发生错误。

**原因分析：** Rs2Used逻辑出错，我之前认为使用了AluSrcB信号都是用了立即数，因此不会使用rs2。但是SW指令是一个反例，它虽然没有访问寄存器中rs2地址的值，但是将其他值写入了寄存器的rs2地址。出错的逻辑如图4.7 Rs2Used的错误逻辑所示。

**解决方案：**将AluSrcB信号和~S\_Type信号相与再作为输入传入或非门中。正确的电路图如图4.8 Rs2Used的正确逻辑所示。

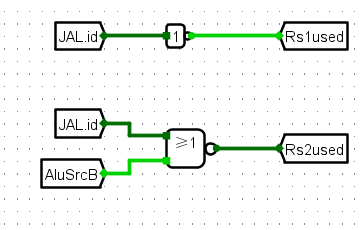


图4.7 Rs2Used的错误逻辑

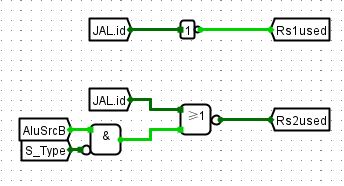


图4.8 Rs2Used的正确逻辑

### Vivado仿真故障

Verilog实现单周期CPU：执行到移位指令时运算错误。

**故障现象：**如图4.9 程序故障输出所示，当仿真到129800ns时，LedData的输出出现错误。原本应该输出fff11111，实际输出为2ffff111。

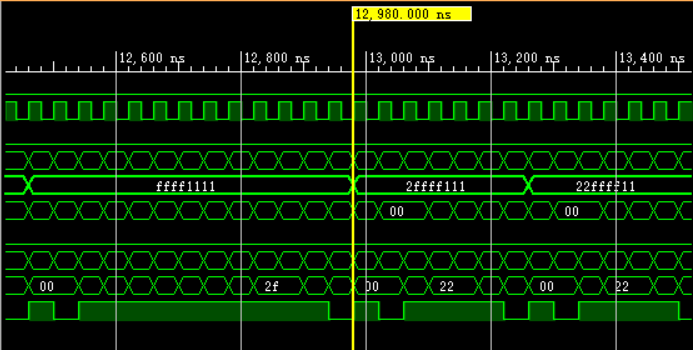


图4.9 程序故障输出

**原因分析：**在对该时刻左右的指令进行排查后，发现是sub t0,t0,t1指令出现问题。仔细检查ALU的实现后，发现我的减法器逻辑是Result = X + ~Y。我去查了资料才知道，~Y是Y的反码。因此需要在后面加上+1。

**解决方案：**将Result = X + ~Y改为Result = X + ~Y + 1即可。

### BGE指令故障

测试程序无法正确输出BGE指令的预期值。

**故障现象：** 在LedData归零之后，测试程序没有正确停止，LedData变成FFFFFFF后继续减少。如图4.10 Bge指令故障输出所示。

**原因分析：**BGE指令需要比较R1和R2值的大小。然而ALU自带的>=信号是无符号比较，测试程序运行时如果变量的值比0小就会停止程序，而ALU在比较0和-1的大小时会认为-1更大，这也是程序无法正确停止的原因。

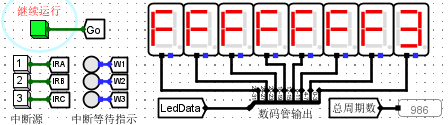


图4.10 Bge指令故障输出

**解决方案：**自己重新用有符号比较器给给出>=信号即可。

## 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RISC-V指令手册，完成数据通路的基本构建。 |
| 第二天 | 填写控制信号表，在Logisim平台实现了单周期CPU的整体框架。 |
| 第三天 | 研究书中流水线的相关内容，完成了理想流水线并通过了评测。 |
| 第四天 | 完成气泡流水线相关内容并通过评测。 |
| 第五天 | 完成重定向流水线相关内容并通过了评测。 |
| 第六天 | 补充了已经完成部分的ccab指令。 |
| 第七天 | 实现单级中断的相关内容并通过了评测。 |
| 第八天 | 研究多级中断，厘清多级中断的相关逻辑。 |
| 第九天 | 完成练度了多级中断的相关内容且通过了品格。开始研究单周期上板。 |
| 第十天 | 利用工具将单周期cpu的线路转换为verilog代码，之后自己参考转换好的电路手写了部分基础组件。 |
| 第十一天 | 完成了控制器等多个大组件的编写以及数据通路的编写，并实现了testbench |
| 第十二天 | 在vivado上调试写好的cpu.v文件，正确仿真后将代码烧到FPGA平台并且正确显示。 |
| 第十三天 | 补充了FPGA平台切换功能 |

# 设计总结与心得

## 课设总结

在这次计算机组成原理课程设计，我主要完成了以下内容：

1. 实现了单周期RISC-V CPU的数据通路。
2. 设计了5段流水线中的4个流水寄存器。
3. 实现了了理想流水线的数据通路。
4. 设计了气泡流水线和重定向流水线的数据冲突逻辑。
5. 实现了气泡流水线和重定向流水线的数据通路。
6. 设计了单级中断和多级中断的相关逻辑。
7. 设计了中断流水线的相关逻辑。
8. 完成了各部分内容在educoder平台的评测。
9. 完成了单周期CPU FPGA平台的正确演示。

## 课设心得

本次课设应该是我入学以来投入最多，做的最认真的一次课设。我在大概开学前两天开始写课设，每天都记录了进展。现在来看，不仅是成果，心路历程也是非常丰富。

我完成本次课设的时间和老师ppt里面的时间安排大致相同。第一部分设计单周期cpu可能算是比较困难的，因为后面的内容都是在此基础上添枝加叶，而这部分需要从头开始。特别是最开始的时候没有一个完整的设计思路，很多细节在任务书上也没有注意到。比如数据通路画完了都还没找到ecall指令的a7寄存器对应的是哪个值；指令中的CSRRSI和CSRRCI指令究竟是什么意思；哪些类型的指令的立即数需要移位，移多少（最后是同学告知的）等等。当时遇到的困难，现在看来都不算难事，不过对于刚刚上手的人可能会比较头疼。

后面的流水线部分其实完成的非常顺利，主要原因是给的资料把每一步都说的很清楚，很大程度上降低了难度。ccab指令的添加也比较顺利，只有部分细节上的问题卡壳了，不过在反复调试后和正确输出对比也能找到问题所在。

另一个比较困难的地方是中断部分。和流水线部分相反，中断部分给的资料非常模棱两可，所以花的时间会多一些。我认为单级中断完全可以多给一些提示，像CSRRSI和CSRRCI指令其实只在多级中断里面用到，单级中断明明没有使用，任务书中却专门提到了“RISC-V 处理器中没有专门的开中断、关中断指令，具体实现时是可利用 CSR 寄存器组访问指令 csrrsi、csrrci 实现”，在我看来是误导信息。多级中断虽然更难，不过只要完成了单级中断，完全是可以自己思考设计出来的。在完成单级中断和多级中断设计之后，流水中断也就很容易了。

最后一部分是单周期上板。我首先考虑的是用老师给的程序尝试将logisim平台的电路转换成verilog代码，然而遇到了很多问题，例如ALU中的乘法器和除法器不支持转换、移位器转出来完全不对、Regifile不支持转换、转换出来的代码不支持负数等等。所以最后我只得到了一个大概的框架，于是决定自己用verilog重新实现。除了时钟部分我复用了转换出来的代码，其他部分还是挺顺利的，只是工作量比较大，重复性质的操作过多，而且debug比较困难（主要因为使用的vivado版本较低）。最后看到FPGA平台上显示出正确的结果时，可谓是成就感满满。

这次课设相比于数据结构的课设，我感觉难度会稍微低一点。主要原因还是课程组的老师们把任务说的非常清楚，每个阶段都可以检查自己完成的正确性，资料也给的相对详实。我的建议是可以修改任务书中断的部分内容，给出更加清楚明确的指示。此外FPGA部分开发的参考资料其实参考价值也不大，给出部分Verilog相关资料以及完成单周期CPU设计的步骤和思路就可以了，完成了这部分，后面流水线的上板大概也能仿照着完成。

# 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: 徐子路** |