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：	CS1702
学　　号：	U201714529
姓　　名：	陈益欣
电　　话：	18651560826
邮　　件：	xinxrong@foxmail.com

目 录

1 课程设计概述 1

1.1 课设目的 1
1.2 设计任务 1
1.3 设计要求 1
1.4 技术指标 2

2 总体方案设计 4

2.1 单周期 CPU 设计 4
2.2 中断机制设计 9
2.3 流水 CPU 设计 11
2.4 气泡式流水线设计 13
2.5 数据转发流水线设计 13

3 详细设计与实现 15

3.1 单周期 CPU 实现 15
3.2 中断机制实现 17
3.3 流水 CPU 实现 20
3.4 气泡式流水线实现 20
3.5 数据转发流水线实现 21

4 实验过程与调试 23

4.1 测试用例和功能测试 23
4.2 主要故障与调试 24
4.3 实验进度 25

5 团队项目 26

5.1 总体设计 26

5.2 项目分工.....26
5.3 我的实现内容.....26
5.4 出错分析.....26
5.5 成果展示.....26
6 设计总结与心得 27
6.1 课设总结.....27
6.2 课设心得.....27
参考文献 28

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 指令集前 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	SRLV	逻辑可变右移	指令格式参考 MIPS32 指令集，最终功能以 MARS 模拟器为准。
29	SRAV	算术可变右移	
30	SH	存储半字	
31	BLEZ	小于等于 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。将系统分成指令存储器 IM、操作控制器、寄存器文件、运算器 ALU、数据存储器 DM、地址逻辑转移等几大模块。从指令存储器中取出一条指令，将 OP 和 FUNC 字段送至操作控制器，将 RS、RT 和 RD 字段送至寄存器文件通过控制信号进行选取。操作控制器根据指令解析出各模块间的数据通路控制信号，寄存器文件选取出相应寄存器的值作为 ALU 的操作数，ALU 的运算结果送到数据存储器或者寄存器文件。地址逻辑转移决定了下一条指令的地址。该处理器支持表 1.1 指令集指令集所列出的所有指令。以上所有操作都在一个周期内完成，不需要考虑各种数据冲突。但是取指令和取数据都设计到访存，可能会有冲突，因此采用哈弗结构将指令存储器和数据存储器分开，用增加硬件的方式解决此冲突。

总体结构图如图 2.1 所示。

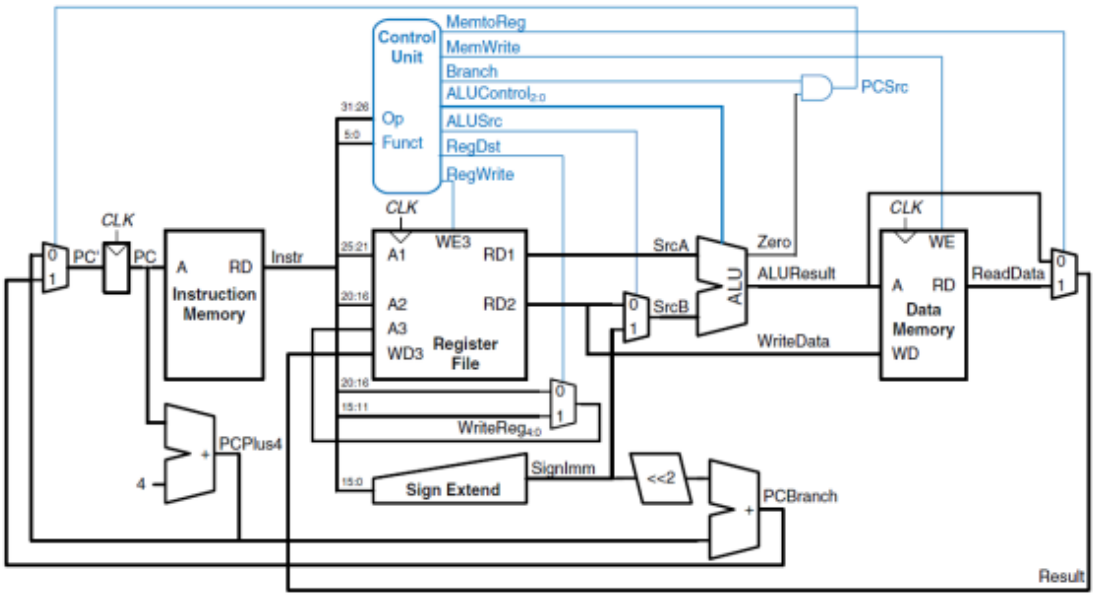

图 2.1 总体结构图

2.1.1 主要功能部件

1. 程序计数器 PC

程序计数器决定所取指令的地址，程序计数器接收地址转移逻辑的输出作为其输入，在每个时钟的上升沿到来时会输出一个地址到指令存储器。因为这里需要暂时存储指令的地址，所以需要用一个寄存器来实现。该寄存器需要的引脚有：时钟信号、清零信号和使能端接停机信号。当停机信号到来时，PC 寄存器使能端为 0，PC 不再改变，系统停机。

2. 指令存储器 IM

为了简化实现，采用哈弗结构将指令存储器和数据存储器分开实现。指令是只读的因此用 ROM 存储器来存储。MIPS 指令都是 32 位定长指令，因此指令存储器的数据位宽设置为 32 位。程序计数器 PC 输出的是字节地址，所以指令存储器应该以字寻址，将 PC 截去第二位后作为指令存储器的输入。考虑到实现的是一个模拟 CPU，因此只选取 10 位地址即可，即选取程序计数器 PC 的 2-11 位作为指令存储器的输入。

3. 数据存储器 DM

数据存储器要求可读可写，因此用 RAM 存储器来实现。LW，SW 指令要求读写字，写使能信号决定是否写数据，时钟上升沿控制写入。

4. 运算器

运算器接受两个 32 位的操作数，根据操作码决定对两个数进行何种运算，输出一个 32 位的结果，并输出一位结果 Equal 表示两个操作数是否相等。运算器 ALU 的输入输出引脚描述如表 2.1 算术逻辑运算单元引脚与功能描述所示。

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

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

5. 寄存器堆 RF

寄存器文件中包含 MIPS 的 32 个 32 位寄存器，一次可以读取 R1#和 R2#两个寄存器，读取出的寄存器的值由 R1 和 R2 表示，R1 和 R2 送至运算器 ALU 作为操作数进行运算。一次可以写一个寄存器用 W#表示，写入的数据由 Din 表示。用时钟信号来控制寄存器的工作，时钟上升沿寄存器堆对寄存器进行操作。一位写使能信号 WE 用来表示当前周期是否需要写数据，写入和读出的数据都是 32 位的。

2.1.2 数据通路的设计

在单周期 CPU 的基础上增加了 4 条拓展指令：两条运算指令 SUBU 和 XORI，存储访问指令 LH，以及分支指令 BLTZ。单周期 CPU 的数据通路如表 2.2 指令系统数据通路框架所示。

表 2.2 指令系统数据通路框架

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
SLL	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	0	无	无
SRA	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	1	无	无
SRL	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	2	无	无
ADD	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	5	无	无
ADDU	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	5	无	无
SUB	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	6	无	无

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
AND	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	7	无	无
OR	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	8	无	无
NOR	PC+5	PC	Rs	Rt	Rd	ALU	R1	R2	10	无	无
SLT	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	11	无	无
SLTU	PC+4	PC	Rs	Rt	Rd	Rt	R1	R2	12	无	无
JR	GPR[rs]	PC	Rs	无	无	无	无	无	无	无	无
SYSCALL	PC+4	PC	无	无	无	无	无	无	无	无	无
J	$PC+4_{[31:28]}$+立即数$_{[26:0]}$<<2	PC	无	无	无	无	无	无	无	无	无
JAL	$PC+4_{[31:28]}$+立即数$_{[26:0]}$<<2	PC	无	无	31	PC+4	无	无	无	无	无
BEQ	PC+4+立即数[32:0]<<2	PC	Rs	Rt	无	无	R1	R2	无	无	无
BNE	PC+4+立即数[32:0]<<2	PC	Rs	Rt	无	无	R1	R2	无	无	无
ADDI	PC+4	PC	Rs	Rt	Rt	ALU	R1	立即数	5	无	无
ANDI	PC+4	PC	Rs	Rt	Rt	ALU	R1	立即数	7	无	无
ADDIU	PC+4	PC	Rs	Rt	Rt	ALU	R1	立即数	5	无	无
SLTI	PC+4	PC	Rs	Rt	Rt	ALU	R1	立即数	11	无	无
ORI	PC+4	PC	Rs	Rt	Rt	ALU	R1	立即数	8	无	无
LW	PC+4	PC	Rs	Rt	Rt	MEM	R1	立即数	5	ALU	无
SW	PC+4	PC	Rs	Rt	无	MEM	R1	立即数	5	ALU	无
SRLV	PC+4	PC	Rs	Rt	Rt	ALU	R1	立即数	6	无	无
SRAV	PC+4	PC	Rs	Rt	Rd	ALU	R1	R2	7	无	无
SH	PC+4	PC	Rs	Rt	Rt	MEM	R1	立即数	5	ALU	无
BLEZ	$PC+4_{[31:28]}$+立即数$_{[26:0]}$<<2	PC	Rs	无	无	无	R1	无	11	无	无

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 主

控制器控制信号的作用说明所示。

表 2.3　主控制器控制信号的作用说明

控制信号	取值	说明
MemToReg	0	得到的值来自 ALU 的运算结果
	1	得到的值来自数据存储器
MemWrite	0	不需要向数据存储器写入
	1	需要向数据存储器写入
ALU_SRCB	0	选择从 R2 读出的数据
	1	选择立即数
RegWrite	0	寄存器不需要进行写操作
	1	寄存器需要进行写操作
SYSCALL	0	不是 SYSCALL 指令
	1	If $v0==10 halt(停机指令)else　数码管显示$a0 值
SignedExt	0	立即数无符号拓展
	1	立即数符号扩展
RegDst	0	W#选择 rt
	1	W#选择 rd
BEQ	1	R1 和 R2 相等
BNE	1	R1 和 R2 不相等
JR	1	JR 指令，跳转寄存器 Rs 中存放的地址
JMP	1	JAL,JR,J 三种指令，无条件跳转指令
JAL	1	JAL 指令

对照所有控制信号，依次分析各条指令，分析该指令执行过程中需要哪些控制信号，对于与本条指令无关的控制信号，控制信号的取值一律为 0，以简化控制器电路的设计。该控制信号表的框架如图 2.2　主控制器控制信号框架所示。

#	指令	OpCode (十进制)	FUNCT (十进制)	ALU_OP	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL	SRLV	SRAV	SH	BLEZ	ERET
1	SLL	0	0	0				1			1										
2	SRA	0	3	1				1			1										
3	SRL	0	2	2				1			1										
4	ADD	0	32	5				1			1										
5	ADDU	0	33	5				1			1										
6	SUB	0	34	6				1			1										
7	AND	0	36	7				1			1										
8	OR	0	37	8				1			1										
9	NOR	0	39	10				1			1										
10	SLT	0	42	11				1			1										
11	SLTU	0	43	12				1			1										
12	JR	0	8	X							1			1	1						
13	SYSCALL	0	12	X					1		1										
14	J	2	X	X											1						
15	JAL	3	X	X				1							1	1					
16	BEQ	4	X	X								1									
17	BNE	5	X	X									1								
18	ADDI	8	X	5			1	1		1											
19	ANDI	12	X	7			1	1													
20	ADDIU	9	X	5			1	1		1											
21	SLTI	10	X	11			1	1		1											
22	ORI	13	X	8			1	1													
23	LW	35	X	5	1		1	1		1											
24	SW	43	X	5		1	1			1											
25	SRLV	0	6	2				1			1						1				
26	SRAV	0	7	1				1			1							1			
27	SH	41	X	5		1	1			1									1		
28	BLEZ	6	X	11																1	
29	ERET	16	24	X																	1

图 2.2 主控制器控制信号框架

2.2 中断机制设计

2.2.1 总体设计

中断实验主要分为两部分：单级中断和多级中断，这里说明多级中断的实现机制，其过程分为如下几个阶段。

(1). 取指令，执行指令，判断是否有中断信号

(2). 若没有中断信号，重复上述操作

(3). 若有中断信号，此时中断周期内执行：关中断，保存断点至 EPC，进行中断识别然后将中断服务程序地址送至 PC。

(4). 中断服务程序进行保护现场，设置新的中断屏蔽字(若有)，保存 EPC。然后开中断，执行中断服务子程序，关中断，恢复现场，开中断，最后进行中断返回。

用独立请求的方式响应中断，每个中断对应一个中断号。根据中断号确定执行哪段中断程序，用类似于中断向量表的方式存放中断程序入口，可以事先获取中断入口地址，以常量形式进行选择。

当中断信号来临时需要打断主程序，高优先级的中断可以中断低优先级的中断，每次主程序被中断或者有更高优先级的中断中断低优先级的中断的时候，将当前的 PC 存放到一个寄存器 EPC 里进行压栈保存。在多级中断中，中断信号的产生，优先级的比较以及 PC 值的保存时使用硬件实现的，其余是使用软件的方式实现的。

在执行某些操作时，不允许任何外部事件中断，需要完全关中断，所以还需要添

加中断使能寄存器 IE。

2.2.2 硬件设计

① 中断计数器

利用一个计数器实现，位宽为 2，时钟上升沿触发，正常执行时写入 0，表示处理的是第几个中断。计数器使能端接 ERET 或 CPU 中断请求，即代表进入中断或者推出中断时计数器的值需要进行改变。如果有中断被打断进入更高的优先级，或者由没中断产生到有中断产生即此时 CPU 请求信号为 1，下一个时钟周期到来时计数器值加 1。当高优先级的中断执行完毕时，此时 ERET 信号为 1，计数器进行减 1 操作，表示进入低优先级的中断。

② 中断识别

三个中断按钮都可能被按下，但是一次只能响应一个，此时优先响应优先级高的中断，并且优先级高的中断可以中断优先级低的中断，中断筛选电路的实现需要使用当前的中断优先级、新中断的优先级以及优先编码器来判断当前进行的中断的优先级是否是最高的。

③ 中断使能寄存器 IE

利用寄存器实现，寄存器位宽为 1 位，当 IE 寄存器为 0 时，代表可以进行中断嵌套；当 IE 寄存器为 1 时，代表不可以进行中断嵌套。首先 CPU 中断请求到来时，会设置 IE 寄存器值为 1 表示关闭中断，此时进行保护现场，PC 压栈等操作。其次 MFC0 表示开中断，此时设置 IE 寄存器值为 0 表示可以进行中断嵌套。然后 MTC0 表示关中断，此时设置 IE 寄存器值为 1 不可以进行中断嵌套，此时进行恢复现场等操作。最后进行中断返回，ERET 信号为 1，并设置 IE 寄存器值为 1，代表可以进行新的中断。

④ 修改控制操作器

因为多级中断需要开关两次中断，此时需要增加三条指令 MTC0、MFC0 和 ERET 指令。信号产生同之前指令一样只需判断指令的 OP 和 FUNC 字段即可。

⑤ EPC 寄存器

用 3 个 32 位寄存器实现，上升沿触发。如果有中断被打断进入更高的优先级，或者更高的优先级执行完毕进入之前没进入的中断，通过中断计数器调整相应 EPC 寄存器使能端置 1，写入当前 PC+4 的值作为返回地址。中断执行过程中，使能端一直

为 0，不许写入新的值。中断返回时，根据计数器的值选取相应 EPC 寄存器的值作为中断返回地址。

2.2.3 软件设计

① 保护现场和恢复现场

当进入中断程序之前，需要进行现场的保存。主要是将各个寄存器的值进行压栈处理。在中断处理子程序用将之后子程序用用到的寄存器的值全都进行压栈保存，在中断返回是将栈的寄存器的值全部弹出。

② 开关中断

本次实验的开关中断是使用软件来实现，使用 mfc0 和 mtc0 实现。MFC0 和 MTC0 都作为一个简单的控制信号来控制 IE 寄存器的取值从而实现开关中断的效果。

2.3 流水 CPU 设计

2.3.1 总体设计

在流水线的设计过程中，将 MIPS 指令的执行过程可以划分为 5 个阶段，分别是取值 IF,译码 ID,执行 EX,访存 MEM 和写回 WB。每个阶段之后都有一个锁存器用于锁存本段处理完成的数据，从而能将此阶段的结果保留给下一阶段使用，后端对数据的加工处理依赖于前段接口传递过来的信息。

PC 计数器和 5 段流水之间的 4 个锁存器都用公共的时钟同步，因此各个阶段可以并行运行，从而提高系统的执行效率。5 段流水线的总体设计如图 2.3　5 段流水线总体设计图所示。

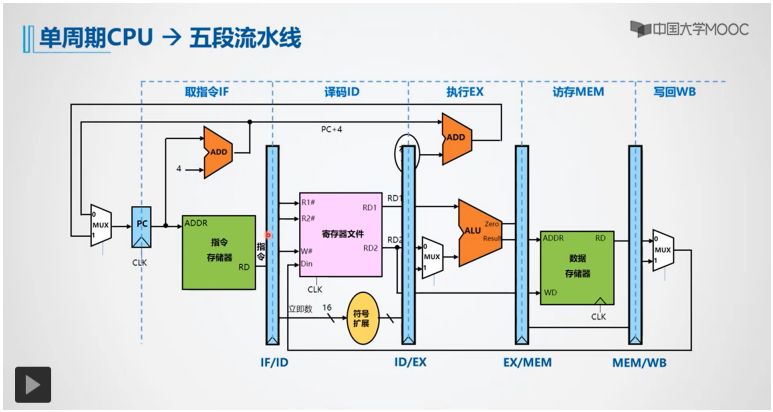

图 2.3　5 段流水线总体设计图

2.3.2 流水接口部件设计

流水线在段间设置流水接口部件，流水接口部件的作用是将上一段中的数据和控制信号锁存一个周期，供下一段使用，应该用寄存器实现。通过接口传递与指令相关的数据、控制、反馈信息。流水接口部件采用时钟上升沿触发；并且其同时具有使能端接停机引脚；将接口部件内所有寄存器清零的引脚。各段的流水接口结构大体相同，只是传输的数据不同。后段对数据的加工处理依赖于前段接口传递过来的信息。5 段流水线的流水接口部件设计如所示。

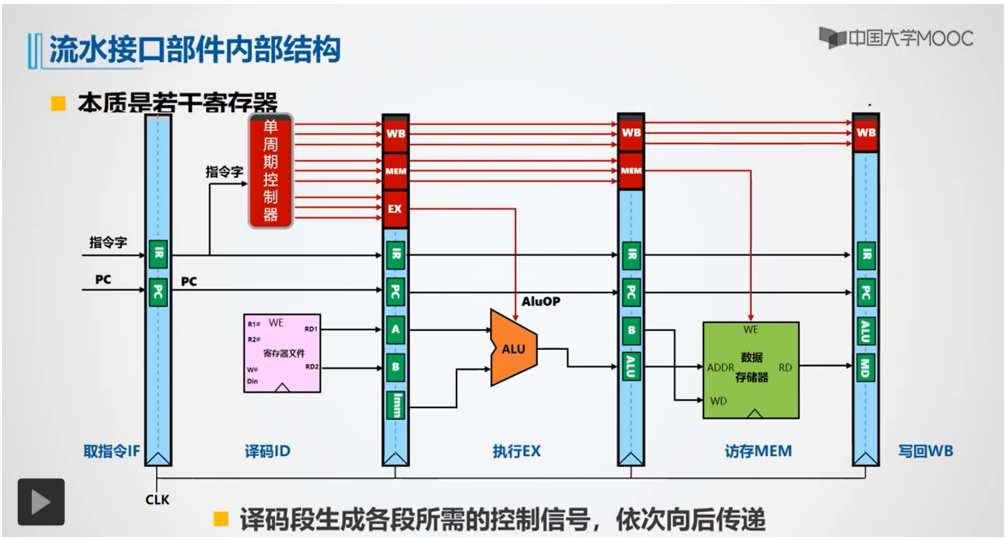

图 2.4 5 段流水线的流水接口设计

2.3.3 理想流水线设计

理想流水线阶段数相同，段时延相同，无资源冲突，无不存在

将单周期 CPU 中的各部件依次划分到五个段中。

① 取指：程序计数器 PC 将指令地址送至指令存储器 IM，取出指令。

② 译码：IR 将 FUNC 和 OP 字段送至单周期控制器，将 Rs 和 Rt 字段送至寄存器堆取出 R1 和 R2 的值，R1、R2 和控制器产生的各个控制信号送至下一段。

