华中科技大学

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

目 录

1 课程设计概述

1.1 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。

1.2 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。

1.3 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。

1.4 技术指标

（8） 支持表 1.1 指令集前 24 条基本 32 位 RISC_V 指令；

（9） 支持教师指定的 4 条扩展指令；

（10）支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（11）支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

（12）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。

（13）能运行教师提供的标准测试程序，并自动统计执行周期数

（14）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	

#	指令助记符	指令类型	简单功能描述	备注
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	

2 总体方案设计

2.1 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。

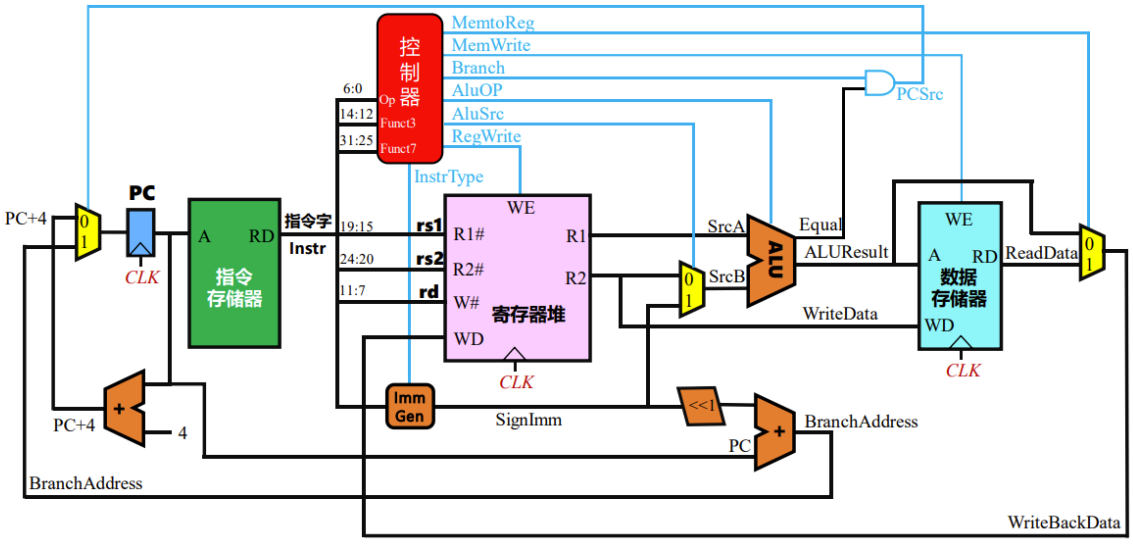

图 2.1 总体结构图

2.1.1 主要功能部件

1. 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。

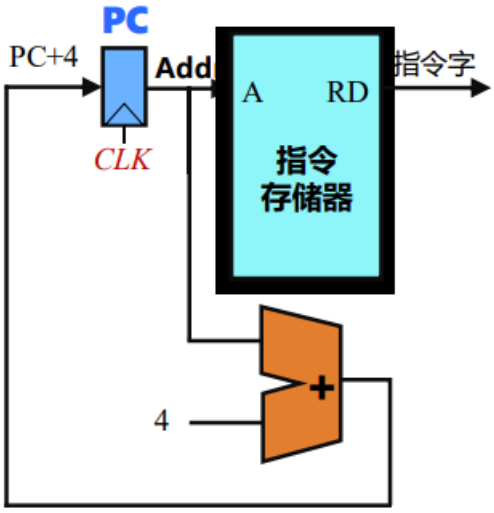

图 2.2 取指令数据通路

2. 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.

3. 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

引脚	输入/输出	位宽	功能描述
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

ALU_OP	十进制	运算功能
		符号乘法
0100	4	Result = X/Y;　Result2 = X%Y　无符号除法

4. 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

2.1.2 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。

2.1.3 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

2.2 中断机制设计

2.2.1 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）。

2.2.2 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。

2.3 流水 CPU 设计

2.3.1 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。

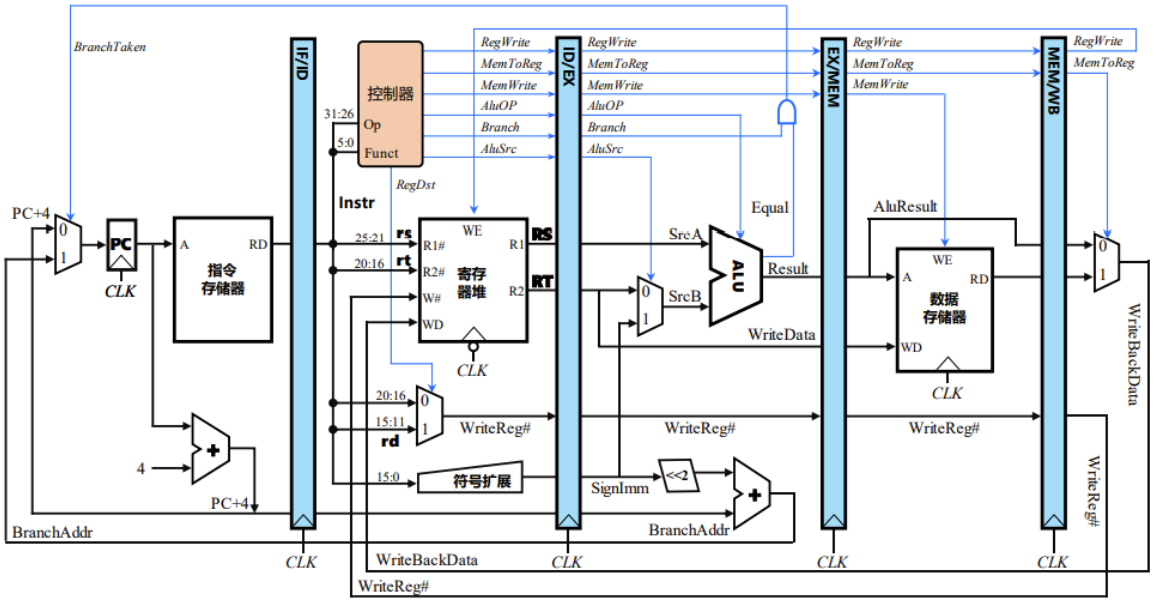

图 2.3 流水 CPU 总体结构图

2.3.2 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）。

2.3.3 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。

2.4 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。

2.5 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。

3 详细设计与实现

3.1 单周期 CPU 实现

3.1.1 主要功能部件实现

1) 程序计数器（PC）

① 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。

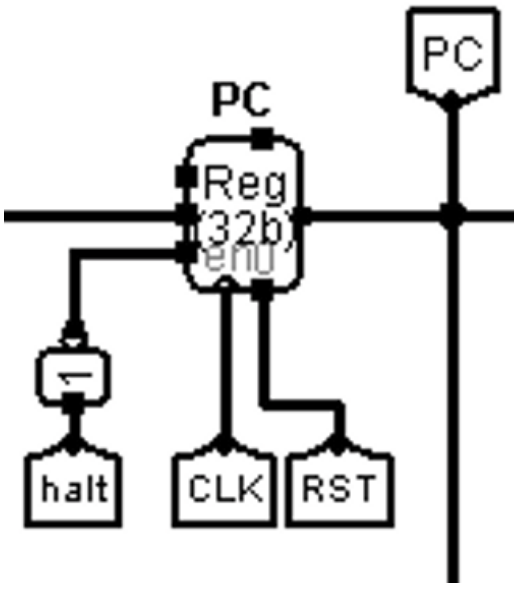

图 3.1 程序计数器（PC）

② 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);
```

2) 指令存储器（IM）

① 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。

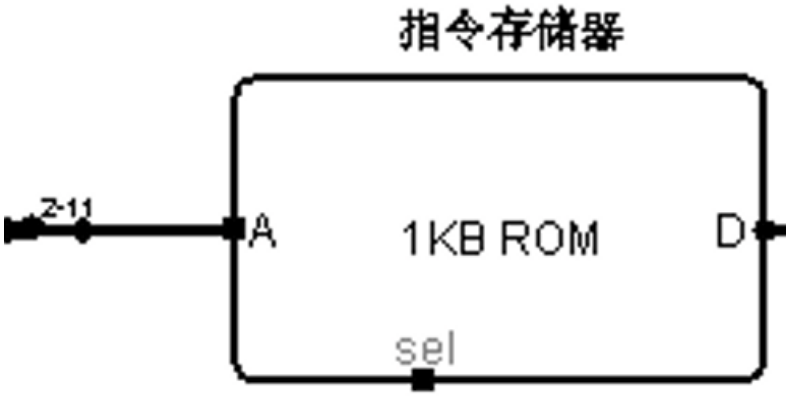

图 3.2 指令存储器（IM）

② 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。

3.1.2 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	-	-	-	-	-	-	-

指令	PC	RF				ALU			DM	
		R1#	R2#	W#	Din	A	B	OP	Addr	Din
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。

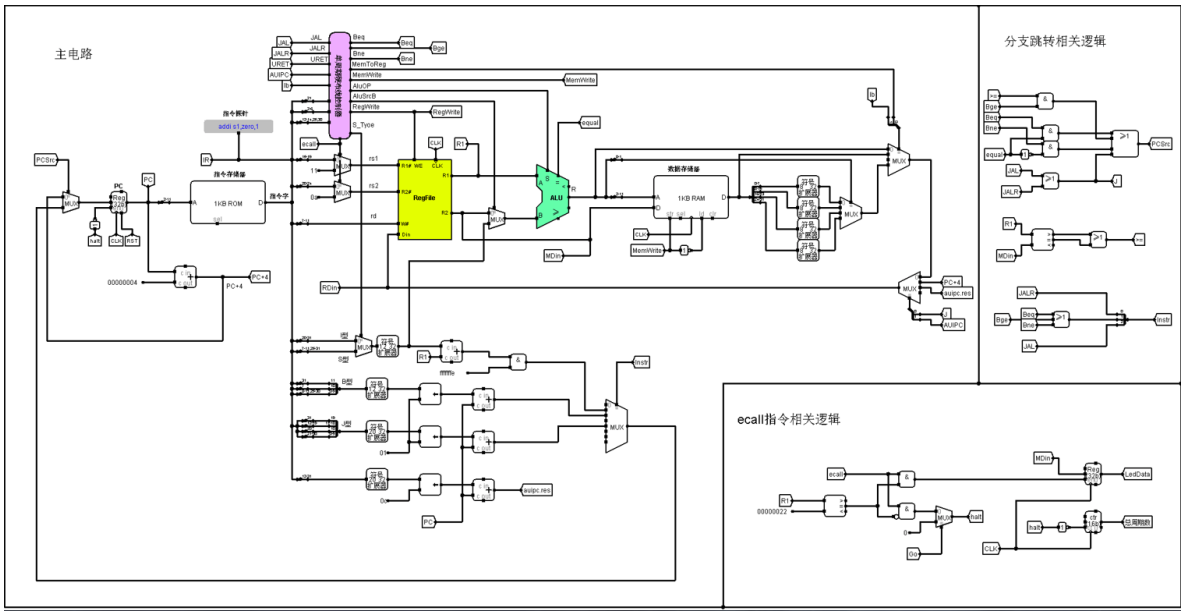

图 3.3 单周期 CPU 数据通路（logisim）

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。

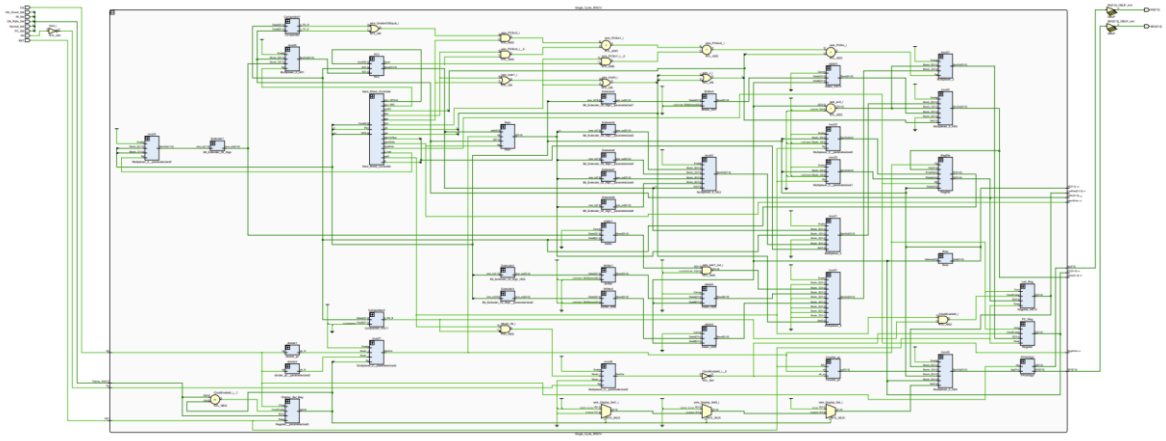

图 3.4 单周期 CPU 数据通路（FPGA）

3.1.3 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。

#	指令	Funct7 (十进制)	Funct3 (十进制)	OpCode (十六进制)	ALU_OP	MemtoReg	MemWrite	ALU_Src	RegWrite	ecall	S_Type	BEQ	BNE	Jal	jalr	AUIPC	LB	BGE	CSRRSI	CSRRCI
1	add	0	0	c	5				1											
2	sub	32	0	c	6				1											
3	and	0	7	c	7				1											
4	or	0	6	c	8				1											
5	slt	0	2	c	11				1											
6	sltu	0	3	c	12				1											
7	addi		0	4	5			1	1											
8	andi		7	4	7			1	1											
9	ori		6	4	8			1	1											
10	xori		4	4	9			1	1											
11	slti		2	4	11			1	1											
12	slli	0	1	4	0			1	1											
13	srli	0	5	4	2			1	1											
14	srai	32	5	4	1			1	1											
15	lw		2	0	5	1		1	1											
16	sw		2	8	5		1	1			1									
17	ecall	0	0	1c						1										
18	beq		0	18								1								
19	bne		1	18									1							
20	jal			1b					1					1						
21	jalr		0	19				1	1						1					
22	CSRRSI		6	1c															1	
23	CSRRCI		7	1c																1
24	URET	2	0	1c																
25	AUIPC			5					1							1				
26	SLTIU		3	4	12			1	1											
27	LB		0	0	5	1		1	1								1			
28	BGE		5	18														1		

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。

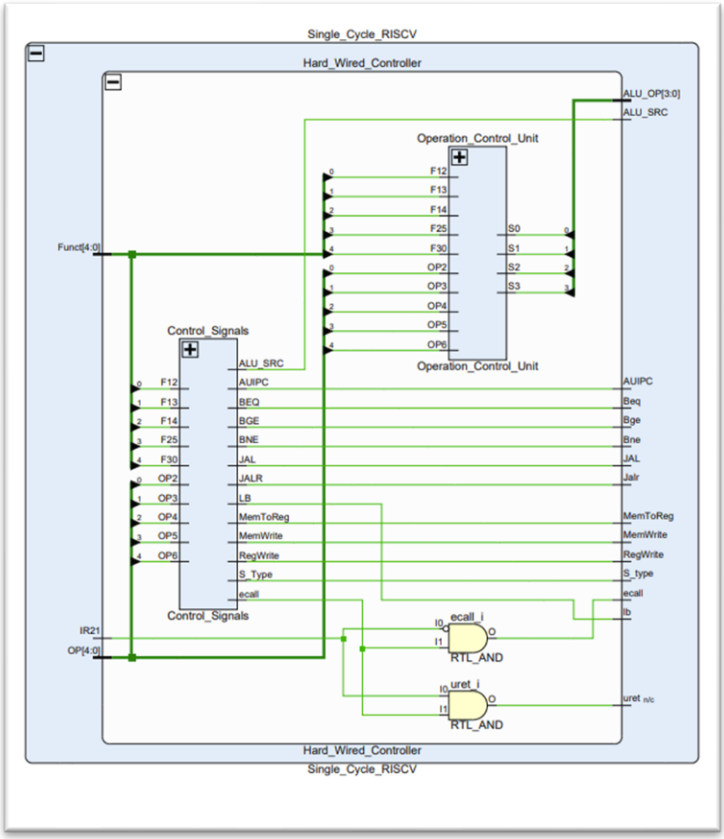

图 3.6 主控制器原理图

3.2 中断机制实现

3.2.1 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。

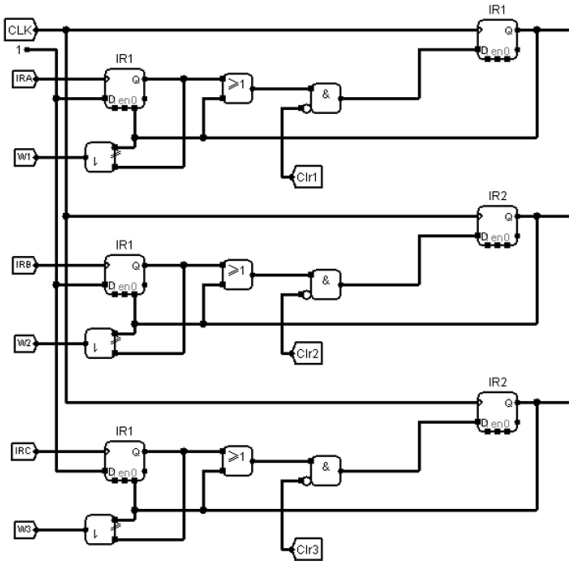

图 3.7 中断按键信号采样电路

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。

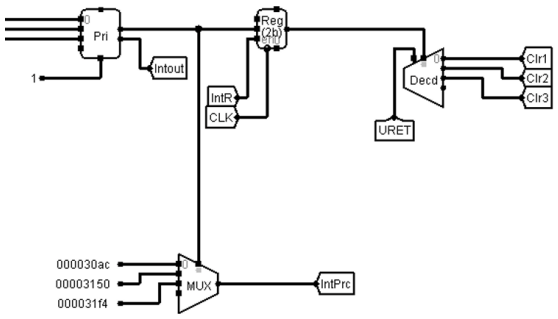

图 3.8 中断识别逻辑电路

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。

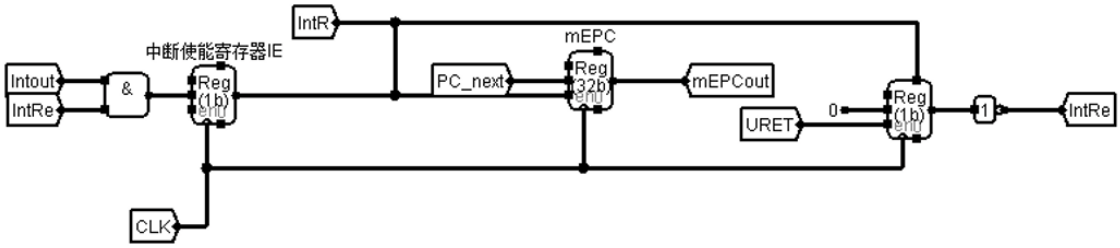

图 3.9 中断使能寄存器数据通路

3.2.2 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。

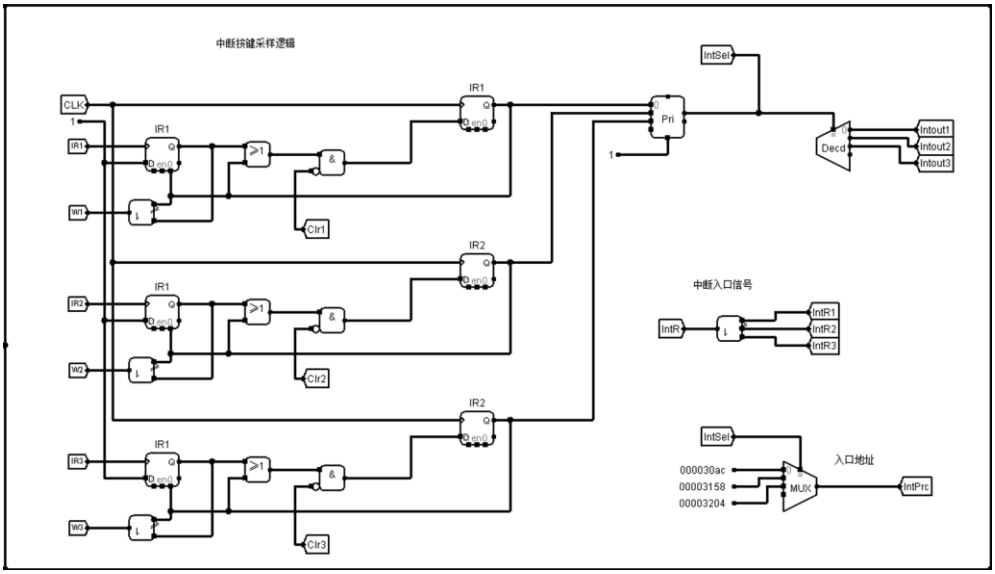

图 3.10 中断按键采样逻辑电路

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**错误!未找到引用源。**所示。IntSuper 信号的生成逻辑如图 3.12 IntSuper 信号生成逻辑所示。

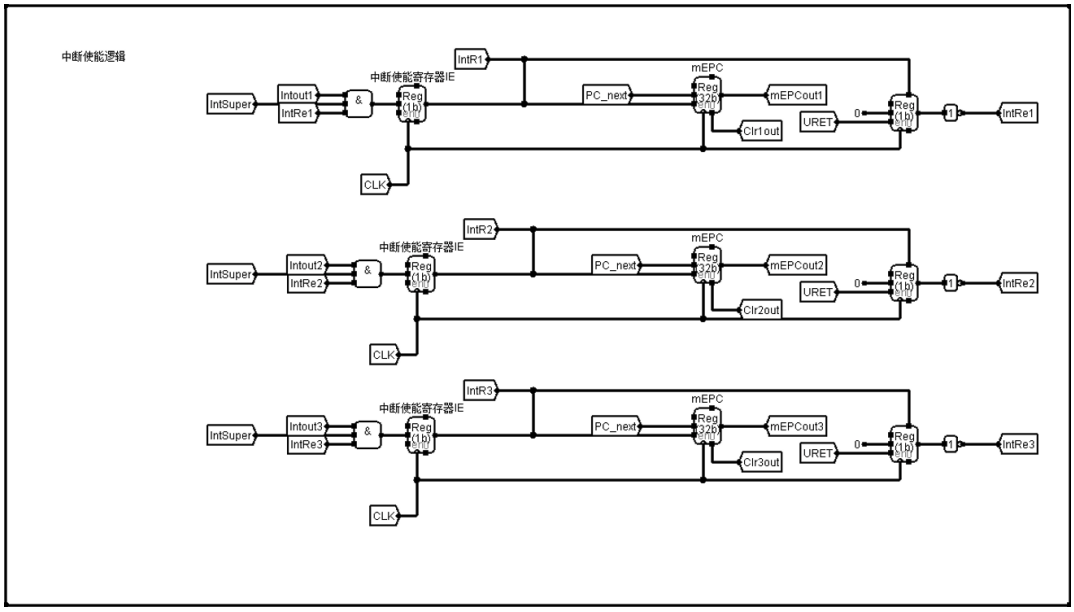

图 3.11 中断使能控制逻辑电路

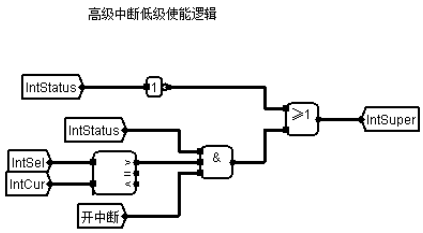

图 3.12 IntSuper 信号生成逻辑

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

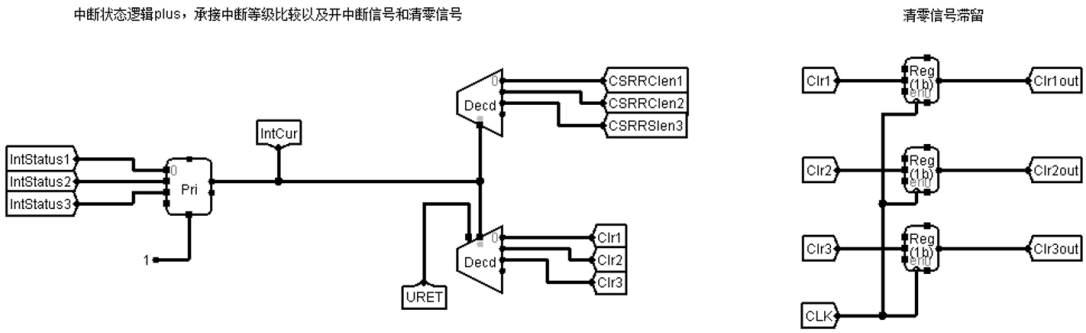

图 3.14 中断状态逻辑第二部分

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 CSRRCIen 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。

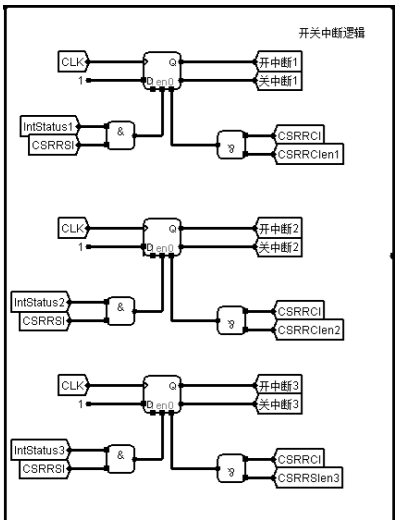

图 3.15 开关中断逻辑

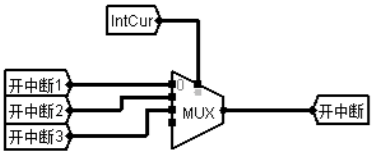

图 3.16 程序的开关中断信号

3.3 流水 CPU 实现

3.3.1 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。

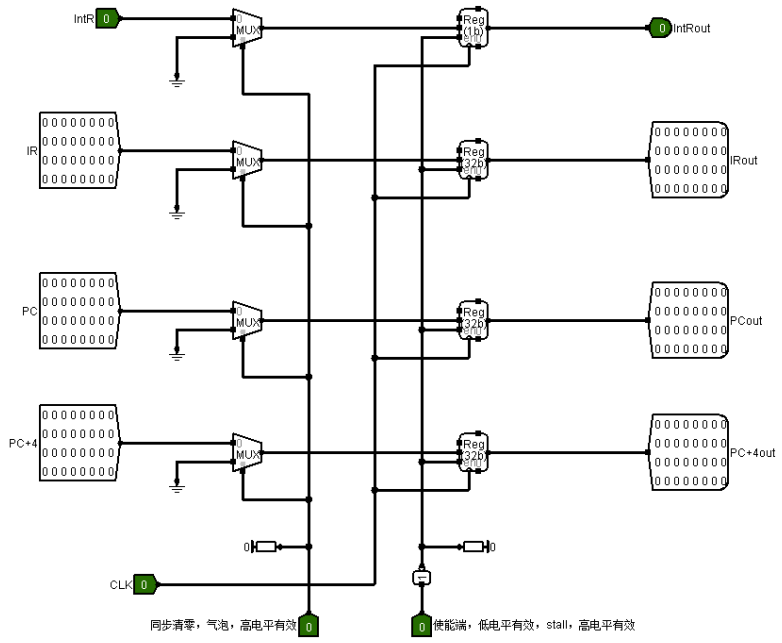

图 3.17 IF/ID 流水寄存器 logisim 连线

3.3.2 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。

3.4 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。

```
DataHazzard =  RsUsed & (rs≠0) & EX.RegWrite  & (rs==EX.WriteReg#)
             + RtUsed & (rt≠0) & EX.RegWrite  & (rt==EX.WriteReg#)
             + RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#)
             + RtUsed & (rt≠0) & MEM.RegWrite & (rt==MEM.WriteReg#)
# rs、rt 分别表示指令字中的 rs、rt 字段，分别对应指令字中的 25~21、20~16 位
# RsUsed、RtUsed 分别表示 ID 段指令需要读 rs、rt 字段对应的寄存器
# EX.RegWrite 表示 EX 段的寄存器堆写使能控制信号 RegWrite，锁存在 ID/EX 流水寄存器中
# MEM.WriteReg#表示 MEM 段的写寄存器编号 WriteReg#，锁存在 EX/MEM 流水寄存器中
```

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。

3.5 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。

```
IF (RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#))
  RsFoward = 2            # ID 段与 EX 段数据相关
else IF (RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#))
  RsFoward = 1            # ID 段与 MEM 段数据相关
else RsFoward = 0           # 无数据相关
```

图 3.19 重定向流水线数据冲突逻辑

```
LoadUse =  RsUsed & (rs≠0) & EX.MemRead & (rs==EX.WriteReg#)
+ RtUsed & (rt≠0) & EX.MemRead & (rt==EX.WriteReg#)
# 注意单周期 CPU 实现中为了简化电路，只实现了 MemWrite 写信号，没有实现 MemRead 信号，但由
于该信号和 MemToReg 信号是同步的，所以可以用 MemToReg 信号代替 MemRead 信号
```

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。

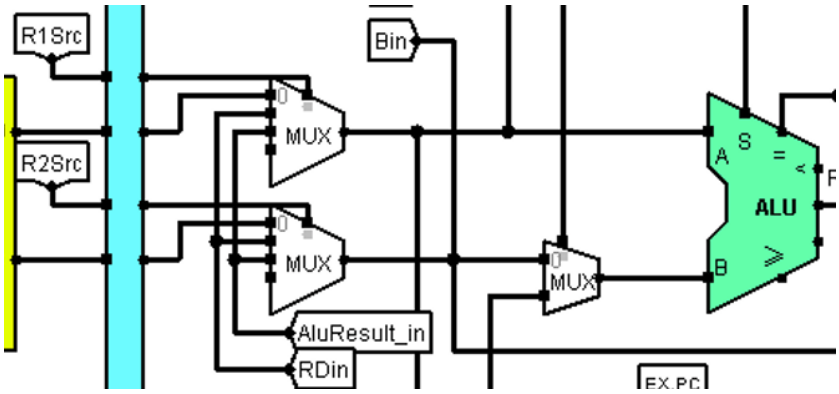

图 3.21 ALU 输入修改

4 实验过程与调试

4.1 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）

4.1.1 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。

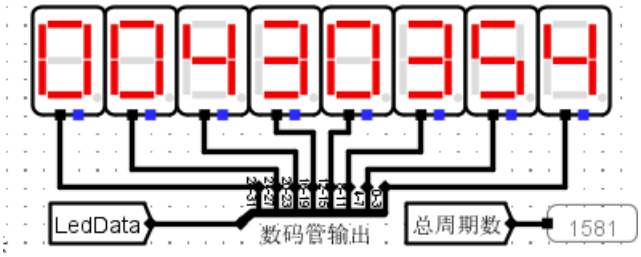

图 4.1 AUIPC 指令的输出结果

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。

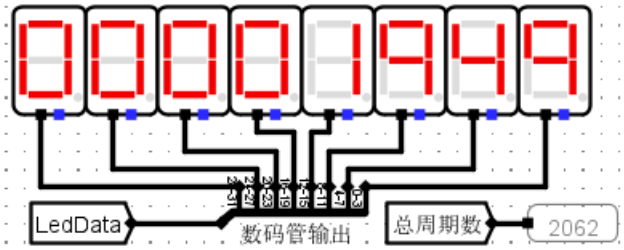

图 4.2 SLTIU 指令的输出结果

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。

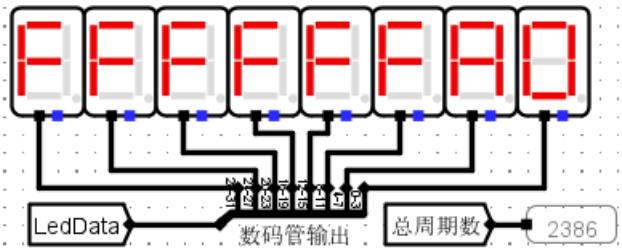

图 4.3 LB 指令的输出结果

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。

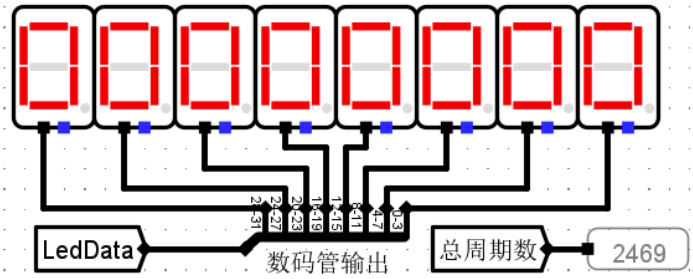

图 4.4 Bge 指令的输出结果

4.2 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。

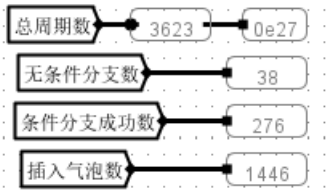

图 4.5 气泡流水线运行结束后数据结果

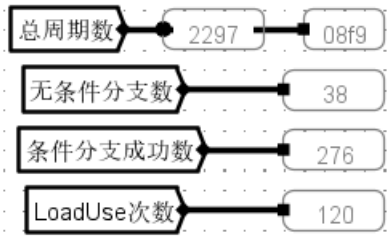

图 4.6 重定向流水线运行结束后数据结果

4.3 主要故障与调试

4.3.1 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。

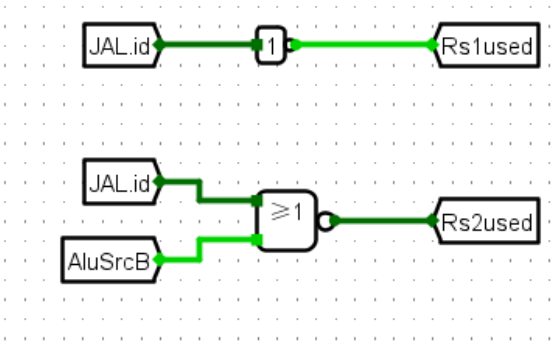

图 4.7 Rs2Used 的错误逻辑

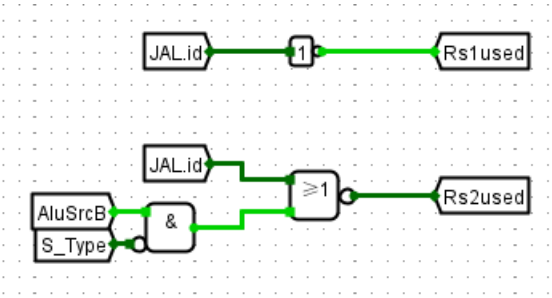

图 4.8 Rs2Used 的正确逻辑

4.3.2 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**错误!未找到引用源。**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。故障输出如图 4.9 所示。

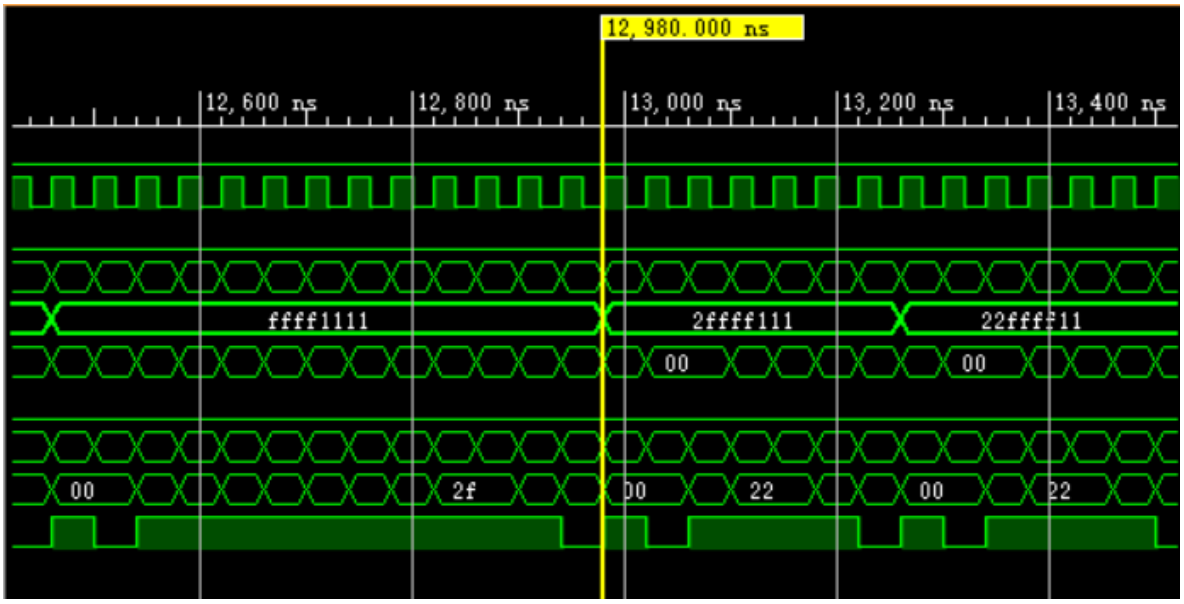

图 4.9 程序故障输出

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。

4.3.3 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。

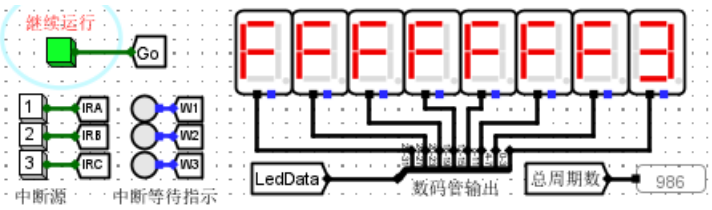

图 4.10 Bge 指令故障输出

解决方案：自己重新用有符号比较器给给出>=信号即可。

4.4 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

5 设计总结与心得

5.1 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。

5.2 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 徐子路