

华中科技大学

2021

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：CS1806

学号：U201813676

姓名：刘汉鹏

电话：1837267700

邮件：1359947339@qq.com

华中科技大学课程设计报告

目 录

1	课程设计概述.....	3
1.1	课设目的.....	3
1.2	设计任务.....	3
1.3	设计要求.....	3
1.4	技术指标.....	4
2	总体方案设计.....	6
2.1	单周期 CPU 设计.....	6
2.2	中断机制设计.....	11
2.3	流水 CPU 设计.....	13
2.4	气泡式流水线设计.....	14
2.5	数据转发流水线设计.....	15
2.6	动态分支预测机制.....	16
3	详细设计与实现.....	18
3.1	单周期 CPU 实现.....	18
3.2	中断机制实现.....	22
3.3	流水 CPU 实现.....	26
3.4	气泡式流水线实现.....	27
3.5	重定向流水线实现.....	29
3.6	动态分支预测机制实现.....	31
4	实验过程与调试.....	35
4.1	性能分析.....	35
4.2	主要故障与调试.....	35
4.3	实验进度.....	37

华中科技大学课程设计报告

5	团队任务	38
5.1	选题与设计.....	38
5.2	负责部分.....	38
6	设计总结与心得	41
6.1	课设总结.....	41
6.2	课设心得.....	41
	参考文献.....	43

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及实现能力的培养”。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；

华中科技大学课程设计报告

- (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	SU _b	减	
11	OR	或	
12	ORI	立即数或	

华中科技大学课程设计报告

#	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	
24	SYSCALL	系统调用	If \$v0==10 halt(停机指令) 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	SUBU	无符号减	
30	LHU	加载半字(无符号)	
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方案，且实现指令和数据不共用一个存储器的方式完成方案的设计。在一个周期内，控制器根据读入的指令进行相应的解析指令操作，给出对应的控制信号，同时其余各功能部件根据得到的控制信号完成相应功能的实现，如对存储器的存储、寄存器的访问、ALU 的控制。在实施的过程中，全部采用 logisim 电路实现。

总体结构图如图 2.1 所示。

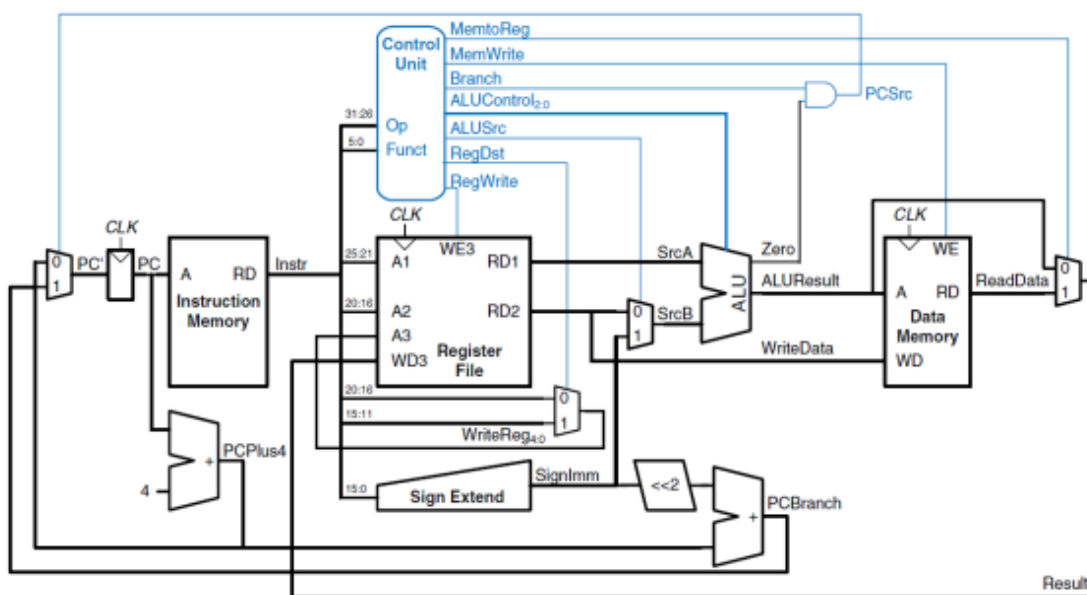


图 2.1 总体结构图

2.1.1 主要功能部件

主要用到的功能部件用：程序计数器 PC、指令存储器 IM、运算器 ALU、寄存器堆 RF、数据存储器 DM 以及符号扩展模块及数据选择器等。具体设计思路如下：

1. □ □ □ □ □ PC

用于存储下一条指令在指令存储器中的地址。输入端口有：下一条指令在 IM

华中科技大学课程设计报告

中地址、控制 PC 是否跳转到输入的使能信号、时钟信号和复位信号；输出为取值阶段的指令地址，在 logsim 中使用时钟上升沿触发的寄存器即可。

2. □ □ □ □ □ IM

用于存储二进制形式的 MIPS 指令。输入端口有：指令的存储地址；输出端口有：二进制 MIPS 指令。在 logsim 的具体实现中，使用 ROM 器件即可，指令的加载直接手动实现即可。在取指令时根据 PC 寄存器输出端口的值来进行指令的读取，一条指令占 4 个字节，所以需要将读取的 PC 的值去掉末两位再取第 2-11 位作为地址进行读取。

3. □ □ □

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

表 2.2 ALUOP 和功能描述

ALU_OP	十进制	功能
0000	0	Result = X << Y 逻辑左移 (Y 取低五位) Result2=0
0001	1	Result = X >>> Y 算术右移 (Y 取低五位) Result2=0
0010	2	Result = X >> Y 逻辑右移 (Y 取低五位) Result2=0
0011	3	Result = (X * Y) _[31:0] ; Result2 = (X * Y) _[63:32] 无符号乘法
0100	4	Result = X/Y; Result2 = X%Y 无符号除法

华中科技大学课程设计报告

ALU_OP	十进制	功能
0101	5	Result = X + Y (Set OF/UOF)
0110	6	Result = X - Y (Set OF/UOF)
0111	7	Result = X & Y 按位与
1000	8	Result = X Y 按位或
1001	9	Result = X ⊕ Y 按位异或
1010	10	Result = ~(X Y) 按位或非
1011	11	Result = (X < Y) ? 1 : 0 符号比较
1100	12	Result = (X < Y) ? 1 : 0 无符号比较

4. □ □ □ □ RF

寄存器堆采用MIPS 自带的寄存器实现，包含32 个通用寄存器，该寄存器堆的输入输出引脚以及相应功能的描述如表2.3 所示。

表 2.3 寄存器堆的引脚与功能描述

引脚	功能描述
R1#	读寄存器 1 的编号
R2#	读寄存器 2 的编号
W#	写寄存器的编号
Din	要写入的数据
WE	写使能
CLK	时钟信号
R1	[R1]寄存器中的数据
R2	[R2]寄存器中的数据所有操作有效

5. 控制器

采用硬布线控制，通过解析出的指令给出不同的控制信号，根据输入的指令通过组合逻辑生成相应的控制信号即可。

华中科技大学课程设计报告

2.1.2 数据通路的设计

表 2.4 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		
ADDI	PC+4	PC	rs		rt	alu	r1	SignImm	5		
ADDIU	PC+4	PC	rs		rt	alu	r1	UnsignImm	5		
ADDU	PC+4	PC	rs	rt	rd	alu	r1	r2	5		
AND	PC+4	PC	rs	rt	rd	alu	r1	r2	7		
ANDI	PC+4	PC	rs		rt	alu	r1	SignImm	7		
SLL	PC+4	PC		rt	rd	alu	r2	SignImm	0		
SRA	PC+4	PC		rt	rd	alu	r2	SignImm	1		
SRL	PC+4	PC		rt	rd	alu	r2	SignImm	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	SignImm	8		
NOR	PC+4	PC	rs	rt	rd	alu	r1	r2	10		
LW	PC+4	PC	rs		rd	ReadData	r1	SignImm	5	alu	
sw	PC+4	PC	rs	rt			r1	SignImm	5	alu	r2
BNE/BEQ	PC+4+SignImm{32:0}	PC	rs	rt			r1	r2			
J	{PC+4{31:28}, IR{25:0}, 2b' 0}	PC	rs	rt			r1	r2			
JAL	{PC+4{31:28}, IR{25:0}, 2b' 0}	PC	rs		R[31]	ReadData	r1	r2			
JR	R[rs]	PC	rs		rt	ReadData	r1	SignImm		alu	r2
Syscall	PC+4	2#	4#				r1	r2			

华中科技大学课程设计报告

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 2.5。

表 2.5 主控制器控制信号的作用说明

控制信号	取值	说明
MemToReg	0/1	写内存控制信号，为 1 时代表向内存 RAM 中写入值
RegWrite	0/1	寄存器写使能信号，为 1 时代表向寄存器中写入值
AluOP	0000~1100	见表 2-2
MemToReg	0	寄存器写入数据来自运算器 ALU 的结果
	1	寄存器写入数据来自存储器
RefDst	0	写入寄存器编号为 rt
	1	写入寄存器编号为 rd
AluSrcB	0	运算器 B 来自寄存器输出 R2
	1	运算器 B 来自符号拓展
SignExt	0/1	为 0 代表对立即数无符号拓展，为 1 代表对立即数有符号拓展
JR	0/1	寄存器跳转指令译码信号，为 1 代表当前为 JR 指令
JAL	0/1	为 1 时代表当前指令为 JAL 指令
JMP	0/1	无条件分支控制信号
Beq	0/1	Beq 指令译码信号
Bne	0/1	Bne 指令译码信号
Syscall	0/1	Syscall 指令译码信号
SLLV	0/1	SLLV 指令译码信号
SUBU	0/1	SUBU 指令译码信号
LHU	0/1	LHU 指令译码信号
BLEZ	0/1	BLEZ 指令译码信号

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

华中科技大学课程设计报告

的设计。该控制信号表的框架如图 2.2 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
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		1				
17	BNE	5	X	X						1			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	ERET	16	24	X												
26	SLLV	0	4	0				1			1					
27	SUBU	0	35	6				1			1					
28	LHU	37	X	5	1		1	1		1						
29	BLEZ	6	X	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

在本实验中的中断是异步中断，中断请求与指令无关，CPU 需要在指令执行结束后才响应外部中断。为了实现 MIPS 中断处理，需要从硬件和软件两个方面进行设计。在硬件方面，我们需要实现中断请求生成逻辑电路、中断仲裁与中断识别电路、异常程序计数器 EPC 以及中断使能寄存器 IE；在软件方面我们要实现相关的指令支持，包括中断返回指令、开关中断指令等，还要实现程序支持，改造主程序和中断服务程序。单中断和多中断的执行过程如图 2.3 所示。

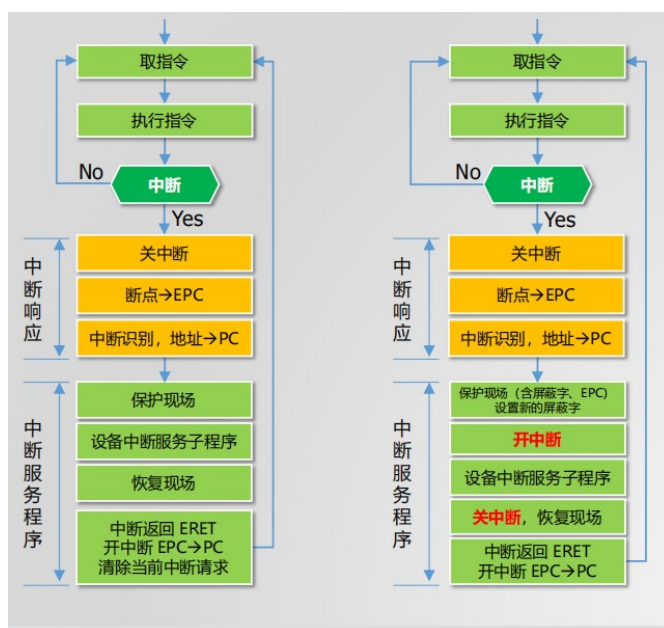


图 2.3 中断执行流程图（左单中断，右多中断）

2.2.2 硬件设计

在外部中断的实现过程中，我们首先要实现按键中断外围电路，能够通过点击编号按钮产生对应的中断请求，锁存中断请求并在完成对应的中断请求服务后清除中断请求。

我们使用一个优先编码器实现中断优先级仲裁，在有多个中断请求信号时，输出优先级最高的中断请求信号。然后获取三个中断服务程序的入口地址通过一个选择器输出相应的中断入口地址实现中断入口识别。

异常程序计数器 EPC 的实现在当有中断请求产生时才将当前的 PC 地址压入寄存器中。

单级中断和多级中断分别利用终端因指令、ERET 信号和 MFC0、MTC0 实现中断使能寄存器 IE 来控制开关中断。

多级中断需要在单级中断的硬件设计上进行一定的修改，主要是中断使能寄存器 IE 和异常程序计数器 EPC。由于涉及到中断嵌套，也就是可以在一个中断服务程序正在执行时响应另外一个中断请求信号，我们需要利用三个寄存器构成一个 EPC 硬件堆栈来实现保存多个程序地址断点以及对应的中断号。同时在实现多级嵌套前还要考虑优先级，所以中断请求信号此时不应该仅仅只考虑当前按键产生的中断号，还要与当前正在执行的中断程序的优先级，优先级更高才可以嵌套，但在单级中断中不涉

及嵌套就没有这方面的考究。

2.2.3 软件设计

多级中断和单级中断程序中都存在与中断服务程序有关的指令来实现开关中断和中断返回。因此我们要对这些指令进行译码产生相应的指令控制信号。由图 2.3 可知在单级中断中，响应时根据中断隐指令关中断，在执行 `eret` 后执行开中断，所以我们只需要设计指令 `ERET`，该指令由控制信号产生器根据指令译码产生。在多中断中，除了中断返回指令 `ERET` 外还涉及到单独的开关中断——`MFC0` 和 `MTC0`，这两个信号也通过指令译码产生，两个控制信号指令译码的低六位均为 `0x10`，区别在于 `MFC0` 的第 25-21 位为 `0x0`，`MTC0` 为 `0x4`。两个控制信号只需要与门和比较器就可以得到。

2.3 流水 CPU 设计

2.3.1 总体设计

我们将 MIPS 指令的执行过程细分为 5 个阶段：

- ◆ 取指令(IF)：从指令存储器中取出指令
- ◆ 指令译码(ID)：操作控制器对指令字译码，同时从寄存器堆取操作数
- ◆ 指令执行(EX)：执行运算操作或者地址计算
- ◆ 访存(MEM)：对存储器进行读写操作
- ◆ 写回(WB)：将指令执行结果写回寄存器堆

我们在每一个阶段后面都设计一个流水寄存器用于锁存本阶段处理完成的所有数据或结果，以保证本段的执行结果能够在下一个时钟周期给下一个阶段使用。

2.3.2 流水接口部件设计

流水接口部件实质上就是一个锁存器，用于锁存前段加工处理完毕的数据和控制信号，这些数据通常会横穿流水寄存器传递到下一段。五段流水线需要设计四个流水寄存器分别命名为：`IF/ID`, `ID/EX`, `EX/MEM`, `MEM/WB`。所有的流水接口部件都使用统一的始终 `CLK` 来同步，并通过设计同步清零实现清除流水寄存器中的数据，设计使能端实现流水寄存器的暂停，设计若干寄存器来锁存相应的数据。

不同流水寄存器锁存和传递的数据信息不尽相同。其具体用到的数据入表 2.6 所

示。

表 2.6 各流水部件需要传递的数据

流水部件	锁存和传递的数据
IF/ID	PC, PC+4, IR
ID/EX	PC, PC+4, IR, RegWrite, MemToReg, MemWrite, AluOp, Alusrc, SignExt, JMP, JAL, JR, Beq, Bne, Syscall, RS,RT,写寄存器编号 WriteReg#, 符号拓展, 无符号拓展
EX/MEM	PC, PC+4, IR, RegWrite, MemToReg, MemWrite, JAL, Syscall, 写寄存器编号 WriteReg#, 数据存储器待写入数据 Writedata, v0, AluResult
MEM/WB	PC, PC+4, IR, RegWrite, MemToReg, JAL, Syscall, 写寄存器编号 WriteReg#, 数据存储器待写入数据 Writedata, v0, AluResult, 数据存储器读出数据 RD

2.3.3 理想流水线设计

由于进入理想流水线的对象不受其他阶段的影响, 仅仅运行无数据相关, 没有分支指令, 我们可以将流水寄存器中和控制信号产生部件中有关跳转指令的端口悬空, 并在单周期 MIPS 的基础上删除有关分支跳转和无条件转移的电路, 然后将对应的信号和数据连接对应的流水寄存器接口即可。

2.4 气泡式流水线设计

实际流水线存在很多指令相关和数据依赖, 从而引起流水线的停顿。指令相关会导致结构冲突和控制冲突, 数据相关则会带来数据冲突。

结构冲突往往是因为多条指令在同一周期使用同一个操作部件, 这可以通过增设这个冲突部件来解决。

控制冲突主要体现在分支指令, 当流水线遇到分支指令或其他会改变 PC 值的指令时, 在分支指令之后载入流水线的相邻指令可能因为分支跳转不能进入执行阶段。为了解决控制冲突, 需要在执行程序分支跳转的时候清除已经进入流水寄存器中的分支指令后续的若干条误取指令。当选定在 EX 阶段执行分支指令, 此时有两条指令分别进入了 IF/ID, ID/EX 流水寄存器, 需要将分支成功跳转的信号 BranchTaken 输送到这两个流水寄存器的清零端进行清空操作。同时在此阶段将计算出来的跳转地址送入到 PC 的选择器中。

华中科技大学课程设计报告

数据冲突主要体现在：后续指令 I2 要用到前面指令 I1 的结果，如果指令 I2 在指令 I1 将结果写寄存器之前就在 ID 段读取了该寄存器的旧值，则读取数据就是错误的。为了解决这样的数据冲突，我们在存在冲突的指令之间插入空指令直到冲突消失。为此，我们首先需要设计在理想流水线的基础上增加一个数据冲突检测子电路，该子电路在 ID 段确认指令使用的源寄存器是否在前两条指令中被写入。由于 MIPS 指令只包括 0~2 个源操作数，分别是 rs,rt 字段对应的寄存器，我们只要检查 EX、MEM 段的寄存器堆写入信号 RegWrite 是否为 1 且写寄存器编号 WriteReg# 和源寄存器编号是否相同即可。当数据冲突检测子电路检测到冲突产生冲突信号 Stall 时，将该信号取反送入到 IF/ID、PC 寄存器的使能端就可以实现 IF、ID 段指令的停顿。

而为了解决 ID 段和 WB 阶段的数据冲突，可以让寄存器堆写入控制采用下降沿触发，而其他流水寄存器使用上升沿触发，这样在一个时钟节拍的中间时刻可以完成寄存器数据的写入而后半段可以正确读取。气泡流水线的数据通路如图 2-5 所示。

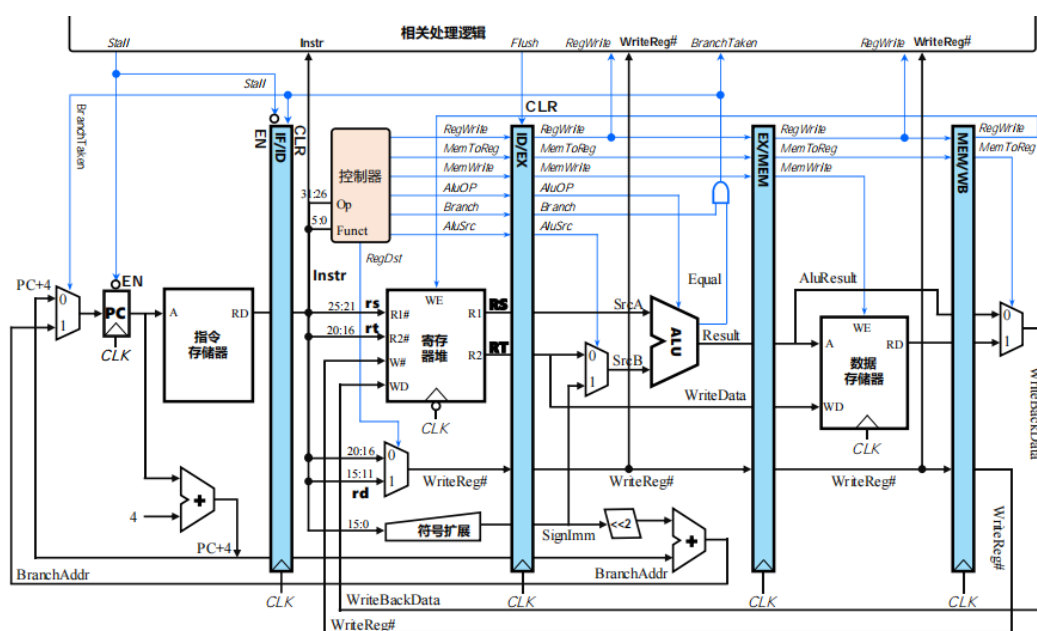


图 2.4 气泡流水线数据通路

2.5 数据转发流水线设计

实际上的程序存在大量的数据冲突和控制冲突，采用气泡流水线将带来大量气泡和停顿严重影响流水线的性能。我们知道当 ID 段检测到数据冲突时，此时 EX 阶段取到的操作数一定是错误的，但此时的正确的操作数其实已经在 MEM 或 WB 阶段产生。所以我们可以直接将正确的操作数从其所在的位置重定向到 EX 段的合适位置而

不需要进行数据检测从而避免产生气泡导致流水线的停顿。

EX 阶段源操作数有两个 RS,RT。为了选择正确的操作数，我们需要设计相关处理电路生成两个重定向选择信号 RsForward 和 RtForward。如果产生冲突的两条指令的前一条为 Load 指令且位于 EX 段时，此时无法通过重定向解决，因为理论上虽然可以但此时关键延迟将变长而使流水线的性能变差。因此我们还是要检测由 Load 带来的数据相关。重定向依然还需要解决由分支指令带来的控制冲突，基本原理和 2.4 节中气泡流水线解决方案一致。

最终重定向电路将在遇到 Load 指令冲突时使 PC 寄存器产生停顿，在遇到分支跳转时对 IF/ID 和 ID/EX 流水寄存器进行清空。

2.6 动态分支预测机制

重定向流水虽然很大程度上解决了数据冲突带来的停顿，但还是存在由分支指令带来的控制冲突，因此我们可以采用动态分支预测的方法，设计一个分支预测缓冲器 BTB 表，根据分支指令的分支跳转历史不断对预测策略进行动态调整，当预测成功时不需要停顿，以此提升性能。

为了实现对分支的预测，我们设计一个双预测位的状态转换机。该状态机主要操作为两步：① 分支预测；② 状态修改。能够根据分支跳转实际执行情况和当前状态修改下一个状态，其状态转换图如图 2.5 所示。从图中可以看出，只有连续两次预测错误才会改变对分支去向的预测。

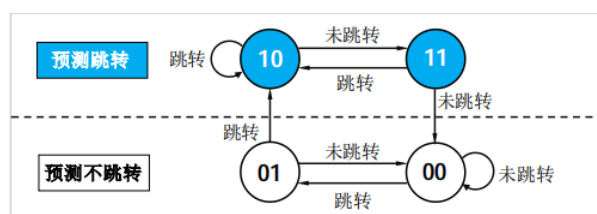


图 2.5 双预测位状态转换图

由状态图不难看出，当双预测位的高位为 1 时，BTB 表预测“分支成功”，为 0 时预测“分支失败”。

BTB 表本质上是一个全相联的 cache 结构。其每个 cache 槽主要包括 valid 位、执行过的成功分支指令地址（标识），预测的分支目标地址，历史跳转信息描述位（预测状态位）、淘汰计数五项。采用 BTB 的处理过程如图 2.6 所示。其数据通路如图 2.7 所示。

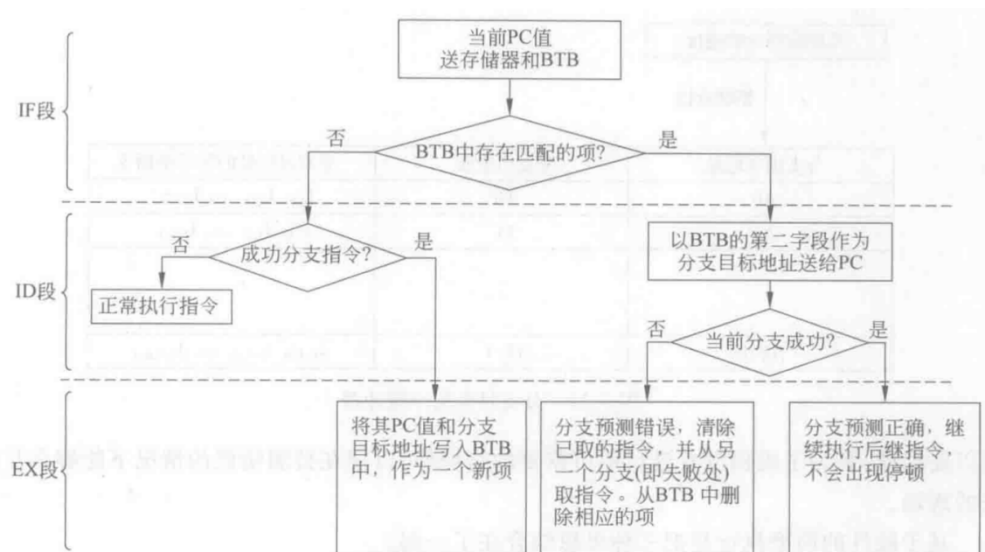


图 2.6 采用 BTB 时的处理步骤

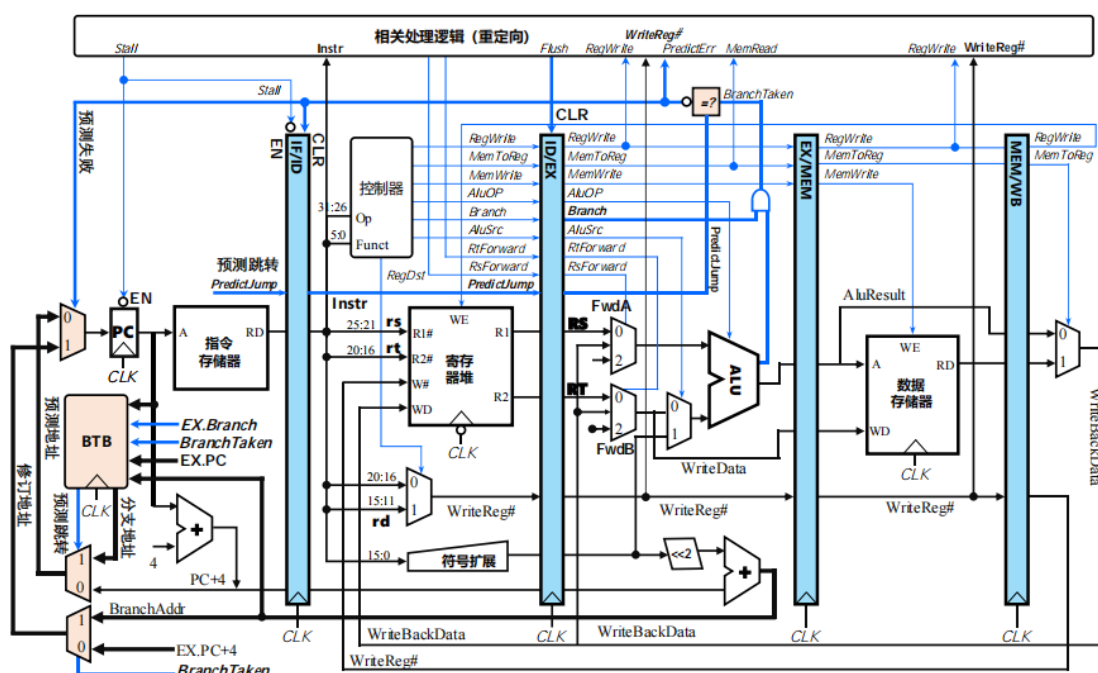


图 2.7 使用动态分支预测的五段流水线数据通路

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，将此控制信号通过非门取反之后作用于程序计数器 PC 的使能信号，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，屏蔽时钟信号，使整个电路停机。如图 3.1 所示。

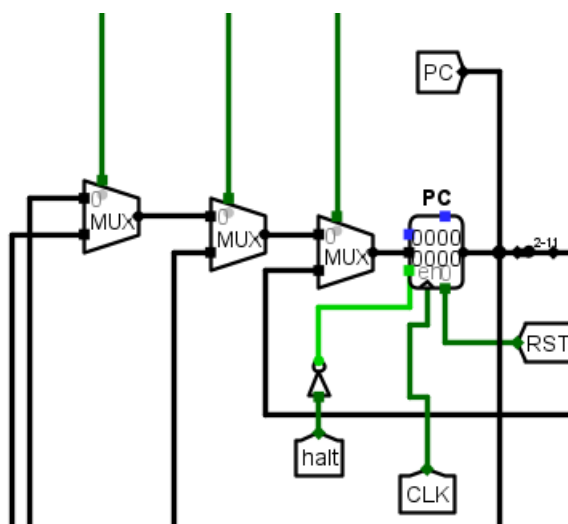


图 3.1 程序计数器 (PC)

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。

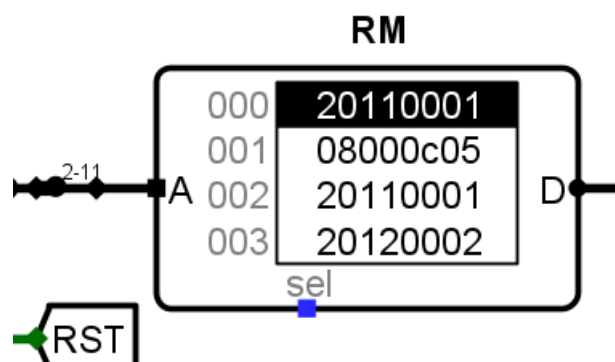


图 3.2 指令存储器 (IM)

3) 运算器 (ALU)

运算器采用 Logisim 所带的库 cs3410.jar 中的运算器电路。运算器的 X 输入端来自寄存器堆的 R1 输出，Y 输入端取决于 AluSrcB 和 SignedExt 信号，当二位宽数据 {AluSrcB, SignedExt}=2b'0|2b'1 时，选择寄存器堆的 R2 输出端输入，为 2b'2 时选择有符号拓展立即数，为 2b'3 时选择无符号拓展的立即数。如图 3.3 所示。

4) 寄存器堆 (RF)

寄存器堆器件也采用 cs3410.jar 中自带的子电路。输入端口 R1#和 R2#取决于 syscall 指令，如果此时为 syscall 指令则 R1#选择 2 号寄存器，否则选择 rs 字段对应的寄存器；R2#选择 4 号寄存器，否则选择 rd 字段对应的寄存器。输入端口 W#取决于 RegDst 和 JAL 两个控制信号，JAL 为 1 选择 0x1f 号寄存器，为 0 时如果 RegDst 为 1 则选择 rd 字段对应的寄存器，否则选择 rt 字段对应的寄存器。输入端口 Din 取决于 JA 控制信号，当 JAL 为 1 时将 PC+4 的值写入，否则选择数据存储器中的内容。如图 3.3 所示。

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)。和指令存储器一样，输入的地址运算结果只取 2-11 位输入。RAM 的输入端来自于寄存器堆的 R2 输出，并在 MemWrite 信号的控制下进行向输入的地址所在的地方写入数据，并在 MemToReg 信号为 1 时将存储器中的数据写回到寄存器堆中。如果指令为 LHU 指令，读的是高字节还是低字节取决于存储器输入的 32 位地址的第 1 位，第 1 位为 1 代表为高字节，此时选取存储器的 16~31 位按照 0 拓展为 32 位输出，否则选择 0~15 位 0 拓展输出。如图 3.3 所示。

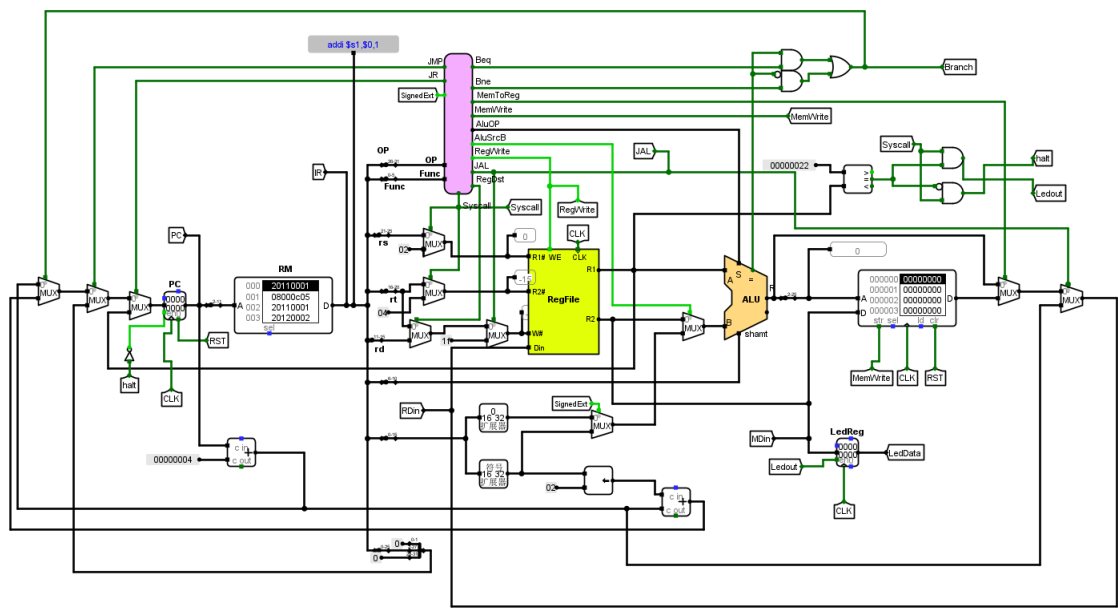


图 3.3 RF、ALU、DM 电路结构图

3.1.2 数据通路的实现

本次课程设计采用的工程化的设计模式，一次性构建所有的数据通路。主要实现方法为，对于每一条指令，将其改写成 RTL（Register Transfer Level），忽略控制类信号，仅保留数据类信号，根据 RTL 功能填写对应指令的数据通路表，描述五大部件之间的连接关系，记录各部件输入端数据来源。

根据总体方案设计中数据通路设计那一小节的详细内容，具体分析每一条指令在执行过程中各个主要部件的输入和输出端口的连接，完成指令系统数据通路表的填写，如表 3.1 所示。

表 3.1 指令系统数据通路表

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
ADD	PC+4	PC	rs	rt	rd	alu	r1	r2	5			
ADDI	PC+4	PC	rs		rt	alu	r1	立即数	5			
ADDIU	PC+4	PC	rs		rt	alu	r1	立即数	5			
ADDU	PC+4	PC	rs	rt	rd	alu	r1	r2	5			

华中科技大学课程设计报告

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
AND	PC+4	PC	rs	rt	rd	alu	r1	r2	7			
ANDI	PC+4	PC	rs		rt	alu	r1	立即数	7			
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			
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	立即数	8			
NOR	PC+4	PC	rs	rt	rd	alu	r1	r2	10			

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。

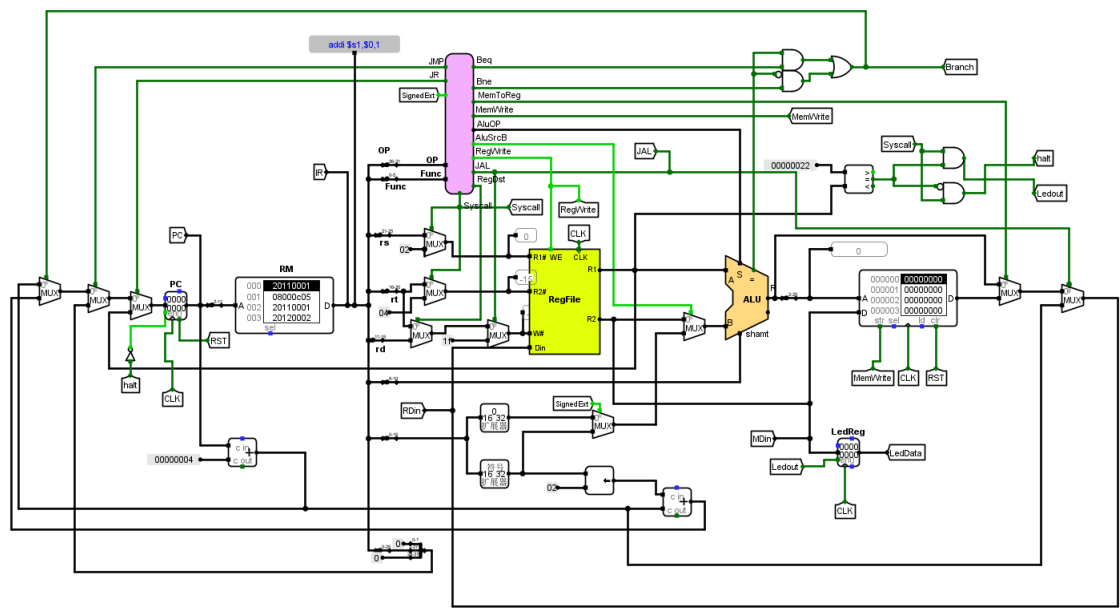


图 3.4 单周期 CPU 数据通路 (Logisim)

3.1.3 控制器的实现

根据图 2.2 所示的控制信号表，利用 Logisim 自动生成组合逻辑的功能实现控制信号部件和运算器控制器部件。电路如图 3.5 所示。

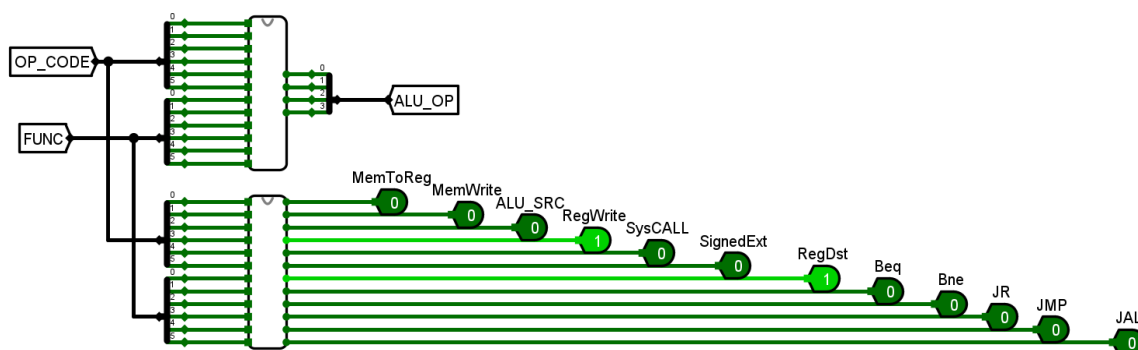


图 3.5 单周期控制器部件

3.2 中断机制实现

3.2.1 单级中断

1. 按键中断外围电路

中断信号采样电路如图 3.6 所示，左侧的 D 触发器在当按下中断按键时被触发，该信号通过右侧的 D 触发器锁存，同时 Clr 信号控制该触发器实现同步清零。

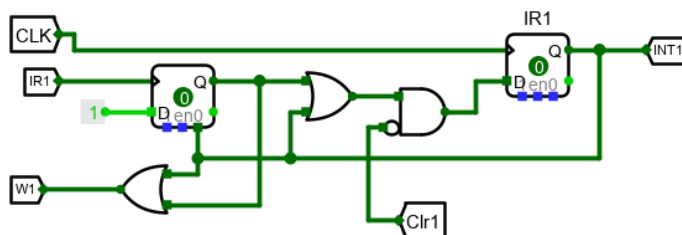
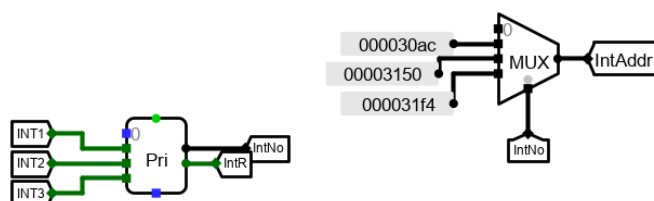


图 3.6 中断信号采样电路

2. 中断优先仲裁电路和中断入口识别电路



华中科技大学课程设计报告

图 3.7 中断优先仲裁和入口识别电路

3. 异常程序 EPC 寄存器和 IE 寄存器

在单级中断中存在 ERET 指令，该指令信号由指令译码给出，中断程序将在该指令执行时执行开中断操作，因此在 IE 寄存器中，中断请求信号 IntR 和 ERET 取非后相与一起送到 D 触发器的输入端，这样在当有按键中断请求产生且 ERET 为 0 时，触发器输出为 1，表示关中断，且一旦进入一个中断程序之后，即使有新的优先级更高的中断请求产生，因为当前中断程序尚未执行完毕，ERET 信号为 0，D 触发器的输出始终为 1，始终为关中断从而不会进入其他中断程序。只有在中断服务程序执行完毕后，中断返回 ERET 为 1，D 触发器输出为 0，才开中断允许响应其他中断请求。IE 寄存器的输出取反后与 IntR 信号相与可以得到真正的被允许的中断请求信号。

EPC 寄存器在当中断请求信号为 1 时触发时钟，为 0 时屏蔽，使得程序 PC 地址正确保存到寄存器中。

在当 ERET 信号为 1 时，执行中断返回，PC 寄存器将获取 EPC 中保存的程序断点作为下一条执行的指令；当中断请求产生时，PC 寄存器将获取当前的中断入口程序地址作为下一条执行的指令地址，否则按照正常程序执行取址。

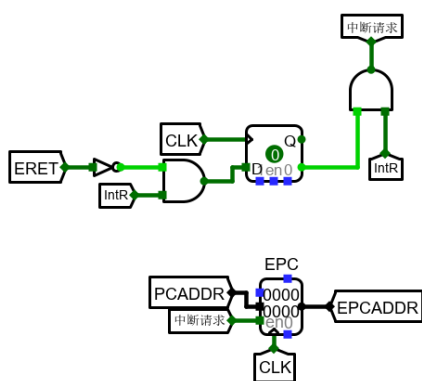


图 3.8 EPC 寄存器和 IE 寄存器

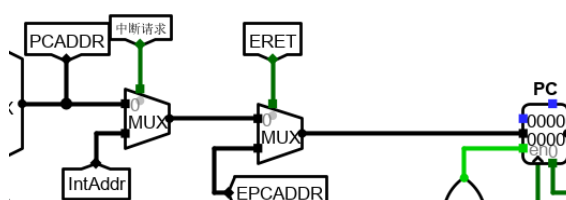


图 3.9 添加中断的数据通路

3.2.2 多级中断

1. 异常程序计数器 EPC 堆栈

依据栈先进后出的思想，连续使用三个寄存器实现一个堆栈。当有多个中断请求信号先后到来时，最先执行的低优先级中断程序的地址将被触发存入后面的寄存器中，而在当一个中断服务程序执行完后，`ERET` 为 1，寄存器被触发选择后一个寄存器的内容作为输入从而实现恢复断点。中断服务号同理。

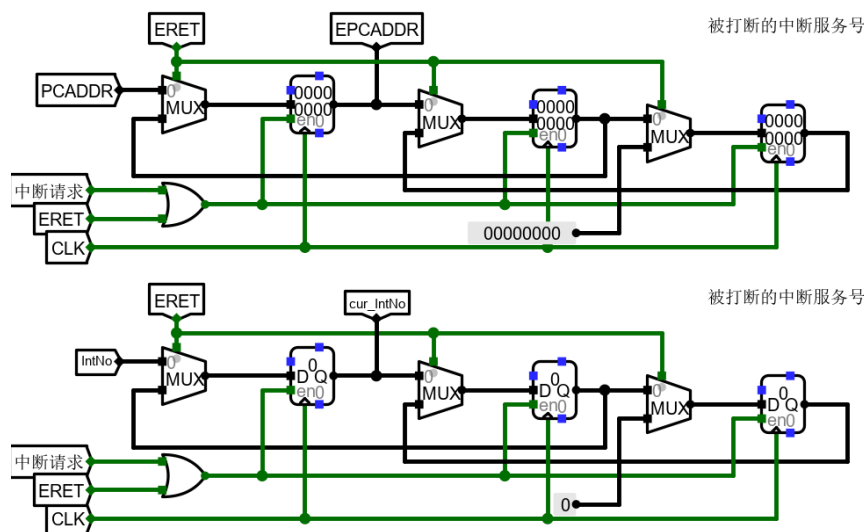


图 3.10 多级中断 EPC 寄存器堆栈和中断服务号堆栈

2. 开关中断信号产生逻辑

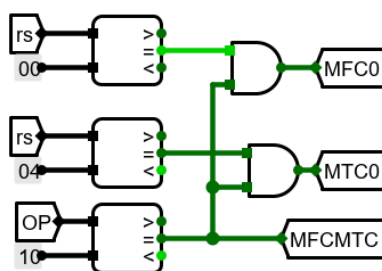


图 3.11 MFC0 和 MTC0 信号产生逻辑

3. IE 使能寄存器 and 中断请求产生逻辑

IE 使能通过 1 位宽的寄存器实现。关中断信号 MTC0 和中断请求信号相或作为寄存器的输入，中断请求信号、关中断、开中断信号 MFC0 和 ERET 信号相或作为寄存器的使能端，其中有一个为 1 时就触发时钟更改 IE 的值。在当有允许的中断请求产生时，程序执行 MTC0 指令，MTC0 信号为 1，寄存器被触发，从而 IE=1 关中断，而当程序执行完毕后，执行 MFC0 开中断指令，MFC0 信号为 1 再次触发寄存器，此时输入端为 0，所以 IE=0，实现开中断。根据优先级实现中断嵌套体现在当按键产生的中断如果比当前正在执行的中断程序的优先级高，IntR 与 cur_IntNo 比较结果输出为 1，再与 IE 寄存器的输出取反相与后，会在 IE=1 关中断时产生被允许的中断请求信号。电路如图 3.12 所示。

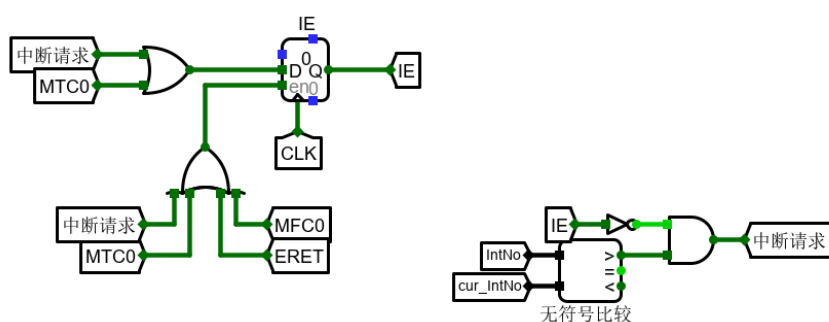


图 3.12 IE 中断使能寄存器和中断请求信号产生逻辑

3.2.3 流水中断

流水中断是基于气泡流水线和单中断实现的。由于是异步中断，不一定求立即响应，可以在处理器方便的时候进行处理。设定当流水线检测到异步中断时，在 WB 阶段进行中断响应。如果响应中断时，位于 WB 阶段的指令是分支指令，且分支跳转成功，或者如果当前由于数据冲突 WB 阶段的指令为空指令，则 EPC 寄存器保存的 WB.PC+4 就是错误的断点地址；所以我们认为此时暂时不响应该中断，所以在单周期中断逻辑部分，我们修改中断请求信号的产生逻辑和 EPC 寄存器，分别如图 3.13 和图 3.14 所示。

当中断请求产生或者 ERET 信号为 1 时需要清除 IF/ID、ID/EX，EX/MEM 流水寄存器的误取指令。

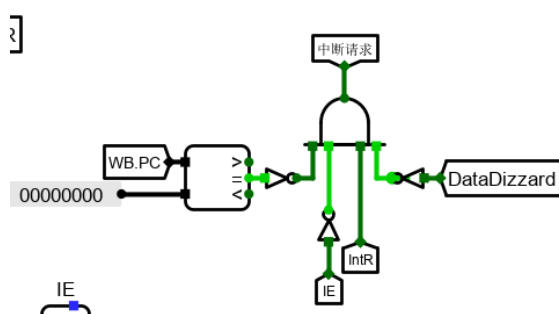


图 3.13 流水中断中断请求信号产生逻辑

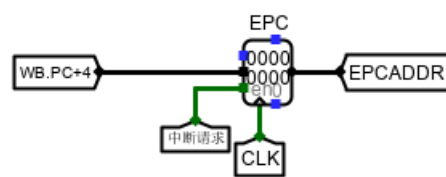


图 3.14 流水中断 EPC 寄存器设计

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

根据表 2-6 我们设计流水接口部件的电路。图 3.15 给出气泡流水线 IF/ID 部分。

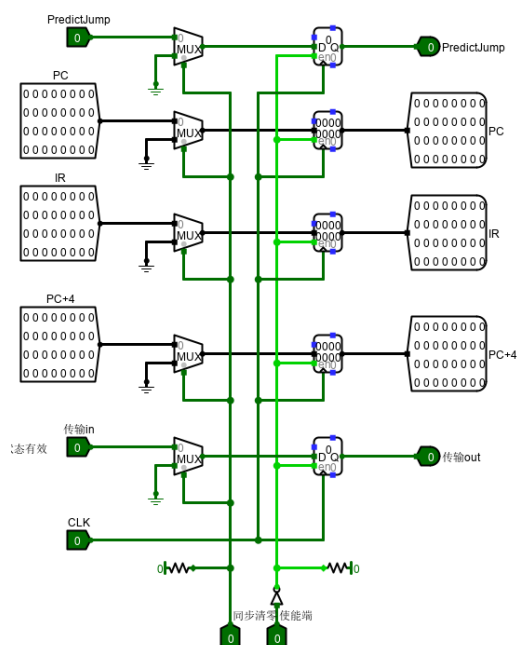


图 3.15 ID/ID 流水接口部件电路

3.3.2 理想流水线实现

在单周期的基础上，将指令执行的操作分为五个阶段，并简化跳转指令，删除有关跳转逻辑的部分，最后得到的电路如图 3.16 所示。

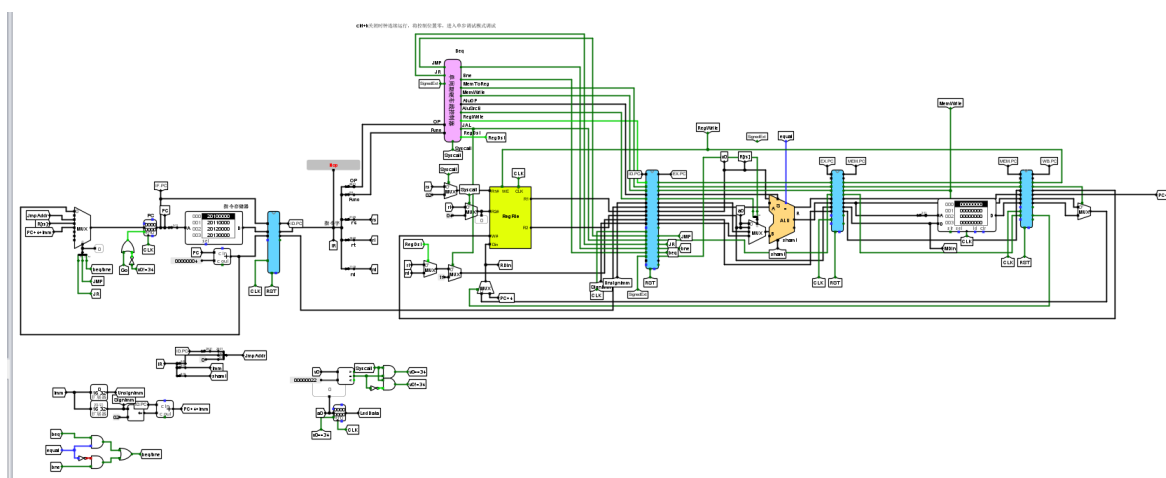


图 3.16 理想流水线通路

3.4 气泡式流水线实现

1. 在理想流水线的基础上补充跳转指令的逻辑并完善通路。
2. 实现流水线分支跳转逻辑

(1) 完成 EX 段分支地址与 IF 段 PC 输入多路选择器选择信号的连接。在 EX 段计算出分支跳转的地址，并作为 PC 寄存器的选择端输入。

(2) 实现分支相关的气泡逻辑。当 EX 段为 bne 指令且 euqal 为 0 或为 beq 指令 equal 为 1 时或者为无条件转移指令 JMP 信号为 1 或 JR 信号为 1 时产生分支清零的信号。

3. 实现数据相关检测逻辑。

(1) 构建源寄存器使用情况子电路。输入为 OP,Func, 输出为 R1_USED,R2_USED。填写控制器真值表利用 Logisim 自动生成电路。表如图 3.17 所示。电路如图 3.18 所示

#	指令	OpCode (十进制)	Func1 (十进制)	R1_USED	R2_USED
1	SLL	0	0		1
2	SRA	0	3		1
3	SRL	0	2		1
4	ADD	0	32	1	1
5	ADDU	0	33	1	1
6	SUB	0	34	1	1
7	AND	0	36	1	1
8	OR	0	37	1	1
9	NOR	0	39	1	1
10	SLT	0	42	1	1
11	SLTU	0	43	1	1
12	JR	0	8	1	
13	SYSCALL	0	12	1	1
14	J	2	X		
15	JAL	3	X		
16	BEQ	4	X	1	1
17	BNE	5	X	1	1
18	ADDI	8	X	1	
19	ANDI	12	X	1	
20	ADDIU	9	X	1	
21	SLTI	10	X	1	
22	ORI	13	X	1	
23	LW	35	X	1	
24	SW	43	X	1	1
25	ERET	16	24		
26	SLLV	0	4	1	1
27	SUBU	0	35	1	1
28	LHU	37	X	1	
29	BLEZ	6	X	1	

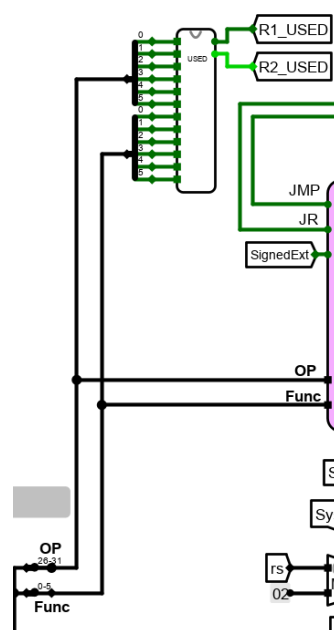


图 3.17 源寄存器使用情况真值表

图 3.18 源寄存器使用信号产生电路

(2) 构建数据相关检测逻辑子电路。输入为 :R1_USED, R2_USED,R1#,R2#,EX.WriteReg#, MEM.WriteReg#, RegWrite, 输出为数据相关信号 DataDizzard、EX 相关信号 EX_CONFLICT 和 MEM 相关信号 MEM_CONFILCT。其实现逻辑表达式如图 3.19 所示。其实现电路如图 3.20

所示。

```

DataHazard = RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#)
            + RtUsed & (rt≠0) & EX.RegWrite & (rt==EX.WriteReg#)
            + RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#)
            + RtUsed & (rt≠0) & MEM.RegWrite & (rt==MEM.WriteReg#)
# rs、rt 分别表示指令中的 rs、rt 字段，分别对应指令字中的 25~21、20~16 位
# RsUsed、RtUsed 分别表示 ID 段指令需要读 rs、rt 字段对应的寄存器
# EX.RegWrite 表示 EX 段的寄存器堆写使能控制信号 RegWrite，锁存在 ID/EX 流水寄存器中
# MEM.WriteReg# 表示 MEM 段的写寄存器编号 WriteReg#，锁存在 EX/MEM 流水寄存器中
    
```

图 3.19 源寄存器使用情况逻辑表达式

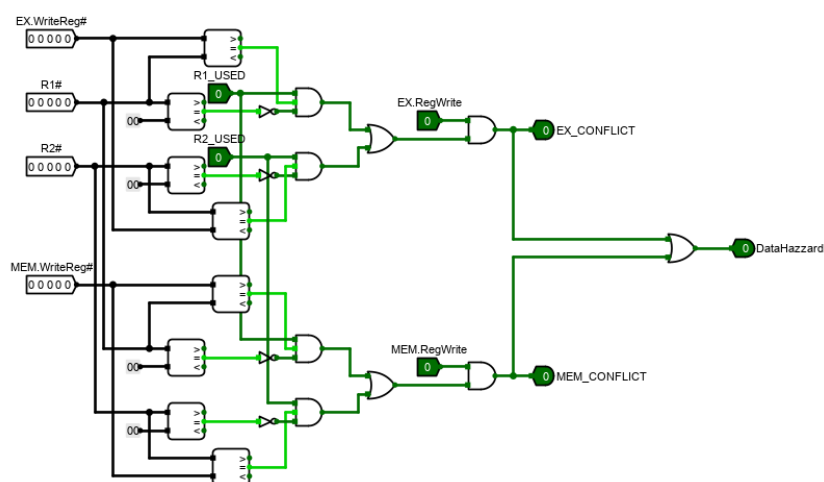


图 3.20 数据相关检测逻辑电路

4. 实现 ID，ID 段暂停逻辑和 EX 段气泡逻辑

各寄存器的暂停和清除信号表达式如图 3.21 所示

```

Stall = DataHazard # 数据相关时要阻塞暂停 IF、ID 段指令的执行
PC.EN = ~Stall # 程序计数器 PC 使能端输入
IF/ID.EN = ~Stall # IF/ID 寄存器使能端输入
IF/ID.CLR = BranchTaken # 出现分支跳转时要清空 IF/ID
ID/EX.CLR = Flush = BranchTaken + DataHazard # 出现分支或数据相关时要清空 ID/EX
    
```

图 3.21 暂停和气泡表达式

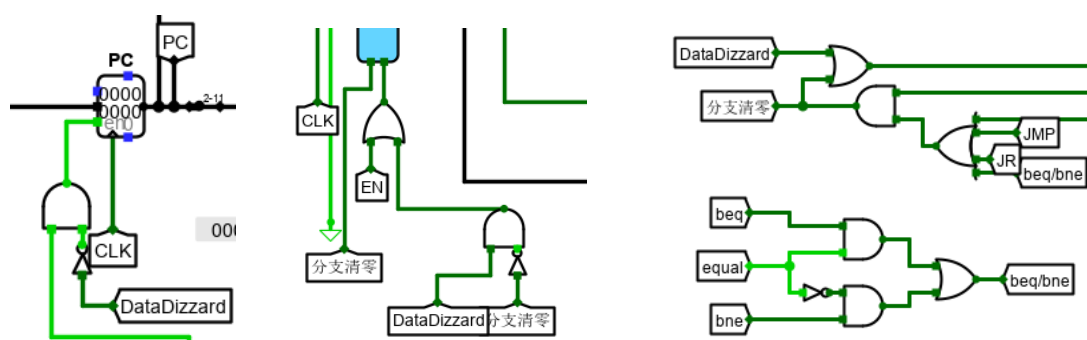


图 3.22 暂停和气泡逻辑实现电路

3.5 重定向流水线实现

重定向流水线将在气泡流水线的基础上进行一定的修改。主要分为四个步骤：

1. 构建重定向通路。需要在第一次使用寄存器的位置增加多路选择器并连接可能的重定向通路。通路图 3.26 如所示
2. 构建重定向逻辑。在 ID 段根据数据相关情况产生对应的重定向控制信号。以 RsForward 例，其表达式如图 3.23 所示。

```
IF (RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#))
    RsFoward = 2          # ID 段与 EX 段数据相关
else IF (RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#))
    RsFoward = 1          # ID 段与 MEM 段数据相关
else RsFoward = 0        # 无数据相关
```

图 3.23 RsForward 选择信号表达式

3. 构建 Load-Use 检测器。Load-Used 的逻辑表达式如图 3.24 所示

```
LoadUse = RsUsed & (rs≠0) & EX.MemRead & (rs==EX.WriteReg#)
+ RtUsed & (rt≠0) & EX.MemRead & (rt==EX.WriteReg#)
# 注意单周期 CPU 实现中为了简化电路，只实现了 MemWrite 写信号，没有实现 MemRead 信号，但由于该信号和 MemToReg 信号是同步的，所以可以用 MemToReg 信号代替 MemRead 信号
```

图 3.24 Load-Used 逻辑表达式

最终重定向选择信号和数据冲突检测信号合并的子电路设计如图 3.25 所示。

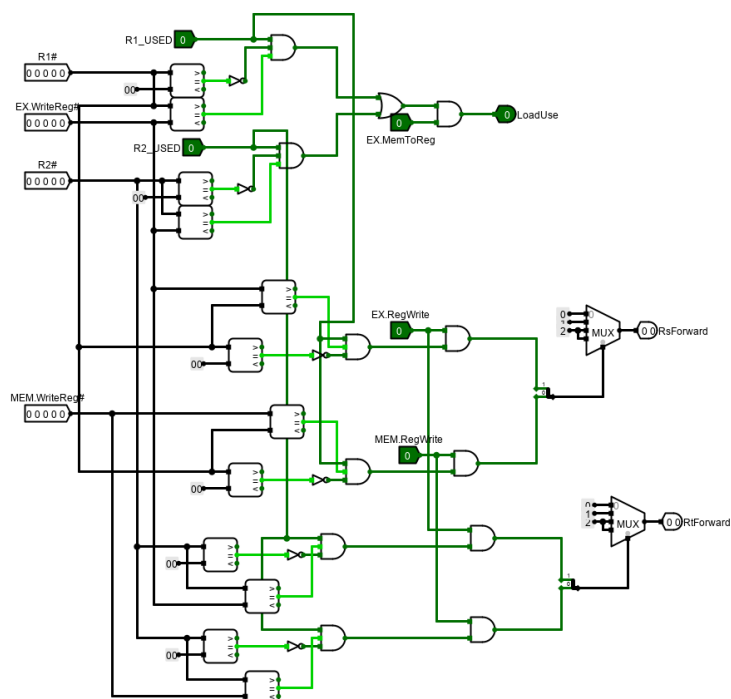


图 3.25 重定向选择信号和数据冲突检测信号实现电路

- 在 ID 段增加 Load-Use 插入气泡逻辑。在气泡流水的基础上将 DataDizzard 信号全部换为 Load-Used 信号即可。

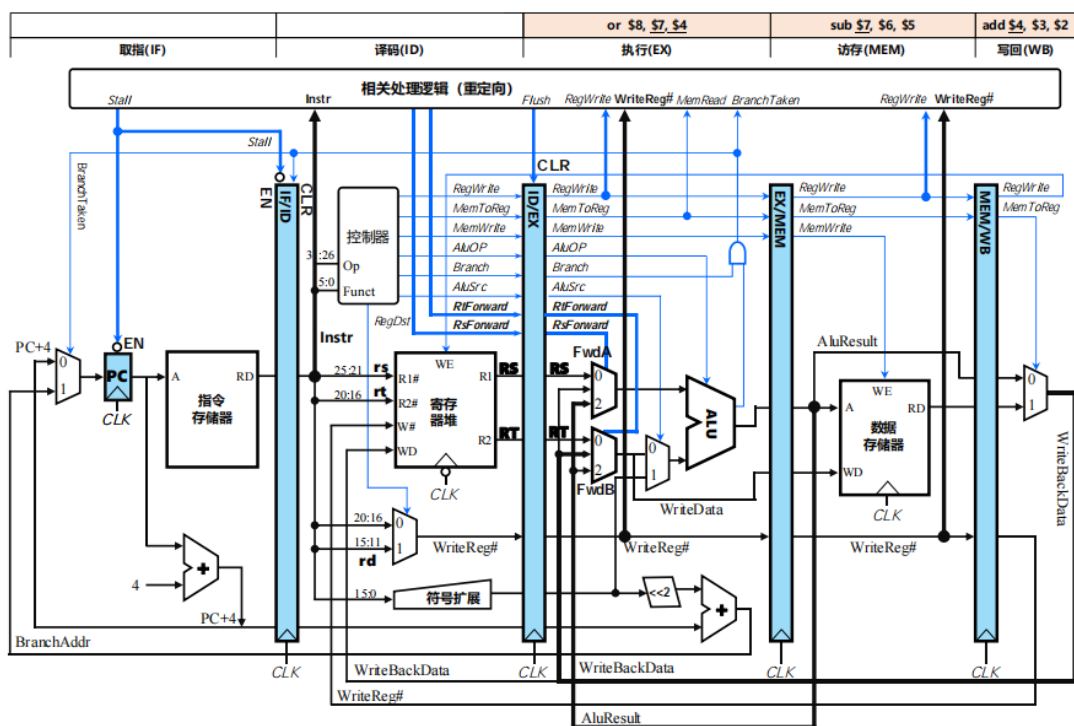


图 3.26 重定向流水线数据通路

3.6 动态分支预测机制实现

1. 历史预测位状态转换机的实现。

根据图 所示的双预测位历史状态转换图，在 Logism 中填写对应的真值表即可自动生成状态机电路。如图 3.27 所示。

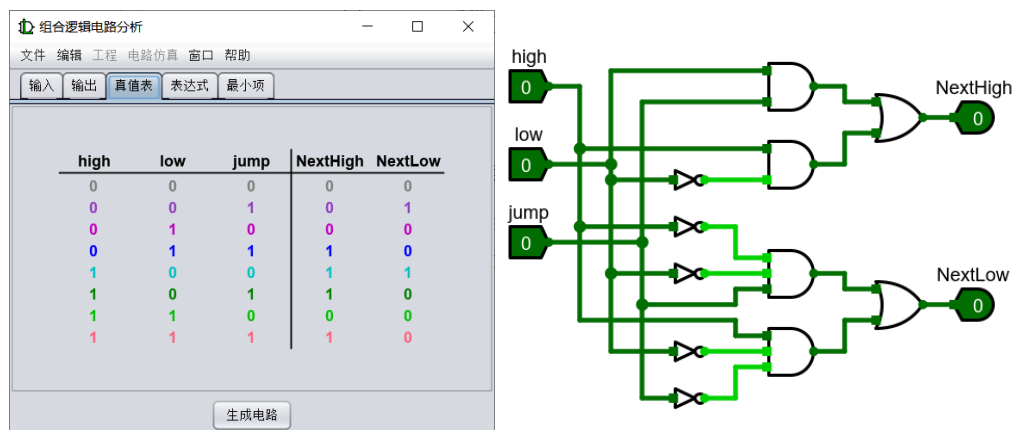


图 3.27 双预测位状态转换机真值表和实现电路

2. 实现 BTB8 项全相联 cache 结构

(1) cache 槽。

包括有效位、标志、淘汰计数器和分支跳转地址。淘汰计数器采用 LRU 的思想，淘汰最近不常使用的槽，即淘汰时，计数器计数值最大的 cache 槽，而在该 cache 槽被淘汰或被写 write 信号为 1 或者命中 L 信号为 1 时将计数置零重新开始表示最近被使用。Tag 寄存器中保存着曾经跳转执行成功过的分支指令地址。有效位为 1 表示当前 cache 槽内有数据。EX 的分支目标地址在命中且写信号为 1 时触发寄存器写入输出为 BTB 的预测分支目标地址。

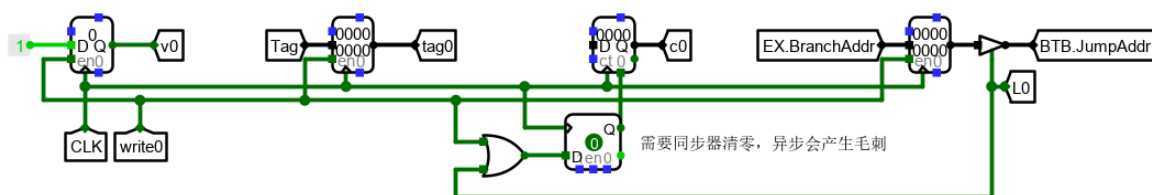


图 3.28 cache 槽

(2) 查找、写入和更新预测位逻辑

根据图 BTB 的处理流程，当前 IF 段的 PC 值和 BTB 中的分支指令地址进行比较查找。如果比较结果相等且当前有效位为 1 则产生 L 命中信号，并且根据分

华中科技大学课程设计报告

支预测实力位的值输出预测跳转信息位 PredictJump。当 EX 阶段执行分支指令，即 EX.Branch 为 1 时，与 BTB 中的分支指令地址进行比较查找，如果找到产生 M 命中信号，此时需要根据 EX 段的实际跳转情况 EX.Branched 更新预测位状态机中的信息。如果 BTB 中所有的槽的 M 信号都为 0 表示这个分支地址缺失，需要写入，且如果此时 BTB 是满的，采用 LRU 算法淘汰掉最近最不常使用的槽。



图 3.29 查找逻辑和 EX 写入逻辑

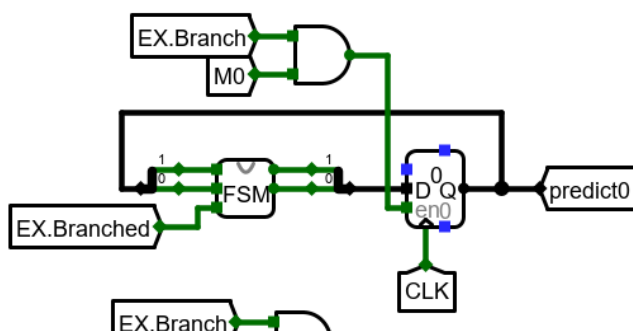


图 3.30 预测位状态机更新逻辑

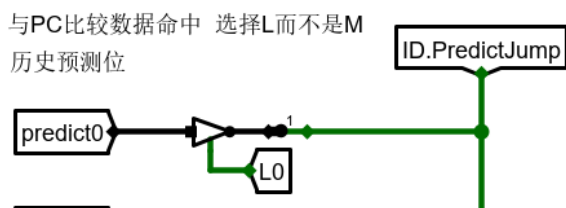


图 3.31 预测跳转信息逻辑

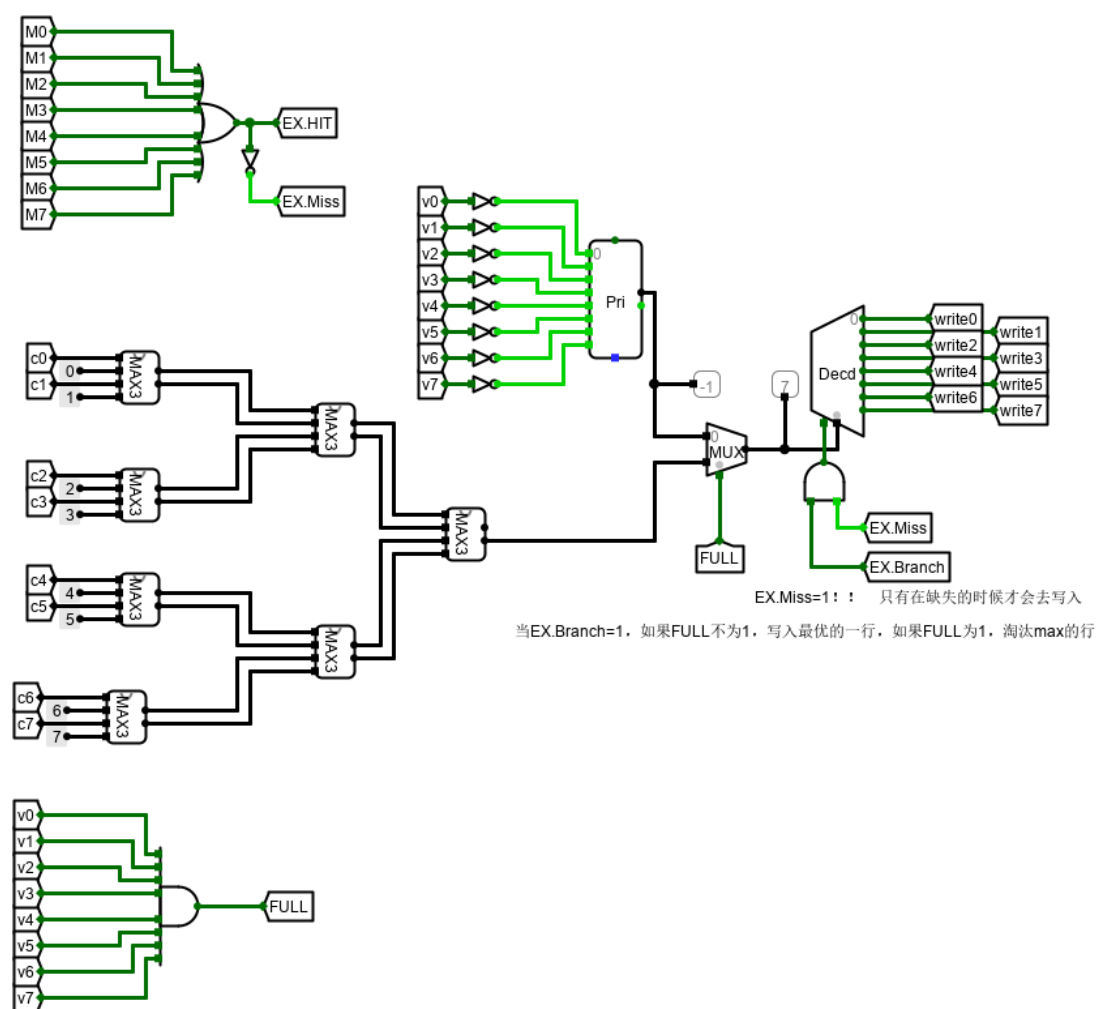


图 3.32 cache 槽写信号产生和选择淘汰逻辑

- 根据图 2.7 给出的动态分支预测通路完成连线。包括 BTB 连接和气泡产生部分。气泡产生的实现是当 BTB 预测失败 PredictErr 信号为 1 时，分别输送到 IF/ID 和 ID/EX 的清零端。BTB 的主要部分如图 3.33 所示。

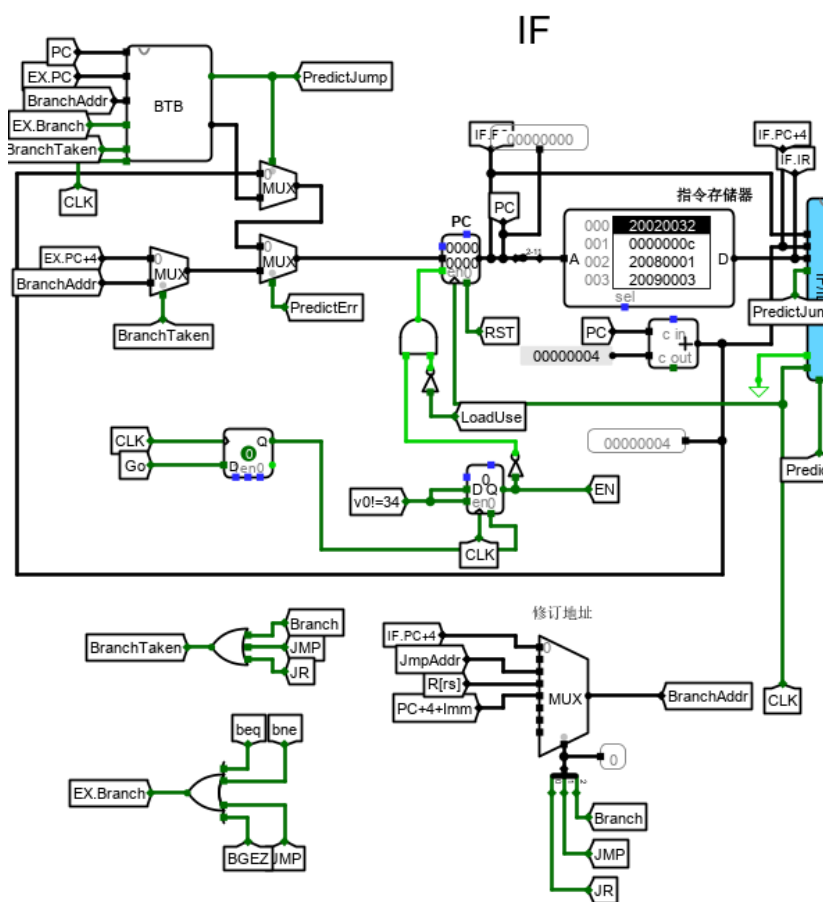


图 3.33 BTB 主要实现部分

4 实验过程与调试

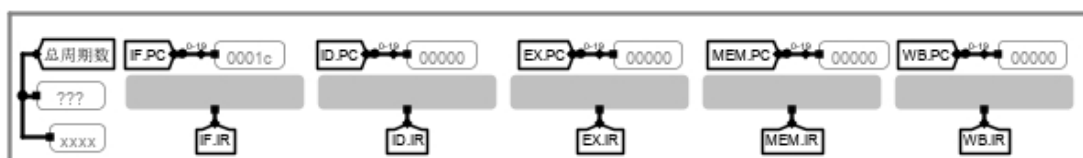
4.1 性能分析

运行同一段程序 bechmark, 单周期电路周期数为 1546, 气泡流水线的周期数为 3624, 重定向流水线的周期数为 2298, 使用动态分支预测后的重定向流水线为 1782。流水线的时钟周期数虽然比单周期多, 但是在单位时间内处理的指令数却大于单周期。遇到分支指令和数据相关较多的程序时, 气泡流水线会产生较多的气泡和停顿, 而重定向流水由于很大程度上解决了数据冲突, 仅在 Load 指令时需要进行停顿, 性能上相比气泡流水要改善很多, 性能提高约 1.53 倍。由于程序的空间局部性和时间局部性, 程序会执行重复的操作, 动态分支预测基于这一点采用合理的算法提前预测下一次的分支, 能够减少由于分支指令带来的气泡, 性能提高了 0.78 倍。

4.2 主要故障与调试

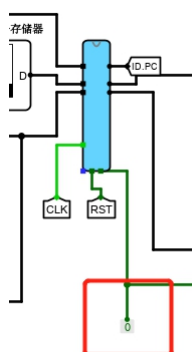
4.2.1 理想流水线故障

故障现象: 执行指令时 PC 的值无法传递。如下图所示:



原因分析: 查看 if/idID 电路, 在时钟运行时, IF.PC 值变化无法传递到选择器的输出, 选择器的选择段蓝色意味着无法确定的值。

解决方案: 在外电路中将选择器的选择信号置 0, 如下图所示。

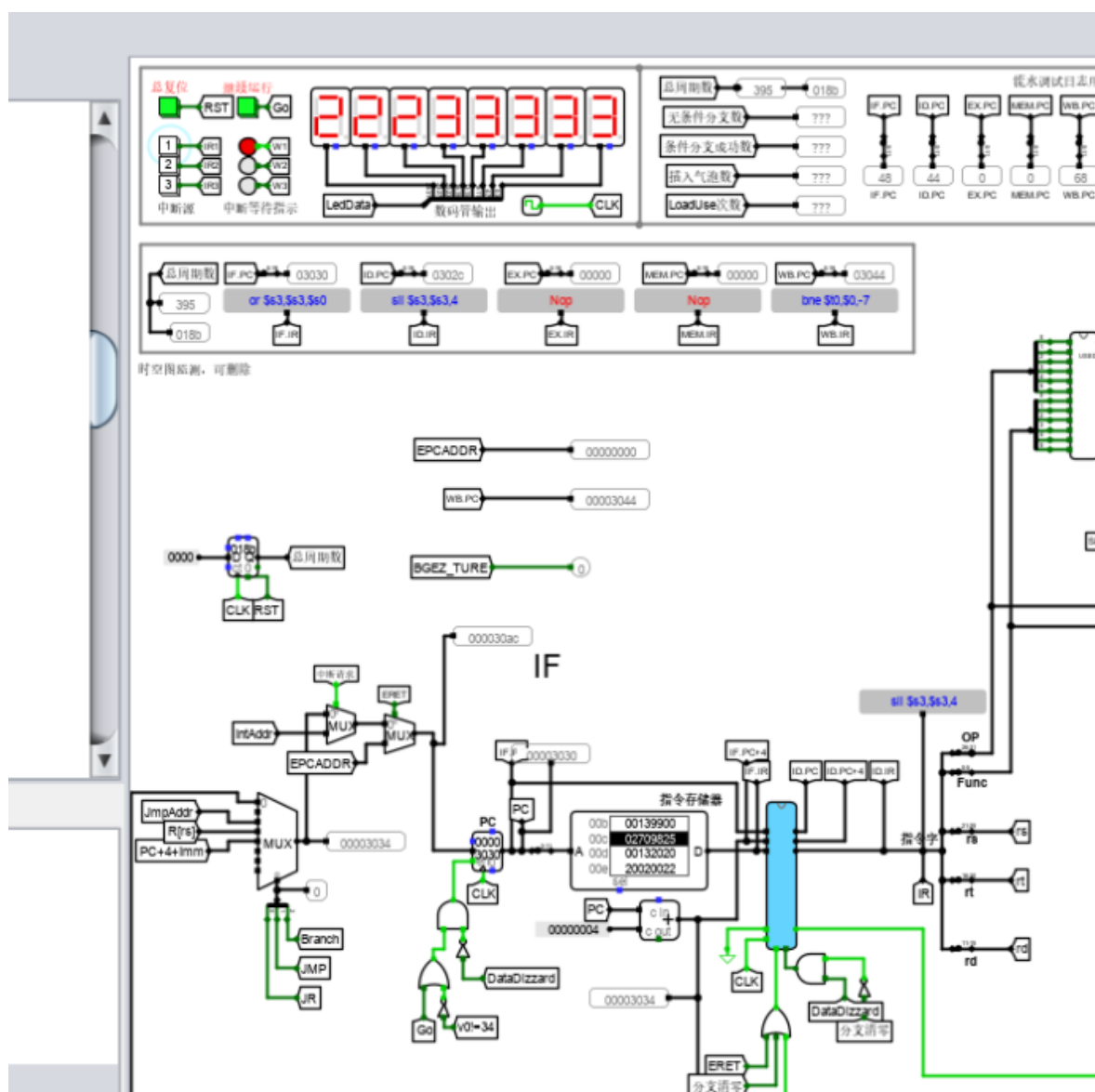


华中科技大学课程设计报告

4.2.2 流水中断故障

故障现象：在测试流水中断的时候，发现有时中断程序执行完毕后无法恢复到现场。

原因分析：单步调试后发现保护现场时传入给 EPC 寄存器的 WB.PC 为 0，发现中断响应的时候正在 WB 阶段的是气泡，而这时不应该响应中断，可以通过设置一个与门，和 WB.PC 进行相与得到中断请求的信号。在测试流水中断的发现，如果响应中断的时候真正 WB 阶段的是分支跳转指令（beq/bne 等），此时传入给 EPC 的指令地址 WB.PC+4 并不是正确的下一条要执行指令，如下图所示



解决方案：如果将中断响应的前提 WB.PC+4 不为 0 改为 WB.PC 不为 0，可以避

华中科技大学课程设计报告

免上述情况的发生，就是会延迟中断响应。

4.3 实验进度

表 4.1 课程设计进度表

时间	进度
寒假期间	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，完成了单周期 MIPS
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 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 Logism 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 的故障报告，并且通过了 Logism 单级中断 CPU 的检查。
第四天	完成了理想流水线，设计了理想流水线 CPU 的理想流水接口，并进行了改进设计了对应的 IF/ID、ID/EX、EX/MEM、MEM/WB 接口，了解了气泡流水的相关概念
第五天	完成了气泡流水和重定线，在研究流水中断，由于单级中断的 ERET 指令和额外的差异化指令，会带来很多的改动，目前计划是先查阅资料，弄清数据通路的控制过程。
第六天	完成了流水中断的数据通路，以及差异化指令的数据通路设计，解决中断过程中的控制信号的实现
第七天	能够初步实现流水中断的调试，但进入中断的过程还会出现错误，进中断的过程有待改进
第八天	对流水中断中存在的问题进行了改进，查阅了关于流水中断在触发进中断过程的资料
第九天	通过前一天对流水中断中进中断过程的进一步了解，对流水中断做了进一步的完善，能够顺利的实现流水中断。
第十天	完成了分支预测的电路设计，使运行程序的周期降到 1782，达到了老师的要求

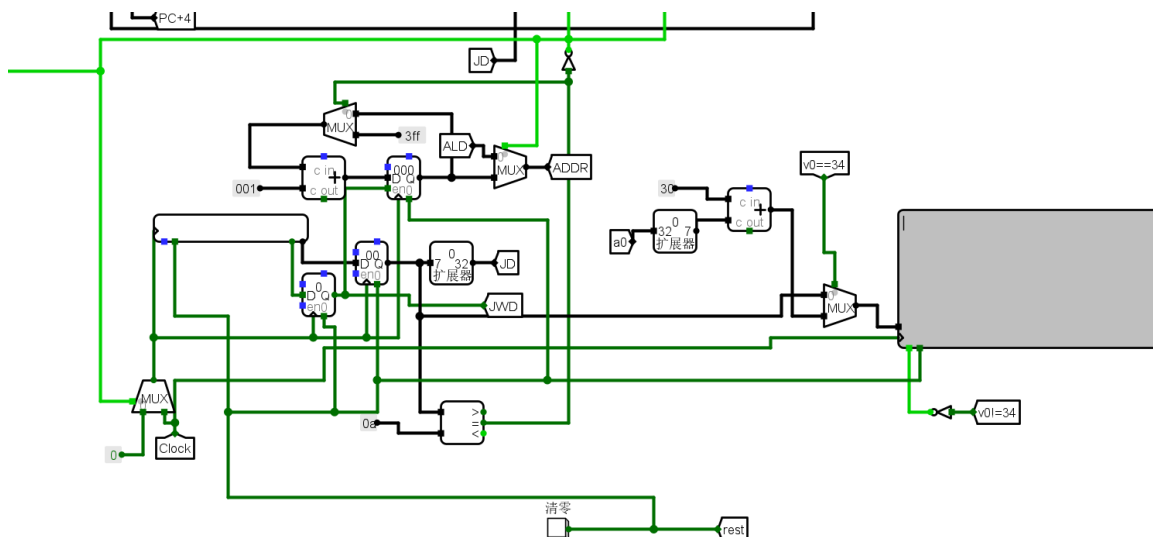
5 团队任务

5.1 选题与设计

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。

关于整个的选题过程，我们小组最开始的想法是设计一个视频播放器，后来做出来之后发现图片像素点过多，在 LCD 屏幕上描绘出来时，加载时间过长，无法达到动态连续播放视频的效果。所以果断的更改了课题，制作一个简单的计算器。

实现计算器的加减乘除，并将表达式和结果在文本显示器上显示出来。如下图所示是对单周期 CPU 增加的电路部分。



5.2 负责部分

我在本次团队任务中，主要负责的是 CPU 数据通路的改造，以及协调整个团队的分工与合作，安排整个的计划流程。

团队的分工：

数据通路的搭建和改造——刘汉鹏

程序代码的实现——刘美

github 资料的查阅及整理——吴志远

实验日志的记录——凯撒·衣马木艾山

华中科技大学课程设计报告

在电路的改造中主要是通过 logisim 的元件键盘读取输入的表达式，并将表达式的数字和运算符进行解析储存到数据存储器中。由于是先输入再进行 CPU 的运算，所以需要对回车键进行解析，标志输入的结束，计算的开始。

每输入一个数，数据寄存器的地址进行自动加一，同时将其显示在文本显示器上，当检测到回车键“\n”时，地址寄存器归零，CPU 时钟信号正常输入，键盘驱动的时钟信号归零。开始进行求解，当程序运行结束后，通过 Syscall 指令，将数据存储器中的运算结果显示到文本显示器上，这期间存在着键盘驱动时钟信号和 CPU 时钟信号的切换，而文本显示器的时钟信号是一直进行着刷新。

此外，我们还设计了归零按钮，方便进行多次计算。

代码如下：

```
addi $s0,$zero,0    #A 的地址
```

```
addi $s1,$zero,4    #+的地址
```

```
addi $s2,$zero,8    #B 的地址
```

```
start:
```

```
    lw $s0,0($s0)
```

```
    sub $s0,$s0,0x30
```

```
    lw $s1,0($s1)
```

```
    lw $s2,0($s2)
```

```
    sub $s2,$s2,0x30
```

```
    beq $s1,0x2b,ad
```

```
    beq $s1,0x2d,su
```

```
    beq $s1,0x2a,mu
```

```
    beq $s1,0x25,di
```

```
ad: add $s0,$s0,$s2
```

```
    j    display
```

```
su:  sub $s0,$s0,$s2
```


华中科技大学课程设计报告

```
j display
```

```
mu:  mul $s0,$s0,$s2
```

```
j display
```

```
di: div $s3,$s0,$s2
```

```
addu  $a0,$0,$s3      # display $t0
```

```
addi   $v0,$0,34       # system call for LED display
```

```
syscall                                # display
```

```
addi   $v0,$zero,10     # system call for exit
```

```
syscall                                # we are out of here.
```

```
display:
```

```
addu  $a0,$0,$s0      # display $t0
```

```
addi   $v0,$0,34       # system call for LED display
```

```
syscall                                # display
```

```
addi   $v0,$zero,10     # system call for exit
```

```
syscall                                # we are out of here.
```

在团队任务的后期，我还负责了视频的录制以及视频的发布。视频连接：

<https://www.bilibili.com/video/BV1GA411V7iW/>

6 设计总结与心得

6.1 课设总结

本次实验完成了 MIPS 单周期和 5 段流水线的设计，并尝试了气泡流水和重定向流水来解决 MIPS 指令流水线中的各类冒险冲突，最终设计完成的 CPU 能够运行基本的 24 条指令和差异化指令，并进一步增加了中断异常处理机制。作了如下几点工作：

- 1) 完成方案总结：设计了单周期 CPU、理想流水线 CPU、气泡流水线 CPU、定向流水线 CPU、支持中断以及分支预测的重定向流水线、以及流水中断 CPU 且扩展设计了 ccmb 四条指令。
- 2) 功能总结：将差异化指令加载在了动态分支预测的基础之上，这就要求重定向流水线的通路以及控制信号器的重新搭建，实验难度又上了一个档次。
- 3) 团队任务总结：与同学一起完成了团队任务的选题、设计、具体构建，并且自己搭建电路以支持相应的功能，完成了团队任务。

6.2 课设心得

本次课程设计可以说是迄今为止所有实验以及课程设计中难度最大的一门。两个星期从早到晚的不懈努力才终于完成了整个课程设计的设计任务。现在再来回顾整个课程设计的整个过程，满满的成就感自是不用说，但是其中也有不少细节值得我去深思与体会。

课程设计刚刚开始的时候，第一个任务是使用 Logism 设计单周期 CPU，这个任务的难点在于对整个数据通路的理解和掌握，清楚地知道每个控制信号在每个阶段对每个器件的控制作用，难点在于对整体的把握。单级中断是在单周期 CPU 的基础上实现的，主要是增加了 ERET 控制信号，实现对中断程序的加载，在将单周期 CPU 完全理解的基础上进行改造非常的容易。多级中断是基于 EPC 堆栈实现的，主要的难点在于对中断信号优先级的判断，以及对于多次进入中断时现场的保护。由于老师给的资料较少，做任务时出现的错误较多。

紧接着，理想流水线 CPU 的设计并没有什么难度。在使用插入气泡、数据重定向技术对于流水线 CPU 进行冒险处理时，老师给的新书有了详细的描述和介绍，做

华中科技大学课程设计报告

起思路非常的清晰，对于自己出现的错误也能做出及时的纠正。

然而对于本次课程设计，我还有一些小小的建议和改进。本次课设设计对于中断的介绍非常的少，特别是对多级中断的介绍，导致花费了大量的重复劳作的时间，希望老师们对于多级中断的实现能够有更加详细的介绍。此次团队任务中，跟小组成员之间也有较深的交流，避免了重复错误的出现，节约了大量的时间。但是对于实验最后的任务 FPGA 的上板，由于存在着软件版本不同，还有许多莫名的报错显示，使我异常的烦躁，尽管在群里获得了许多老师和同学的帮助，但是还是用很多错误无法解决，打消我要上板的想法。希望老师们能够像流水线一样能够给出上板的一些详细介绍和常规报错的解决办法。

最后在这里也感谢三位老师对于我在本次课程设计中无数问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。我相信组成原理课程设计必将办的越来越好，对学生的帮助也越来越大。

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] 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.
- [5] 谭志虎, 秦磊华, 胡迪青. 计算机组成原理实践教程. 北京: 清华大学出版社, 2018.
- [6] 袁春风编著. 计算机组成与系统结构. 北京: 清华大学出版社, 2011 年.
- [7] 张晨曦, 王志英. 计算机系统结构. 高等教育出版社, 2008 年.

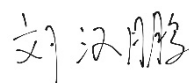
• 指导教师评定意见 •

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：刘汉鹏



44