③ 执行：运算器 ALU 根据操作码执行各种运算，并将运算结果送至下一段。

④ 访存：指令对数据存储器进行访问，写入数据或者读取数据送至下一段。

⑤ 写回：与译码段使用的是同一个寄存器堆，不过执行的是写寄存器操作，为了防止数据冲突，寄存器堆需要采用下降沿写入，保证下一个上升沿到来时，译码阶段能取到最新的寄存器值。

2.4 气泡式流水线设计

理想流水线只能顺序执行，并未解决分支相关和数据相关。

若指令中存在分支指令，因为将分支指令放在 EX 阶段进行执行，所以分支指令的跳转在 EX 段才能够成功判断。此时若此指令为分支指令，则 IF 段和 ID 段提前取出的指令已经作废即这条指令不需要执行，因此需要将 IF 和 ID 段的流水接口部件进行清空，此时需要插入两个气泡。

若相邻指令间存在数据相关，因为指令都是顺序执行的，所以可以对结果产生影响的只有先写后读，即后面的指令要读取寄存器的值，但此时寄存器中的值为旧值，此时存在三种可能，EX 段与 ID 段产生数据相关、MEM 段与 ID 段产生数据相关以及 WB 段与 ID 段产生数据相关。因此需要插入气泡，使得之前的指令执行完毕后，ID 段再对寄存器的值进行读取。

有条件指令和无条件指令都在 EX 段进行跳转，若指令成功进行跳转，则当前 IF 段和 ID 段取出的指令都已作废，此时需要将这两段的流水接口部件进行清空，并将目标地址送至程序计数器 PC，执行新的指令。若指令没有成功跳转此时指令仍按照顺序执行。

经过分析可知流水线中产生的数据相关可能有三种。对于 WB 段与 ID 段产生的数据冲突只需要修改寄存器堆为下降沿写入即可确保在下一个上升沿到来时，ID 段能取到的值为最新值。对于 MEM 段与 ID 段产生的冲突，此时 IF 和 ID 段的指令都需要等待 MEM 段的指令执行完毕后才可以继续执行，因此此时需要插入一个气泡使得 IF 与 ID 段指令停止执行一个周期。对于 EX 和 ID 产生的冲突，此时 IF 和 ID 段的指令都需要等待 EX 段的指令执行完毕后才可以继续执行，因此此时需要插入两个气泡使得 IF 与 ID 段指令停止执行两个周期。

判断 ID 段与 EX 或 MEM 段是否产生冲突的逻辑为如果 MEM 或 ID 段的写寄存器的编号与 ID 段读寄存器的编号相同则产生数据冲突，但是由于 0 号寄存器的值恒为 0 因此需要排除这种情况。

2.5 数据转发流水线设计

重定向对于分支相关的解决方案并没有进行改变，对于数据相关采用了不同的思路。重定向通过将 ID 段需要用到的寄存器的值，从 EX 或 MEM 段直接送到其所需

要的地方。重定向检测位于 ID 段，通过判断 ID 段所需要的寄存器编号与 EX 或者 MEM 段所写的寄存器编号是否相同，若相同则产生冲突，此时需要将新值重定向到相应的位置。但是如果产生 LOAD-USE 冲突，即 EX 段的指令为 LW，此时仍需要插入一个气泡使得指令流水线停顿一个周期。

LOAD-USE 相关在 ID 段进行检测，当 ID 段与 EX 段产生数据相关且 EX 段指令为 LBU 指令或 LW 指令时，产生了 LOAD-USE 相关，此时需要在指令流水线中插入一个气泡使得指令停顿一个周期，从而流水线可以顺利进行。

重定向是从 MEM 段或者 WB 段的值重定向到 ALU 的两个输入端。两段的数据均可能定位到 ALU 的两个输入端，因此 ALU 的两个输入端的输入数据是相同的，但多路选择器的选择信号不同。对于任意一个输入端其有 4 种可能取值，因此其控制信号的位宽为 2 位，直接从寄存器文件中取出、来自 MEM 段的 ALU、来自 WB 段的 ALU 和来自 WB 段的 MD。通过检测其产生数据相关的位置以及指令的类型可以判断新值的取值位置，从而得到每个输入端的多路选择器的控制信号。

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）

① Logism 实现：

使用一个 32 位寄存器实现程序计数器 PC，触发方式为下降沿触发，输入为下一条将要执行的指令的地址，输出为当前执行指令的地址。Halt 为停机信号，将此控制信号通过非门取反之后和时钟相与，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，与时钟信号相与，屏蔽时钟信号，使整个电路停机。如图 3.1 所示。

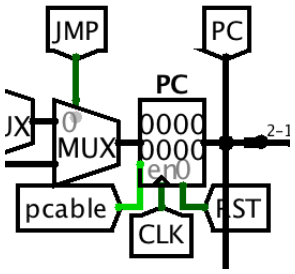

图 3.1 程序计数器（PC）

2) 指令存储器（IM）

① Logism 实现：

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

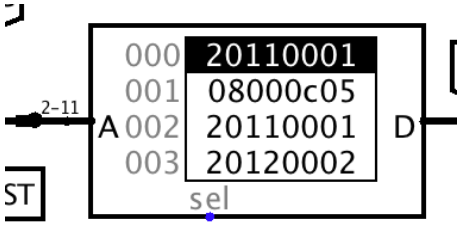

图 3.2 指令存储器（IM）

3.1.2 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。数据通路表已在表 2.2 指令系统数据通路框架中给出

根据列出的数据通路表，进行多指令数据通路的合并输入数，表，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。数据通路如图 3.3 单周期 CPU 数据通路（Logism）所示。

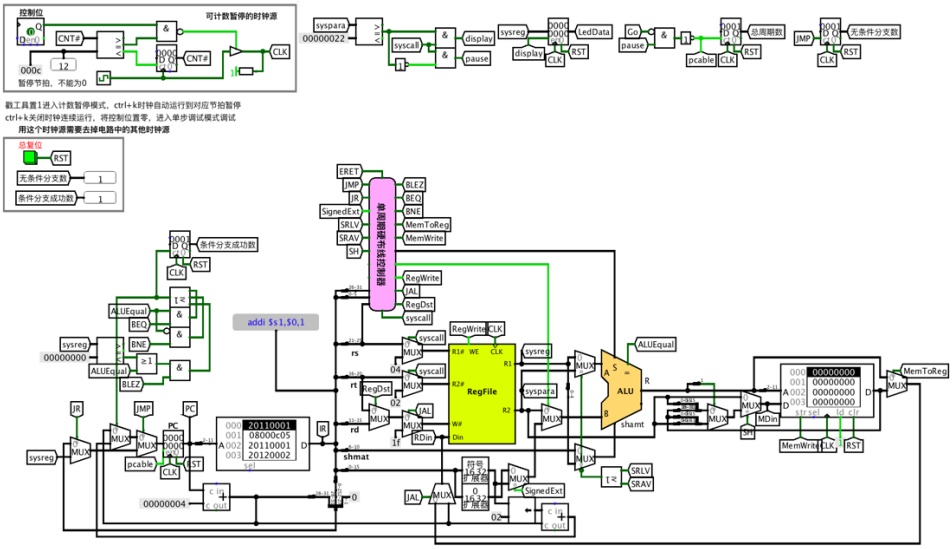

图 3.3 单周期 CPU 数据通路（Logism）

3.1.3 控制器的实现

控制信号表如图 2.2 主控制器控制信号框架所示。根据指令的 OP 和 FUNC 字段对指令进行判断，然后利用信号表中生成的表达式在 Logisim 中自动生成电路。得到控制器原理图如图 3.4 主控制器原理图所示。

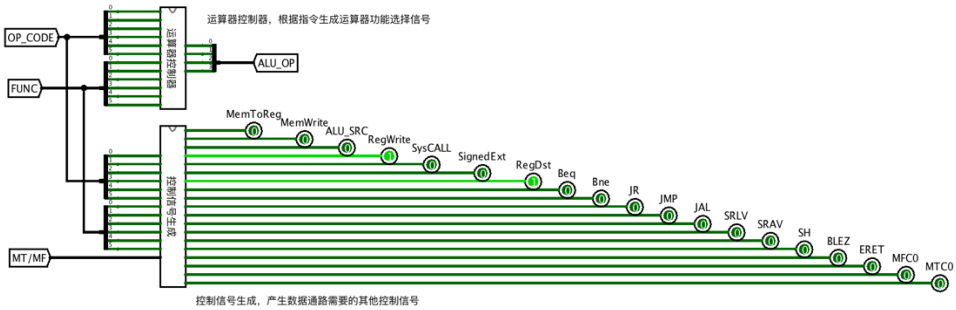

图 3.4　主控制器原理图

3.2 中断机制实现

3.2.1 硬件实现

① 中断信号产生

用两个 D 触发器实现，中断按键信号接到第一个触发器的时钟端，D 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。电路如图 3.5　中断信号产生电路所示。

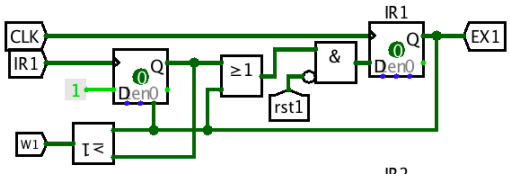

图 3.5　中断信号产生电路

② 中断入口地址选择

利用优先编码器对产生的中断进行判断，选出优先级最高的中断。将各个中断的入口地址以常数的形式接入到多路选择器的输入端，用优先编码器产生的输出作为选择信号，选出的地址即为应当进入的中断程序的入口地址。电路如图 3.6　中断入口地址选择所示。

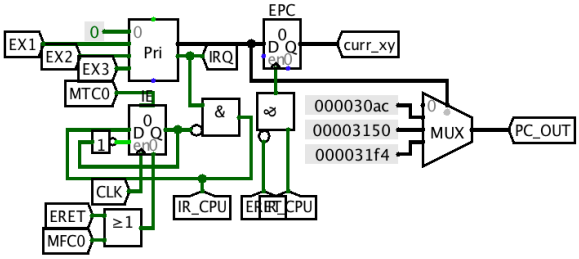

图 3.6　中断入口地址选择

③ 中断计数器

利用一个位宽为 2 的计数器实现，时钟上升沿触发，表示目前正在程序中有几个中断服务程序正在执行。当有更高优先级的字段产生时即 CPU 中断请求信号为 1 时，中断计数器加一。当高优先级执行完毕进入低优先级时，程序计数器减一。电路如图 3.7　中断计数器所示。

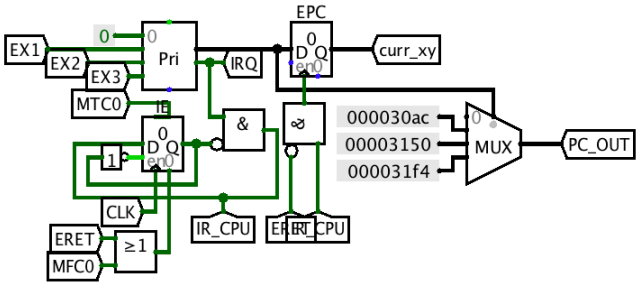

图 3.7　中断计数器

④ EPC 寄存器堆栈

用 3 个 32 位寄存器实现，当有更高优先级的中断进入时，此时中断计数器加 1，然后根据中断计数器的值来判断当前返回地址 PC+4 应该存储到哪一个 EPC 寄存器中。电路如图 3.8　EPC 寄存器堆栈所示。

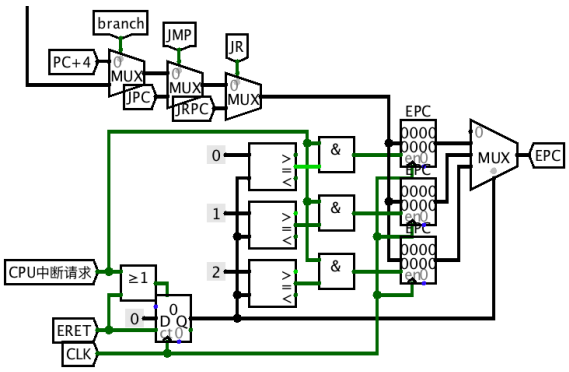

图 3.8　EPC 寄存器堆栈

⑤ IE 寄存器

IE 寄存器利用寄存器实现，寄存器位宽为 1 位，当 IE 寄存器为 0 时，代表可以进行中断嵌套；当 IE 寄存器为 1 时，代表不可以进行中断嵌套。首先 CPU 中断请求到来时，会设置 IE 寄存器值为 1 表示关闭中断，此时进行保护现场，PC 压栈等操作。其次 MFC0 表示开中断，此时设置 IE 寄存器值为 0 表示可以进行中断嵌套。然后 MTC0 表示关中断，此时设置 IE 寄存器值为 1 不可以进行中断嵌套，此时进行恢复现场等操作。最后进行中断返回，ERET 信号为 1，并设置 IE 寄存器值为 1，代表可

以进行新的中断。电路如图 3.9　IE 寄存器所示。

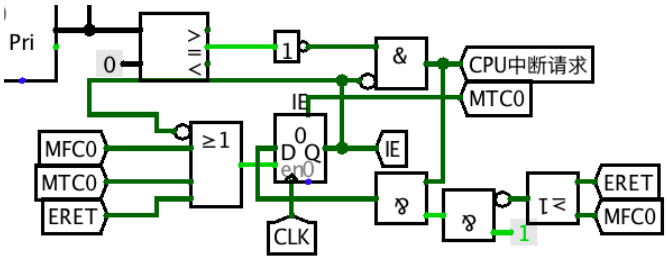

图 3.9　IE 寄存器

⑥ 中断控制信号生成

已经在单周期硬布线控制器中加入了，只需比较指令的 OP 和 FUNC 字段即可。电路如图 3.10　中断控制信号生成所示。

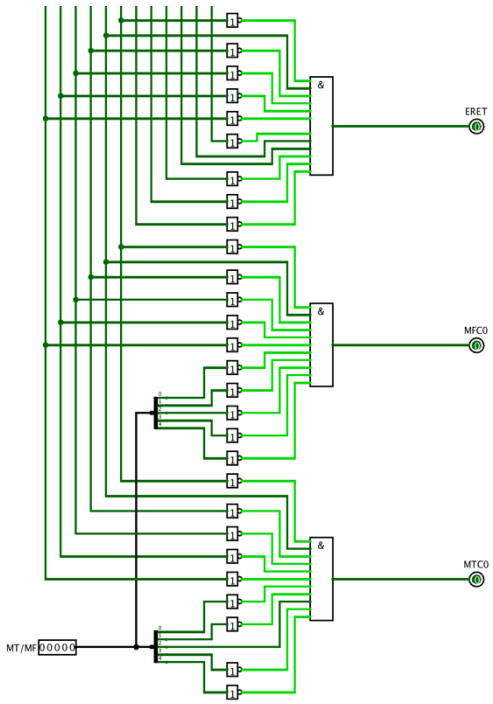

图 3.10　中断控制信号生成

3.2.2　软件实现

① 开关中断

需要增加一条指令 MTC0 和 MFC0 进行一次开中断和关中断。

② 保护现场和恢复现场

进入程序后，要进行保护现场，将程序中用到的寄存器的值全部压入栈中，然后开中断。在程序将要结束时，关中断然后进行恢复现场。保护现场和恢复现场用到的

堆栈都是在数据存储器选取一部分作为实现。

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

以 ID/EX 段为例进行说明，所有信号都有相应位宽的寄存器进行存储，当时钟上升沿到来时，存储器将值进行传递。流水接口部件如图 3.11 流水接口部件图所示。

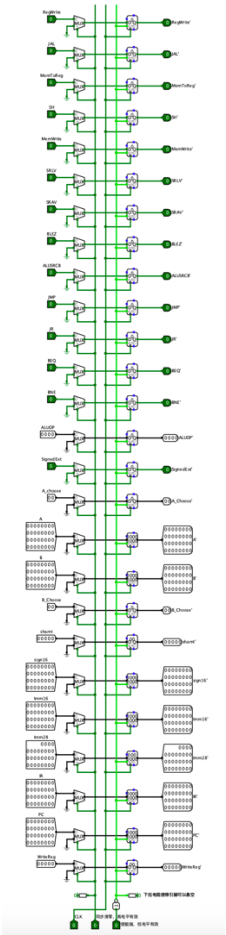

图 3.11 流水接口部件图

3.3.2 理想流水线实现

直接完成气泡流水线，使用气泡流水线测试。

3.4 气泡式流水线实现

对于无条件跳转语句 JMP、J 和 JAL 指令，在 EX 段计算出其目标地址后进行跳转，并将 IF 和 ID 段取出的废指令清空。对于有条件跳转指令，先计算出其目标地址

然后通过运算器 ALU 计算其是否满足跳转条件，若满足条件则跳转并清空 ID 和 IF 段，若不满足条件则流水线继续进行。有条件跳转电路如图 3.12　有条件跳转处理所示。

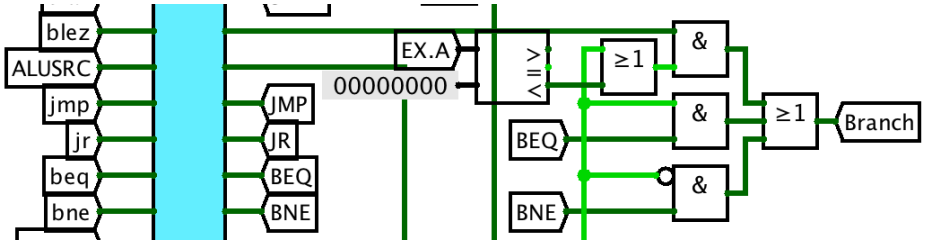

图 3.12　有条件跳转处理

对于顺序执行的流水线其只会产生先写后读的冲突。在该流水线中一共会产生三种数据相关：ID 段读寄存器编号和 EX 段写寄存器编号相同、ID 段读寄存器编号和 MEM 段写寄存器编号相同、ID 段读寄存器和 WB 段写寄存器编号相同。

对于 WB 段的冲突只需要将寄存器改为下降沿写入即可，对于 EX 段和 MEM 段的冲突需要在流水线中插入相应数量的气泡。判断寄存器冲突的逻辑为：首先对于不同指令判断它使用了 1 个还是 2 个寄存器，然后对于 EX 段或 MEM 段的写寄存器编号同 ID 的读寄存器编号想比较，若相同则产生冲突。由于 0 号寄存器恒 0，因此 0 号寄存器不会产生冲突。电路如图 3.13　判断是否产生数据相关所示。

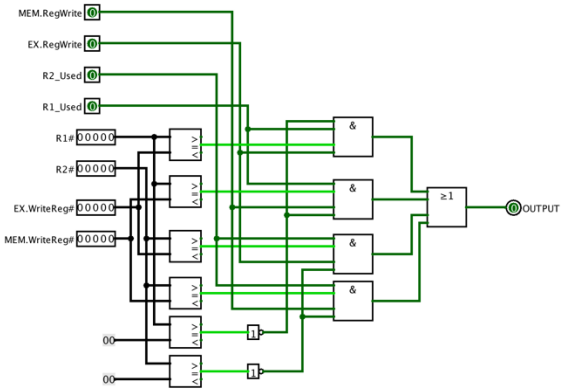

图 3.13　判断是否产生数据相关

3.5 数据转发流水线实现

仅需判断 ID 段的读寄存器与 EX 段的写寄存器编号是否相同，若相同且 EX 段的指令为 LW 或 LBU 指令，则该流水线产生了 LOAD-USE 相关，此时需要在流水线中插入一个气泡。LOAD-USE 检测电路如图 3.14　LOAD-USE 相关检测所示。其中，插入气泡逻辑子电路即判断是否数据相关子电路

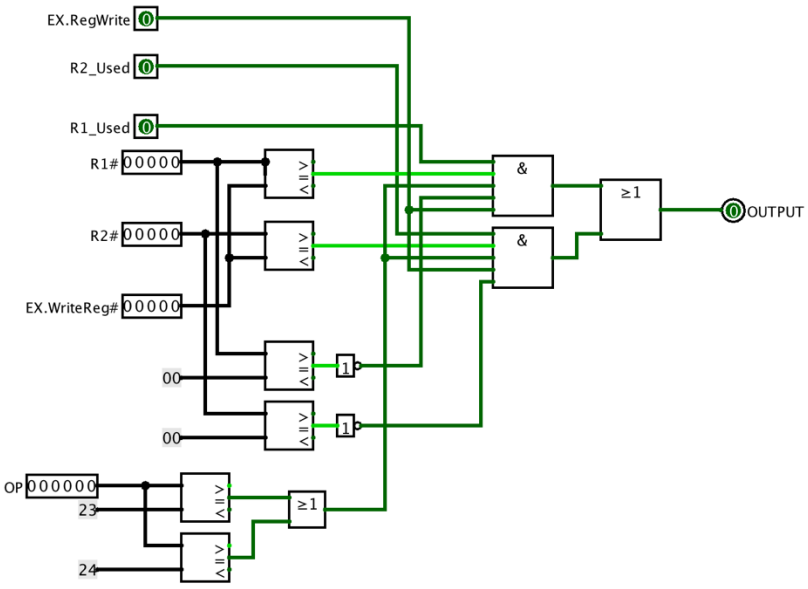

图 3.14　LOAD-USE 相关检测

通过对 ID 段的读寄存器编号与 EX 和 MEM 段的写寄存器编号进行比较，若产生数据相关，通过判断该数据的最新值在哪，然后通过多路选择器传至 ALU 的两端即可。重定向检测逻辑如图 3.15　重定向检测逻辑所示。

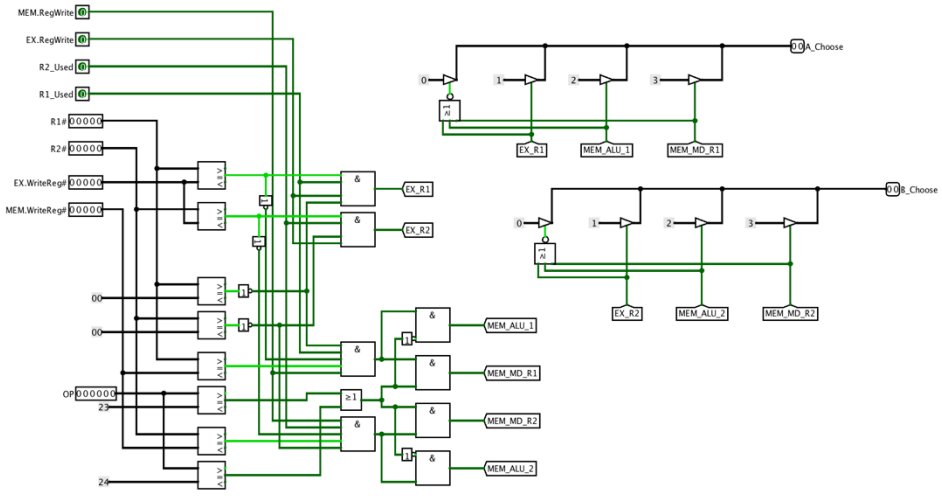

图 3.15　重定向检测逻辑

4 实验过程与调试

4.1 测试用例和功能测试

对各个电路分别进行测试，多级中断程序利用特定的多级中断.hex 进行测试，其他电路都用 benchmark 和四条扩展指令 SRLV、SRAV、SH、BLEZ 进行测试。

4.1.1 单周期 CPU

用 benchmark 标准程序进行测试，最后结果如图 4.1 单周期 benchmark 测试结果所示。

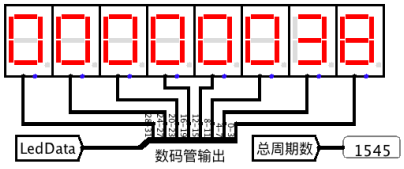

图 4.1 单周期 benchmark 测试结果

4.1.2 理想流水线

用理想流水线测试.hex 进行测试，理想周期数应为 20，实测总周期数为 20，且运行完毕后数据存储器存储的数据如**错误!未找到引用源。**所示，前四个字节分别存有 0、1、2、3。

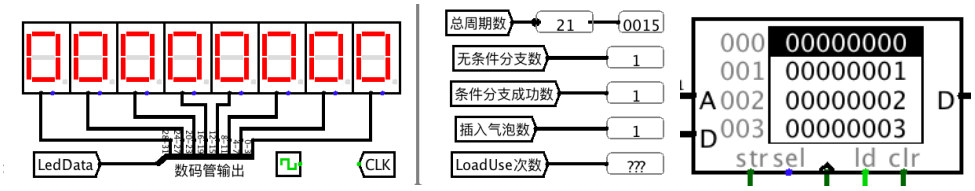

图 4.2 理想流水线测试结果

4.1.3 气泡流水线

用 benchmark.hex 程序对气泡流水线进行测试，预测时钟周期数 3623，实际时钟周期数为 3623。其中存在大量空转时钟周期，插入了 1447 个气泡，性能不好。

用 benchmark 标准程序进行测试，最后结果如图 4.3 气泡流水线 benchmark 测试结果所示。

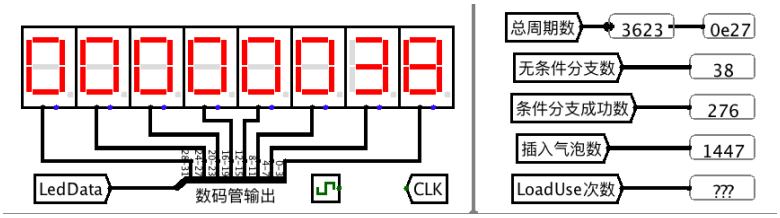

图 4.3 气泡流水线 benchmark 测试结果

4.1.4 重定向流水线

用 benchmark.hex 程序对重定向流水线进行测试，预测时钟周期数 2297，实际时钟周期数为 2297。其中存在少量的空转时钟周期，共插入了 120 个气泡，性能较好。相对于气泡流水线，其数据相关减少了很多插入的气泡数。

用 benchmark 标准程序进行测试，最后结果如图 4.4 重定向流水线 benchmark 测试结果所示。

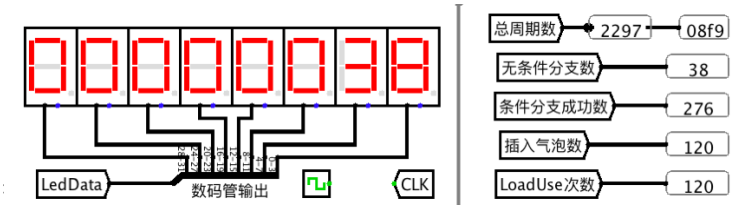

图 4.4 重定向流水线 benchmark 测试结果

4.2 主要故障与调试

4.2.1 中断时寄存器写回错误

单周期单级中断：寄存器写回错误。

故障现象：第三周期时寄存器不写回数据。

原因分析：完成流水线时我直接将 MIPSRegfile 修改为下降沿触发，导致单周期出错。

解决方案：重新封装一个 MIPSRegfile，原有的继续使用上升沿触发，给单周期使用，流水线使用新的下降沿触发。

4.2.2 SH 指令故障

扩展指令：SH 指令故障。

故障现象：单步执行时 SH 并未按照预设情况完成内存写入，始终写入在低字节。

原因分析：完成设计时未对 Addr 低两位进行处理，判断写入在高字节还是低字节。

解决方案：从 Addr 中取出低两位，增加多路选择器，完成数据的选择写入。

4.2.3 BLEZ 故障

扩展指令：BLEZ 故障。

故障现象：执行写回时，Addr 始终为 0。

原因分析：单步执行后，发现 ALUop 在 BLEZ 指令时输出为 0，返回查看硬布线控制器 Excel，找到问题是 BLEZ 指令的 ALUop 未定义。

解决方案：将 BLEZ 指令对应的 ALUop 信号输出设置为 5，重新完成布线。

4.3 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	复习组成原理 CPU 相关理论知识，阅读课设任务书，阅读 MIPS 指令手册，并列出 CPU 各部件的数据通路表，并完成数据通路的基本构建。
第二天	完成单周期 CPU 的控制信号表，使用 Logisim 搭建控制器，实现了单周期 CPU 并且通过了测试。完成部分 Logism 单周期 CPU 故障报告。
第三天	完成 Logism 单周期 CPU 的故障报告，并且通过了 Logism 单周期 CPU 的检查。
第四天	分析重定向流水线需求，着手设计流水接口部件。
第五天	完成流水接口部件设计。
第六天	完成气泡流水线 ccmb 扩展并做测试。
第七天	再次查阅 mooc 相关内容，完成 LoadUse 冲突和重定向逻辑。
第八天	完成重定向流水线，并进行 benchmark 测试。
第九天	与团队成员开会，讨论团队任务实现内容。
第十天	完成分配给自己的团队任务部分，并开始着手测试。
第十一天	将团队任务整合，进行测试联调，解决 bug。
第十二天	发现 ccmb 指令中 SH、BLEZ 指令出错，进行梳理修正，录屏，完成报告。

5 团队项目

5.1 总体设计

实现一个 bf 语言解释器，详细内容请参见当前目录下“bf 解释器文档.pdf”。

5.2 项目分工

陈益欣：完成 bf 解释器除输入输出以外部分的编写

李宇曈：完成 bf 解释器输入输出部分的编写并在 logisim 中增加对应逻辑

李墨池、马腾宇：完成测试样例的编写和测试

5.3 我的实现内容

① 参与讨论任务设计与分工。

② 完成 bf 解释器除输入输出以外部分的编写。此部分请参见当前目录下“bf 解释器文档.pdf”。

③ 协助完成测试联调，解决出现的问题。

④ 完成答辩视频。

5.4 出错分析

在最后的测试联调阶段，我发现我的循环逻辑出现了问题，对于多重循环出口判断存在逻辑漏洞，在李宇曈的协助下我修改了汇编代码，完成了最终版本的 bf 解释器。

5.5 成果展示

见答辩视频。

6 设计总结与心得

6.1 课设总结

基于对象的存储是为了克服当前基于块的存储存在的诸多难题，在存储接口和结构层次的重要发展。可以根据应用负载选择优化的存储策略。作了如下几点工作：

1） 在 Logisim 中完成了单周期 CPU；在单周期 CPU 上增加了中断逻辑，实现了单级中断和多级中断。

2） 实现了理想流水线；在理想流水线中加入了数据相关和分支相关的处理，实现了气泡流水线；对气泡流水线进行了优化，实现了重定向流水线。

3） 完成了团队任务代码的部分撰写工作；完成了团队答辩视频的制作。

6.2 课设心得

本次课程设计收获非常大，每完成一个部分的内容，成就感就不断的累积，整个过程中让我停顿时间最长的就是流水接口部件的设计，我的进度是先完成了不包含 ccmb 的单周期 CPU，然后在此基础上快速实现了理想流水线，当我开始着手实现气泡流水线时，我发现由于数据通路变得更加复杂，导致流水接口部件需要大改，这一定程度上给我带来了很大的挫败感，我停顿了两天，决定一次性完成单周期 ccmb 的所有数据通路，并直接在这基础上完成气泡流水线，构建好数据通路后，我专门花了一天的时间设计了全新的流水接口部件，并预留了重定向流水线需要的接口，这大大提高了我后续完成重定向流水线的效率，可见提前进行规划和整体设计对于完整的系统来说非常重要。

另一个让我体会颇深的就是团队协作，在完成流水线和中断的过程中，我遇到了很多的问题，非常庆幸我的队友帮助我答疑解惑，在此表示感谢。

最后，对本课程设计，我提出一些建议，希望能够加入一些非“内卷”的激励措施，这不论是对个人还是对团队协作都有着非常大的帮助，我的团队“摸鱼”了很长时间，需要不停的去督促才能有较小的完成度，对于个人，这样也可以让大家能够积极的讨论相关内容，而不是单纯的抱怨课程内容繁重（这是“内卷”化的必然结果），群内匿名戾气非常大，我觉得这样不利于教学任务的开展。

最后，再次感谢在这次课程设计中辛勤付出的老师和同学们。

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.

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: