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：	CS2011
学　　号：	U202010755
姓　　名：	路昊东
电　　话：	15671615911
邮　　件：	2441164168@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 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	简单功能描述	备注
1	ADD	加法	指令格式参考 RISC-V32 指令集，最终功能以 RARS 模拟器为准。
2	ADDI	立即数加	
3	AND	与	
4	ANDI	立即数与	
5	SLL	逻辑左移	
6	SRA	算数右移	
7	SRL	逻辑右移	
8	SUB	减	
9	OR	或	
10	ORI	立即数或	
11	XORI	立即数异或	
12	LW	加载字	
13	SW	存字	

#	指令助记符	简单功能描述	备注
14	BEQ	相等跳转	
15	BNE	不相等跳转	
16	SLT	小于置数	
17	SLTI	小于立即数置数	
18	SLTU	小于无符号数置数	
19	JAL	转移并链接	
20	JALR	转移到指定寄存器	if ($a7==34) LED 输出 $a0 值 else 暂停并等待 Go 按键
21	ECALL	系统调用	
22	CSRRSI	访问 CSR	中断相关，可简化为开中断
23	CSRRCI	访问 CSR	中断相关，可简化为关中断
24	URET	中断返回	清中断，mEPC 送 PC，开中断
25	SLL	逻辑左移	C - 1
26	AUIPC	PC 加立即数	C - 5
27	LHU	无符号半字加载	A - 4
28	BLTU	无符号小于时分支	B - 3

2 总体方案设计

2.1 单周期 CPU 设计

单周期 CPU（Single Cycle Processor）是指所有指令均在一个时钟周期内完成的处理器。是指一条指令在一个时钟周期内完成并开始下一条指令的执行。由时钟的“上升沿”和“下降沿”控制相关操作。两个相邻的“上升沿”或“下降沿”之间的时间间隔就是 CPU 的“时钟周期”，而这个时钟周期必须设计成对所有指令都等长。在单周期 CPU 中，一条指令的执行过程基本分为取指令、指令译码、指令执行、存储器访问、结果写回，这期间数据通路的任何资源都不能被重复使用，任何需要被多次使用的资源都需要设置多个。

本阶段使用硬布线控制器实现 RISC-V 单周期 CPU，支持中断相关以外的 24 条基本指令和 4 条个性化指令。由于实验取指令和指令执行阶段均需要使用存储器，故采用哈佛结构[3]。

总体结构图如图 2.1 所示。

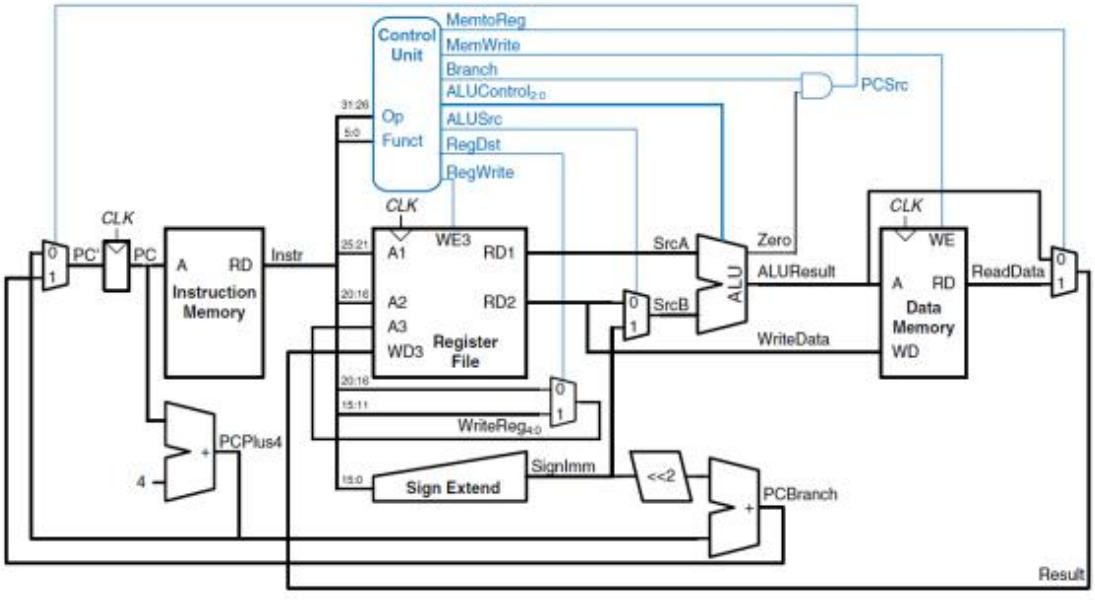


图 2.1 总体结构图

下面将从主要功能部件、中断机制设计、流水CPU设计、气泡式流水线设

计、重定向流水线与单级中断机制设计，这5个方面介绍单周期 RISC-V CPU 的设计。

2.1.1 主要功能部件

1. 程序计数器 PC

在本实验中。PC 的作用是保存下一条指令的地址，用一个 32 位寄存器实现。

首先由跳转指令信号确定正确的输入地址，通过多路选择器选出并输入 PC。同时我加入了复位信号，用来实现异步复位。

2. 指令存储器 IM

本实验中，IM 主要用于保存需要执行的指令本身，用 ROM 实现，并配合 PC 值取出当前需要执行的指令。需要载入新的程序时，需要手动载入。

运行前要将指令载入 IM。输入地址取 PC 的输出地址的 2-11 位输入，是字地址；输出则是相应地址的指令。同时加入片选信号，高电平有效。

3. 运算器 ALU

ALU 用于实际执行运算，本实验中规定 ALU 有三个输入和五个输出。

其中输入引脚 X、Y 均为 32 位，代表 ALU 的操作数。4 位的输入引脚 AluOp 为运算器功能码，具体功能见表 2.1。

表 2.1 运算器功能

AluOP	D	运算表达式	功能
0000	0	Result = X << Y Result2=0	逻辑左移（Y 取低五位）
0001	1	Result = X >>>Y Result2=0	算术右移（Y 取低五位）
0010	2	Result = X >> Y Result2=0	逻辑右移（Y 取低五位）
0011	3	Result = (X * Y)[31:0] Result2 = (X * Y)[63:32]	实现无符号乘法
0100	4	Result = X / Y Result2 = X % Y	实现无符号除法

0101	5	Result = X + Y	(Set OF/UOF)
0110	6	Result = X - Y	(Set OF/UOF)
0111	7	Result = X & Y	实现按位与
1000	8	Result = X | Y	实现按位或
1001	9	Result = X ⊕ Y	实现按位异或
1010	10	Result = ~ (X | Y)	实现按位或非
1011	11	Result = (X < Y) ? 1 : 0	实现有符号比较
1100	12	Result = (X < Y) ? 1 : 0	实现无符号比较

输出引脚 Result 和 Result2 同样为 32 位，分别代表了 ALU 的运算结果以及结果的第二部分（用于乘法指令的结果高位或除法指令的余数位，其他操作时 Result2 为零）；其余的输出引脚均为 1 位。当 x ≥ y 时，≥ 引脚输出 1，当 x ＜ y 时，＜ 引脚输出 1，当 x＝y 时，Equal 引脚输出 1，否则为 0；

4. 寄存器堆 RF

RF 读取寄存器编号直接取得指令中对应位置的编号即可，包含 32 个寄存器，其中 0 号寄存器恒 0，在该实验中使用的是资源包 CS3410 中封装好的 Register File，配合具体指令实现。RF 有 6 个输入引脚和 2 个输出引脚，其功能描述如表 2.2 所示。

表 2.2 Regfile 单元引脚与功能

引脚	位宽	I/O	功能
CLK	1	输入	时钟引脚
WE	1	输入	写使能端
W#	5	输入	写入寄存器编号
R1#	5	输入	将被读取的寄存器 R1 的编号
R2#	5	输入	将被读取的寄存器 R2 的编号
Din	32	输入	写入数据
R1	32	输出	读取的寄存器 R1 的值
R2	32	输出	读取的寄存器 R2 的值

5. 数据存储器 DM

DM 作为内存来使用，存放一般数据。本实验中，由于需要同时支持读和写，DM 通过 RAM 实现，既可以通过 Logisim 提供的库文件实现片选，同时可以参考存储实验，将四个小 RAM 组合。

2.1.2 数据通路的设计

本实验中采用简单迭代法实现单周期 CPU。

首先完成支持一条固定 R 型指令的基本通路，再在此基础上不断增加新的数据通路，支持新的指令，直至所有指令都能正常运行。

要支持一条指令的运行必须有取指令逻辑和执行指令的逻辑，首先应该完成取指令的数据通路。主要需要完成的是取指令、R 型指令、I 型指令、S 型指令、B 型指令、U 型指令、J 型指令的数据通路。

2.1.3 控制器的设计

设计控制器首先要统计控制信号，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。统计后信号的功能如表 2.3 所示。

表 2.3 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
RegWrite	寄存器写使能	寄存器写回信号
MemWrite	写内存控制信号	s 型指令
AluOP	运算器操作控制符（4 位）	根据指令功能选择
MemToReg	寄存器写入数据来自寄存器	lw 指令
S_Type	S 型指令译码信号	s 型指令
AluSrcB	运算器 B 输入选择	访存指令，立即数运算类指令
Jalr	JALR 指令译码信号	JALR 指令
Jal	JAL 指令译码信号	JAL 指令，选择寄存器写回编号，写回值
Beq	Beq 指令译码信号	Beq 指令，有条件分支控制

Bne	Bne 指令译码信号	Bne 指令，有条件分支控制
ecall	ecall 指令译码信号	根据 a7 寄存器的值，选择是停机还是输出
Bltu	Bltu 指令译码信号	Bltu 指令
Lhu	Lhu 指令译码信号	Lhu 指令
Readers1	Rs1 使用信号	使用 Rs1 寄存器
Readers2	Rs2 使用信号	使用 Rs2 寄存器
CSRRSI	CSRRSI 指令译码信号	csrrsi 指令，开中断
CSRRCI	CSRRCI 指令译码信号	csrrci 指令，关中断
Auipc	Auipc 指令译码信号	Auipc 指令

统计发现，控制器主要分为运算器控制信号以及其他控制信号的生成，输入 IR 中的信息用来区分不同的指令，根据不同指令的功能不同，要求的控制信号也不同，据此填写表格生成电路。

这里不得不提 MemToReg，它代表写回寄存器的数据是来自内存还是 Alu 的计算结果。经过 MemToReg 后还会要经过 LUI 等其他控制信号的选择。

2.2 中断机制设计

2.2.1 总体设计

中断分为外部中断和内部中断，本次实验中只考虑可屏蔽的外部中断，需要考虑的又有硬件实现的中断支持和软件实现的中断处理。

本部分分为单级中断设计以及多级中断设计。要求在原单周期 CPU 的基础上增加单级中断以及多级中断处理机制，支持三个外部按键中断源，需要增加中断按键信号采样电路，实现与中断相关的寄存器，设计中断识别逻辑，增加中断隐指令数据通路，增加 URET 指令数据通路，增加 CSRRI、CSRRCI、CSRRW 指令的数据通路，设计中断服务程序等。多级中断有硬件堆栈保护和内存堆栈保护两种实现形式。

首先考虑单级中断：由于执行中断程序完毕后需要能够返回进入中断的位置并恢复整个 CPU 的状态，所以在进入中断时首先要将 PC 的值保存在 EPC 寄存器中，保存 CPU 的状态，并在退出中断时将 EPC 的值返回给 PC（恢复）。在流水

中断中要考虑传入的 PC 值是哪个阶段的（什么时候加载中断处理程序的地址）。这里我使用优先编码器选择优先级最高的中断信号。用 IE 保存开关中断情况。

实现多级中断基于单级中断，由于多级中断存在优先级的问题，除了实现存储 PC 的堆栈外，还需要对不同等级的中断进行处理，由硬件进行判断新的中断能否打断旧的中断。此时的 PC 保存要采用硬件栈的形式。进入中断以后关中断，保护现场以后 CSRRSI 开中断，要结束的时候 CSRRCI 关中断，恢复现场完毕 URET 后开中断。这里尤其要注意在退出多级中断时,也需要判断继续执行一条被打断的低级中断还是开始执行一个已经在排队的低级中断。

2.2.2 硬件设计

1. 单级中断：

使用中断按键参考电路输入相应的中断信号，在同时有多个中断信号的时候使用优先编码器选择出最先执行的中断号。假设现在 CLK 为低电平，单击中断按钮时，EPC 将 PC 中的值（指向第 a 条指令）保存下来，CLK 到达上升沿，把第 a + 1 条指令的 PC 值加载到了 PC 寄存器中，随后中断信号到达中断信号存储，经过优先编码器，产生了一个中断信号 INT，获得中断处理程序入口 INTPC。 INT 产生后，异步置零 IE，这里用 ReadyInt 寄存器来存储 INT 的产生情况（在 INT 消亡以后，ReadyInt = 1）。此时仍在同一次高电平。在本次高电平和下一次低电平期间，第 a + 1 条指令完成了计算，并在下一个上升沿来临的时候立即写入。同时，ReadyInt 控制的指令地址的多路选择器选择从 INTPC 取得下一条指令的地址，而非 PC 寄存器。EPC 取得了 PC 寄存器的值，也就是第 a+2 条指令的地址，也就是中断程序的返回地址。ReadyInt 经过了一个寄存器的延迟，随着 CLK 同步清空了自己。

在中断程序结束退出中断时会有 URET 指令，此时将 EPC 中的值重新送到 PC 中复位：URET 到来时，NextPC 的值从 EPC 选择，并且清空中断信号存储。URET 结束的时候，置 IE 为 1，结束了本次中断的处理。

这里要注意，EPC 中的值只有当中断发生时才需要更新，其他时候不需要也不能进行更新操作。PC 的输入值也需要对中断发生信号和 URET 指令进行判断，以输入中断处理程序地址和复位的 EPC。此过程需要结合扩展的 URET 指令进行。在流水中断中，还需要考虑送入 EPC 中的 PC 是哪一阶段的 PC 值,避免指令超前或者滞后的问题。

2. 多级中断：

由于可能会有多个有优先级的中断反复发生，必须有相应结构存储每次中断发生时的相应信息。

首先，相比于单级中断发生时的一个 EPC 寄存器，针对此次实验中的三级中断设置 3 个 EPC 寄存器（使用硬件栈），分别存放至多 3 个中断都发生时的 PC 值，这里需要有计数器来存放硬件栈顶，当 ReadyInt ==1 时计数器增加，当 URET==1 时计数器减小。另外也需要比较器和寄存器结合，判断当有中断处理程序在执行时如果发生新的中断，应该执行哪个中断（优先级更高）。具体实现就是将优先编码器得到的中断号 INTNUM 和现在在运行的中断号 OLD_INT 进行比较，不一样就是有更高级别的。可以用一个 IE 控制的多路选择器来让 INTNUM 通过，这样 IE == 0 的时候参与比较的 INTNUM 不会变化，就可以实现屏蔽中断的功能，并且也不会受到时钟的影响。

在多级中断中,还用到了开关中断指令，结合中断发生信号和 URET 指令来进行是否发生中断的判断,也需要修改控制器逻辑使之支持这些指令。中断信号的复位电路和单级中断思路类似。

2.2.3 软件设计

具体设计而言，主要需要考虑保存寄存器的值。通过编写程序来保护现场,即进入中断时将寄存器压入堆栈,并在中断返回前从堆栈中弹出寄存器的值来保护所有在中断处理程序中需要用到的寄存器。

由于 PC 的值已经通过硬件实现的堆栈进行保存,在程序中无需考虑 PC 的压栈。同时，由于使用了自动测试的逻辑，在判断中断时直接使用 RARS 得到中断处理程序的入口地址。

2.3 流水 CPU 设计

2.3.1 总体设计

流水线数据通路路分成 5 个阶段：取指、译码、执行、访存、写回。总体结构如图 2.2 所示。

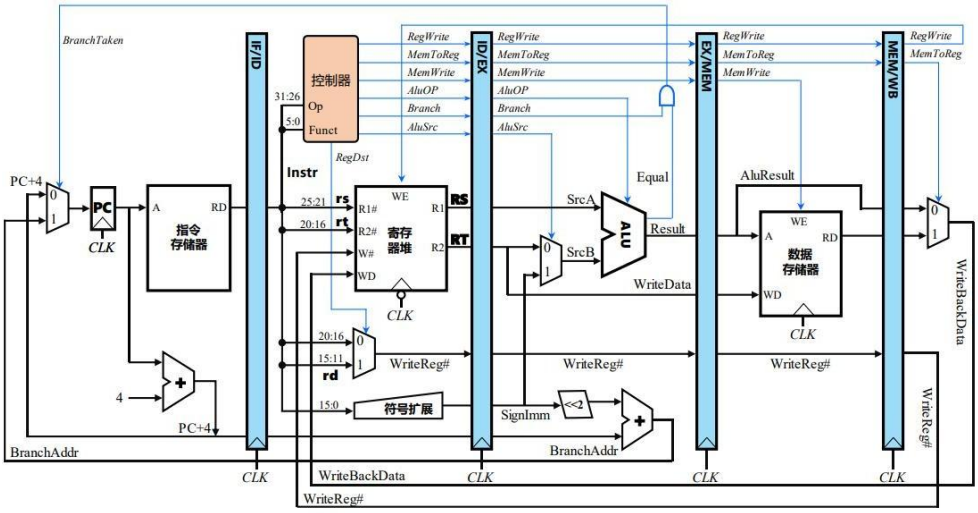


图 2.2 五段流水结构

取指（IF）是从指令寄存器里取得指令；译码（ID）是将指令转换为控制信号，取得一些立即数或者寄存器值；执行（EX）是利用 ALU 进行运算，或者计算一些分支地址；访存（MEM）是读写内存；写回（WB）是将计算结果或者访存结果写回寄存器堆。

针对理想流水线，我们不需要考虑数据冲突以及跳转指令等问题，但理想流水线是后续气泡流水线以及重定向流水线的基础，在理想流水阶段设计的流水部件在后续的流水中也会用到。而后续的气泡则是使用插入气泡的方法解决冲突问题，重定向则是尝试使用重定向的方式解决冲突，减少插入气泡的次数，提高效率。流水的不同段使用流水寄存器进行连接，同时调整各信号。

2.3.2 流水接口部件设计

流水接口部件大部分参考了字库实验中所用到的流水部件，主要区别是接口个数不同。每一个流水部件包含时钟、使能端、清零端、数据输入端以及数据输出端，要传递 JAL 等控制信号、寄存器读出来的值、ALU 和内存的结果、指令中的各种立即数、PC、IR 等内容，还要支持时钟、使能、同步清零、暂停功能。5 段流水需要 4 个流水接口部件。考虑到实现的方便，设计一个通用的流水接口部件可以存储各种信号，然后各阶段之间的流水部件传递需要的信号即可。示例如图 2.3 所示：

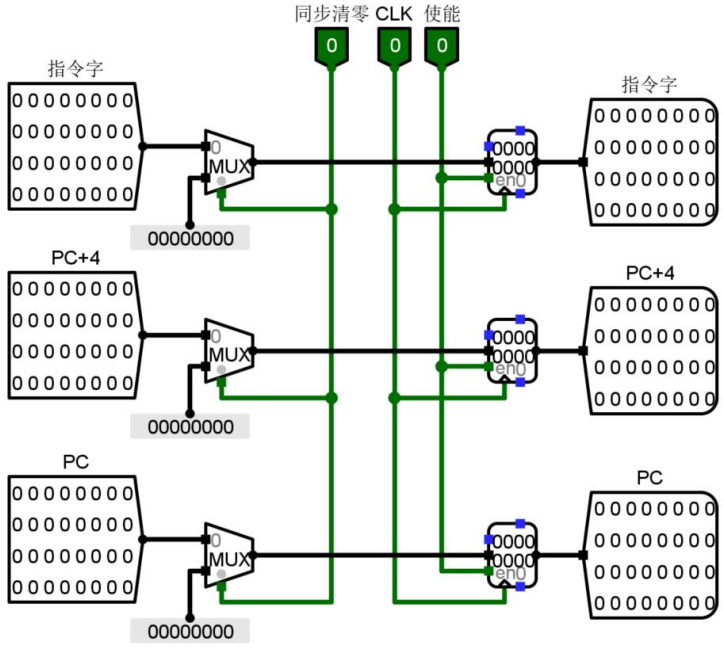


图 2.3 流水寄存器

2.3.3 理想流水线设计

理想流水线中我们假设指令之间没有数据冲突，也执行不了分支指令，所以只需要根据五段流水结构图出总体结构即可，为后面气泡流水线和重定向流水线做准备。

2.4 气泡式流水线设计（EX 段分支）

2.4.1 总体设计

气泡流水线要支持中断的 24 条基本指令和四条个性化指令，会有一些冲突：

1. 控制冲突（冒险）：遇到分支和跳转时，在我们得到下一条指令的地址之前，已经把 PC+4 对应的指令取出了。这样会导致错误的指令进入流水线。可以通过在流水线部件之间插入气泡来解决，即，如果跳转发生，我们将之前的错误指令清除。

2. 结构冲突（冒险）：同一个周期内，两个阶段的指令可能会用到相同硬件。例如把指令存储器和数据存储器合二为一时，IF 和 MEM 同时要从存储器中读取数据，产生了不可调和的冲突。可以通过采用哈佛架构和改善硬件方面来解决，本次流水线不用特别更改。

3. 数据冲突（冒险）：后面的指令需要用到前一个指令的结果产生的冲突。但

由于前一条指令还未写回，导致后一条指令无法读入正确的值。具体分析，可以通过将寄存器堆写入控制特别设置为下降沿触发来解决 ID 和 WB 段发生数据相关；通过暂停 IF/ID 流水线部件和 PC 解决 ID 和 MEM 相关以及 ID 和 EX 相关的情况，比如在 ID/EX 流水线部件插入气泡。

2.5 重定向流水线设计（EX 段分支）

2.5.1 总体设计

主要是解决气泡流水线插入气泡过多导致流水线性能下降的问题，这里修改气泡流水线的逻辑，在发生数据冲突时，不再进行插入气泡的操作，而是在下一个时钟周期将尚未写回至寄存器堆的数据直接重定向至需要数据的 EX 段中，这就相当于接上了一条数据的快速通路，就没有必要产生那么多气泡了。

同时重定向流水线也可加入单级中断，在 EX 段处理中断，当发生中断时通过判断气泡情况来选择正确的断点，并对误取指令进行清空，同时中断返回时也需要清空误取指令。

需要注意的是重定向流水线会引入 Load-Use 问题,即在一条 load 指令之后若紧随有一条需要使用该寄存器的指令,不能直接将数据存储器 DM 的输出重定向到 EX 段中,因为这样会使整个 CPU 的关键路径包含访存,从而关键路径变长影响 CPU 整体的工作效率。为解决此问题,需要在 Forward 模块中添加对 Load-Use 情况进行判断的逻辑,在检测到该情况发生时,向 EX 段添加一个气泡,便能消除 Load-Use 现象[3]。

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 Logisim 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，使能端高电平有效，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作为寄存器的使能端，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，忽略时钟信号，使整个电路停机。如图 3.1 所示。

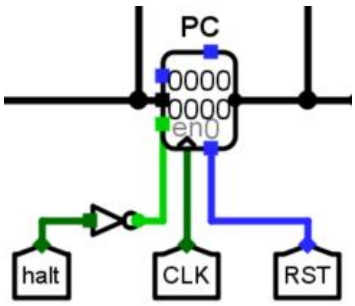


图 3.1 程序计数器（PC）

2) 指令存储器（IM）

① Logisim 实现：

使用一个只读存储器 ROM 实现指令存储器（IM）。设置该只读存储器的地址位宽为 10 位，数据位宽为 32 位。因为 PC 中存储的指令地址有 32 位，而 ROM 地址线宽度有限，仅为 10 位，故将 32 位指令地址高位部分和字节偏移部分直接屏蔽，使用分线器只取 32 位指令地址的 2-11 位作为指令存储器的输入地址。如图 3.2 所示。

图 3.2 指令存储器（IM）

3) 寄存器堆（RF）

① Logisim 实现：

直接使用提供的 RegFile。主体部分是 32 个寄存器，其中 0 号寄存器恒 0，输入设为 0。利用译码器将输入 WAdr 变成对应寄存器的写入信号，与写使能 WE 相与送入寄存器写使能端。寄存器输出通过两个多路选择器，R1Adr 与 R2Adr 分别作为选择端选择输出 R1 与 R2。如图 3.3 所示。

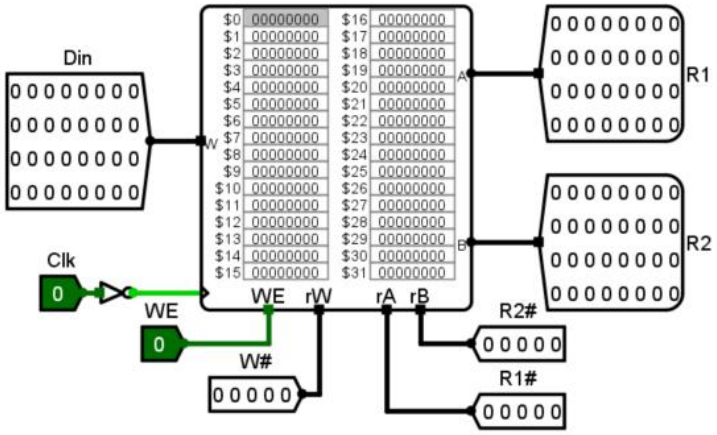


图 3.3 寄存器堆 RF

4) 数据存储器（DM）

① Logisim 实现：

使用提供的 MIPS RAM，由于输入 20 位字地址，故需要将 Alu_result 通过分线器输出 2-21 位并输入 DM，如图 3.4 所示。

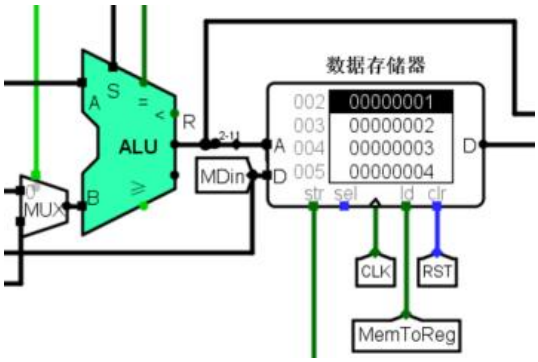


图 3.4 MIPS RAM1

5) 运算器

运算器的实现较为简单，根据表 2.1，并加入多路选择器选择输出结果，即可实现。如图 3.5 所示。

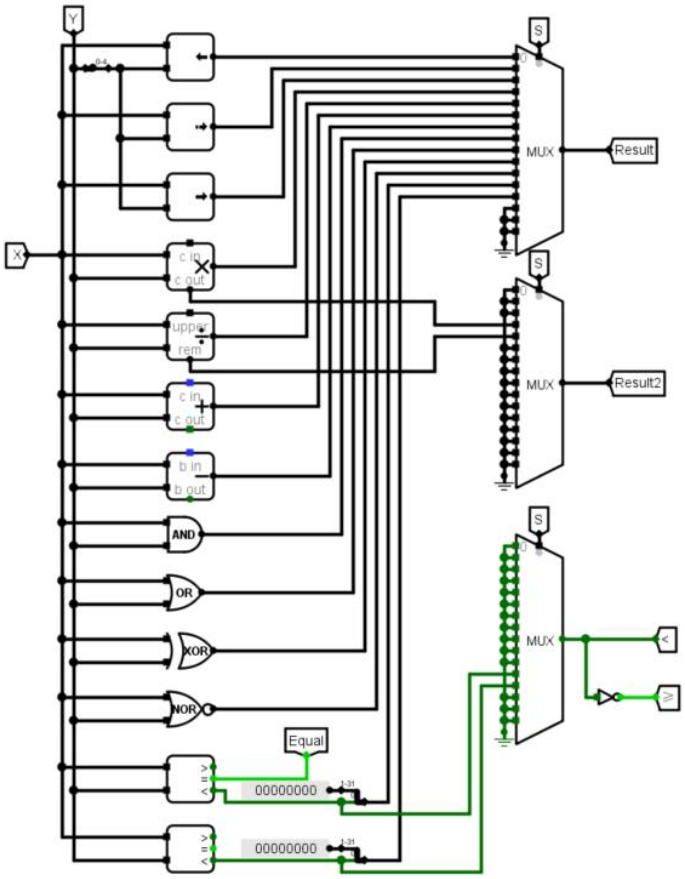


图 3.5 ALU 的Logisim 原理图

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 所示。

表 3.1 指令系统数据通路表

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
ADD	PC+4	PC	rs1	rs2	rd	alu	r1	r2	5		
ADDI	PC+4	PC	rs1		rd	alu	r1	立即数	5		

AND	PC+4	PC	rs1	rs2	rd	alu	r1	r2	7		
ANDI	PC+4	PC	rs1		rd	alu	r1	立即数	7		
SLLI	PC+4	PC	rs1		rd	alu	r1	立即数	0		
SRAI	PC+4	PC	rs1		rd	alu	r1	立即数	1		
SUB	PC+4	PC	rs1	rs2	rd	alu	r1	r2	6		
OR	PC+4	PC	rs1	rs2	rd	alu	r1	r2	8		
ORI	PC+4	PC	rs1		rd	alu	r1	立即数	8		
XORI	PC+4	PC	rs1		rd	alu	r1	立即数	9		
LW	PC+4	PC	rs1		rd	DM_out	r1	立即数	5	alu	
SW	PC+4	PC	rs1				r1	立即数	5	alu	r2
BEQ	PC+offset1	PC	rs1	rs2			r1	r2	6		
BNE	PC+offset1	PC	rs1	rs2			r1	r2	6		
SLT	PC+4	PC	rs1	rs2	rd	alu	r1	r2	11		
SLTI	PC+4	PC	rs1		rd	alu	r1	立即数	11		
SLTU	PC+4	PC	rs1	rs2	rd	alu	r1	r2	12		
JAL	PC+offset2	PC			rd	PC+4					
JALR	(PC+offset3)&~1	PC	rs1		rd	PC+4	r1	立即数	5		
ECALL	PC+4	PC									
CSRRSI	PC+4	PC			rd	t					
CSRRCI	PC+4	PC			rd	t					
URET	PC+4	PC									
AUIPC	PC+4	PC			rd	PC+imm					

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，在 Logisim 中搭建的 24 条基本指令的数据通路的原理图如图 3.6 所示。

由于实验采用简单迭代法实现单周期处理器，故可在原电路基础上直接加上四条指令 CCAB 指令，新的数据通路的原理图如图 3.7 所示。

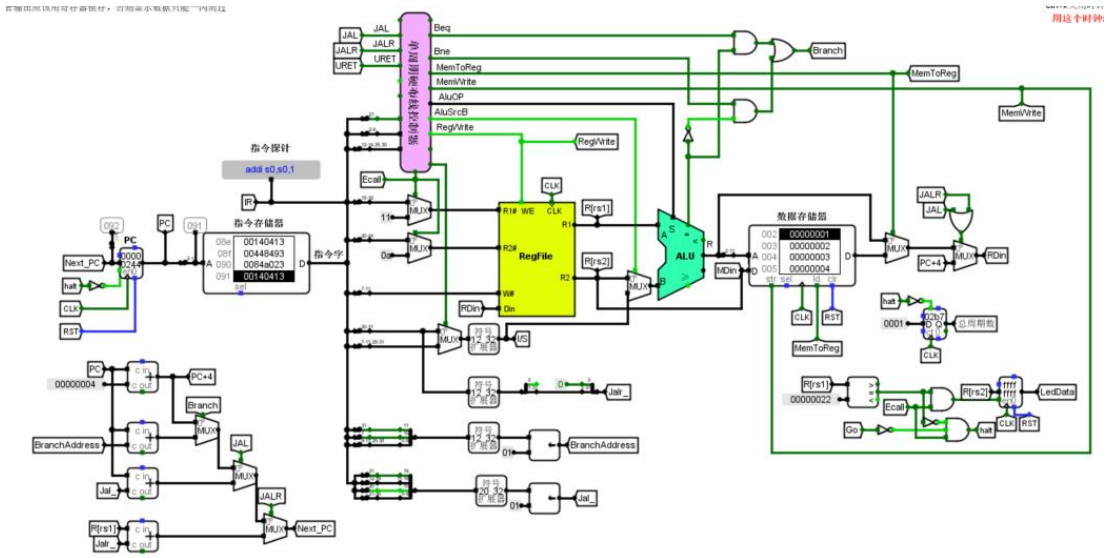


图 3.6 24 指令单周期 CPU 数据通路（Logisim）

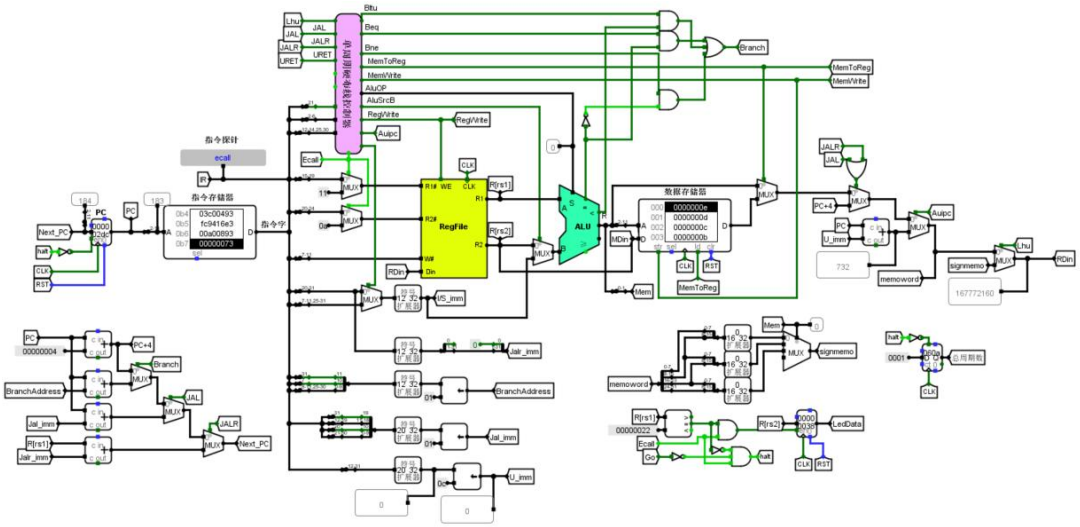


图 3.7 24+4 指令单周期 CPU 数据通路（Logisim）

3.1.3 控制器的实现

根据总体方案设计中控制器的设计那一小节的相关内容，根据表 2.3，将运算控制器生成电路与控制信号生成电路的 Excel 表格填写完整，自动生成电路。Excel 表格即相关表达式如图 3.8 所示，外部封装如图 3.9 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	Readrs1	Readrs2	BLTU	LHU	AUIPC	CSRRSI	CSRRCI
1	add	0	0	c	5				1							1	1					
2	sub	32	0	c	6				1							1	1					
3	and	0	7	c	7				1							1	1					
4	or	0	6	c	8				1							1	1					
5	slt	0	2	c	11				1							1	1					
6	sltu	0	3	c	12				1							1	1					
7	addi		0	4	5			1	1							1						
8	andi		7	4	7			1	1							1						
9	ori		6	4	8			1	1							1						
10	xori		4	4	9			1	1							1						
11	slti		2	4	11			1	1							1						
12	slli	0	1	4	0			1	1							1						
13	srli	0	5	4	2			1	1							1						
14	srai	32	5	4	1			1	1							1						
15	lw		2	0	5	1		1	1							1						
16	sw		2	8	5		1	1			1					1	1					
17	ecall	0	0	1c						1						1	1					
18	beq		0	18								1				1	1					
19	bne		1	18									1			1	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	0	0	1c																		
25	sll	0	1	c	9				1							1	1					
26	auipc			5					1											1		
27	lhu		5	0		1		1	1							1			1			
28	bltu		6	18	12											1	1	1				

图 3.8 单周期硬布线控制器表达式

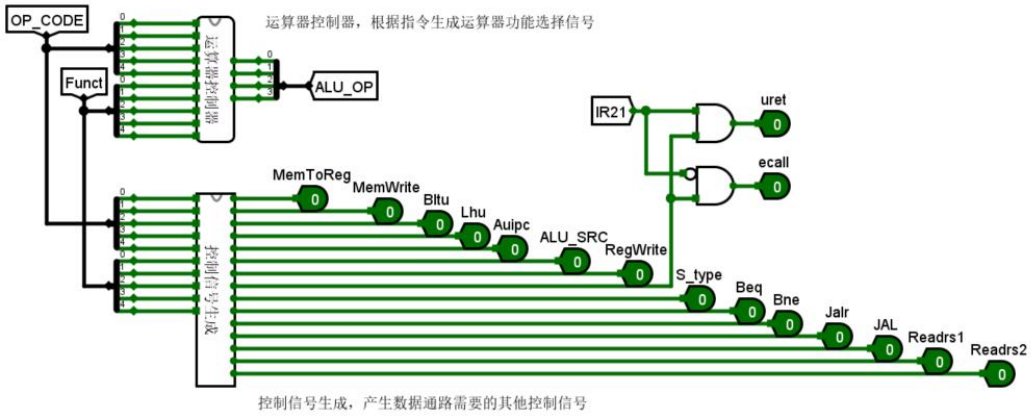


图 3.9 单周期硬布线控制器外部封装

3.2 中断机制实现

3.2.1 单级中断实现

在实现单级中断时，中断按键信号产生电路使用参考电路即可；

对于中断地址选择电路，因为有三个不同的中断,在一个中断发生时要根据此时具体的中断选择中断处理程序地址，使用优先编码器决定目前执行的中断号，利用多路选择器选择出地址。由于测试程序中直接给出了中断地址，所以被硬编码到电路中，如图 3.10 所示。

对于中断信号复位电路，当中断处理完毕后,要将中断信号复位,具体根据 URET 指令结合当前执行中断的中断号判断，将具体的某一个中断信号复位。其中

中断号寄存器的使能端输入为是否有中断发生的信号,当有中断发生时不进行复位,不然中断信号一产生就被复位，如图 3.11 所示。

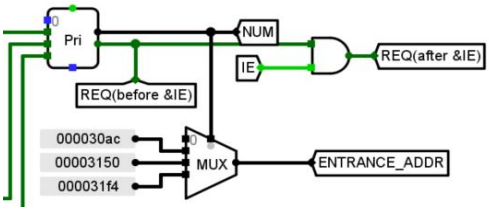


图 3.10 中断处理程序地址选择电路

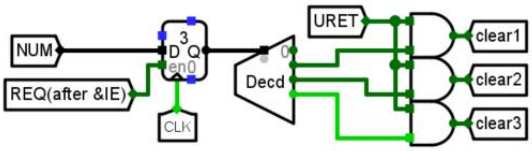


图 3.11 中断信号复位电路

对于 EPC 寄存器及中断使能 IE 寄存器电路，在有中断发生时要将 PC 寄存器的值保存起来，在中断结束时取出。同时 IE 寄存器保存当前中断的相关状态，以方便判断是否有中断发生，即在有中断发生或者中断处理程序完成返回时更新 IE 的值。

对于 PC 输入电路，在 PC 输入端,一方面要在中断发生时选择合适的值送入 EPC 中，另一方面要选择合适的值送入 PC。

综上我们设计出单级中断机制的数据通路，如图 3.12 所示。

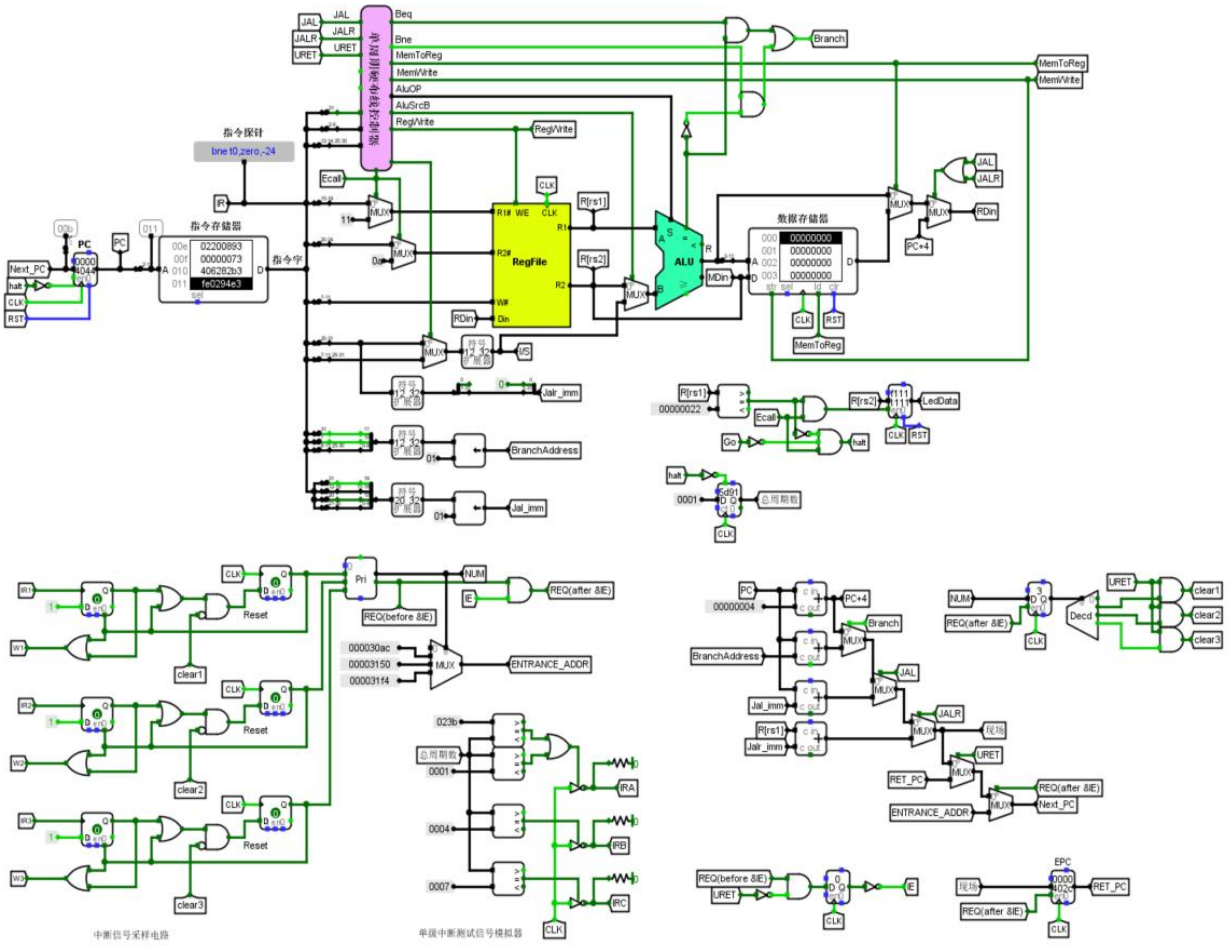


图 3.12 单级中断数据通路

3.2.2 多级中断实现

多级中断机制的一些包含主要功能的子电路数据通路如图 3.13 所示。

同理，中断按键参考电路中包含了 IR，可直接使用，产生的中断请求通过优先编码器，得到最高中断号，IE 信号是中断开关信号，只有当IE 为1 且有中断请求时才会产生中断信号。提前获取三个中断服务程序的入口地址，利用中断号对服务地址进行选择，与单级中断相似。然而由于优先编码器选择出的中断号并不一定是当前运行的中断号，所以此处新添加一个当前中断号，只有当前中断号下有中断信号中断才会有效，用寄存器保存有效中断号，并通过优先编码器，中断号越大越优先。注意此处清零信号要经过一个触发器寄存。

由于一些具体思路设计在总体设计部分谈过，这里只给出一些简略的解释。

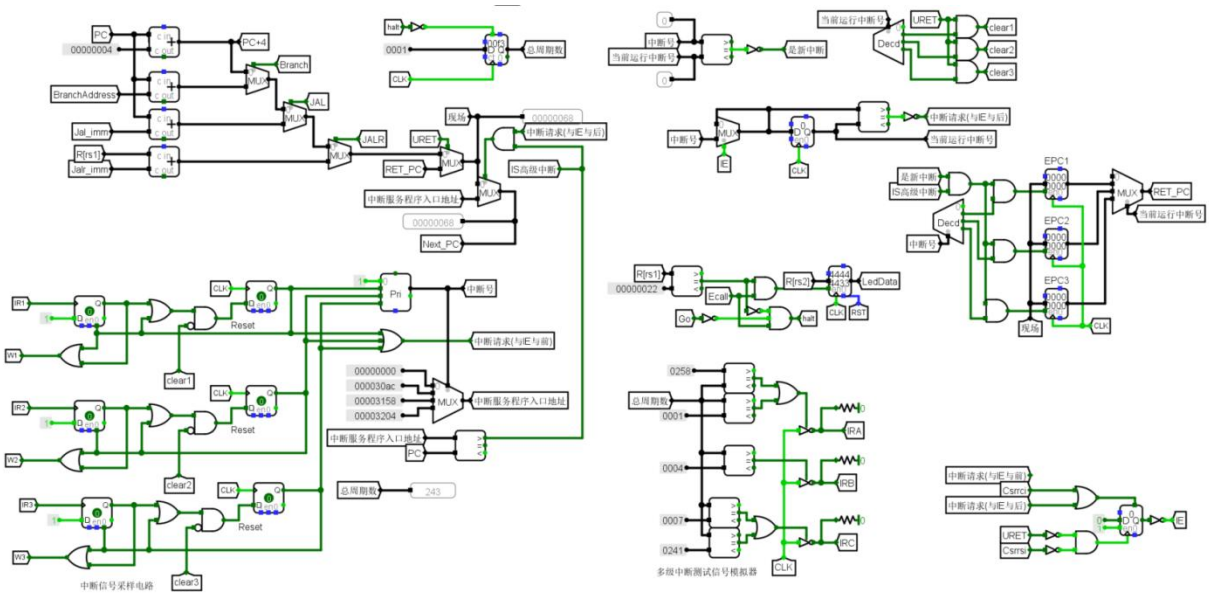


图 3.13 多级中断数据通路（部分）

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水接口负责通过时钟输入在不同的执行阶段之间传递数据，进行缓存，并能够进行插入气泡以及清零等基本操作。各个流水寄存器的实现大同小异，主要是所传递的信号和数据有所不同，具体的实现参考了字库实验中所用到的流水部件。

实验中不同的接口部件对应 CPU 五段操作之间的传递过程。这里给出不同阶段流水寄存器所传递的信号和数据，如表 3.2 所示。

表 3. 2 不同阶段流水寄存器所传递的信号和数据

信号	数据
IF\ID	PredictJump(动态分支预测）、IR、PC、PC+4
ID\EX	PredictJump(动态分支预测)、beq、bne、memtoreg、memwrite、alusrcb、regwrite、uret、ecall、jal、jalr、auipc、bgeu、sb、aluop、rd、imm、R1、R2、IR、PC、PC+4、Rs1Forward(动态分支预测)、Rs2Forward(动态分支预测)
EX\MEM	memtoreg、memwrite、regwrite、halt、ecall、jal、jalr、auipc、uret、sb、rd、imm、R2、IR、PC、PC+4、result
MEM\WB	memtoreg、regwrite、halt、ecall、jal、jalr、auipc、uret、sb、rd、imm、R2、IR、PC、PC+4、result、data

每个流水接口的内部布线，不仅包括了由控制器给出的控制信号，同时也包括了个人拓展指令所对应的指令信号。

3.3.2 理想流水线实现

根据总体设计部分中理想流水线的设计解析，向单周期 CPU 中新增流水接口，并对单周期数据通路修改连接使其成为流水线数据通路。重新连接数据通路后的理想流水线 Logisim 电路图如图 3.14 所示。

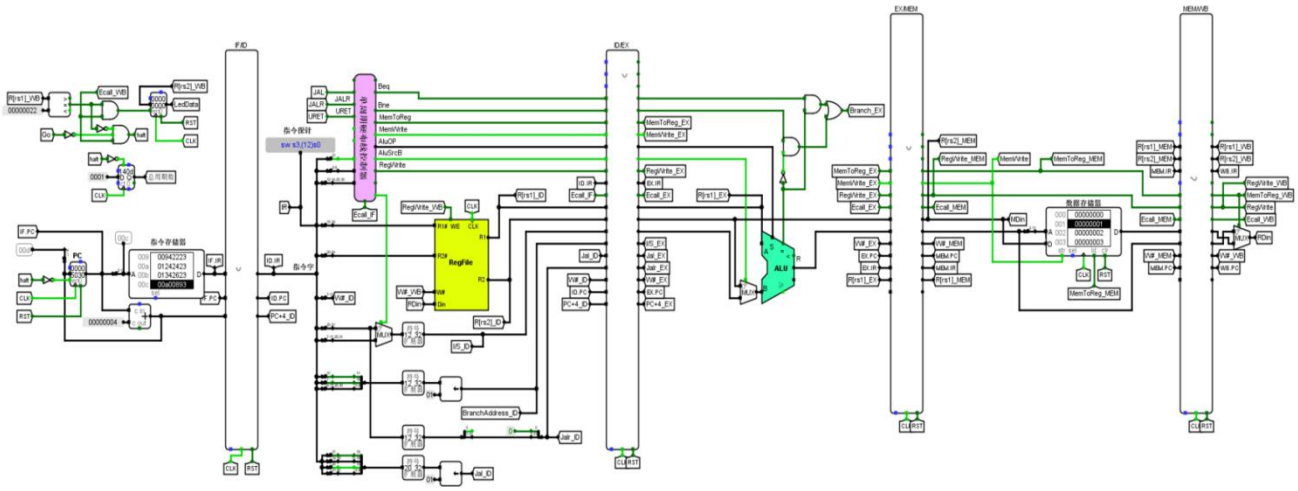


图 3. 14 理想流水线

3.4 气泡式流水线实现

3.4.1 数据相关信号实现

我在实验中将数据相关逻辑封装，同时进行了源寄存器冲突以及数据冲突两种情况。分析可得流水线中的数据相关检测逻辑如下：

DataHazzard
= Rs1Used & (rs1≠0)
& EX.RegWrite & (rs1 == EX.WriteReg#)
+ Rs2Used & (rs2≠0)
& EX.RegWrite & (rs2 == EX.WriteReg#)
+ Rs1Used & (rs1≠0)
& MEM.RegWrite & (rs1 == MEM.WriteReg#)
+ Rs2Used & (rs2≠0)
& MEM.RegWrite & (rs2 == MEM.WriteReg#)
+ ecall & EX.RegWrite
& ((EX.WriteReg# == 11) | (EX.WriteReg# == 0a))
+ ecall & MEM.RegWrite
& ((MEM.WriteReg# == 11) | (MEM.WriteReg# == 0a))

上述表达式中的 Rs1Used 和 Rs2Used 表示该指令是否用到了 Rs1 或者 Rs2，说明在实现气泡发生器后，为了让整个流水线正常的工作，需要通过外部手动生成（改造控制器）得到相关信号，以避免在实现重定向 CPU 时出现问题。

3.4.2 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 3.15 所示。相较于理想流水线，气泡流水线能够处理全部的 28 条指令，并能够正确的处理发生的所有冲突。

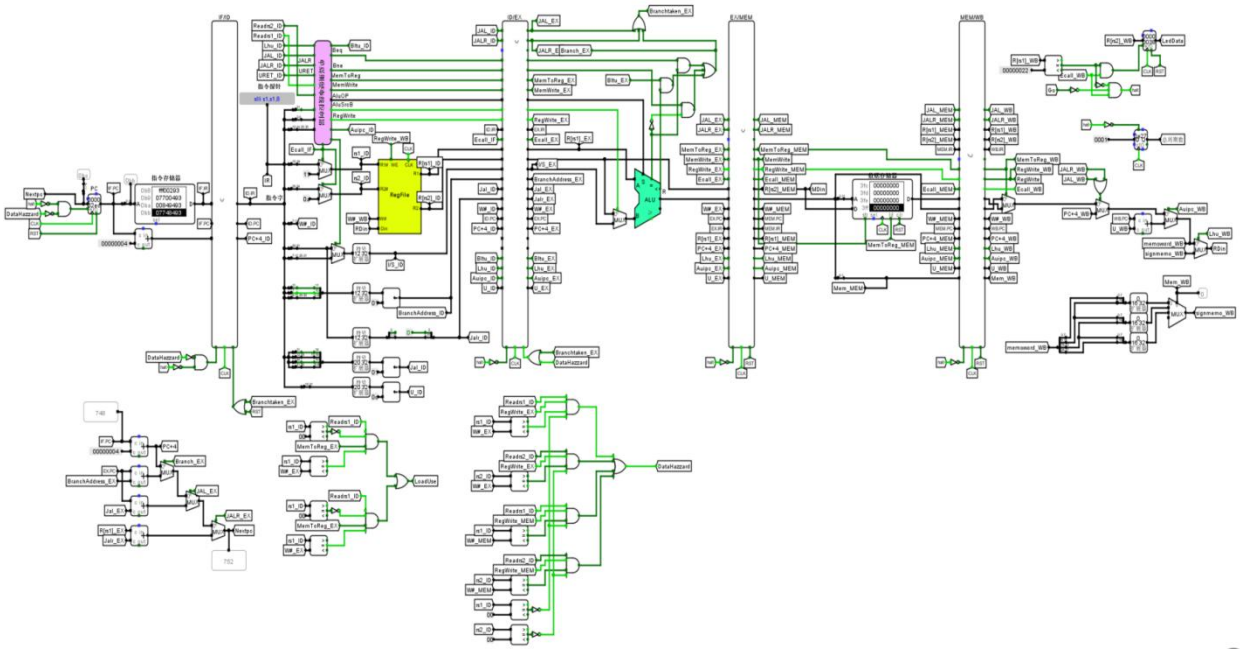


图 3.15 气泡流水线

3.5 重定向流水线实现

完成了气泡流水线后，为了减少大量插入气泡带来的性能损失，需要将其改造为重定向流水线。而首先需要实现的就是重定向模块。重定向模块从不同段读取数据并判断，不仅要算出有没有数据冲突，还要知道 R1、R2 和 EX、MEM 之间哪个阶段冲突。同时，如果 R1 或者 R2 和EX 冲突，并且 EX 阶段是加载型指令，就说明是 Load-Use 情况，产生对应的信号。

完成了重定向模块的实现以及气泡流水线的改造后，需要将其整合进入气泡流水线中，使之成为重定向流水线。改造完成后的重定向流水线如图 3.16 所示。

在完成了重定向模块后，气泡流水线的功能就被限制在处理 Load-use 冲突了。

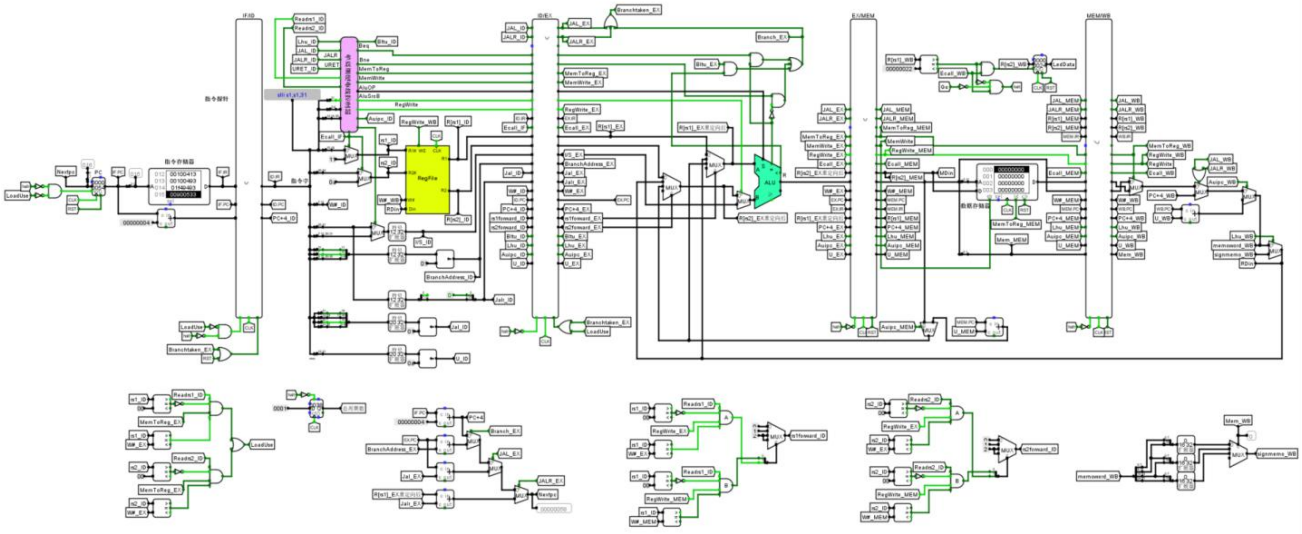


图 3.16 重定向流水线

4 实验过程与调试

4.1 测试用例和功能测试

本次课设我做了单周期 CPU、理想流水线、EX 段分支气泡流水线、EX 段分支重定向流水线、单级中断、多级中断六个模块，已经通过 Educoder 上测试和线下 CCAB 检查，加之有报告页数的限制，这里只是简要说明一些测试情况。

备注：单周期 CPU、气泡流水线以及重定向流水线的测试用例相同，所展示效果基本相同，这里只对该测试用例的效果进行展示。单级中断与多级嵌套中断的效果单独展示。

4.1.1 测试用例 1

单周期 CPU、气泡流水线以及重定向流水线的常规测试用例使用的是 risc-v-benchmack_ccab.hex，部分结果相同，只是时钟周期有区别，在性能分析一节会做报告。

加载测试程序，打开自动仿真，统计系统总周期数计数、LED 显示区内容、数据存储器中的数据分布情况等，部分输出与常规测试结束之后的数据存储器内容分别如图 4.1、4.2 所示。

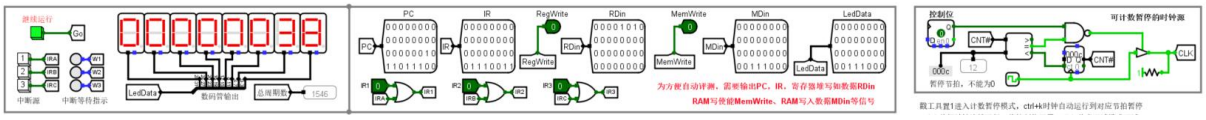


图 4.1 常规测试结果展示（单周期 CPU）

```
00000 0000000e 0000000d 0000000c 0000000b  0000000a 00000009 00000008 00000007  00000006 00000005 00000004 00000003  00000002 00000001 00000000 ffffffff
00010 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
```

图 4.2 risc-v-benchmack_ccab.hex 部分结果展示

关于 CCAB 指令，分别为 SLL、AUIPC、LHU、BLTU。将四条指令对应的汇编文件加入 benchmark.asm 结尾，重新加载测试程序并运行测试，调节时钟频率并仔细观察 LED 显示屏上的输出内容并与答案相对照，此处只列出四条指令的最后输出：0x00000000、0x00430074、0x0000c0bf、0x00000000，可验证得输出无误。

4.1.2 测试用例 2

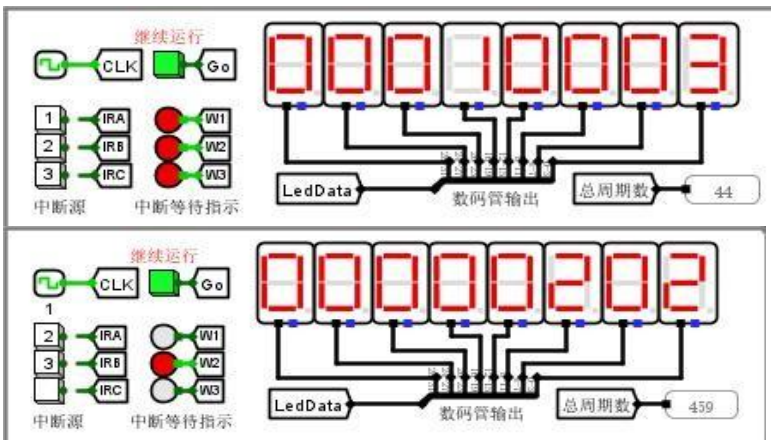
(1) 单级中断测试

在指令存储器中加载.hex 测试程序，启动时钟自动运行，观察主程序走马灯运行状态，在主程序运行时任意点击 1、2、3 号按键中的任何一个，对应中断指示灯应该点亮，CPU 应正常进入中断演示程序执行，中断演示程序执行能正确返回主程序，同时中断指示灯熄灭。如中断服务程序执行期间又有新的按键被按下，中断指示灯点亮，当前中断服务程序结束时可入新的中断服务程序响应。本次实验中，以相对较快的速度依次点击 1、2、3 号中断源按键，能正常响应中断（会一直执行当前在运行的中断）；

(2) 多重嵌套中断测试

同样加载测试程序，观察走马灯，在主程序运行时任意点击 1、2、3 号按键中的任何一个，对应中断指示灯应该点亮，CPU 应正常进入中 25 断演示程序执行，中断演示程序执行能正确返回主程序，同时中断指示灯熄灭。如中断服务程序执行期间又有新的按键被按下，中断指示灯点亮，高优先级中断服务程序能打断低优先级中断服务程序，中断后应返回低优先中断服务程序直至主程序。这里同样以较快的速度依次点击 1、2、3 号中断按键，先后进入1 - 2 - 3 - 2 - 1 - CPU；依次点击 2、3、1 号中断按键，先后进入 2 - 3 - 2 - 1 - CPU，相应中断无误。

测试中可得，当处理中断时，走马灯的前七位从左到右循环播放当前处理的中断号，最后一位是当前剩余的循环次数，部分截图如图 4.3 所示。



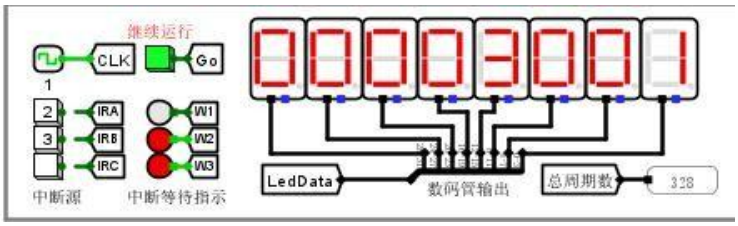


图 4.3 单/多级中断效果展示

4.2 性能分析

当测试环境不计入 CCAB 差异化，对于使用相同测试用例的单周期 CPU、气泡流水线、重定向流水线，程序执行完毕时的周期数如 4.1 所示，其中气泡流水线结果符合“总周期数 = 1546 + 4 + 气泡数目 + 分支误取深度 * 分支数 - 1”的要求；重定向流水线符合“总周期数 = 1546 + 4 + 分支误取深度 * 分支数 + load-Use 次数”的要求（这里注意因 ECALL 实现的不同，总周期数可能和标准答案有很小的差异）。

表 4.1 各CPU 的总周期数

项目	总周期数
单周期 CPU	1546
气泡流水线	3624
重定向流水线	2298

其中单周期 CPU 的时钟周期最短（就是指令数 1546），但是其每一个周期的时间最长，整体来说效率最差；

气泡流水线的周期数最高，因为相邻指令之间经常会出现冲突，且只能使用插入气泡的方式来解决，正常的程序中必然存在大量的相关，也就导致其周期数大幅增加，但总的来说相较于单周期硬布线效率还是更高，5 段流水在理想情况下可以提供 5 倍的的运行速度；

重定向流水线能很好地降低总周期数，只是在分支和访存的时候还是有一些气泡。

4.3 主要故障与调试

4.3.1 跳转故障

多级中断：中断执行时跳转地址有误。

故障现象：故障由头歌平台反馈得到。在周期数临近时分别按下 1、2、3 号中断，对于多级中断来说，理论上是应该在 1 号中断执行开中断时进入 3 号中断，3 号中断结束后执行 2 号中断，2 号中断结束后返回 1 号中断。这时 1 号中断已经发生，应该从 1 号中断的中断服务程序的某一条指令开始继续执行，但却从 1 号中断服务程序的开始处执行。

原因分析：2 号中断结束返回 1 号时，电路不能区分 1 号中断是否执行过，只是由于此时满足了发生中断的条件，直接跳转到了 1 号中断处理程序的开始。

解决方案：利用 3 个 mEPC 存储的断点判断是否有低级中断执行过，如果 2 号或者 3 号中断的断点在某区间 A 内（面向头歌平台测试编程），说明 1 号中断的中断服务程序已经执行（hasrun1=1）；如果 3 号中断的断点在某区间 B 内，说明 2 号中断的中断服务程序已经执行（hasrun2=1）；同时限定，如果当前中断号是 1 或 2 且 hasrun1/hasrun2 = 0，才可以产生发生中断信号。

4.3.2 气泡流水线故障

气泡流水线：流水线暂停故障。

故障现象：加载CCAB 相关的benchmark 程序以后，尽管已暂停，后面的流水线还是会不断读取前面的流水线的内容。

原因分析：流水线组件暂停功能不完善，只考虑了和 ID 相关的组件，EX\MEM 以及 MEM\WB 组件的暂停功能都有缺陷。

解决方案：四个流水线组件统一采用 halt 信号控制暂停的方案。

4.3.3 重定向故障

重定向流水线：数据重定向问题。

故障现象：重定向流水线导致排序数据错误。

原因分析：重定向时只将写入 Alu 的寄存器的值进行了重定向处理，而忘记考虑传入 PC 以及 Ecall 的值，导致部分跳转指令出错。

解决方案：将从 ID/EX 引出来的寄存器的值直接进行重定向，此后所有需要用到 EX 段寄存器的模块全部使用重定向以后的值，如图 4.4 所示。

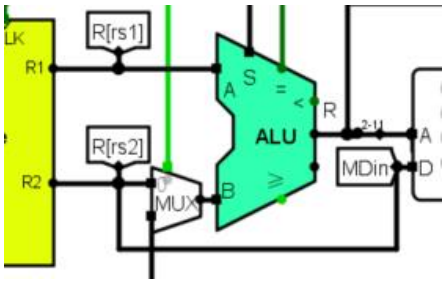


图 4.4 修改后的正确的重定向数据通路

4.4 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	复习组成原理 CPU 设计部分，学习流水线章节，学习慕课，阅读任务书和 RISC-V 指令手册。
第二天	完成 Logisim 单周期 CPU 的控制信号表，实现了包括 CCAB 的单周期 CPU 并且通过了测试。
第三、四天	继续学习流水线，了解理想流水线的设计思路，进行通用流水线部件的设计，完成理想流水线并通过测试。
第五、六天	看文档以及搜集资料，了解流水线中冲突的解决方法，完成气泡流水线并通过测试。完善单周期 CPU 和气泡流水线的暂停功能。
第七天	学习课本，了解重定向机制，完成重定向流水线并通过测试。
第八天	了解中断机制，仔细研究中断的时钟机制，完成单级中断和多级中断并通过测试。
第九天	调试解决流水线以及中断中 CCAB 指令的 bug，学习动态分支预测的相关内容，使用 Logisim 实现重定向流水线动态分支预测，未果。

5 设计总结与心得

5.1 课设总结

本次课设通过循序渐进的方式完成了支持 RISC-V 指令集的从单周期 CPU 到功能完善的多级 CPU 的实现，在这个过程中具体做了如下的工作：

1） 设计了单周期 RISC-V 版本 CPU 的数据通路，完成了单周期 CPU 基础指令和 CCAB 指令的电路；

2） 设计了能够接收、存储、传递信号的流水线部件，实现了用流水部件简单连接 CPU的各个部分形成的理想流水线；

3） 实现了气泡流水线中跳转逻辑的添加，实现了加气泡解决结构冲突、加气泡和暂停解决数据冲突的电路；

4） 设计了数据重定向方案，解决了重定向流水线中的各个阶段的数据冲突的问题，完成了重定向流水线；

5） 完成了单级和多级中断机制，设计了处理中断的基本逻辑，实现了 PC 的获取和注入，实现了保存返回地址的硬件栈。

5.2 课设心得

在本次课程设计中，通过两个星期的工作实现了一个简易 CPU，难度适中，但从中可以学习到很多东西。

有了上学期的组原实验中单周期 CPU 的基础，本次实验开始时做单周期 CPU 时完成起来还是比较顺利的，针对后续的流水线等电路也有相关的文档。不过从流水线的难度开始上升，一方面是流水部件的设计，另一方面是需要将自己的 4 条拓展指令融入进去；至于中断，在实现多级中断时需要考虑很多问题，比如对中断发生时发起另一个中断时的处理以及恢复时的先后顺序问题等，的确算是一个比较艰辛但是快乐的过程。

这里不得不提到两个好助手。首先就是 Educoder 平台，不论之前评价如何，它在这次课设中的确给予了我很大的帮助，Educoder 的在线评测可以非常准确的给出

是否正确的判断，并对错误的地方给出标注，极大推动了我的进度。再一个就是漫长的读文档和读文档，虽然之前有接触过 RISC-V，但是本次实验中还是出现了不少奇怪的 bug，不得不说文档中的一些有趣且有用的特性给予了我莫大的启发。

个人认为本次实验的出众之处是采用软件工程中的演化模型对于 CPU 进行开发，可以使人清晰的认识到单周期 CPU 转变为流水线 CPU 并一步步进行功能的增加、性能的优化这一过程，并能够清晰的了解到 CPU 各个部件之间的关系。这里也不得不提一嘴 Verilog。的确从 Logisim 到 Verilog 的映射也能够使人感受到从直观到抽象的设计过程：Logisim 直观但是设计所需的工 作量大，Verilog 虽然抽象但能够快速的实现大量的逻辑。也正是因为这些特性，才使得需要在进行 Logisim 实现时考虑如何使逻辑达到最简化以减少工作量，同时简化的逻辑也能够提升 CPU 的性能。遗憾的是由于个人时间原因（身体原因以及一些线上实习）只是完成了 Logisim 的相关内容。

然而，对于本次课程设计，我也有一些小小的建议和改进：在除了使用一个标准的 benchmark 程序对于 CPU 进行测试之外，还可以让同学们自己编写程序实现一些有意思的功能，如原 CS3410 课程中提供了一个 LCD 模块用于绘图。个人之前有一定的 STM32 和 ESP32 的开发经验，想必在本次实验提供的开发板上也会有串口、VGA 等各种可供扩展的模块，如果能够整合进入一种或多种这种模块并让同学门自己编写一段程序的话理论上想必可以增加不少课程的趣味性。

最后也感觉各位老师在疫情当中依然坚持为我们答疑解惑，并且提供了如此优质的课程内容！

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: