華中科技大學

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：	U202112071
姓　　名：	王彬
电　　话：	15005276201
邮　　件：	2457537174@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。见下方 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	指令格式参考 MIPS32 指令集，最终功能以 MARS 模拟器为准。
2	ADDI	立即数加	
3	ADDIU	无符号立即数加	
4	ADDU	无符号数加	
5	AND	与	
6	ANDI	立即数与	
7	SLL	逻辑左移	
8	SRA	算数右移	
9	SRL	逻辑右移	
10	SUb	减	
11	OR	或	

#	指令助记符	简单功能描述	备注
12	ORI	立即数或	
13	NOR	或非	
14	LW	加载字	
15	SW	存字	
16	BEQ	相等跳转	
17	BNE	不相等跳转	
18	SLT	小于置数	
19	STI	小于立即数置数	
20	SLTU	小于无符号数置数	
21	J	无条件转移	
22	JAL	转移并链接	
23	JR	转移到指定寄存器	If $v0==10 halt(停机指令)
24	SYSCALL	系统调用	else 数码管显示$a0 值
25	MFC0	访问 CP0	中断相关，可简化
26	MTC0	访问 CP0	中断相关，可简化
27	ERET	中断返回	异常返回
28	SLL	逻辑左移	
29	SRA	算术右移	
30	SB	低位字节存入内存	
31	BGE	大于等于时分支	

2 总体方案设计

2.1 单周期 CPU 设计

本阶段使用硬布线控制器实现 RISC-V 单周期 CPU。本次我们采用硬布线控制方案，且指令和数据分开存储，实现指令寄存器和数据寄存器不共用一个存储器的方式完成方案的设计。我们使用时钟周期进行同步，在单一周期中，控制器首先读取指令并通过硬布线控制器转化为 ALU 对应指令，取得目的寄存器或立即数的数值后经过计算再写回。

总体结构图如**错误!未找到引用源。**所示。

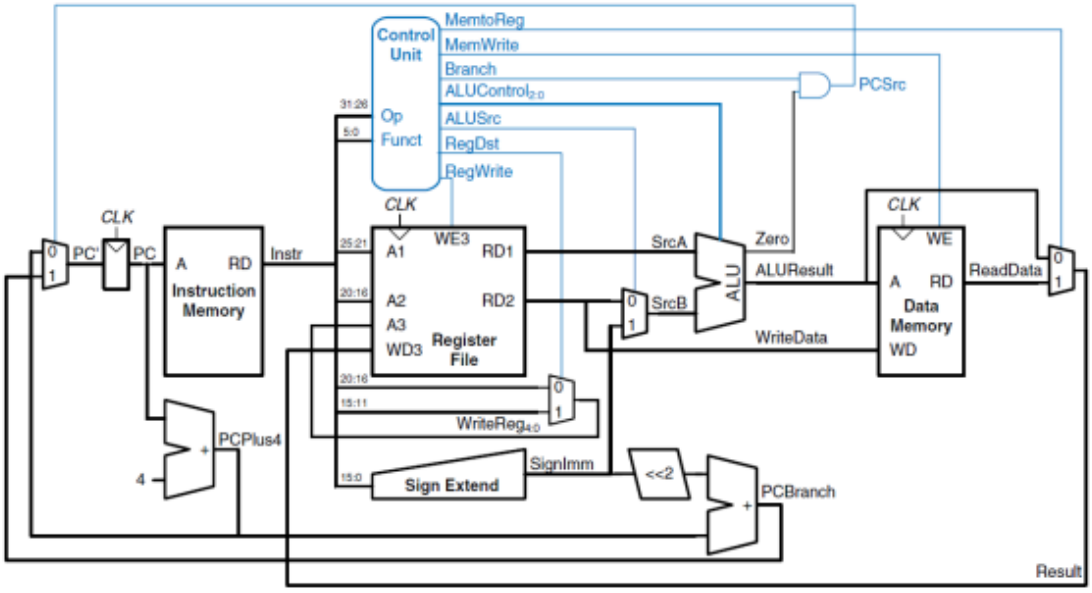

图 2.1 总体结构图

2.1.1 主要功能部件

单周期 CPU 的几个主要部件具体设计思路如下。

1. 程序计数器 PC

程序计数器 PC 用一个 32 位寄存器实现，用于存储下一条待运行指令的地址。它由多个下址字段通过多路选择器决定，在每个时钟周期后将待取指令送至 IM，等待下一步的解析。

2. 指令存储器 IM

指令存储器（IM）存储程序的可执行指令，由 PC 的输入对应地址所决定。每条指令均为 32 位，这是由于我们实现的指令集 RISC-V 为定长指令集。

3. 运算器

运算器根据控制器给出的 ALU_OP 指令，对输入操作数进行对应运算操作。表 2.1 为算术逻辑运算单元的输入输出接口及其功能的叙述。

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

ALU 组件对于不同的运算器输入 ALU_OP，其功能表如下。

表 2.2 算术逻辑运算 ALU_OP 与功能描述

输入(ALU_OP)	功能描述
0	逻辑左移
1	算术右移
2	逻辑右移
3	无符号乘法，高 4 字节输出为 Result2
4	无符号除法，余数输出为 Result2
5	加法

输入(ALU_OP)	功能描述
6	减法
7	按位与
8	按位或
9	按位异或
10	按位或非
11	有符号小于比较
12	无符号小于比较

4. 寄存器堆 RF

寄存器堆 RF 由 32 个 32 位寄存器组成，我们给出其输入与输出引脚定义。

表 2.3 寄存器输入与输出引脚与功能描述

引脚	输入/输出	位宽	功能描述
R1#	输入	5	源操作寄存器 1 下地址
R2#	输入	5	源操作寄存器 2 下地址
W#	输入	5	写操作寄存器下地址
WE	输入	1	写使能信号
RDin	输入	32	写入寄存器数据
R1	输出	32	源操作寄存器 R1#数据
R2	输出	32	源操作寄存器 R2#数据
CLK	输入	1	时钟信号

2.1.2 数据通路的设计

对于单周期处理器，我们需要在一个时钟周期内实现指令的取指、执行等操作。对于定长 RISC-V 指令集，不同类型的指令将有不同的数据通路，对于 R 型、I 型、J 型等指令均有不同类型的数据通路。例如，对于 R 型指令，需要先从指令存储器中读出指令字中的源寄存器字段 rs、rt 分别送入寄存器堆的两个读寄存器编号端 R1#和 R2#，并将 rd 送至写寄存器编号端 W#，将 R1 和 R2 端口输出至运算器，并以指令字

中的 ALU_OP 字段决定 ALU 的运算方式，运算结果被送入寄存器的 Din 端口写回目的寄存器中；对于 I 型指令，则需要由 16 位立即数 imm16 通过符号扩展单元转换为 32 位后参与运算。

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.。

表 2.4 主控制器控制信号的作用说明

控制信号	取值	说明
ALU_OP	0~12	运算器操作控制符（4 位），具体控制方式见 2.1.1 运算器部分
MemToReg	0	数据存储器数据不写入寄存器堆
	1	数据存储器数据写入寄存器堆
MemWrite	0	数据存储器不执行写入
	1	数据存储器执行写入 R2 或 SB 指令下的特殊处理数据
ALU_Src	0	运算器的第二个操作数为寄存器 R2
	1	运算器的第二个操作数为 32 位扩展立即数
RegWrite	0	寄存器堆不执行写入
	1	寄存器堆写使能，将 Din 写入寄存器 Rd
ecall	0	该指令不为 ecall
	1	该指令为 ecall
S_Type	0	该指令不为 S 型指令
	1	该指令为 S 型指令
BEQ	0	该指令不为 BEQ
	1	该指令为 BEQ
BNE	0	该指令不为 BNE
	1	该指令为 BNE
BGE	0	该指令不为 BGE
	1	该指令为 BGE

控制信号	取值	说明
Jal	0	该指令不为 Jal
	1	该指令为 Jal
Jalr	0	该指令不为 Jalr
	1	该指令为 Jalr
SB	0	该指令不为 SB
	1	该指令为 SB
RS1	0	该指令无需使用寄存器 rs
	1	该指令需要使用寄存器 rs
RS2	0	该指令无需使用寄存器 rt
	1	该指令需要使用寄存器 rt
CSRRSI	0	该指令不为 CSRRSI 指令
	1	该指令为 CSRRSI 指令
CSRRCI	0	该指令不为 CSRRCI 指令
	1	该指令为 CSRRCI 指令

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。主要控制信号表的框架如表 2.5 所示，对于特殊指令所建立的特殊指令信号，如我们实现的 CCAB 指令的 BGE、SB 指令等，它们具有单独控制信号，因此不在该表中列出。

表 2.5 主控制器控制信号框架

指令	MemToReg	MemWrite	ALU_Src	RegWrite	S_Type	ALU_op	RS1	RS2
R 型	0	0	0	1	0	依据指令	1	1
I 型	依据指令	0	依据指令	依据指令	0	依据指令	依据指令	依据指令
B 型	0	0	0	0	0	依据指令	1	1
S 型	0	1	1	0	1	依据指令	1	1
J 型	0	0	0	1	0	依据指令	0	0

2.2 中断机制设计

2.2.1 总体设计

中断机制包括内部中断和外部中断，我们设计的中断类型为可屏蔽外部中断。在本次实验设计中，我们分别构建了单周期 CPU 单级中断及多级中断、重定向流水线单级中断，支持 3 个按键中断事件。对于单级中断，CPU 执行中断程序时需要等待当前中断服务完成，才能响应其它中断请求；对于多级中断，我们定义的中断优先级为 3>2>1，高优先级的中断可以打断低优先级中断服务，并进入更高等级的中断服务程序。

2.2.2 硬件设计

本实验的中断设计我们使用硬件方式进行存储。在对应中断按键被按下后，中断请求将送入对应 D 触发器，而后在下一时钟周期中，触发器输出与中断屏蔽位进行与运算，由此判定是否响应中断请求。其中，对于单级中断和多级中断的中断设计分别如下。

（1） 单级中断

对于单级中断，我们对每个中断信号均使用一个中断寄存器保存当前的中断程序返回地址，从而实现对每个中断进行响应的任务。同时，用优先编码器实现中断识别，并通过硬逻辑实现中断向量表，对于不同的中断响应程序均给定对应入口地址。

（2） 多级中断

对于多级中断，我们需要三套中断寄存器 mEPC 保存中断程序的返回地址，并采用硬件堆栈的方式进行储存。同样地，为了实现中断优先级的动态选择，我们使用优先编码器对中断信号进行选择，以支持中断程序的三个优先级的切换。

2.2.3 软件设计

在软件设计方面，我们主要使用 csrrsi 和 csrrci 指令实现中断的开和关。指令通过硬布线控制器进行译码，得到的 csrrci 信号与 INT 信号进行逻辑或后置位至 INT_IE 寄存器中，而 csrrsi 与 URET 信号逻辑或后为该寄存器的使能端，通过这样的设计实现中断的关与开。

2.3 流水 CPU 设计

2.3.1 总体设计

本实验中，我们实现了五段流水线 CPU。我们将指令的执行过程分为五个功能段，即取指 IF、译码 ID、执行 EX、访存 MEM 和写回 WB。每个阶段完成对应操作后，我们将运行结果保存入对应的流水寄存器堆中，例如在取指和译码过程中，我们增设 IF/ID 寄存器存放上一个阶段的操作结果，供下一阶段使用。所有寄存器均使用统一时钟信号，对应逻辑架构示意见图 2.2。

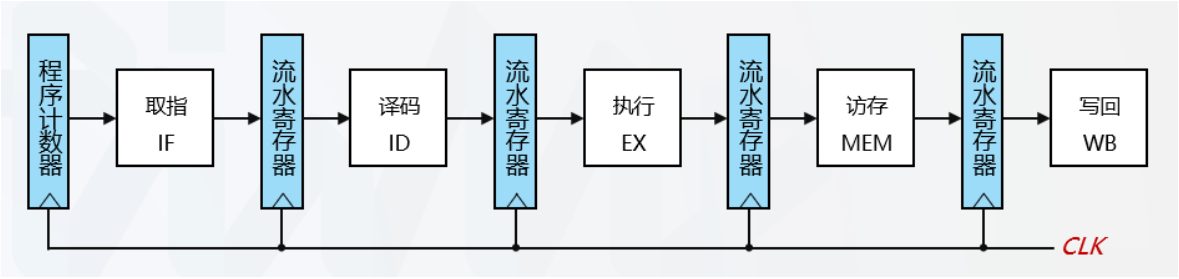

图 2.2 流水逻辑架构

2.3.2 流水接口部件设计

（1） IF/ID 流水寄存器堆

该寄存器堆连接取指 IF 和译码 ID 功能段，寄存器堆需要存放指令寄存器 IR、该指令地址 PC 和下一指令地址 PC+4。

（2） ID/EX 流水寄存器堆

该寄存器堆连接译码 ID 和执行 EX 功能段，寄存器堆需要存放硬布线控制器输出的所有控制信号、寄存器堆的输出 R1/R2、扩展至 32 位的立即数 Imm、该指令地址 PC 和下一指令地址 PC+4。

（3） EX/MEM 流水寄存器堆

该寄存器堆连接执行 EX 和访存 MEM 功能段，寄存器堆需要存放访存需要的控制信号，诸如 MemWrite 或 MemToReg 信号。同时还需要数据输出 ALU 部件的运算结果、该指令地址 PC 和下一指令地址 PC+4，以及作为读指令需要的指令寄存器 IR。

（4） MEM/WB 流水寄存器堆

该寄存器堆连接访存 MEM 和写回 WB 功能段，该流水寄存器堆需要的寄存器和 EX/MEM 段类似，需要对应控制信号、向前反馈信号及向前回传的数据。

2.3.3 理想流水线设计

我们根据上述设计，将单周期 CPU 直接划分为不同的部件，并在不同的功能段间加入流水寄存器，即可构成理想流水线。

需要注意的是，理想流水线会出现指令相关和数据依赖，导致流水线的阻塞或暂停。如果两条指令间存在某种依赖关系，可能发生流水线冲突，包括结构冲突、控制冲突或数据冲突。

2.4 气泡式流水线设计

在指令流水线中，为了消除因指令间存在的依赖关系而导致的数据冲突这三类流水线冲突，可以通过插入气泡来加以解决。在五段流水线设计中，ID 段从寄存器堆中取操作数时才会发生数据相关，故只需要考虑 ID 段指令与 EX、MEM、WB 段的 3 条指令之间的数据相关性。

ID 段和 WB 段的数据相关采用先读后写的方式解决，寄存器堆写入控制采用下跳沿触发，而所有流水线均采用上跳沿触发，这样在一个时钟周期的中间时刻（下跳沿）就可以完成寄存器堆的数据写入，在时钟周期的后半段就可以利用组合逻辑读取寄存器正确的值。一旦解决了 ID 段和 WB 段之间的数据相关，我们只需要考虑连续 3 条指令的数据相关性，下面我们处理 ID 段和 EX、MEM 段之间的数据相关性。

我们在流水线中加入硬件逻辑实现 ID 段与 EX、MEM 段指令的数据相关性检测。为了确认 ID 段指令使用的源寄存器是否在前两条指令中写入，只需要检查 EX、MEM 段的寄存器堆写入控制信号 RegWrite 是否为 1，且写寄存器编号 Write#是否和源寄存器编号相同即可，因此流水线中的数据相关检测逻辑如下。

```
DataHazzard = RsUsed & (rs !=0) & EX.RegWrite & (rs==EX.WriteReg#)
            + RsUsed & (rt !=0) & EX.RegWrite & (rt==EX.WriteReg#)
            + RsUsed & (rs !=0) & MEM.RegWrite & (rs==MEM.WriteReg#)
            + RsUsed & (rt !=0) & MEM.RegWrite & (rt==MEM.WriteReg#)
```

我们接下来考虑如何暂停 IF、ID 段指令的执行以及如何插入气泡的问题。插入气泡可以参考控制相关的流水清空信号 Flush，当发生数据相关时给 ID/EX 流水寄存器一个同步清空信号 Flush；为了暂停 IF、ID 段指令的运行，需要保证程序计数器 PC 和 IF/ID 流水寄存器的值保持不变。这样，只需要控制寄存器使能端即可，并将数据

相关检测逻辑生成的数据相关信号 DataHazzard 作为暂停信号 Stall 取反后送入对应使能端即可。

2.5 重定向流水线设计

气泡流水线通过延缓 ID 段取操作数动作的方式解决数据冲突问题，但气泡的插入会影响指令流水线的性能。我们下面考虑重定向流水线设计，即除 Load 类访存指令外，目的操作数都已存放至 EX/MEM、MEM/WB 流水寄存器中，需要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。

EX/MEM 流水寄存器中的 AluResult 或 WB 段的 WriteBackData 直接送到 EX 段的 rs 寄存器处，作为 SrcA 送入 ALU 中参与运算。rt 寄存器也可以进行这种处理，重定向方式无需插入气泡，可以解决大部分数据相关问题的同时，避免插入气泡引起的流水线性能下降，大大优化流水线性能。

需要注意的是，如果相邻两条指令中，前一条指令为访存指令（即 Load-Use 相关），这类数据相关不能采用重定向的方式进行处理，必须强制插入一个气泡来消解这种相关。在本实验中，我们插入气泡在 ID 段实现。

具体地，当 Load-Use 相关发生时，我们需要暂停 IF、ID 段指令的执行，并在 EX 段中插入气泡。同时，控制 PC 使能端 EN、IF/ID 使能端 EN、ID/EX 清零端 CLR；在 EX 段执行分支指令时会清空 ID 段、EX 段中的误取指令，使用 IF/ID 清零端 CLR 和 ID/EX 清零端 CLR。

2.6 动态分支预测机制

在重定向机制之上，为了进一步提高流水线效率，我们需要减少流水线中的控制冲突对流水线的影响。亦即，我们需要减少分支延迟损失，将分支指令的执行尽可能提前。

动态分支预测依据分支指令的分支跳转历史，对预测策略进行动态调整，预测命中率较高。程序中的分支指令的分支局部性让分支行为可以预测，一种实现动态分支预测的策略是分支预测缓冲器（BTB 表）。该表用于存放分支指令的分支跳转历史统计信息，每个表项包括 valid 位、分支地址指令、分支目标地址、历史跳转信息描述位及置换标记 5 项，本质上是一个全相联 cache。

每一条分支指令执行时，都会将分支指令地址、分支目标地址是否发生跳转等信

息送入 BTB 表。BTB 以分支指令地址为关键字，在 BTB 表内进行全相联并发比较，如果数据缺失则将分支指令的相关信息载入；如果数据命中，则更具本次分支是否发生跳转调整对应表项中的分支预测历史位，以提高预测准确率，并处理与淘汰相关的置换标记信息。

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）

Logism 实现：

使用一个 32 位寄存器实现程序计数器 PC，触发方式为下降沿触发，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 所示。

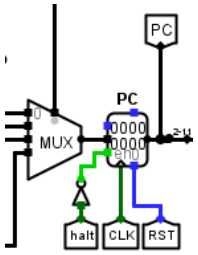

图 3.1 程序计数器（PC）

2) 指令存储器（IM）

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 所示。

图 3.2 指令存储器（IM）

3) 运算器

Logisim 实现：使用运算组件堆构成运算器，使用多路选择器通过 AluOp 选择运算功能，分别实现表 2.2 中列出的所有 ALU 功能。其中，对于乘法和除法运算，将其输出以 Result2 的 32 位字段输出至结果；特别地，对于比较运算操作，用“<”“≥”输出引脚进行表示，如图 3.3 所示。

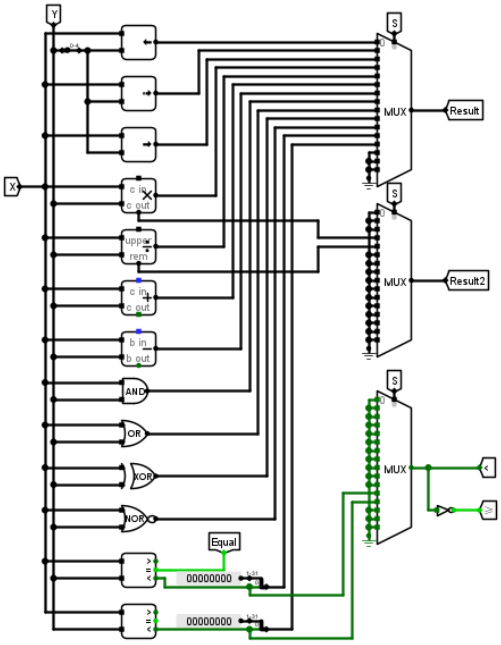

图 3.3 运算器 ALU

4) 寄存器堆 RF

寄存器堆 RF 模块使用 32 个双字寄存器按地址排列的阵列堆，在单周期 CPU 时钟端上跳沿触发，在各类流水线 CPU 中均以下跳沿触发，该组件的实现见图 3.4。

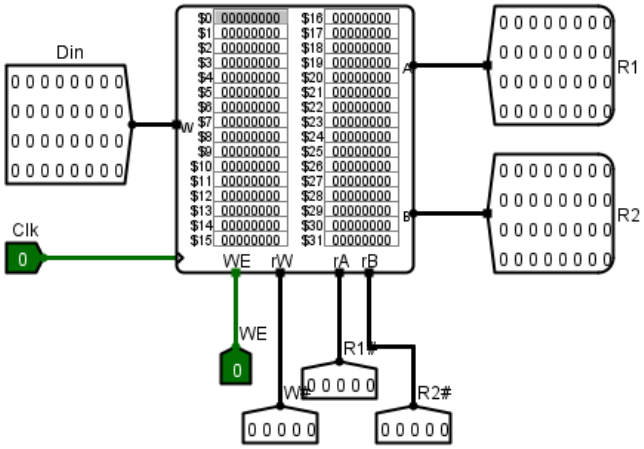

图 3.4 寄存器堆 RegFile 实现

3.1.2 数据通路的实现

根据不同的指令类型，我们实现的数据通路均有一定的不同。总体上，数据的流动顺序为 PC->指令寄存器 IM->控制器、寄存器->运算器 ALU->数据寄存器 MEM，对于部分指令，在这一系列数据通路后还需要写回寄存器。

首先，如图 3.5（左）所示，我们需要实现 PC 下址的选择。我们使用多路选择器对 PC+4 字段进行实现，选择端使用优先编码器实现如图 3.5（右）。硬件实现需要将 PC 加上不同的偏移量，再从中选择正确的下址地址。这样我们可以实现顺序下址地址和 I、J、B 型指令下址的选择，使得程序正常跳转。

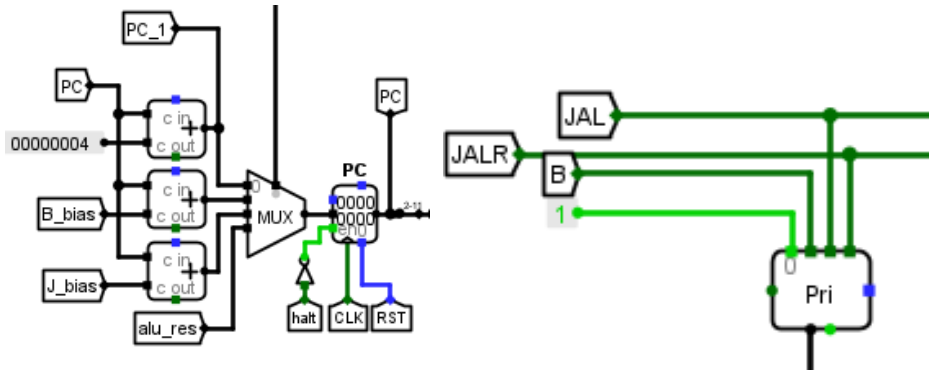

图 3.5 数据通路 PC 下址实现与选择端

随后，我们对取得的指令进行解析。控制器将对指令字进行运算并得到各式控制信号，这将在下一小节集中论述。同时我们将把指令中蕴含的不同数据进行取得，以便后续的操作。寄存器的写入使用控制器输出的写使能信号 WE 控制，并将从寄存器堆读出的两个源寄存器值经由 R1、R2 端口输出到运算器，这一部分数据通路见图 3.6（左）。

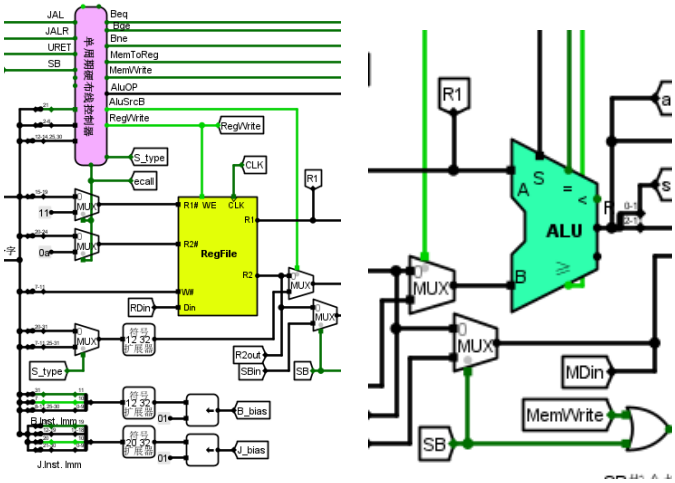

图 3.6 指令解析及运算数据通路

指令字通过控制器输出的 AluOp 字段将控制 ALU 功能，将寄存器堆的输出进行运算。其中，对于不同的指令类型，运算器的 B 输入字段有不同的输入方式，例如指令字 S 类型输出等等，如图 3.6（右）所示。其中，对于 CCAB 指令，例如 SB 指令，我们使用了自定义的硬件计算通路实现，因此需要用控制器单独输出的 SB 信号堆输入数据进行调控。

对于 I 型指令，运算器 ALU 的运算结果会作为地址访问数据存储器 MEM。例如访存指令 sw，寄存器堆读出的 rt 寄存器值将送入 MEM 写数据端口 WD。对于部分指令，寄存器堆需要写入的，RegWrite 信号设置为 1，并将读取的内存数据实现寄存器写入。

除了上述数据通路部分，我们还需要实现停机的硬件逻辑。如图 3.7 所示，当收到解析得到的 ecall 信号且寄存器堆输出 R1 不等于 0x22 时，控制 CPU 进行停机。否则，处理器将 a0 值输出至 LedData。为了保证数码管的输出连续，我们采用一个 32 位寄存器对输出数据进行锁存。

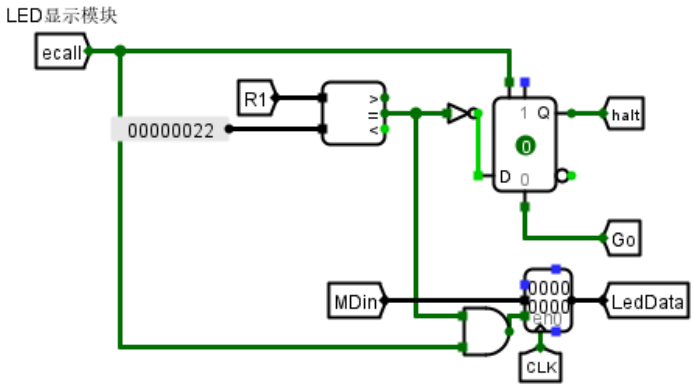

图 3.7 停机及 LedData 显示

这样我们对数据通路实现了总体的支持，最终我们得到的单周期 CPU 如图 3.8 所示。其中，我们对 CCAB 指令实现了硬件级别的支持，例如对于 SB 指令进行了硬布线的取字节操作支持。

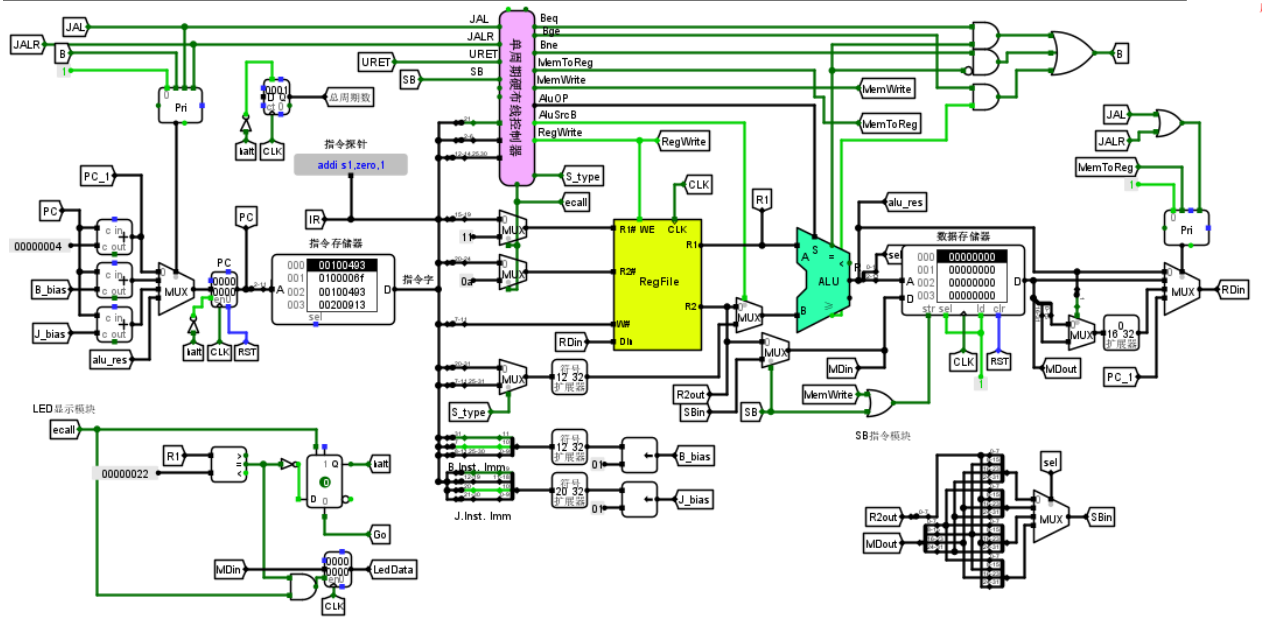

图 3.8 数据通路总结构

3.1.3 控制器的实现

根据总体方案设计（见 2.1.3 小节），我们在 Logisim 上具体实现主控制器，其中各个控制信号的取值见表 3.1。

表 3.1 各控制信号取值表

指令	MemToReg	MemWrite	ALU_Src	RegWrite	S_Type	RS1	RS2
add	0	0	0	1	0	1	1
sub	0	0	0	1	0	1	1
and	0	0	0	1	0	1	1
or	0	0	0	1	0	1	1
slt	0	0	0	1	0	1	1
sltu	0	0	0	1	0	1	1
addi	0	0	1	1	0	1	0
andi	0	0	1	1	0	1	0
ori	0	0	1	1	0	1	0
xori	0	0	1	1	0	1	0
slti	0	0	1	1	0	1	0

指令	MemToReg	MemWrite	ALU_Src	RegWrite	S_Type	RS1	RS2
slli	0	0	1	1	0	1	0
srli	0	0	1	1	0	1	0
srai	0	0	1	1	0	1	0
lw	1	0	1	1	0	1	0
sw	0	1	1	0	1	1	1
ecall	0	0	0	0	0	0	0
beq	0	0	0	0	0	1	1
bne	0	0	0	0	0	1	1
jal	0	0	0	1	0	0	0
jalr	0	0	0	1	0	1	0
CSRRSI	0	0	0	0	0	0	0
CSRRCI	0	0	0	0	0	0	0
URET	0	0	0	0	0	0	0
sll	0	0	0	1	0	1	1
sra	0	0	0	1	0	1	1
sb	0	0	1	1	1	1	1
bge	0	0	0	0	0	1	1

通过对相关指令的控制信号值的确定，我们可以生成单周期硬布线控制器表达式，再对组合逻辑与控制信号生成器进行封装，以此实现主控制器硬布线逻辑，见图 3.9。

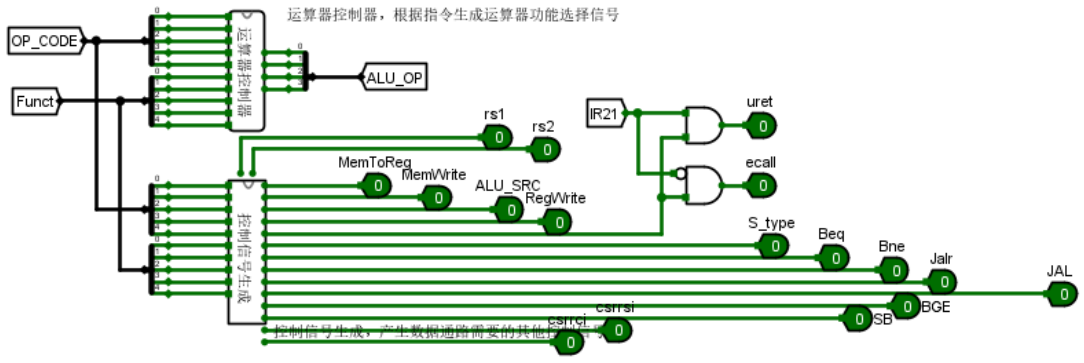

图 3.9 主控制器硬布线逻辑

3.2 中断机制实现

3.2.1 单级中断实现

我们采取实验包给定的电路产生中断信号，如图 3.10 所示。对于单级中断，我们使用两个 D 触发器分别在中断源产生脉冲时载入中断信号、并在下一时钟上升沿载入保存；之后，我们使用优先编码器对中断进行输出，见图 3.11。

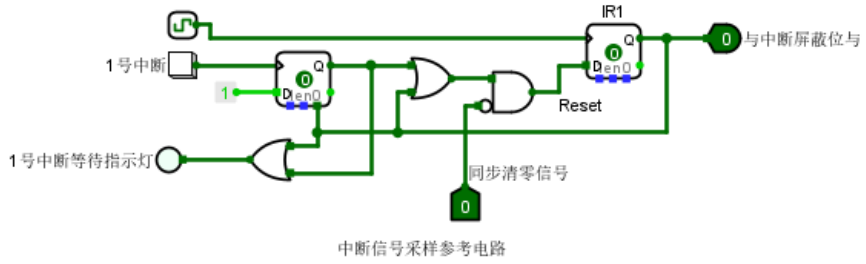

图 3.10 中断信号产生电路

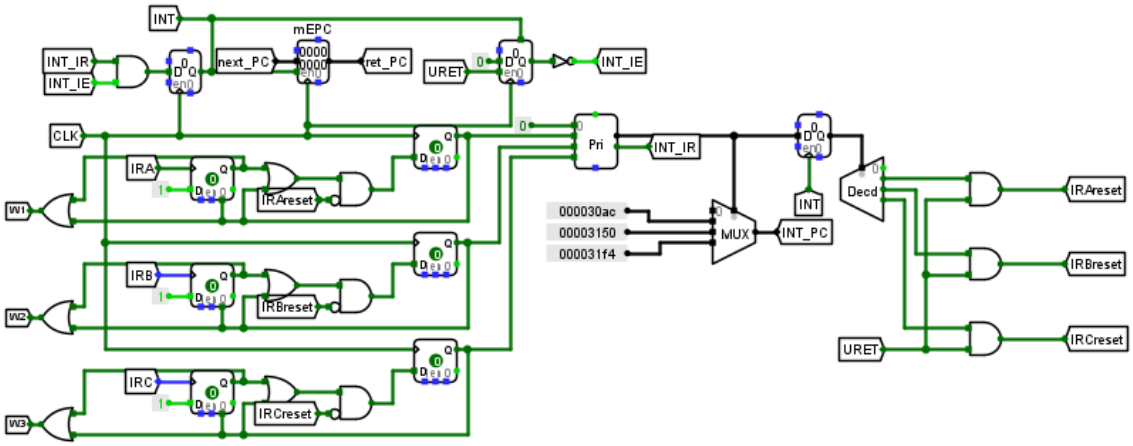

图 3.11 单级中断寄存与编码输出

在获取中断信号后，我们使用硬逻辑编码出各中断响应程序的起始地址作为中断入口逻辑，通过优先编码器的输出作为选择信号，选择多路选择器前存储的响应程序入口地址，并用 INT_PC 通道连接至 PC 下址。换言之，如果中断信号产生，PC 寄存器的下址即为对应中断响应程序的下一地址。

单级中断在响应中断后会利用中断指令关中断，并在硬布线控制器传出 URET 信号后开中断。在响应中断程序后，中断寄存器 mEPC 需要将断点存入，并在中断响应退出后（即 URET 退出时）恢复原 PC，这可以通过将原 PC 下址运算和中断寄存器下址用多路选择器并联实现。

3.2.2 多级中断实现

多级中断即在响应中断程序时，终端服务程序可以被更高优先级的中断请求中断。因此，我们需要实现硬件堆栈对多级中断寄存器进行存储。其实现方式为，先将中断信号用优先编码器进行编码，再使用解码器解码出对应的响应信号，并对每个中断信号用一个 mEPC 存储其返回 PC 值，通过这种方式支持中断的现场保护。随后，我们使用一个选择器对多个 PC 值进行选择，以筛选出真实的指令下址。多级中断响应及寄存部分详见图 3.12。

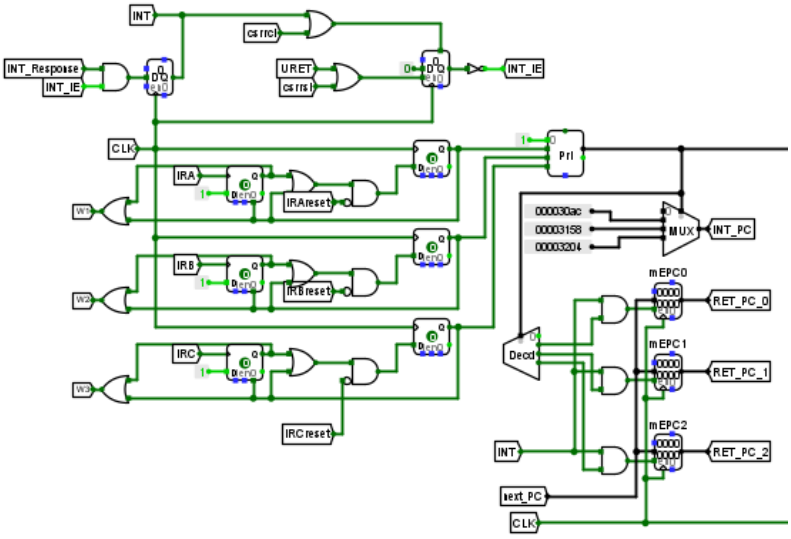

图 3.12 多级中断响应及寄存部分

在中断响应程序结束后，我们通过多路选择器对原程序 PC 值进行选择，以重定向至原先程序的执行地址。中断清零信号则参考单级中断逻辑，以寄存的中断号进行解码，解码信号与 URET 信号逻辑与后即为对应中断的清零信号，现场保护及清零信号详见图 3.13。

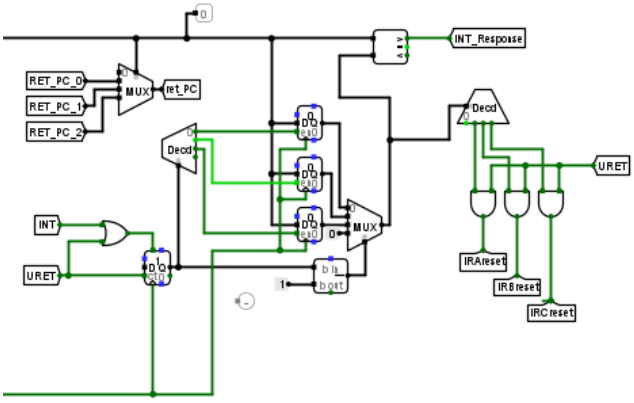

图 3.13 多级中断现场保护及清零信号

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

在本部分，我们主要阐述流水线接口的电路设计。在流水接口部件中，需要实现一系列的程序状态寄存器来寄存上一流水线的运行结果，同时还需要存储相应的复位信号（flush 等）对流水线的不同实现方式进行支持。例如，我们制作的 ID/EX 段流水接口如图 3.14 所示。

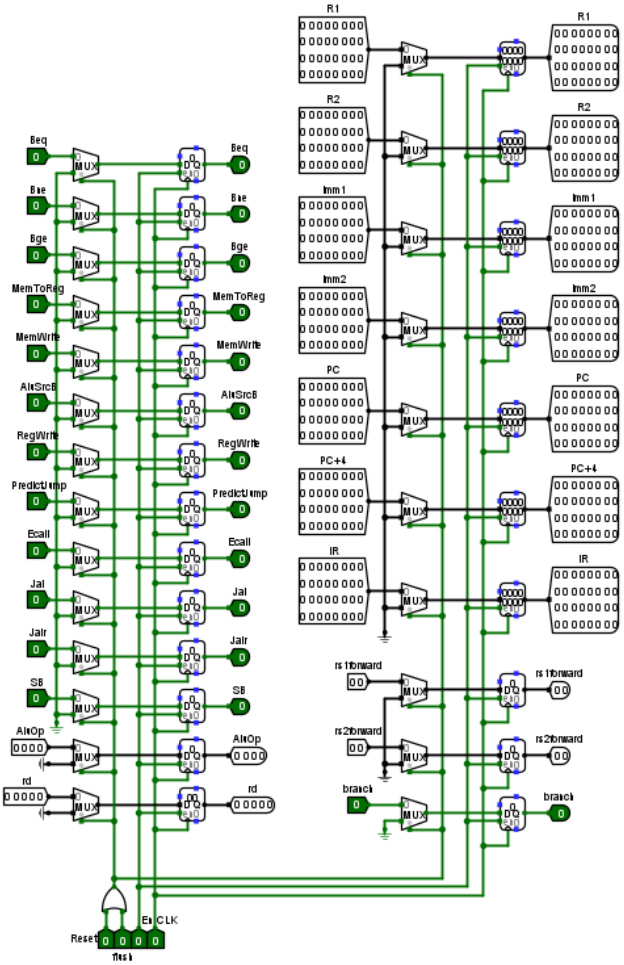

图 3.14 ID/EX 段流水接口组件

对于不同的流水线段，需要传递的控制信号也有差异。控制信号控制的功能段和信号来源并不一致，因此对于不同的流水接口部件，其进行存储的信号也有区别。在上一流水段执行完成后，硬件会通过时钟上升沿寄存入流水寄存器，再在下一时钟周期中取用上一控制信号。

3.3.2 理想流水线实现

理想流水线可以直接将单周期 CPU 修改为流水线形式。我们按照 IF、ID、EX、MEM 和 WB 阶段划分 CPU 流水，尤其需要注意在流水线中寄存器堆 RF 以下降沿跳变触发。我们设计的理想流水线 CPU 见图 3.15。

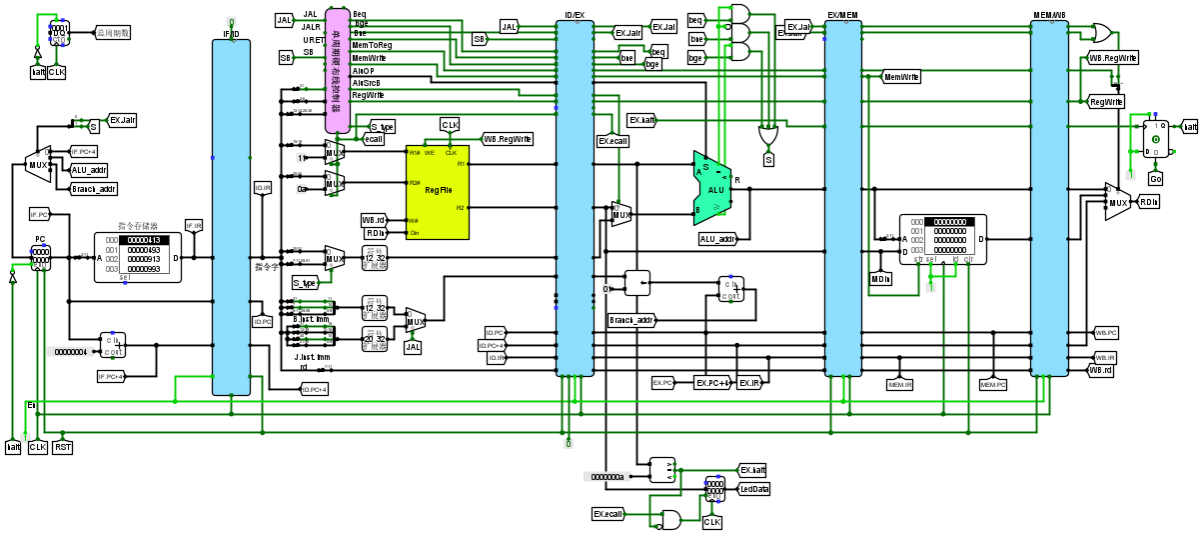

图 3.15 理想流水线电路实现

3.4 气泡式流水线实现

气泡流水线的相关设计已在 2.4 小节中阐述。根据我们先前的设计，气泡的产生和处理逻辑见图 3.16，我们需要检测 EX 和 MEM 段寄存器堆写入信号 RegWrite、写寄存器编号与源寄存器编号，对应表达式也已经在前文中阐明。

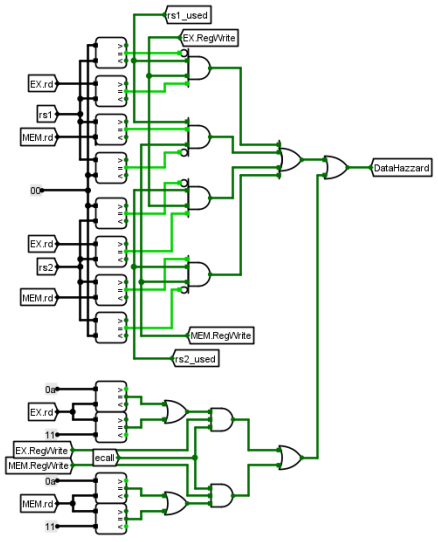

图 3.16 气泡处理逻辑信号生成

在实现气泡处理逻辑后，我们将其添加至流水线中。产生的 DataHazzard 信号与 BranchTaken 信号进行逻辑或后产生 flush 信号，并与 ID/EX 流水寄存器复位端 RST 相连即可完成气泡流水线设计，相关流水线如图 3.17 所示。

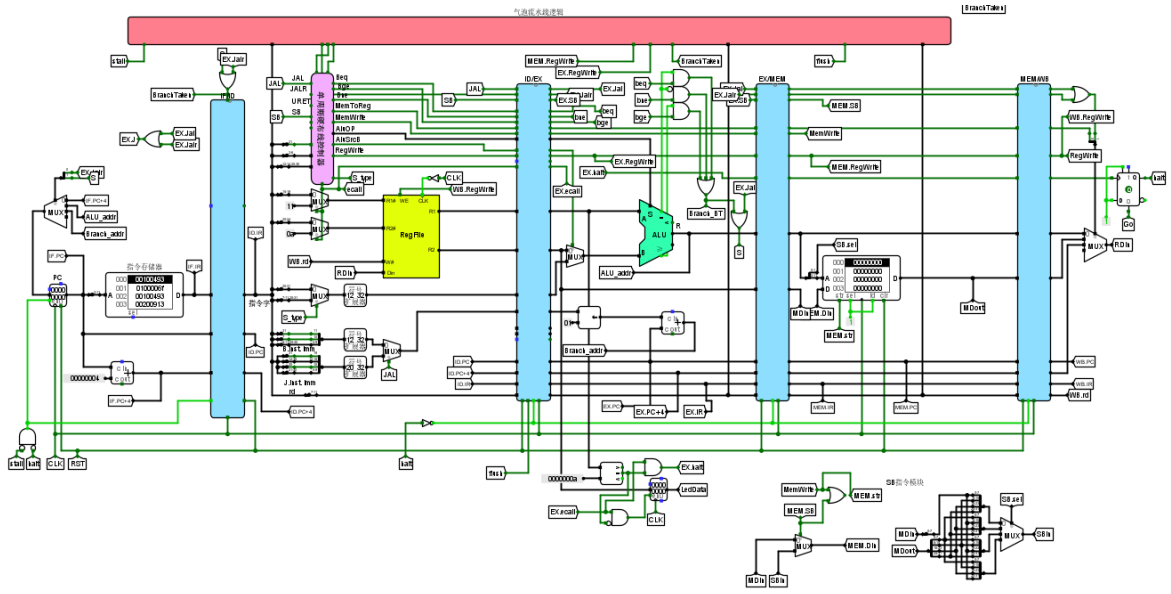

图 3.17 气泡流水线电路实现

3.5 重定向流水线实现

我们重构气泡流水线的结构。在重定向流水线中，我们不需要使用 DataHazzard 信号对气泡进行控制，其 Load-Use 数据相关逻辑为

```
Load-Use = RsUsed & (rs !=0) & EX.RegWrite & (rs==EX.rd)
         + RsUsed & (rt !=0) & EX.RegWrite & (rt==EX.rd)
```

我们只需要对于相应重定向选择信号实现支持，重定向处理逻辑如图 3.18 所示。

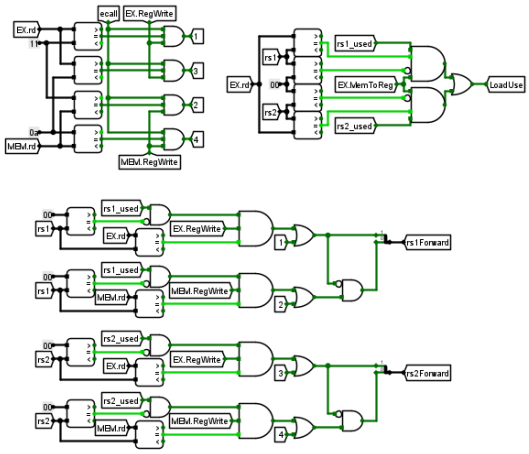

图 3.18 重定向处理逻辑

我们将重定向处理逻辑加入流水线中，依据 flush 信号为 BranchTaken 和 Load_Use 的逻辑与、stall 信号为 Load-Use 的逻辑，我们将气泡流水线进行修改以实现重定向流水线，详见图 3.19。

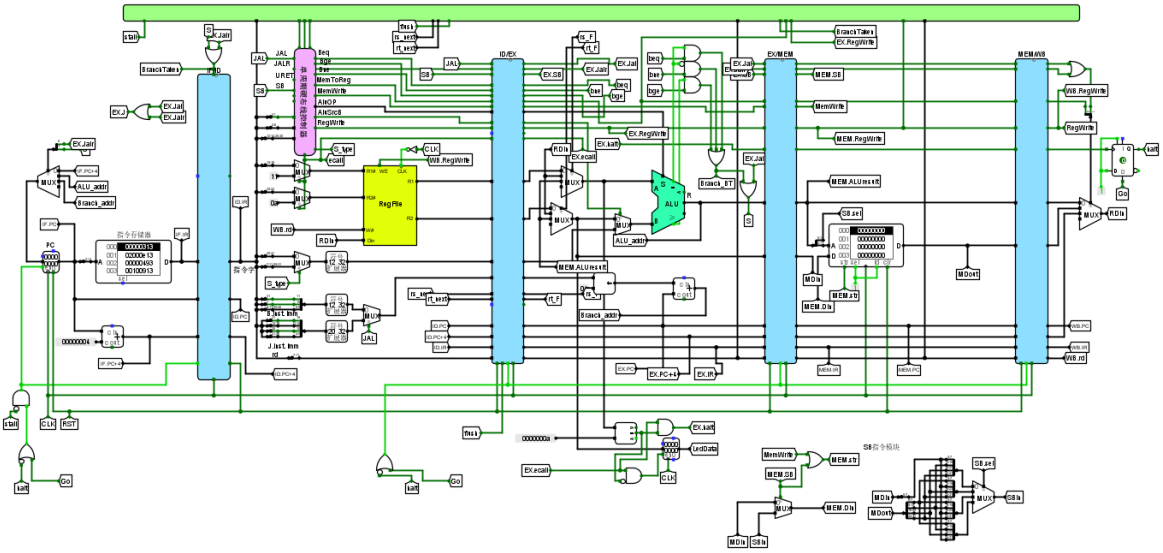

图 3. 19 重定向流水线电路实现

3.6 动态分支预测机制实现

动态分支预测需要在重定向的基础上，增加 BTB 逻辑。BTB 表需要我们实现 8 位全相联 cache，其 cache 槽如图 3.20。

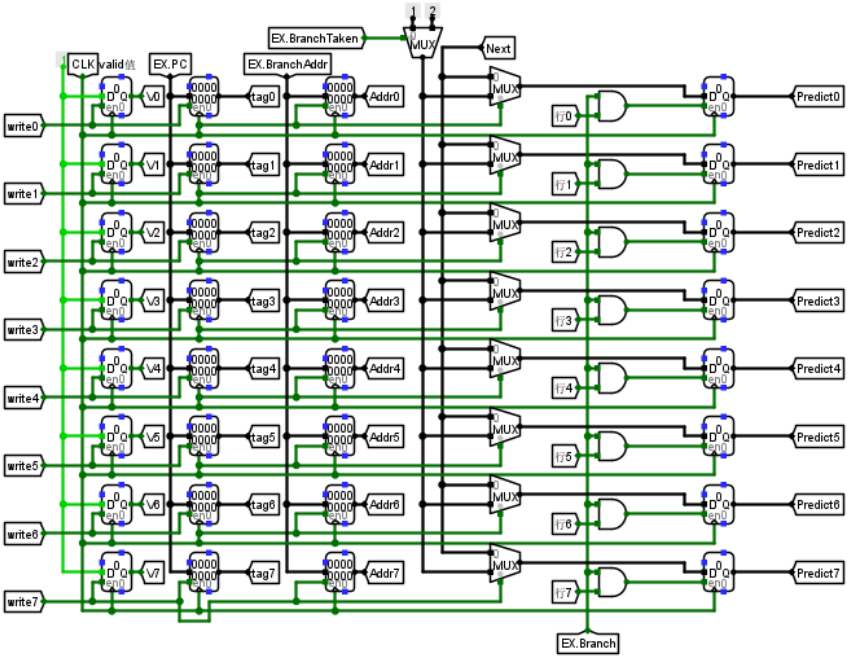

图 3. 20 BTB 表逻辑

在 EX 段执行分支后，全相联进行比较，一旦比较成功，即可根据对应表项的分支预测历史位的值输出跳转信息位，预测位见图 3.21（左）。此后，根据优先覆盖时间最久记录的原则，实现 LRU 硬件电路如图 3.21（右）所示。

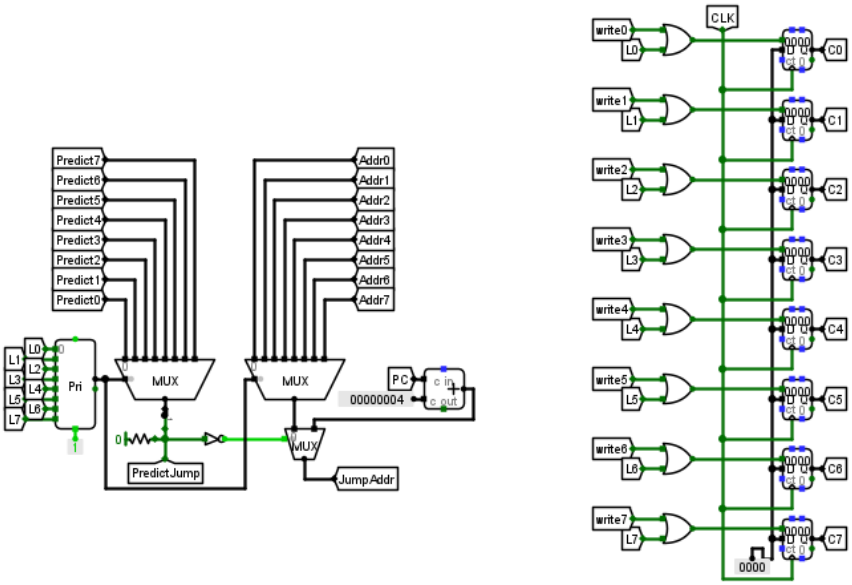

图 3.21 LRU 硬逻辑实现

最后我们将各部分的电路综合起来，即可实现动态分支预测电路。最终电路如图 3.22 所示。

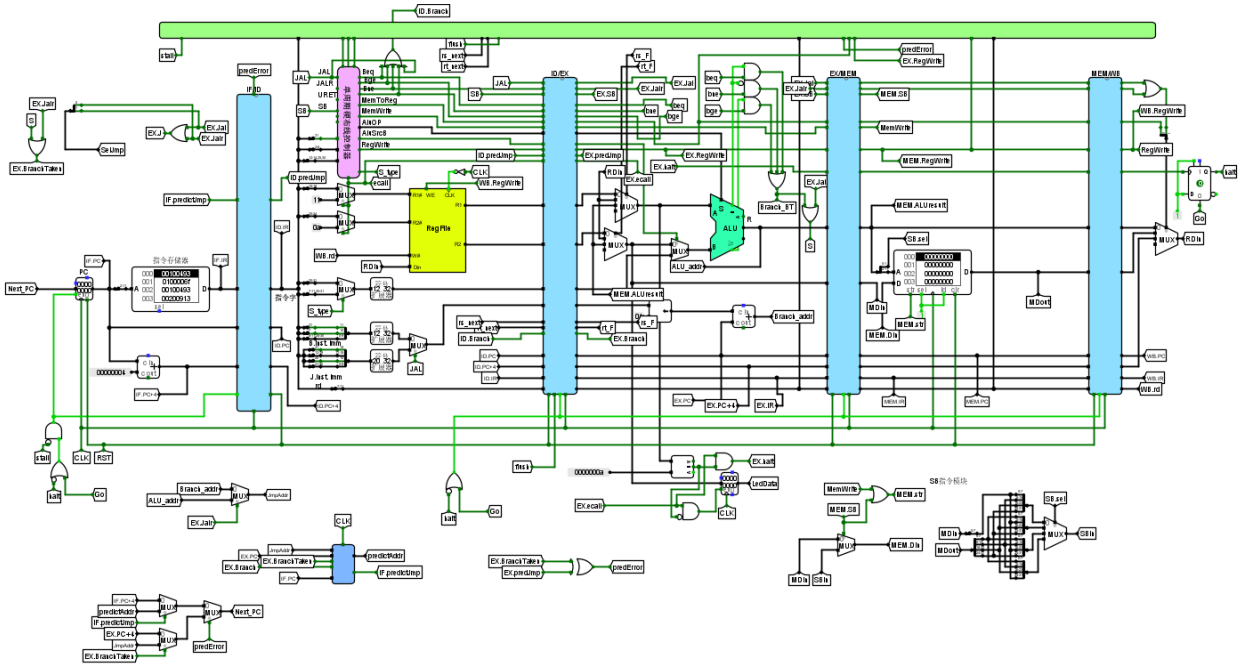

图 3.22 动态分支预测电路实现

4 实验过程与调试

4.1 测试用例和功能测试

在本次实验中，我们实现了单周期 CPU、气泡流水线、重定向流水线、单级与多级中断、动态分支预测流水线等设计与测试，下文将重点对 benchmark 测试程序、CCAB 指令、单/多级中断和动态分支预测进行阐述。

4.1.1 基准测试

我们首先对 risc-v-benchmark_ccab.asm 转换为.hex 文件后运行，对于不同的实现 CPU 运行测试结果如下。经检查，各测试用例的总周期数和分支数均与标准情形相符。

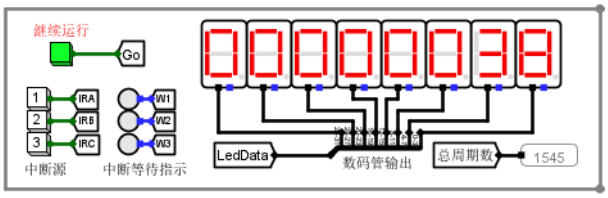

图 4.1 单周期 CPU 运行结果

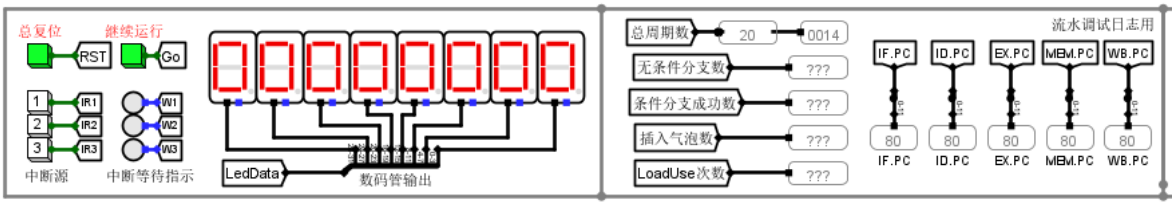

图 4.2 理想流水线 CPU 运行结果

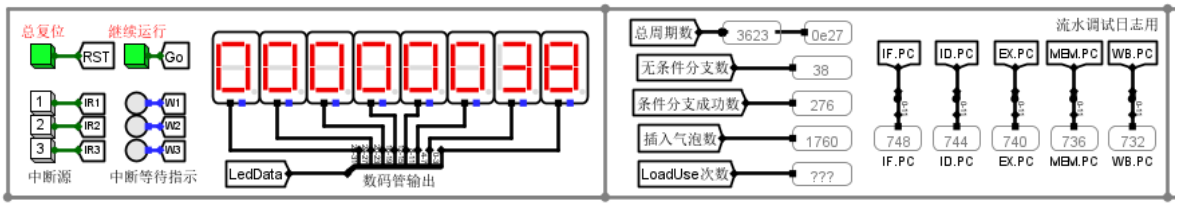

图 4.3 气泡流水线 CPU 运行结果

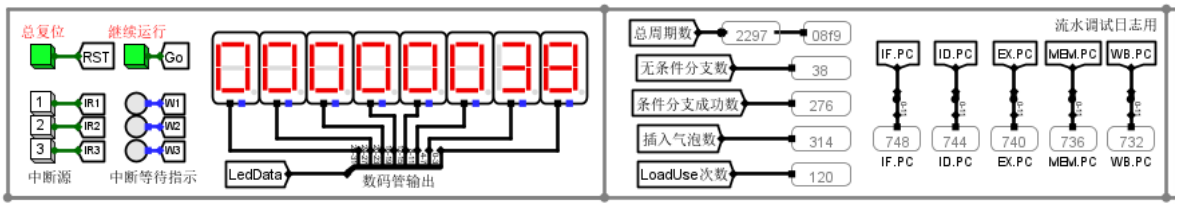

图 4.4 重定向流水线 CPU 运行结果

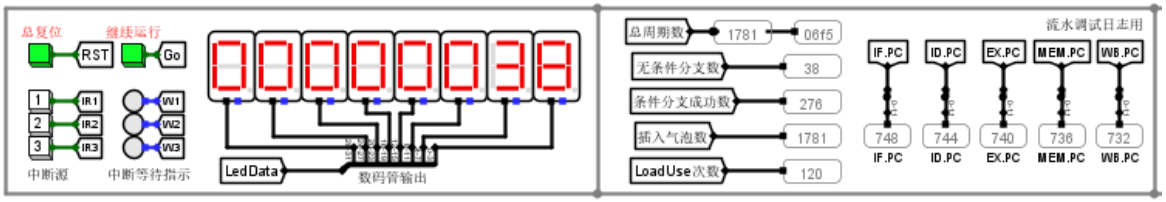

图 4.5 动态分支预测优化后重定向流水线 CPU 运行结果

4.1.2 中断响应测试

对于三种不同类型的中断响应，我们需要分别测试多级中断和流水线中断。对多级中断而言，优先级较低的中断可以被更高优先级的中断打断，并载入高级别中断执行程序，待高级中断响应完成后再进入低级中断未执行的程序。而对于单级中断，只需依次响应不同类型的中断即可。

我们在多级中断中，依次执行 1、3、2 级中断。我们期望程序先进入 1 级中断处理程序，并在 3 级中断到来的时刻进入高优先级响应。3 级中断响应完成后进入 2 级中断响应，最后执行 1 级中断未响应完成的程序。

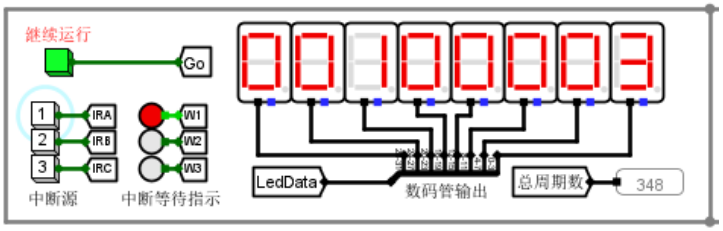

图 4.6 首先响应 1 级中断

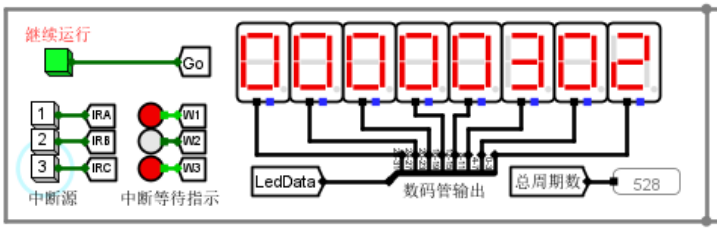

图 4.7 1 级中断被高优先级中断打断

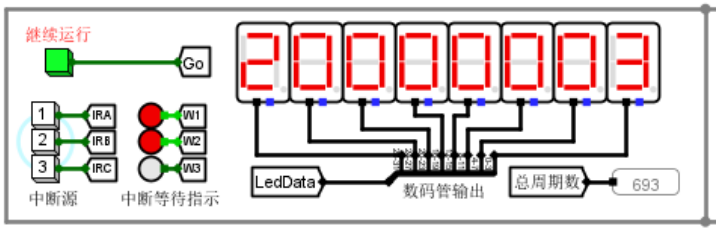

图 4.8 在高优先级中断程序完成后进入低级中断程序

4.1.3 CCAB 指令测试

我们分配到的 CCAB 指令分别为 SLL、SRA、BGE 和 SB 指令。下面我们依次对各指令的测试程序进行运行，运行结果如下。

（1） SLL 指令

启动后，数字“876”右移，和预期相同。

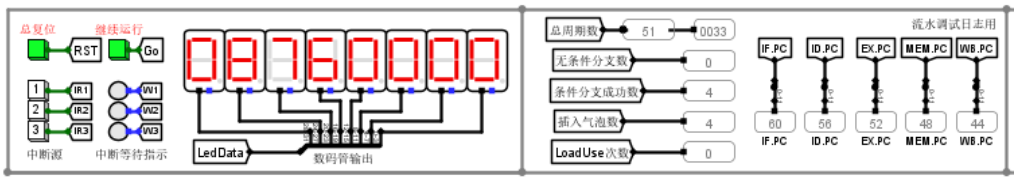

图 4.1 SLL 指令测试（1）

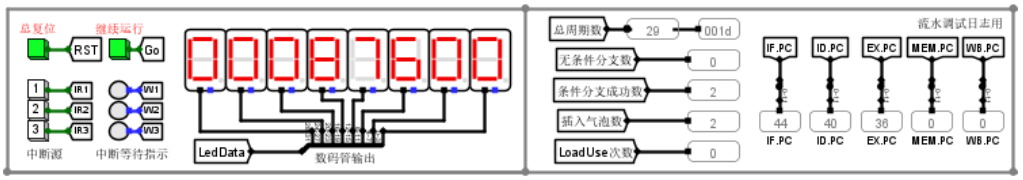

图 4.2 SLL 指令测试（2）

（2） SRA 指令

启动后，数字“876”逐步右移，同时数字左侧填充“F”，符合预期。

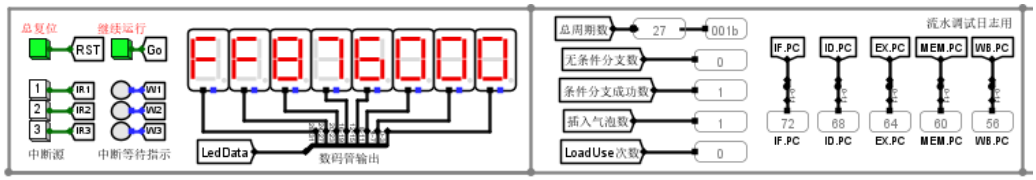

图 4.3 SRA 指令测试（1）

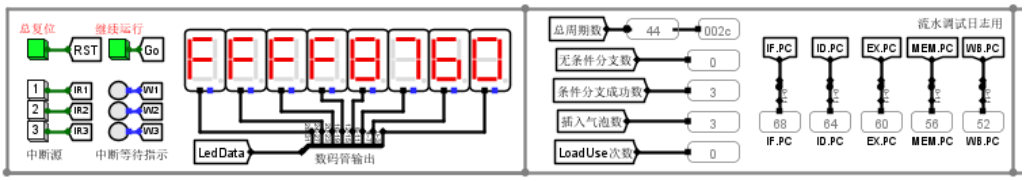

图 4.4 SRA 指令测试（2）

（3） BGE 指令

启动后，屏幕显示数字从“F”变小，符合预期。

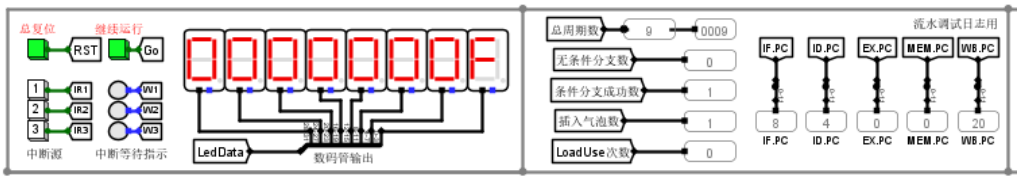

图 4.5 BGE 指令测试（1）

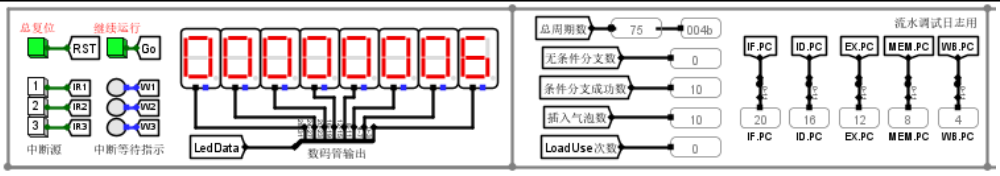

图 4.6 BGE 指令测试（2）

（4） SB 指令

启动后，屏幕先进行倒计时，之后屏幕依次显现四个 16 进制数增长，如下图所示，符合预期。

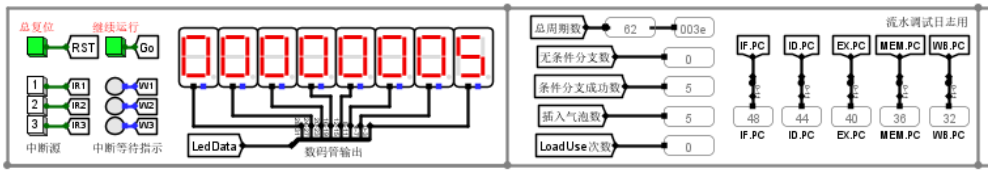

图 4.7 SB 指令测试（1）

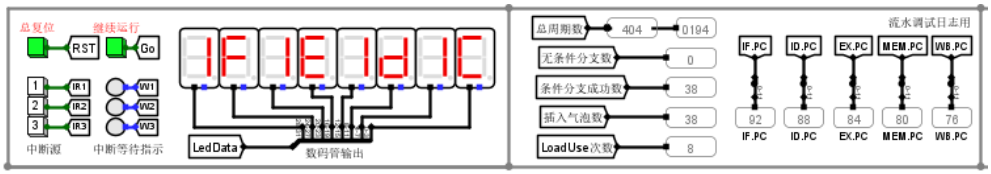

图 4.8 SB 指令测试（2）

4.2 主要故障与调试

4.2.1 流水线锁存故障

气泡流水线： 寄存器堆 RF 未正确写入。

故障现象：对于 benchmark 程序需要使用寄存器部分无法正常显示。

原因分析：通过调试发现，出现错误的指令前后为访存指令 lw。对于流水线 CPU，如果寄存器堆时钟端为上跳沿刷新，当 CLK 跳变时 WB 段写回数据需要进过时间延迟，因此在寄存器中的数据仍然是上一次数据通路的数据。但是当程序以访存指令进行操作时，则会因为在相应的时钟周期内取不出数据，导致内存中写入的数据为零，导致 benchmark 程序出错，相关示意图见图 4.1。

解决方案：对寄存器堆的时钟信号增加一个非门，使得数据能够在接口处顺利传递。

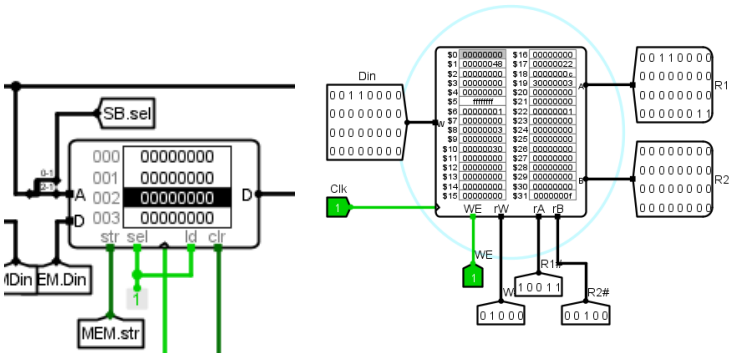

图 4.1 无法向内存中写入数据示意图（左），但寄存器可以写入（右）

4.3 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 相关理论知识，阅读课设任务书，阅读 MIPS 指令手册，并列出 CPU 各部件的数据通路表，并完成数据通路的基本构建。
第二天	完成单周期 CPU 的控制信号表，使用 Logisim 搭建控制器，实现了单周期 CPU 并且通过了测试。完成部分 Logism 单周期 CPU 故障报告。
第三天	完成 Logism 单周期 CPU 的故障报告，并且通过了 Logism 单周期 CPU 的检查。
第四天	继续使用 Logisim 进行实现理想流水线 CPU 的工作，完成了数据通路的改写并调通基础 benchmark 程序。
第五天	完成气泡流水线的电路设计和单级中断的实验。
第六天	完成多级中断的中断寄存队列设计，并成功实现多级中断。
第七天	复习关于指令流水线的知识点，完成重定向流水线部分，并开始以重定向流水线 CPU 为基础尝试流水中断。
第八天	完成流水中断部分，并写完动态分支预测的相关硬件支持。
第九天	动态分支预测可以正常运行，于是开始制作小组团队任务。
第十天	小组团队任务制作中。
第十一天	小组团队任务完成。

5 团队任务

5.1 团队任务描述

本项目基于单周期 RISC-V CPU，以键盘，按钮和 TTY 作为输入输出设备，完成与用户的交互过程。在输入输出过程中加入 input，output 等 ecall 系统调用，将系统调用号寄存在 a7 中并与中断号比较，来模拟当有 IO 操作时的外部中断。

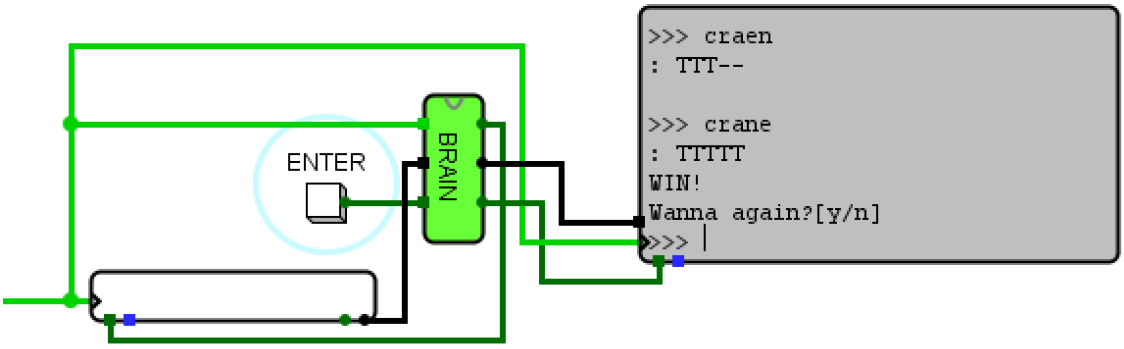

图 5.1 WORDLE 游戏界面

本次实验我们选择模拟 wordle 猜字游戏的过程。游戏规则如下：

玩家随机输入五个字母的单词，来猜测目标答案，每一次输入都会将本次输入单词与目标单词匹配并向玩家反馈提示信息，帮助玩家进一步确定最终的结果。

Normal 模式：该模式下若猜测字母在正确单词中出现，但不在正确位置，则会在该字符在猜测单词中的相应位置给出‘-’提示。若在正确位置，则相应位置给出‘T’提示

Hard 模式：该模式下若猜测字母不在单词正确位置，不会给出‘-’提示

若本次猜测错误，则继续对本轮单词猜测，否则进行下一个单词猜测直到全部单词猜测完成。

对三个中断号的系统调用的说明如下所示：

0x33：当有输出信号中断时，设置输出信号 IS_OUTPUT 至 TTY，同时将 a0 寄存器中的值输出到 NEXT_CHAR 至 TTY 进行打印输出。

0x22：当有字符输入信号中断时，将键盘输入的 INPUT_CHAR 中的值符号扩展 32 位，设置输入信号 IN 为 1，同时将数据保存到数据存储器中。

0x44：当按下 enter 按键时，进入等待输入状态，将 ENTER_HALT 信号至 1，程

序停止运行，等待键盘输入。

5.2 小组分工

本次团队任务中，由我负责汇编程序的编写，同时由洪泽涛同学负责电路实现，高书恒同学负责展示制作，张楠同学负责视频制作。

我制作了猜测单词表，考虑先将单词表存储至内存中，再依次取用单词表中的单词供玩家进行游玩。每次猜测的单词是不一样的，只要玩家选择继续游戏，就会使用单词表中的下一个单词进行猜测。

在玩家猜测单词的时候，我们依次对字符进行匹配，从而输出提示信息。如果玩家猜测正确，如果玩家希望继续，则将单词表指针递增，再跳转回 wordle 程序。

6 设计总结与心得

6.1 课设总结

本次课程设计是关于不同类型的基于 RISC-V 指令架构的 CPU 设计，我作了如下几点工作：

1） 完成了单周期 CPU 的设计工作，同时支持个人分配的 CCAB 指令任务；

2） 完成了对气泡流水线、重定向流水线的设计，可以处理相应数据冲突，同时以后者为基础实现了动态分支预测；

3） 设计了单级中断和多级中断的支持；

4） 完成了小组中对于 Wordle 游戏的开发。

6.2 课设心得

本次课程设计可以说是迄今为止所有实验以及课程设计中难度最大的一门。开学前两周的辛苦调试和十月初的团队任务的加班加点才终于完成了整个课程设计的设计任务。现在再来回顾整个课程设计的整个过程，满满的成就感自是不用说，但是其中也有不少的细节值得我去深思与体会。

课设刚开始的时候我一筹莫展。上个学期我们实现过单周期 CPU 的不同组件的设计，可是在要做如此多种类的 CPU 的时候我有些担心完不成，于是我阅读了任务书和发放给我们的参考文档 ，这才逐渐清晰这次实验要我们具体做些什么，也开始订立计划对实验进行完成。

我们从单周期 CPU 开始。这个任务并不算难，但是写入控制器的指令信号还是遇到了一定的麻烦，例如有一处信号填错导致 CPU 无法正常运行。同时，CCAB 指令的信号设计也有一定的障碍，这个我们采取使用多余的控制信号加以解决。

理想流水线 CPU 的设计并没有什么难度，这个只需要对单周期 CPU 进行调整即可。繁琐之处是设计流水寄存器，每一阶段之间需要传递的指令信号和程序状态字都有差异，这也消耗了一定量的时间。同时需要特别注意的是，寄存器的时钟端为下跳沿。而气泡流水线、重定向流水线则需要更多的控制信号，我花了许多时间对电路进行调整。

单级中断和多级中断需要我们设计中断响应队列，我们采用优先编码器与解码器的组合实现多级中断的中断号选择。这一部分的设计我参考了许多教材上的内容，我也向同学了解设计的方式，从而完成了这一部分的设计。

动态分支预测是这次实验设计中比较困难的部分，在构造 BTB 表的时候我参考了上学期全相联 Cache 的相关实现，历史分支的预测与跳转的实现也不算过于繁琐。但是这种技术的实现让我感觉很有意思，毕竟，通过 LRU 竟然除了命中 Cache 外居然还有新的用途！

总而言之，这次实验课设虽然实现的过程有过一些障碍，但还是感觉收获良多！同时感谢小组的成员在本组团队任务的辛勤付出与帮助，我感到这次课设，是一场很有意思的旅途。

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.

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: