华中科技大学

2020

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：	CS1705
学　　号：	U201714707
姓　　名：	练炳诚
电　　话：	18674030550
邮　　件：	1270464410@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	SRSV	算术可变右移	指令格式参考 MIPS32 指令集，最终功能以 MARS 模拟器为准。
29	XOR	异或	
30	LH	加载半字	
31	BLEZ	大于等于 0 转移	

2 总体方案设计

2.1 单周期 CPU 设计

本次我们采用的方案是硬布线控制，采用哈弗结构，将数据和指令分开存储，即采用硬布线控制方式，实现数据储器（DM）和指令存储器（IM）不共用一个存储器的方式完成方案的设计。在一个时钟周期内，从 IM 中取出指令送入硬布线控制器进行指令译码，产生各功能部件的控制信号，完成各指令的功能，并计算出下一条指令的地址。

总体结构图如图 2.1 所示。

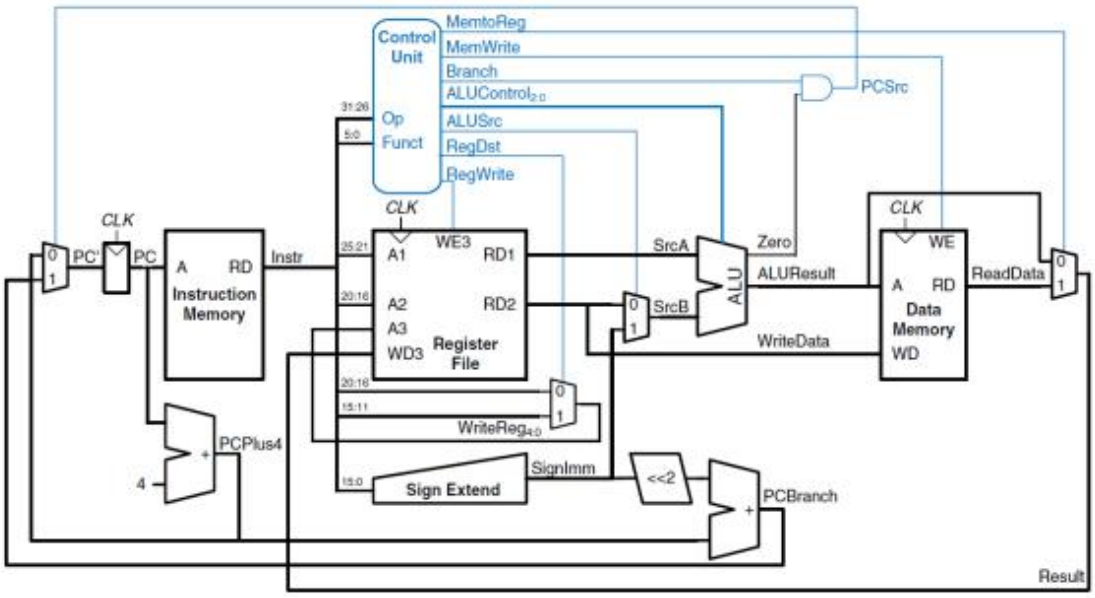

图 2.1 总体结构图

2.1.1 主要功能部件

1. 程序计数器 PC

程序计数器 PC 中保存了当前指令的字节偏移量，由于是实现的 MIPS 指令集是 32 位，每条指令占 4 个字节，IM 中数据按字节编址，因此访问 IM 取指令时将当前 PC 寄存器的值右移两位即可计算出当前指令在 IM 中的地址。PC 寄存器的输入端是下一条指令的 PC 值，该值根据当前指令是顺序寻址方式还是跳跃寻址方式由数据选择器选择输入，数据选择器的选择信号将由硬布线控制器和指令执行的结

果产生。当时钟周期到来时，将下一条指令的 PC 写入 PC 寄存器，若执行到停机信号，将 PC 寄存器的使能端置为 0，不再修改 PC。

2. 指令存储器 IM

IM 使用 ROM 保存了程序的机器码，将一段 MIPS 程序翻译成机器码后，烧录到 ROM 中。采用 32 位 MIPS 指令集时，ROM 每个存储单元也是 32 位，即每个存储单元存储一条指令的机器码。当时钟周期到来时，将程序计数器 PC 值逻辑右移两位后得到指令在 ROM 中的地址，访问 ROM，加载出目标指令的机器码，进行指令译码。

3. 寄存器堆 RF

RF 是由多个 32 位寄存器组成的阵列，对应于 MIPS 中的$0~$31 寄存器，由于一条 MIPS 指令最多涉及两个寄存器的同时读出和一个寄存器的写入，故 RF 支持两个寄存器的同时读出及一个寄存器的写入，RF 输入端有三个 5 位寄存器编号、一个写使能信号和写入数据，输出端为两个 32 位寄存器的读出数据。时钟周期到来时，若写使能有效，根据写入寄存器的编号将数据写入到对于的寄存器，否则根据两个读出寄存器的编号将相应寄存器的数据读出到输出端。

4. 运算器 ALU

ALU 主要负责指令执行阶段中的运算工作，包括算术运算类型指令的算术运算结果、分支类型指令的分支目标地址、逻辑运算类型指令的逻辑运算结果，其运算功能选择信号由硬布线控制器指令译码给出，ALU 规格见表 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
shamt	输入	5	移位运算移动的位数
ALU_OP	输入	4	运算器功能码，具体功能见下表

引脚	输入/输出	位宽	功能描述
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

5. 数据存储器 DM

使用 RAM 存储程序中使用的数据，内部每个存储单元保存一个 32 位的数据，输入端包括数据地址、写使能信号、写入数据，其中数据地址由访存指令译码后计算得到，写使能信号由控制器产生，写入数据由 ALU 计算或指令译码计算得到，输出端是读出数据，由于 MIPS 指令中的访存指令只访存一次，且读取出的数据均存放在寄存器中，故输出端将与寄存器堆的输入数据相连。

2.1.2 数据通路的设计

根据上述功能部件的设计和输入输出端描述，分析每条指令的数据通路，对图 2.1 设计的总体结构设计数据通路表的框架见表 2.2。

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	取值	说明
ALUOP	0000	ALU 进行逻辑左移运算

控制信号	取值	说明
	0001	ALU 进行算术右移运算
	0010	ALU 进行逻辑右移运算
	0011	ALU 进行无符号乘法运算
	0100	ALU 进行无符号除法运算
	0101	ALU 进行加法运算
	0100	ALU 进行减法运算
	0111	ALU 进行按位与运算
	1000	ALU 进行按位或运算
	1001	ALU 进行按位异或运算
	1010	ALU 进行按位或非运算
	1011	ALU 进行符合比较运算
	1100	ALU 进行无符号比较运算
RegDst (RD)	0	寄存器堆 Din 端数据写入 rt 字段指示的寄存器
	1	寄存器 Din 端数据写入 rd 字段指示的寄存器
RegWrite (RW)	0	时钟周期到来时不更新寄存器堆
	1	时钟周期到来时，将 Din 端数据写入相应寄存器
ALUSrc	0	ALU 的 Y 输入端连接到寄存器 R2 端口
	1	ALU 的 Y 输入端的值连接到指令中立即数 Imm 拓展后的值
MemWrite (MW)	0	数据存储器不进行写入
	1	时钟周期到来时，将数据存储器 D 端数据写入 A 端地址指示的位置
MemToReg (MTR)	0	写回寄存器的数据由 ALU 运算给出
	1	写回寄存器的数据由数据存储器 DM 的输出端给出
BEQ	0	当前指令不是条件分支指令
	1	若 ALU 的运算结果为 0，则下个 PC 的值由 PC+4+Imm16<<2 得到
BNE	0	当前指令不是条件分支指令

控制信号	取值	说明
	1	若 ALU 的运算结果不为 0，则下个 PC 的值由 PC+4+Imm16<<2 得到
BLEZ	0	当前指令不是条件分支指令
	1	若 ALU 的运算大于等于 0，则下个 PC 的值由 PC+4+Imm16<<2 得到
J	0	当前指令不是无条件跳转指令
	1	当前指令是无条件跳转指令，下个 PC 的值根据指令类型决定
JR	0	当前指令不是无条件跳转指令
	1	下个 PC 的值由寄存器堆读出的 R1 的值决定
JAL	0	当前指令不是无条件跳转指令
	1	下个 PC 的值由$[PC]_{31..28}$ | $[IR]_{27..2}$ | 00 决定
SRAV	0	ALU 的 shamt 输入端由指令中的 shamt 字段决定
	1	ALU 的 shamt 输入端由$[R1]_{4..0}$ 决定
Syscall (SC)	0	寄存器堆的 R1#、R2#输入端由 rs、rt 字段决定
	1	寄存器堆的 R1#、R2#输入端分别为 0x02、0x04($a0、$v0 寄存器)
ERET	0	非中断返回指令
	1	中断返回指令，从 EPC 中取出 PC 值
SignExt (SE)	0	16 位立即数 Imm 字段无符号拓展为 32 位
	1	16 位立即数 Imm 字段符号拓展为 32 位
LH	0	不是加载半字指令
	1	根据地址第 1 位是否为 1 决定加载高半字还是低半字，并将结果符号拓展后写回寄存器 Din 端

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	RD	RW	ALUSrc	MW	MTR	ALUOP	BNE	BEQ	BLEZ	J	JR	JAL	SRAV	LH	SC	ERET	SE

2.2 中断机制设计

2.2.1 总体设计

单级中断中，对中断的处理过程大致分为以下几步：

1) 中断仲裁：选择同时产生的多个中断源中优先级最高的中断源进行处理；

2) 中断识别：识别中断源，将中断号送 CPU 并设置中断屏蔽字；

3) 获取中断服务程序地址：根据中断号查找中断服务程序的入口地址；

4) 中断响应：CPU 当前指令周期结束后，判断中断信号，进入中断隐指令周期，关中断，保存当前 PC，中断服务程序入口地址送 PC 寄存器；

5) 中断服务：保护现场，执行中断服务程序的指令；

6) 中断返回：恢复现场，恢复 PC 寄存器的值，开中断。

单级中断的流程如图 2.2 所示。

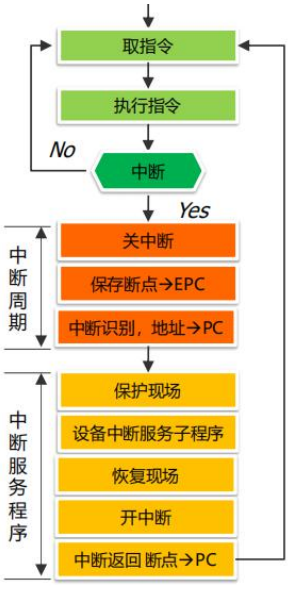

图 2.2 单级中断流程图

多级中断相对单级中断的区别在于执行中断服务程序时也能被其它优先级高的中断源中断，故在保护现场时，需要额外保存当前的中断屏蔽字，并在保护现场结束后开中断，执行中断服务程序，中断返回恢复现场时，需要先关中断，防止在恢复过程中被其他中断源中断，恢复完成后，开中断，将断点送入 PC。多级中断的流程如图 2.3 所示。

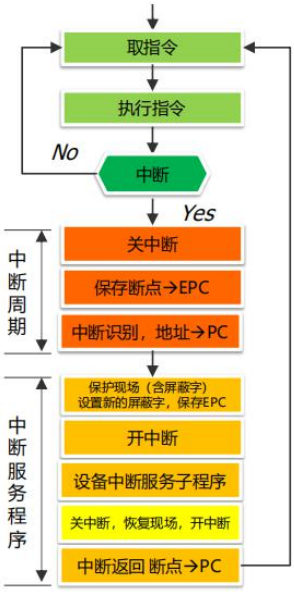

图 2.3　多级中断流程图

考虑到实际计算机系统中支持的中断源较多，为简便起见，只设置 3 个中断源，优先级递增，即 3>2>1>CPU，同级优先级按发起时间优先的方式处理，即中断服务程序不能被同级或低级中断源所中断。在只设置 3 个中断源的前提下，对 PC 的保存方式采用 EPC 硬件堆栈保存的方式，即使用硬件保存中断时的 PC 寄存器的值。中断源的优先级见表 2.5。

表 2.5 中断优先级表

中断源	3	2	1	CPU
3	×	√	√	√
2	×	×	√	√
1	×	×	×	√
CPU	×	×	×	×

关于中断流水线的相关设计，见 2.6。

2.2.2 硬件设计

中断源产生时，CPU 可能正在执行一条指令，需要等该指令执行完毕后再进行识别，故需要将中断源保存到中断请求寄存器 IR 中，不同中断源的 IR 寄存器输出端通过优先编码器进行中断仲裁，选出优先级最高的中断号。当 CPU 执行完当前指令后，检测中断信号是否有效，若是，则根据中断号取中断服务程序的 PC，保存当前 PC 到 EPC 寄存器，关闭中断使能信号，即置 IE 寄存器的值为 0。中断返回时，清除中断源，从 EPC 中取出送入 PC 寄存器，置 IE 为 1 开中断。相关功能部件如下：

1. 中断请求信号寄存器 IR

对每个中断源，均有 IR 寄存器保存请求信号，将中断信号变为同步信号，使 CPU 能执行完当前指令后再进行中断识别。

2. EPC 堆栈寄存器组

使用寄存器组实现 EPC 硬件堆栈，对单级中断，只有一个 EPC 寄存器，对设置了 3 个中断源的多级中断，级联 3 个 EPC 寄存器，通过控制信号模拟堆栈活动。

3. 中断使能寄存器 IE

寄存器值为 1 时表示开中断，此时 CPU 每执行完一条指令后检查是否有中断源并进行响应，值为 0 时表示关中断，CPU 不对中断源进行识别。

4. 中断识别电路逻辑

根据中断源的中断号选择其中断服务程序的 PC 值，本次课设中为方便起见，将 3 个中断服务程序的入口地址写死，故采用多路选择器将中断服务程序入口 PC 连接到输入端，将中断号作为选择端输入。

2.2.3 软件设计

采用 EPC 硬件堆栈方式实现 PC 值的保存，无需在 MIPS 汇编程序中编写保存 EPC 和 ESP 的代码，在执行中断服务代码前，需要先保护现场，即将中断服务程序

中需要修改的寄存器的值压栈，执行完中断服务代码后，再逆序从堆栈中取出相应寄存器的值恢复现场，最后调用 iret 通知 CPU 进行中断返回。

2.3 流水 CPU 设计

2.3.1 总体设计

根据 MIPS 指令执行的流程，可将指令执行分为五个阶段：取指令（IF）、指令译码（ID）、执行指令（EX）、访存（MEM）、写回（WB），IF 段主要根据 PC 寄存器的值，从指令寄存器 IM 中取出指令，ID 段主要对取出的指令进行指令译码，将 Func 字段和 OP 字段送入硬布线控制器产生控制信号，将 rs、rt 字段送入寄存器堆 RF 取操作数，EX 字段主要实现指令的功能，如算术运算、逻辑运算和执行分支指令（在后续流水线的设计中均是在 EX 段执行分支），MEM 段主要完成从数据寄存器 DM 中读出数据及将运算结果写入 DM 的工作，这将是流水线的瓶颈，WB 段主要将 EX 段的运算结果或 MEM 段取出的数据写回到寄存器堆 RF。

流水线结构图如图 2.4 所示，控制信号在 ID 段产生，直接复用硬布线控制器。

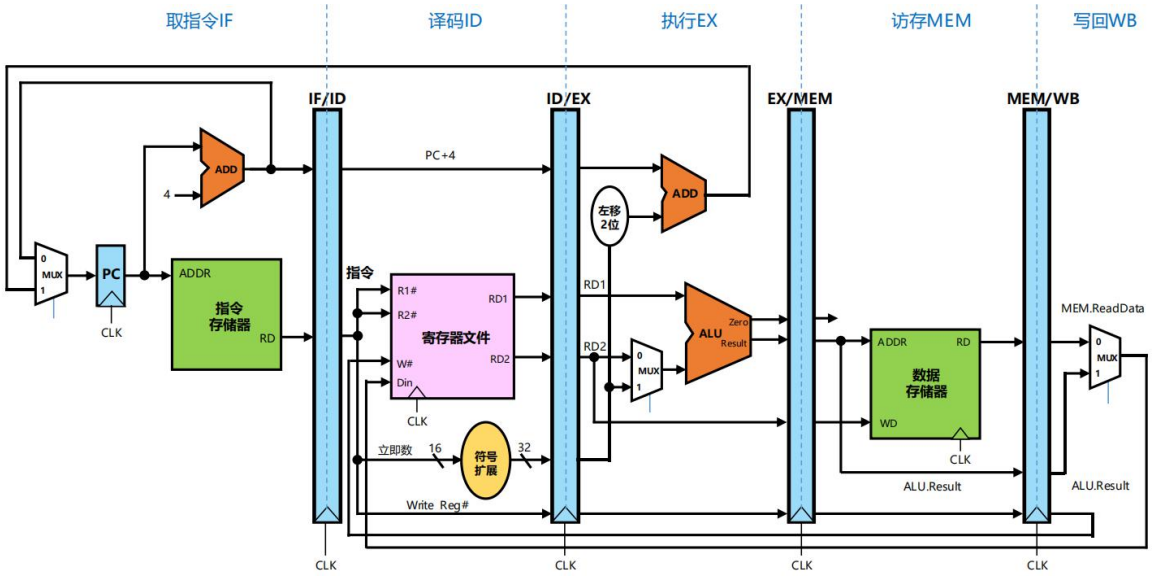

图 2.4 流水线结构图

2.3.2 流水接口部件设计

1. IF/ID 接口部件

该部件主要将 IF 段取出的指令和 PC+4 送入锁存器，供接下来的段使用。

2. ID/EX 接口部件

该部件将指令译码后产生的各类控制信号及从寄存器中取出的操作数或指令中的立即数送入锁存器，供接下来的段使用。

3. EM/MEM 接口部件

该部件将 ID/EX 接口部件输出的控制信号、ALU 的运算结果及 PC 值送入锁存器锁存，供接下来的段使用。

4. MEM/WB 接口部件

该部件主要将 EM/MEM 接口部件输出的控制信号、访存结果及 PC 值送入锁存器，供 WB 段使用。

2.3.3 理想流水线设计

在理想流水中，没有数据相关，没有分支指令，指令一直按顺序寻址方式进行，故流水线中除了写回数据外没有反馈回路，按图 2.4 原理构建流水接口部件和数据通路即可。需要说明的是 PC 寄存器输入端一直是 PC+4，不需要添加多路选择器选择输入源,当 WB 段停机信号 Syscall 有效时，PC 寄存器及流水接口部件使能端置为 0，实现流水暂停。

2.4 气泡式流水线设计

实际 MIPS 五段流水线中，会出现数据冲突和控制冲突，数据冲突由数据相关产生的，故须设计数据相关检测电路，通过 MIPS 指令中源寄存器使用编号和功能段中使用的寄存器编号即可进行数据相关检测。

在 ID 段指令译码时，即可获取源寄存器的使用情况，数据冲突分为 ID 段和 EX 段（源寄存器的数据还未计算出来）、ID 段和 MEM 段（源寄存器的数据还未读出或写入 DM）、ID 段和 WB 段（源寄存器的数据还未写回），其中 ID 段和 WB 段的冲突可以通过先写后读解决，即时钟周期下跳沿写回寄存器，时钟周期上升沿读取寄存器，其他冲突只能通过如下解决方案解决：

（1） ID 段和 EX 段：将 IF/ID 接口部件暂停，并暂停 PC 寄存器写入信号，

进而暂停 IF、ID 段的执行达到延缓取操作数的效果，同时 EX 段插入气泡，即设置同步清零信号，防止 ID 段的指令向后传送，下一个时钟周期到来时，转换为 ID 段和 MEM 段冲突。

（2） ID 段和 MEM 段：和（1）中相同处理方式，暂停 IF、ID 段执行，在 EX 段插入气泡，下一个时钟周期到来时，转换为 ID 段和 WB 段冲突。

（3） ID 段和 WB 段：采用先写后读方式解决冲突，此时 IF、ID 段启动，PC 寄存器使能置为 1。

对于控制冲突，在 EX 段执行分支的流水设计中，当 EX 段执行分支时，IF、ID 段按顺序寻址方式读入的指令必须清除，即在 IF、ID 段插入气泡，EX 段分支目标地址通过多路选择器反馈到 PC 寄存器的输入端。

2.5 数据转发流水线设计

为进一步提高气泡流水线性能，可在 ID 段不考虑数据冲突，在执行指令时将数据通过旁路技术重定向到使用的位置，这种流水线中可能出现的数据相关有 EX 段和 MEM 段的数据相关及 EX 段和 WB 段的数据相关。

具体来说，就是 EX 段送入 ALU 进行运算的两操作数可能不是最新值，其值在 MEM 段或 WB 段中还未来得及写回寄存器，故将 MEM 段或 WB 段未写回的值通过多路选择器输入到 ALU 的操作数端，多路选择器的控制信号通过设计组合逻辑电路产生，该电路接受 EM 段和 MEM 段的控制信号及源寄存器使用情况作为输入，输出多路选择器的选择信号，将该选择信号送入 ID/EX 接口部件，在 EX 段进行数据选择。

在 EX 段和 MEM 段的相关中，对于 MEM 段加载数据，EX 段使用数据的 Load-Use 相关，若采用上述方案，会造成 EX 段的时间延迟变为运算器延迟+访存延迟，在 2.3.1 中介绍过，MEM 段是整个流水线的瓶颈段，正是因为访存的时间开销很大，上述方案会使得 EX 段延迟大大增加，降低流水线的性能，故对 Load-Use 相关，可在 EX 段插入一个气泡，并暂停 ID、IF 段的执行，下一个时钟周期到来时，Load-Use 相关转换为 EX 段和 WB 段的相关。

2.6 中断流水线设计

为流水线添加支持中断的功能，为简便起见，添加支持单级中断，采用硬件堆

栈方式保存 PC。由于 MIPS 五段流水线中最多同时有 5 条指令在执行，中断响应的过程和中断返回过程实际上也是属于分支执行，故对中断源的识别考虑设置到 EX 段执行，中断仲裁及 EPC 硬件堆栈可直接复用单周期 CPU 中的中断设计方案，将中断信号送入流水接口部件传递，在 EX 段识别，执行如下步骤：

（1） MEM 段和 WB 段的指令继续执行。

（2） IF 段和 ID 段插入气泡，即取消这两个功能段中的指令。

（3） EX 段本身这条指令取消执行，并将 EX 段指令的 PC 值送入 EPC 寄存器保存。

（4） 中断服务程序入口 PC 送入 PC 寄存器，进行中断服务。

（5） 中断返回，将 EPC 中的值送 PC 寄存器。

2.7 动态分支预测机制设计

在数据转发流水线中，通过旁路技术消除了除 Load-Use 相关外的数据相关插入气泡造成的流水停顿，但对分支指令，仍然会插入两个气泡，使流水停顿。据统计，程序中的分支指令占比在20%左右，根据程序的局部性原理和加快经常性事件原理，可设计分支历史表 BHT 保存分支指令的分支目标地址及分支情况，对分支进行预测。BHT 采用全相联 Cache 方案，替换策略为 LRU 算法，保存分支指令的 PC、分支目标指令的 PC 及分支历史情况，采用两位分支预测历史，此外还有有效位 Valid、LRU 计数器，BHT 格式见表 2.6。

表 2.6 BHT 格式

Valid(1：有效)	分支指令地址	分支目标指令地址	分支预测历史(2 位)	LRU 计数器

其中，两位分支预测历史的状态转换如图 2.5 所示。

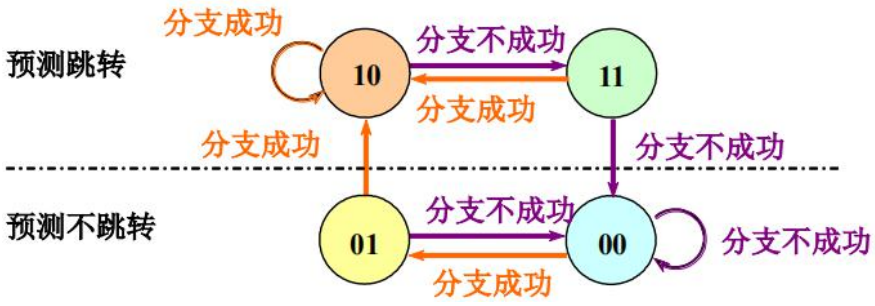

图 2.5 分支预测历史的状态转换

动态分支预测采用 IF 段进行预测，EX 段进行 BHT 写入及状态修改的策略。在 IF 段，将 PC 值送 BHT 进行相联查找，若命中，则根据分支预测历史预测分支是否成功，否则预测分支失败；在 EX 段，根据 PC 值判断预测是否正确，并修改 BHT 中相应表项，若 BHT 未命中，则进行写入（涉及 LRU 淘汰策略）。

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 为停机信号，当需要进行停机时，Halt 控制信号为 0，Go 信号为 1 时，PC 寄存器继续接受输入，如图 3.1 所示。

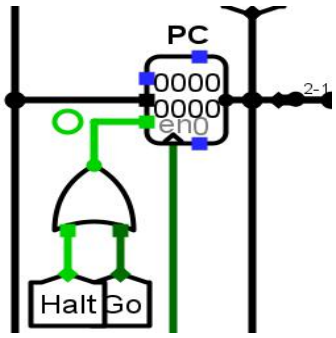

图 3.1 程序计数器（PC）

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) 寄存器堆 RF

使用 32 个 32 位寄存器阵列实现寄存器堆 RF，课设资源包中提供了封装好的寄存器堆 RF，输入端口有 Din、R1#、R2#、W#、WE，分别表示写入数据、寄存器 1 编号、寄存器 2 编号、写入寄存器编号、写使能信号，输出端口有 R1、R2，表示寄存器 1、2 的数据，如图 3.3 所示。

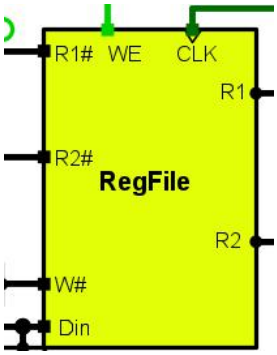

图 3.3 寄存器堆（RF）

4) 运算器 ALU

按表 2.3 中介绍的功能实现 32 位 ALU 部件，输入端有 X、Y、shamt、OP，分别表示两个操作数、指令中的移位字段、控制器产生的 ALUOP 字段，输出端口有 Result1、Result2 和 Equal，分别表示两操作数的运算结果和是否相等，其中 Result2 只输出乘除运算结果，实现的指令集中没有乘除运算，故该接口实际未使用，如图 3.4 所示。

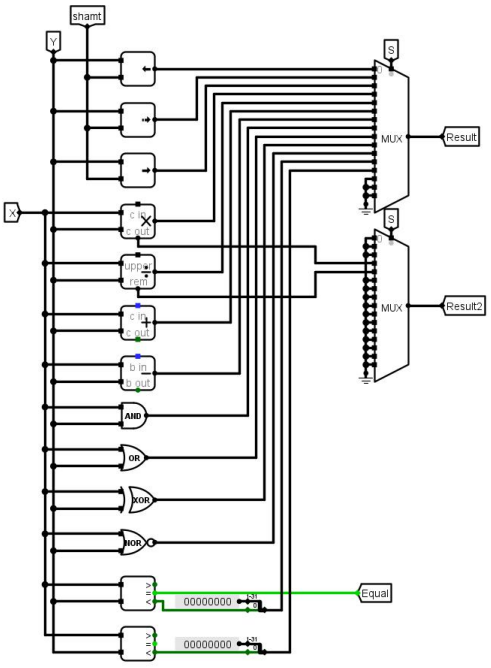

图 3.4 运算器（ALU）

5) 数据存储器 DM

使用一个随机存储器 RAM 实现数据存储器（DM）。该随机存储器的地址位宽和数据位宽与 IM 中相同，输入端有 Addr、Din、str，分别表示 10 位地址、写入数据、存储信号，输出端有 Dout，为 DM 中读出的数据，如图 3.5 所示。

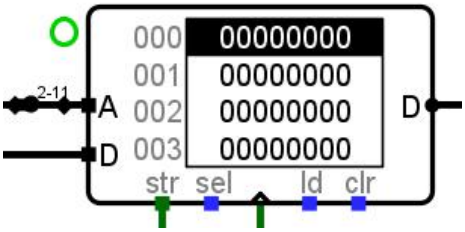

图 3.5 数据存储器（DM）

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

类型	指令	PC	IM	RF				ALU			DM	
				R1#	R2#	W#	Din	A	B	OP	Addr	Din
24 条基本指令	SLL	PC+4	PC		rt	rd	alu	R2	立即数	0		
	SRA	PC+4	PC		rt	rd	alu	R2	立即数	1		
	SRL	PC+4	PC		rt	rd	alu	R2	立即数	2		
	ADD	PC+4	PC	rs	rt	rd	alu	R1	R2	5		
	ADDU	PC+4	PC	rs	rt	rd	alu	R1	R2	5		
	SUB	PC+4	PC	rs	rt	rd	alu	R1	R2	6		
	AND	PC+4	PC	rs	rt	rd	alu	R1	R2	7		
	OR	PC+4	PC	rs	rt	rd	alu	R1	R2	8		

类型	指令	PC	IM	RF				ALU			DM	
				R1#	R2#	W#	Din	A	B	OP	Addr	Din
24 条基本指令	NOR	PC+4	PC	rs	rt	rd	alu	R1	R2	10		
	SLT	PC+4	PC	rs	rt	rd	alu	R1	R2	11		
	SLTU	PC+4	PC	rs	rt	rd	alu	R1	R2	12		
	JR	R1	PC	rs								
	SYSCALL	PC+4	PC	0x2	0x4							
	J	$PC+4_{31..28}\|\|IR_{25..0}\|\|00$	PC									
	JAL	$PC+4_{31..28}\|\|IR_{25..0}\|\|00$	PC			0x31	PC+4					
	BEQ	PC+4+Ext(Imm16)<<2	PC	rs	rt			R1	R2			
	BNE	PC+4+Ext(Imm16)<<2	PC	rs	rt			R1	R2			
	ADDI	PC+4	PC	rs		rt	alu	R1	立即数	5		
	ANDI	PC+4	PC	rs		rt	alu	R1	立即数	7		
	ADDIU	PC+4	PC	rs		rt	alu	R1	立即数	5		
	SLTI	PC+4	PC	rs		rt	alu	R1	立即数	11		
	ORI	PC+4	PC	rs		rt	alu	R1	立即数	8		
	LW	PC+4	PC	rs		rt	MDout	R1	立即数	5	alu	
	SW	PC+4	PC	rs		rt		R1	立即数	5	alu	R2
CCMB	SRAV	PC+4	PC	rs	rt	rd	alu	R1	R2	1		
	XOR	PC+4	PC	rs	rt	rd	alu	R1	R2	9		
	LH	PC+4	PC	rs		rt	MDout 半字符号拓展	R1	立即数	5	alu	
	BLEZ	PC+4+Ext(Imm16)<<2	PC	rs	rt			R1	R2			

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，将各个主要功能部件进行连接，根据数据通路合并的最终结果，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。

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 上进行硬布线控制器的具体实现，根据表 2.4 的设计，对 24 条基本指令和 4 条 CCMB 拓展指令填入控制信号，最后使用 Logisim 分析电路的自动生成功能即可构建硬布线控制器，如表 3.2 所示。（未填表示无关）

表 3.2 主控制器控制信号

指令	RD	RW	ALUSrc	MW	MTR	ALUOP	BNE	BEQ	BLEZ	J	JR	JAL	SRAV	LH	SC	ERET	SE
SLL	1	1	0	0	0	0000	0	0	0	0	0	0	0	0	0	0	0
SRA	1	1	0	0	0	0001	0	0	0	0	0	0	0	0	0	0	0
SRL	1	1	0	0	0	0010	0	0	0	0	0	0	0	0	0	0	0
ADD	1	1	0	0	0	0101	0	0	0	0	0	0	0	0	0	0	0
ADDU	1	1	0	0	0	0101	0	0	0	0	0	0	0	0	0	0	0
SUB	1	1	0	0	0	0110	0	0	0	0	0	0	0	0	0	0	0
AND	1	1	0	0	0	0111	0	0	0	0	0	0	0	0	0	0	0
OR	1	1	0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0
NOR	1	1	0	0	0	1010	0	0	0	0	0	0	0	0	0	0	0
SLT	1	1	0	0	0	1011	0	0	0	0	0	0	0	0	0	0	0
SLTU	1	1	0	0	0	1100	0	0	0	0	0	0	0	0	0	0	0
JR	0	0	0	0	0		0	0	0	1	1	0	0	0	0	0	0
SYSCALL	0	0	0	0	0		0	0	0	0	0	0	0	0	1	0	0
J	0	0	0	0	0		0	0	0	1	0	0	0	0	0	0	0
JAL	0	1	0	0	0		0	0	0	1	0	1	0	0	0	0	0
BEQ	0	0	0	0	0		0	1	0	0	0	0	0	0	0	0	1
BNE	0	0	0	0	0		1	0	0	0	0	0	0	0	0	0	1
ADDI	0	1	1	0	0	0101	0	0	0	0	0	0	0	0	0	0	1
ANDI	0	1	1	0	0	0111	0	0	0	0	0	0	0	0	0	0	0
ADDIU	0	1	1	0	0	0101	0	0	0	0	0	0	0	0	0	0	1

SLTI	0	1	1	0	0	1011	0	0	0	0	0	0	0	0	0	0	1
ORI	0	1	1	0	0	1000	0	0	0	0	0	0	0	0	0	0	0
LW	0	1	1	0	1	0101	0	0	0	0	0	0	0	0	0	0	1
SW	0	0	1	1	0	0101	0	0	0	0	0	0	0	0	0	0	1
SRAV	1	1	0	0	0	0001	0	0	0	0	0	0	1	0	0	0	0
XOR	1	1	0	0	0	1001	0	0	0	0	0	0	0	0	0	0	0
LH	0	1	1	0	1	0101	0	0	0	0	0	0	0	1	0	0	1
BLEZ	0	0	0	0	0		0	0	1	0	0	0	0	0	0	0	1
ERET	0	0	0	0	0		0	0	0	0	0	0	0	0	0	1	0

完成硬布线控制器的控制信号设计后，在 Logism 中进行电路自动生成，将控制信号连接到数据通路的控制信号端，多输入端用多路选择器连接选择输入，最终实现 Logsim 中的电路如所图 3.6 示。

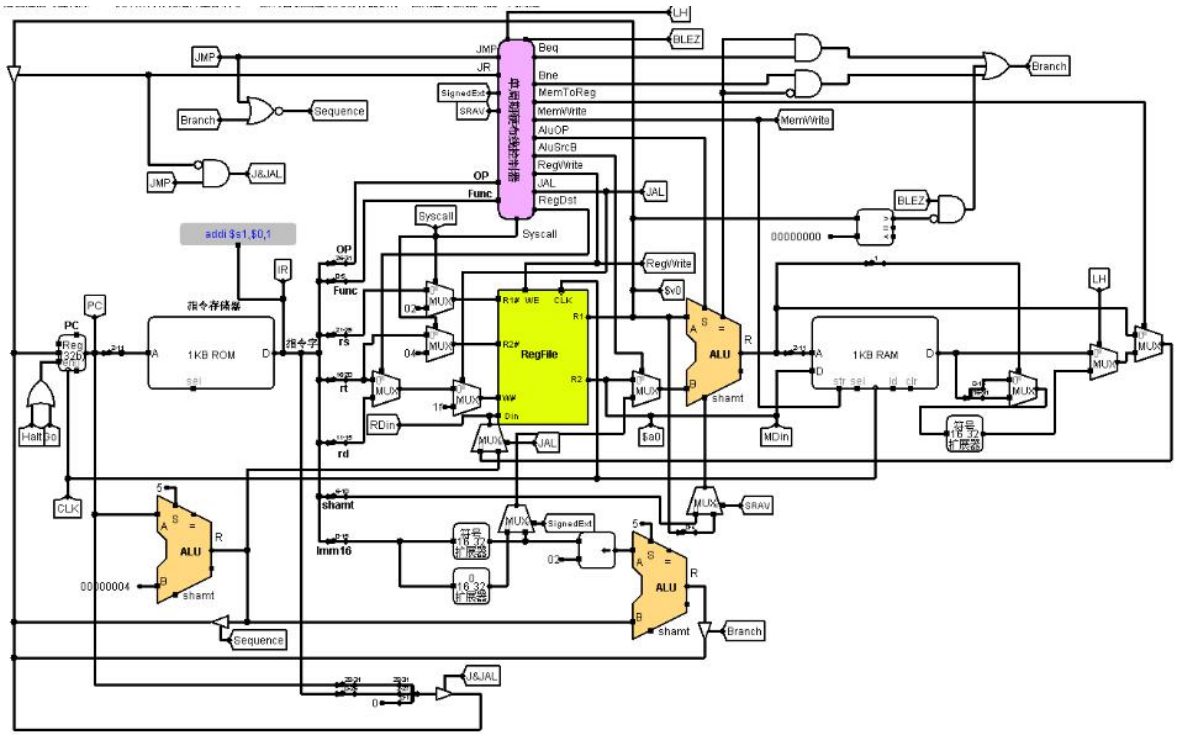

图 3.6 单周期 CPU 电路

3.2 中断机制实现

3.2.1 单级中断

按 2.2 节的设计思想，设计中断仲裁电路如图 3.7 所示，中断源产生后，经过

优先编码器输出中断号（优先级 3>2>1>CPU）。

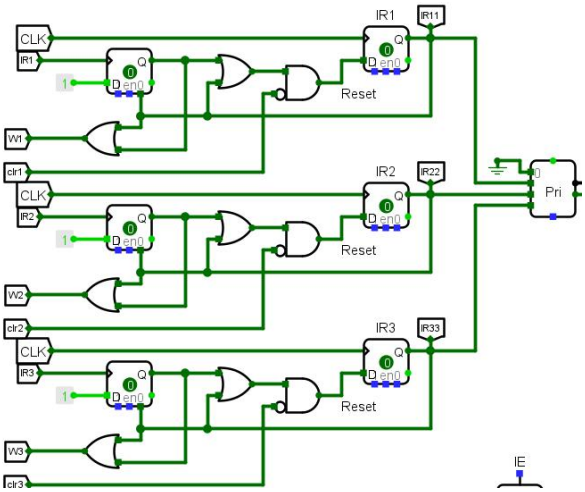

图 3.7　中断仲裁电路

设计中断使能寄存器 IE，其值为 0 表示关中断，为 1 表示开中断，初始值为 1，当有中断时置为 0，中断返回后置为 1，设计 EPC 硬件堆栈保存中断返回地址，中断信号有效时将 PC+4 写入该寄存器，EPC 和 IE 的电路实现如图 3.8 所示。

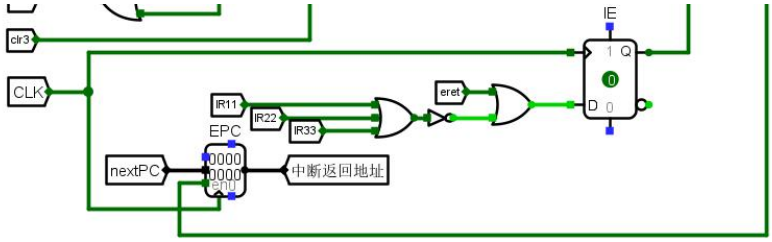

图 3.8　IE 和 EPC 电路

中断进入的入口地址通过中断号由多路选择器选择输入到 PC 的输入端，入口地址是由中断测试程序翻译为机器码后人工添加，如图 3.9 所示。

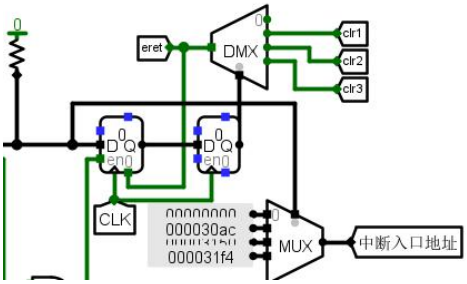

图 3.9　中断入口地址选择电路

最后，在图 3.6 实现的电路基础上，在 PC 的输入端添加中断入口地址和中断返回地址输入端，通过三态门连接到 PC 总线上，三态门的控制由中断信号和中断返回信号进行，如图 3.10 所示。

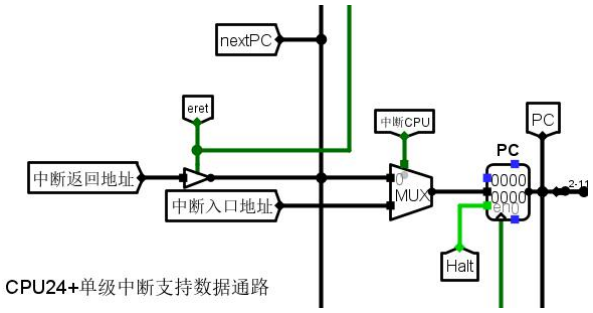

图 3.10 PC 输入端电路

3.2.2 多级中断

按 2.2 节的设计，在单级中断的基础上，对 EPC 硬件堆栈进行改造，使之能够保存最多 3 个断点处的 PC，同时对中断使能寄存器 IE 进行改造，使之能够在完成断点保存后就开中断，在中断返回时再次关闭中断，恢复断点后再次开中断。中断使能寄存器 IE 的电路如图 3.11 所示。

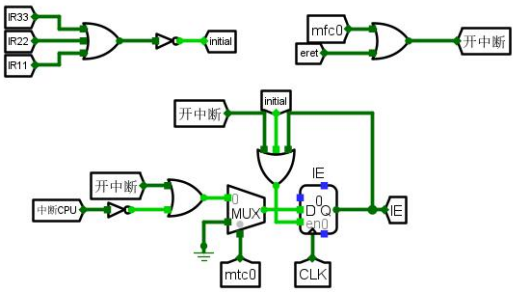

图 3.11 多级中断使能寄存器 IE

其中，用到了中断相关的 mfc0 和 mtc0 指令，mfc0 用于开中断，mtc0 用于关中断，即在完成断点保存后即可使用 mfc0 开中断，在中断返回恢复断点时使用 mtc0 关中断。EPC 硬件堆栈采用首位相连的三个寄存器构成循环堆栈，当更高优先级的中断源到来时，即可保存断点，此外由于中断返回时需要根据中断号产生信号，故还要将中断号随之保存，EPC 硬件堆栈电路如图 3.12 所示。

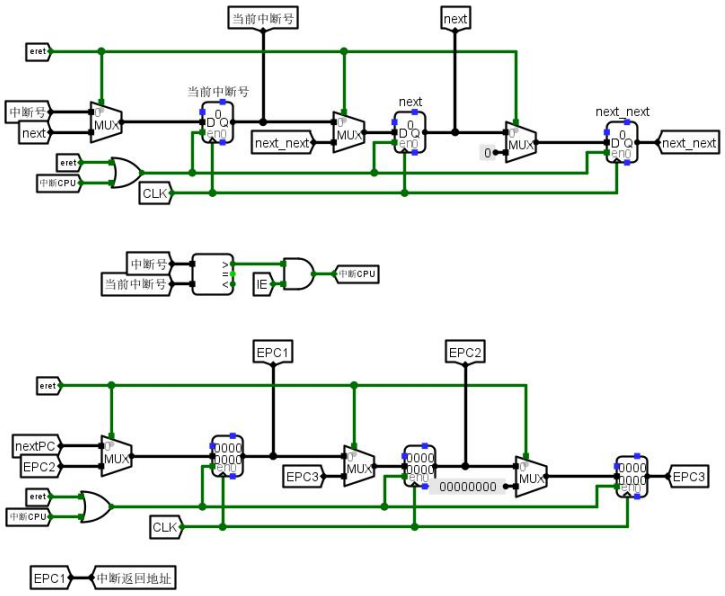

图 3.12　多集中断 EPC 硬件堆栈

多级中断其余电路与单级中断相同，PC 输入总线的修改见图 3.10。

3.3　流水 CPU 实现

3.3.1　流水接口部件实现

根据 122.3.2 节中的五段流水设计，流水接口部件使用寄存器锁存上个功能段的各控制信号和下个功能段需要用到的数据，作为两个功能段间的连接，以最复杂的 ID/EX 接口部件为例，设计电路如图 3.13 所示，所以存储器件的时钟信号段统一连接，通过多路选择器 1 端接地，选择端为清零信号实现同步清零。

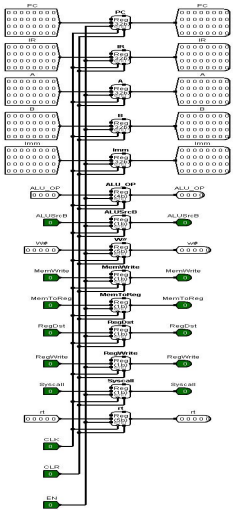

图 3.13　流水接口部件实现

3.3.2 理想流水线实现

理想流水线中没有分支指令和数据相关，故直接按功能段划分，将 3.3.1 节中实现的流水接口部件进行封装，在各功能段直接插入部件即可。经此改造后的流水线电路如图 3.14 所示。

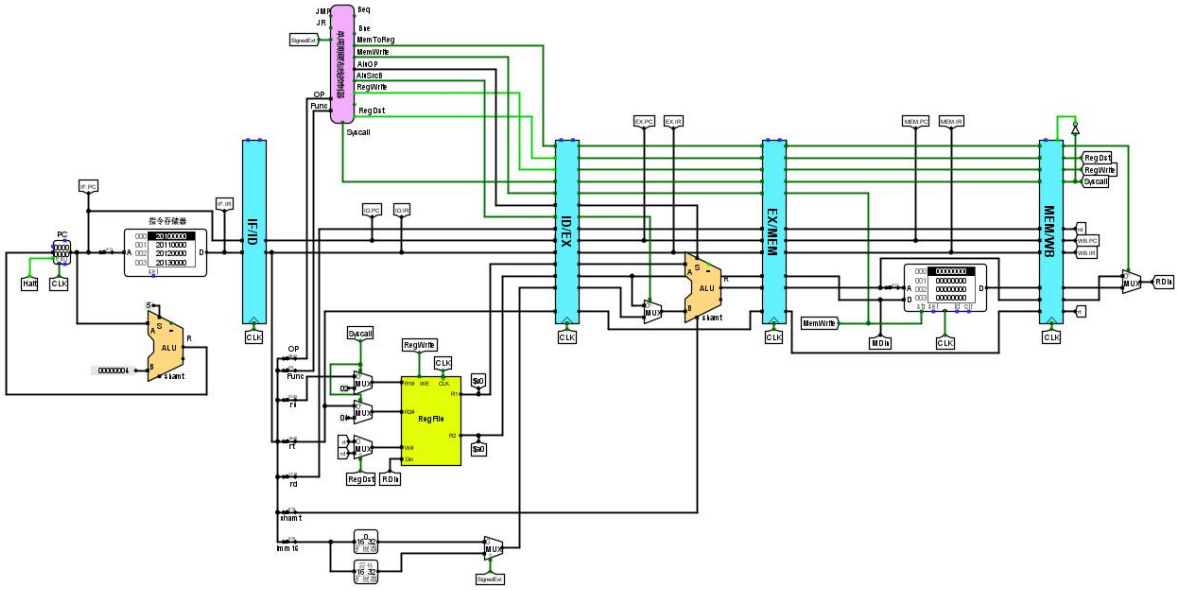

图 3.14 理想流水线电路图

3.4 气泡式流水线实现

按 2.4 节的设计，需要实现寄存器使用情况检测电路及数据相关检测电路，寄存器使用情况可以通过逐条指令分析得到，按照控制器电路的自动生成方法，填写寄存器使用情况表表 3.3。

表 3.3 寄存器使用情况表

指令	R1_USED	R2_USED	指令	R1_USED	R2_USED	指令	R1_USED	R2_USED
SLL	0	1	JAL	0	0	ORI	1	0
SRA	0	1	BEQ	1	1	LW	1	0
SRL	0	1	BNE	1	1	SW	1	1
ADD	1	1	ADDI	1	0	SRAV	1	1
ADDU	1	1	ANDI	1	0	XOR	1	1
SUB	1	1	ADDIU	1	0	LH	1	0
AND	1	1	SLTI	1	0	BLEZ	1	0

指令	R1_USED	R2_USED	指令	R1_USED	R2_USED	指令	R1_USED	R2_USED
OR	1	1	SLT	1	1	JR	1	0
NOR	1	1	SLTU	1	1	J	0	0
SYSCALL	1	1						

完成此表后，即可通过 Logisim 中的自动生成功能产生寄存器使用情况检测电路，然后将使用情况作为数据相关检测的输入端，根据此前的分析，在 ID 段检测出的数据冲突包括 ID 段和 EX 段、ID 段和 MEM 段，将 EX 段和 MEM 段的寄存器写使能有效，且写入寄存器的编号与 ID 段使用的编号相同时产生数据冲突，注意 0 号寄存器恒 0，需要排除在外，设计数据相关检测电路如图 3.15 所示。

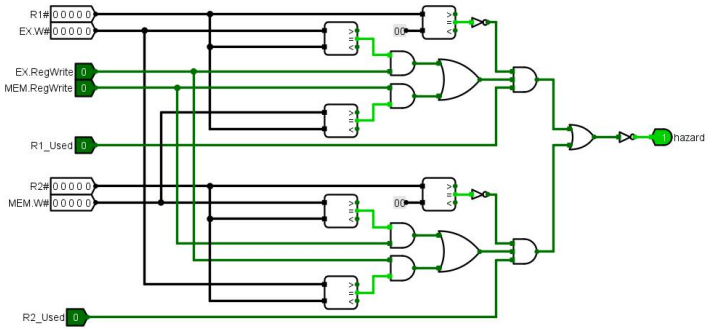

图 3.15 数据相关检测电路

将以上两个电路进行封装，对理想流水的接口部件进行适当改造，分支时在前两个功能部件中插入气泡，数据相关时 IF/ID 部件禁用，ID/EX 部件插入气泡，实现的气泡流水如图 3.16 所示。

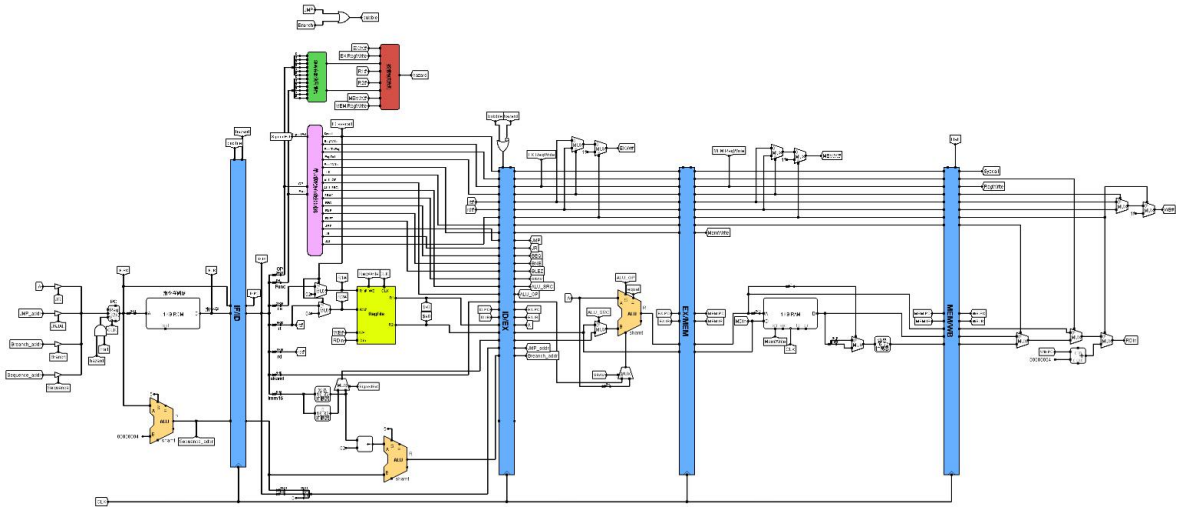

图 3.16 气泡流水线电路图

3.5 数据转发流水线实现

按 2.5 节中的数据转发流水设计，需要数据重定向信号的检测，数据相关在 ID 段检测出，对 EX 段和 MEM 段，分别判断其寄存器写信号 RegWrite、访存信号 MemToReg 及相应的寄存器编号是否与 ID 段指令的寄存器使用一致，若一致则产生重定向信号，将 EM 段或 MEM 段获得的数据在 EX 段进行重定向。为此，对气泡流水线进行适当改造，在 ID/EX 段增加重定向信号接口，EX 段 ALU 的输入源的由重定向信号通过多路选择器选择，如图 3.17 所示。

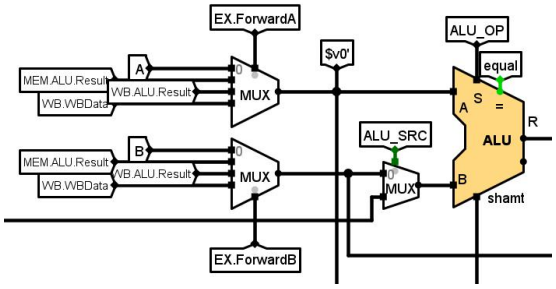

图 3.17 EX 段 ALU 器件输入重定向

在重定向选择信号产生电路中，检测 R1 和 R2 的数据是否在 EM 段或 MEM 段计算或访存得到，若是则产生相应的选择信号，选择信号通过多路选择器选出，此外需要识别出 load_use 相关，即 EX 段或 MEM 的 MemToReg 有效且 ID 段用到该数据，此时禁用 IF/ID 接口部件，并在 ID/EX 接口部件插入一个气泡消除该相关。电路如图 3.18 所示。

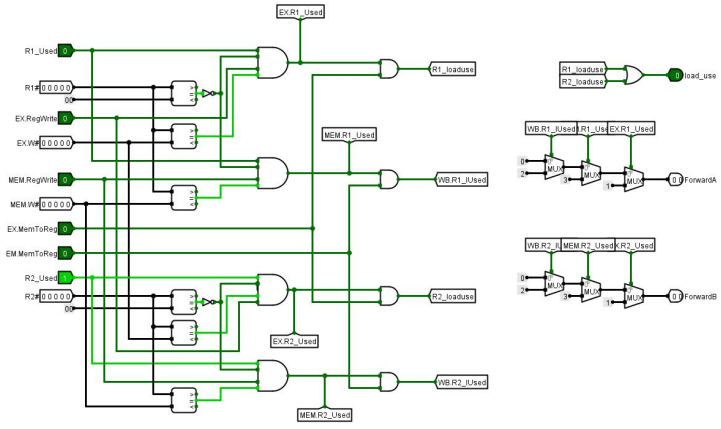

图 3.18 重定向信号产生电路

3.6 中断流水线实现

按 2.6 节的设计，中断相关电路直接复用了单周期 CPU 的单级中断电路，主要对流水线进行改造。在 EX 段进行中断响应，即 EX 段检测是否有中断，因此产生

的中断信号也需要送入流水传输，故流水接口部件需要增加中断信号，需要对中断断点进行判断，在数据转发流水线中，最多连续出现两个气泡，故 EX 段执行中断时，IF 段、ID 段和 EX 段的 PC 可能有以下三种情况：

（1） EX 段不为气泡，此时断点为 EX 段的 PC+4；

（2） EX 段为气泡，ID 段不为气泡，此时断点为 ID 段的 PC+4；

（3） EX 段、ID 段为气泡，此时断点为 IF 段的 PC+4。

设计断点判断电路如图 3.19 所示。

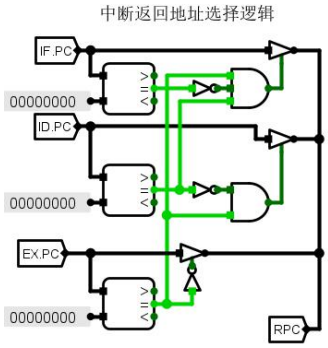

图 3.19 中断流水线断点检测电路（EX 段执行中断）

为实现 EX 段执行中断时取消预取指令的执行，以及中断返回时取消预取指令的执行，对数据转发流水线的 IF/ID 接口部件和 ID/EX 接口部件，增加同步清零信号中断和中断返回 eret，如图 3.20 所示。

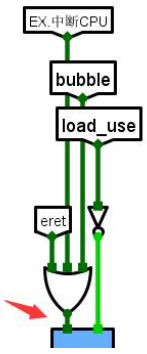

图 3.20 流水接口部件清零信号

3.7 动态分支预测机制实现

根据 2.7 节的设计，在数据转发流水线的基础上，改造流水线，添加分支历史表 BHT。BHT 采用全相联 Cache 实现，使用 LRU 淘汰算法，Cache 槽如图 3.21 所示，其中 FSM 是状态机，按图 2.5 的分支转换思想实现，如图 3.22 所示。

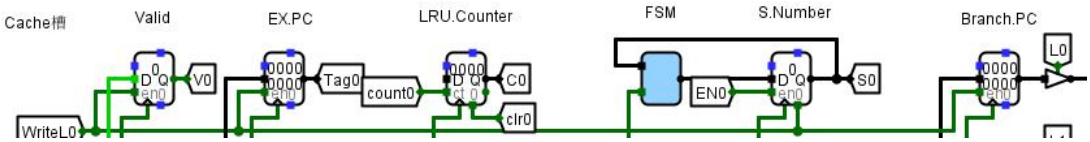

图 3.21 Cache 槽设计

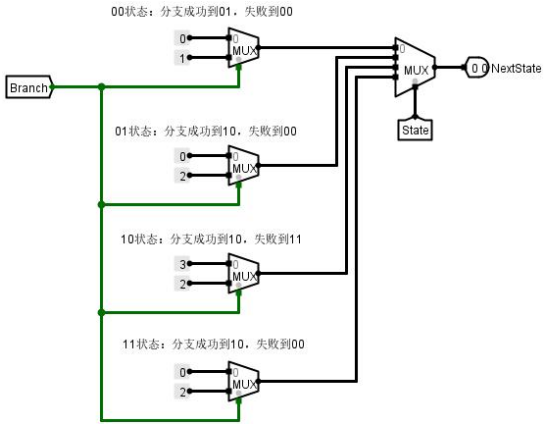

图 3.22 FSM 电路设计

LRU 淘汰计数信号 count 是对 Valid 为 1 的 Cache 槽进行计数，每次访问 BHT 命中时进行计数，BHT 的空行选择逻辑和无空行淘汰逻辑通过并发比较 8 个 Cache 槽的计数值及 Valid 位实现，若全部 Valid 均为 1，则选择计数值最大的淘汰，否则选择 Valid 为 0 的写入，如图 3.23 所示。

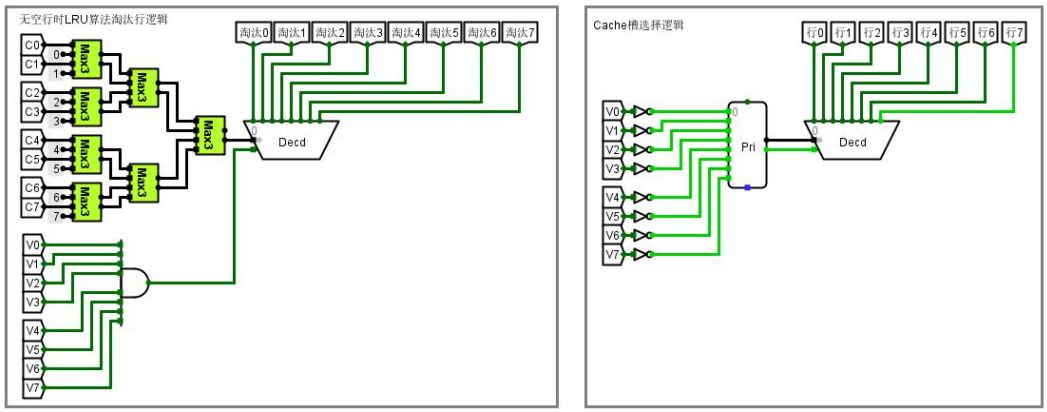

图 3.23 Cache 槽选择逻辑电路

BHT 在流水线中的工作原理是在 IF 段将 PC 值送入进行查找，若命中则根据当前状态进行预测，并将预测结果送入 PC 的输入端，若缺失则默认预测分支失败，将 PC+4 送 PC 输入端，在 EX 段判断预测是否正确，即若 EX 段的 PC 值在 BHT 表里，且 EX 段的下一个 PC 值与 BHT 表中记录一直，则表示预测正确，否则预测错误，然后更新状态，缺失进行淘汰替换。动态分支预测流程如图 3.24 所示。

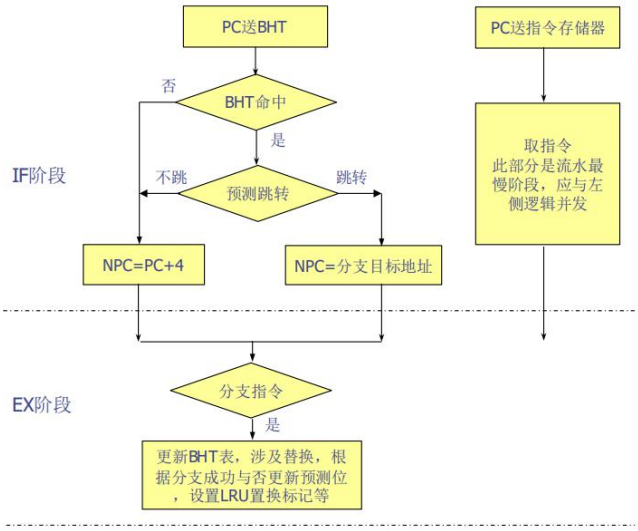

图 3.24 动态分支预测流程图

使用 BHT 后，PC 输入端的值将由 BHT 决定，设计下一个 PC 值的决定电路如图 3.25 所示，即若当前状态是预测分支成功，则由 BHT 中保存的分支目标地址决定，否则预测分支失败，下个 PC 为 PC+4，若 BHT 缺失则为 PC+4。

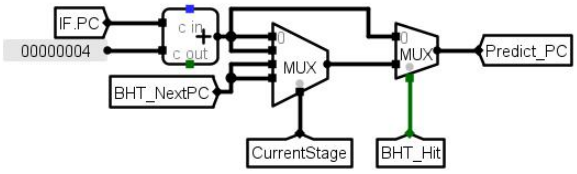

图 3.25 下个 PC 值的决定电路

在 EX 段，根据分支指令是否执行分支，以及 ID 段的 PC 值，即可判断是否预测正确，若预测错误，则需要在 IF 段和 ID 段插入气泡取消掉预取的指令，同时，更新 BHT 表，包括添加数据和替换数据，添加和替换的选择逻辑电路此前已实现，关于 PC 输入端的电路，如图 3.26 所示。

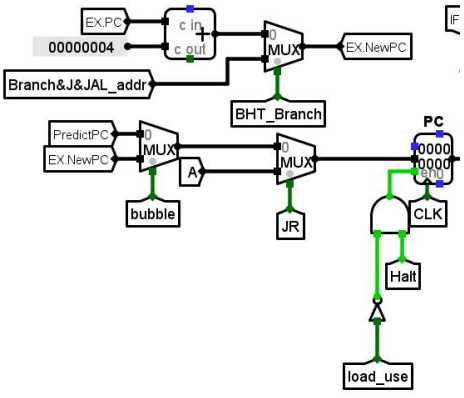

图 3.26 动态分支预测 PC 输入端电路

其中，Bubble 为气泡信号，即预测错误后插入气泡，此时的 PC 值由 EX 段的下个 PC 值决定，这个值要么是 PC+4，要么是分支目标指令的 PC 值，若 Bubble 值为 0，则由 predict PC 决定，即 BHT 给出的分支预测。

在数据转发流水线中加入 BHT 及相关逻辑后，电路如图 3.27 所示。

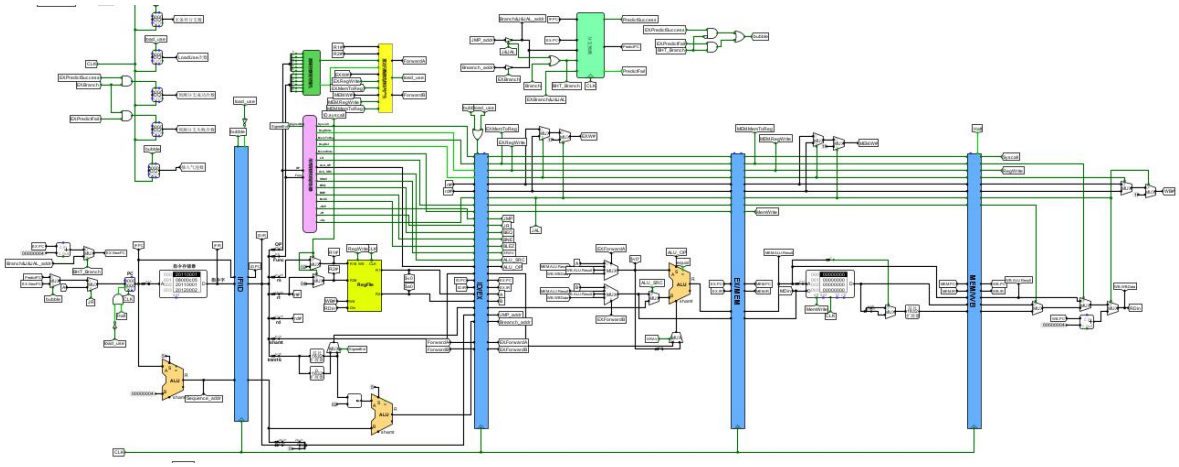

图 3.27　动态分支预测流水线

4 实验过程与调试

4.1 测试用例和功能测试

在课设过程中，首先使用单条指令的测试 MIPS 汇编代码进行逐条指令测试，完成后使用 benchmark 基准测试程序进行 24 条指令的联调。benchmark 指令执行后，会在数据存储器 DM 中写入 15~-1 的数据。对于中断 CPU 的设计，用中断测试主程序进行测试，1~3 号中断源对应不同的跑马灯。

4.1.1 单周期 24 指令测试结果

周期数为 1545（最后的停机指令未计入周期数），满足要求，数据存储器 DM 数据如图 4.1 所示，符合预期。

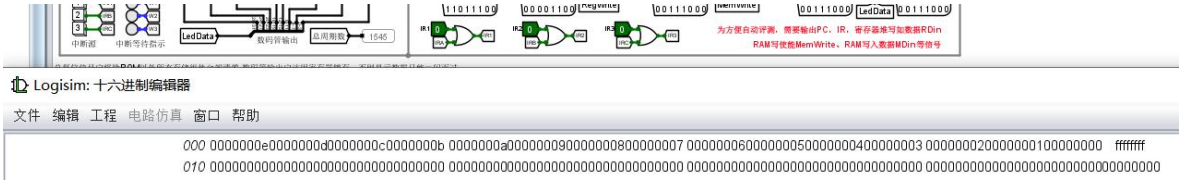

图 4.1 单周期 CPU 的 benchmark 测试结果

4.1.2 单周期单级中断测试结果

程序一开始是以跑马灯的形式在运行，如图 4.2 所示。

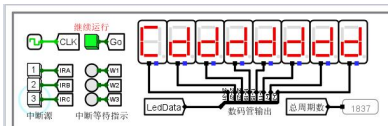

图 4.2 中断程序显示跑马灯

点击 2 号按钮后，再点击 3 号、1 号按钮，由于是单级中断，故首先进入 2 号中断服务程序，如图 4.3 所示。

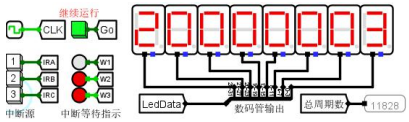

图 4.3 进入 2 号中断服务程序

2 号中断服务程序结束后，由于 3 号中断优先级最高，故先进入 3 号中断服务程序，如图 4.4 所示。

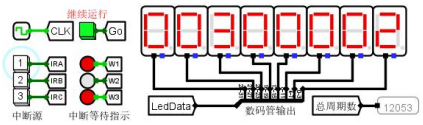

图 4.4 进入 3 号中断服务程序

3 号中断服务程序执行完毕后，最近进行 1 号中断的服务程序，最后，程序返回图 4.2 的断点处继续执行，如图 4.5 所示。

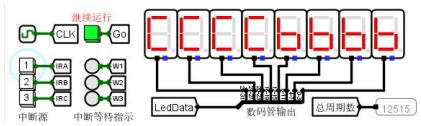

图 4.5 返回中断断点

4.1.3 单周期多级中断测试结果

先点击 1 号中断进入 1 号中断服务程序，如图 4.6 所示。

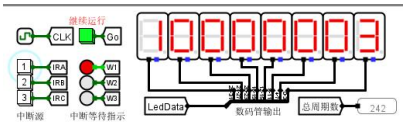

图 4.6 多级中断进入 1 号中断服务程序

然后点击 2 号中断，中断 1 号中断服务程序，进入 2 号中断服务程序，如图 4.7 所示。

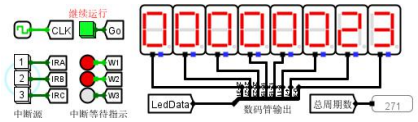

图 4.7 进入 2 号中断服务程序

2 号中断服务程序返回后，恢复的断点，继续执行 1 号中断服务程序，如图 4.8 所示。

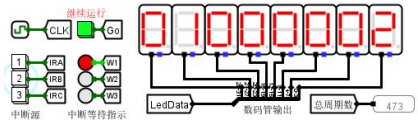

图 4.8 返回 1 号中断服务程序

4.1.4 理想流水测试结果

理想流水测试程序无分支指令和数据相关，执行完毕后，数据存储器 DM 中的前四个位置按序存入数据 0、1、2、3，时钟周期为 20（最后一条停机指令未计入），结果如图 4.9 所示。

图 4.9 理想流水测试结果

4.1.5 气泡流水测试结果

在气泡流水上运行 benchmark 测试程序，结果如图 4.10 所示，周期数为 3623。

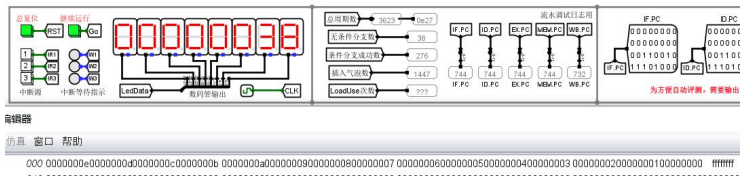

图 4.10 气泡流水测试结果

4.1.6 数据转发流水测试结果

在数据转发流水上运行 benchmark 测试程序，结果如图 4.11 所示，周期数为 2297。

图 4.11 数据转发流水测试结果

关于中断流水、CCMB 拓展指令、动态分支预测的测试，在随课设提交的演示视频中进行。

4.2 性能分析

资料给出 benchmark 测试程序共需执行 1546 次指令，其中无条件分支指令 38 次，条件分支成功 276 次，数据相关引入气泡数为 1447，load_use 相关数 120 次。

对单周期 CPU、气泡流水、数据转发流水的性能分析见表 4.1。

表 4.1 流水性能分析

	理想周期数	计算过程	实际周期数
单周期	1546	单周期CPU每个时钟周期执行1条指令	1545
气泡流水	3624	1546+38*2+276*2+1447-1+4	3623
数据转发流水	2298	1546+38*2+276*2+120+4=2298	2297

实际执行的时钟周期符合预期（没有统计最后一条停机指令的时钟周期），虽然流水的时钟周期数比单周期多，但单周期 CPU 中每条指令都要经过一个完整的 ALU+访存延迟，而流水细分功能段，每条指令只需经过其流经功能部件的延迟，故实际上流水运行的时间相较于单周期更快。

4.3 主要故障与调试

4.3.1 单周期 J 和 JAL 指令故障

单周期 CPU： J 指令和 JAL 指令分支地址计算问题。

故障现象：J 指令和 JAL 指令不能正确转移到目标位置。

原因分析：在计算 J 指令和 JAL 指令的分支目标地址时，由于对指令不熟悉，错误地设置成了$[PC]_{31..2} \| 00$，而实际的分支目标地址是$[PC]_{31..28} \| [IR]_{27..2} \| 00$。

解决方案：修改 J 指令和 JAL 指令的分支目标地址计算电路。使用分线器进行合适的连接，如图 4.12 所示。

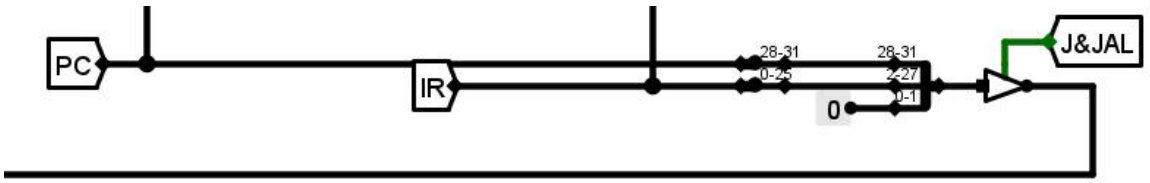

图 4.12 J 指令和 JAL 指令的分支目标地址计算

4.3.2 中断故障

单周期 CPU：中断返回的地址不对。

故障现象：在中断服务程序执行 eret 指令后，返回的位置不是检测到中断时的下一条指令，而是进入中断前执行的最后一条指令，导致 CPU 重复执行了一条指令，跑马灯显示出现错误。

原因分析：在使用 EPC 硬件堆栈保存断点时，错误地将当前 PC 进行保存，应该保存 PC+4。

解决方案：修改 EPC 接入端为 PC+4。

4.3.3 流水停机故障 1

理想流水：执行停机指令时未按预期暂停。

故障现象：在运行理想流水测试程序时，执行最后一条停机指令后，PC 值仍在不断增加，不停读取 IM 中的 nop 指令执行。

原因分析：在 WB 段执行停机指令前，PC 寄存器使能端均有效，使得读取 IM 中停机指令后的 nop 指令送入流水，WB 段执行停机指令后，PC 寄存器虽然锁定，但流水接口部件使能端悬空，导致在执行停机指令后流水接口部件任然有效，会不停的读取当前 PC 指向的 nop 指令。

解决方案：对流水接口部件的使能端进行修改，当执行停机指令后，使能端为低电平，如图 4.13 所示。

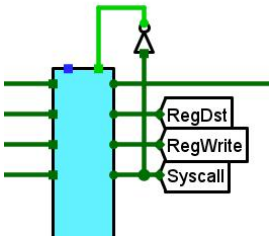

图 4.13 流水接口部件使能端

4.3.4 气泡流水故障

气泡流水：数据相关检测异常。

故障现象：气泡流水运行 benchmark 测试程序后，显示结果与存储器内排序结果正确，但时钟周期数大于 3624。

原因分析：通过 educoder 平台导出实际时空图与标准时空图的比对结果发现，在执行 syscall 指令进行 LED 显示时，错误地插入了气泡，原因是在 ID 端访问寄存器取 R1、R2 寄存器的编号时，多路选择器的选择端输入用成了 WB 段的 syscall 控制信号。

解决方案：将 R1、R2 寄存器编号的多路选择器使能端修改为 ID 段的 syscall 控制信号，如图 4.14 所示。

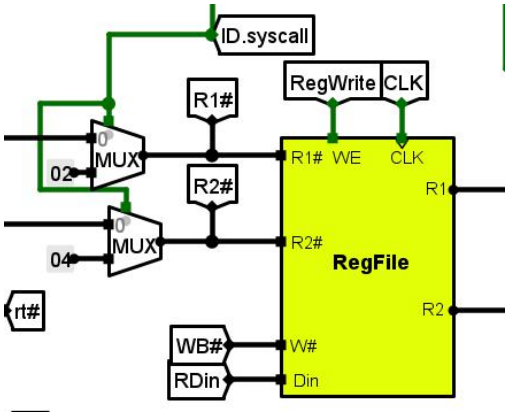

图 4.14 ID 端 R1、R2 寄存器编号

4.3.5 流水停机故障 2

数据转发流水：未能按预期停机。

故障现象：电路在执行停机指令后未按预期停机，但能通过 educoder 平台的测试。

原因分析：由于数据转发流水是在气泡流水的基础上添加数据重定向技术实现的，而气泡流水中在 WB 段检测是否为停机信号是通过将$v0 寄存器的值在 ID 段送入流水实现的，数据相关采用的是插入气泡同步等待的策略，因此这种方案不在气泡流水中不会有问题，而在数据转发流水中，出现数据相关时采取的是数据重定向策略解决，因此按照气泡流水的停机逻辑会使得 WB 端进行停机判断时使用的$v0 的值并非 EX 段真正修改的值。

解决方案：$v0 寄存器的值不从 ID 段送入流水，而是从 EX 段重定向后再送入流水，这样在 WB 段获取的$v0 寄存器的值是重定向后的正确数据。

4.3.6 中断流水故障

中断流水：未能按预期放弃 EX 段指令的执行。

故障现象：EX 段检测到中断信号后，IF、ID 段指令通过清零信号成功取消，而 EX 段指令却未能取消，流入 MEM 段。

原因分析：按设计在 EX 段执行中断，取消掉预取的两条指令，同时取消掉 EX 段本身这条指令，将 EX 段的中断信号作为 IF\ID、ID\EX 流水接口部件的同步清零信号可以成功的清理掉预取指令，但由于同步清零信号是在下个时钟周期到来时清零，所以 EX 段本身这条指令会送入 MEM 段，导致没有取消掉该指令。

解决方案：$使用异步清零信号，将中断信号送入 MEM 段，下个时钟周期到来时，EX 段指令流水到 MEM 段，此时 MEM 段中断信号有效，则通过异步清零可以成功取消原本 EX 段的指令，而 MEM 段本身的指令已送入 WB 段，不会产生影响，如图 4.15 所示。

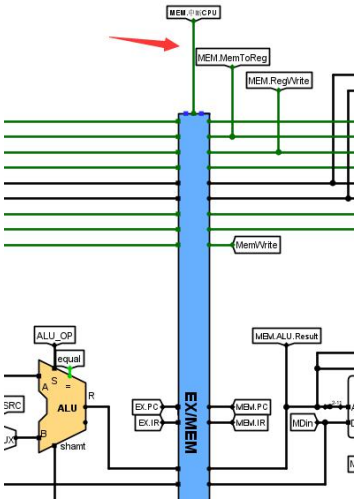

图 4.15 添加异步清零信号

4.3.7 动态分支预测故障 1

动态分支预测：降低时钟周期数很少，BHT 未按预期发挥性能。

故障现象：在数据转发流水的基础上加入动态分支预测技术后，时钟周期只降低了不到 100，而正常情况应该降低到 1800 左右，BHT 没有发挥预期的性能。

原因分析：通过自己编写了含大量条件分支的测试程序测试发现，LRU 淘汰算法的计数清零端逻辑有问题，一开始设置的是该 Cache 槽的读出信号 L 和写入信号 W 有效时，对计数进行清零，但由于计数器的清零端是异步清零，当淘汰算法选择计数最大值的 Cache 槽产生 W 信号时，将其异步置为 0，导致淘汰算法的归并比较输出改变，进一步改变选择的 Cache 槽，这样会产生连锁反应，将所有的计数器都置为 0，从而选择写入的 Cache 槽永远是固定的。

解决方案：改进方法是通过将 W 信号传入寄存器，在下一个时钟周期到来时进行清零，虽然计数器清零与写入相差一个时钟周期，但这样避免了异步时序产生的错误，且 LRU 算法选择的 Cache 槽永远是计数值最大的，如所示。

图 4.16　使用寄存器将清零信号置为同步信号

4.3.8　动态分支预测故障 2

动态分支预测：改进 BHT 的 LRU 淘汰计数清零逻辑后，时钟周期降低效果仍然不明显。

故障现象：在对故障 4.3.7 改进以后，运行 benchmark 测试程序，发现时钟周期数在 2000 左右，仍未达到预期效果。

原因分析：在进行逐指令调试后没发现导致影响性能的关键所在，遂使用 Logisim 的日志打印功能，将每个时钟周期下五个功能段的 PC 值打印输出，并请完成了动态分支预测的同学也打印一份日志，编写了一个简单的 C 程序对日志内容进行比对，发现问题在于 benchmark 测试程序中的排序功能测试部分，进一步通过打印排序测试程序的日志比较发现问题在于分支临界点处设置了错误的分支转移地址，传入 BHT 的分支目标地址存放了 EX 段计算出的实际情况下的下一个指令的 PC，但应该保存分支目标地址（即使分支失败），即写入 PC 的值中顺序寻址的计算错误地使用了 ID 段的值（数据转发流水中是在 ID 段计算），由于使用了 BHT，下一个指令的 PC 应该由 BHT 预测给出，故若分支失败或顺序寻址应该使用 EX.PC+4，而不是 ID.PC+4.。

解决方案：修改了写入 BHT 的预测地址的输入，改为分支指令的目标地址，预测地址由当前状态判断分支成功还是分支失败，修改后动态分支预测将时钟周期降低到了 1789，如图 4.17 所示。

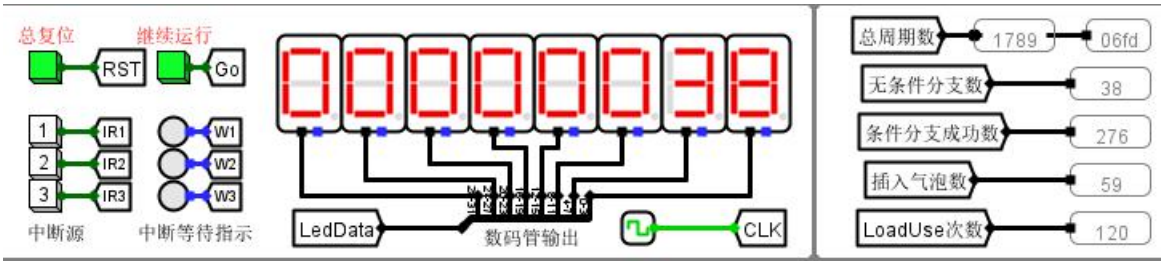

图 4.17　动态分支预测的时钟周期

4.4 实验进度

表 4.2 课程设计进度表

时间	进度
第一天	阅读课程设计任务书，了解课设任务，初步进行数据通路的搭建
第二天	完成数据通路的搭建和硬布线控制器控制信号的产生，最终完成支持 28 条指令的单周期 CPU 并通过 educoder 平台测试，记录了问题及解决方案
第三天	完成单周期单级中断 CPU 的设计，通过 educoder 平台测试，记录了问题及解决方案
第四天	完成单周期多级中断 CPU 的设计，通过 educoder 平台测试，记录了问题及解决方案
第五天	完成理想流水线的设计，通过 educoder 平台测试，完成较为顺利未遇到问题
第六天	完成气泡流水线的设计，通过 educoder 平台测试，记录了问题及解决方案
第七天	完成数据转发流水线的设计，通过 educoder 平台测试，记录了问题及解决方案
第八天	完成中断流水线的设计，自行测试通过，记录了问题及解决方案
第九天	完成 BHT 表的电路设计，对电路进行了信号仿真，确保 BHT 表的功能正确
第十天	使用 BHT 表改造数据转发流水线，实现了动态分支预测流水，修改电路细节，使得动态分支预测流水具有较好的性能，记录遇到的问题及解决方案
第十一天	初步开展团队项目，搜索可选择的选题方案，进行可行性分析
第十二天	确定选题，搭建了最初版本的电路和驱动代码，进行测试工作
第十三天	通过测试反馈不断修正电路和代码，完成团队项目的设计工作
第十四天	制作答辩 PPT、录制答辩视频，录制 CCMB、中断流水和 BHT 演示视频

5 团队项目：井字棋游戏

我们的团队项目是利用支持单级中断的单周期 CPU 实现了一个井字棋的对战小游戏，游戏支持两个玩家交替操作，并可显示棋子，下棋子通过点击按钮实现，9 个按钮对应 9 个井格，棋盘使用 CS3410 提供的 LCD Vedio 器件，电路如图 5.1 所示。

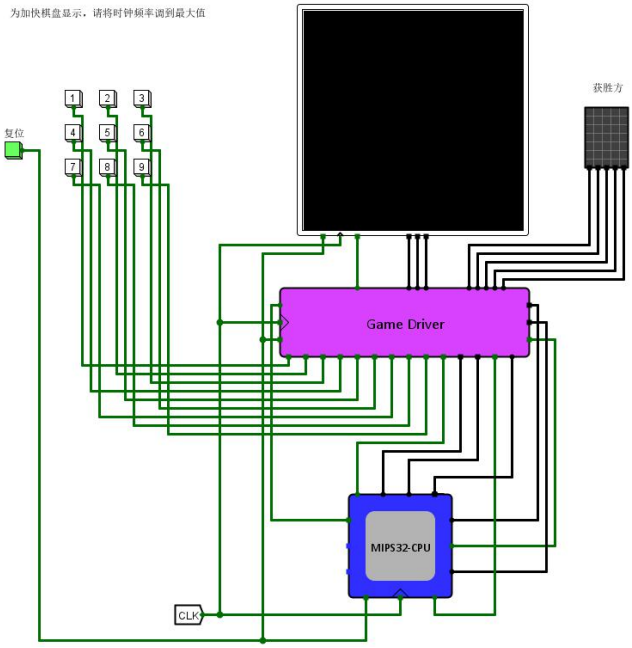

图 5.1 井字棋游戏电路

主要设计思想是游戏的控制由 Game Driver 电路完成，LCD 的显示驱动由 CPU 执行驱动代码完成。电路启动时钟仿真后，会首先绘制棋盘格局，然后进入监听循环，玩家点击按钮落子后，由 Game Diver 电路负责对游戏数据存储器的一系列修改及产生中断信号，通知 CPU 进入显示棋子的中断服务程序，CPU 通过使用系统调用，通知 Game Driver 电路，进而将显示数据传递到 LCD Video。

我主要在团队中负责电路的设计、MIPS 汇编代码的编写工作。首先我对单周期 CPU 进行改造，将原本的数据存储器 DM 取消，通过接口访问 Game Driver 电路中的游戏数据存储器 Game Buffer，然后对 CPU 进行了封装方便调用，接着就是设计项目中重要的游戏驱动电路 Game Driver，该电路负责落子合法性检测、修改数据、通知 CPU、胜负相关判断等四项工作，落子合法检测和修改数据的功能用到了状态

机和总线的设计技术，胜负相关判断使用了组合逻辑电路进行并发比较。最后，我编写了 LCD Video 的显示驱动代码，使之通过 2 号系统调用驱动 LCD 显示棋盘及棋子，最终驱动代码量在 270 行左右，游戏演示如图 5.2 所示。

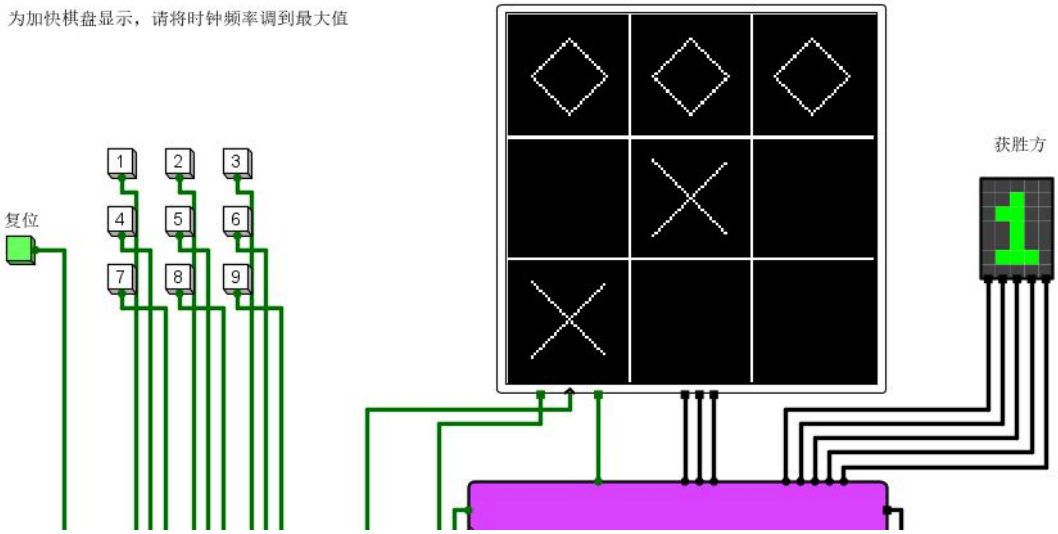

图 5.2 井字棋游戏演示

游戏电路支持一定的异常处理，包括落子合法性检测、胜负判断检测、玩家交替逻辑。

6 设计总结与心得

6.1 课设总结

在本次课程设计的个人任务部分，做了如下几点工作：

1） 实现支持 24 条基本指令的单周期 CPU，并通过测试；

2） 实现了个人分配的 CCMB 拓展指令，并在所有后续电路中提供支持；

3） 实现了理想流水、气泡流水、数据转发流水，均通过测试；

4） 实现了 EPC 硬件堆栈的单级中断、多级中断 CPU，并通过测试；

5） 实现了单级中断流水，能够正确运行中断测试程序；

6） 实现了基于数据转发流水的动态分支预测技术，设计了 BHT，电路最终能够正确执行 benchmark 测试程序，并将时钟周期数由 2297 降低到了 1789，使流水性能提升幅度达到 22.1%。

在本次课程设计的团队项目部分，做了如下几点工作：

1） 与组员叶笑共同讨论，确定了团队项目的选题；

2） 实现电路的设计及汇编代码的编写，通过组员蒋韬和屈芷萱进行测试反馈，进一步修正电路和代码；

3） 制作了答辩 PPT 初稿，录制了答辩视频。

6.2 课设心得

由于疫情的影响，本次组成原理的课程设计只能在家完成，之前就听说这个课设难度大，具有挑战性，但可以自主发挥，按自己所能一步步点亮技能，于是在课设之前，我定下了完成所有模块的目标。但在两周的课设时间内，我并没有按自己所设想的完成所有的模块，因为刚过完年，在家懒散，没有在学校跟同学一起奋斗的劲，让我逐渐松懈了对自己的要求。好在课设延期提交，让我抓住了这根救命稻草，在后续的空闲时间中完成了自己定下的目标。

回顾整个课设过程，有痛苦，也有丰收的喜悦。在顺利完成单周期的 CPU 设计和理想流水线的设计后，在气泡流水和数据转发流水的实现上遇到了很多困难，因为组成原理课程并没有介绍流水线的相关知识，好在课程组提供了流水线相关的

实验资料，再搭配 MOOC 课程的学习，我逐渐了解了流水线技术的原理，并在不断的犯错和改错的过程中完成了气泡流水和数据转发流水。然后回到单周期的单级中断和多级中断设计上，由于中断相关知识课程中学习过，故照着实验指导可以顺利完成，在理解了流水线技术的原理后，中断流水和动态分支预测其实知识对流水的进一步改造，虽然也遇到了很多问题，但通过这两个电路的实现让我对流水线有了更深刻的理解。

本次课设的团队项目是用实现的 CPU 做一些项目，没有选题限制，但由于没有过这样的经验，我们不知道可以用一块 CPU 来做些什么，因此在选题阶段很迷茫。后来通过大量的讨论，以及受 MOOC 上优秀作品演示的启发，我们最终确定了选题，做一个小游戏。在实现过程中，没有指导，全凭自己摸黑前进，一步步设计出游戏驱动电路，编写显示驱动代码，最终完成后，内心充满了成就感。

然而对于本次课程设计，我还有一些小小的建议和改进。虽然引入了团队任务，但由于电路设计这方面交接较为困难，很难调动团队的成员一起参与到其中，而且由于是大三下学期，很多人在准备考研，所以难免有划水的现象，我们组团队项目的很多工作都是我个人完成的。希望课程组能够在以后对团队任务进行一些改进，能够让组员们都参与进来。

最后，感觉课程组老师在疫情期间能够积极为我们解答困惑，以及在 educoder 上提供自动测试和测试代码，让我们在测试环节上省去了不少时间。这次课设虽然完成了，但学习的道路没有停止，这次课设激发了我对计算机组成原理的浓厚兴趣，希望在将来能够更进一步。

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] 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011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.

• 指导教师评定意见 •

一、原创性声明

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

特此声明！

作者签字: 练炳诚