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：	CS1703
学　　号：	U201714668
姓　　名：	葛松
电　　话：	15272052183
邮　　件：	1148690954@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	

#	指令助记符	简单功能描述	备注
13	NOR	或非	
14	LW	加载字	
15	SW	存字	
16	BEQ	相等跳转	
17	BNE	不相等跳转	
18	SLT	小于置数	
19	STI	小于立即数置数	
20	SLTU	小于无符号数置数	
21	J	无条件转移	
22	JAL	转移并链接	
23	JR	转移到指定寄存器	If $v0==10 halt(停机指令)
24	SYSCALL	系统调用	else 数码管显示$a0 值
25	MFC0	访问 CP0	中断相关，可简化，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	SLLV	逻辑可变左移	
29	SRAV	算术可变右移	
30	LBU	加载字节(无符号)	
31	BGEZ	大于等于 0 转移	

2 总体方案设计

2.1 单周期 CPU 设计

单周期 CPU 中，我们几乎是沿用了上学期的 CPU，采用了硬布线的方式，数据存储器和指令存储器分开。在上学期的基础上，对 24 条基本指令以及额外的 4 条拓展指令提供了支持。核心部件为单周期硬布线控制器。在以给出的元件基础上，通过 Excel 表格添加了对拓展指令的支持，将拓展指令作为统一的集合固化在控制器中。由于前期的设计问题，所以在单周期 CPU 中拓展指令并没有通过控制器生成相关的控制信号，而只是对拓展指令进行了识别，控制信号采用了外围手动添加的方式。后续的流水线会对其进行改进。完成电路后通过导入 benchmark 文件进行测试。

总体结构图如图 2.1 所示。

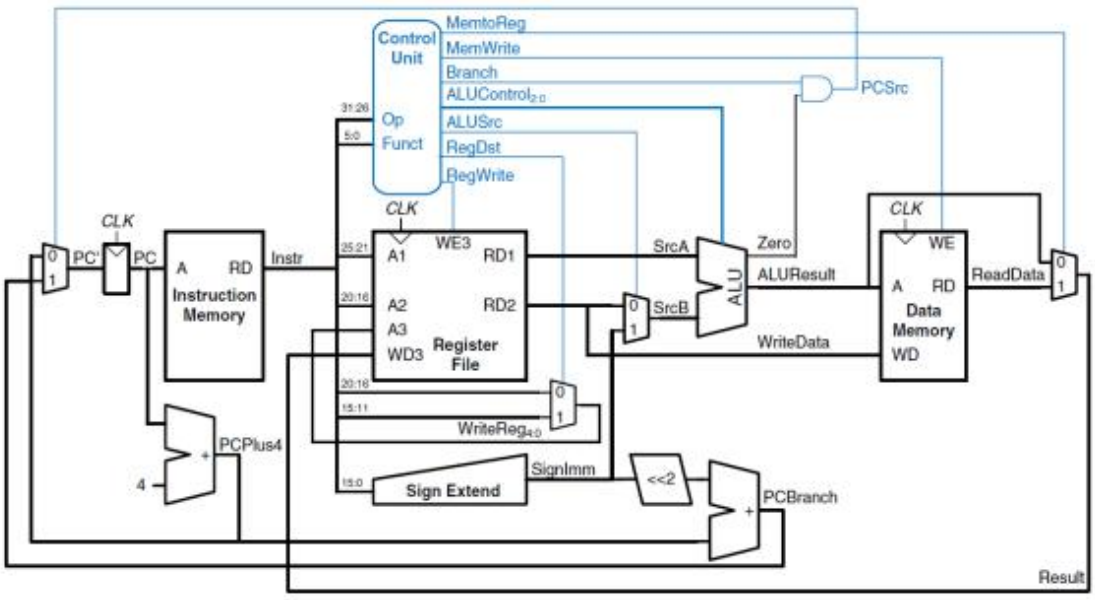

图 2.1 总体结构图

2.1.1 主要功能部件

主要的功能部件有程序计数器、指令存储器、寄存器堆、运算器、、数据存储器等。

1. 程序计数器 PC

程序计数器主要用于提供下一条指令的地址，随时钟推动整个 CPU 的运行，其取值在实际运行中可能来自于多个地方，如原本的 PC 值加 4，表示当前指令的下一条地址，或是 JMR, JR 等跳转指令所计算出来的跳转地址，需要对不同的情况加以判断和选择。具体实现时采用上跳沿触发的寄存器。

2. 指令存储器 IM

指令存储器主要用于保存需要执行的指令本身，由于是程序，为不可写性质的存储，所以在 logisim 中采用 ROM(只读存储器)来实现，并配合 PC 值取出当前需要执行的指令。需要载入新的程序时，需要手动载入。

3. 运算器

运算器用于实际执行运算，数据来源于指令中的立即数或从 regfile 中获取的寄存器值，输出将用于写入数据存储或者作为地址访问数据存储。

表 2.1 算术逻辑运算单元引脚与功能描述

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

4. 寄存器堆 RF

寄存器堆由多个寄存器组成。在该实验中使用的是资源包 CS3410 中康奈尔大

学封装好的 MIPS regfile，根据 MISP 的格式，提供了 32 个通用寄存器，配合具体指令使用。

5. 数据存储器

数据存储器作为内存来使用，存放一般数据。在 logisim 中具体使用 RAM 来实现，由于需要同时支持读和写。

2.1.2 数据通路的设计

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
ADD	PC+4	PC	RS	RT	RD	ALU	R1	R2	5	--	--
ADDU	PC+4	PC	RS	RT	RD	ALU	R1	R2	5	--	--
ADDI	PC+4	PC	RS	--	RT	ALU	R1	RD(EXT)	5	--	--
ADDIU	PC+4	PC	RS	--	RT	ALU	R1	RD(EXT)	5	--	--
AND	PC+4	PC	RS	RT	RD	ALU	R1	R2	7	--	--
ANDI	PC+4	PC	RS	--	RT	ALU	R1	RD(ZXT)	7	--	--
SLL	PC+4	PC	RS	RT	RD	ALU	R1	R2	0	--	--
SRA	PC+4	PC	RS	RT	RD	ALU	R1	R2	1	--	--
SRL	PC+4	PC	RS	RT	RD	ALU	R1	R2	2	--	--
SUB	PC+4	PC	RS	RT	RD	ALU	R1	R2	6	--	--
OR	PC+4	PC	RS	RT	RD	ALU	R1	R2	8	--	--
ORI	PC+4	PC	RS	--	RT	ALU	R1	RD(ZXT)	8	--	--
NOR	PC+4	PC	RS	RT	RD	ALU	R1	R2	10	--	--
LW	PC+4	PC	RS	--	RT	DW	R1	RD(EXT)	5	ALU	R2
SW	PC+4	PC	RS	RT	RT	--	R1	RD(EXT)	5	ALU	R2
BEQ	RF/PC+4	PC	RS	RT	--	ALU	R1	R2	6	--	--
BNE	BF/PC+4	PC	RS	RT	--	ALU	R1	R2	6	--	--
SLT	PC+4	PC	RS	RT	RD	ALU	R1	R2	11	--	--
SLTI	PC+4	PC	RS	RD	RT	ALU	R1	RD(EXT)	11	--	--

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
SLTU	PC+4	PC	RS	RT	RD	ALU	R1	R2	12	--	--
J	IM	PC	--	--	--	--	--	--	--	--	--
JAL	IM	PC	--	--	常数	PC+4	--	--	--	--	--
JR	RFR1	PC	RS	--	RD	--	--	--	--	--	--
SYSCALL	--	PC	常数	常数	RD	--	R1	R2	--	--	--
MFC0	--	PC	--	--	--	ALU	R1	R2	--	--	--
MTC0	--	PC	--	--	--	ALU	R1	R2	--	--	--
ERET	--	PC	--	--	--	ALU	R1	R2	--	--	--
SLLV	PC+4	PC	RS	RT	RD	ALU	R1	R2	0	--	--
SRAV	PC+4	PC	RS	RT	RD	ALU	R1	R2	1	--	--
LBU	PC+4	PC	RS	--	RT	DW	R1	RD(EXT)	5	ALU	R2
BGEZ	RF/PC+4	PC	RS	常数	--	ALU	R1	常数	11	--	--

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	信号说明	产生条件
RegWrite	寄存器写使能	寄存器写回信号
MemWrite	写内存控制信号	sw 指令,未单独设置 MemRead 信号
AluOP	运算器操作控制符(4 位)	R 型指令根据 Func 选择
MemToReg	寄存器写入数据来自存储器	lw 指令
RegDst	写入寄存器编号 rt/rd 选择	R 型指令
AluSrcB	运算器 B 输入选择	lw 指令,sw 指令,立即数运算类指令
SignedExt	立即数符号扩展	ADDI、ADDIU、SLTI 指令

控制信号	信号说明	产生条件
JR	寄存器跳转指令译码信号	JR 指令
JAL	JAL 指令译码信号	JAL 指令 ,选择寄存器写回编号,写回值
JMP	无条件分支控制信号	J、JAL、JR 指令,选择无条件分支地址
Beq	Beq 指令译码信号	Beq 指令,用于有条件分支控制
Bne	Bne 指令译码信号	Bne 指令,用于有条件分支控制
Syscall	Syscall 指令译码信号	根据$V0 寄存器的值,决定是停机还是输出
SLLV	Sllv 指令译码信号	Sllv 指令
LBU	Lbu 指令译码信号	Lbu 信号
BGEZ	Bgez 指令译码信号	Bgez 信号
SRAV	Srav 指令译码信号	Srav 信号

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。其中新增了对拓展指令的判断，并给出了相关的信号，方便后续处理

表 2. 4 主控制器控制信号框架

指令	ALU_OP	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL	SLLV	SRAV	BGEZ	LBU	R1_Used	R2_Used
SLL	0				1			1											1
SRA	1				1			1											1
SRL	2				1			1											1
ADD	5				1			1										1	1
ADDU	5				1			1										1	1
SUB	6				1			1										1	1
AND	7				1			1										1	1
OR	8				1			1										1	1
NOR	10				1			1										1	1

SLT	11				1			1										1	1
SLTU	12				1			1										1	1
JR											1	1						1	
SYSCALL						1												1	1
J												1							
JAL					1							1	1						
BEQ							1		1									1	1
BNE							1			1								1	1
ADDI	5			1	1		1											1	
ANDI	7			1	1													1	
ADDIU	5			1	1		1											1	
SLTI	11			1	1		1											1	
ORI	8			1	1													1	
LW	5	1		1	1		1											1	
SW	5		1	1			1											1	1
SLLV	0				1			1						1				1	1
SRAV	1				1			1							1			1	1
LBU	5	1		1	1		1										1	1	
BGEZ	11						1									1		1	

2.2 中断机制设计

2.2.1 总体设计

中断分为外部中断和内部中断,本次实验中只考虑可屏蔽的外部中断。对于外部中断机制的设计,需要考虑的又有硬件实现的中断支持和软件实现的中断处理。首先考虑单级中断的实现,由于执行中断程序完毕后需要能够返回进入中断的位置并恢复整个 CPU 的状态,所以在进入中断时首先保存 CPU 的状态,即需将 PC 的值保存在 EPC 中,并在退出中断时将 EPC 的值返回给 PC 寄存器。在流水中断中要考虑传入的 PC 值是哪个阶段的。实现多级中断时,由于多级中断存在优先级的问题,除了实现存储 PC 的堆栈外,还需要对不同等级的中断进行处理,由硬件进行判

断新的中断能否打断旧的中断。在退出多级中断时,也需要判断继续执行一条被打断的低级中断还是开始执行一个已经在排队的低级中断。注意此时的 PC 保存要采用堆栈的形式。

2.2.2 硬件设计

单级中断的硬件相对简单,使用中断按键参考电路输入相应的中断信号,在同时有多个中断信号的时候使用优先编码器选择出最先执行的中断号。保存 PC 的 EPC 使用普通的寄存器,将发生中断时 PC 中的值保存下来,在中断程序结束退出中断时会有 ERET 指令,此时将 EPC 寄存器中的值重新送到 PC 寄存器中复位,另外使用解码器结合中断号可以将中断信号复位。还需要注意,EPC 寄存器中的值只有当中断发生时才需要更新,其他时候不需要也不能进行更新操作。PC 寄存器的输入值也需要对中断发生信号和 ERET 指令进行判断,以输入中断处理程序地址和复位的 EPC。此过程需要结合扩展的 ERET 指令进行。在流水中断中,还需要考虑送入 EPC 寄存器中的 PC 是哪一阶段的 PC 值,避免指令超前或者滞后的问题。

多级中断比较复杂,由于可能会有多个有优先级的中断反复发生,必须有相应结构存储每次中断发生时的相应信息。首先,相比于单级中断发生时的一个 EPC 寄存器,针对此次实验中的三级中断设置 3 个 EPC 寄存器,分别存放至多 3 个中断都发生时的 PC 值,另外也需要比较器和寄存器结合,判断当有中断处理程序在执行时如果发生新的中断,应该执行哪个中断,并且将正在执行的中断号和被打断的终端号都保存在相应寄存器中,在每次执行中断时更新这些寄存器中的值重新进行判断。在多级中断中,还用到了开关中断指令 mfc 和 mtc,结合中断发生信号和 ERET 指令来进行是否发生中断的判断,也需要修改控制器逻辑使之支持这些指令。中断信号的复位电路和单级中断思路类似。

2.2.3 软件设计

对于中断程序的具体设计而言,主要需要考虑保存寄存器的值。通过编写程序来保护现场,即进入中断时将寄存器压入堆栈,并在中断返回前从堆栈中弹出寄存器的值来保护所有在中断处理程序中需要用到的寄存器。由于 PC 的值已经通过硬件实现的堆栈进行保存,在程序中无需考虑 PC 的压栈。同时，由于使用了自动测试的逻辑，在判断中断时是直接对写在测试程序中的中断地址来实现的。

2.3 流水 CPU 设计

2.3.1 总体设计

针对理想流水线，我们不需要考虑数据冲突以及跳转指令等问题，但理想流水线是后续气泡流水线以及重定向流水线的基础，在理想流水阶段设计的流水部件在后续的流水中也会用到。而后续的气泡则是使用插入气泡的方法解决冲突问题，重定向则是尝试使用重定向的方式解决冲突，减少插入气泡的次数，提高效率。流水的不同段使用流水接口部件进行连接，完成信号的缓存与传递。

2.3.2 流水接口部件设计

流水接口部件实际上大部分参考了上学期的字库实验中所用到的流水部件，唯一的区别就是接口个数不同。每一个流水部件包含：时钟、使能端、清零端、数据输入端以及数据输出端。同时由于设计上的问题，可能对一些接口实现了复用，由于需要传递的数据是随着流水线的传递逐渐减少的。一些后续没有使用的端口可能会被用来传输中途新的数据，如 ALU 的计算结果等。

2.3.3 理想流水线设计

在完成了对于流水接口的设计后,只需要考虑每一条指令的数据通路并将对应的信号连接到对应的模块端口即可。需要注意在取得控制信号时,要由对应的流水段取出,如不能将 ID 段的 RegWrite 信号与 WB 段的数据 Data 同时向 RegFile 输入,否则会形成错误的逻辑。

2.4 气泡式流水线设计

2.4.1 总体设计

在理想流水线中我们假设指令之间没有数据冲突。每一条指令的执行相互不影响。但在实际的程序中，各种相关都非常普遍。而气泡流水线通过先检测相关，再插入气泡的方式，使得程序可以正常执行

2.4.2 数据相关模块设计

我们将数据相关元件封装在一个元件中，通过输入#R1，#R2，Mem.WriteReg#，MEM.WriteReg 等信号，可以输出是否相关的 Correlated 信号。从而判断是否需要插入气泡

2.4.3 气泡流水线设计

在理想流水线的基础上,需要进行如下修改:

1. PC 寄存器的使能端口加上“数据相关”信号的判断逻辑以保存发生数据相关时的 PC 状态

2. 修改每个流水接口的使能输入以及清零输入,使发生数据相关时能够插入气泡或者清零

3. 增加数据相关模块,通过“源寄存器使用情况”子模块和“数据相关检测”子模块的实现检测是否有数据冲突发生,生成“数据相关”信号。具体的实现为:若信号发生则清零 ID/EX 流水接口并暂停 IF/ID 以及 PC 模块,发生跳转指令时也 需要将 IF/ID 和 ID/EX 流水接口清零。完成上述工作后流水接口理论上能够处理所有的指令,并能够正确运行 benchmark 程序。

2.5 重定向流水线设计

2.5.1 总体设计

为了解决气泡流水线插入气泡过多导致流水线性能下降的问题,修改气泡流水线的逻辑,在发生数据冲突时不再进行插入气泡的操作,而是在下一个时钟周期将尚未写回至寄存器堆的数据直接重定向至需要数据的 EX 段中。

需要注意的是重定向流水线会引入 LoadUse 问题,即在一条 load 指令之后若紧随有一条需要使用该寄存器的指令,不能直接将数据存储器 DM 的输出重定向到 EX 段中,因为这样会使整个 CPU 的关键路径包含访存,从而关键路径变长影响 CPU 整体的工作效率。为解决此问题,需要在 Forward 模块中添加对 LoadUse 情况进行判断的逻辑,在检测到该情况发生时,向 EX 段添加一个气泡,便能消除 LoadUse 现象。

2.5.2 LoadUse 检测模块设计

由于在重定向流水线中对数据相关检测模块的功能需求进行了扩展,需要将其改造为 Forward 模块。具体而言,在 Forward 器件中对 R1 冲突和 R2 冲突分别设计判断电路,同时对每个寄存器的冲突而言,依据与 EX 发生冲突的流水段和信号不同,对冲突结果进行编码,从而产生两位的 RD1st 和 RD2st 信号用于控制 ALU 两个输入端的信号分别源自何处。以 RD1st 为例,若在 ID 段发现与 EX 段存在数据冲突则 RD1st==1, ALU 输入端 A 的输入数据由 EX/MEM 流水接口中的 Aluout 重定向而来;若在 ID 段发现与 MEM 段存在数据冲突且 MemToReg==0 则 RD1st==2, ALU 输入端 A 的输入数据由 MEM/WB 流水接口中的 Aluout 重定向而来,否则若 MemToReg==1 则 RD1st==3,ALU 输入端 A 的输入数据由 MEM/WB 流水接口中的 Memout 重定向

而来;其余情况没有产生数据冲突,直接将 ID/EX 流水接口中的寄存器取值作为输入即可。此外,若在 Forward 模块中检测到 ID 与 EX 段出现数据冲突且 EX 段中指令种类为 Load,则生成 LoadUse 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）

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，接入 PC 寄存器的使能端，屏蔽时钟信号，使整个电路停机。如图 3.1 所示。

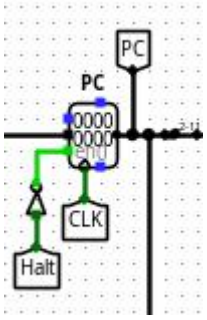

图 3.1 程序计数器（PC）

2) 指令存储器（IM）

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。

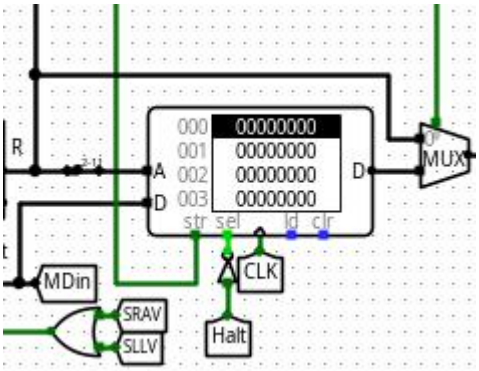

图 3.2 指令存储器（IM）

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。

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，在完成指令系统数据通路表的填写之后，根据列出的数据通路表，进行多指令数据通路的合并输入数，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。如图 3.3 所示

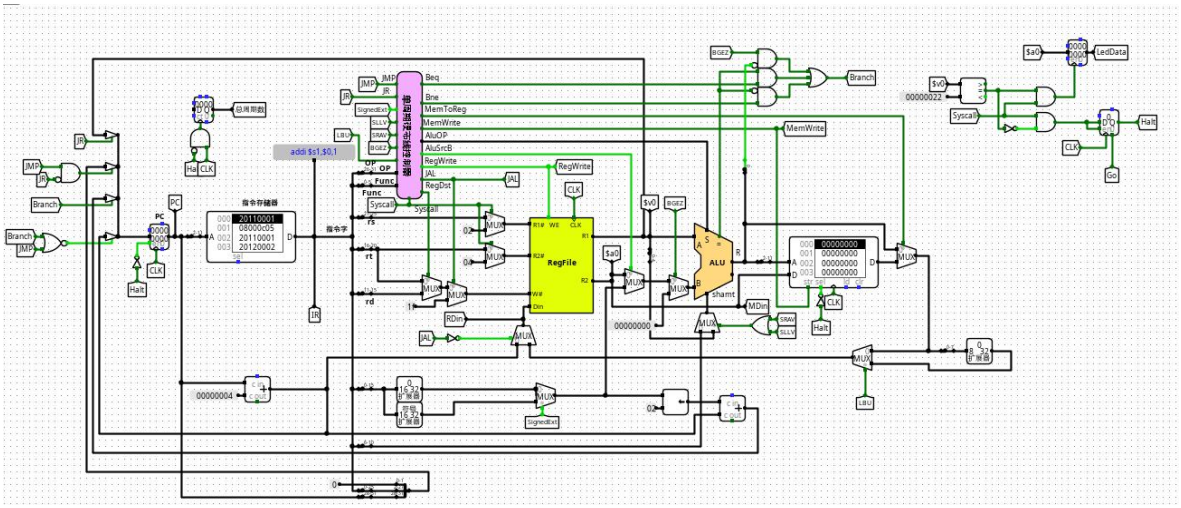

图 3.4 单周期 CPU 数据通路（Logism）

3.1.3 控制器的实现

根据总体方案设计中控制器的设计的相关内容,在 Logism 上进行主控制器、Branch 控制器、SYSCALL 控制器的实现,在具体实现中,单周期控制器分为运算器控制器和控制信号生成两个子模块,前者根据指令输出 ALU_OP 的功能选择输入,后者同样是根据指令生成各个控制信号,具体如表 2.4 所示

3.2 中断机制实现

3.2.1 单级中断实现

(1) 中断按键信号产生电路

此电路统一给出,实际使用中输入中断信号,输出相应中断信息如图 3.5 所示。

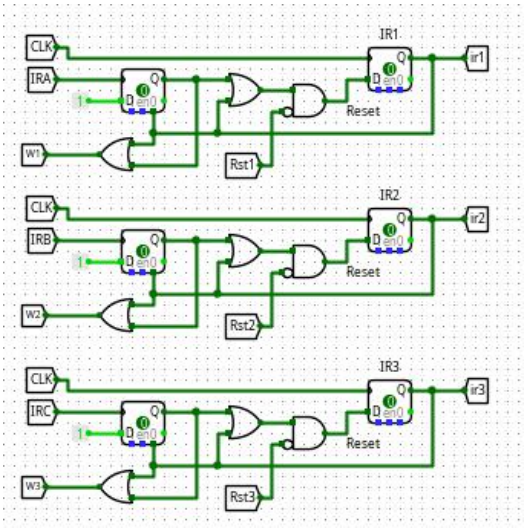

图 3.5 中断按键信号产生电路

(2) 中断地址选择电路

因为有三个不同的中断,在一个中断发生时要根据此时具体的中断选择中断处理程序地址,使用优先编码器决定目前执行的中断号,利用多路选择器选择出地址如图 3.6 所示。由于测试程序中直接给出了中断地址，所以被硬编码到电路中。

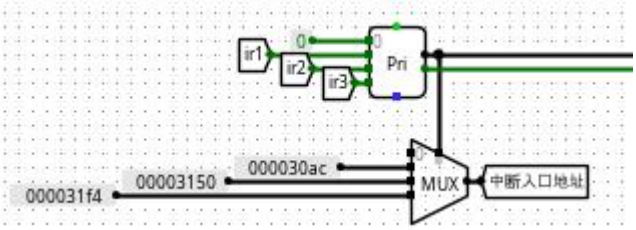

图 3.6 中断处理程序地址选择电路

(3) 中断信号复位电路

当中断处理完毕后,要将中断信号复位,具体根据 ERET 指令结合当前执行中断的中断号判断,将具体的某一个中断信号复位,其中中断号寄存器的使能端输入为是否有中断发生的信号,当有中断发生时不进行复位,不然中断信号一产生就被复位,如图 3.7 所示。

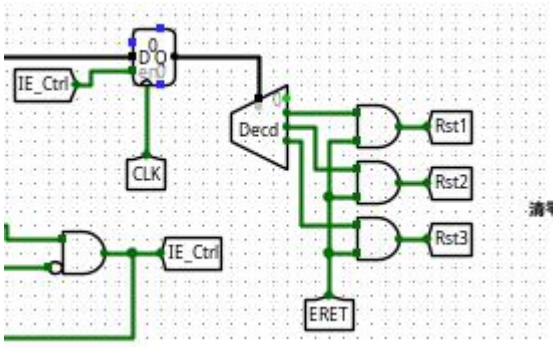

图 3.7 中断信号复位电路

(4) EPC 寄存器及中断使能 IE 寄存器电路

在有中断发生时要将 PC 寄存器的值(toPC)保存起来,在中断结束时取出,同时 IE 寄存器保存当前中断的相关状态,以方便判断是否有中断发生,如图 3.8 所示,在有中断发生时才需要保存 PC 的值,在有中断发生或者中断处理程序完成返回时更新 IE 的值。

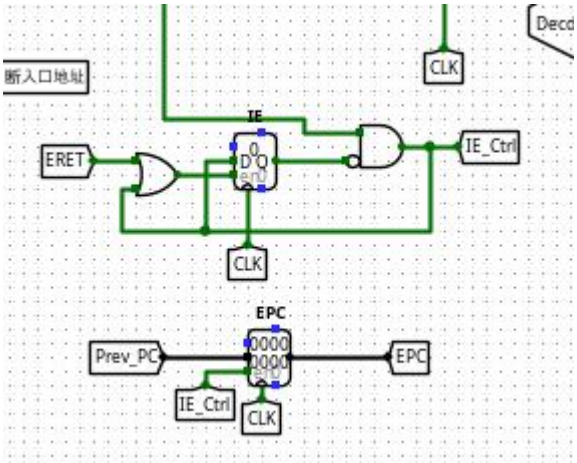

图 3.8 EPC 及 IE 电路

(5) PC 输入电路

在 PC 输入端,一方面要在中断发生时选择合适的值送入 EPC 中,另一方面要选择合适的值送入 PC,如图 3.9 所示。

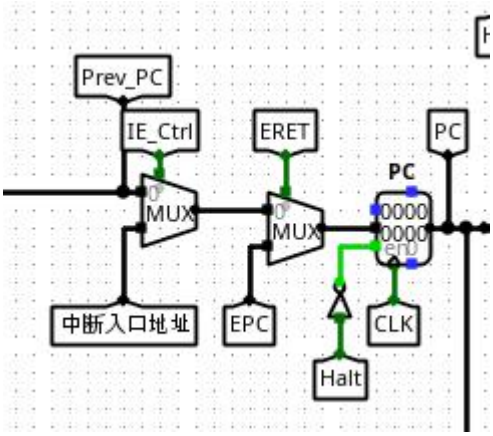

图 3.9 PC 输入选择电路

3.2.2 多级中断实现

多级中断中,最主要的变化在于多个中断信号发生时的中断信号选择以及每一级信号的现场保护所需要的 EPC 寄存器构成的硬件堆栈逻辑,同时 IE 电路逻辑也与单级中断有所不同,因为引入了 MTC 和 MFC 这两条指令,IE 寄存器电路和单周期控制器的电路也需要进行相应的修改,扩展这两条指令。其余的 PC 输入、中断按键信号产生电路和中断信号复位电路与单级中断类似。

(1) EPC 寄存器堆栈

因为是多级中断,当一个中断发生时要根据寄存器内现有的中断号进行判断,进而决定是继续执行还是执行新的中断处理程序,有因为有三级中断,所以设置三个寄存器存中断号,同时要注意中断复位信号和中断发生信号发生时中断号的更新,具体电路如图 3.10 所示。

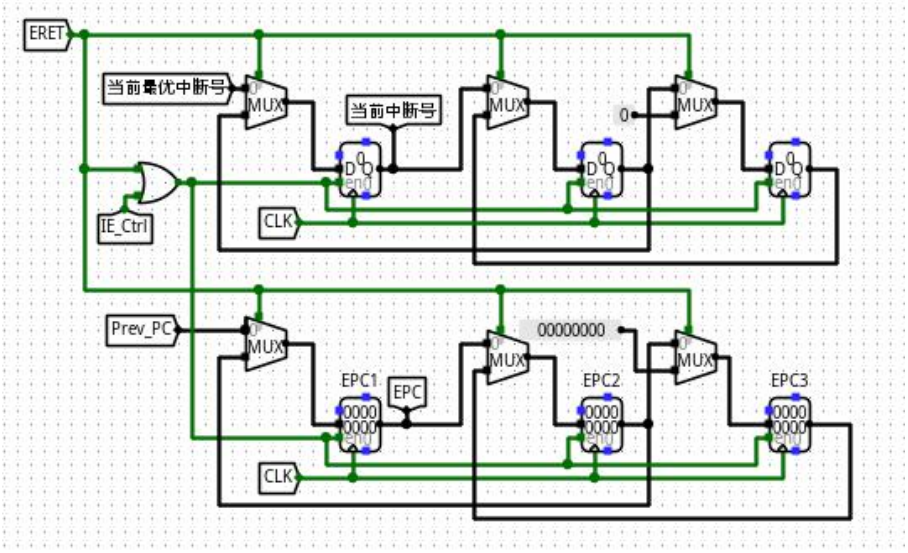

图 3.10 使用硬件堆栈保护现场

(2) 开关中断

开关中断通过 MFC0、MTC0 指令直线，如图 3.11 所示

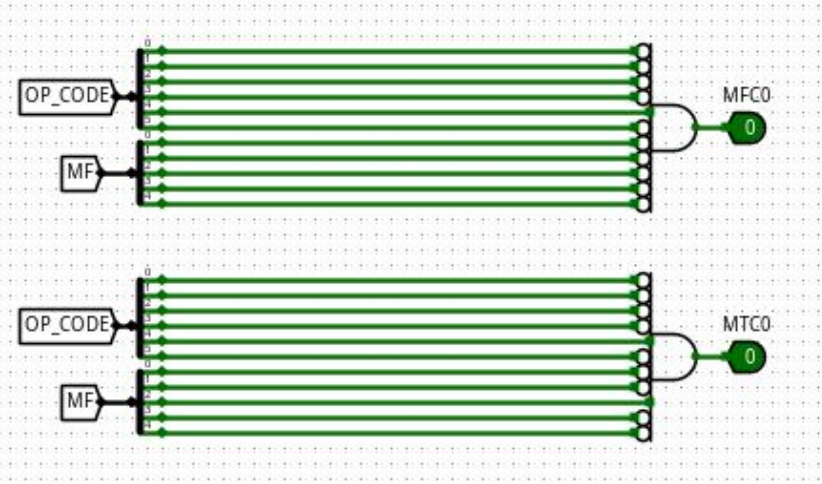

图 3.11 MFC0、MTC0 生成电路

实现开关中断的逻辑如图 3.11 所示

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水接口负责通过时钟输入在不同的执行阶段之间传递数据，进行缓存，并能够通进行插入气泡 以及清零等基本操作。具体的实现参考了字库实验中所用到的流水部件。本次实验中共有 5 个接口部件，对应 CPU 五段操作之间的传递过程。

以其中最复杂的 ID/EX 流水接口为例，内部布线如图 3.12 所示。其中包括了由控制器给出的控制信号，以及个人拓展指令所对应的指令信号。

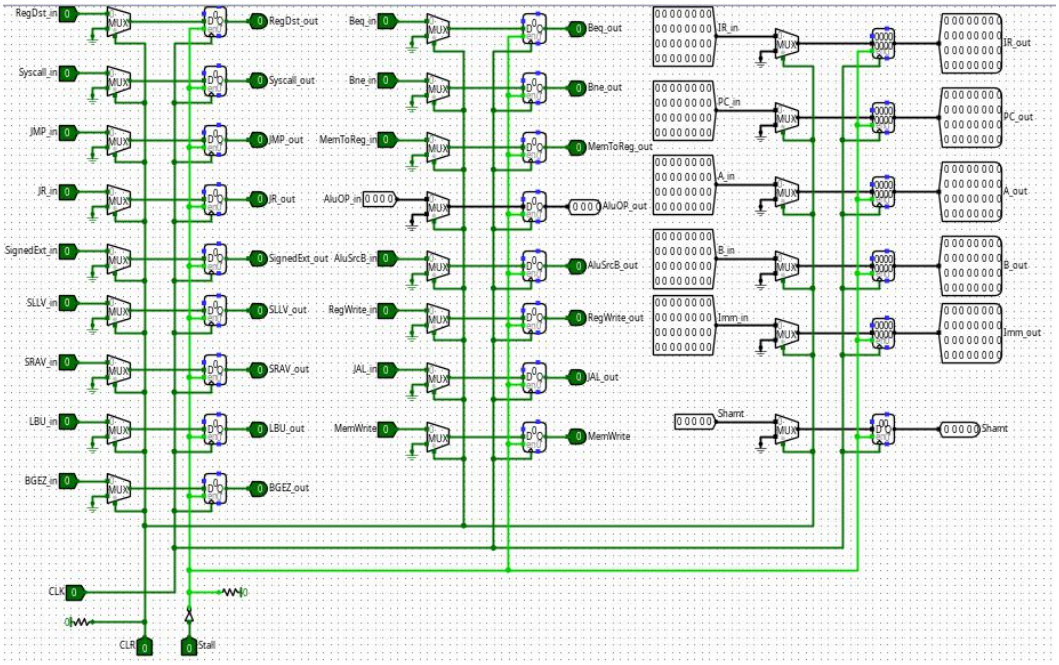

图 3.12 ID/EX 内部布线图

3.3.2 理想流水线实现

根据 2.3 中理想流水线的设计,向单周期 CPU 中新增流水接口,并对单周期数据通路修改连接使其成为流水线数据通路。重新连接数据通路后的理想流水线 Logisim 电路图如图 3.13 所示。

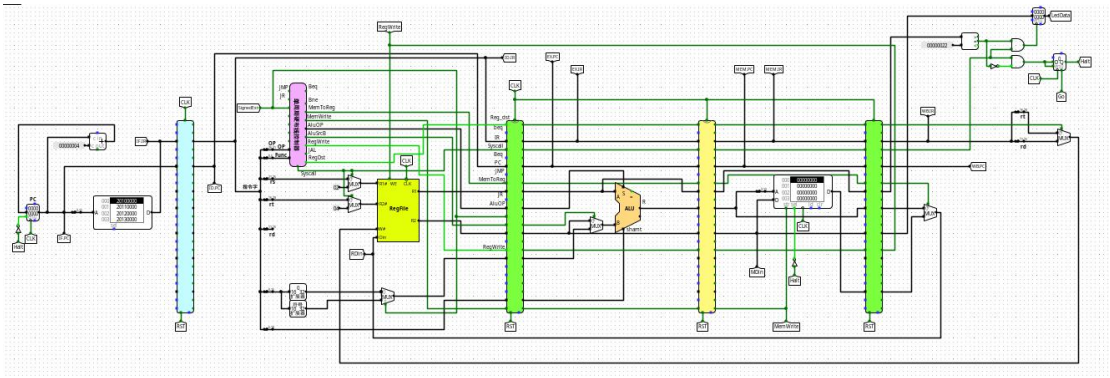

图 3.13 理想流水线电路图

3.4 气泡式流水线实现

3.4.1 数据相关信号实现

我们在实验中将数据相关逻辑封装在一个单独的组件中，同时进行了源寄存器冲突以及数据冲突两种情况，电路如图 3.14 所示

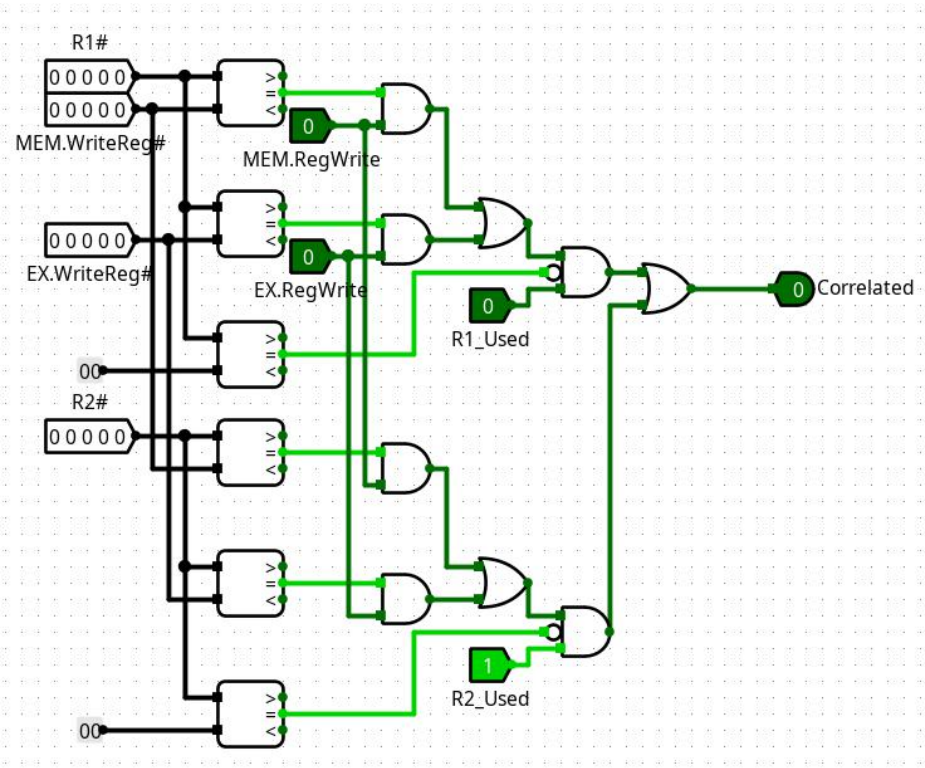

图 3.14 数据相关检测逻辑

而 R1_Used 以及 R2_Used 信号则在外部手动生成如图 3.15 所示

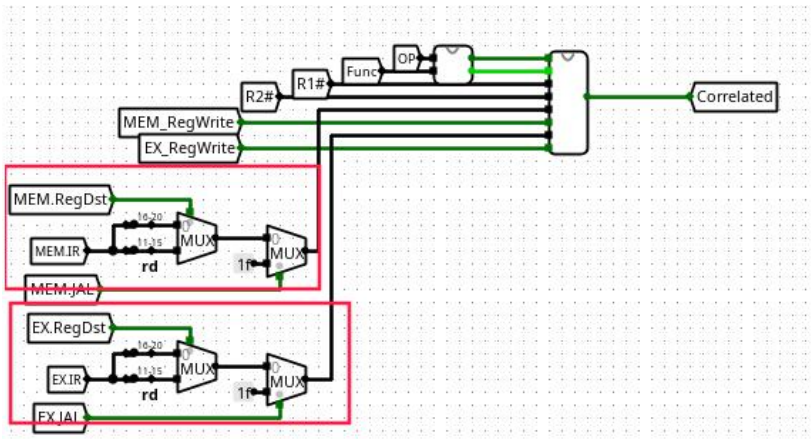

图 3.15 R1_Used 以及 R2_Used 信号

3.4.2 气泡流水线实现

气泡流水线的整体电路图如所示

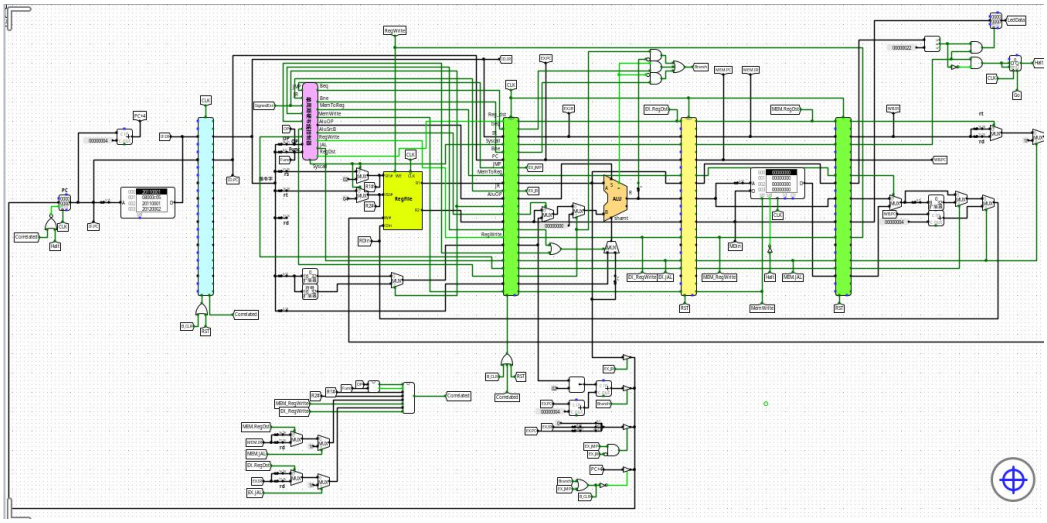

图 3.16 气泡流水线电路图

当发生冲突的时候需要暂停 PC 寄存器，并对相应的接口部件进行暂停以及插入气泡等操作。

3.5 重定向流水线实现

由于采用了重定向技术，所以对 ALU 的两个输入端增设了两个多路选择器，当检测到符合要求的冲突时就将 ALU 的输入替换为对应的数据输入，如在 MEM 断的 ALU 输入（上一次 ALU 计算的结果）。其中注意会对 LoadUse 冲突进行处理。相应的检测电路如所示

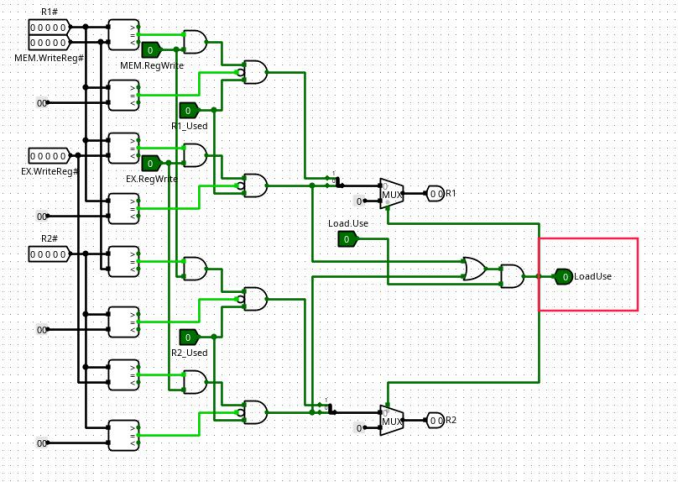

图 3.17 冲突检测以及 LoadUse 生成

综合上述检测电路以及 ALU 输入的重定向电路，最终的重定向电路图如图 3.18 所示

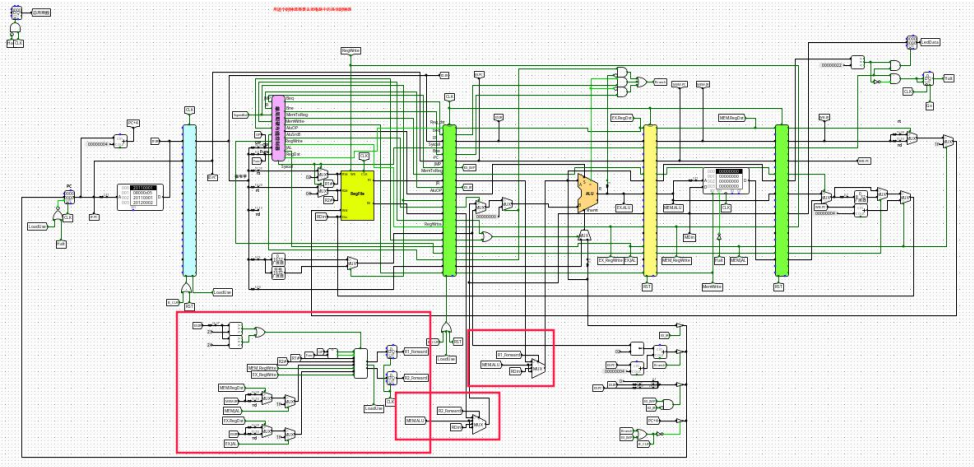

图 3.18 重定向电路图

4 实验过程与调试

4.1 测试用例和功能测试

4.1.1 单周期 CPU 测试

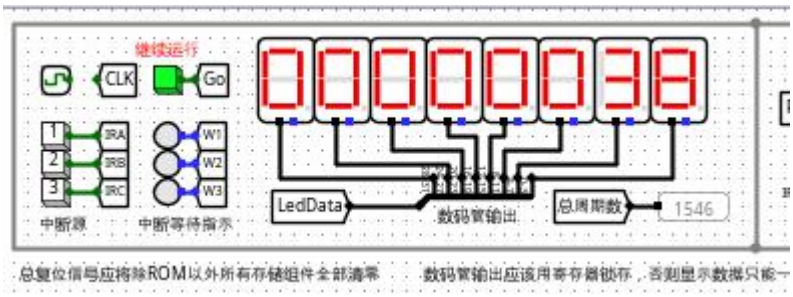

图 4.1 单周期 CPU 测试

4.1.2 个人指令测试

个人指令共有 4 条，分别为 SLLV、SRAV、LBU 以及 BGEZ

(1) SLLV 测试截图如图 4.2 所示

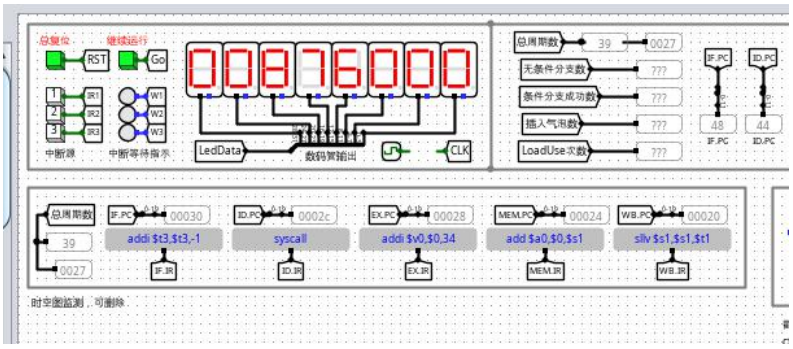

图 4.2 SLLV 测试

SLLV 测试效果是数字”876”从右到左左移，截图是移动的中间过程。

(2) SRAV 测试截图如所示

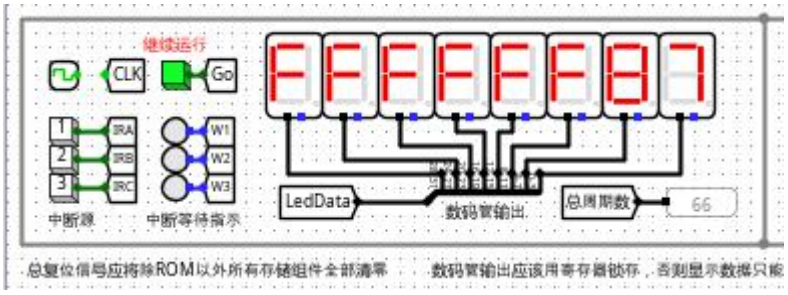

图 4.3 SRAV 测试

(3) LBU 测试

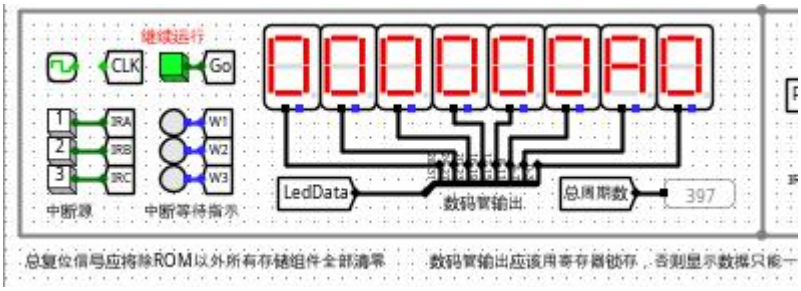

图 4.4 LBU 测试

(4) BGEZ 测试

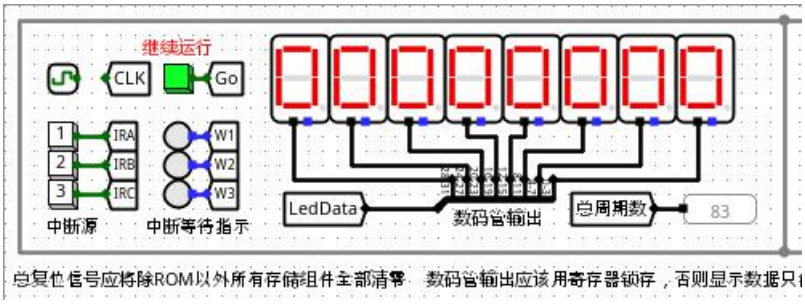

图 4.5 BGEZ 测试

最终输出为 0，与预测相符

4.1.3 中断测试

(1) 单级中断测试

向指令存储器中导入单级中断测试程序，将会依次发生 1，2，3 号中断，并依次执行 1，3，2 号中断，执行过程如图 4.6 所示

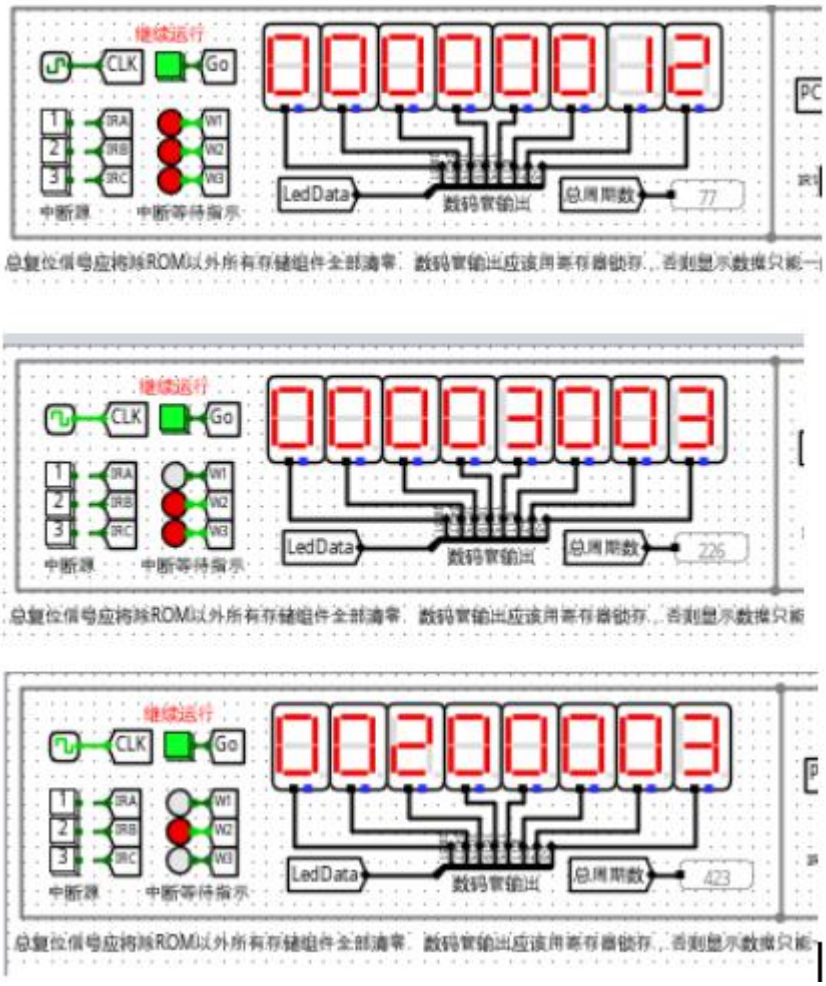

图 4.6 单级中断测试

(2) 多级中断测试

同样导入自动测试程序，运行截图如图 4.7 所示

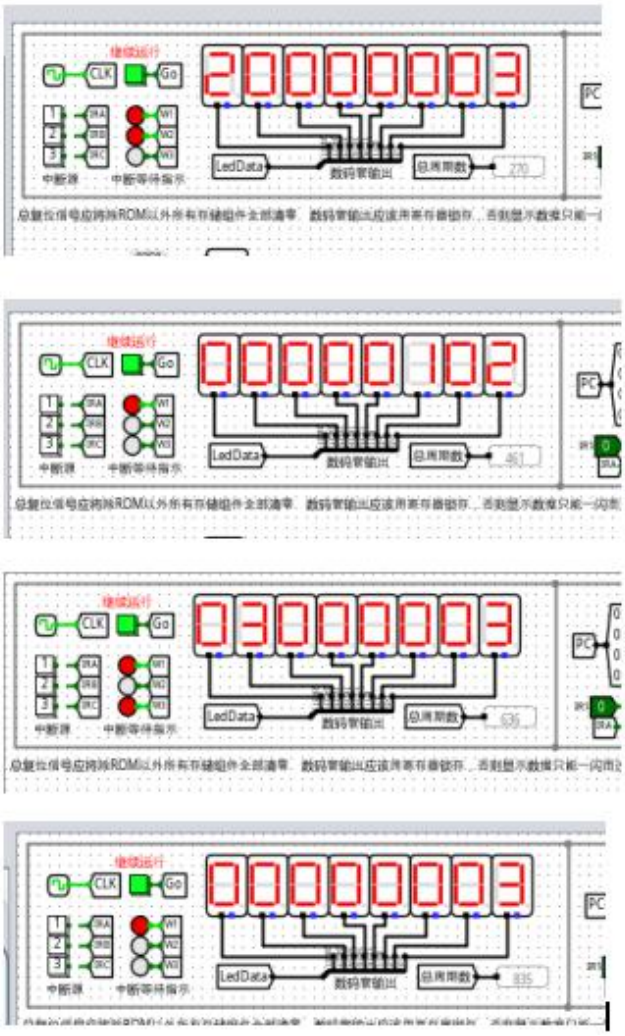

图 4.7 多级中断测试

4.1.4 流水线测试

(1) 理想流水线

运行理想流水线测试，运行完成后数据存储器内容如图 4.8 所示

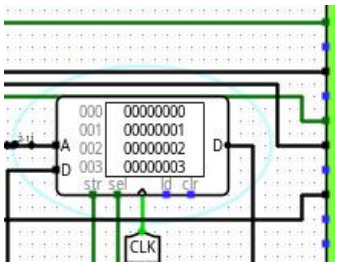

图 4.8 理想存储器测试

(2) 气泡流水线

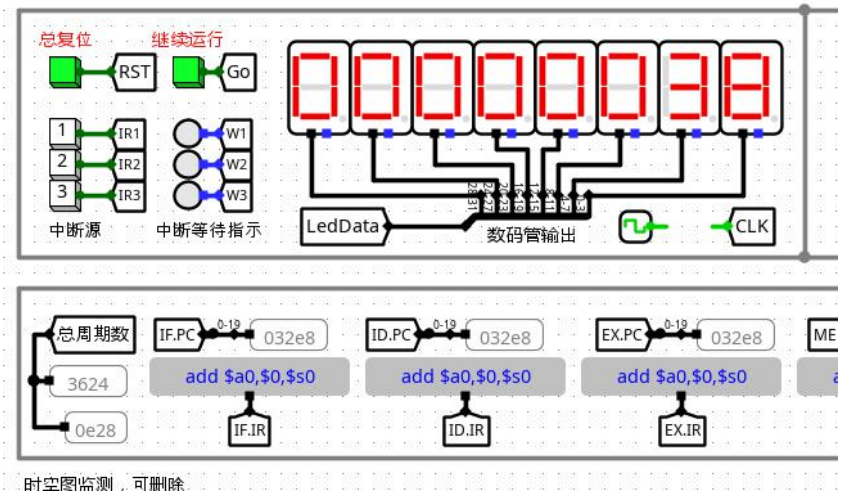

图 4.9 气泡流水线测试

(3) 重定向流水线

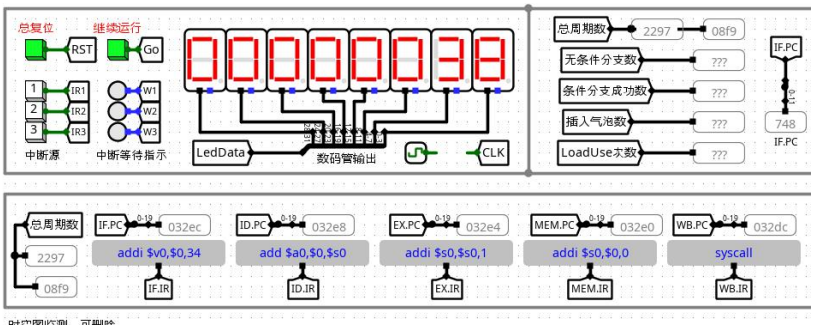

图 4.10 重定向流水线

上述所有电路均通过了 educoder 在线测试，如图 4.11 所示

图 4.11 educoder 通过截图

4.2 性能分析

表 6 不同方案时钟周期数

方案	时钟周期数
单周期硬布线	1545
气泡流水线	3624
重定向流水线	2297

其中单周期硬布线的时钟周期最短，但是其每一个周期的时间最长，整体来说效率最差。而气泡流水线周期最多，主要原因是对于冲突等情况只能使用插入气泡的方式来解决，而正常的程序中必然存在大量的相关，也就导致其周期数大幅增加，但总的来说相较于单周期硬布线效率还是更高，5 段流水在理想情况下可以提供 5 倍的的运行速度。而重定向流水线采用了重定向的方式，解决了一部分气泡插入，

所以周期数有所减少。

4.3 主要故障与调试

4.3.1 气泡流水线故障

气泡流水线： 针对 Branch 或者 Jump 时的清空流水线有问题

故障现象：执行 branch 或是 jump 指令时发成错误

原因分析：如图 4.1，气泡流水线中，流水部件的有两个操作，一个是插入气泡，一个是清空，前者在发生冲突的时候进行，而后者在检测到分支相关信号时进行。。但实现的时候忘记了对分支相关指令发生时的清空操作，导致出错。

解决方案：在在检测到 branch 或者 jump 指令，即分支相关指令的时候，生成一个名为 B_CLR 的信号，并在相关的流水部件中添加对该信号的处理，如图 4.12 所示

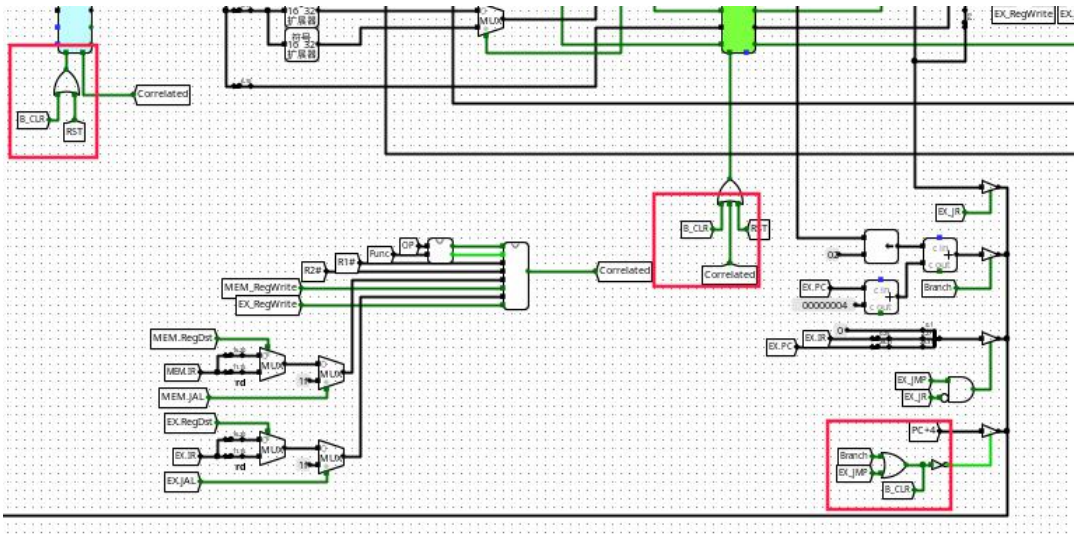

图 4. 12 添加对分支相关指令所导致的清空操作处理

4.3.2 个人拓展指令故障

LBU 指令需要 MEM 中的数据取出其中的低 8 位作为写入 regfile 的数据

故障现象：LBU 指令相关测试无法通过。

原因分析：根据 LBU 的指令含义(Load Byte Unsigned)，应该需要对从 MEM 中获取的指令截取其低 8 位并进行 32 位位拓展，再写入 regfile

解决方案：修在相关电路的 MEM 数据流出端，加入数据选择器，针对 LBU

的情况，将数据改为低 8 位的 32 位逻辑拓展，如图 4.13 所示。

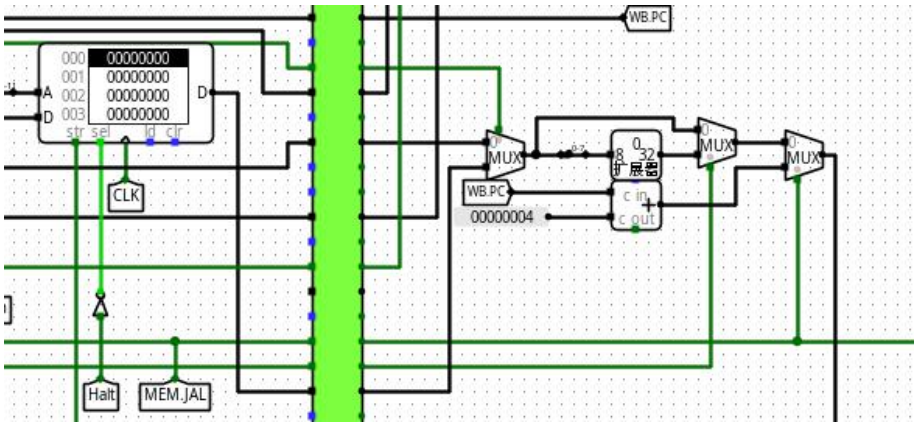

图 4.13 LBU 检测相关逻辑

4.4 实验进度

表 4.2 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。
第三天	完成 Logism 单周期 CPU 的报告,并且通过了 Logism 单周期 CPU 的检查。
第四天	完成理想流水线
第五天	在理想流水线中加入 ccmb 指令并通过测试
第六天	完成气泡流水线，加入 ccmb 指令并通过测试
第七天	完成重定向流水线，加入 ccmb 指令并通过测试
第八天	学习中断相关知识，完成单级中断以及多级中断
第九天	规划团队任务，并开始着手构建
第十天	调试团队任务代码，尝试解决问题
第十一天	团队代码基本完成，电路调通

5 团队任务

5.1 选题与设计

5.1.1 要求

团队作品展示要求在带中断机制的单周期 CPU 电路上完成,需要具备输入输出设备,拥有完整的软硬件系统,通过中断机制能够实现和用户的交互。

5.1.2 选题

通过团队内的讨论，我们一开始选定了两个选题，一个是贪吃蛇小游戏，用户可以通过点击按钮或者操作手柄来移动屏幕中的小蛇，另外一个是数字识别，通过结合机器学习相关知识以及来对用户的点阵输入进行识别。最终由于时间以及难度等因素，我们选择了数字识别作为了我们的选题。

5.1.3 设计

(1) 模型选择与训练

首先我们需要选择机器学习的模型，常见的手写识别可以使用简单的线性分类器或者复杂的神经网络来实现。而我们受限于 CPU 的性能以及支持的指令（如不支持乘法指令），选择了较为简单的线性分类器 SVM 作为我们的模型。具体的训练数据我们采用了比较经典的 mnist 手写数据集，由于原本的数据集是灰度图像，一个像素点的取值从 0 到 255，与 logisim 中能够提供的输入不太相符，并且一个图片有 28*28 个像素点，我们将训练图片转化为只有 0/1 两种取值的黑白图像，并进行训练。可以得到一个 10 * 784 的权重矩阵，分别对应 10 个数字的权重函数，在 7 万张图片上的正确率可以达到 85%，还是比较好的

(2) 数据存储

我们在训练完成后发现得到的权重值均为小于 0 的小数，而我们的 CPU 并不支持小数的计算，所以我们采用了将每一个权重值乘上 1000 之后再截断的方法，既保证了可以使用 CPU 来计算权重函数，又保持了 85%的正确率。同时我们也在 7 万张图片计算的过程中记录了期间出现的最大值以及最小值，保证不论是结果还是中

间过程都不会发生溢出。

得到权重之后就是如何存储的问题，我们一开始是设想将所有的权重利用汇编代码的方式在开始写入数据存储器，但考虑到 10 * 784 的数据确实比较多，可能会比较耗时，所以我们先通过代码自动生成了可以导入 logisim 数据存储器的文本文件，在预测之前手动导入数据存储器即可。其中预留了 784 个位置用于导入用户输入的点阵信息

(3) 输入输出

本次任务中的输入主要就是用户的点阵输入，我们一开始采用的是 4 * 28 个 32 位的 pin 来作为输入，但是不仅视觉效果不好，而且对于后续的计算也非常不方便，因为每一个点均为一个位，在操作的时候需要不断的移位来取出对应位置的值。所以我们最终采用了 784 个单独的 pin，并通过去除了外壳的封装元件(增大点击范围)将每一行的输入整合起来，便于作为 LED 点阵的输入。而输出我们则采用了 LED 点阵来反应用户输入的数字图像，同时使用数码管来展示最终的预测结果。

5.1.4 分工

整个程序的分工分为代码实现以及电路部分。我主要负责所有的代码以及大部分电路的连接。具体包括前期机器学习代码的编写。权重文件的生成，以及汇编代码的编写，和用户的数据导入逻辑。

5.2 具体实现

5.2.1 汇编代码

由于前期的机器学习代码其实和组成原理不是特别相关，所以主要还是以之后的汇编代码为主来介绍整个项目。

汇编代码中用到的主要的寄存器的所存储的值的含义如所示

寄存器名	含义
$s0	截止运行时，概率最大的数字，初始化为 0
$s1	存放 10 组权重函数中目前最大的函数值，由于函数值有可能为负数，故初始化为-20000
$s2	存放当前正在计算的权重函数的累加值
$s3	存放当前的像素点对应具体权重所在的偏移量范围为[784 784 + 784 * 10)
$s4	当前正在计算的权重函数所对应的数字
$s6	存放用户的 784 个输入的 0/1 值

整体的逻辑如下:

分别计算 10 个数字对应的权重函数的函数值，在计算的过程中保存最大的函数值，以及其对应的数字，如果在后续的计算中遇到了函数值更大的函数，就更新最大函数值以及对应数字。在计算过程中通过 syscall 系统调用，在数码管上显示当前计算的数字，即进度显示。在 10 个权重函数计算完后，输出最大函数值所对应的数字作为针对用户输入的点阵信息的预测。并将其显示在数码管上。

5.2.2 用户点阵信息载入

在用户输入完点阵信息后，开启时钟将会自动进行用户数据的导入。导入完成后将会以发光二极管提示用户点阵数据导入完成，用户点击开始计算之后将会执行前文描述的预测程序。值得注意的是，虽然用户输入的信息为 28 * 28 的点阵信息，使用一个二进制位来表示一个数据应该会更合适，但是为了便于后续代码的编写以及调试，还是选择了用一个字来存储一个像素点的方法。具体的导入逻辑电路如图 5.1 所示

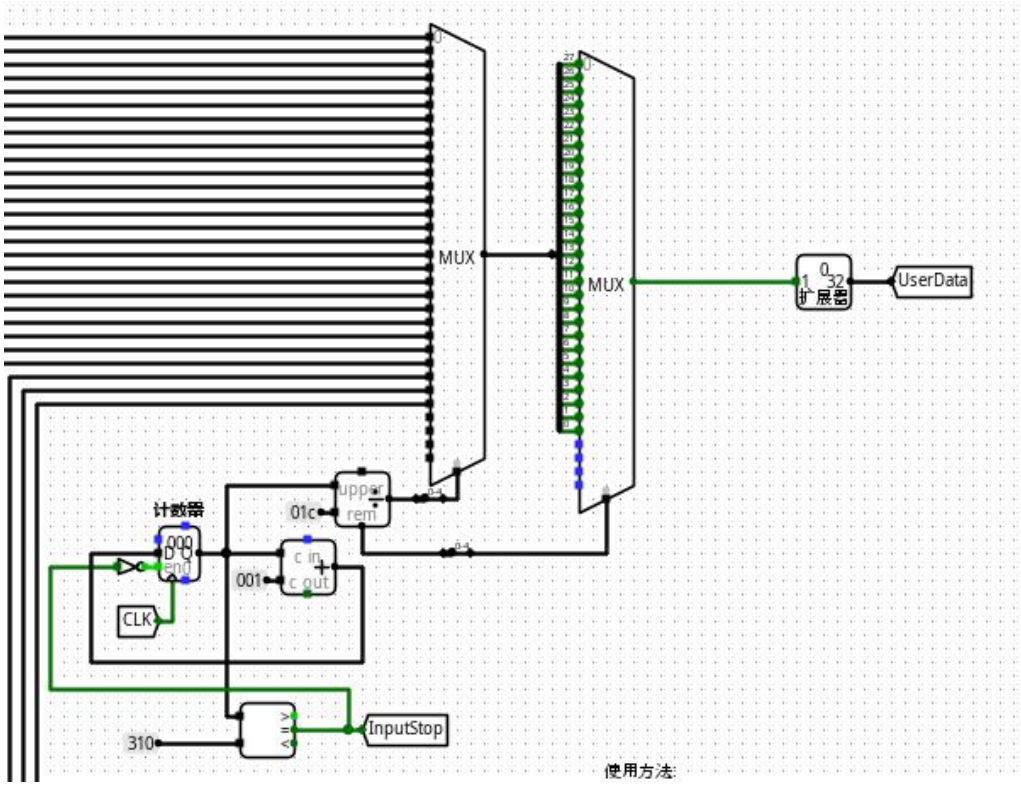

图 5.1 用户数据导入逻辑

5.3 测试与故障

5.3.1 测试和展示

测试步骤

(1) 点击数据 RAM，加载本地 raw_data.txt 文件，导入权重信息。

(2) 点击点阵图，绘制出需要进行判断的数字，此处以 2 为例

(3) 开启时钟频率，等待点阵信息导入完成

(4) 点击”开始计算”，等待 CPU 计算完成（期间可以进入 regfile 以加速）

(5) 等待计算完成后查看主界面的预测结果

最终结果如图 5.2 所示

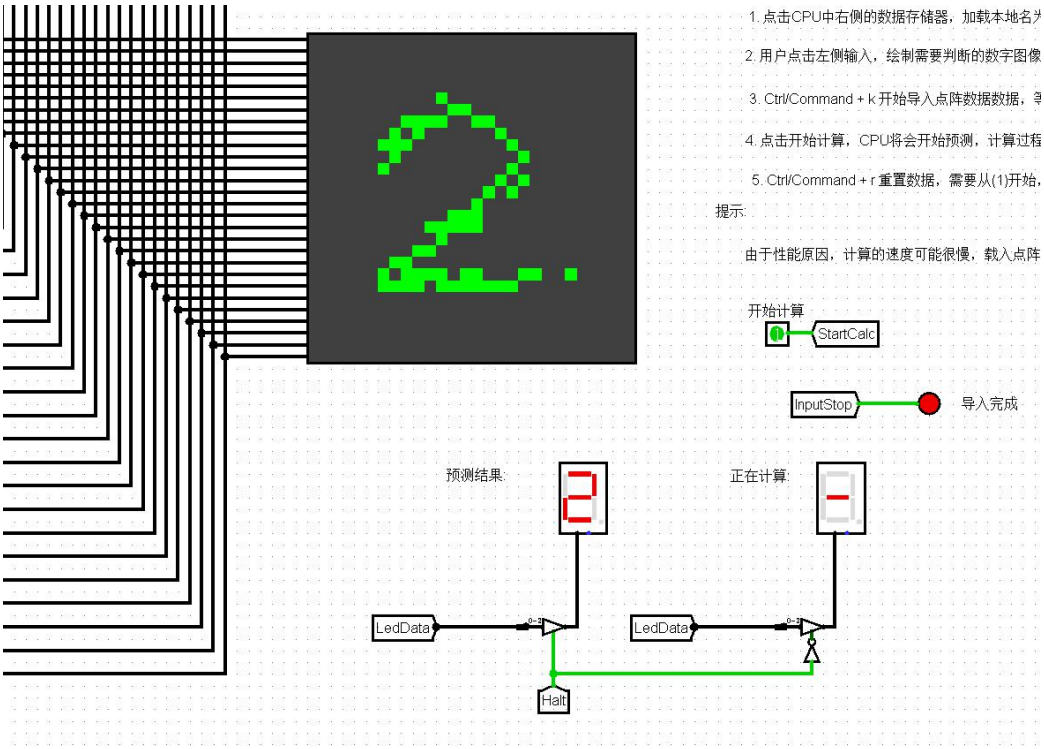

图 5.2 数字二预测结果

5.3.2 权重偏移量出错

由于使用的一个字来存储用户点阵信息以及权重信息，在汇编代码使用 lw 载入对应的权重值以及点阵值的遍历过程，每一个循环应该+4 而不是+1，需要与一个字 4 个字节对应，因为 CPU 以字节编址。

5.3.3 计算过程时间过慢

一方面是因为在 logisim 中运行复杂的电路时频率确实会下降很多，尤其是我们有 784 个用户输入的情况，为此我们采用了两方面的改进措施，一方面是改进代码，由于计算过程需要 lw 两次，一次是载入用户的某一个位置的点阵信息，其值为 0 或者 1，另一个是对应位置的权重信息。可以通过先对点阵信息进行判断，如果是 0 就直接跳过权重信息的导入以及后续的计算过程，也算是减少了一部分运行时间。另一个方面则是在运行时手动从运行的电路图，进入 regfile。可能是因为 logisim 渲染的原因。进入到子组件后可以将时钟频率提高 4 到 10 倍，极大的提高了运行速度。同时又能够对照前文所述寄存器使用表来查看当前运行的动态信息。

6 设计总结与心得

6.1 课设总结

基本次课设耗时将近两周，我从最简单的单周期 CPU 开始，到理想流水线，再到更加复杂的重定向流水线，最终完成了团队任务。回顾整个课程设计，共完成以下几点工作:

1） 使用 Logisim 完成了单周期 CPU，理想流水线，气泡流水线，重定向流水线，单级中断以及多级中断。并通过了相关测试文件的测试以及 educoder 的在线测试

2） 在所有的 CPU 上均完成了 24+4 条指令功，气泡流水线可以处理较为复杂的程序，而重定向流水线则进一步提高了效率能总结。

3） 与队友们完成了团队任务的选题、设计并完成了具体实现。自己编写了相关的汇编代码并在单周期硬布线 CPU 上运行成功，理论正确率可以达到 85%

6.2 课设心得

本次课程在大三下学习进行，有了上学期的组原实验中单周期 CPU 的基础，完成起来还是比较顺利的，针对后续的流水线等电路也有相关的文档。尤其是 educoder 在线评测可以非常准确的给出是否正确的判断，并对错误的地方给出标注。

这次课程的难度其实还是比较大的。课程的周期比较长，有上学期的 CPU 的基础，单周期硬布线的电路连接还是比较顺利的。但是后续的三条流水线的难度就开始逐级上升了。一方面是流水部件的设计，另一方面是需要将自己的 4 条拓展指令融入进去，还是需要对整个 CPU 的工作流程有一定的理解的。

其次便是中断，单级中断的难度还算比较简单。但是多级中断就需要考虑很多问题，比如对中断发生时发起另一个中断时的处理以及恢复时的先后顺序问题，等等。

团队任务个人其实由于另外三个队友中两个队友都在实习，再加上最后时间也有点赶，所以选择了一个较为简单的任务。但是整个项目从选题到设计完成以及测

试都走过了比较完整的流程。并且最终实现的效果还是比较好的，融合了本学期学习的机器学习的相关知识，并仅仅使用 CPU 所支持的 24 条基本指令就完成了相关的计算。由于运算时间的问题并没有做非常大量的测试，但是从已经测试的结果来看，数字识别的正确率还是比较高的。并且理论上正确率应该可以达到 85%左右。个人还是比较满意的。

关于对本次课程的建议。其实我由于做的比较晚，使用的工具不管是 Excel 表格还是测试代码都在其他同学的反馈之下比较完善了。所以做的过程还是比较顺利的。主要问题还是由于疫情所导致的流程过长，以至于后续的报告编写都比较困难，尤其是实验过程与调试一节，由于之前并没有特地记录遇到的问题，所以现在回想起来都不太记得当时遇到了什么问题以及是怎么解决的。但这些应该只是疫情导致的特殊情况。说来惭愧，虽然给了很多时间，我还是没有完成后续的动态分支预测以及流水中断，但是之后的团队任务还算是完成了，个人感觉效果还不错，虽然界面什么的还是简陋了一些。

最后也感觉各位老师在疫情当中依然坚持为我们答疑解惑，并且提供了如此优质的课程内容。

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: