华中科技大学

2022

计算机组成原理　　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：

学　　号：

姓　　名：[作者]

电　　话：

邮　　件：

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

（1） 支持表 1.1 前 24 条基本 32 位 RISC-V 指令；

（2） 支持表 1.1 后 4 条教师指定的 4 条扩展指令；

（3） 支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（4） 支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

（5） 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。

（6） 能运行教师提供的标准测试程序，并自动统计执行周期数

（7） 能自动统计各类分支指令数目，如不同种类指令的条数、冒险冲突次数、插入气泡数目、load-use 冲突次数、动态分支预测流水线能自动统计预测成功与失败次数。

表 1.1 指令集

#	指令助记符	简单功能描述	备注
1	ADD	加法	指令格式参考 RISC-V32 指令集，最终功能以 RARS 模拟器为准。
2	ADDI	立即数加	
3	AND	与	
4	ANDI	立即数与	
5	SLLI	立即数逻辑左移	
6	SRAI	立即数算数右移	
7	SRLI	立即数逻辑右移	
8	SUB	减	
9	OR	或	
10	ORI	立即数或	
11	XORI	立即数异或	
12	LW	加载字	

#	指令助记符	简单功能描述	备注
13	SW	存字	
14	BEQ	相等跳转	
15	BNE	不相等跳转	
16	SLT	小于置数	
17	SLTI	小于立即数置数	
18	SLTU	小于无符号数置数	
19	JAL	转移并链接	
20	JALR	转移到指定寄存器	If $a7==10 halt(停机指令)
21	ECALL	系统调用	else 数码管显示$a0 值
22	CSRRSI	访问 CSR 寄存器	中断相关，简化为开中断
23	CSRRCI	访问 CSR 寄存器	中断相关，简化为关中断
24	URET	中断返回	异常返回
25	SRL	逻辑右移	运算指令（C）
26	AUIPC	PC 加立即数	
27	LHU	无符号半字加载	存储访问（A）
28	BGEU	无符号大于等于时分支	跳转指令（B）

2 总体方案设计

2.1 单周期 CPU 设计

在单周期 CPU 设计中，采用硬布线控制的方式控制各部件的协调工作，各部件时钟同步。为使取指与访存不产生结构冲突，采取指令存放位置与数据存放位置分开的方案，即实现指令存储器和数据存储器不共用一个存储器的方式避免冲突发生。

总体结构图如图 2.1 所示。可以将其划分为五大结构，取指结构、译码结构、执行结构、访存结构与写回结构。取指结构中，由 PC 寄存器提供指令存储器的取指地址。译码结构中，在单周期硬布线控制器内通过（OP，Funct）字段实现指令字的译码与对应总控制信号的输出，并通过分线器获取寄存器与立即数相关信息，通过寄存器堆得到寄存器数据。执行结构中，由 ALU 运算器实现指令所需计算。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与存放。写回结构中，实现寄存器写回操作。

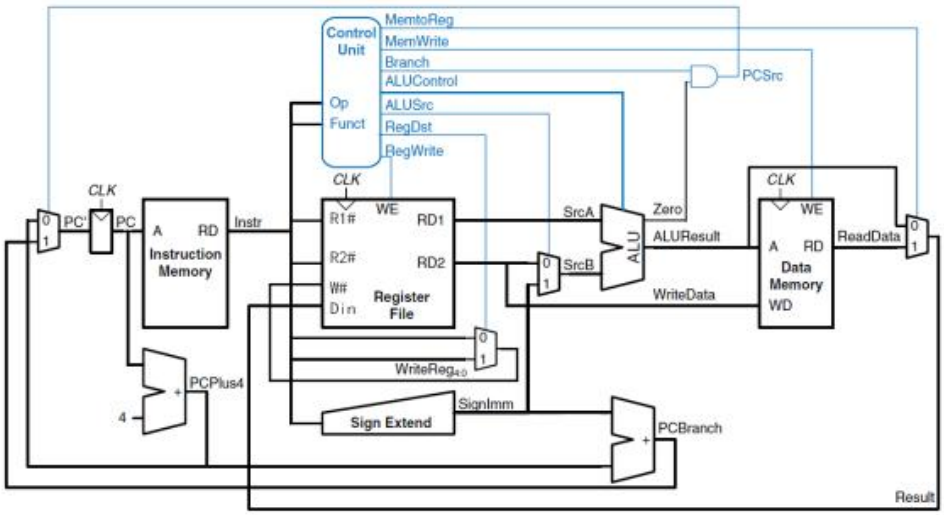


图 2.1 总体结构图

2.1.1 主要功能部件

对于结构图中的各部件部分，主要包含程序计数器、指令存储器、运算器、寄

存器堆、控制器和数据存储器，具体设计思路如下。

1. 程序计数器 PC

程序计数器 PC 用于提供指令存储器的取指地址。由于存储器的数据位宽为 32 位，按字节存储，那么在取下一条指令时，将原值+4 后的数据载入 PC 中。另外，当进行分支跳转时，需将跳转地址载入 PC 中。PC 寄存器设置上升沿触发。

2. 指令存储器 IM

指令存储器存放 RISC-V32 指令程序的二进制数据，在 logisim 中显示为十六进制。选取 PC 上[11:2]位的数据进行存储器的读取，输出为 32 位的指令字。选取 ROM 控件实现 IM。

3. 运算器 ALU

运算器实现指定功能的计算。ALU_OP 控制运算功能。两个输入数 X、Y 进行对应的计算，输出结果 Result 以及有关状态信息。具体引脚与功能如表 2.1 所示。其中＜与≥的比较，根据 ALU_OP 可以分为符号比较与无符号比较。

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
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效
＜	输出	1	X＜Y 时，输出 1，反之输出 0
≥	输出	1	X≥Y 时，输出 1，反之输出 0

4. 寄存器堆 RF

寄存器堆是 32 个通用寄存器集成，通过控制信号与输入数据进行寄存器的读写。其中，输入端 R1#、R2#接收读寄存器号，W#接收写寄存器号。而 WE 与 Din 联合进行写寄存器操作，WE 为写使能，Din 为写入数据。读取的数据通过 R1、R2 端输出。寄存器堆设置上升沿触发。

5. 控制器 CON

控制器采用硬布线控制方式。对指令字的 OP[6:2]和 Funct[30,25,24:12]字段进行译码，通过组合逻辑电路输出对应的控制信号和运算功能号。

6. 数据存储器 DM

数据存储器用于存放或读取数据。通过运算器算出的地址来进行数据的读或写操作，分别受到 MemToReg 和 MemWrite 信号的控制。选取 RAM 控件实现 DM。

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

PC 寄存器存放当前取指令字地址，传给指令存储器的地址输入端。当在顺序执行方式下，对 PC 进行+4 操作；在跳转方式下，传给 PC 跳转地址，通过多路选择器进行不同方式的选择。

2. R 型指令数据通路

R 型指令字通常包含源寄存器号 rs1 与 rs2，以及目的寄存器号 rd。所做操作通常为 R[rd] = R[rs1] op R[rs2]。那么需要通过寄存器堆读寄存器、运算器运算和写回寄存器堆的过程，数据通路中经过这三个主要部件。

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]和源寄存器号 rs1，以及目的寄存器号 rd。操作通常为 R[rd] = R[rs1] op I，特别的，对于 lw 指令操作为 R[rd] = M[R[rs1]+I]。因此，数据通路与 R 型指令类似，只是运算器的两个输入端分别为寄存器数据和扩展立即数，且 lw 指令下的数据通路还包含读数据存储器的通路。

4. S 型指令数据通路

在此次设计中，S 型数指令只包含 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]。数据通路中，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

对于分支指令，可以分为条件分支指令与无条件分支指令。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，即可实现跳转，如 jal 指令则是直接根据指令字中的立即数扩展来跳转的。相应的，数据通路根据不同指令，仅在 ALU 有所区别。

6. U 型指令数据通路

扩展指令中含有 U 型指令 Auipc。该指令将 PC 与指令内立即数相加，存入目的寄存器号 rd 中。故只需要进行取指、计算、写回的过程即可。在计算时可直接使用加法器。

7. Ecall 指令数据通路

对于 ecall 指令，拥有分支功能。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 输入端置 a7 寄存器号 17，rs2 输入端置 a0 寄存器号 10，获取对应寄存器数据。再进行对应的分支判断，进行 LED 显示或各部件置无效使能。

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 2.2 所示。

表 2.2 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）
MemToReg	0/1	寄存器写入数据来自数据存储器
MemWrite	0/1	写数据存储器控制信号
ALU_Src	0/1	运算器 B 输入端选择控制信号
RegWrite	0/1	寄存器写使能
Ecall	0/1	ecall 指令译码信号
S_Type	0/1	S 型指令译码信号
BEQ	0/1	Beq 指令译码信号
BNE	0/1	Bne 指令译码信号
JAL	0/1	Jal 指令译码信号
JALR	0/1	Jalr 指令译码信号

控制信号	取值	说明
BGEU	0/1	Bgeu 指令译码信号
U_Type	0/1	U 型指令译码信号
LHU	0/1	Lhu 指令译码信号

对于各指令进行对应控制信号的输出，依次分析各条指令执行过程中需要的控制信号。将所需的控制信号置 1，无关信号不填写。对于需要使用 ALU 的指令，将 ALU_OP 置为对应的运算功能号。根据控制信号框架表输出每个控制信号的表达式，据此来生成硬布线控制器逻辑线路。

2.2 中断机制设计

2.2.1 总体设计

此次中断设计仅考虑外部中断，3 个外部中断依次将其中断号编写为 1、2、3，优先级依次升高。单级中断和多级中断都包含两个关键过程：中断响应和中断服务。中断处理流程图 2.2 如所示。对于中断响应过程由硬件实现，而中断服务过程由软件实现。

单级中断与多级中断的中断服务过程有较大不同。单级中断机制下，在处理中断服务时，不允许被其他外部中断打断，在此期间需要屏蔽所有其他外部中断，即使优先级更高也同样屏蔽。多级中断机制下，在中断服务处理时，由于通过软件方式进行了开中断，那么在中断服务期间允许其他更高优先级的中断请求打断中断服务，此时需要对中断服务的断点进行保存。同时有多个中断请求待处理时，选取优先级高的中断优先响应。

2.2.2 硬件设计

对于中断处理的硬件部分设计，首先需要对中断信号进行采样，即保存一个突发的中断请求事件，在此次设计中为一个按键的按下。当产生中断请求时，通过采样电路产生持续中断信号，直到该中断返回时消除该信号。除此之外，需要寄存器存储当前中断号，需要 EPC 寄存器存放断点地址，需要 IE 寄存器存放中断使能。对于相关处理设计，单级中断与多级中断有些许不同。

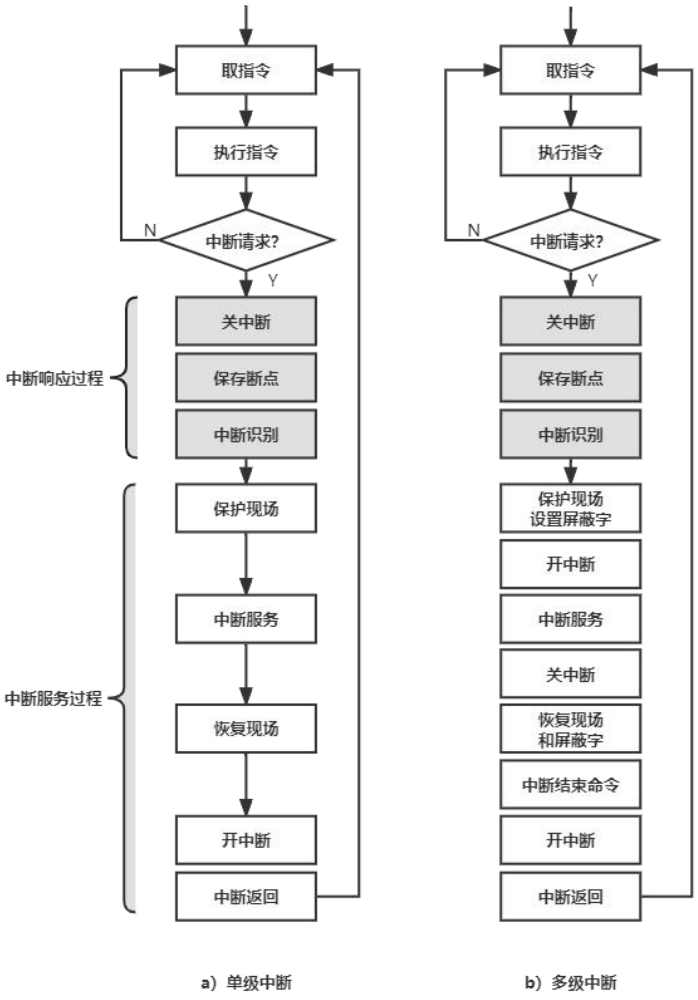


图 2.2 中断处理流程图

单级中断中，对中断信号采用优先编码器传入到中断号寄存器中，中断号寄存器内存放当前中断处理的中断号，使能由 IE 控制，只有当开中断时才进行数据的更新。当使能 IE 为 1 时，表示可接收中断。在接收到中断信息时，将 IE 置关中断使能状态，进行关中断，并将断点保存到 EPC 寄存器中，将对应的中断服务地址传入 PC 寄存器内，由此完成中断响应过程。在中断处理完成，接收到返回指令 uret 时，开中断，置 IE 为开中断使能状态，并将 EPC 内保存断点地址返回给 PC，产生对应的返回信息消除持续中断信号。

多级中断中，由于存在嵌套中断的情况，且每次都要保留断点地址，则对于 EPC 寄存器采用硬件堆栈的形式存放。由于最多嵌套 2 个中断，那么只需要开辟 3 个 EPC 数据区即可，采用寄存器堆的形式实现，使用 3 个寄存器即可。IE 寄存器记录当前 CPU 中断使能状态，可以由硬件开中断、CSRRSI 开中断、CSRRCI 关中断、中断

返回开中断协同控制。此外，还需使用屏蔽字进行中断响应判断。在处理当前中断时，若 IE 位于开中断状态且到来的中断请求优先级高于当前中断，则进行中断响应操作。由于响应级别随中断号依次增加，故采用优先编码器即可完成屏蔽字功能。

2.2.3 软件设计

对于中断处理的软件部分设计，主要是在指令程序中，实现现场保护、现场恢复、CSRRSI 开中断指令和 CSRRCI 关中断指令。在中断响应时，进入中断服务程序，首先需要对现场进行保护，即将原程序使用的寄存器数值进行压栈操作。而在结束中断服务后，需要将栈中的数据返回给对应的寄存器。而在多级中断中，还需要在保护现场后开中断，在 IE 为开中断状态下进行中断服务，期间可以响应其他高优先级中断。在恢复中断前进行关中断，因为在现场保护和恢复现场时不允许打断，否则造成原程序数据错误。在返回中断 uret 指令时，进行 EPC 向 PC 赋值，同步开中断操作。

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。

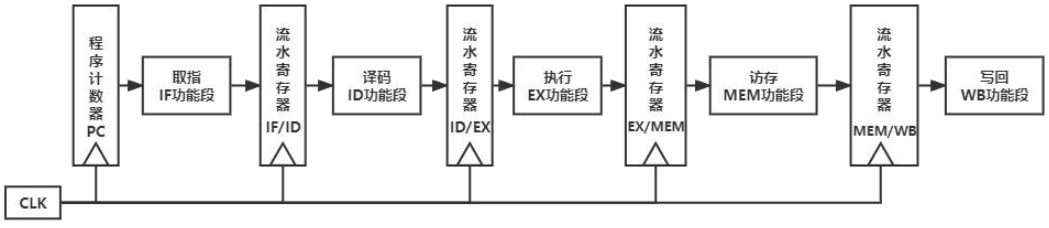


图 2.3 5 段流水 CPU 框架图

上述架构仅仅是在理想情况下的设计，但当存在跳转指令以及相关指令时，会导致流水线冲突。为此，需要在基础架构上进行迭代，以消除冲突。后续将采用插入气泡或重定向的方式处理数据冲突，EX 段分支处理分支冲突，采用哈佛结构处理结构冲突。此外，为了减少分支延迟的损失，考虑采用动态分支预测机制来优化流水线性能。

2.3.2 流水接口部件设计

流水寄存器用于锁存功能端的处理结果和数据，因此对于每个数据应该提供一个对应的输入接口，相应的，下一个功能段通过该数据的输出接口获取数据。数据寄存输入端与输出端是成对存在的。理想流水线的各流水寄存器锁存数据见表 2.3。需要注意的是，控制信号根据不同段所需信号进行传递，各寄存器锁存控制信号不同。

表 2.3 理想流水线的锁存数据说明

流水寄存器	锁存数据说明
IF/ID	PC、IR
ID/EX	PC、IR、各类型指令的立即数、R1、R2、W#、alu_op、控制信号、Halt 信号
EX/MEM	PC、IR、W#、Result、WriteData、控制信号、Halt 信号
MEM/WB	PC、IR、W#、Result、WriteBackData、控制信号、Halt 信号

另外，为满足同步、暂停、清空寄存器的需求，需要提供时钟输入端输入同步时钟信号 CLK，清零输入端输入清空信号 Flush，以及使能输入端输入流水寄存器 Stall 暂停需求。设置流水寄存器触发条件为上升沿触发。

2.3.3 理想流水线设计

理想流水线中没有分支指令，且不产生数据冲突，故采用哈佛结构处理结构冲突即可，将指令存储器存放在 IF 功能段，数据存储器存放在 MEM 功能段。控制信号通过流水寄存器进行段间传输，每个段所需的控制信号只能是传入到该段的控制信号，切勿跨段使用和传输。理想流水线结构框架图如图 2.4 所示。

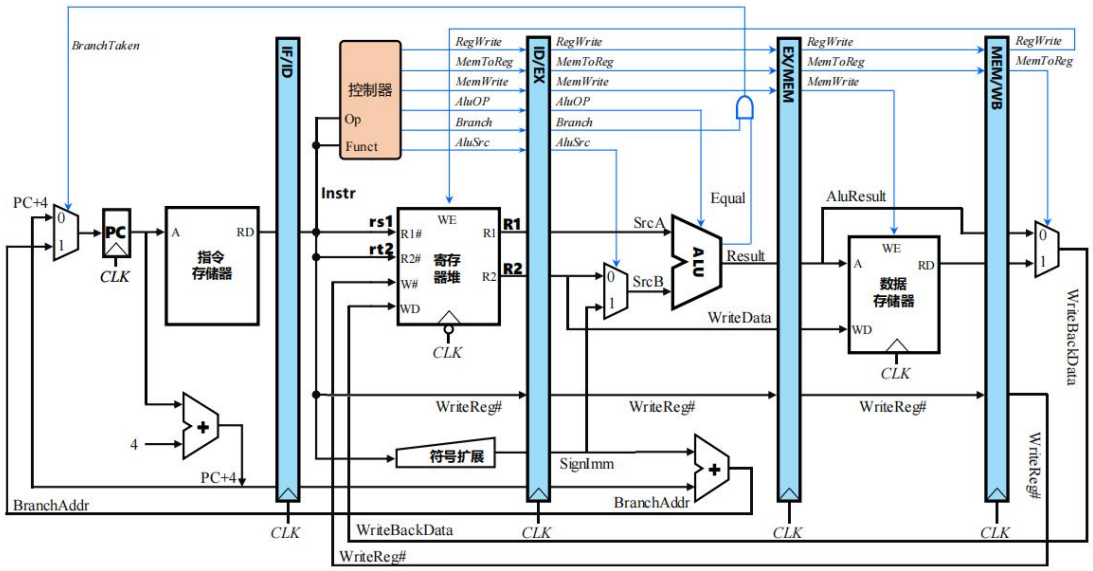


图 2.4 理想流水线结构框架图

在理想流水线 CPU 中，对于 ID 和 WB 段的数据冲突，可以采用先写后读的方式解决。将寄存器堆的触发条件更改为下降沿即可，将其与流水寄存器的触发条件置反。由此可以在一个时钟周期的前半段完成写入操作，后半段完成读操作。

2.4 气泡流水线设计

2.4.1 总体设计

气泡流水线在理想流水线架构的基础上，增加 EX 段分支处理以及数据冲突检测与处理机制。EX 段分支用于处理分支冲突，当在 EX 段检测出需要跳转时，将 EX 段内得到的跳转地址传入到 IF 的 PC 中。此时 IF/ID 和 ID/EX 寄存器内的数据为无效数据，将这两个寄存器同步清零 Flush。

而对于数据冲突，只有在从 ID 段从寄存器堆中取数时才会发生，则在 ID 端进行检测与 EX、MEM、WB 段的数据相关性。ID 与 WB 的数据冲突在上述基础设计中已经考虑，故只需检测与 EX、MEM 的数据相关即可，当产生冲突时，置 IF/ID 的使能端为 0，进行暂停操作 Stall，且清空 ID/EX 寄存器数据，进行气泡的插入。

2.4.2 EX 段分支设计

控制信号与数据经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，Jal 指令则直接在 EX 段进行分支，跳转地址传递到 IF 的 PC 中，而对于 Jalr 指令则需要使用 ALU 计算的结果作为跳转地址，传递到 IF 的 PC 中。

在分支跳转的同时，由于 IF 和 ID 段内的指令为误取指令，故需要将 IF/ID、ID/EX 流水寄存器清空。当 EX 段产生 BranchTaken、Jal、Jalr 信号时，进行清空操作，以消除分支冲突。

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。检测机制需要将 ID 段的源寄存器号与 EX 段、MEM 段的目的寄存器号进行比较。由于源寄存器号包含 R1#与 R2#，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。那么需要对 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。将有效的源寄存器号分别与 EX 段的目的寄存器号 W#和 MEM 段的目的寄存器号 W#进行比较，若 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，表明出现数据冲突。

为消除上述检测出的冲突，采用插入气泡的方式解决。暂停 IF/ID 流水寄存器，将 ID/EX 流水寄存器清空，在 ID 段后增加一个空指令气泡。一般的，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。

2.5 重定向流水线设计

2.5.1 总体设计

在理想流水线的基础上，使用 EX 段分支处理分支冲突，通过重定向方法处理数据相关冲突。EX 段分支处理与气泡流水线处理一致，在此不作赘述。下面进行重定向方法处理数据冲突的设计。

重定向机制首先在 ID 段进行数据相关性的检测，与气泡类似。所不同的是，需要输出更多的信息，而非单纯的数据相关信号。需要将 R1#是否与 EX.W#冲突或 MEM.W#冲突，R2#是否与 EX.W#冲突或 MEM.W#冲突进行输出，由于对于 R1#

或 R2#的相关性检测有 3 种状态，故将相关性信息输出端设置为 2 位。当为 0 时，表明该寄存器无冲突产生；为 1 时，表明该寄存器与 MEM 段的目的寄存器冲突；为 2 时，表明该寄存器与 EX 段的目的寄存器冲突。采用优先编码器，优先处理 EX 段冲突，因为 EX 段距离 ID 段最近，为最新的待写入数据。

将 ID 段获取的相关性状态通过 ID/EX 流水寄存器传入 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。由此提前获得写入寄存器堆的数据，避免了增加气泡导致的性能损耗。

特别的，对于 Load-Use 相关的数据冲突，因为产生冲突的是访存指令，由于需要等待数据存储器的读取数据，会增大 EX 段的关键路径延迟，使流水线频率大大降低。因此，对其进行特殊处理。

2.5.2 Load-Use 相关检测与处理设计

对于 Load-Use 相关的处理，为了避免增大关键路径，选择采用插气泡的方式减小性能损耗。在 ID 段检测出 EX 段存在数据冲突，且 EX 段的指令为访存指令时，表明存在 Load-Use 相关冲突。对于 Load-Use 相关冲突的处理，与气泡流水线相似，对 IF/ID 流水寄存器进行使能暂停，对 ID/EX 流水寄存器进行清空。

通过插入 1 个气泡的方式解决 Load-Use 冲突是局部的，而采用重定向处理处理 Load-Use 冲突则是全局的，影响整体 CPU 频率。因此对于特别的情况，应采取损失更小的方式解决。

2.6 动态分支预测机制设计

在进行重定向处理数据相关后，流水线性能得到很大提升。但对于分支处理，每次当分支跳转成功时都需要插入 2 个气泡，影响性能。因此采用动态分支预测机制，动态预测分支，获得大概率预测成功以使插入气泡的频率减少。

对于条件分支，预测 B 型指令；对于无条件分支，预测 Jal 指令。为了提高预测性能，避免初启动问题，采用分支超强获取跳转地址的机制，在 IF 段就计算出跳转地址。由于较多跳转通过循环产生，故设置初状态为跳转。若在 IF 段未命中，则将地址与状态预先填入 BTB 表。与在 EX 段修改 BTB 表不同的是，改善了相隔较近的跳转指令的预测。若在 EX 表填入前，又接收到跳转指令，大概率为循环跳转，

此时等到 EX 段填入后才能进行预测状态更新获取更准确的状态，期间损失了一定的性能。

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。在 EX 段 BTB 命中的前提下，需要根据双位预测状态图，如图 2.5 所示，对 BTB 中的预测状态位进行更新。

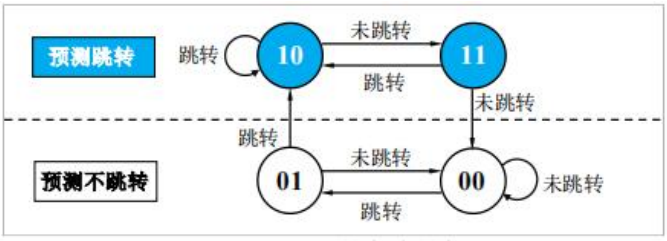


图 2.5 双位预测状态转换图

对于 BTB 表的设计，参考全相联 cache 设计思路，本质上是一个全相联 cache。而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，那么 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。且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。而在 EX 段，则仅需要在 EX 段 Hit 时，进行分支预测位的更新即可。另外，对于淘汰算法采用 LRU 最近最久未使用算法进行 cache 行的淘汰。对每个 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。

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 为暂停信号，取反与 PC 寄存器的使能端相连，为 1 表示电路暂停，即置屏蔽 PC 寄存器时钟信号。RST 信号为清空信号，为 1 时将 PC 寄存器清零。具体实现如图 3.1 所示。

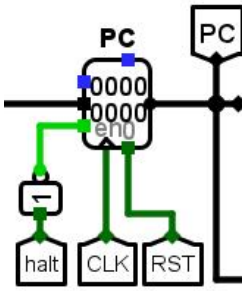


图 3.1 程序计数器（PC）

2) 指令存储器（IM）

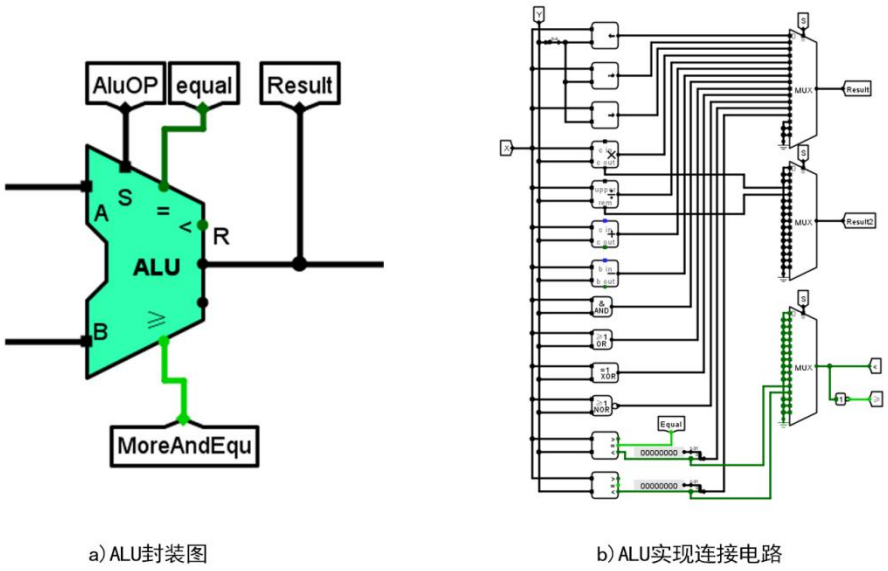
使用一个只读存储器 ROM 实现指令存储器。设置该只读存储器的地址位宽为 10 位，数据位宽为 32 位。因为 PC 中存储的指令地址有 32 位，而 ROM 地址线宽度有限，仅为 10 位，故将 32 位指令地址高位部分和字节偏移部分直接屏蔽，使用分线器取指令地址的[11:2]位作为指令存储器的输入地址。具体实现如图 3.2 所示。

图 3.2 指令存储器（IM）

3) 运算器 ALU

运算器则是多种计算功能的集合。通过输入端 X，Y 输入 32 位运算数据，输入

端 ALUOP 输入 4 位功能号，输出端 Result 输出 32 位运算结果，其他状态输出端输出 1 位状态信号。采取并行运算的方式，对 X 和 Y 进行并行计算，通过多路选择器，输出 ALUOP 指示的输入端口作为运算结果。由于运算位数为 32 位，当存在立即数运算时，应将其扩展为 32 位的值传入 ALU 中。具体实现如图 3.3 所示。



a) ALU封装图　　b) ALU实现连接电路

图 3.3 运算器（ALU）

4) 寄存器堆 RF

寄存器堆为通用寄存器的封装。为 32 个通用寄存器进行编号 0~31，通过 5 位寄存器号来进行查找，每个寄存器可存放 32 位数据。输入端设置 5 位的 R1、R2 寄存器号输入，对应的输出端输出 32 位寄存器数据。1 位使能端 WE 与 5 位寄存号端 W#，32 位输入端 Din，合作进行目的寄存器的数据存放。具体实现如图 3.4 所示。

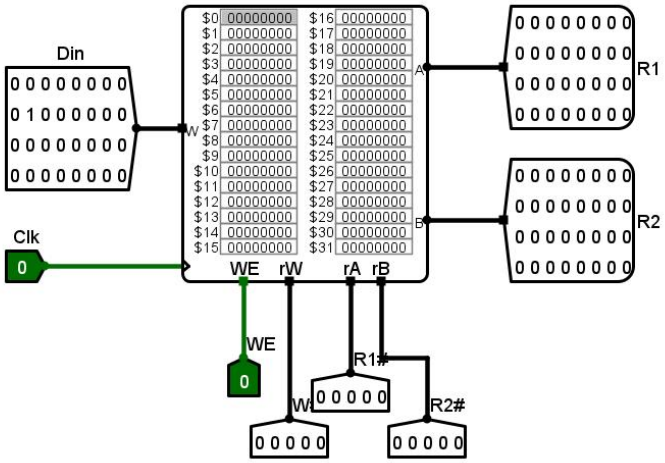


图 3.4 寄存器堆（RF）

5) 数据存储器 DM

使用一个 RAM 实现数据存储器。设置该存储器的地址位宽为 10 位，数据位宽为 32 位。访存地址通过 ALU 计算的地址位数有 32 位，故其地址高位部分和字节偏移部分屏蔽，使用分线器取指令地址的[11:2]位作为数据存储器的输入地址。

另外，数据存储器包含读与写两个功能实现，对于读数据则需要将 MemToReg 信号置于加载端，而写数据则不仅需要将 MemWrite 信号置于写使能端，还需要将数据置于数据写入端。由于存在半字读取指令 lhu，还需要对读出的数据进行高 16 位和低 16 位的拆分，将地址字段的第 1 位作为选择位。具体实现如所示。

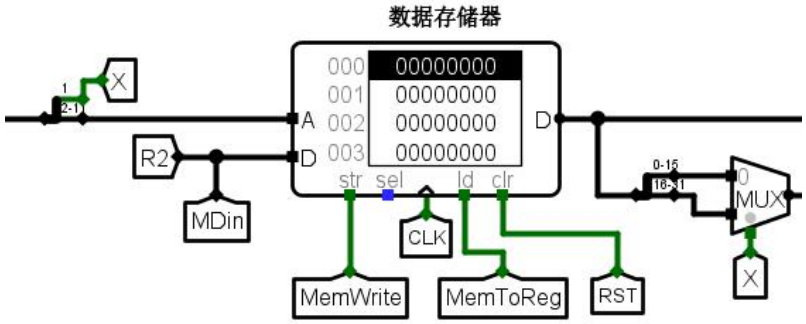


图 3.5 数据存储器（DM）

3.1.2 数据通路的实现

根据单周期 CPU 的设计思路，将各主要功能部件构建好后，具体通过 5 个功能，取指、译码、执行、访存与写回，依次对部件进行连接，从而构造相应功能的基础数据通路。最后，再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 3.6 所示。

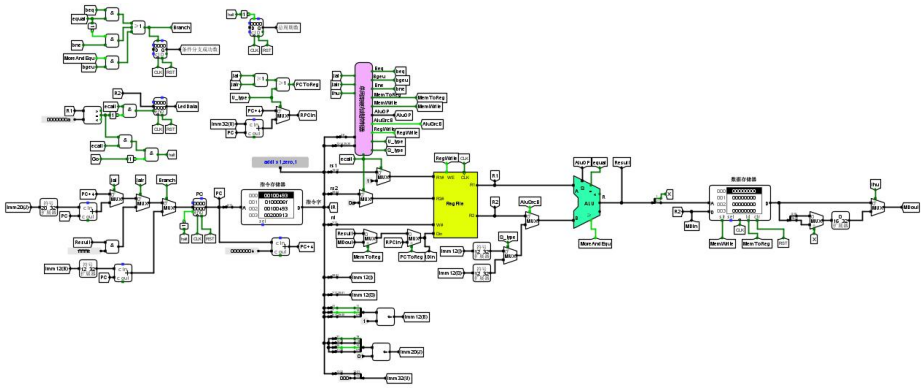


图 3.6 单周期 CPU 数据通路

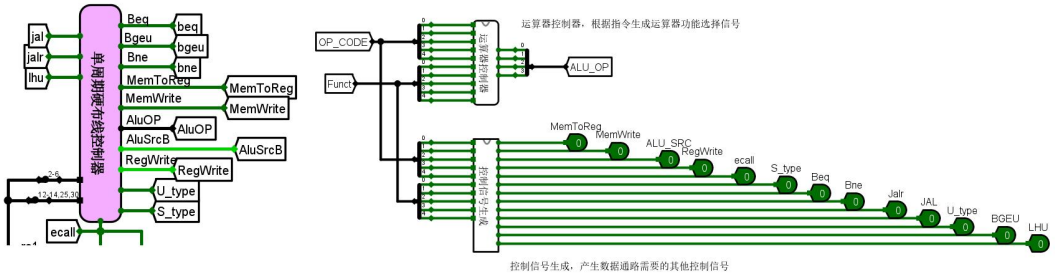
3.1.3 控制器的实现

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 3.7 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。

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

图 3.7 硬布线控制器控制信号框架

根据指令产生对应的控制信号，硬布线控制器的具体实现如图 3.8 所示。



a) 硬布线控制器封装图　　b) 硬布线控制器实现电路

图 3.8 硬布线控制器（CON）

3.2 中断机制实现

3.2.1 单级中断实现

根据单级中断机制设计，需要设计中断信号采样电路、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。进行如下详细实现介绍。

1) 中断信号采样实现

对于一个突发的中断请求，需要进行持续中断信号的产生来让电路察觉，并做出中断响应。具体实现如图 3.9 所示。

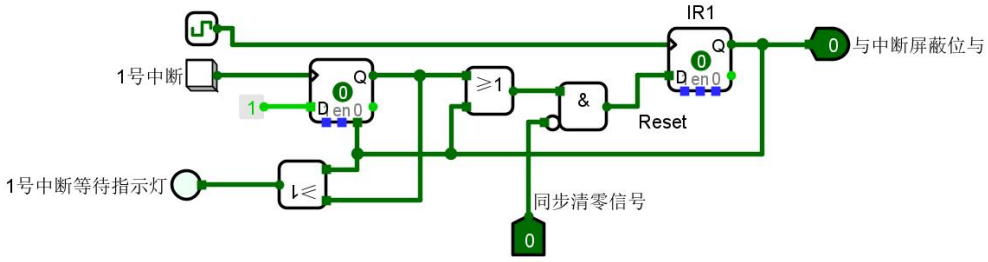
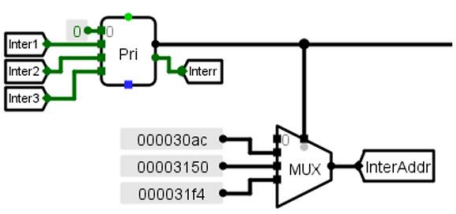


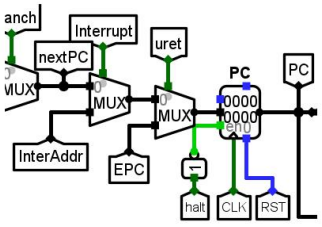
图 3.9 中断信号采样电路

2) 中断响应实现

对于中断信号 inter1、inter2、inter3 进行优先编码器筛选，选出优先级更高的中断信号优先处理，并将对应的跳转地址送入 PC。送入采用多路选择器，输出对应端口的中断地址。具体实现如所示。



a）中断响应选择



b）地址传入

图 3.10 中断响应

3) 中断使能状态实现

设置一个 1 位寄存器为 IE 寄存器，为 0 时表示开中断，为 1 时表示关中断。在中断响应时，需要改变 IE 状态为关中断，初始状态为开中断，且产生 interrupt 信号指示中断电路工作。具体实现如所示。

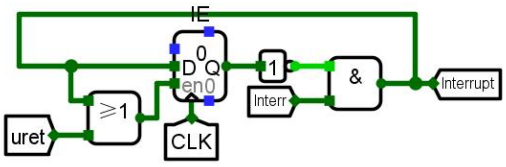


图 3.11 中断使能状态

4) EPC 断点保存实现

使用一个 32 位寄存器作为 EPC 断点保护寄存器。接收到 interrupt 后，将断点处的下一跳地址存入 EPC 寄存器中。当需要中断返回时，将保存的地址送入 PC。具体实现如图 3.12 所示。

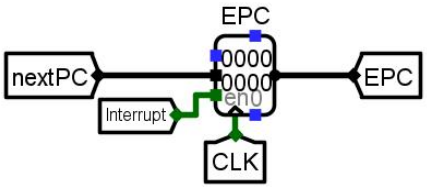


图 3.12 EPC 断点保护

5) 中断返回实现

当前中断号存放在一个寄存器中，当中断处理完成后，进行该中断返回。置 IE 状态为开中断，且将 EPC 断点保存地址传入 PC 寄存器中，并关闭对应的持续中断信号。具体实现如图 3.13 所示。

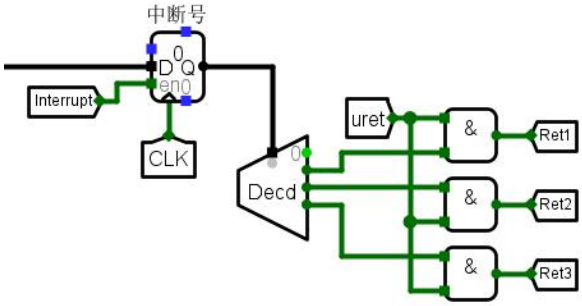


图 3.13 中断返回

3.2.2 多级中断实现

根据多级中断机制设计，其与单级中断的差别在于中断使能状态条件设置更多，包含硬件中断使能调整和软件中断使能调整。另外，由于多级中断允许中断嵌套，那么不仅对于源程序断点，对于每个外层中断断点同样需要进行断点保护。对此两点扩展，进行进行如下详细实现介绍。

1) 中断使能状态实现

多级中断包含硬件使能调整和软件使能调整。硬件使能调整为中断响应时的关中断和中断返回时的开中断，而软件使能调整为 csrrsi 指令开中断和 csrrci 指令关中断。具体实现如图 3.14 所示。

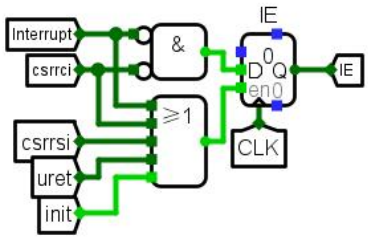


图 3.14 多级中断的中断使能状态

由于中断信号持续产生，那么需要一个屏蔽字来进行低优先级信号的屏蔽，高优先级信号的响应。这里由于优先级按编号递增，那么只需要一个优先编码器即可，但存在当前相同中断不能屏蔽的问题。那么采用中断号入栈来进行管理，当响应一个中断请求时，将依次入中断号响应栈，若接收到了更高优先级的中断请求，那么将该中断号入栈。每次响应中断请求时，总是接收栈顶的中断号的请求。当进行中断切换时，发出 shield 信号。由中断请求信号、IE 状态信号和 shield 信号协同产生 interrupt 信号。响应栈采用 3 个寄存器来实现，栈指针由计数器来实现。具体实现如图图 3.15 所示。

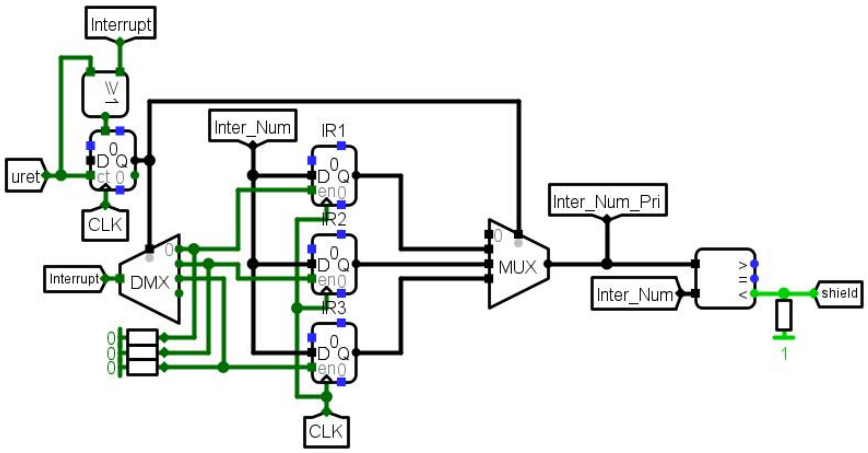


图 3.15 中断号响应栈

2) EPC 硬件堆栈断点保护实现

在进行中断响应时，需要保存当前程序的断点地址信息。采用硬件堆栈来保存断点。由于最多 3 个中断同时响应，那么堆栈开辟 3 字大小即可，使用 3 个 32 位寄存器来实现。根据当前中断的中断号对应的寄存器来存放信息。具体实现如图 3.16 所示。

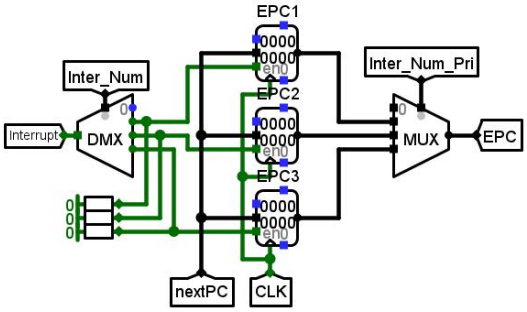


图 3.16 EPC 硬件堆栈断点保护

3.2.3 流水线中断实现

流水线中断采用重定向流水线电路，实现单级中断。因此流水线中断机制与单级中断机制相同，只是在一些细节上需要注意。

由于在 EX 段需要进行分支且重定向机制的介入，对应处理的断点信息会比较复杂。因此选取 MEM 段地址作为断点地址较优。当产生中断时，将 MEM 段的地址存入 EPC 寄存器中，相应的将 IF、ID、EX 取得的指令清空，即清空 IF/ID、ID/EX 与 EX/MEM 流水寄存器。当中断返回时，使用 WB 段 uret 信号，将所有的流水寄存器清空，并将 EPC 寄存器保存的断点送入 IF 段的 PC 寄存器中。

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水寄存器包含 3 个控制端口，CLK 时钟控制端、RST 清零端和 EN 使能端。其他端口则是成对的数据端口，用于传入或传出寄存器的存储数据。流水寄存器本质上是若干个寄存器的封装。以 IF/ID 流水寄存器为例，具体实现如图 3.17 所示。

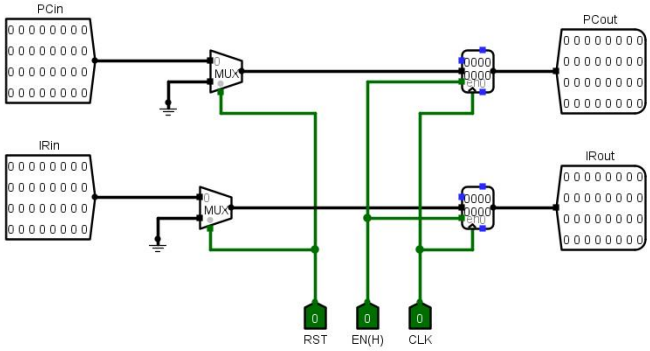


图 3.17 流水接口部件（IF/ID 为例）

3.3.2 理想流水线实现

构造各部件后，进行各功能段的组装。按照理想流水线设计思路，进行取指、译码、执行、访存、写回，5 个功能段的功能实现，对各段内的部件进行数据通路连接。具体实现如所示。

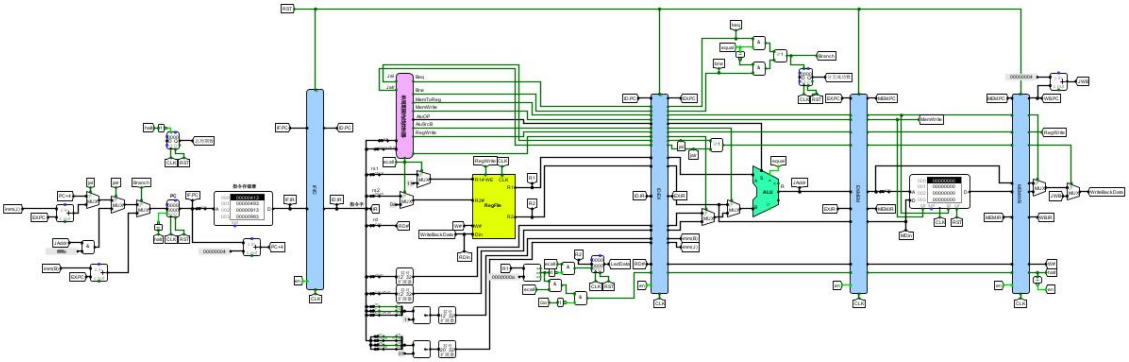


图 3.18 理想流水线

3.4 气泡流水线实现

3.4.1 EX 段分支实现

在 EX 段进行分支跳转，即在 EX 段检测出条件分支与无条件分支，对于条件分支判断是否满足跳转条件。条件分支检测如图 3.19 所示。当在 EX 段检测出满足条件的条件分支和无条件分支时，进行跳转地址向 IF 段 PC 寄存器的赋值，且对 IF/ID、ID/EX 清空。

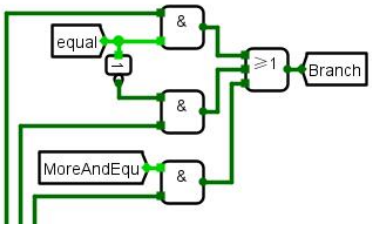


图 3.19 条件分支检测

3.4.2 数据冲突检测和处理实现

根据数据冲突检测思路，首先进行寄存器有效性判断，使用组合逻辑来判断 ID

段指令的 R1、R2 寄存器是否有效。再将 ID 段的有效源寄存器分别与 EX 段、MEM 段的目的寄存器进行并行比较，若产生冲突则输出数据相关。具体实现如图 3.20 所示。

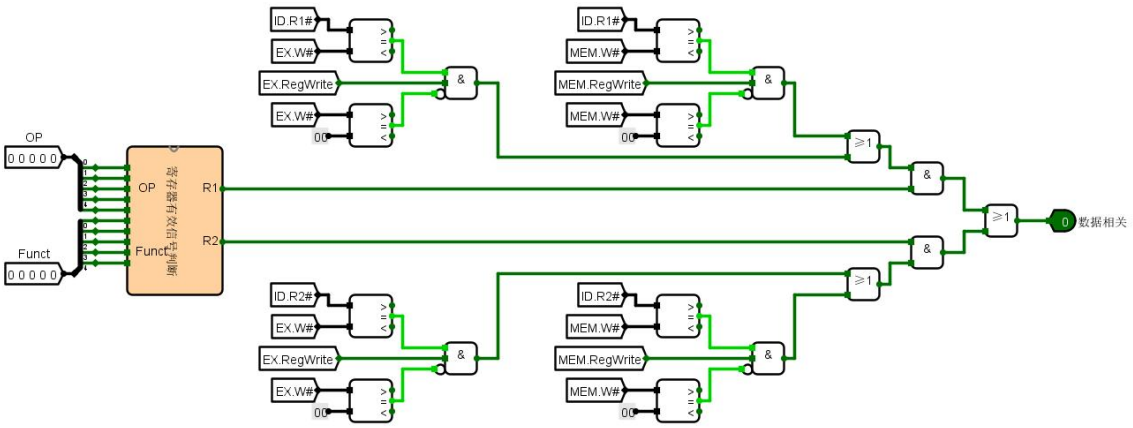


图 3.20 数据相关检测

当检测出数据相关时，需要在 EX 段插入气泡。即清空 ID/EX 段，对 IF 段和 ID 段进行暂停控制。

3.4.3 气泡流水线实现

EX 段分支解决分支冲突问题，数据相关检测和先写后读机制解决数据冲突问题。在理想流水线上进行迭代，具体实现如图 3.21 所示。

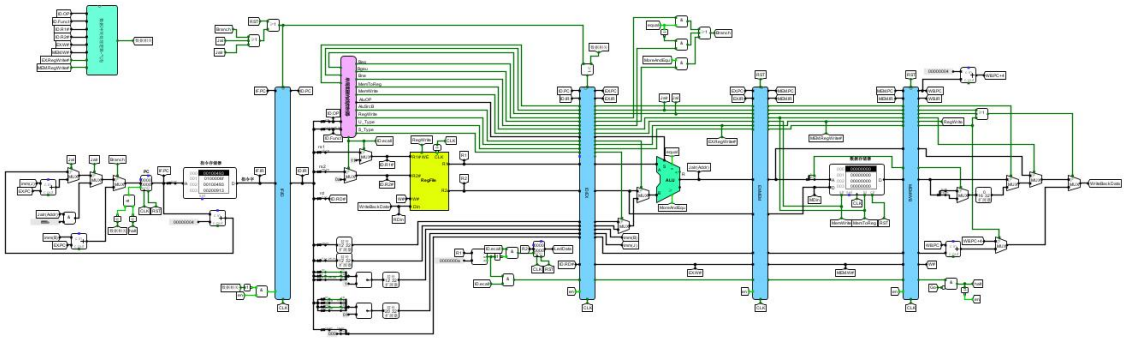


图 3.21 气泡流水线

3.5 重定向流水线实现

3.5.1 含 Load-Use 的数据相关检测与处理实现

数据相关检测与气泡流水线的数据相关检测相同，在此基础上添加了具体的冲突状态输出，以及 Load-Use 检测。冲突状态采用 2 位数据记录 3 种状态：0 为不冲突，1 为与 MEM 段冲突，2 为与 EX 段冲突。Load-Use 则检测 ID 段与 EX 段冲突时，EX 段的冲突指令是否为访存指令。具体实现如图 3.22 所示。

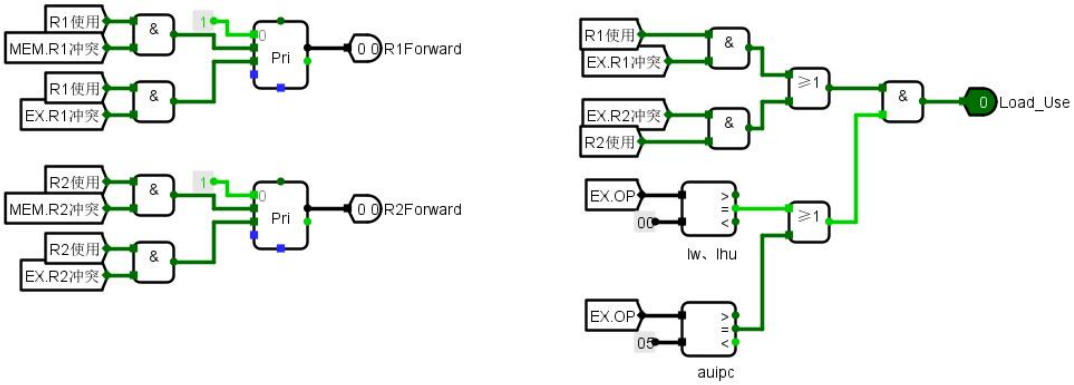


图 3.22 冲突状态扩展与 Load-Use 检测

对与 Load-Use 相关冲突，采取在 EX 段插入气泡的方式解决，参考气泡流水线数据冲突处理。

3.5.2 重定向流水线实现

在 ID 段检测的冲突状态传递到 EX 段后，对 R1 和 R2 的值进行重定向，状态为 0 时，保持原传递数值；为 1 时，重定向到 WB 段的写回数据；为 2 时，重定向到 MEM 段的计算结果数据。具体实现如图 3.23 所示。

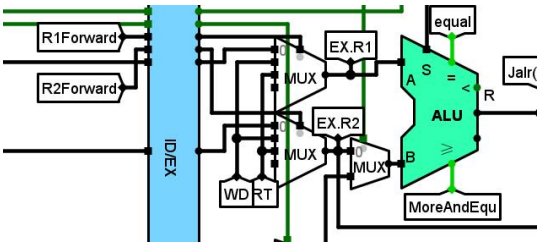


图 3.23 重定向机制

其他功能参考 EX 段分支处理流水线，重定向流水线在其基础上进行迭代，重定向流水线电路图如图 3.24 所示。

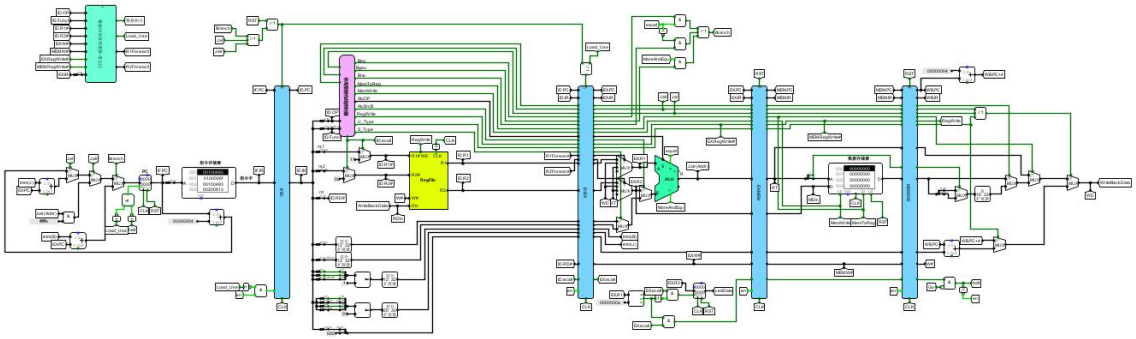


图 3.24 重定向流水线

3.6 动态分支预测机制实现

在重定向流水线的基础上，实现动态分支预测。主要包括 3 个过程实现，IF 段预测处理、EX 段分支判断和分支调整。

IF 段预测处理，则是在 IF 段命中或不命中 BTB 表的处理。采用超前预测的机制，在 IF 段计算出跳转地址，写入 BTB 表中，且置状态为 11 跳转状态。超前预测地址实现如图 3.25 所示。

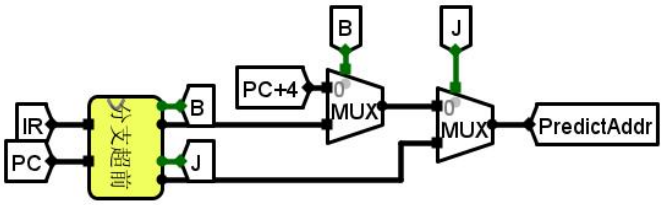


图 3.25 分支超前预测

EX 段分支判断，则是在 EX 段进行查找 BTB 表，并对分支预测历史位根据双位预测状态进行调整。并且匹配实际跳转与预测跳转的一致性，来进行预测正确与否的判断。BTB 表的 cache 行如图 3.26 所示。

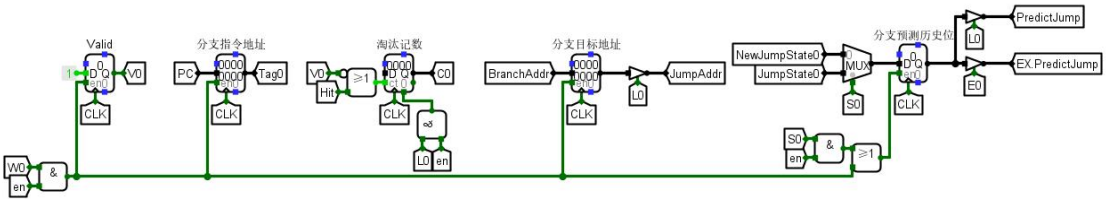


图 3.26 BTB 表内的 cache 行

根据 EX 段判断的预测正确性进行分支调整。若判断正确，则不对分支进行调

整；若判断失败，则修改分支，将正确的调整地址送入 IF 段的 PC 寄存器，且清空 IF/ID、ID/EX 流水寄存器。分支调整的具体实现如图 3.27 所示。

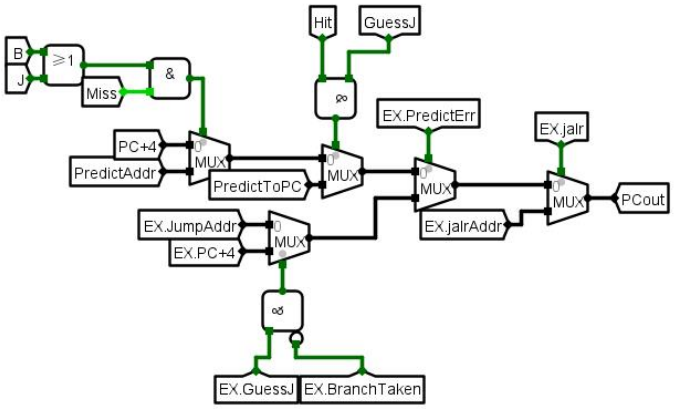


图 3.27 分支调整

根据上述机制，将其与重定向流水线的相关接口进行连接，实现带有动态分支预测的重定向流水线，如图 3.28 所示。

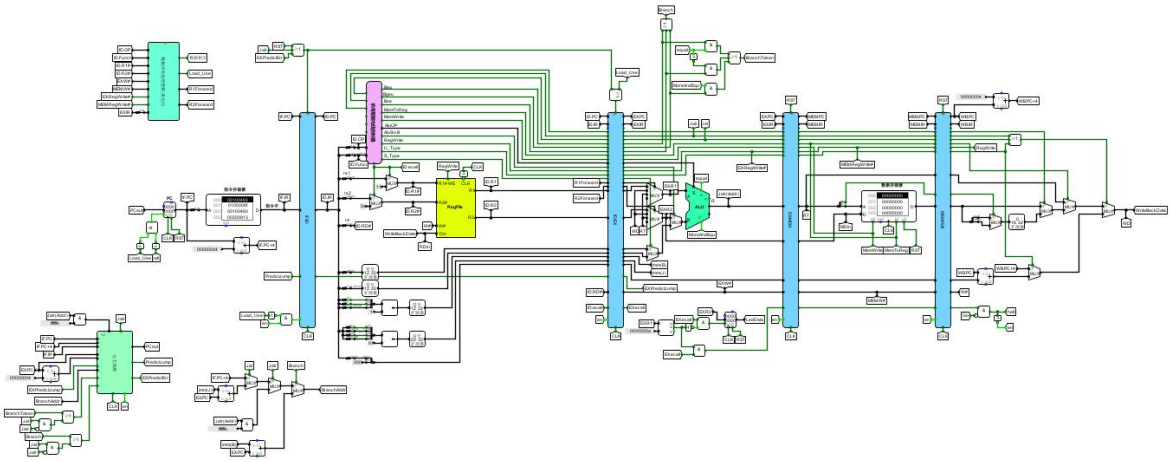


图 3.28 带动态分支预测的重定向流水线

4 实验过程与调试

4.1 测试用例和功能测试

对实现完成的单周期 CPU、中断机制、各类流水线以及动态分支预测机制进行功能测试。测试用例包含 RISC-V 带扩展指令的 benchmark 测试程序、中断相关的单级中断测试程序和实现 EPC 硬件堆栈保护的多级中断测试程序。具体测试如下。

4.1.1 单周期 CPU 测试

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，测试截图如所示。功能测试成功。

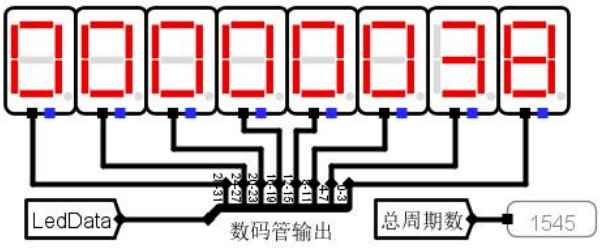


图 4.1 单周期 CPU-benchmark 测试

2) 扩展指令 CCAB 测试

测试扩展指令 CCAB，分别为 SRL、AUIPC、LHU、BGEU。

SRL 指令测试时，LED 显示 876 从左向右移位。AUIPC 指令测试时，LED 显示当前 PC 与 0x430 左移 12 位的立即数相加的和。测试截图如图 4.2 所示。LHU 指令测试时，LED 输出从 0x8281 到 0xc0bf 的半字等差数列。BGEU 指令测试时，输出从 0xf 到 0x1 的递减数列。

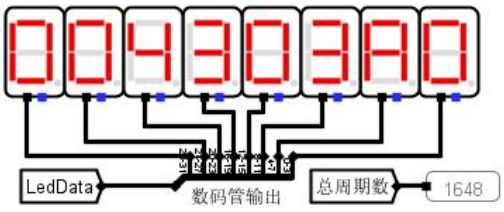


图 4.2 单周期 CPU-扩展指令 Auipc 测试

4.1.2 中断机制测试

1) 单级中断测试

将单级中断测试程序转为.hex 文件后，载入指令存储器中。在循环运行过程中，按下按钮 1，进入 1 号中断子程序，如图 4.3 所示。

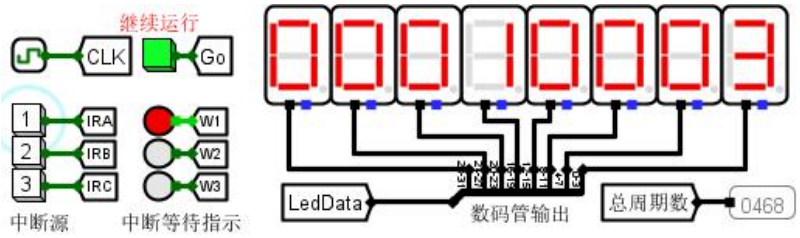


图 4.3 单级中断测试 1

在处理 1 号中断时，按下按钮 2 与按钮 3。没有被其他中断打断，当 1 号中断返回后，3 号中断被优先响应，如图 4.4 所示。单级中断机制测试成功。

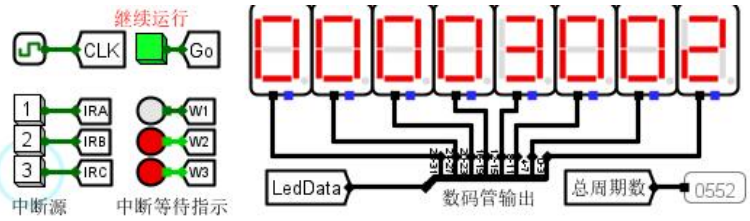


图 4.4 单级中断测试 2

2) 多级中断测试

将使用 EPC 硬件堆栈保护的多级中断程序转为.hex 文件后，载入指令存储器中。在循环过程中，按下按钮 1，进入 1 号中断子程序，如图 4.5 所示。

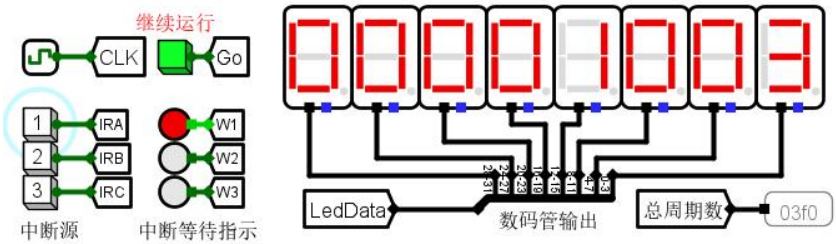


图 4.5 多级中断测试 1

依次按下按钮 2 与按钮 3，程序进入 3 号中断子程序。如图 4.6 所示。在 3 号中断返回后，进入 2 号中断子程序，如图 4.7 所示。多级中断机制测试成功。

3) 流水线中断测试

中断测试方法与单级中断测试相同，不再赘述。流水线中断测试成功。

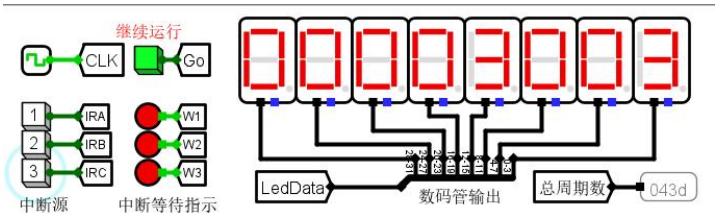


图 4.6 多级中断测试 2

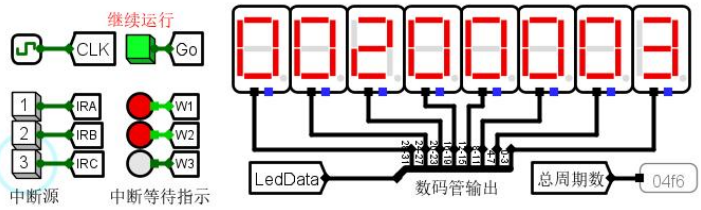


图 4.7 多级中断测试 3

4.1.3 5 段流水 CPU 测试

1) 理想流水线测试

由于理想流水线为最基本架构，故在后续测试气泡流水线与重定向流水线时包含对理想流水线的测试。

2) 气泡流水线测试

将带扩展指令的 benchmark 程序转为.hex 文件后，载入指令存储器中。当运行到第一个停机 ecall 指令时，表示基础 benchmark 运行完成，不包含最后 ecall 指令的总周期数为 3623，结果如图 4.8 所示。插入气泡数为 2074。点击继续运行按钮，测试 4 条扩展指令，LED 显示符合预测值，测试成功。

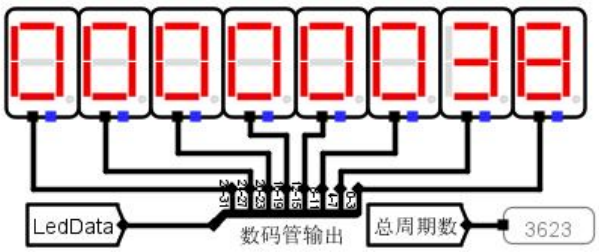


图 4.8 气泡流水线测试

3) 重定向流水线测试

将带扩展指令的 benchmark 程序转为.hex 文件后，载入指令存储器中。当运行到第一个停机 ecall 指令时，表示基础 benchmark 运行完成，不包含最后 ecall 指令的总周期数为 2297，结果如图 4.9 所示。Load-Use 次数为 120 次。点击继续运行按

钮，测试 4 条扩展指令，LED 显示符合预测值，测试成功。

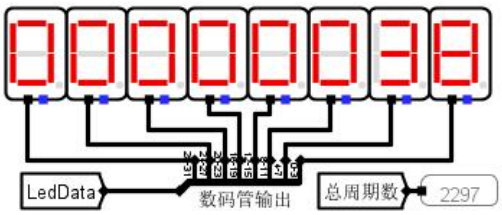


图 4.9 重定向流水线测试

4.1.4 动态分支预测机制测试

在重定向流水线上增加动态分支预测机制，测试方式与重定向测试相同。运行基础 benchmark 程序后的不包含最后 ecall 指令的总周期数为 1801，如图 4.10 所示。

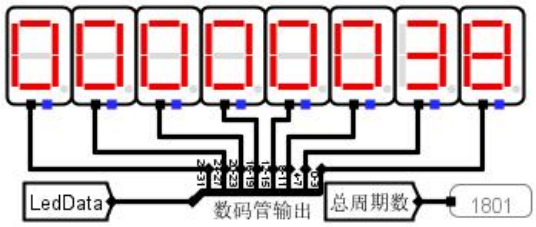


图 4.10 动态分支预测机制测试

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，不含最后一条停机 ecall 指令。

表 4.1 benchmark 运行周期数统计表

CPU 方案	Benchmark 程序运行总周期数（不含停机指令）
单周期 CPU	1545
气泡流水线	3623
重定向流水线	2297
带动态分支预测的重定向流水线	1801

分析表中数据，单周期 CPU 总周期数最少，但单周期 CPU 的时钟周期约为 5 段流水 CPU 的时钟周期的 5 倍，故总体而言，单周期 CPU 性能差于 5 段流水 CPU。分析比较 5 段流水 CPU 中的不同实现方案，气泡流水线性能较差，原因是对于数据冲突和分支冲突统一采用插入空指令的方式避免，增加了周期数。而重定向流水线

在数据冲突部分做出进一步优化，相比于气泡流水线得到了很大提升。动态分支预测则进一步降低了分支冲突处理的时延，由于处理分支冲突固定需要插入 2 个空指令，不妨进行动态分支预测，在大概率预测成功的情况下，减少了空指令的插入，进一步提高了 CPU 性能。

4.3 主要故障与调试

4.3.1 单周期 CPU 故障

1) 指令处理问题

故障现象：J 型指令与 B 型指令提取立即数时产生跳转错误。

原因分析：jal 指令属于 J 型指令，beq、bne、bgeu 指令属于 B 型指令。对于上述指令字的立即数部分只产生[20:1]位与[12:1]位的数据。

解决方案：如图 4.11 和图 4.12 所示，用分线器分解好 offset 字段后，将字段整合并左移 1 位。

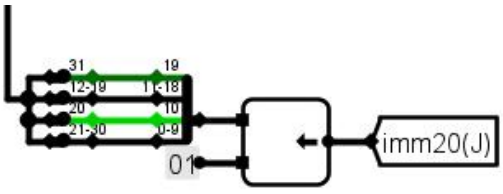


图 4.11 J 型指令译码解决方案

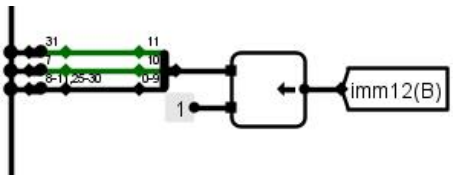


图 4.12 B 型指令译码解决方案

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。

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 a7=10。

4.3.2 中断机制故障

1) 存储断点地址问题

故障现象：如图 4.13 所示，若将 PC+4 作为断点地址，若断点处为分支指令，则会产生返回错误的情况。

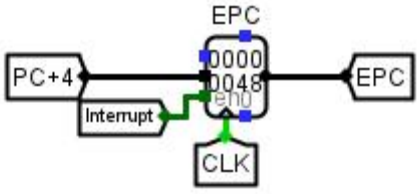


图 4.13EPC 断点设置图

原因分析：在进行一般顺序执行时，在源程序执行时发生中断后，断点地址确实是该地址的下一跳地址 PC+4。但若当前正处于分支指令，且需要分支跳转时，跳转地址并非 PC+4，而是计算的地址值。

解决方案：修改传入 EPC 的地址，设置为下一跳进入 PC 寄存器的地址值，这样即包含了顺序执行的情况，也包含了分支跳转的情况。

4.3.3 5 段流水 CPU 故障

1) 气泡流水线：halt 信号生效位置问题

故障现象：如图 4.14 所示，在 EX 进行暂停功能时，会导致后续指令无法完成的情况。

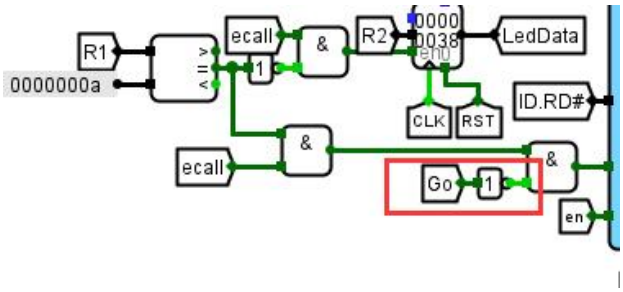


图 4.14 气泡流水线 halt 停机信号生效位置问题

原因分析：当在 ID 段产生 ecall 信号时，需要传到在 WB 段进行对应的功能响应，否则提前响应会导致后续指令停留在流水线中的问题。

解决方案：将 halt 信号通过各段间流水线传递到 WB 段后，执行停机。

2) 重定向流水线：halt 信号产生位置问题

故障现象：当如气泡流水线一样，在 ID 段分析的 ecall 指令功能时，会出现停机信号 halt 产生失败的问题。

原因分析：由于重定向在 ID 段分析好数据相关性后，在 EX 段进行寄存器数值重定向。而 ecall 指令同样需要使用 R1 和 R2 两个寄存器，故也需要在 EX 段进行重定向后，进行 R1 和 R2 寄存器的分析，进而产生 halt 指令。

解决方案：如图 4.15 所示，将 ecall 信号传递到 EX 段进行执行，使用重定向后的 R1 和 R2 寄存器值分析获得 halt 信号。

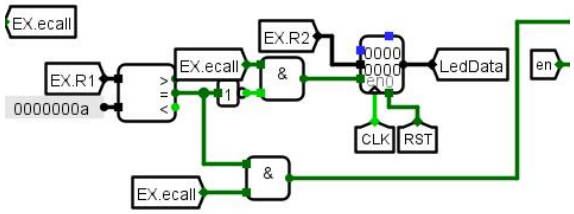


图 4.15 重定向流水线 halt 停机信号有效产生位置

4.4 实验进度

表 4.2 课程设计进度表

时间	进度
第 1~4 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 5~6 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 7 天	完成气泡流水线的搭建，调试并排除故障。
第 8 天	完成重定向流水线的搭建，调试并排除故障。
第 9~10 天	学习中断机制相关内容，完成单级中断机制实现，完成断点保护、中断响应、中断返回的硬件实现。
第 11 天	完成多级中断机制和流水线中断，完成 EPC 堆栈、中断号栈等硬件实现。
第 12~13 天	学习动态分支机制相关内容，完成动态分支预测机制，搭建 BTB 表，实现预测数据通路。
第 14~20 天	完成团队任务，设计任务思路，完成硬件搭建，完成软件实现，调试并排除故障。

5 团队任务

5.1 项目选题与分工

5.1.1 项目选题

本队因成员变动的原因，再三斟酌，选择五子棋的简易版本“井字棋”作为本次团队项目。井字棋规格大小为 3×3 方格，O 和 X 分别代表两名游戏玩家。轮流在格子里留下对应的标记，相同的三个标记连成一条横线、竖线或斜线时，则对应玩家获胜。

5.1.2 团队分工

该项目可分成 3 大模块，硬件模块、软件模块与输入输出模块，由我与 两人共同完成设计。我主要负责实现硬件部分和软件实现中与输入输出相关部分， 主要负责实现软件部分。软硬接口与其他工作由我们共同负责完成。

5.2 项目设计

整体项目设计包括 3 个方面，分别为硬件设计、软件设计与输入输出设计。详细设计如下：

1) 硬件设计

项目选择支持单级中断的 RISC-V 单周期 CPU 的基础上进行硬件修改与扩展。首先，软件部分需要进行软件中断，故需要提供硬件支持接口，选取设置一个中断寄存器来进行软件中断控制。其次，考虑到数据需要多次反复获取，为了节约时间、提升系统效率，采用寄存器寻址的方式存放与获取游戏数据，此时硬件中需要对相应寄存器设置获取数据的通路。对于硬件中断部分，由原来的 3 个中断服务扩展为 9 个，作为下子的硬件中断实现部分。同时中断入口地址也相应的扩展为 9 个入口选择。对于 ecall 指令的数据通路需要做修改，将其 LED 显示功能修改为 LCD 显示屏显示功能，且保留停机功能。

2) 软件设计

在此仅对我负责的绘图功能软件相关部分做出具体设计描述。LCD 显示器通过

两个接口 X 与 Y 获取的（X，Y）坐标信息来显示像素点。为此，我们将绘图功能交由软件实现。设计通过 1 号与 2 号通用寄存器来进行 X 与 Y 坐标的传递。由于 X 与 O 的图像大小固定，且各图像内像素点的相对位置固定，故可以统一设计模块来对图像进行绘制，只需要确定图像中心位置即可。

3) 输入输出设计

输入设备包括 9 个落子按键与 1 个重新游戏按键。9 个落子按键进行中断请求，每个落子按键的按下对应一个中断请求。重启游戏按键产生 Reset 信号，用于重置 CPU 各功能部件以及清空显示屏。

输出设备包括 1 个 LCD 显示屏，用于显示游戏界面。LCD 显示屏由时钟信号控制，一个时钟周期内，根据传入的坐标信息显示 1 个像素点。而（X，Y）坐标信息通过软件给设定的通用寄存器赋值，经过硬件设置的相关寄存器数据通路来获取。

5.3 项目实现

1) 硬件实现与输入输出接口实现

由于硬件与输入输出设备联系紧密，故在此将其一同实现。

对于中断机制的扩展，首先实现中断服务数的扩展与入口地址选择端的扩展，如图 5.1 所示。并在中断使能信号的产生时，添加软件中断控制信号的与控制。

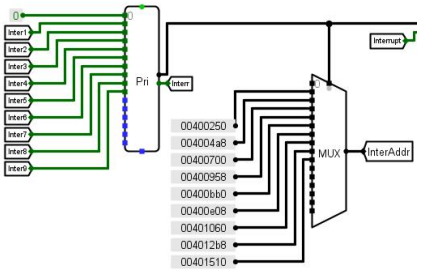


图 5.1 扩展的中断服务入口选择端

获取（X，Y）坐标信息的数据通路如图 5.2 所示。采用两个寄存器存放 X 坐标与 Y 坐标，在 ecall 信号到来时进行对应坐标像素点的显示。将对应硬件实现线路与 LCD 显示屏的有关接口进行连接。

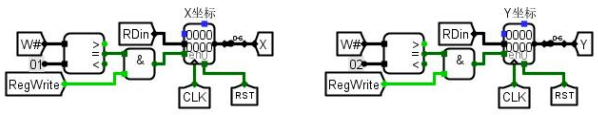


图 5.2 获取坐标信息的数据通路

2) 软件实现

与图像绘制相关的软件模块中，包括初始化阶段与对应图像绘制阶段。初始化阶段中，需要绘制 3×3 的棋盘，通过采用嵌套循环进行 X 与 Y 坐标的递增，逐渐绘制一条直线。而对于图像绘制阶段，则将图像中心坐标赋给接口寄存器 s2 与 s3 中，接着调用固定模块即可。例如对于 X 绘制模块如图 5.3 所示，其中 ra、sp 为坐标参数，循环进行 ecall 显示像素点，绘制出线段。

```
X_Win:              ##X_Win结果
addi t6,zero,1
addi t2,zero,1
addi s2,zero,63
addi s3,zero,63

##画X
addi ra,s2,-17
addi sp,s3,-17
addi a4,zero,34
Draw_X1x:
ecall
addi ra,ra,1
addi sp,sp,1
addi a4,a4,-1
bne a4,zero,Draw_X1x
```

图 5.3 X 图像绘制模块

5.4 项目测试

5.4.1 功能测试

1) X 胜利测试

将软件程序转为十六进制程序并导入指令存储器中，运行系统。LCD 显示 3×3 网格。依次点击按钮 1、2、5、6、9 表示 X 与 O 的落子顺序与位置。LCD 显示落子与获胜结果如图 5.4 所示。

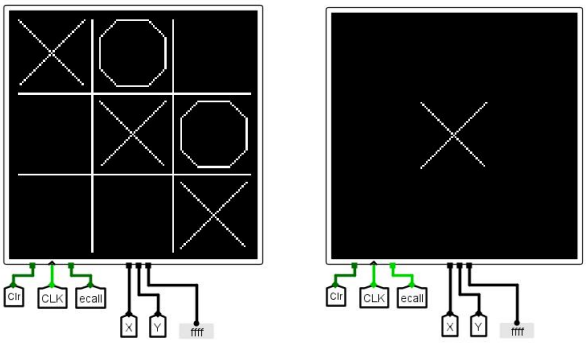


图 5.4 ×获胜情况的测试与显示

2) O 胜利测试

点击重新开始游戏。LCD 显示 3×3 网格。依次点击按钮 1、2、3、5、6、8，表示 X 与 O 的落子顺序与位置。LCD 显示落子与获胜结果如图 5.5 所示。

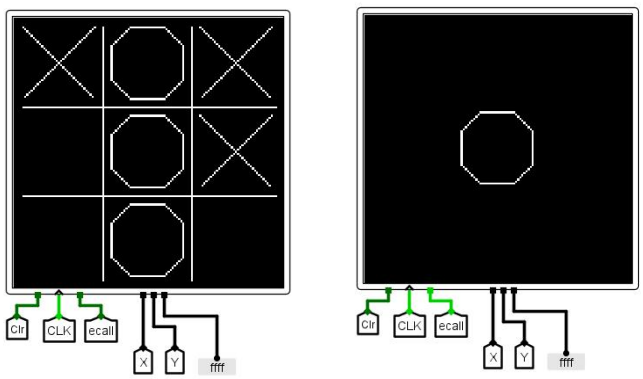


图 5.5 O 获胜情况的测试与显示

3) 平局测试

点击重新开始游戏。LCD 显示 3×3 网格。依次点击按钮 5、1、3、7、4、6、8、2、9，表示 X 与 O 的落子顺序与位置。LCD 显示落子与获胜结果如图 5.6 所示。

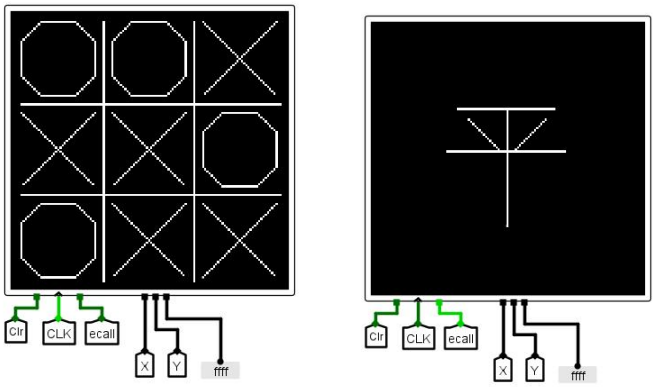


图 5.6 平局情况的测试与显示

5.4.2 问题故障与解决

在进行团队合作时，需要进行合理的分工。因为我们一人负责硬件，一人负责软件，其中势必会涉及到接口问题，比如寄存器的使用规定等。由于团队人数较少，可以直接进行面对面交流讨论，对于寄存器的使用规定问题，我们则使用在线文档的方式进行归纳。对于软件使用的寄存器，硬件则设计对应的读取数据通路，来实现软件所需的功能。

6 设计总结与心得

6.1 课设总结

对于此次组成原理课程设计，共完成如下几点工作：

1）设计并实现了支持 24 条基础指令与 4 条扩展指令的单周期 CPU。

2）设计并实现了带单级中断、多级中断的单周期 CPU 和带单级中断的流水线。

3）设计并实现了 5 段流水 CPU，包括理想流水线、气泡流水线和重定向流水线。

4）设计并实现了对重定向流水线进行动态分支预测机制的扩展。

5）完成了单周期 CPU、中断机制与 5 段流水 CPU 的调试与测试。

6）完成了线上 educoder 平台的测试与线下的检查。

7）完成了团队任务的设计与实现，答辩 PPT 与答辩视频的制作。

6.2 课设心得

此次课程设计收获颇丰，让我理解并掌握了多种 CPU 架构，巩固了中断的实现机制。此次设计最大的感受就是由浅入深，层层递进。从最开始的单周期 CPU 开始摸索，设计单周期 CPU 的各个通路，理解各类指令的解析，实现硬布线控制器以及完成各阶段的联系。不仅加深了对组成原理知识的理解，也通过动手实践连线，理解了底层各功能部件的内在联系，这对于搭建更好的“上层建筑”有非常重要的实质意义。在实现基础的单周期 CPU 时，遇到了许许多多的错误，如今会看会发现错误非常简单，但正应征了万事开头难这一道理。将单周期 CPU 实现后，会发现后面无论是 5 段流水 CPU 的构建，还是中断机制的实现，都十分如鱼得水。

在单周期理解与实现的基础上，完成 4 条指令的扩展就十分方便。在构建 5 段流水 CPU 时，也是采用层层递进的思想完成。从简单的理想流水线，到可以解决各类冲突的气泡流水线，再到优化数据冲突时延的重定向流水线，最后到优化分支冲突的动态分支预测机制的实现。可以发现前人厉害的思想并不是一蹴而就，而是不断发现问题，不断迭代而来的。这对今后的学习和工作有很大的思想帮助。

接着则是完成了中断机制的实现，从单级中断机制到多级中断机制，再到流水线中断也是采用递进的思想。单级中断实现没有用到软件中断，而多级中断机制在

单级中断的基础上，添加了软件中断与断点保护栈。而流水线中断则是对中断机制的使用，掌握了如何在流水线中插入中断机制的方法与实现细节。

随后的团队任务也是一波三折，从最开始的四人小队，最后缩减为了两人，选题也发生了改变。但即使再曲折，最后也在队友的陪伴下完成了井字棋游戏的设计。当看到使用自己搭建的 CPU 运行自己设计的游戏系统时，成就感爆棚！可以说这是目前的实验中最让我感到心满意足的了。

最后就是对此次课程设计的一些意见和建议了。首先是对线上评测的看法，它大大降低了调试难度，尤其在一些肉眼难以发现的小问题上，可以通过 educoder 测试很快的找到，点赞。然后，对于没有线上测试的实验，实验书与实验 PPT 上没有清晰的标准说明，在调试与测试时很苦恼，比如对于动态分支预测的测试，无法得知达到何种优化程度才算合理，需要翻找群友的聊天记录才能得到一个大概的数据，希望老师能对各实验环节提供测试标准。最后则是关于团队任务的一些建议，由于团队任务的说明文字很少，方向性很难把握，导致组队和选题时产生很大问题。因为对于未知任务的恐惧，一些组队的同学可能在不太清楚目标的情况下选择退缩，这样导致组队和最初选题的落空，有些遗憾。希望老师能够提供一些可选选题和单人选题供大家选择，增加团队任务的目标性和方向性。

以上是我对此次课程设计的心得与建议。时光荏苒，或许会遗忘实验的具体操作，但不会忘记的是与同学老师们一同解决问题的欣喜与豁然，以及实验蕴含的通解与真理。

参考文献

[1] DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第4版).北京：机械工业出版社.

[2] David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社

[3] 谭志虎，秦磊华，吴非，肖亮.计算机组成原理. 北京：人民邮电出版社，2021年.

[4] 谭志虎，秦磊华，胡迪青.计算机组成原理实践教程.北京：清华大学出版社，2018.

[5] 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011年.

[6] 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008年.

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明!

作者签字: