

华中科技大学

2023

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：计卓 2101

学号：U202115285

姓名：高僖

电话：18518472108

邮件：761768812@qq.com

# 华中科技大学课程设计报告

---

## 目 录

<b>1 课程设计概述 .....</b>	<b>3</b>
1.1 课设目的 .....	3
1.2 设计任务 .....	3
1.3 设计要求 .....	3
1.4 技术指标 .....	4
<b>2 总体方案设计 .....</b>	<b>6</b>
2.1 单周期 CPU 设计 .....	6
2.2 中断机制设计 .....	12
2.3 流水 CPU 设计 .....	15
2.4 气泡式流水线设计 .....	18
2.5 重定向流水线设计 .....	19
2.6 动态分支预测机制 .....	20
<b>3 详细设计与实现.....</b>	<b>22</b>
3.1 单周期 CPU 实现 .....	22
3.2 中断机制实现 .....	26
3.3 流水 CPU 实现 .....	29
3.4 气泡式流水线实现 .....	31
3.5 重定向流水线实现 .....	31
3.6 动态分支预测机制实现.....	32
<b>4 实验过程与调试.....</b>	<b>34</b>
4.1 测试用例和功能测试 .....	34
4.2 性能分析 .....	37
4.3 主要故障与调试 .....	38
4.4 实验进度 .....	39

# 华中科技大学课程设计报告

---

<b>5 团队任务</b> .....	<b>40</b>
5.1 选题与设计 .....	40
5.2 团队任务负责部分 .....	40
5.3 实现效果 .....	40
<b>6 设计总结与心得</b> .....	<b>41</b>
6.1 课设总结 .....	41
6.2 课设心得 .....	41
<b>参考文献</b> .....	<b>43</b>

## 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及实现能力的培养”。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；

# 华中科技大学课程设计报告

- (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 27 条基本 32 位 MIPS 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	指令格式与功能参考 RISC-V32 指令集英文手册，或参考 RARS 模拟器。
2	ADDI	立即数加	
3	AND	与	
4	ANDI	立即数与	
5	SLLI	逻辑左移	
6	SRAI	算数右移	
7	SRLI	逻辑右移	
8	SUB	减	
9	OR	或	
10	ORI	立即数或	
11	XORI	或非	
12	LW	加载字	
13	SW	存字	

# 华中科技大学课程设计报告

#	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	
21	ECALL	系统调用	if(\$a7==34)LED 输出\$a0 的值 else 等待 Go 按键暂停
22	CSRRSI	访问 CSR 寄存器	中断相关，可简化为开中断
23	CSRRCI	访问 CSR 寄存器	中断相关，可简化为开中断
24	URET	中断返回	清中断，mEPC 送 PC，开中断
28	SRA	算术右移	扩展运算指令 1
29	SLTIU	无符号数比较	扩展运算指令 2
30	LBU	无符号字节加载	扩展存储访问指令 1
31	BGEU	无符号大于等于分支跳转	扩展跳转指令 1

## 2 总体方案设计

### 2.1 单周期 CPU 设计

在单周期 CPU 设计中，我们采用硬布线控制的方式来协调各部件的工作，确保各部件在时钟同步的情况下协同工作。为了避免结构冲突，我们选择了将指令存放位置与数据存放位置分开的方案，即指令存储器和数据存储器不共用一个存储器，从而有效地避免了冲突的发生。总体结构图如图 2.1 所示，可以划分为五大结构，包括取指结构、译码结构、执行结构、访存结构和写回结构。

在取指结构中，PC 寄存器提供指令存储器的取指地址。在译码结构中，单周期硬布线控制器通过 (OP, Funct) 字段实现对指令的译码，并输出相应的总控制信号。通过分线器获取寄存器和立即数的相关信息，并通过寄存器堆获取寄存器数据。在执行结构中，ALU 运算器负责执行指令所需的计算操作。访存结构中，数据存储器在控制信号的控制下进行数据读取和存储。最后，在写回结构中，实现寄存器写回操作。

这种结构的设计确保了各个阶段之间的协同工作，使得指令的取指、译码、执行、访存和写回等操作能够顺利进行。通过明确的总控制信号和有效的数据通路设计，我们确保了每个阶段的功能正确实现，从而保障了 CPU 的整体正常运行。总体结构图如图 2.1 所示。

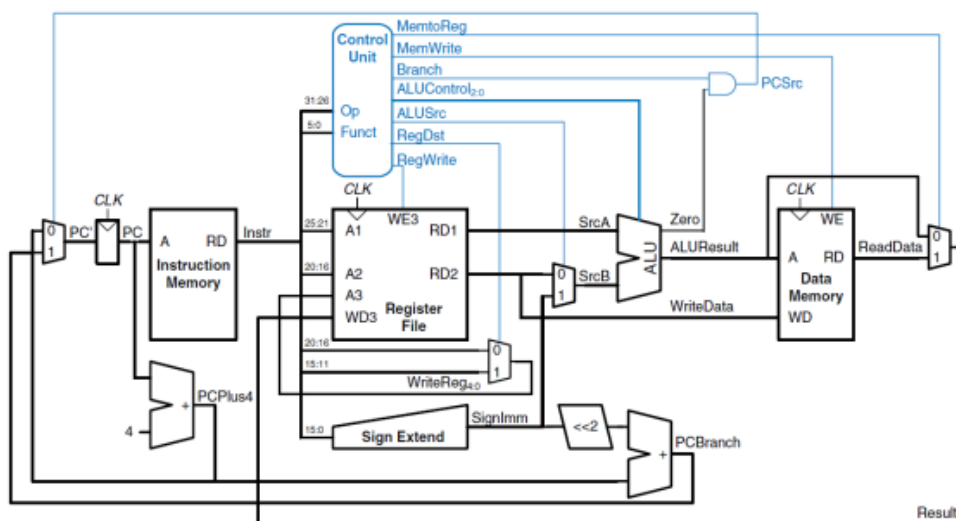


图 2.1 总体结构图

## 2.1.1 主要功能部件

### 1. 程序计数器 PC

在计算机体系结构中，程序计数器（Program Counter，缩写为 PC）扮演着至关重要的角色，作为一个关键寄存器，其主要功能是存储下一条待执行指令的地址。PC 寄存器中保存着当前正在执行的指令的地址，或者下一条即将执行指令的地址，它指导着处理器应该从内存中的哪个地址获取下一条指令。一旦一条指令执行完毕，PC 会在时钟的上升沿触发下一条指令的地址递增，以确保指令的有序执行和程序的正常控制。

在 LOGISIM 中，我们采用自带的数据位宽为 32 的寄存器实现程序计数器。这种设计确保了对 32 位地址的有效存储和处理。同时，我们使用上升沿进行触发，以保证在时钟信号上升时 PC 的值被更新，从而实现同步的指令执行。

为了更好地配合其他程序工作，我们引入了使能端和清零端。使能端用于控制 PC 是否接受新的地址，从而实现对指令执行的灵活控制。清零端用于在需要时将 PC 寄存器清零，例如在程序启动时或者发生特定条件时。这样的设计增强了 PC 的灵活性和可控性，使其更好地适应不同的应用场景。

### 2. 指令存储器 IM

对于指令存储器（Instruction Memory，缩写为 IM），我们了解到它存储了 CPU 将要执行的程序段的二进制文件。该存储器是一个关键的组成部分，通过输入相应的地址值，可以取出存储在该地址处的指令。对于 RISC-V32 指令集，指令长度为定长 32 位，因此我们将数据位宽设置为 32 位。

对于字节地址和指令长度的关系，需要注意到一条指令有四个字节。由于程序计数器（PC）输出的是字节地址，我们需要以字寻址方式获取指令。因此，我们可以将 PC 的最低两位截断，以获取相应的字地址。

考虑到设计的 CPU 并不过于复杂，我们决定采用 10 位字地址。这个决策是基于实验需求和系统的设计特点，确保了足够的寻址范围，同时保持了系统的简单性。因此，我们可以使用 PC 的 2 到 11 位作为 IM 的输入，以获取正确的字地址，从而实现对指令存储器的有效寻址。



# 华中科技大学课程设计报告

## 3. 运算器

ALU（算术逻辑单元）在计算机中扮演着至关重要的角色，它是一个关键组件，负责执行各种算术和逻辑运算，以处理数据和执行指令。ALU 的功能涵盖了执行算术运算、逻辑运算、位移操作、条件分支判断、数据操作等多个方面，同时支持算数和逻辑指令。

在本实验中，ALU 包含三个输入引脚和五个输出引脚，具体引脚数据如表 2.1 所示。这些输入和输出引脚的设计旨在提供足够的灵活性，以满足不同指令和数据处理的需求。在 ALU 的设计中，除了基本的算术和逻辑运算外，还考虑了位移操作、条件分支判断等功能，以确保它在处理各种指令时的全面性。

此外，ALU 的功能也受到 ALU\_OP 信号的控制。ALU\_OP 决定了 ALU 要执行的具体操作，其功能码的定义见表 2.2。通过这种方式，ALU 在不同情况下可以执行不同的操作，使其更具灵活性和通用性。这种设计可以很好地满足不同指令对 ALU 功能的不同需求，同时简化了整个系统的控制逻辑。

表 2.1 算术逻辑运算单元引脚与功能描述

引脚	输入/输出	位宽	功能描述
X	输入	32	操作数 X
Y	输入	32	操作数 Y
ALU_OP	输入	4	运算器功能码，具体功能见下表
Result	输出	32	ALU 运算结果
Result2	输出	32	ALU 结果第二部分，用于乘法指令结果高位或除法指令的余数位，其他操作为零
OF	输出	1	有符号加减溢出标记，其他操作为零
UOF	输出	1	无符号加减溢出标记，其他操作为零
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效
<	输出	1	(x<y)?1:0, 对所有操作有效
≥	输出	1	(x≥y)?1:0, 对所有操作有效

表 2.2 运算器功能码及对应功能

# 华中科技大学课程设计报告

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

## 4. 寄存器堆 RF

寄存器堆（Register File，缩写为 RF）在计算机体系结构中扮演着关键的角色，用于存储和管理 CPU 中的寄存器。它采用上升沿触发，并与统一的同步时钟信号连接，以确保寄存器的同步操作。寄存器堆的主要功能包括支持指令执行、数据操作以及上下文切换。

在实验模组中，已经封装好了 RF 组件，可以直接使用。该 RF 组件具有以下输入和输出：

- 输入：R1#、R2#用于读取寄存器的编号，W#用于指定写寄存器的编号。WE 表示写使能，而 Din 表示写入的数据。输入 A 和 B 寄存器号码的数据位宽为 5 位，可以表示 0 号到 31 号寄存器。

- 输出：两个 32 位的数据输出。

通过这样的设计，寄存器堆提供了灵活的读写功能，同时具备了对 32 个寄存器进

# 华中科技大学课程设计报告

---

行存储和获取数据的能力。这样的寄存器堆设计有助于满足指令执行时对数据的快速访问需求，同时也为上下文切换提供了必要的支持。值得注意的是，寄存器堆的实现采用了同步时钟信号，以确保在时钟上升沿时进行有效的寄存器操作。

## 2.1.2 数据通路的设计

对于不同的指令有不同的数据通路，具体设计如下：

1. 取指令数据通路：在跳转方式下，我们通过多路选择器进行不同方式的选择。多路选择器的控制信号是根据具体情况确定的。在顺序执行方式下，我们对 PC 进行 +4 操作；而在跳转方式下，我们将跳转地址传给 PC，并通过多路选择器选择不同的方式。

2. R 型指令数据通路：对于 R 型指令，指令字通常包含源寄存器号 rs1 和 rs2，以及目的寄存器号 rd。我们进行的操作通常是  $R[rd] = R[rs1] \text{ op } R[rs2]$ 。这涉及到寄存器堆读寄存器、运算器运算和写回寄存器堆的过程，其中可能还有其他的控制信号，比如 ALU\_OP 的值，以确保正确的操作被选择。

3. I 型指令数据通路：I 型指令进行立即数和寄存器的联合运算，指令字包含 12 位立即数 I[11:0]、源寄存器号 rs1，以及目的寄存器号 rd。通常的操作是  $R[rd] = R[rs1] \text{ op } I$ 。特别地，对于 lw 指令，操作为  $R[rd] = M[R[rs1] + I]$ 。数据通路与 R 型指令类似，不同之处在于运算器的两个输入端分别为寄存器数据和扩展立即数。同时，lw 指令下的数据通路还包含读取数据存储器的通路。

4. S 型指令数据通路：对于 S 型指令，我们只考虑 sw 指令。指令字包含两个寄存器号 rs1 和 rs2，以及立即数 I。操作为  $M[R[rs1] + I] = R[rs2]$ 。数据通路依次经过取指、取寄存器数据、运算器相加运算和存储数据的过程，无需写回。

5. 分支指令数据通路：分支指令可分为条件分支指令和无条件分支指令。条件分支指令需要通过 ALU 获取状态信息，即 Beq 需要获取 equal 信号才能跳转。而无条件分支指令无需状态判断，直接根据指令字中的立即数扩展进行跳转。相应的，数据通路根据不同指令，在 ALU 的控制信号上有所区别。

6. U 型指令数据通路：U 型指令中含有 Auipc 指令。该指令将 PC 与指令内立即数相加，结果存入目的寄存器号 rd 中。只需要进行取指、计算、写回的过程即可。在计算时，可以直接使用加法器。

# 华中科技大学课程设计报告

7. Ecall 指令数据通路： Ecall 指令具有分支功能。当 a7 寄存器为 10 时，实现停机功能；否则，显示 a0 寄存器的数据。在 ecall 操作时，将 rs1 输入端设置为 a7 寄存器号 17，rs2 输入端设置为 a0 寄存器号 10，获取对应寄存器数据。然后进行相应的分支判断，执行 LED 显示或将各部件置为无效。

该板块只放出大致框架，具体部分请看下文。

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	

## 2.1.3 控制器的设计

控制器采用硬布线控制方式设计。输入指令字特征字段（OP，Funct）进行指令识别，输出该指令下所需的控制信号。对各个控制信号的取值情况以及功能说明如表 2.4。

表 2.4 主控制器控制信号的作用说明

控制信号	数据 位宽	取值	说明
R1Used	1	0	未使用寄存器堆 R1 口
		1	使用了寄存器堆 R1 口
R2Used	1	0	未使用寄存器堆 R2 口
		1	使用了寄存器堆 R2 口
MemToReg	1	0	寄存器写入数据不来自数据存储器
		1	寄存器写入数据来自数据存储器
MemWrite	1	0	写数据存储器控制信号
		1	
AluOP	4	0-12	运算器操作控制符
AluSrcB	1	0	寄存器堆 R2 读取到的数据
		1	I 型和 S 型立即数扩展后的数据

# 华中科技大学课程设计报告

控制信号	数据 位宽	取值	说明
RegWrite	1	0	寄存器堆禁止写入
		1	寄存器堆写使能
Bne	1	0/1	Bne 指令译码信号
Beq	1	0/1	Beq 指令译码信号
Jal	1	0/1	Jal 指令译码信号
Jalr	1	0/1	Jalr 指令译码信号
LHU	1	0/1	LHU 指令译码信号
BLTU	1	0/1	BLTU 指令译码信号
LUI	1	0/1	LUI 指令译码信号

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。具体细节见后文。

表 2.5 主控制器控制信号框架

指令	R	RW	WE	X	EXT	Y	ALUOp	MemWrite	MemRead	Din	Branch	SYSCALL

## 2.2 中断机制设计

### 2.2.1 总体设计

中断机制的响应方式通常包括以下步骤：

1. 中断源触发：中断可以由多种原因触发，例如外部设备的信号、CPU 执行特定指令、程序异常等。当中断事件发生时，硬件会向 CPU 发送一个中断信号。

2. 保存当前状态：在响应中断之前，CPU 通常会保存当前的执行状态。这包括保存程序计数器（PC）、寄存器内容和标志位等，以确保在中断处理完毕后能够恢复到正常的执行状态。

3. 中断向量定位：每个中断事件都有一个唯一的标识号，被称为中断向量或中断号。这个标识号用于确定要执行哪个中断处理程序。通常，中断向量会映射到中断向

量表，该表存储了中断号与对应中断处理程序的关联关系。

4. 中断处理程序执行：一旦中断向量确定，CPU 会跳转到对应中断处理程序的入口点，开始执行与该中断事件相关的特定操作。中断处理程序通常包括保存寄存器状态、执行中断服务例程、处理中断事件、执行必要的操作，然后最终返回到原来的程序。

5. 中断处理程序完成：中断处理程序执行完毕后，CPU 会从保存的状态中恢复，继续执行原来的程序。这包括还原保存的程序计数器、寄存器内容和标志位等。

6. 服务多个中断：中断系统通常允许多个中断事件同时存在，可能需要有优先级机制来确定哪个中断应该首先处理。这确保了在有多多个中断同时发生时，系统能够合理地确定哪一个中断具有更高的优先级，需要首先得到处理。

这些步骤构成了中断机制的一般执行流程，允许系统在出现中断事件时有序地进行处理，并最终返回到正常的程序执行状态。

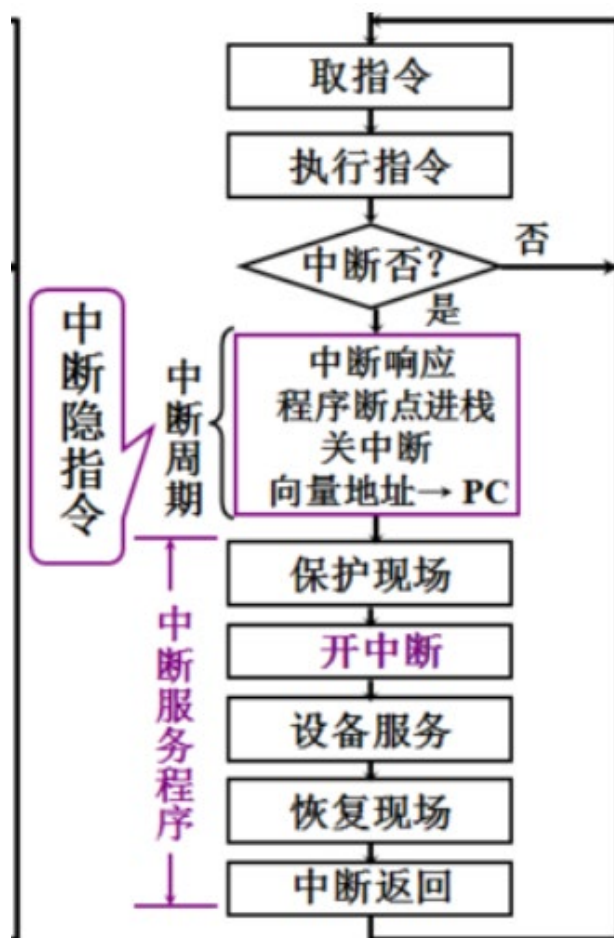


图 2.2 中断流程图

## 2.2.2 硬件设计

1. 中断识别: 为了确定哪个中断请求应该被优先响应, 使用优先编码器对中断请求信号进行编码是一个明智的选择。这确保了系统可以正确地辨别和响应具有高优先级的中断。

2. 中断使能寄存器: 中断使能寄存器可以通过 D 触发器实现。当输出为 1 时, 表示允许中断; 当输出为 0 时, 表示屏蔽所有中断。这个寄存器的状态对中断的响应起到了关键作用, 确保只有在允许中断的情况下, 中断请求才能被有效处理。

3. EPC 寄存器: EPC 寄存器可采用 32 位寄存器实现, 上升沿触发。在有中断请求时, 通过将使能端置 1, 可以将当前 PC+4 的值写入 EPC 寄存器, 作为返回地址。在中断执行过程中, 保持使能端为 0, 防止写入新的值。

4. 控制器修改: 为了支持开中断、关中断以及访问 mEPC 寄存器, 需要增加 csrrsi、csrrci、csrrw 指令数据通路。这三条指令主要用于访问 CSR 寄存器, 具体实现可参考指令手册。

5. 开中断寄存器: 在中断请求发出的一个周期内, 中断信号为 1, 系统响应中断。然而, 在一个周期之后, 应立即关中断, 防止后续中断继续响应。这一功能可以由硬件实现, 将中断信号反馈到 D 触发器的清零端, 确保在一个周期后中断被关闭, 以避免后续中断的连续响应。

这些设计选择和修改确保了中断机制的顺利运行, 包括中断的识别、使能、EPC 寄存器的处理、对控制器的修改以及中断的开关操作。这样的设计在系统中引入了更灵活和可控制的中断管理方式。

## 2.2.3 软件设计

在中断处理的软件部分设计中, 需要实现现场保护、现场恢复、CSRRSI 开中断指令和 CSRRCI 关中断指令。

### 1. 保护现场:

- 在进入中断服务程序时, 首先需要对现场进行保护。这包括将原程序使用的寄存器数值进行压栈操作, 以保存它们的当前状态。

- 可以使用栈指针 (例如, 栈寄存器) 来实现对寄存器的保存, 将寄存器的值按照规定的顺序依次压入栈中。

## 2. 恢复现场:

- 在结束中断服务后, 需要将栈中保存的数据弹出并返回给对应的寄存器, 以还原原程序的执行状态。
- 根据压栈的顺序, 可以使用栈指针逐个弹出数据, 并将其写回到相应的寄存器中。

## 3. CSRRSI 开中断指令和 CSRRCI 关中断指令:

- 在中断服务程序中, 使用 CSRRSI 和 CSRRCI 指令来开关中断。CSRRSI 用于将 CSR 寄存器中的某一位设置为 1, 即开中断; CSRRCI 用于将 CSR 寄存器中的某一位清零, 即关中断。
- 具体的 CSR 寄存器和位需要根据系统的中断控制寄存器的定义来确定。

## 4. 多级中断处理:

- 在多级中断中, 需要在保护现场后开中断, 允许在 IE (中断使能) 为开中断状态下进行中断服务。这允许在当前中断服务过程中响应其他高优先级中断。
- 在恢复现场之前需要进行关中断操作, 因为在现场保护和恢复现场时不允许被中断打断。这可以通过使用 CSRRCI 指令来实现。

## 5. 返回中断 (uret) 指令:

- 在中断服务结束时, 使用 uret 指令来返回。uret 指令会从 EPC (中断入口地址) 中获取返回地址, 同时执行 EPC 到 PC 的赋值, 实现跳转回原程序。
- 在返回时, 需要同步进行开中断操作, 以确保在中断服务结束后系统能够响应其他中断。

以上设计保证了在中断处理软件部分对现场的正确保护和恢复, 同时通过 CSR 寄存器的操作实现了中断的开关。这样的设计使得中断服务能够在多级中断的环境中正确、高效地执行。

## 2.3 流水 CPU 设计

### 2.3.1 总体设计

采用 5 段流水 CPU 作为流水 CPU 的设计架构。5 段流水 CPU 框架图如图 2.3 所示。五个功能段依次为取指 (IF) 段、译码 (ID) 段、执行 (EX) 段、访存 (MEM) 段以及写回 (WB) 段, 一条指令经过上述 5 段后完成指令处理。各段的



# 华中科技大学课程设计报告

---

功能与单周期 CPU 设计思路相同，各部件无较大改动。各段间采用流水寄存器进行各段处理完成后数据和结果的锁存，且在下一时钟周期到来时，传给下一功能段使用。程序计数器 PC 和各个流水寄存器采用统一公共时钟进行同步。

采用 5 段流水 CPU 设计架构是一种有效提高指令执行吞吐量的方式。以下是对 5 段流水 CPU 框架的简要描述：

## 1. 取指（IF）段：

- 负责从指令存储器中取出当前指令。
- 程序计数器（PC）用于存储当前指令的地址，每个时钟周期都会自增。
- 取出的指令被传递到流水寄存器，准备传送到下一个阶段。

## 2. 译码（ID）段：

- 对取出的指令进行解码，识别操作码和操作数。
- 读取寄存器文件，获取源操作数。
- 生成控制信号，用于控制后续阶段的操作。
- 将解码后的信息传递给下一个阶段。

## 3. 执行（EX）段：

- 执行算术运算、逻辑运算或其他操作，具体取决于指令类型。
- 对于分支指令，计算分支目标地址。
- 生成结果，并将其传递到下一个阶段。

## 4. 访存（MEM）段：

- 对于访存指令，访问数据存储器（内存）。
- 对于加载（load）指令，将从内存中读取的数据传递到下一个阶段。
- 对于存储（store）指令，将要存储的数据传递给下一个阶段。

## 5. 写回（WB）段：

- 将执行阶段生成的结果写回寄存器文件。
- 对于访存阶段的加载指令，将从内存中读取的数据写回寄存器文件。
- 完成一条指令的执行，并准备接受下一条指令。

这五个功能段之间通过流水寄存器进行数据传递，确保每个阶段的输出在下一个时钟周期开始时传递给下一个阶段。整个流水线的节奏由公共时钟同步，保证各个阶段的协调和正确执行。这种设计框架有效地将指令的执行划分为多个阶段，提高了指令执行的并行度，从而提高了整个 CPU 的性能。



图 2.3 指令流水线示意图

## 2.3.2 流水接口部件设计

流水接口部件在整个设计中具有关键作用，其任务是在时钟上升沿触发的情况下，将上一段的数据和控制信号锁存一个周期，以便供下一段使用。该部件的实现采用寄存器，使得信号得以稳定地传递。触发时通过使能端连接停机信号，以确保在需要停机的情况下流水接口停止工作。

值得注意的是，清零信号在这个流水接口部件中充当多路选择器的选择信号。在正常情况下，选择输入信号传递数据，而在清零时，则选择常数 0。这种设计灵活性高，使得在需要清零的情况下，可以有效地将数据置零，确保系统状态的准确性。

需要强调的是，各段的流水接口结构在整体上是相似的，但传输的数据可能存在差异。这种一致的结构设计有助于简化整体架构，提高系统的可维护性和可扩展性。

## 2.3.3 理想流水线设计

在理想流水线中，不存在分支指令和数据冲突的情况。因此，我们可以采用哈佛结构来处理结构冲突。具体实现上，将指令存储器放置在 IF（取指）功能段，而数据存储器则安置于 MEM（访存）功能段。这样的结构有助于简化流水线的设计，并提高指令和数据访问的效率。

控制信号在这个理想流水线中通过流水寄存器进行段间传输。值得强调的是，每个段所需的控制信号只能传递到该段，不可跨越段传输和使用。这样的限制确保了控制信号的精准传递，防止不同段之间的干扰，从而提高系统的稳定性和可靠性。

这一设计策略旨在充分利用哈佛结构的优势，最大化提高流水线的整体性能。通过合理的控制信号传递和布局，我们可以更好地实现每个功能段的独立工作，确保流水线的平稳运行。

## 2.4 气泡式流水线设计

### 2.4.1 总体设计

在理想流水线的架构基础上，气泡流水线引入了额外的 EX 段分支处理以及数据冲突检测与处理机制。对于分支冲突，在 EX 段检测到需要跳转时，将得到的跳转地址传递到 IF 段的 PC 中。此时，IF/ID 和 ID/EX 寄存器内的数据被标记为无效，触发 Flush 操作。

对于数据冲突，仅在 ID 段从寄存器堆中取数时可能发生。因此，在 ID 段检测与 EX、MEM、WB 段的数据相关性。由于 ID 与 WB 的数据冲突已经在前文考虑，只需检测与 EX、MEM 的数据相关性。当发生冲突时，置 IF/ID 的使能端为 0，执行 Stall 操作，并清空 ID/EX 寄存器数据，插入气泡。

### 2.4.2 EX 段分支设计

控制信号与数据通过流水寄存器传递到 EX 段。对于条件分支指令，通过 ALU 获取所需的状态信息，若满足跳转条件，则产生 BranchTaken 信号，并将跳转地址传递到 IF 段的 PC 中。对于无条件分支指令，如 Jal 指令，则直接在 EX 段进行分支，跳转地址传递到 IF 的 PC 中。对于 Jalr 指令，需要使用 ALU 计算的结果作为跳转地址，同样传递到 IF 的 PC 中。由于 IF 和 ID 段内的指令为误取指令，在分支跳转时需要清空 IF/ID、ID/EX 流水寄存器。

### 2.4.3 数据冲突检测与处理逻辑设计

在处理完 ID 段与 WB 段的数据冲突基础上，需要检测 ID 段与 EX 段、MEM 段的数据冲突。检测机制比较 ID 段的源寄存器号与 EX 段、MEM 段的的目的寄存器号。由于源寄存器号包含 R1# 与 R2#，并且对于不同的指令，R1# 或 R2# 会出现无效的情况，即不需要使用源寄存器。因此，需要对 ID 段的指令字段 (OP, Funct) 进行逻辑判断，查看 R1# 与 R2# 在该指令下的有效性。有效的源寄存器号与 EX 段的的目的寄存器号 W# 和 MEM 段的的目的寄存器号 W# 进行比较，如果 W# 不为 0 号寄存器且存在源寄存器号与目的寄存器号相等的情况，同时冲突段存在写使能信号 RegWrite，则表明出现数据冲突。

为消除检测出的冲突，采用插入气泡的方式解决。暂停 IF/ID 流水寄存器，清空 ID/EX 流水寄存器，在 ID 段后增加一个空指令气泡。一般情况下，当在 EX 段出现第一次数据冲突时，会在后续产生 2 个气泡；在 MEM 段出现第一次数据冲突时，只需产生 1 个气泡。这种策略有助于维护流水线的稳定运行。

## 2.5 重定向流水线设计

在理想流水线的基础上，为了处理数据冲突，引入了 EX 段分支处理分支冲突，同时采用重定向方法来解决数据相关性问题。重定向机制首先在 ID 段进行数据相关性的检测，类似于气泡流水线的处理方式。不同之处在于，需要输出更详细的信息，而非仅有数据相关信号。具体来说，需要输出 R1# 是否与 EX.W# 或 MEM.W# 冲突，以及 R2# 是否与 EX.W# 或 MEM.W# 冲突的信息。由于 R1# 或 R2# 的相关性检测存在三种状态，因此将相关性信息的输出端设置为 2 位。当输出为 0 时，表示该寄存器无冲突产生；输出为 1 时，表示该寄存器与 MEM 段的目的寄存器冲突；输出为 2 时，表示该寄存器与 EX 段的目的寄存器冲突。采用优先编码器，首先处理 EX 段的冲突，因为 EX 段距离 ID 段最近，是最新的待写入数据。

将 ID 段获取的相关性状态通过 ID/EX 流水寄存器传递到 EX 段，在 EX 段进行重定向设置。当状态号为 1 时，向对应的 ALU 输入端送入 WB 端的写回数据 WriteBackData；当状态为 2 时，向对应的 ALU 输入端送入 MEM 端的计算结果 AluResult。通过这种方式，提前获取写入寄存器堆的数据，避免了由于插入气泡导致的性能损耗。

# 华中科技大学课程设计报告

特别地，对于 Load-Use 相关的数据冲突，由于产生冲突的是访存指令，需要等待数据存储器的读取数据，这会增加 EX 段的关键路径延迟，从而导致流水线的频率大幅降低。为此，对 Load-Use 冲突进行了特殊处理。

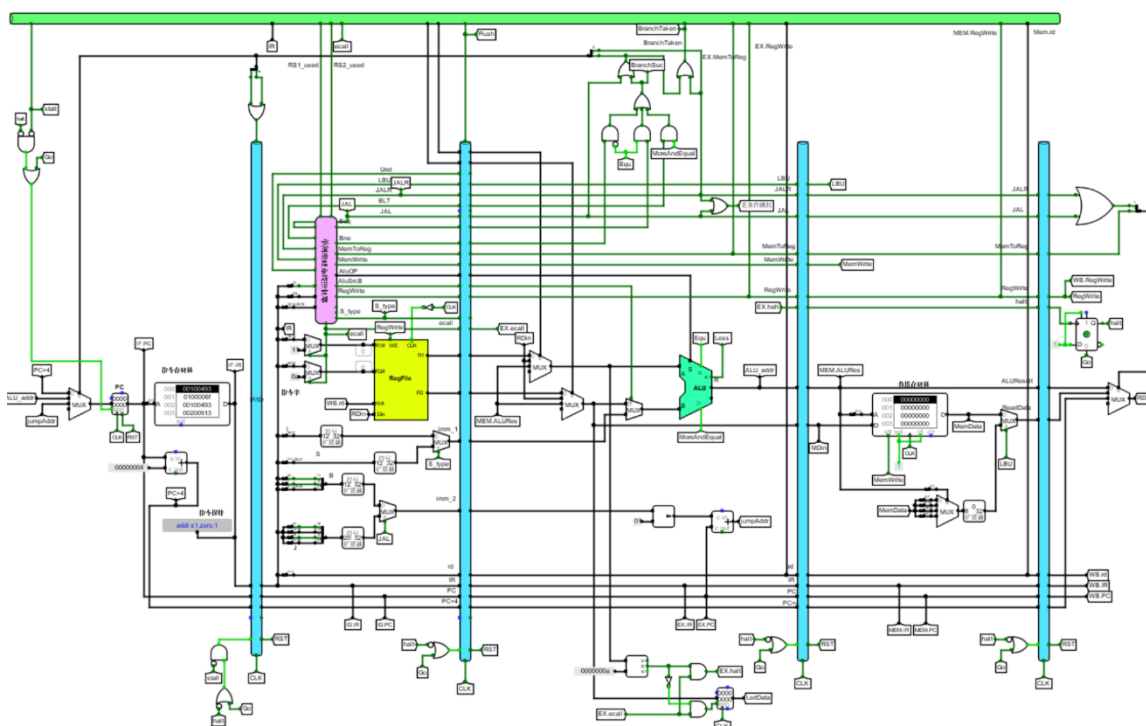


图 2.4 重定向流水线框架图

## 2.6 动态分支预测机制设计

重定向处理数据相关性后，流水线性能显著提升。然而，分支跳转成功时插入的气泡仍然影响性能，因此引入了动态分支预测机制，通过动态预测分支，减少插入气泡的频率。

动态分支预测机制设计：

1. 针对条件分支，采用预测 B 型指令；对于无条件分支，预测 Jal 指令。
2. 为避免启动问题，采用分支超前获取跳转地址的机制，在 IF 段计算跳转地址。若在 IF 段未命中，将地址与状态预先填入 BTB 表。与在 EX 段修改 BTB 表不同，这种机制改善了相隔较近的跳转指令的预测。

# 华中科技大学课程设计报告

3. 当指令来到 EX 段时，进行预测成功与否判断。将跳转信号与预测跳转信号进行比较。若同时预测不跳转或同时预测跳转表示预测成功，无需插入气泡。反之，则预测失败，需要清空 EX 段前取得的指令。

4. 在 EX 段 BTB 命中的前提下，根据双位预测状态图，对 BTB 中的预测状态位进行更新。

BTB 表的设计:

1. BTB 表采用全相联 cache 设计思路，实质上是一个全相联 cache。

2. 对于写使能的设置与单纯的全相联 cache 有些许不同。由于采用了 IF 段超前预测机制，cache 中的 valid、分支指令地址以及分支目标地址可以在 IF 段提前写入。当 IF 段 Miss 时，对上述各数据进行写操作。

3. 对于初始分支预测位的写入，考虑跳转大多存在于循环中，将其置为 11，预测跳转不失为大概率事件。

4. 在 EX 段，仅需在 EX 段 Hit 时，进行分支预测位的更新。淘汰算法采用 LRU 最近最久未使用算法进行 cache 行的淘汰。每个 cache 行设置一个随访问次数增加的计数器，当访问到某个 cache 行时，对该行计数器进行清零；对于没有访问到的 cache 行，计数器加 1。当 cache 满且未命中时，淘汰计数器数值最大的 cache 行。

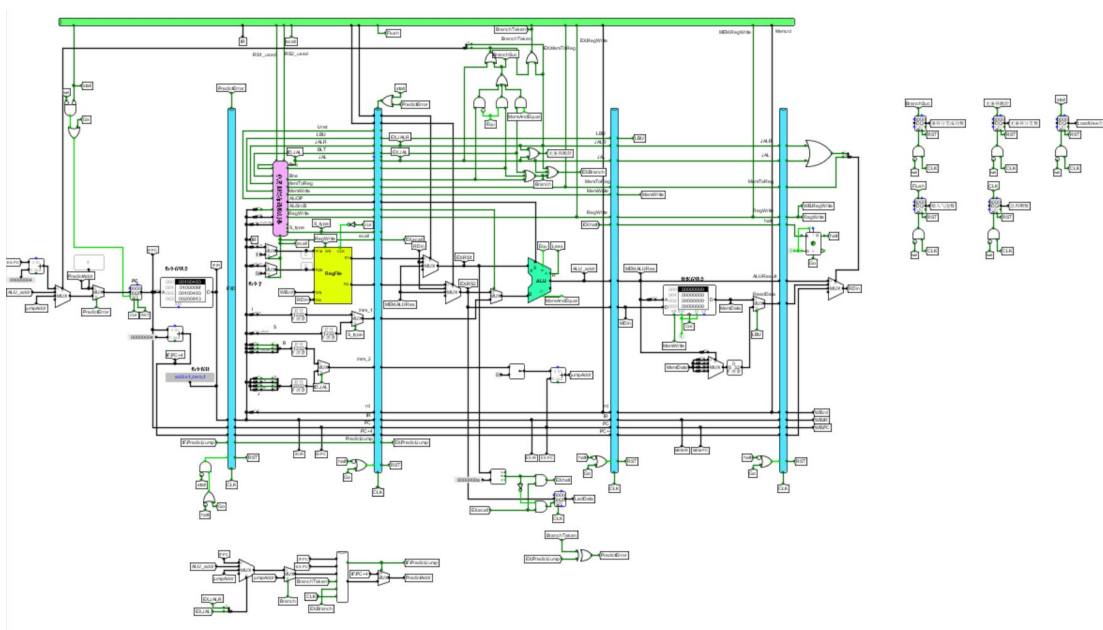


图 2.5 动态预测结构框架图

## 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)

使用一个 32 位寄存器作为程序计数器 PC，触发方式为下降沿触发。具体来说，下一个将要执行的指令的地址被输入到这个寄存器，而输出则是当前执行指令的地址。引入停机信号 (Halt)，通过非门取反并与时钟信号相与，实现停机控制。当需要停机时，Halt 信号为 1，通过非门变为 0，与时钟信号相与后，屏蔽时钟信号，使整个电路停机。这个设计确保了对程序计数器的有效控制，同时支持停机操作。如图 3.1 所示。

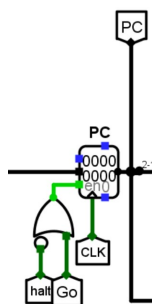


图 3.1 程序计数器 (PC)

##### 2) 指令存储器 (IM)

使用一个只读存储器 (ROM) 来实现指令存储器，其中地址位宽为 10 位，数据位宽为 32 位。由于程序计数器 (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，因此我们需要通过分线器来选择指令存储器的输入地址。如图 3.2 所示。

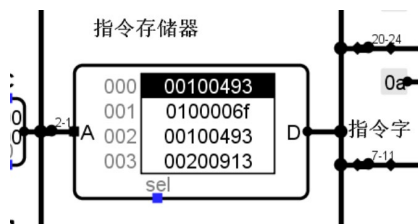


图 3.2 指令存储器 (IM)

## 3) 数据寄存器 (DM)

数据寄存器也是用 RAM 来实现, 由于和指令寄存器类似的原因, 地址线只用 2-11 位作为输入, 需要存入的数据通过接口 D 输入, 使能信号由 MemWrite 提供, 时钟上跳沿更新寄存器信息, 输出数据从 Dout 输出。如图 3.3 所示。

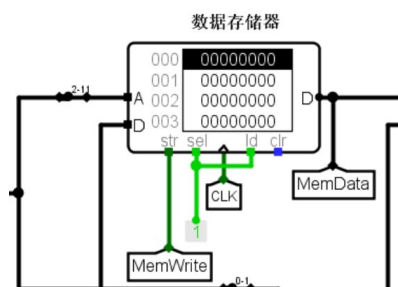


图 3.3 数据寄存器 (DM)

## 4) 寄存器堆 (RegFile)

寄存器堆在 logisim 的 CS3410Components.jar 中已经实现, R1#和 R2#是要读取的寄存器 5 位编号, W#是要写入的寄存器 5 位编号, Din 是写入寄存器的 32 位数据值, WE 是写使能信号, CLK 是时钟信号。R1 和 R2 是读取的 32 位的寄存器数据。具体的实现如图 3.4 所示。

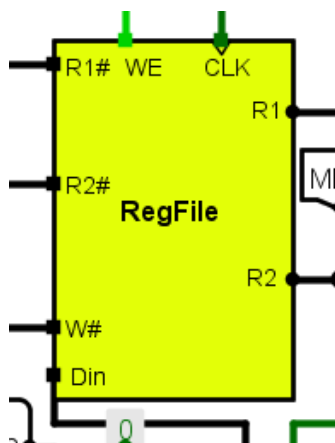


图 3.4 寄存器堆 (RegFile)

## 5) 运算器 (Alu)

在这个运算器设计中, 它实际上就是一个集成了多种计算功能的工具。通过输入 32 位的运算数据 X 和 Y, 以及 4 位的功能号 ALUOP, 我们能够得到一个 32 位的



# 华中科技大学课程设计报告

运算结果 Result。此外，它还输出了一些状态信号，比如 Zero 用于标识是否为零，Negative 用于标识是否为负数。

这个运算器采用了并行运算的方式，同时对输入的 X 和 Y 进行计算。通过一个多路选择器，根据 ALUOP 的指示，选择合适的输入端口作为最终的运算结果。需要特别留意的是，如果有立即数参与运算，那么在输入到 ALU 中之前，会将其扩展为 32 位的值，以确保运算的正确性。具体实现如图 3.5、图 3.6 所示。Alu 具体的操作功能码请见上文。

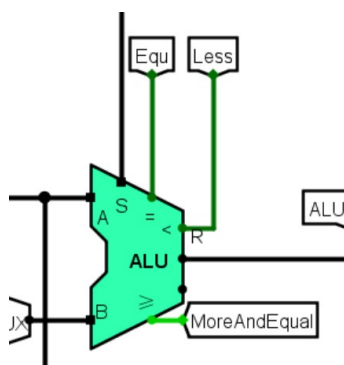


图 3.5 运算器 (ALU)

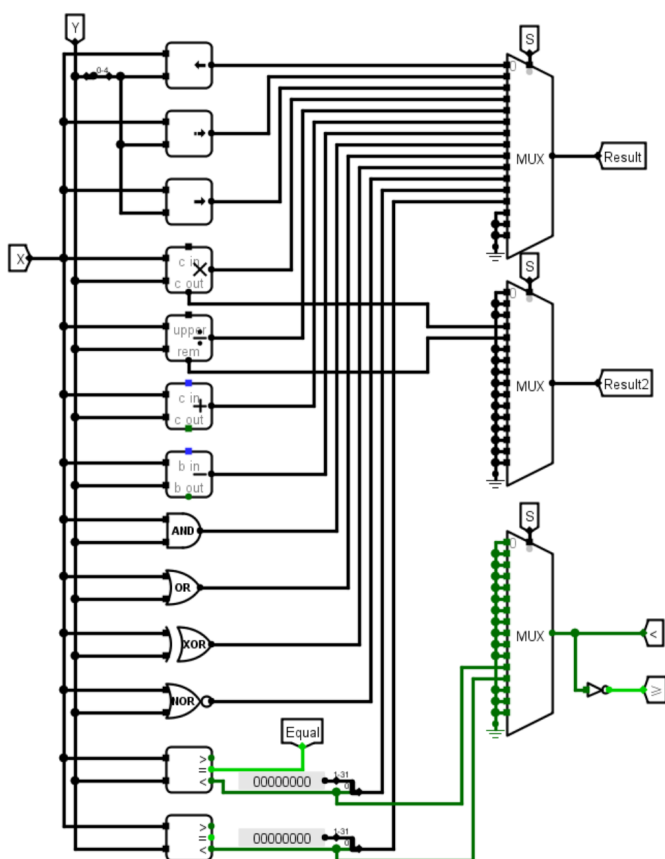


图 3.6 运算器 (ALU)

# 华中科技大学课程设计报告

## 3.1.2 数据通路的实现

按照单周期 CPU 的设计思路，首先构建各主要功能部件，然后通过五个主要功能阶段（取指、译码、执行、访存、写回）依次连接这些部件，从而建立基础的数据通路。最后，根据各类指令在不同功能上的区别，对数据通路进行细节调整。具体数据通路实现如图 3.7 所示。

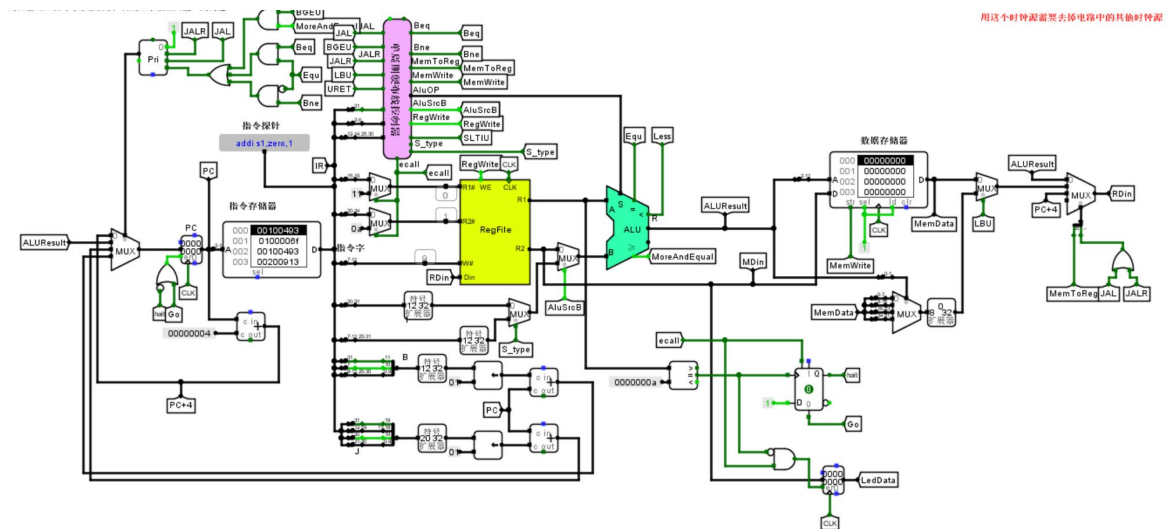


图 3.7 单周期 CPU 数据通路

## 3.1.3 控制器的实现

根据控制器的设计思路，对硬布线控制信号框架图进行填写，硬布线控制器框架图如图 3.8 所示。图中左侧为对应指令的（OP, Funct）译码的指令字段，右侧为对应指令输出的控制信号，需要输入的控制信号置 1，不相关信号不填写。对于每个控制信号，都能生成一个逻辑表达式，以此生成逻辑电路。

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI
* 指令	OpCode (6-10位)	OpCode (11-15位)	OpCode (16-20位)	OpCode (21-25位)	OpCode (26-30位)	OpCode (31-35位)	OpCode (36-40位)	OpCode (41-45位)	OpCode (46-50位)	OpCode (51-55位)	OpCode (56-60位)	OpCode (61-65位)	OpCode (66-70位)	OpCode (71-75位)	OpCode (76-80位)	OpCode (81-85位)	OpCode (86-90位)	OpCode (91-95位)	OpCode (96-100位)	OpCode (101-105位)	OpCode (106-110位)	OpCode (111-115位)	OpCode (116-120位)	OpCode (121-125位)	OpCode (126-130位)	OpCode (131-135位)	OpCode (136-140位)	OpCode (141-145位)	OpCode (146-150位)	OpCode (151-155位)	OpCode (156-160位)	OpCode (161-165位)	OpCode (166-170位)	OpCode (171-175位)
1	add	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	sub	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	and	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	ori	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	sll	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	srl	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	sllw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	addi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	andi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	ori	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	xori	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	sll	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	srl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	sllw	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	sw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 3.8 硬布线控制器控制信号框架

# 华中科技大学课程设计报告

硬布线控制器具体实现如图 3.9、图 3.10

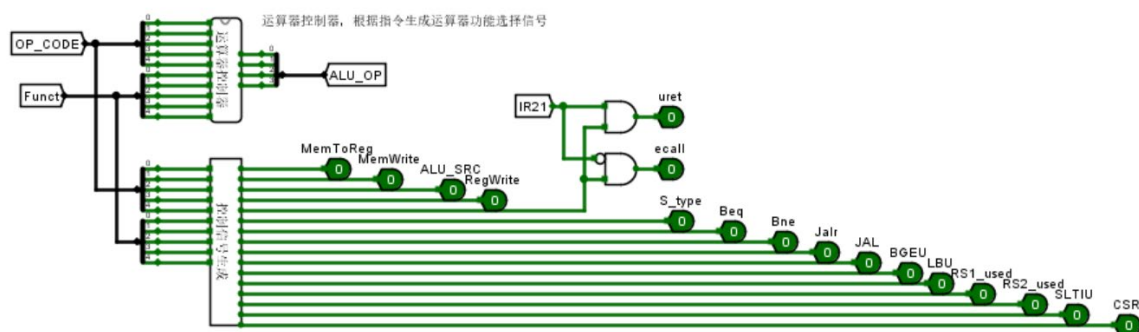


图 3.9 硬布线控制器实现电路

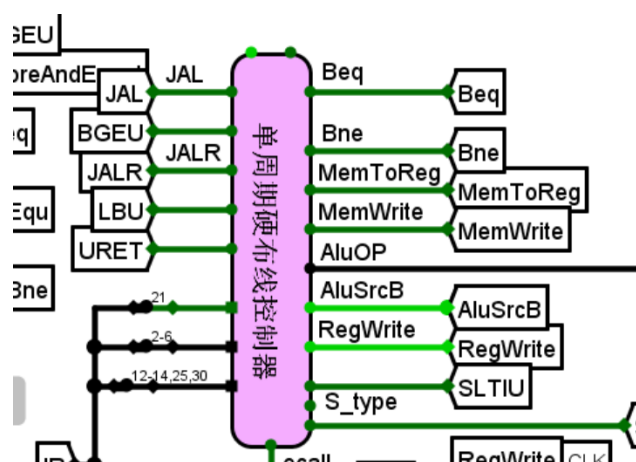


图 3.10 硬布线控制器封装

## 3.2 中断机制实现

### 3.2.1 单级中断实现

根据单级中断机制的设计要求，对中断信号采样电路、中断响应实现电路、中断使能状态电路、EPC 断点保存电路以及中断返回实现电路进行详细实现介绍。

#### 1. 中断信号采样实现：

- 设计一个中断信号采样电路，用于持续检测中断请求信号的状态。
- 在检测到中断请求信号（如`inter1`、`inter2`、`inter3`）时，产生持续中断信号，以使电路察觉到中断请求。

#### 2. 中断响应实现：

- 使用优先编码器对中断信号进行筛选，选出优先级更高的中断信号。
- 将选出的中断信号对应的跳转地址送入程序计数器（PC）。

# 华中科技大学课程设计报告

– 采用多路选择器，根据优先级选择相应的中断地址，将其送入 PC 以实现中断响应。

## 3. 中断使能状态实现：

– 设置一个 1 位寄存器作为 IE 寄存器，用于表示中断使能状态。0 表示开中断，1 表示关中断。

– 在中断响应时，需要改变 IE 状态为关中断（将 IE 寄存器置 1），以防止其他中断的干扰。

– 设定初始状态为开中断。

## 4. EPC 断点保存实现：

– 使用一个 32 位寄存器作为 EPC（异常程序计数器）断点保护寄存器。

– 在接收到中断信号后，将断点处的下一跳地址存入 EPC 寄存器中，以便后续中断返回时使用。

## 5. 中断返回实现：

– 将当前中断号存放在一个寄存器中，用于标识当前中断。

– 在中断处理完成后进行中断返回操作，将 IE 状态置为开中断，将 EPC 断点保存地址传入 PC 寄存器，并关闭对应的持续中断信号。

这样的设计满足了单级中断机制的要求，确保中断能够被及时响应，并且在处理完成后能够正确返回到中断前的程序。

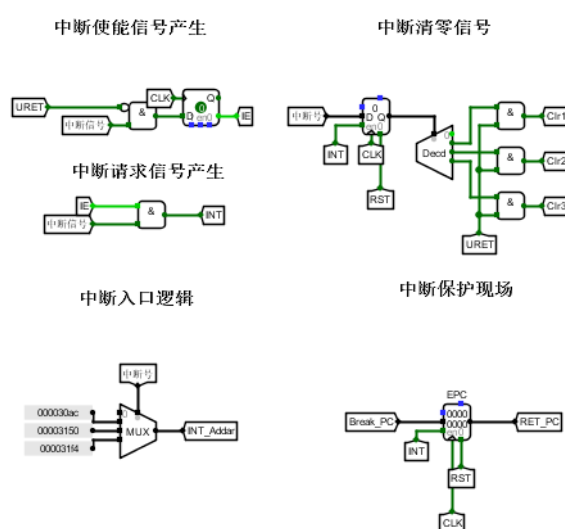


图 3.11 单级中断结构

## 3.2.2 多级中断实现

在多级中断系统中，相对于单级中断，需要额外考虑不同优先级中断相应顺序的逻辑处理问题。以下是对这些问题的详细实现介绍：

### 1. EPC:

为每一个中断引入对应的 EPC 寄存器，用于保存该中断的断点地址。这样，即使在一个中断处理的过程中发生了新的中断请求，可以通过不同的 EPC 寄存器保存各个中断的状态，确保中断处理的正确性。

### 2. 中断优先级比较:

- 在中断响应时，需要对新产生的中断与当前中断进行优先级比较。
- 如果新产生的中断优先级较低，则继续完成当前中断的响应。
- 如果新产生的中断优先级较高，则立即跳转到高优先级的中断上进行处理，确保高优先级中断能够及时响应。

### 3. 记录当前中断号和中断返回:

- 引入一个栈用于记录当前中断号和中断返回的信息，实现先存后取的策略。
- 当中断信号为 1 时，栈计数器加 1，并将当前中断号和返回中断号存入栈中。
- 当 ERET（中断返回）信号为 1 时，栈计数器减 1，并从栈中弹出相应的数据，以获取中断的信息。

这样的设计确保了在多级中断系统中能够正确管理各个中断的状态，实现了优先级比较和中断返回的逻辑处理。整个系统通过合理使用 EPC 寄存器和栈来保持多个中断的上下文，确保中断系统的可靠性。

## 3.2.3 流水单级中断实现

流水单级中断是在重定向流水线与单级中断的基础上组合而成，基本上是将二者整合在一起，在将二者整合的时候需要注意当产生中断响应信号时需要清除 IF/ID 以及 ID/EX 两个流水接口部件；当产生 URET 信号时需要清除 IF/ID 流水接口部件；EPC 接入的地址应该是 ID 段指令的地址。

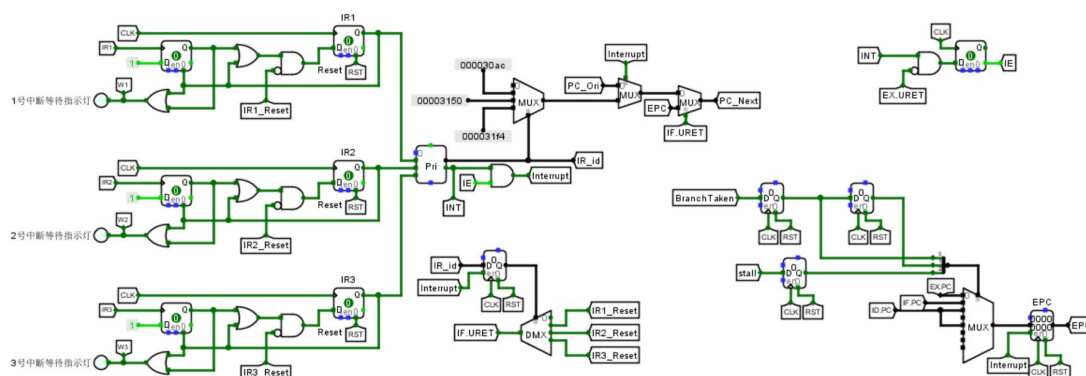


图 3.12 重定向流水线中断逻辑

## 3.3 流水 CPU 实现

### 3.3.1 流水接口部件实现

不同的指令执行阶段要通过流水接口部件来实现数据的锁存和同步清零的功能，也要接收时钟信号和使能信号的输入，主要由上升沿触发的寄存器和多路选择器组成。例如下图 3.13 展示的 MEM/WB 流水接口。

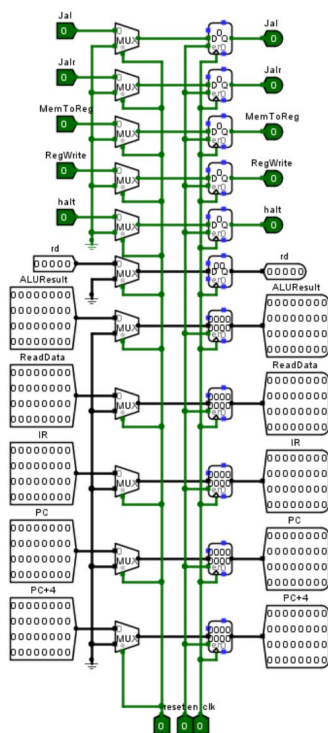


图 3.13 MEM/WB

## 3.3.2 理想流水线实现

取指段（IF）：在取指段，程序计数器（PC）的输出作为指令存储器（IM）的输入，将取出的指令通过 IF-ID 接口部件传送至译码段，同时 PC+4 的值也在该阶段计算并传递至下一段。

译码段（ID）：在译码段，指令解析器将指令字段解析，并将其送至操作控制器和立即数扩展器。操作控制器输出的控制信号中，部分信号在本段发挥作用，余下的信号通过集线器整合后传送至下一段。寄存器输出值和立即数扩展器的输出也在该段传递至后续阶段。

执行段（EX）：执行段使用分线器提取本段所需的控制信号，其余信号通过集线器打包后继续传递。ALU 的运算结果、寄存器输出以及 PC+4 的值向后传递。

访存段（MEM）：在访存段，寄存器的第二个输出作为数据存储器（DM）的输入，ALU 的运算结果作为访存地址。控制信号中，访存模式和写使能信号通过分线器提取，其余信号继续向后传递。访存输出也继续传递至后续阶段。

写回段（WB）：在写回段，ALU 输出、寄存器输出以及 PC+4 的值均传送回寄存器的输入端。选择信号从控制信号中提取。此外，停机信号在该段生成并生效，如果在其他段产生，系统会提前几个周期停机。

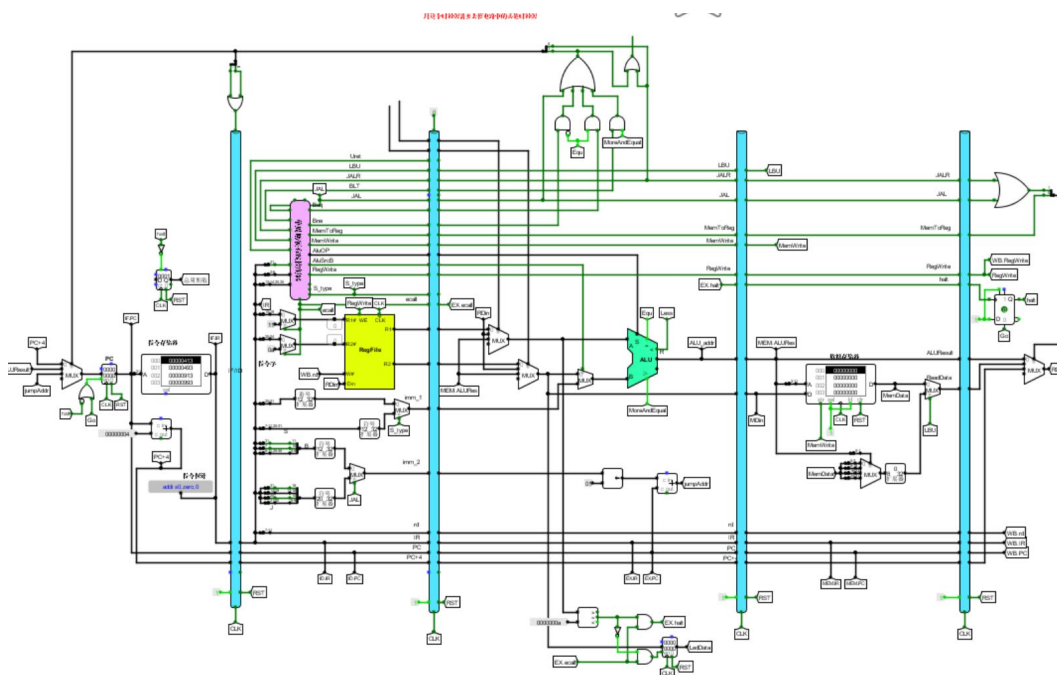


图 3.14 理想流水线

## 3.4 气泡式流水线实现

为解决气泡流水线中可能遇到的分支冲突和数据冲突问题，引入了控制器中的 rs1\_used 和 rs2\_used 位，用于判断在 ID 段是否需要使用 R1 和 R2 寄存器。具体实现过程包括在 Excel 表中增加对应的真值表，并在控制器中添加相应的控制信号。

rs1_used	rs2_used
1	1
1	1
1	1
1	1
1	1
1	1
1	
1	
1	
1	
1	
1	
1	
1	
1	1
1	1
1	1
1	
1	1
1	
1	
1	1

图 3.15 R1R2 真值表

## 3.5 重定向流水线实现

在解决数据相关问题上，引入了三种重定向情况，需要在 ALU 前加上四路选择器，这三路分别是：

1. 寄存器文件给出的数据。
2. WB 段的 aluresult，隧道名字为：RDin。
3. MEM 段的 aleresult，隧道名字为：MEM.aluResult。

为避免在 n-1 周期访存结果执行重定向，采用插入气泡的方式解决数据相关问题。为此，设计了一个 Load-Use 检测单元，以避免最长数据通路变为 EX+MEM，从而影响性能。



# 华中科技大学课程设计报告

Load-Use 检测单元的主要任务是在发现 Load 指令和后续的 Use 指令之间存在数据相关时，插入气泡以阻止数据的传递，从而避免数据相关问题。这种设计可以确保在 Load 指令的数据尚未准备好之前，不会执行后续的 Use 指令。

在 ALU 前加入四路选择器，具体选择哪一路数据取决于 Load-Use 检测单元的输出。这样可以灵活地根据数据相关情况选择合适的数据源，从而解决数据冲突问题。

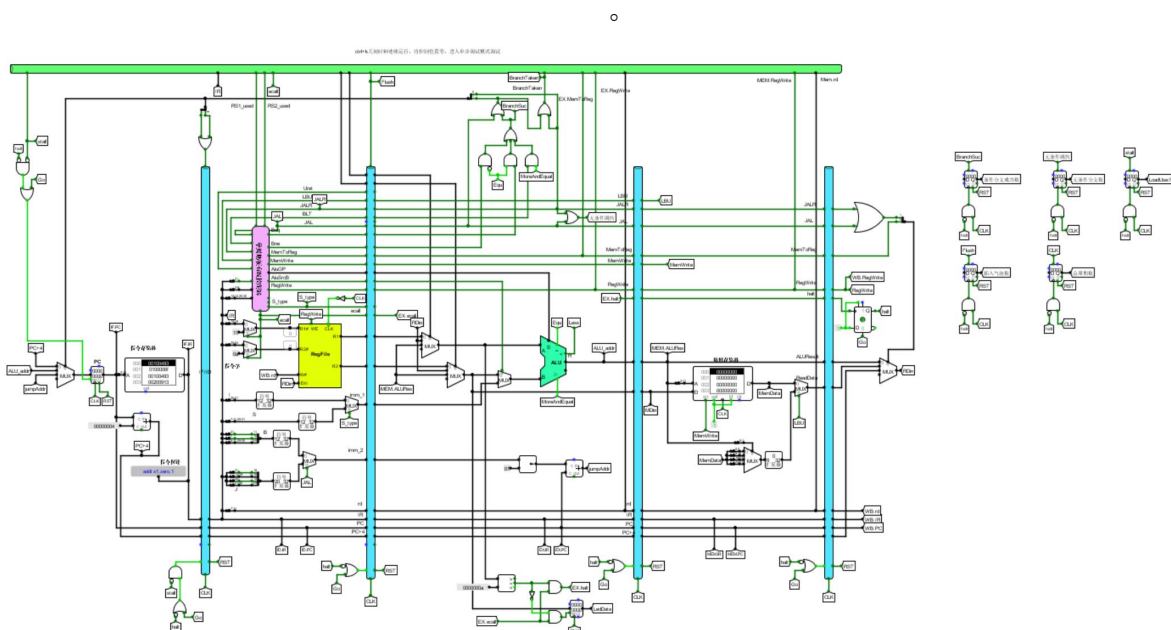


图 3.16 重定向流水线

## 3.6 动态分支预测机制实现

在基于重定向流水线的框架上，实现了动态分支预测，主要包括以下三个过程：

### 1. IF 段预测处理：

- 在 IF 段进行分支指令的预测处理，即判断是否命中 BTB 表（分支目标缓冲器）。采用超前预测机制，计算跳转地址，并将其写入 BTB 表中。

- 若命中 BTB 表，将状态置为“11”表示跳转状态。若不命中，则按照默认处理方式执行。

### 2. EX 段分支判断：

- 在 EX 段，进行 BTB 表的查找，获取分支预测历史位，并根据双位预测状态进行调整。匹配实际跳转与预测跳转的一致性，用于预测是否正确。

- 根据实际跳转和预测跳转的一致性进行预测的正确性判断。

### 3. 分支调整：

# 华中科技大学课程设计报告

- 根据 EX 段判断的预测正确性进行分支调整。如果预测正确，则不对分支进行调整，流水线继续执行。

- 如果预测失败，需要修改分支，将正确的调整地址送入 IF 段的 PC 寄存器，并清空 IF/ID、ID/EX 流水寄存器，以确保正确的指令流执行。

这样的动态分支预测机制通过在流水线的不同阶段进行处理，可以在一定程度上提高分支指令的执行效率，减少分支带来的流水线停顿。通过在流水线中加入分支预测的逻辑，可以更好地利用流水线并发执行的特性，提高整体性能。

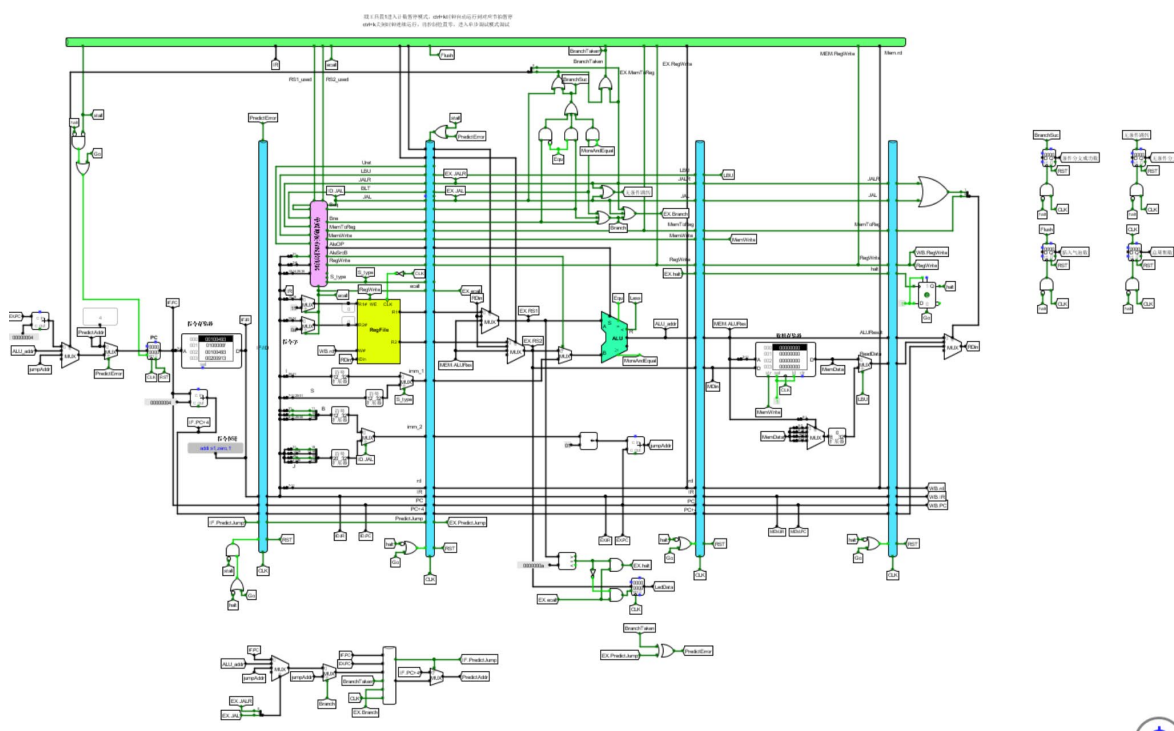


图 3.17 重定向流水线与动态分支预测

## 4 实验过程与调试

### 4.1 测试用例和功能测试

对实现的各项功能进行测试和记录，并利用附带的 hex 测试集对功能效率进行测试，具体的各项测试功能如下。

#### 4.1.1 单周期 CPU

##### 1) 基础 benchmark 测试

将基础 benchmark 程序转为 .hex 文件后，载入指令存储器中，测试运行。最终

周期数不记录最后一条停机 ecall 指令，为 1545，测试成功。

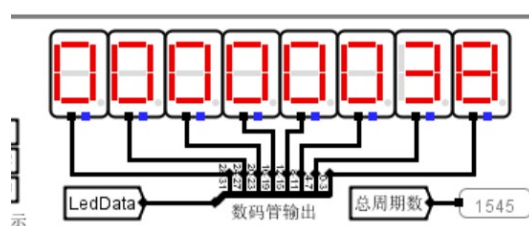


图 4.1 基础指令测试

##### 2) 扩展指令 CCAB 测试

扩展指令分别是 SRA、SLTIU、LBU、BGEU。其输出均符合预定输出，均已通过线下检查。

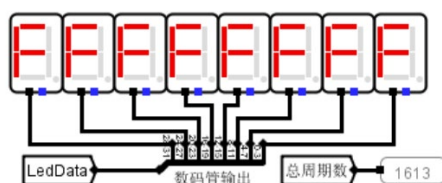


图 4.2 ccab 指令 SRA 测试

#### 4.1.2 中断测试

##### 1) 单级中断测试

# 华中科技大学课程设计报告

载入“单级中断.hex”文件，然后按照顺序分别按下 1 号、2 号、3 号中断按钮。处理 1 号中断时，没有被其他中断打断，而当 1 号中断结束时，优先执行 3 号中断，中断机制正确。

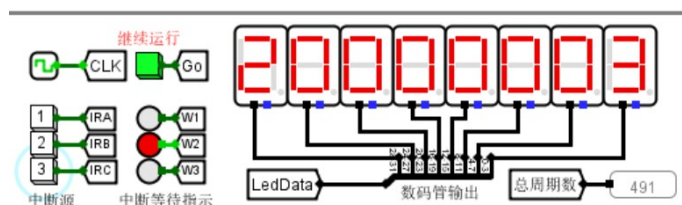


图 4.3 单级中断测试

## 2) 多级中断测试

将 EPC 硬件栈保护实现的多级中断.hex 文件载入后，按照顺序分别按下 1 号、2 号、3 号中断按钮。程序进入 3 号中断，3 号中断结束后进入 2 号中断。测试成功。

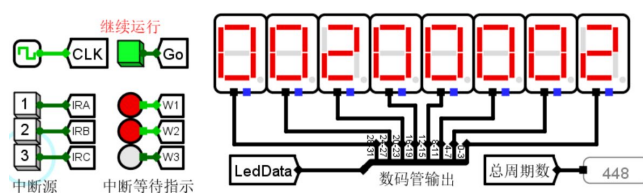


图 4.4 多级中断测试

## 3) 流水中断测试

载入“流水线中断.hex”文件，然后按照顺序分别按下 1 号、2 号、3 号中断按钮。处理 1 号中断时，没有被其他中断打断，而当 1 号中断结束时，优先执行 3 号中断，中断机制正确。

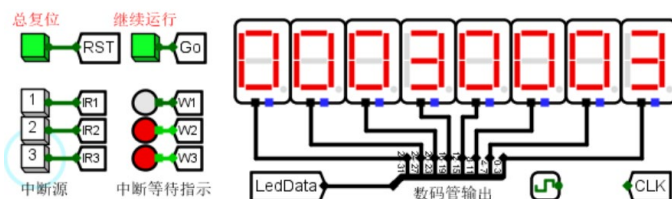


图 4.5 流水中断测试

## 4.1.3 流水 CPU 设计测试

### 1) 理想流水线测试

将“理想流水线.hex”载入指令存储器中。程序中止时观察输出。结果正确，

# 华中科技大学课程设计报告

故测试完毕。

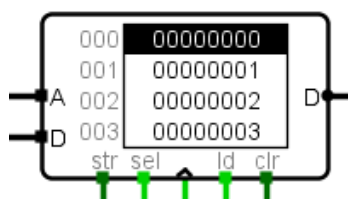


图 4.6 流水中断测试

## 2) 气泡流水线测试

将 risc-v-benchmark.hex 文件，载入指令存储器中，当运行到 ecall 指令时，即为基础的 24 条指令执行完毕，周期数为 3623。然后按下 Go 键继续运行，四条 ccab 指令也输出正确，均符合预期。测试成功。

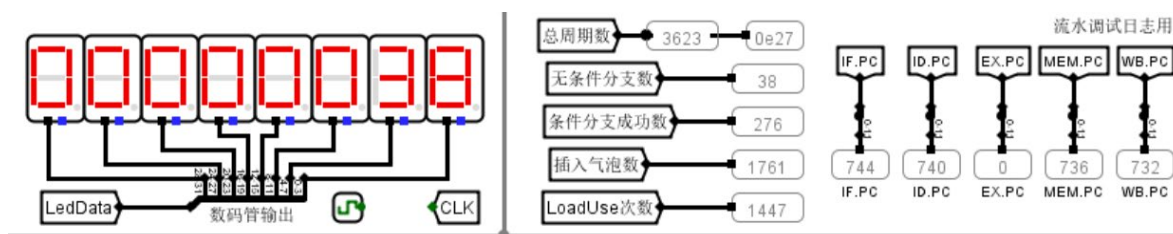


图 4.7 气泡流水线测试

## 3) 重定向流水线测试

将 risc-v-benchmark.hex 文件，载入指令存储器中，当运行到 ecall 指令时，即为基础的 24 条指令执行完毕，周期数为 2297。然后按下 Go 键继续运行，四条 ccab 指令也输出正确，均符合预期。测试成功。

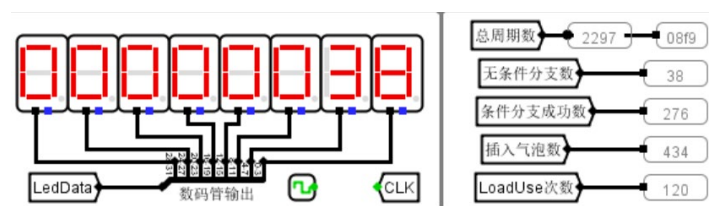


图 4.8 重定向流水线测试

### 4.1.4 动态分支预测测试

在重定向流水线的基础上加上动态分支预测机制，测试方法与气泡流水线还有重定向流水线相同。载入 benchmark.hex 文件，运行基础的 24 条指令后周期数为 1781。测试成功。

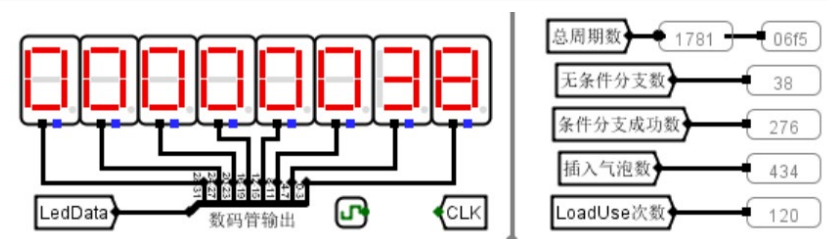


图 4.9 动态分支预测测试

## 4.2 性能分析

以基础 benchmark 程序运行总周期数为依据，分析不同方案的性能差异。各 CPU 方案运行总周期数如表 4.1 benchmark 运行周期数统计表所示。

表 4.1 benchmark 运行周期数统计表

CPU 方案	基础 24 条指令运行周期数
单周期 CPU	1545
气泡流水线	3623
重定向流水线	2297
重定向流水线+分支预测	1781

### 1) 单周期 CPU:

总周期数最少，但由于单周期 CPU 时钟周期约为五段流水 CPU 的五倍，实际性能最差。由于每个指令都需要花费较多的时钟周期，导致整体执行效率低下。

### 2) 气泡流水线:

性能次差，对数据冲突和分支冲突采用插入空指令的方式处理，导致浪费了较多性能。插入空指令的方式使得周期数过多，性能相对较低。

### 3) 重定向流水线:

性能较好，在数据冲突处理上进行了优化，周期数进一步减少。通过重定向机制有效地解决了数据冲突，减少了空指令的插入，提高了 CPU 性能。

### 4) 重定向流水线 + 分支预测:

性能最佳，进一步降低了分支冲突处理耗费的时延。在大概率预测成功的情况下，减少了空指令的插入，提高了 CPU 性能。动态分支预测机制有效地减少了分支带来的流水线停顿，提高了整体执行效率。

综合来看，重定向流水线 + 分支预测的方案在性能上表现最佳，有效地优化了数据冲突和分支冲突的处理，提高了 CPU 的执行效率。这样的设计在实际应用中可能更适合需要高性能的场景。

## 4.3 主要故障与调试

### 4.3.1 LBU 取数据功能异常

**故障现象：**LBU 功能测试时本应在 LED 最右侧两个单元显示递增的两位数，其余显示为 0，但是我的 LED 会每次递增加 4。

**原因分析：**手册中的 LBU 功能并未描述地十分详细，我便以为从数据存储器取出的数据只有最低 8 位进行扩展即可得到结果，结果中间间隔的 24 位数据就被跳过了。

**解决方案：**在 RDin 的多路选择器只前增加多路选择器和符号扩展器，从数据存储器的地址输入截取低 2 位作为多路选择器的输入选取相应的字节，LBU 信号决定是否把扩展后的 32 位数据输出给 RDin。

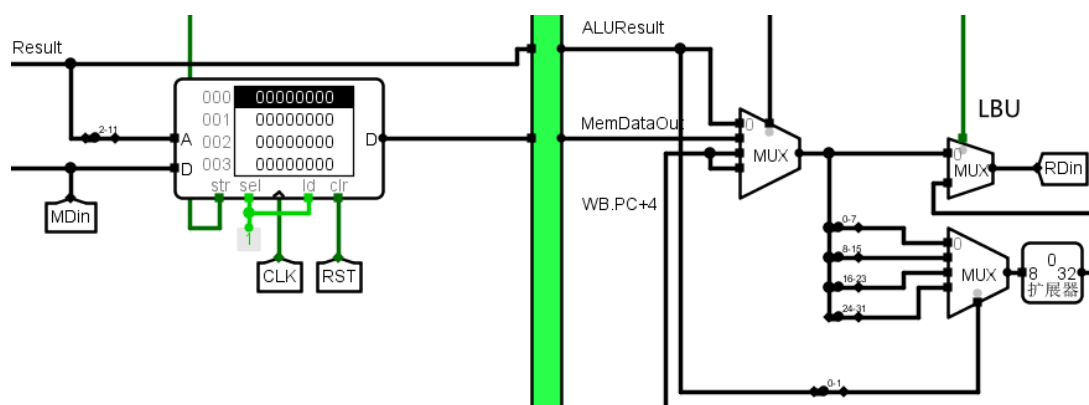


图 4.10 断点设置图

### 4.3.2 ecall 指令的暂停功能问题

**故障现象：**根据任务书要求，ecall 指令的暂停功能条件为 a7 寄存器为 50。而实际却无法达到暂停效果。



# 华中科技大学课程设计报告

**原因分析：**查看测试程序发现，实际的暂停条件为 a7 为 10 时进行暂停，与任务书要求不符。

**解决方案：**以实际测试程序为准，调整 ecall 暂停指令的条件为

## 4.3.3 理想流水线故障

**故障现象：**理想流水线数据清零，无法执行后续指令。

**原因分析：**两个输入引脚高电平有效，在原来的理想流水线里悬空了，所以造成了错误。

**解决方案：**两个引脚用连接常量 0，不能悬空。

## 4.4 实验进度

表 4.1 课程设计进度表

时间	进度
第 1-3 天	学习单周期 CPU 架构，阅读课设任务书和 RISC-V 指令手册，完成各功能部件的实现，构造基础数据通路，完成 CCAB 指令的扩展，调试并排除故障。
第 4-5 天	学习 5 段流水 CPU 架构，完成 5 段流水 CPU 的接口部件实现，完成基础数据通路，测试理想流水线功能。
第 5-6 天	完成气泡流水线部分
第 6-8 天	完成重定向流水线
第 10 天	完成单级中断
第 11 天	完成多级中断
第 12-13 天	完成流水中断
第 14 天	完成动态分支预测



## 5 团队任务

### 5.1 选题与设计

团队任务选定为实现 2048 游戏，玩家利用戳工具选中键盘组件后通过输入 WSAD 来完成上下左右的按键操作，按下一个按键就是触发一个中断源，CPU 会执行相应的中断服务程序实现对棋盘的更新。。

### 5.2 团队任务负责部分

叶逸飞：主要负责前期总体方案设计。

季思涵：主要负责输出模块设计及实现及测试。

宋佳婧：主要负责输入模块以及中断硬件相关的设计实现。

高僊：主要负责答辩 PPT 和演示视频的拍摄。

### 5.3 实现效果

游戏界面如图 5.1。



图 5.1 演示界面

## 6 设计总结与心得

### 6.1 课设总结

1) 成功设计并完成实现 24+4 条指令的单周期 CPU，这项任务构建在组成原理课程实验的基础之上，并进行了补充。尽管相对较为简单，但为后续课设实验内容奠定了基础。

2) 独立完成了单周期多级中断机制的设计，实现了支持嵌套中断的电路。这项工作展现了对中断机制的深入理解和熟练应用。

3) 设计并成功完成理想流水线，其中主要工作包括流水接口的设计、流水线的分割和停机、显示以及清空流水等功能。这一设计在流水线处理中发挥了关键作用。

4) 巧妙地设计了气泡流水线，通过流水阻塞和插入气泡等方法解决了各种冲突。在理想流水线的基础上，成功将这一思想付诸实践。

5) 成功设计并完成了重定向流水线，通过引入旁路机制，着重解决了重定向的数据来源和选择信号的产生问题。这为流水线执行提供了更加灵活和高效的解决方案。

6) 在基于重定向流水的单级中断机制上进行设计，并解决了因流水而引入的问题。成功实现了动态分支预测功能，大幅提高了 CPU 效率。这体现了对流水线设计的深入理解和创新思维的应用。

### 6.2 课设心得

这次的计算机组成原理课程设计可谓是我参与过的所有实验和课程设计中难度最大的一门。回顾整个过程，充满了成就感，但也有很多值得深思和体会的细节。

一开始接触这个实验时，我感到一头雾水，不知道如何着手。通过翻阅课本和 PDF 参数资料，逐渐有了思路，开始明白应该如何着手。在第一个实验（单周期 CPU）中，由于查阅资料，花费了较多时间。然而，随着流水线的处理，情况变得相对简单，因为只需要进行改造，尤其是理想流水线和气泡流水线，书上都提供了详细的步骤。我认为最具挑战的部分是与中断相关的实验内容，但在朋友们和网上各种资料的帮助下，我最终成功完成了这一部分。

然而，在整个课程设计过程中，我也想提出一些建议和改进。首先，我认为可以

# 华中科技大学课程设计报告

---

再增加两条 CCAB 指令。在完成课程设计的过程中，我最喜欢研究各种指令的详解，实现 CCAB 指令也是我非常喜欢的一部分。其次，在中断部分，特别是流水线中断，提供的参考资料相对不足。在仅有两周的时间内实现这么多内容，充分的参考资料对于提高效率非常重要，可以节省很多搜索资料的时间。

总体而言，这次课程设计的内容非常丰富，同时也包含了很多能够得到加分的额外任务。对于我来说，理论课的学习已经足够满足基本要求，而流水线的部分课本上也有详细的介绍，因此实验可以考虑设置成选修。尽管如此，我还是全力完成了基础实验和部分额外内容。收获颇丰，实际动手连接电路让我面临了很多细节上的问题，加深了对 CPU 内部指令和数据流动的认知。特别是在填写关于硬布线控制器的表达式的 Excel 表格之后，我对一些基本指令在 CPU 中的执行过程有了更清晰的了解。

整个电路设计的过程让我感受到计算机不再是一个黑盒，我们清楚地知道了它的工作原理。从逻辑上来说，CPU 的设计十分简洁，它不知疲倦地执行一条条指令，尽管在分支、中断等情况下会面临一些问题，但人们也设计出了一套完美的解决方案。

关于气泡流水线和重定向流水线的部分，实际动手实现电路时，educoder 和教材提供的可参考内容相对较少。尤其是在需要亲自动手实现电路时，我们需要在网络上查询大量的资料。虽然自学能力对于本科生和硕博研究生来说是不可或缺的，但考虑到大三还有大量其他课程要学，以及个人感兴趣的方向需要深入研究，我认为这些教学资料完全可以由老师详细地给出，参考国外实验指导，而不是让学生花费大量时间在网络上查找不同质量的资料。

最后，再次感谢各位老师为实验作出的付出。这门课程设计不仅让我深入理解计算机组成原理，也让我体验到了解决实际问题的挑战和乐趣。

# 华中科技大学课程设计报告

---

## 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, 周军龙, 肖亮. 计算机组成原理实验指导与习题解析. 北京: 人民邮电出版社, 2022.
- [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.

• 指导教师评定意见 •

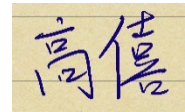
---

## 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：高僊

A handwritten signature in blue ink on a yellow background. The characters are '高僊' (Gao Xian), written in a cursive style.