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：	CS2005
学　　号：	U202090063
姓　　名：	董玲晶
电　　话：	13067217235
邮　　件：	1355532189@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 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	
12	ORI	立即数或	
13	NOR	或非	

#	指令助记符	简单功能描述	备注
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，选做
26	MTC0	访问 CP0	中断相关，可简化，选做
27	ERET	中断返回	异常返回，选做
28	SRL	逻辑右移	
29	AUIPC	PC 加立即数	
30	LH	半字加载	
31	BLTU	小于无符号数置数	

2 总体方案设计

2.1 单周期 CPU 设计

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设计。

总体结构图如图 2.1 所示。

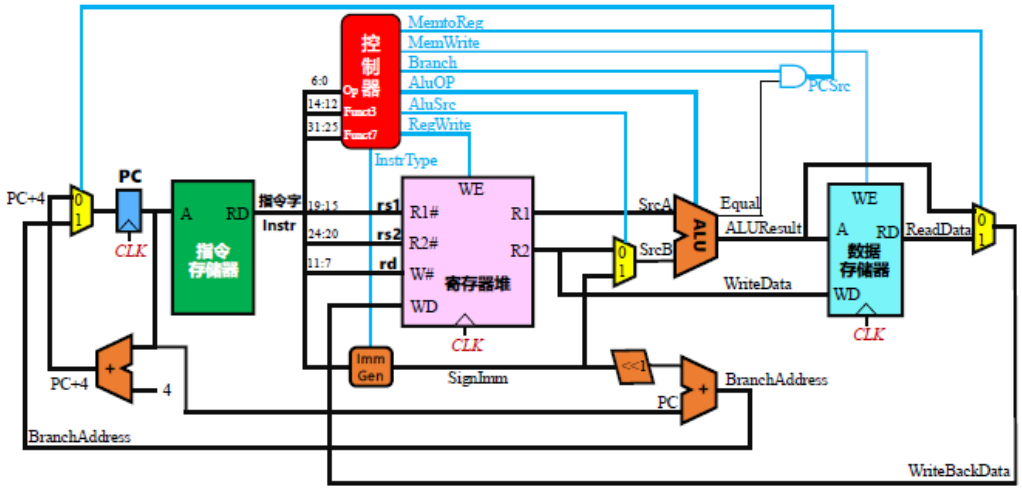


图 2.1 总体结构图

2.1.1 主要功能部件

1. 程序计数器 PC

程序计数器 PC 使用 LOGISIM 自带的数据位宽为 32 的寄存器实现，采用上升沿进行触发；时钟端接统一的同步时钟信号；使能端接`halt`信号取非，只要停止信号不为真就使能；异步清零端全局异步清零信号。

PC 的输入为下一跳指令的地址，当取指令时，其输出作为指令寄存器的输入，同时当前 PC 的值加 4（字节）输入 PC。

2. 运算器

输入为两个 32 位操作数；除了乘除运算分两部分 32 位输出，其余运算输出都只有一个 32 位；其它引脚及功能描述见表 2.1。

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
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
Less	输出	1	Less=(x<y)?1:0, 对所有操作有效
Larger&Equal	输出	1	Larger&Equal=(x>=y)?1:0, 对所有操作有效
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

其中 ALU_OP 为运算器的操作码，对应的功能见表 2.2。

表 2.2 运算器功能码及对应功能

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

ALU OP	十进制	运算功能
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

3. **寄存器堆** RF

使用 LOGISM 的 CS3410 组件库里的 Register File 的组件，采用上升沿触发。时钟端连接统一的同步时钟信号；有一个写入控制信号 WE；输入数据位宽为 32 位；输入 A 和 B 寄存器号码的数据位宽为 5 位，可表示 0 号-31 号寄存器；输出为两个 32 位数据。

2.1.2 数据通路的设计

数据通路分为四个部分，分别是：取指令、PC 程序计数器、立即数数据部分、寄存器堆、运算器、存储器、跳转部分。

表 2.2 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

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 2.3。

表 2.3 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
R1	1	0	未使用寄存器堆 R1 口

控制信号	数据位宽	取值	说明
		1	使用了寄存器堆 R1 口
R2	1	0	未使用寄存器堆 R2 口
		1	使用了寄存器堆 R2 口
IR[21]	1	0	ECALL 指令
		1	URET 指令
OP	5	/	指令 Opcode 字段中的五位，IR[6:2]
Funct	8	/	指令字 Funct3/7 字段，IR[30, 25, 14, 12]
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
S_type	1	0	非 S 型指令
		1	S 型指令
JAL/URET/ECALL/LH/ JALR/AUIPC/BLTU	1	0	当前指令不为 Jal/Uret/Ecall /LH/Jalr/AUIPC/BLTU 型指令
		1	当前指令为 Jal/Uret/Ecall/LH/Jalr/AUIPC/BLTU 型指令
MemToReg	1	0	不写回存储器的数据
		1	写回存储器的数据
Branch	1	0	PC+4 用于顺序执行
		1	J 型指令跳转选择
		2	JALR 指令跳转选择
		3	B 型指令跳转选择
MemWrite	1	0	禁用存储器
		1	存储器有效
AluOp	4	/	Alu 运算操作选择
RegWrite	1	0	寄存器堆禁用
		1	寄存器堆有效

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 2.4 所示。

表 2.4 主控制器控制信号框架

指令译码	R1/2	RegWrite	IR	ALUop	MemWrite	MemToReg	Branch	S_type	AluSrcB

2.2 流水 CPU 设计

2.2.1 总体设计

1. □据通路

将单周期CPU的数据通路分为五个阶段，每个阶段由对应的功能部件完成，这里每个阶段也被称为一个“功能段”。各个功能段串行在一起。逻辑框架如图2.2所示

- 取指令(IF)：负责从指令存储器取出指令；
- 指令译码(ID)：操作控制器对指令字进行译码，同时从寄存器堆取操作数；
- 指令执行(EX)：执行运算操作或计算地址；
- 访存(MEM)：对存储器进行读写操作；
- 写回(WB)：将指令执行结果写回寄存器堆

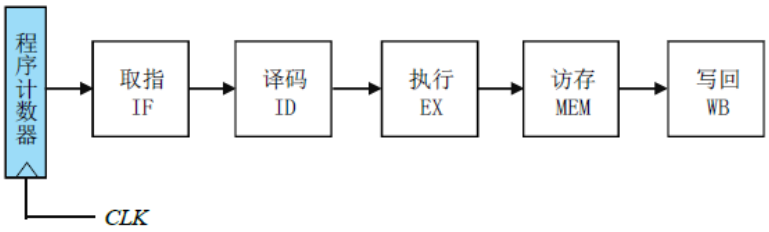


图 2.2 单周期流水线 CPU 逻辑框架

通过在每个阶段的后面增加流水存储器的方法，来锁存本段处理完成的所有数据或结果，以保证本段的执行结果能在下一个时钟周期给下一个阶段使用。程序计数器、流水寄存器均采用统一公共时钟进行同步，每来一个时钟，各段组合逻辑功能部件处理完成的数据将会锁存到段尾的流水寄存器中，作为后段的输入，同时当前段也会接收到前段通过流水寄存器传递过来的新指令或数据，一条指令会依次进入IF、ID、EX、MEM、WB 五个功能段进行处理。指令流水线的逻辑架构如图2.3所示。

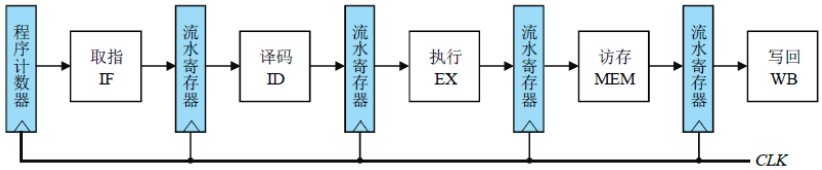


图 2.3 指令流水线逻辑框架

2.2.2 流水接口部件设计

接口部件内部都为“输入-寄存器-输出”的结构。使用寄存器来锁存输入的数据，所有的寄存器时钟信号接统一的时钟控制信号，所有寄存器的使能端也要使用同一个使能信号。

所有的输入的数据都连在一个二路选择器的默认 0 端口，1 端口接地，选择信号使用异步清零信号，实现异步清零。二路选择器选择的结果连入寄存器，寄存器的输出即为此流水接口部件的输出。

统计每个阶段所有需要锁存的值，进行合并，得到下表。

表 2.5.1 xx 阶段流水寄存器锁存值（1）

相关指令译码	RS1/2	AluOp	MemWrite	MemToReg	RegWrite	AluSrcB

表 2.5.2 xx 阶段流水寄存器锁存值（2）

IR	PC	PC+4	R1/2	AluResult	IMM1/2	WriteData	ReadData	RD

2.2.3 理想流水线设计

1. □据通路

在 2.2.1 中我们已经明确了五个阶段，因此，将原单周期 CPU 数据通路中的功能进行细分：其中 IF 段包括程序计数器 PC、指令存储器以及计算下条指令地址逻辑；ID 段包括操作控制器、取操作数逻辑、立即数符号扩展模块；EX 段主要包括算术逻辑运算单元 ALU、分支地址计算模块；MEM 段主要包括数据存储器读写模块；WB

段主要包括寄存器写入控制模块。

在某些情况下，如写入数据时，写入的数据和写入地址分属于不同的指令，若不对数据通路进行改造就会造成错误。因此要将写入地址即是寄存器堆编号的输出位置送入流水寄存器锁存，并在时钟信号的控制下一起进行流水传送，最后由 WB 段的流水寄存器送回。所以每一个流水寄存器都增加位宽存放 WriteReg#数据信息。最终的划分示意图如图 2.4 所示。

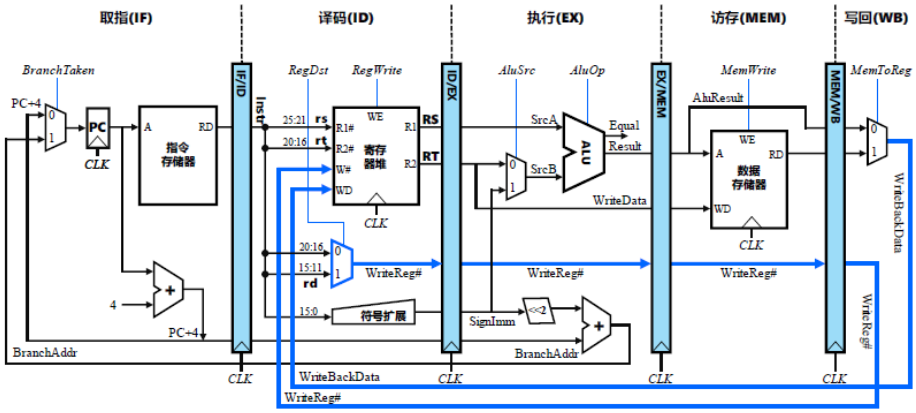


图 2.4 改造后的数据通路细分

2. 控制信号

由于流水线数据通路由单周期数据通路进行改造，与单周期处理器使用相同的操作控制信号，因此流水线可以复用单周期处理器中的操作控制器。

五段指令流水线中ID段负责指令译码生成操作控制信号，所以操作控制器应该设置在译码ID段，操作控制器输入为IF/ID流水寄存器锁存指令字中的opcode和funct字段，内部为组合逻辑电路，输出为7个控制信号，其中MemToReg信号为ID段使用，其他6个后段使用的控制信号输出到ID/EX流水寄存器。RegWrite信号必须传递至WB段后才能反馈到ID段的寄存器。

堆写入控制端 WE，条件分支译码信号 Branch 也需要传递到 EX 段，与 ALU 运算的标志 equal 信号进行逻辑与操作后反馈到 IF 段控制多路选择器进行分支处理。

2.3 气泡式流水线设计

分支指令会引起控制冲突，要解决控制冲突，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。我选择的是在 EX 段执行分支指令，所以只需要清除 IF/ID、ID/EX 流水寄存器即可。

对于数据冲突，可以通过插入气泡消除数据相关的方式进行解决。在流水线中增加硬件逻辑实现 ID 段与 EX、MEM 段指令的数据相关性检测。插入气泡则使用流水清空信号 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；而要暂停 IF、ID 段指令执行，则对寄存器使能端进行控制。 列出相关信号的组合逻辑电路，再分装了一个相关处理逻辑部件即可。

采用插入气泡的方式处理数据相关的处理后，流水线数据通路如图 2.5 所示。

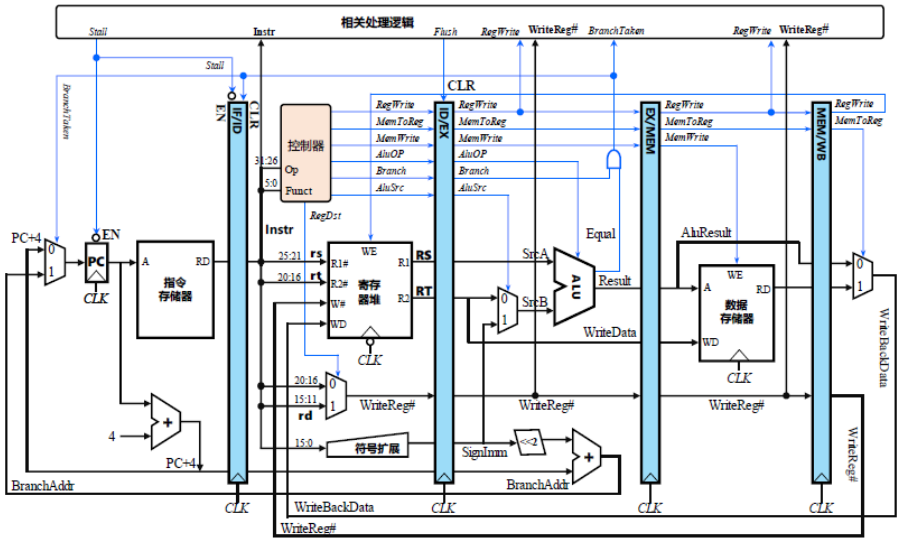


图 2.5　气泡流水线顶层视图

2.4 重定向流水线设计

在取操作数时，先不考虑 ID 段所取的寄存器操作数是否正确，而是等到指令实际使用这些寄存器操作数时再考虑正确性问题。如存在数据相关，EX 段的寄存器操作数 RS、RT 就是错误数据，可以直接将 EX/MEM 流水寄存器中的 AluResult 或 WB 段的 WriteBackData 直接送到 EX 段的 RS 处，作为 SrcA 送 ALU 参与运算，所以接口部件 ID/EX 需要增加两个流水接口。

如果相邻两条指令存在数据相关，且前一条指令是访存指令时，不能采用重定向方式解决数据冲突，必须在发生 Load- Use 的两条相邻指令之间强制插入一个气泡以消除这种相关。

数据重定向的详细通路如图 2.6 所示。

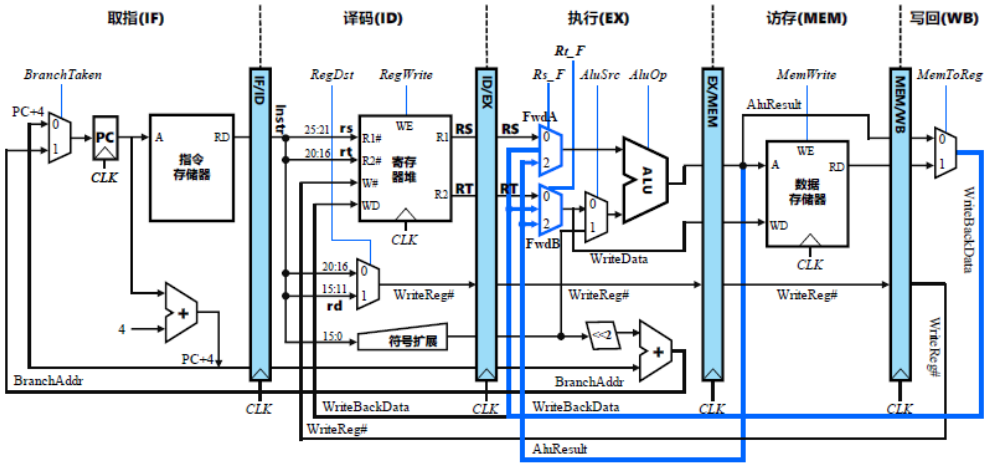


图 2.6 数据重定向数据通路改造

2.5 中断机制设计

2.5.1 硬件设计（单级中断）

1. 实现相关寄存器

中断使能寄存器IE、异常程序计数器mEPC。IE用于开关中断，1表示开中断，0表是关中断，开关中断建议采用同步置位和复位方式。mEPC用于存放中断程序返回地址，在中断响应阶段硬件会自动将主程序PC值送mEPC保存。

2. 实现中断识别逻辑

CPU判断当前有中断响应的依据是连接CPU的中断请求信号IR，当该信号为高电平时，CPU必须进行中断响应。对具体外部设备而言，其中断请求要被CPU响应，必须在该中断请求没有被屏蔽，无更高优先级的中断请求，且中断使能寄存器IE有效，CPU才能收到最终的中断请求信号，当CPU正在执行的指令执行完毕进入公操作阶段时，CPU才会响应该设备的中断请求。

3. 实现中断响应逻辑

设计向量中断机制，可由中断号寻找中断程序入口地址；硬件响应优先级一般采用优先编码器实现，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，优先编码器在Logisim 中

有现成的组件。

4. 开关中断

STI（开中断）、CLI（关中断）指令就是用于控制中断使能寄存器的；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。

5. 增加 URET 数据通路和隐指令数据通路

改造主电路送入PC的值。实现硬件关中断、将主程序断点保存至mEPC寄存器、将中断识别逻辑产生中断服务程序入口地址送PC。同时，将开中断操作集成到中断返回指令URET中一并实现，增加URET指令的数据通路。

2.5.2 硬件设计（多级中断）

可以在单级中断的基础上实现多级中断，由于要实现内嵌中断，低优先级的要被高优先级的打断。所以每当产生一个新的终端你，要将其优先级与之前的进行比较，确定是否要进入新的中断程序。因此要设计硬件栈来保存返回地址，以及在进入中断程序的前后如何保护和恢复现场。在进入中断后关中断，保护现场后用指令MFC0开中断；在恢复现场前用指令MTC0关中断，执行指令ERET时同步开中断。

2.5.3 硬件设计（流水线中断）

流水线中断在重定向流水线的基础上实现，可以参照单级中断，就像把单周期 CPU 改造成重定向流水线一样。

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

以下部件均使用 Logism 实现。

1) 程序计数器（PC）

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，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，使整个电路停机。如图 3.1 所示。

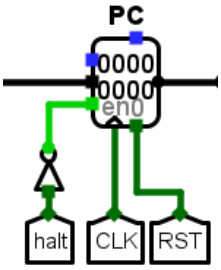


图 3.1 程序计数器（PC）

2) 指令存储器（IM）

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) 数据存储器（MEM）

使用一个随机存储器 RAM 实现数据存储器（MEM）。设置该存储器的地址位宽

为 10 位，数据位宽为 32 位。因为 ALU 的计算结果有 32 位，而 RAM 地址线位宽有限，仅为 10 位，故将 32 位运算结果高地址部分和字节偏移部分直接屏蔽，使用分线器只取 32 位运算结果的 2-11 位作为数据存储器的输入地址。如图 3.3 所示。

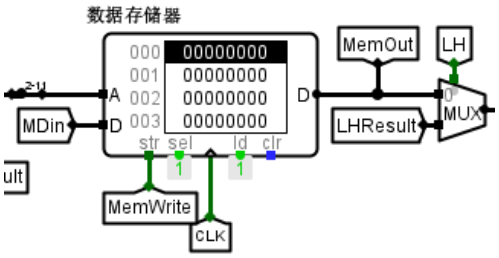


图 3.3 数据存储器（MEM）

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		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
add	PC+4	PC	RS1	RS2	RD	ALU	R1	R2	5			
sub	PC+4	PC	RS1	RS2	RD	ALU	R1	R2	6			
and	PC+4	PC	RS1	RS2	RD	ALU	R1	R2	7			
or	PC+4	PC	RS1	RS2	RD	ALU	R1	R2	8			
slt	PC+4	PC	RS1	RS2	RD	ALU	R1	R2	11			
sltu	PC+4	PC	RS1	RS2	RD	ALU	R1	R2	12			
addi	PC+4	PC	RS1		RD	ALU	R1	IMM	5			
andi	PC+4	PC	RS1		RD	ALU	R1	IMM	7			
ori	PC+4	PC	RS1		RD	ALU	R1	IMM	8			

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
xori	PC+4	PC	RS1		RD	ALU	R1	IMM	9			
slti	PC+4	PC	RS1		RD	ALU	R1	IMM	11			
slli	PC+4	PC	RS1		RD	ALU	R1	IMM	0			
srli	PC+4	PC	RS1		RD	ALU	R1	IMM	2			
srai	PC+4	PC	RS1		RD	ALU	R1	IMM	1			
lw	PC+4	PC	RS1		RD	MEM	R1	IMM		ALU		
sw	PC+4	PC	RS1	RS2			R1	R2		ALU	RF	
ecall	PC+4	PC										
beq	PC+OFFSET	PC	RS1	RS2			R1	R2	6			
bne	PC+OFFSET	PC	RS1	RS2			R1	R2	6			
jal	PC+OFFSET	PC			RD		PC	IMM				
jalr	(x[rs1]+ offset)&~1	PC	RS1		RD		R1	IMM	6			
CSRRSI	PC+4	PC										
CSRRCI	PC+4	PC										
URET	PC+4	PC										
SRL	PC+4	PC	RS1	RS2	RD	ALU	R1	R2	2			
AUIPC	PC+4	PC			RD	ALU	PC	IMM	5			
LH	PC+4	PC	RS1		RD	MEM	R1	IMM		ALU		
BLTU	PC+OFFSET	PC	RS1	RS2			R1	R2	12			

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表格，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，如图 3.4 所示。

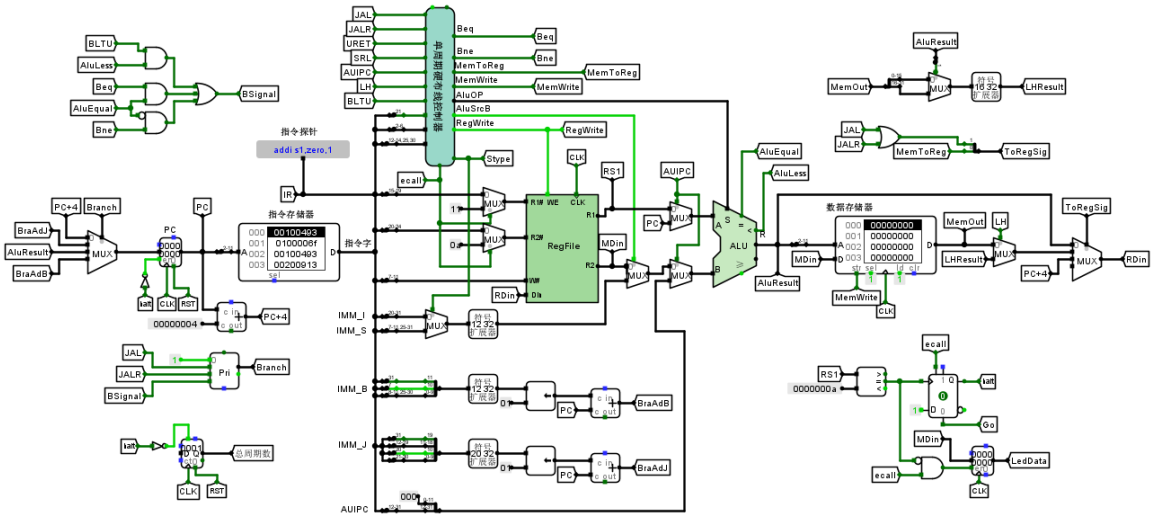


图 3.4 单周期 CPU 数据通路（Logism）

这里展开讲述一下我的差异化 CCAB 指令的设计。

1）SRL 指令可复用所有其他 R 型指令公用的数据通路，因此没有特别的设计。

2）AUIPC 指令实现的功能是：将立即数 [31:12] 左移 12 位并做符号拓展成 32 位加上 PC 的值送入寄存器堆对应的地方。原来立即数的数据来源有两路，分别是 I 型指令所用的立即数和 S 型指令所用的立即数，两类立即数通过一个由 S_type 信号控制的二路选择器进行选择，如图 3.5 所示。

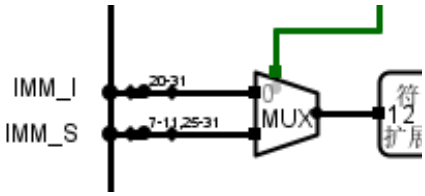


图 3.5 使用 S_type 信号进行选择立即数

此时由于 AUIPC 指令的存在，多了一路立即数，因此再次使用一个二路选择器，选择信号使用 AUIPC 指令的译码信号；

同时，由于运算器的输入 A 也多了一个来源：PC，所以也要使用一个二路选择器，也使用 AUIPC 指令的译码信号来对 RS1 和 PC 进行选择。最终效果如图 3.6 所示。

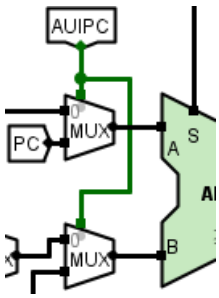


图 3.6 使用 AUIPC 译码信号选择运算器两个操作数

3）LH 指令实现的功能是：从地址 x[rs1]＋sign-extend(offset)中读取两个字节，经符号位扩展后写入 x[rd]。和 LW 类似，只不过 LW 加载字，LH 加载半字，因此部

分数据通路可以进行复用，但是仍需进行一部分改造。

LH 读取的半字可以看作 LW 读取内容的一半，因此将读取到的内容 MemOut 经过分线器分成两个部分，再用地址的第 1 位做片选信号，选择数据。选择后再经符号拓展，就得到结果。如图 3.7 所示。

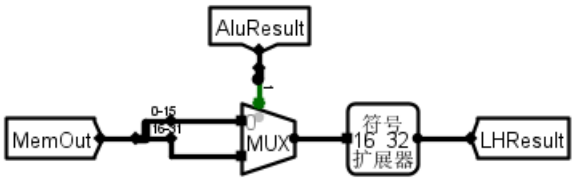


图 3.7 LH 特殊数据通路

4）BLTU 也可以复用 B 型指令的数据通路，因此没有特别的数据通路设计。但在控制信号方面有做特殊处理，详细见下一章节。

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，在 Logism 上进行主控制器、Branch 控制电路、写回数据控制电路的具体实现。

1. Branch 控制电路

1）指令地址选择器

为了实现 J、B、Jalr 型指令，在将下一条指令的地址送入 PC 前，需要通过多路选择器对个各种地址进行选择。多路选择器的信号选择端为 Branch 信号，其位宽为 2 位，因此最多可以选择 4 路。第 0 路默认为 PC+4，顺序执行时下一条指令的地址；第 1 路设置为 J 型指令的跳转地址；第 2 路为 Jalr 型指令的跳转地址；第 3 路为 B 型指令的跳转地址。结构如图 3.5 所示。

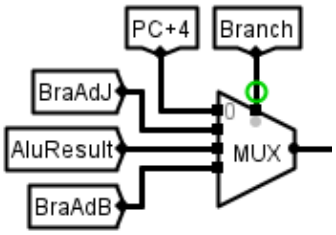


图 3.5 指令地址选择器

2）Branch 信号编码器

由（4）可得指令地址是由 Branch 信号进行选择，所以需要由对应指令的译码信

号进行编码得到 Branch 信号。根据上面的顺序，默认输出 0 号；当 JAL 译码信号为 1 时，说明时 J 型指令，因此 Branch 输出 1；当 JALR 译码信号为 1 时，说明此时为 JALR 型指令，此时 Branch 信号输出 2；当 B 型译码信号为 1 时，说明此时为 B 型指令，此时 Branch 信号输出为 3。如图 3.6 所示。

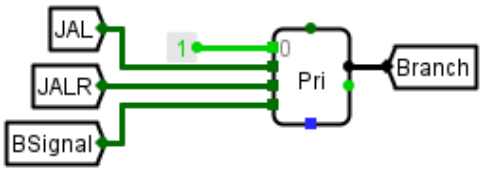


图 3.6 Branch 信号编码器

3）B 指令信号

由于 B 型指令有很多种，如 BEQ、BNE、BLTU（本人差异化 CCAB 指令），需要将各条指令的译码信号与其它相关信号进行操作后得到 B 指令信号。具体如下：

- BEQ：if (rs1 == rs2) pc += sext(offset) 因此将 BEQ 译码信号和 ALU 的 EQUAL 信号进行与操作；
- BNE：if (rs1 ≠rs2) pc += sext(offset) 因此需要将 ALU 的 EQUAL 信号取非后和 BNE 译码信号进行与操作；
- BLTU：if (rs1 <u rs2) pc += sext(offset) 因此需要将 ALU_LESS 信号和 BLTU 译码信号进行与操作；

最后的组合逻辑电路如图 3.7 所示。

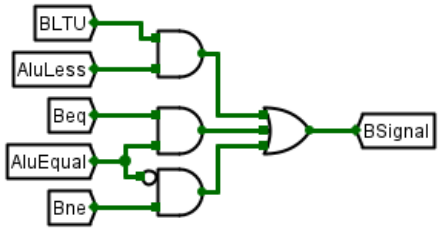


图 3.7 B 指令信号组合逻辑电路

2. 主控制器

根据总体方案设计中控制器及控制信号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所需要的控制信号，完成控制器控制信号真值表。最终情况如表 3.2 所示。

表 3.2 主控制器的控制信号表

指令	指令译码	R1	R2	RegWrite	MemWrite	MemToReg	Branch	S_type	AluSrcB
add		1	1	1			0		
sub		1	1	1			0		
and		1	1	1			0		
or		1	1	1			0		
slt		1	1	1			0		
sltu		1	1	1			0		
addi		1		1			0		1
andi		1		1			0		1
ori		1		1			0		1
xori		1		1			0		1
slti		1		1			0		1
slli		1		1			0		1
srli		1		1			0		1
srai		1		1			0		1
lw		1		1		1	1		1
sw		1	1		1		0	1	1
ecall	ECALL						0		
beq	BEQ	1	1				0		
bne	BNE	1	1				0		
jal	JAL			1			0		
jalr	JALR	1		1			0		1
CSRRSI							0		
CSRRCI							0		
URET							0		
SRL	SRL	1	1	1					

指令	指令译码	R1	R2	RegWrite	MemWrite	MemToReg	Branch	S_type	AluSrcB
AUIPC	AUIPC			1					
LH	LH	1		1		1			1
BLTU	BLTU	1	1						

3. 写回数据控制电路

由于最终写回寄存器堆的数据有多个来源，因此也要设计电路来选择当前需要写回的数据。实现方法如下：

数据有三个来源：ALU 的运算结果、数据存储器读取到的内容、执行 JAL 和 JALR 指令时需要写入寄存器堆的 PC+4。因此选择信号 ToRegSig 数据位宽为 2 位。由 MemToReg 信号填充低半比特，JAL 和 JALR 指令译码信号的或填充高半比特。如图 3.8 所示。

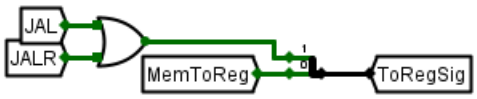


图 3.8 写回数据的多路选择器选择信号

当两者均为 0 时表示非 JAL、JALR 指令，也非从数据存储器写回寄存器堆，那就是返回 ALU 计算结果；JAL 和 JALR 任意为 1，MemToReg 为 0，表示当前指令是 JAL、JALR 任意一条指令，写回 PC+4；当只有 MemToReg 信号为 1 时，将数据存储器读取到的内容写回寄存器堆。最终结构如图 3.9 所示。

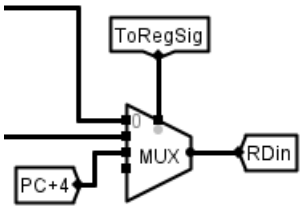


图 3.9 写回数据选择器选择电路的实现

3.2 流水 CPU 实现

3.2.1 流水接口部件实现

1. IF-ID 流水寄存器

由 2.3.1 中的设计图 2.5 可得，在 IF 阶段只是进行了取值，与取指令相关的信号或数据就只有 PC、PC+4 和指令 IR。

IF 取指阶段需要进行锁存的信号如表 2.5 所示。打勾为需要锁存，空则表示不需要锁存。

表 3.3.1 IF 阶段流水寄存器锁存值（1）

相关指令译码	RS1/2	AluOp	MemWrite	MemToReg	RegWrite	AluSrcB

表 3.3.1 IF 阶段流水寄存器锁存值（2）

IR	PC	PC+4	R1/2	AluResult	IMM1/2	WriteData	ReadData	RD
✔	✔	✔						

将表中打勾的需要锁存的值作为流水接口部件的输入，如图 3.10 所示。

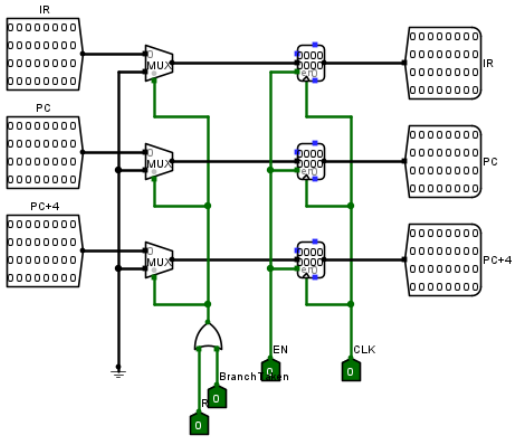


图 3.10 IF-ID 流水接口部件电路图

2. ID-EX 流水寄存器

在 ID 译码阶段，进行译码后得到了寄存器堆两个口的数据（R1 R2）、立即数数据以及改造过的数据通路中的写信号，同时还有硬布线控制器中输出的各种控制信号和输入的译码信号。

因此需要锁存的值如表 2.6 所示。

表 3.4.1 ID 阶段流水寄存器锁存值（1）

相关指令译码	RS1/2	AluOp	MemWrite	MemToReg	RegWrite	AluSrcB
✔	✔	✔	✔	✔	✔	✔

表 3.4.2 ID 阶段流水寄存器锁存值（2）

IR	PC	PC+4	R1/2	AluResult	IMM1/2	WriteData	ReadData	RD
✔	✔	✔	✔		✔			✔

将表中打勾的需要锁存的值作为流水接口部件的输入，部分细节如图 3.11 所示。

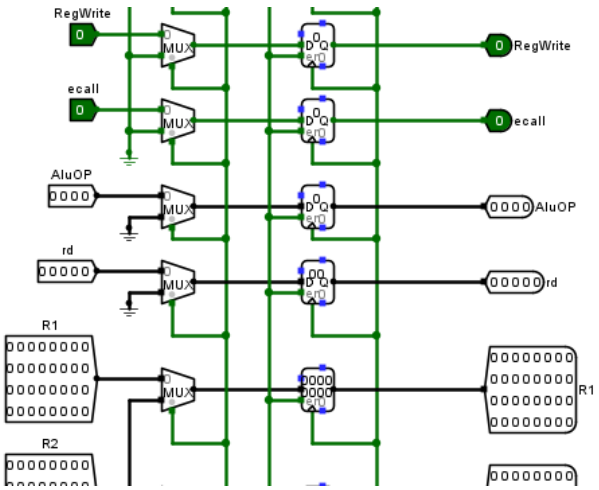


图 3.11 ID-EX 流水接口部件电路图

3. EX-MEM 流水寄存器

EX 阶段需要进行运算并选择数据送入下一个 WB 阶段，因此需要锁存的信号和数据就有 MemWrite、MemToReg、RegWrite、RD 等和读取存储器、写回数据相关的

信号，以及 WriteData 数据、AluResult 数据；同时也要锁存 URET 和 J 型指令译码信号，以及固定锁存的 IR、PC、PC+4。

锁存值如表 2.7 所示。

表 3.5.1 EX 阶段流水寄存器锁存值（1）

相关指令译码	RS1/2	AluOp	MemWrite	MemToReg	RegWrite	AluSrcB
✔			✔	✔	✔	

表 3.5.2 EX 阶段流水寄存器锁存值（2）

IR	PC	PC+4	R1/2	AluResult	IMM1/2	WriteData	ReadData	RD
✔	✔	✔		✔		✔		✔

将表中打勾的需要锁存的值作为流水接口部件的输入，部分细节如图 3.12 所示。

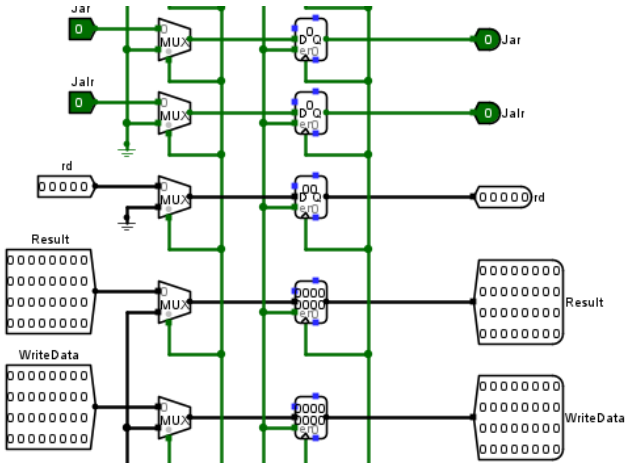


图 3.12 EX-MEM 流水接口部件电路图

4. MEM-WB 流水寄存器

在 MEM 阶段，需要选择是否读取 MEM 数据，将结果写回寄存器堆，因此需要锁存的数据和信号有 RegWrite 信号、MemToReg 信号、ReadData 数据（读取存储器得到的数据）、AluResult 数据（不读取存储器时直接写回的信号）、RD（写回的寄存器编号）；以及固定的 IR、PC、PC+4 数据。

锁存值如表 2.8 所示。

表 3.6.1 MEM 阶段流水寄存器锁存值（1）

相关指令译码	RS1/2	AluOp	MemWrite	MemToReg	RegWrite	AluSrcB
✔				✔	✔	

表 3.6.2 MEM 阶段流水寄存器锁存值（2）

IR	PC	PC+4	R1/2	AluResult	IMM1/2	WriteData	ReadData	RD
✔	✔	✔		✔			✔	✔

将表中打勾的需要锁存的值作为流水接口部件的输入，部分细节如图 3.13 所示。

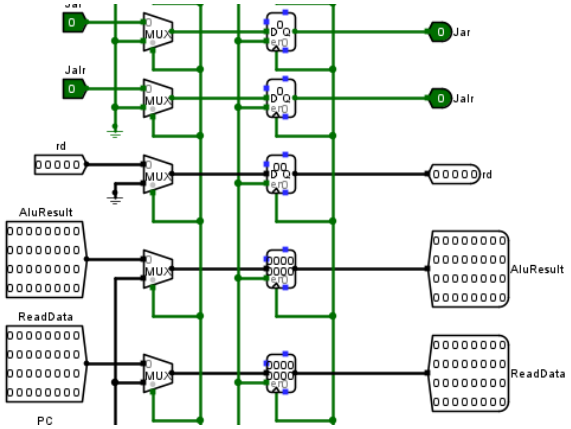


图 3.13 MEM-WB 流水接口部件电路图

3.2.2 理想流水线实现

由 2.2.3 节中对理想流水线的设计，在单周期 CPU 的基础上对数据通路和控制信号相关电路进行更改。将连接好的流水接口部件进行封装，接入主电路。最终数据通路如图 3.14 所示。其中每个阶段的值都来自于上一个阶段的流水接口部件。

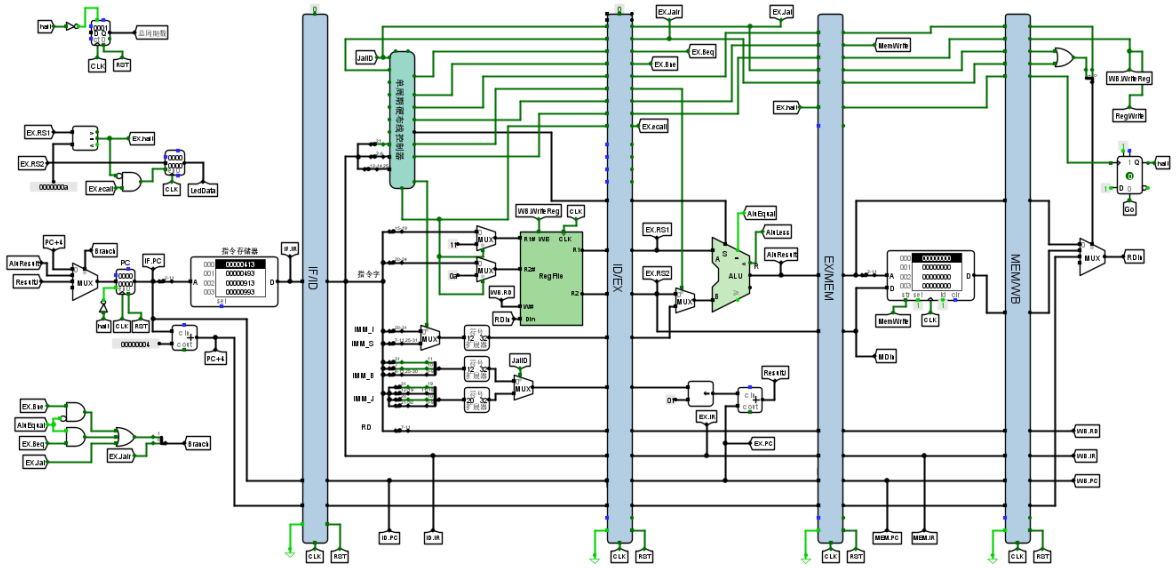


图 3.14 理想流水线数据通路（Logism）

3.3 气泡式流水线实现

1. 数据相关检测逻辑

首先在原有的控制信号中添加两个新的控制信号，R1_Used 和 R2_Used，为 1 有从寄存器堆的 R1#和 R2#读取数据，为 0 则表示没有。如图 3.15 所示。

BLTU	RS1	RS2	XXX
	1	1	
	1	1	
	1	1	
	1	1	
	1	1	
	1	1	
	1		
	1		
	1		
	1		
	1		

图 3.15 控制信号表中新添两个值

由 2.3 节中的设计，要考虑相关性。要想确认 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 WriteReg#是否和源寄存器编号相同即可，因此数据相关检测的逻辑如下：

DataHazzard = R1Used & (rs≠0) & EX.RegWrite & (r1==EX.WriteReg#)
+ R2Used & (rt≠0) & EX.RegWrite & (r2==EX.WriteReg#)

+ R1Used & (rs≠0) & MEM.RegWrite & (r1==MEM.WriteReg#)

+ R2Used & (rt≠0) & MEM.RegWrite & (r2==MEM.WriteReg#)

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

2. 气泡插入和流水阻塞

插入气泡可以参考控制相关中的流水清空信号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即可；而要暂停IF、ID段指令执行，只须保证程序计数器PC的和IF/ID流水寄存器的值不变即可，要做到这一点，只需要控制寄存器使能端即可，当使能端为1时，寄存器正常工作，为0时则忽略时钟输入，寄存器值保持不变。只需要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取反后送对应的使能端即可。

进一步综合控制冲突处理逻辑可知流水线清空信号IF/ID.Flush、ID/EX.Flush，以及流水线阻塞暂停信号Stall的逻辑表达式如下：

Stall = DataHazzard　# 数据相关时要阻塞暂停 IF、ID 段指令的执行

PC.EN = ~Stall　# 程序计数器 PC 使能端输入

IF/ID.EN = ~Stall　# IF/ID 寄存器使能端输入

IF/ID.CLR = BranchTaken　# 出现分支跳转时要清空 IF/ID

ID/EX.CLR = Flush = BranchTaken + DataHazzard　# 出现分支或数据相关时要清空 ID/EX

根据以上所有逻辑表达式，连接电路，电路输入和输出如图3.16所示。电路连接如图3.17所示。

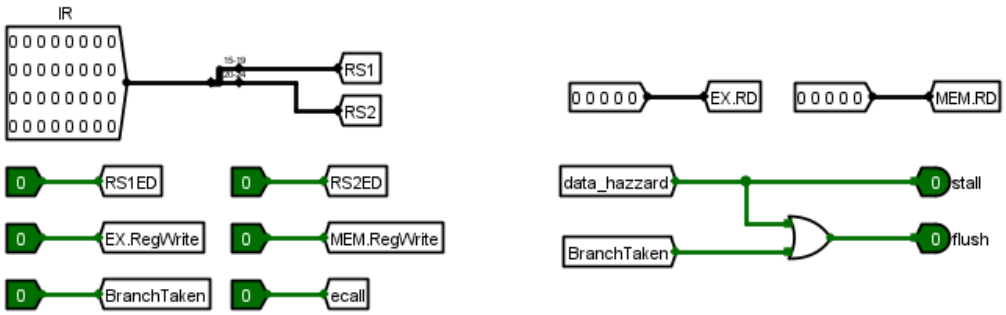


图 3.16 气泡流水线相关处理逻辑电路输入/输出

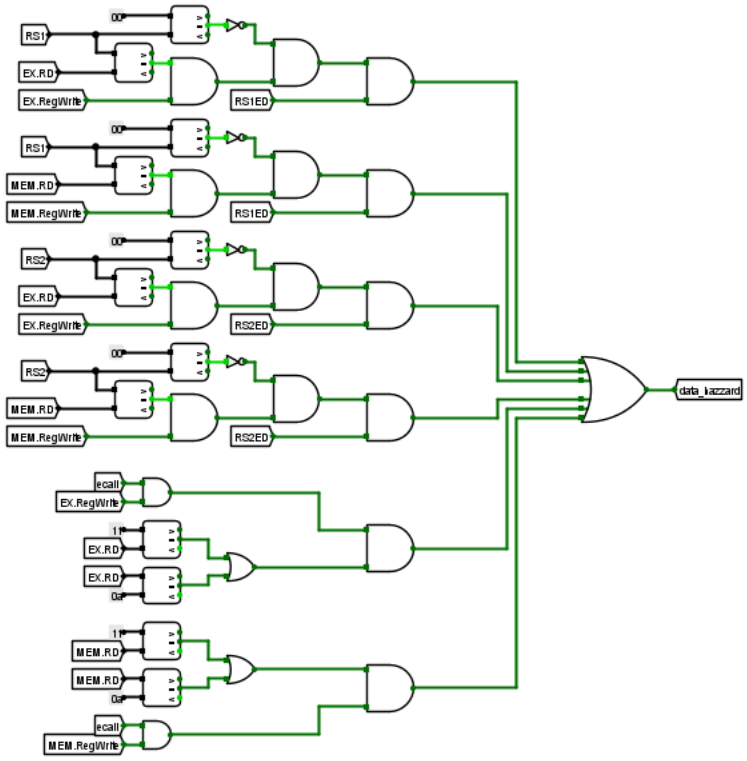


图 3.17 气泡流水线相关处理逻辑电路连接

3. 部件封装与电路连接

在理想流水线的基础上对电路进行调整，并将相关处理逻辑电路封装后接入，最终效果如图3.18所示。

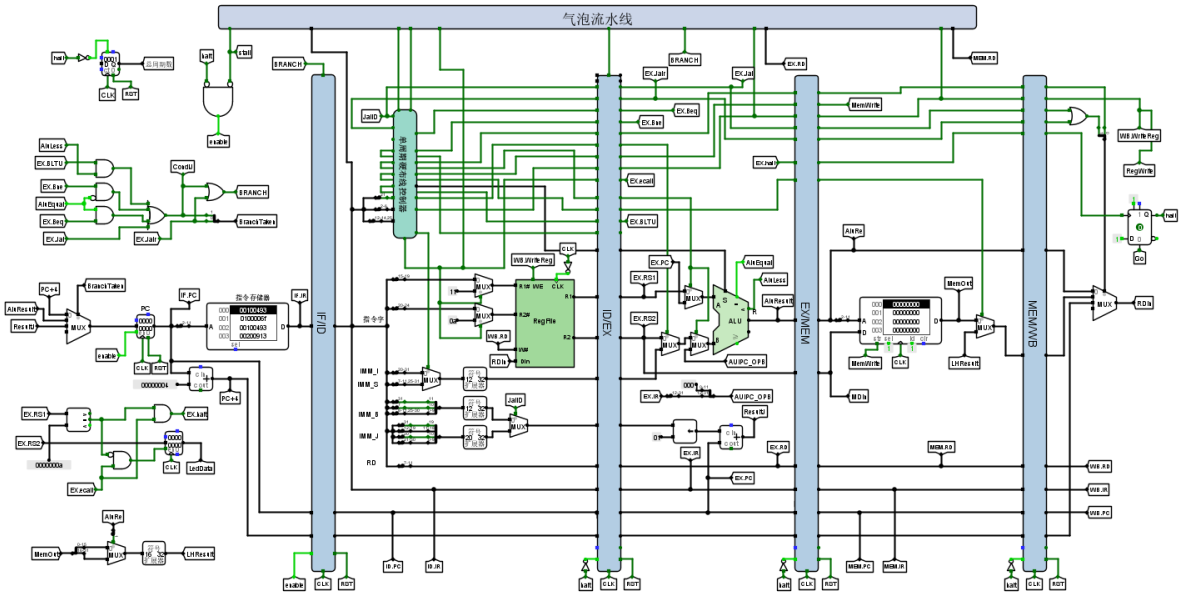


图 3.18 气泡流水线电路

3.4 重定向流水线实现

1. 数据相关检测与处理

采用重定向机制后，流水线中的相关处理逻辑必须进行适当的修订，由于重定向中 Load-Use 数据相关仍然需要通过插入气泡方式进行消除，所以相关处理逻辑应该能检测出 Load-Use 相关，其逻辑表达式如下：

```
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
```

其他数据相关都可以采用重定向方式以无阻塞的方式解决，相关处理逻辑只需要在 ID 段生成两个重定向选择信号 RsFoward、RtFoward 传输给 ID/EX 流水寄存器即可，以 RsFoward 为例，其赋值逻辑如下：

```
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
```

当发生 Load-Used 相关时，需要暂停 IF、ID 段指令执行、并在 EX 段插入气泡，需要控制 PC 使能端 EN、IF/ID 使能端 EN、ID/EX 清零端 CLR，而 EX 段执行分支指令时会清空 ID 段、EX 段中的误取指令，会使用 IF/ID 清零端 CLR、ID/EX 清零端 CLR。综合两部分逻辑，可以得到相关处理逻辑阻塞信号 Stall、清空信号 Flush，各控制端口的逻辑：

```
Stall = LoadUse     # Load-Use 相关时要暂停 IF、ID 段指令执行
IF/ID.CLR = BranchTaken     # 出现分支跳转时要清空 IF/ID
ID/EX.CLR = Flush = BranchTaken + LoadUse     # 分支跳转或 Load-Use 相关时要清空 ID/EX
PC.EN = ~Stall     # 程序计数器 PC 使能端输入
```

2. 电路连接

根据逻辑表达式进行电路连接，输入输出如图 3.19 所示。

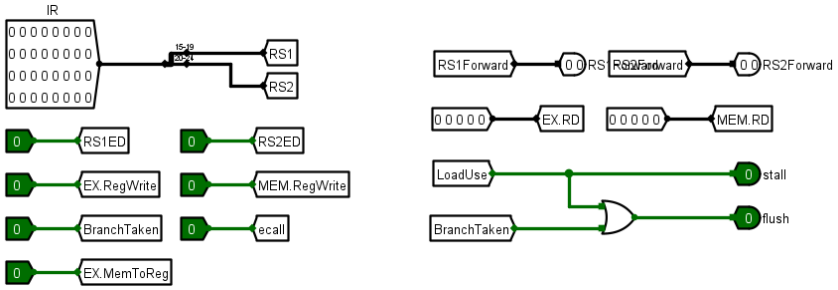


图 3.19 重定向流水线电路输入/输出

处理逻辑电路连接如图 3.20 所示。

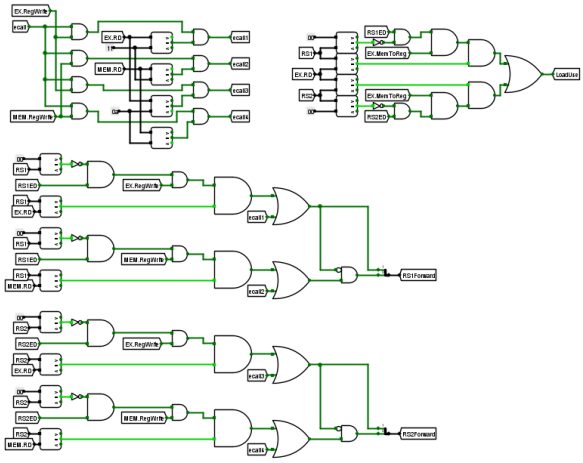


图 3.20 重定向流水线电路相关处理逻辑电路

在理想流水线的基础上对电路进行调整，并将相关处理逻辑电路封装后接入，最终效果如图3.21所示。

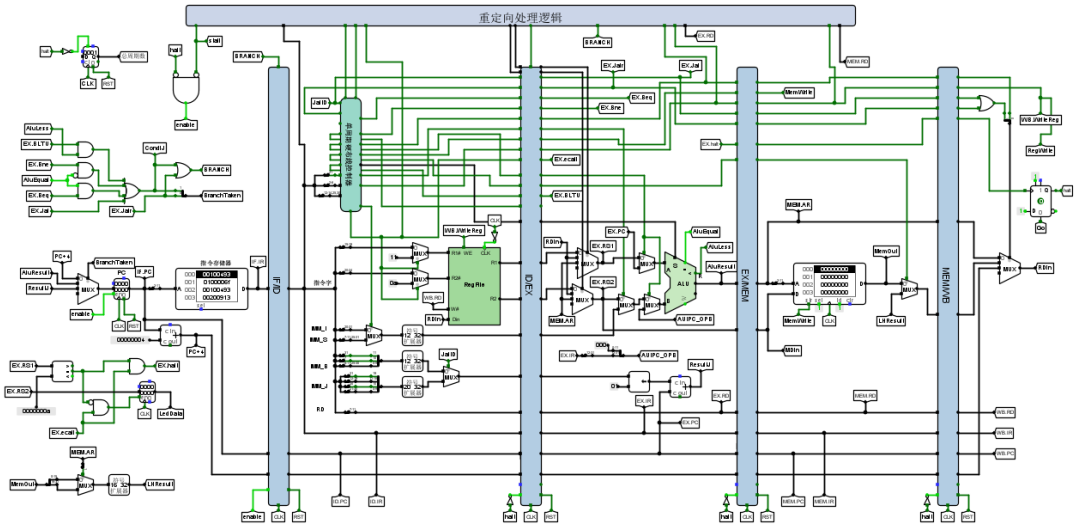


图 3.21 重定向流水线电路

3.5 中断机制实现

3.5.1 单级中断

用寄存器和组合逻辑电路叠加实现 1-3 号中断请求的检测，同时保存所有的中断请求，并在中断返回的时候清空对应的中断信号；用两个触发器实现，中断按键信号接到第一个触发器的时钟端，触发器 1 的输出和复位信号取反相与，作为第二个触发器的输入。再对第二个触发器的结果作编码，得到当前优先级最高的中断，输出得到中断信号、中断号。而中断号作为多路选择器的选择端，寻找到固定的中断入口地址，如图 3.22 所示。

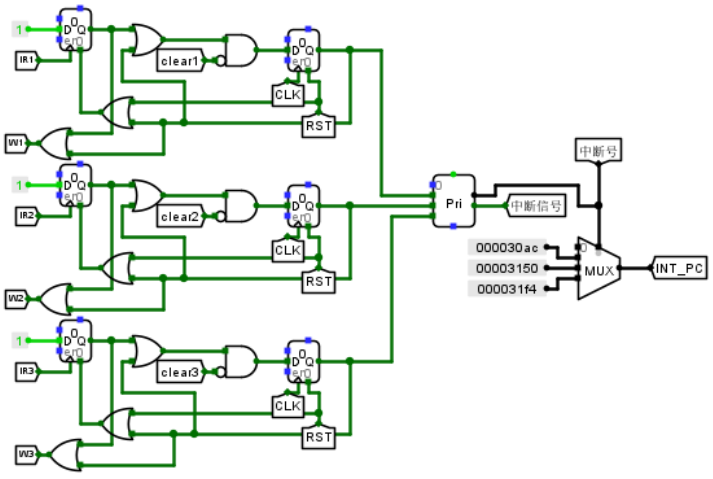


图 3.22 单级中断识别逻辑电路

中断使能寄存器IE、异常程序计数器mEPC都使用寄存器来实现。如图3.23所示。此部分电路负责保存返回地址到RET_PC，并在执行某个中断程序时关中断，中断返回后开中断。

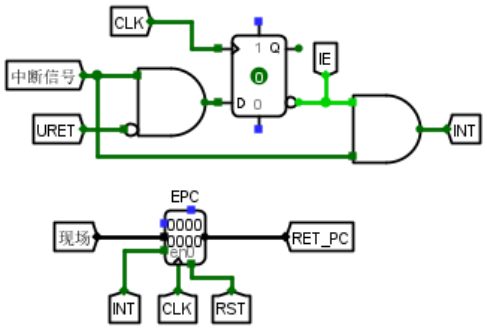


图 3.23 单级中断开关和保存断点

将中断号输入解码器进行解码，再将解码的信号与URET指令译码信号相与，当有URET信号返回时则产生清除信号，清除中断寄存器内容。如图3.24所示。

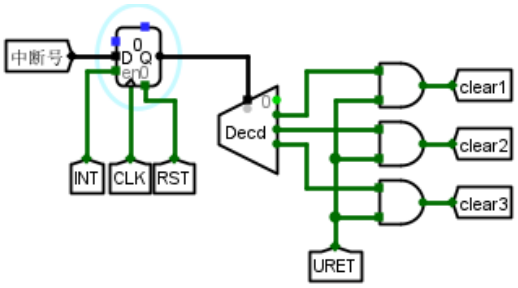


图 3.24 单级中断中断返回和清除

3.5.2 多级中断

中断信号检测逻辑、中断分支逻辑、中断清除和返回逻辑相关电路可以复用单级中断中的电路。但仍有其他部分需要进行修改。修改如下：

增加开中断与关中断（MFC0 与 MTC0）控制逻辑。当 OP 字段都为 0x1c，RS 寄存器字段为 0x06 以及 0x07 时，输出得到 MFC0 与 MTC0 的控制信号。如图 3.25 所示。

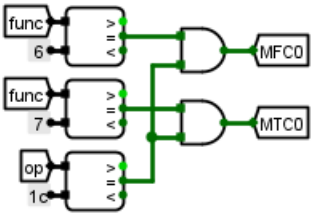


图 3.25 MFC0 信号和 MTC0 信号的生成

由于要支持多级中断，应当设计机制保存多个返回地址。使用三个 EPC 寄存器，让 3 个 EPC 并联，当新的中断产生时且优先级更高，则将新的 PC+4 地址放入 EPC 中，剩余的低优先级中断向后顺延延后执行。如图 3.26 所示。

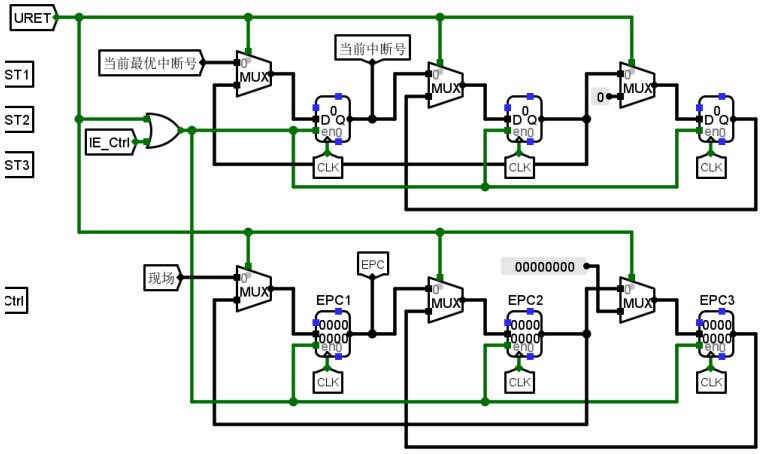


图 3.26 使用 EPC 寄存器实现保存多重返回地址

修改主电路数据通路，当开中断时，送入中断入口地址，否则返回原来的现场；若遇到返回指令则返回原来 EPC 中保存的断点值。如图 3.27 所示。

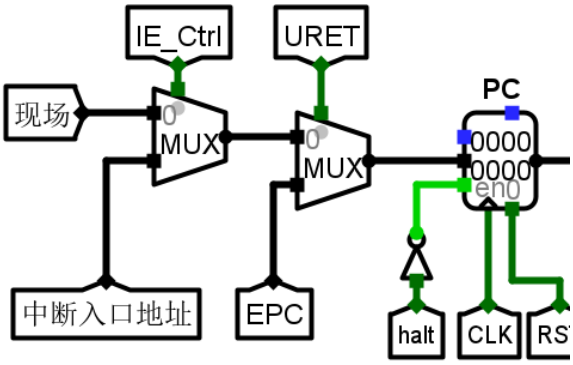


图 3.27 中断隐指令数据通路

3.5.3 流水线中断

中断信号检测、中断信号生成、入口地址寻找电路参考图 3.22。现场保存和使能寄存器、mEPC 参考图 3.23。隐式数据通路参考图 3.27。

4 实验过程与调试

4.1 测试用例和功能测试

单周期 CPU24 条指令以及理想流水线、气泡流水线、重定向流水线已在 EDUODER 平台通关，因此这里只展示各电路中个人 CCAB 差异化指令和各中断电路的测试图。

4.1.1 CCAB 差异化指令

1. SRL 指令

从左往右依次输出：0x87600000 0x08760000 0x00876000 0x00087600 0x00008760 0x00000876 0x00000087 0x00000008 0x00000000。如图 4.1 所示。

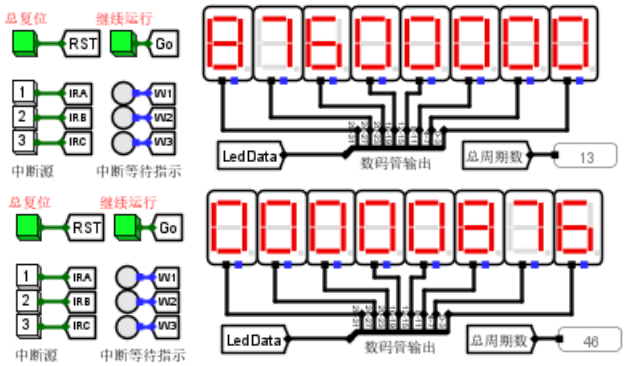
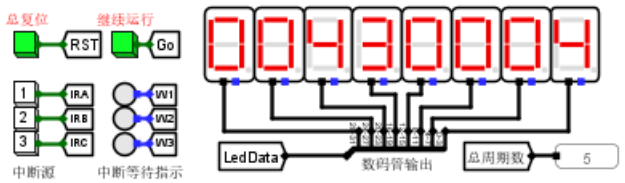


图 4.1 单周期 CPU 中 SRL 指令数码管输出

2. AUIPC 指令

从左往右依次输出：0x00430004 0x00430014 0x00430024 0x00430034 0x00430044 0x00430054 0x00430064 0x00430074。如图 4.2 所示。



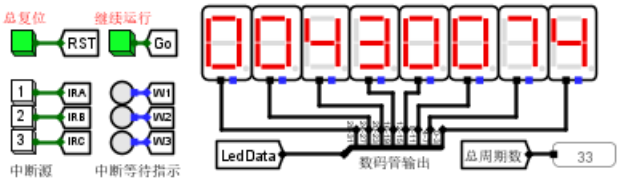


图 4.2 单周期 CPU 中 AUIPC 指令数码管输出

3. LH 指令

从左往右依次输出：0xffff8281 0xffff8483 0xffff8685 0xffff8887 …… 0xffffbebd 0xffffc0bf。如图 4.3 所示。

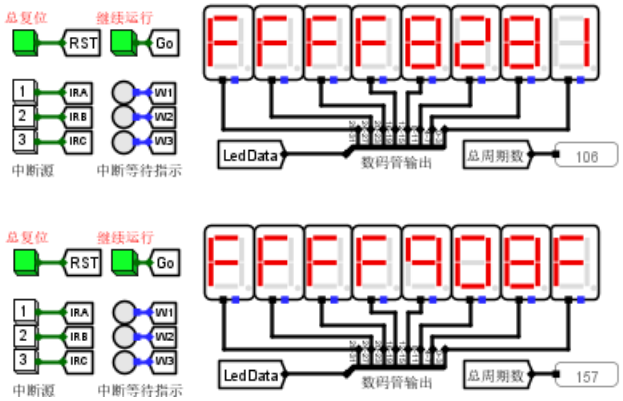


图 4.3 单周期 CPU 中 LH 指令数码管输出

4. BLTU 指令

从左往右依次输出：0xfffffff1 0xfffffff2 0xfffffff3 0xfffffff4 0xfffffff5 0xfffffff6 0xfffffff7 0xfffffff8 0xfffffff9 0xfffffffa 0xfffffffb 0xfffffffc 0xfffffffd 0xfffffffe 0xffffffff。如图 4.4 所示。

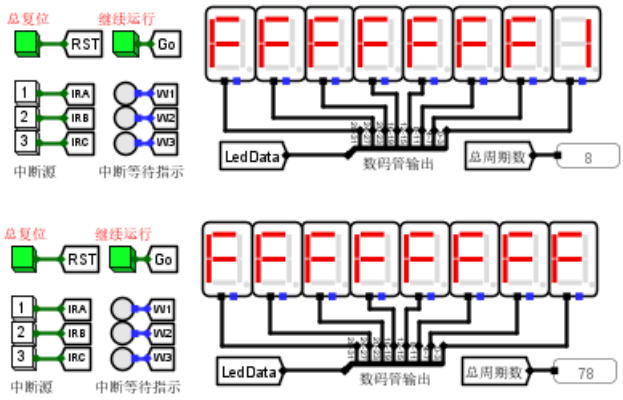


图 4.4 单周期 CPU 中 BLTU 指令数码管输出

4.2 性能分析

单周期 CPU24、气泡流水线、重定向流水线三种方案分别执行“benchmark.hex”文件的时钟周期数如下：

表 4.1 各方案周期数统计表

方案	CPU24+4	气泡流水线	重定向流水线
周期数	1546	3624	2298

气泡流水线周期数是最多，因为数据相关和分支相关的存在插入了大量的气泡。如图4.5所示，插入气泡数目达到了1760次，导致周期数猛增。

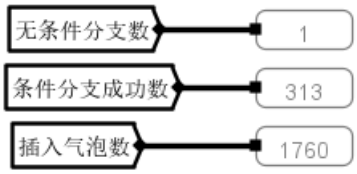


图 4.5 气泡流水线插入气泡数和分支数统计

重定向流水线用旁路技术代替插入气泡，一定程度提高了流水的性能，但是分支相关和Load_Use相关仍然需要插入气泡。如图4.6所示，插入气泡数与气泡流水线相比降低了非常多，这也是重定向与气泡流水线相比周期数低很多的原因。

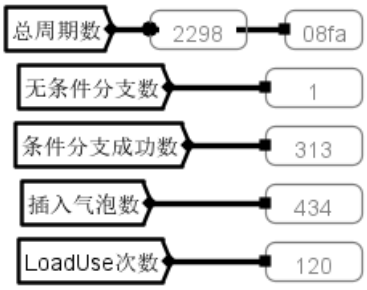


图 4.6 重定向流水线插入气泡数和分支数统计

4.3 主要故障与调试

4.3.1 理想流水线故障（1）

理想流水线： 流水接口部件连接问题。

故障现象：执行指令时控制信号和数据全为 0。

原因分析：线打结短路了全部接低，但是打结部位被隐藏在多路选择器后面了，表面看不出来。

解决方案：逐个对输入信号进行调试，发现了打结点，去掉就行。

4.3.2 理想流水线故障（2）

理想流水线： 流水接口部件连接问题。

故障现象：做完气泡流水线后回头测试理想流水线，原来可以执行的指令变得不能指令，数据又被全清零了。

原因分析：在气泡流水线里对 IF-ID 和 ID-EX 流水接口部件进行了改造，多了两个输入（BranchTaken 和 Flush），如图 4.7 和 4.8 所示。但流水接口部件是共用的。这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。

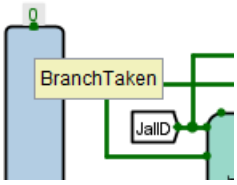


图 4.7 IF-ID 流水接口部件 BranchTaken 输入

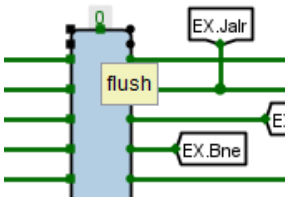


图 4.8 ID-EX 流水接口部件 Flush 输入

4.3.3 流水线 AUIPC 指令

气泡流水线和重定向流水线：AUIPC 指令问题。

故障现象：AUIPC 指令的 LED 输出结果一直是错误的，一直差 8。

原因分析：1. 执行 AUIPC 指令时输入运算器的操作数 A：PC 值，PC 的来源不对。原来错误输出所用的 PC 值是当前正处在 IF 阶段的指令的 PC 值，该 PC 值是其他指令（后两条指令）的 PC 值，不是 AUIPC 指令的 PC 值。**2.** 取的立即数来源也是错误的，也取成了 IF 阶段的 IR 中的立即数。

解决方案：此时 AUIPC 指令正执行到 EX 阶段，所以应该取流水线上 EX 段的 PC，这个 PC 值才是正确的 PC 值。如图 4.9 所示。

同理，也应该从 EX 阶段的 IR 中分离出立即数并做符号扩展，得到输入运算器的另一个操作数 B。如图 4.10 所示。

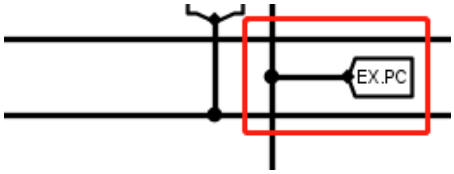


图 4.9 正确的 PC 值来源

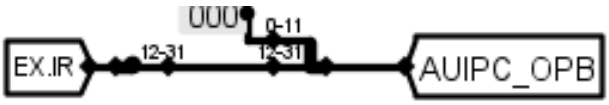


图 4.10 正确的立即数值来源

4.4 实验进度

表 4.2 课程设计进度表

时间	进度
第 1-3 天	完成单周期 CPU 部分。
第 4-5 天	完成理想流水线部分。
第 5-6 天	完成气泡流水线部分。
第 6-8 天	完成重定向流水线部分。
第 10 天	完成单级中断部分。
第 11 天	完成多级中断部分。
第 12-13 天	完成重定向流水线中断部分。

5 设计总结与心得

5.1 课设总结

本次课设中基于 CPU 设计和实现，我做了如下工作：

1） 方案总结：完成了单周期 CPU、理想流水线、气泡流水线、重定向流水线、单级中断、多级中断、流水线中断；

2） 功能总结：单周期 CPU、气泡流水线和重定向流水线支持 24+4 条指令，理想流水线可以支持少部分指令，实现各种中断功能；

3） 学习总结：加深了对 RISC-V 指令集的理解和 CPU 指令执行过程的掌握。

5.2 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。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。

“万事开头难。”我觉得最难的其实是一开始刚接触这个实验的时候，一头雾水，不知道要如何下手。在疯狂翻阅课本和 PDF 参数资料之后才慢慢有了头绪，开始知道要怎么取做。第一个实验（单周期 CPU）我自己感觉由于查资料所以是花了比较多时间。到了后面流水线的部分就简单了起来，因为只需要进行改造就行，尤其是理想流水线和气泡流水线，书上都给了很详细的步骤。最难的部分我觉得还是中断相关的实验内容，但好在在朋友们和网上各种资料的帮助下最后也完成了。

然而对于本次课程设计，我还有一些小小的建议和改进：1. 我认为可以再多配两条 CCAB 指令，做这个课设的过程中我最喜欢做的事情是看各种指令的详解，实现 CCAB 指令也是我很喜欢的一个部分；2. 中断部分尤其是流水线中断给的参考资料不足，只有两周去实现这么多内容，参考资料尤为重要，可以让人节约下来很多搜索资料的时间。

最后在这里也感谢三位老师对于我在本次课程设计中无数问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。我相信组成原理课程设计必将成为我整个大学生涯中一段无比难忘的回忆。

参考文献

[1] DAVID A.PATTERSON(美).

[2] 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.

[3] 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

[4] 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.

[5] 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.

[6] 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.

[7] 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：董玲晶