

华中科技大学

2022

硬件综合训练

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： CS2008

学 号： U2020215533

姓 名： 徐瑞达

电 话： 17837353795

邮 件： 2014027378@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.....	12
2.3	理想流水线 CPU 设计	13
2.4	气泡流水线 CPU 设计	14
2.5	重定向流水线 CPU 设计	14
2.6	动态分支预测机制设计	15
3	详细设计与实现.....	16
3.1	单周期 CPU 实现	16
3.2	中断机制实现.....	19
3.3	理想流水线 CPU 实现	23
3.4	气泡流水线 CPU 实现	24
3.5	重定向流水线 CPU 实现	25
3.6	动态分支预测机制实现	27
4	实验过程与调试.....	29
4.1	测试用例和功能测试.....	29
4.2	性能分析	31
4.3	主要故障与调试.....	31
4.4	实验进度	33

华中科技大学课程设计报告

5	团队任务	34
5.1	选题介绍	34
5.2	任务分工	34
5.3	问题与解决方案.....	34
5.4	版本提交历史.....	37
6	设计总结与心得	38
6.1	课设总结	38
6.2	课设心得	38
	参考文献.....	40

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；
- (5) 设计出实现指令功能的硬布线控制器；

华中科技大学课程设计报告

- (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 Risc-V 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 RISC-V32 指令集，最终功能以 RARS 模拟器为准。
2	ADDI	立即数加	
3	AND	与	
4	ANDI	立即数与	
5	SLLI	逻辑左移	
6	SRAI	算数右移	
7	SRLI	逻辑右移	
8	SUB	减	
9	OR	或	
10	ORI	立即数或	
11	XORI	立即数异或	
12	LW	加载字	
13	SW	存字	

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
14	BEQ	相等跳转	
15	BNE	不相等跳转	
16	SLT	小于置数	
17	SLTI	小于立即数置数	
18	SLTU	小于无符号数置数	
19	JAL	转移并链接	
20	JALR	转移到指定寄存器	
21	ECALL	系统调用	if (\$a7==34) LED 输出\$a0 的值 else 暂停等待 Go 按键继续运行
22	CSRRSI	访问 CSR 寄存器	中断相关，可简化为开中断
23	CSRRCI	访问 CSR 寄存器	中断相关，可简化为关中断
24	URET	中断返回	清中断，mEPC 送 PC，开中断
25	AUIPC	立即数+PC 送寄存器	指令格式参考 RISC-V32 指令集，最终功能以 RARS 模拟器为准。
26	SLTIU	小于无符号立即数置数	
27	LHU	加载半字	
28	BGE	大于等于跳转	

2 总体方案设计

2.1 单周期 CPU 设计

本次设计单周期 CPU 采用的方案是硬布线控制器，且采用指令存储器和数据存储器相分离的结构以避免资源冲突。通过填写提供的 Excel 表格得到各种信号的产生逻辑，由此合并不同指令的数据通路来构造支持表 1.1 中指令集的单周期 CPU。在实现过程中，使用 Logisim 仿真平台提供的硬件构建电路，完成电路后导入 benchmark 程序进行联调测试。

总体结构图如图 2.1 所示。

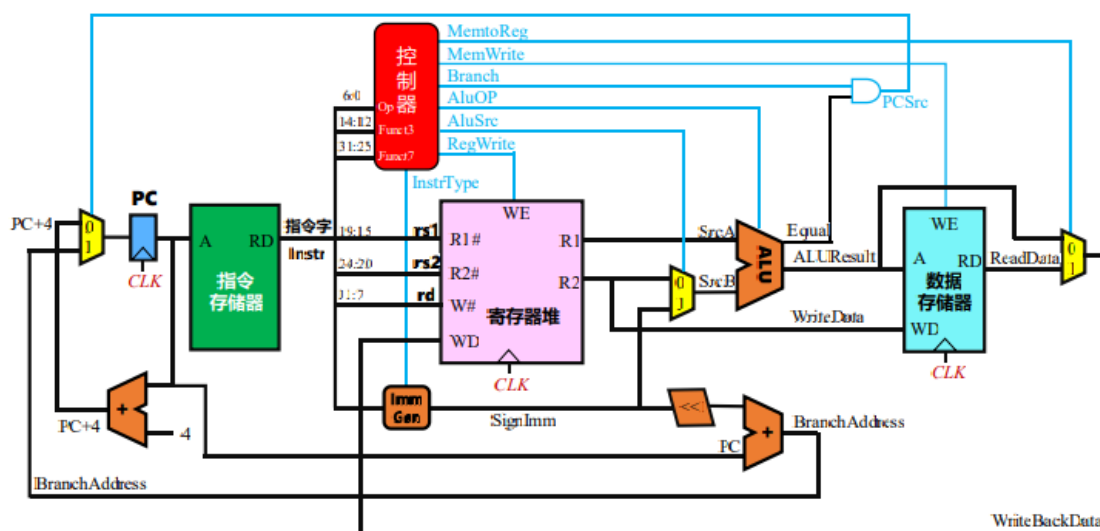


图 2.1 单周期 CPU 总体结构图

2.1.1 主要功能部件

主要的功能部件有程序计数器 PC、指令存储器 IM、寄存器堆 Regfile、运算器 ALU、数据存储器 DM。

1. 程序计数器 PC

程序计数器 PC 主要用于提供下一条指令的地址，以从指令存储器取出指令。一般情况下，程序计数器的数据来源是原 PC 值加 4 个字节得到的结果，也即当前指令

华中科技大学课程设计报告

的下一条指令的地址，其他数据来源也可能是条件跳转指令如 BEQ、无条件跳转指令 JAL。

2. 指令存储器 IM

指令存储器 IM 用于存储需要执行的指令，由于在执行过程中不可进行更改，因此在 Logisim 仿真平台中采用只读存储器 ROM 实现。在执行不同的测试程序时，需要加载对应的数据镜像。

3. 运算器 ALU

运算器用于进行各种运算，以从 Regfile 中读取的寄存器内容或者指令中的立即数扩展得到的值作为运算数，输出运算结果或者比较信号供其他电路使用。ALU 引脚功能描述如表 2.1 所示，支持的运算功能及运算码如表 2.2 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
ALU_OP	输入	4	运算器功能码，具体功能见下表 2.2
Result	输出	32	ALU 运算结果
Result2	输出	32	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
Equal	输出	1	$\text{Equal}=(x==y)?1:0$ ，对所有操作有效

表 2.2 运算码及其运算功能描述

ALU_OP	运算功能
0000	$\text{Result} = X \ll Y$ 逻辑左移 (Y 取低五位) $\text{Result2}=0$
0001	$\text{Result} = X \gg Y$ 算术右移 (Y 取低五位) $\text{Result2}=0$
0010	$\text{Result} = X \gg Y$ 逻辑右移 (Y 取低五位) $\text{Result2}=0$
0011	$\text{Result} = (X * Y)[31:0]$; $\text{Result2} = (X * Y)[63:32]$ 无符号乘法
0100	$\text{Result} = X/Y$; $\text{Result2} = X\%Y$ 无符号除法
0101	$\text{Result} = X + Y$ (Set OF/UOF)

华中科技大学课程设计报告

ALU_OP	运算功能
0110	Result = X - Y (Set OF/UOF)
0111	Result = X & Y 按位与

4. 寄存器堆 Regfile

寄存器堆是对 CS3410 包中 Regfile 的封装，提供了 32 个通用寄存器。具体寄存器说明如表 2.3 所示。

表 2.3 寄存器说明

寄存器	别名	说明
x0	zero	恒零寄存器
x1	ra	返回地址
x2	sp	栈指针
x3	gp	全局指针
x4	tp	线程指针
x5-x7	t0-t2	临时变量寄存器 0 到 2
x8	s0/fp	保存寄存器 0/栈帧指针
x9	s1	保存寄存器 1
x10-x17	a0-a7	函数参数 0 到 7
x18-x27	s2-s11	保存寄存器 2 到 11
x28-x31	t3-t6	临时变量寄存器 3 到 6

5. 数据存储器

数据存储器用来存储程序运行中产生的数据，需要支持读写操作，因此使用 RAM 来实现。在按字访问、半字访问、字节访问的不同模式下，需要使用多片 RAM 实现。

2.1.2 数据通路的设计

表 2.4 指令系统数据通路框架

华中科技大学课程设计报告

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
ADD	PC+4	PC	RS1	RS2	RD	ALU	RS1	RS2	5	—	—
SUB	PC+4	PC	RS1	RS2	RD	ALU	RS1	RS2	6	—	—
AND	PC+4	PC	RS1	RS2	RD	ALU	RS1	RS2	7	—	—
OR	PC+4	PC	RS1	RS2	RD	ALU	RS1	RS2	8	—	—
SLT	PC+4	PC	RS1	RS2	RD	ALU	RS1	RS2	11	—	—
SLTU	PC+4	PC	RS1	RS2	RD	ALU	RS1	RS2	12	—	—
ADDI	PC+4	PC	RS1	—	RD	ALU	RS1	Imm	5	—	—
ANDI	PC+4	PC	RS1	—	RD	ALU	RS1	Imm	7	—	—
ORI	PC+4	PC	RS1	—	RD	ALU	RS1	Imm	8	—	—
XORI	PC+4	PC	RS1	—	RD	ALU	RS1	Imm	9	—	—
SLTI	PC+4	PC	RS1	—	RD	ALU	RS1	Imm	11	—	—
SLLI	PC+4	PC	RS1	—	RD	ALU	RS1	Imm	0	—	—
SRLI	PC+4	PC	RS1	—	RD	ALU	RS1	Imm	2	—	—
SRAI	PC+4	PC	RS1	—	RD	ALU	RS1	Imm	1	—	—
LW	PC+4	PC	RS1	—	RD	DM	RS1	Imm	5	ALU	—
SW	PC+4	PC	RS1	RS2	—	—	RS1	Imm	5	ALU	R2
ECALL	PC+4	PC	\$a7	\$a0	—	—	RS1	RS2	—	—	—
BEQ	PC+OFFSET/PC+4	PC	RS1	RS2	—	—	RS1	RS2	—	—	—
BNE	PC+OFFSET/PC+4	PC	RS1	RS2	—	—	RS1	RS2	—	—	—
JAL	PC+OFFSET	PC	—	—	RD	PC+4	—	—	—	—	—
JALR	ALU	PC	RS1	—	RD	PC+4	RS1	Imm	5	—	—
CSRRSI	PC+4	PC	—	—	—	—	—	—	—	—	—
CSRRCI	PC+4	PC	—	—	—	—	—	—	—	—	—
URET	PC+4	PC	—	—	—	—	—	—	—	—	—
AUIPC	PC+4	PC	—	—	RD	PC+OFFSET	—	—	—	—	—
SLTIU	PC+4	PC	RS1	—	RD	ALU	RS1	Imm	12	—	—
LHU	PC+4	PC	RS1	—	RD	DM	RS1	Imm	5	ALU	—

华中科技大学课程设计报告

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
BGE	PC+OFFSET/PC+4	PC	RS1	RS2	-	-	RS1	RS2	11	-	-

2.1.3 控制器的设计

首先对控制信号进行统计，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 所示。

表 2.5 主控制器控制信号的作用说明

控制信号	取值	说明
ALU_OP	0-12	选择 ALU 要进行的运算类型，具体运算见表 2.2
MemToReg	0	选择 ALU 运算结果为寄存器写入值
	1	选择数据存储器读出值为寄存器写入值，如 LW、LHU 指令
MemWrite	0	-
	1	加载值到数据存储器，如 SW 指令
ALU_SrcB	0	ALU 的输入端送入 R2
	1	ALU 的输入端 B 送入立即数
RegWrite	0	-
	1	将 RDin 写入寄存器 Rd
ecall	0/1	ecall 指令译码信号
S_Type	0/1	S 型指令译码信号
BEQ	0/1	BEQ 指令译码信号
BNE	0/1	BNE 指令译码信号
JAL	0/1	JAL 指令译码信号
JALR	0/1	JALR 指令译码信号
BGE	0/1	BGE 指令译码信号
AUIPC	0/1	AUIPC 指令译码信号
LHU	0/1	LHU 指令译码信号
R1Used	0	未使用 R1#源寄存器

华中科技大学课程设计报告

控制信号	取值	说明
	1	使用 R1#源寄存器
R2Used	0	未使用 R2#源寄存器
	1	使用 R2#源寄存器

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 2.6 所示。

表 2.6 主控制器控制信号框架

指令	ALU_OP	MemToReg	MemWrite	ALU_SrcB	RegWrite	ecall	S_Type	BEQ	BNE	JAL	JALR	BGE	LHU	AUIPC	R1used	R2Used
ADD	5				1										1	1
SUB	6				1										1	1
AND	7				1										1	1
OR	8				1										1	1
SLT	11				1										1	1
SLTU	12				1										1	1
ADDI	5			1	1										1	1
ANDI	7			1	1										1	1
ORI	8			1	1										1	1
XORI	9			1	1										1	1
SLTI	11			1	1										1	1
SLLI	0			1	1										1	1
SRLI	2			1	1										1	1
SRAI	1			1	1										1	1
LW	5	1		1	1										1	
SW	5		1	1			1								1	1
ECALL						1									1	1
BEQ								1							1	1

华中科技大学课程设计报告

指令	ALU_OP	MemToReg	MemWrite	ALU_SrcB	RegWrite	ecall	S_Type	BEQ	BNE	JAL	JALR	BGE	LHU	AUIPC	RIUsed	R2Used
BNE									1						1	1
JAL					1					1						
JALR	5			1	1						1				1	
CSRRSI																
CSRRCI																
URET																
AUIPC					1											
SLTIU	12			1	1									1	1	
LHU	5	1		1	1								1		1	
BGE	11											1			1	1

2.2 中断机制设计

2.2.1 总体设计

中断可以分为外部中断和内部中断，而本次实验中仅考虑可以屏蔽的外部中断源。在设计中断时，应考虑单级中断和多级中断，需要硬件和软件的共同支持。对于单级中断的设计，在转到中断服务程序前，需要在中断断点寄存器 EPC 中维护断点（PC 的值）以确保执行完毕中断服务程序后能够返回进入中断的位置。对于多级中断，由于涉及到不同中断源的处理优先级，需要在硬件中判断新的中断能否打断当前中断，并使用堆栈存储处理多级中断时的不同断点值。

2.2.2 硬件设计

单级中断硬件设计：使用中断按键参考电路存储中断请求信号，当同时存在多个中断请求信号时，使用优先编码器输出优先级最高的中断号。当正在处理某个中断时，其他中断请求无法打断当前中断。在进行中断响应前，在中断断点寄存器 EPC 中维护发生中断时 PC 值。当中断服务程序执行完毕时，CPU 会根据 URET 指令执行中断返回操作，将 EPC 中的值送至 PC 以跳转到断点处以继续执行主程序。只有当进

华中科技大学课程设计报告

行中断响应时更新 EPC 中的值，并且需要根据中断号向 PC 送入正确的中断服务程序入口地址。

多级中断硬件设计：与单级中断不同，当正在处理某个中断时，其他比当前中断优先级高的中断请求可以打断当前中断。因此需要存储每次中断响应时的相关信息。首先，与单级中断只需设置一个 EPC 不同，三级中断需要设置 3 个 EPC 以存放至多 3 个中断同时发生时的 PC 值。其次，当在处理中断时，如果发生新的中断，需要根据中断优先级判断应该处理哪个中断（先执行优先级高的中断），如果要处理新的中断，需要将新的中断号和被打断的中断号保存在相应寄存器中。此外，控制器还需要支持开关中断指令 CSRRSI 和 CSRRCI，与中断请求信号和 ERET 指令一起组成是否响应中断的逻辑。

2.2.3 软件设计

软件设计方面，主要是支持 CSRRSI、CSRRCI、URET 指令，配合硬件完成开关中断、中断返回的操作。同时需要从程序中获取中断服务程序入口地址，作为常量连接到硬件相关逻辑电路中以达到模拟中断向量表的目的。

2.3 理想流水线 CPU 设计

2.3.1 总体设计

RISC-V 指令执行过程可以分为 5 个阶段：取指令阶段 IF、译码取数阶段 ID、指令执行阶段 EX、访存阶段 MEM、写回阶段 WB。通过在不同阶段之间插入流水寄存器锁存当前段的数据，使得不同阶段能够处理不同的指令，提高执行效率。理想流水线并不考虑分支冲突、数据冲突等冒险问题，是后续气泡流水线和重定向流水线的基础，因此需要在理想流水线设计中完成流水线寄存器的设计及流水线基本结构。

2.3.2 流水接口部件设计

在设计流水接口部件时，参考了设计运算器时使用到的乘法流水线接口部件。每个流水接口部件包括时钟端 CLK，同步清零端，使能端以及数据输入输出端口。由于不同阶段需使用的数据不同，因此不同阶段间的流水接口部件的数据端口不尽相同，需要根据实际情况修改流水寄存器需要所存的数据。

2.3.3 理想流水线设计

完成流水接口部件的设计后，只需要将单周期 CPU 设计中的不同阶段需要用到的部件分离，并在不同阶段间插入流水接口部件。在设计时需要注意细微的逻辑变化，如在写入寄存器时，Rd#、RDin、RegWrite 应该均来自 WB 段，否则会导致错误逻辑。

2.4 气泡流水线 CPU 设计

2.4.1 总体设计

气泡流水线对于分支冲突的处理方法：当在 EX 段需要进行分支跳转时，清除 IF/ID 流水寄存器和 ID/EX 流水寄存器的内容以清除误取指令。

气泡流水线对于数据冲突的处理方法：在译码 ID 段判断是否与 EX 段或 MEM 段发生数据相关，如果发生数据相关，则暂停 ID/EX 段流水寄存器的执行（使能端赋高电平）以插入空气泡，等待数据不再相关后继续推进流水线的执行。

2.4.2 气泡逻辑设计

- 数据相关模块：通过源寄存器的使用情况、EX 段或 MEM 段是否写入寄存器（RegWrite 信号）、ID 段寄存器 R1#/R2#与 EX 段 WriteReg#或 MEM 段 WriteReg#是否相等组成的逻辑来检测是否发生数据相关。
- 由数据相关信号、分支跳转信号、halt 信号输出 PC 寄存器的使能端信号、各个流水线寄存器的使能端和清零端信号，以送入各个部件中。

2.5 重定向流水线 CPU 设计

2.5.1 总体设计

重定向流水线对于分支冲突的处理方法与气泡流水线相同。

重定向流水线对于数据冲突的处理方法：为了解决气泡流水线中插入气泡过多的问题，当发生数据相关时，不再插入气泡，而是在下一个时钟周期将正确的尚未写回到寄存器中的数据重定向到 EX 段。

LoadUse 相关：如果相邻两条指令存在数据相关，且前一条指令是访存指令时，这种数据相关不能用重定向进行处理。如果采用重定向方式，需要将数据存储器的输

出重定向到 EX 段，使得关键路径包含访存阶段，从而加长关键路径影响 CPU 工作效率。因此，需要额外判断 LoadUse 相关，并使用插入气泡的方式解决冲突。

2.5.2 重定向逻辑设计

- 重定向选择信号模块：通过对气泡流水线中数据相关模块稍加修改，输出 Fwd1 和 Fwd2 信号，由 Fwd1 和 Fwd2 信号作为多路选择器的选择端以选择 EX 段需要重定向的正确数据。
- LoadUse 相关模块：通过源寄存器的使用情况、EX 段是否访存（MemToReg 信号）、ID 段寄存器 R1#/R2#与 EX 段 WriteReg#是否相等组成的逻辑来检测是否发生 LoadUse 相关。
- 由 LoadUse 相关信号、分支跳转信号、halt 信号输出 PC 寄存器的使能端信号、各个流水线寄存器的使能端和清零端信号，以送入各个部件中。

2.6 动态分支预测机制设计

2.6.1 总体设计

动态分支预测基于重定向流水线，通过优化对分支冲突的处理以减少插入气泡数。

2.6.2 BTB 表设计

- 分支预测缓冲器：在 BTB 表中用于存放分支指令的分支跳转历史统计信息，通过八路并发比较进行全相联查找，其包含的表项为 Valid 位、分支指令地址、分支跳转地址、分支预测历史位。
- 淘汰算法：在 BTB 表已满时，使用 LRU 淘汰算法进行淘汰置换。
- 双位状态预测机：以输入的现状和是否进行分支跳转的信号得到次态输出，其现态即为 BTB 表中的分支预测历史位，根据是否进行分支跳转来预测是否进行跳转。双位预测状态转换如图 2.2 所示。

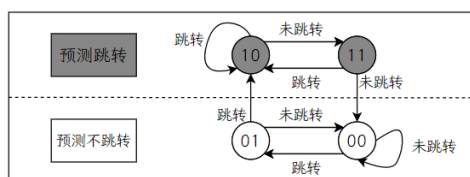


图 2.2 双位预测状态转换图

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送入使能端。当需要进行停机时，halt 控制信号为 1，经过非门之后为 0，从而忽略时钟输入，使整个电路停机。如图 3.1 所示。

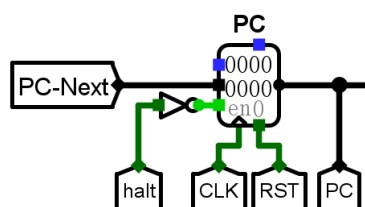


图 3.1 程序计数器 (PC)

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)，并设置指令存储器的地址位宽为 10 位，数据位宽为 32 位。而 PC 中存储的指令地址有 32 位，因此需要将 32 位指令地址高位部分和字节偏移部分屏蔽后送入 ROM 作为指令地址得到指令字 IR。实现时，分线器取 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。

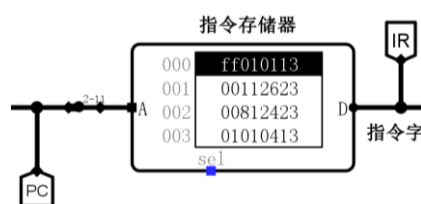


图 3.2 指令存储器 (IM)

3) 运算器 ALU

使用 cs3410.jar 包中 MIPS ALU 部件即可，输入为运算数 A、运算数 B、运算码 Alu_OP，输出为运算结果 Result、是否相等 Equal 等。

4) 数据存储器

由于存在需要按半字访问数据存储器的指令，因此使用 4 片随机存储器 RAM 实

华中科技大学课程设计报告

现数据存储器。每片 RAM 的地址位宽为 10 位，数据位宽为 8 位。在按字节访问时使用低两位作为片选信号来片选 4 片 RAM,按半字访问时使用第 1 位作为片选信号。如图 3.3 所示。

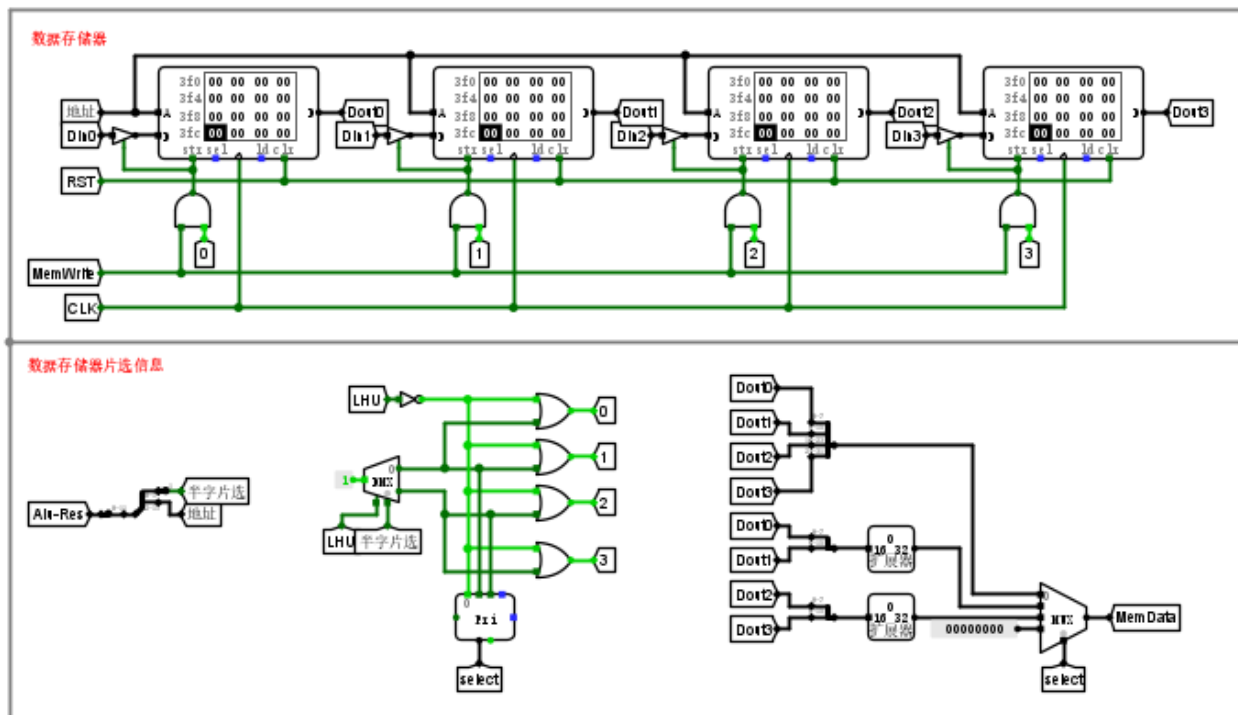


图 3.3 数据存储器

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。根据总体方案设计中数据通路设计的内容，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 2.4 指令系统数据通路框架所示。

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，对于所有的多输入部件使用多路选择器进行输入选择，最终完成数据通路的搭建。具体数据通路如图 3.4 所示。

华中科技大学课程设计报告

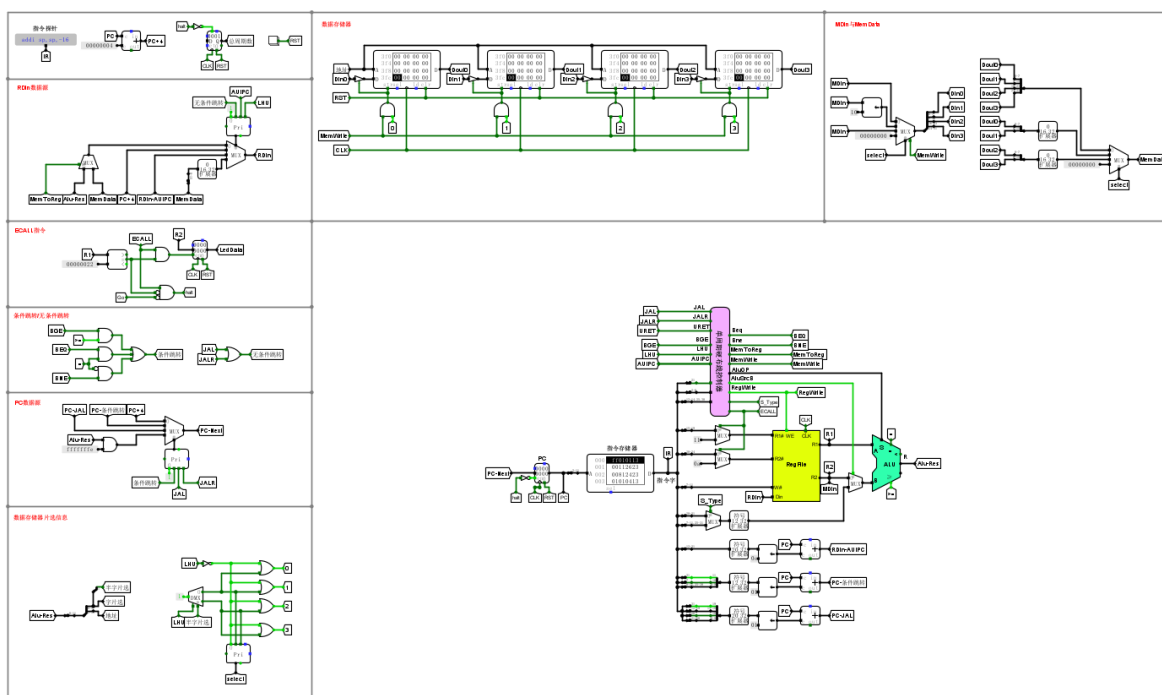


图 3.4 单周期 CPU 数据通路 (Logisim)

3.1.3 控制器的实现

1) 主控制器

主控制器控制信号如表 2.6 所示，主控制器的具体实现如图 3.5 所示。

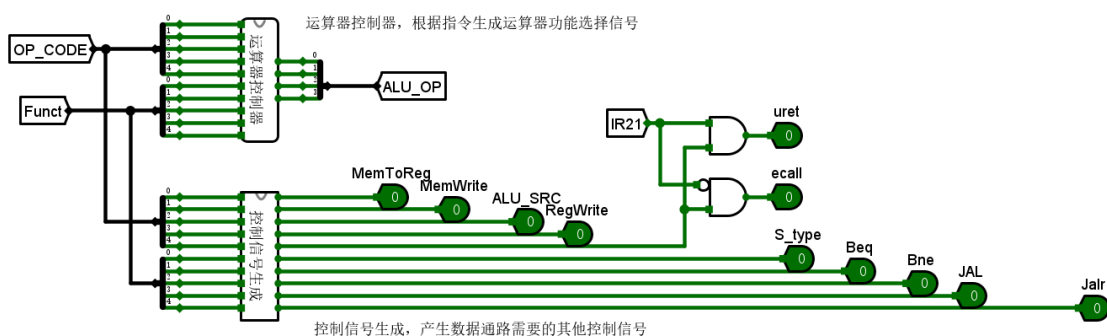


图 3.5 主控制器的实现

2) Branch 控制器

Branch 控制器主要负责根据条件分支和无条件分支决定下一条指令的地址，主要使用多路选择器和优先编码器实现，具体实现如图 3.6 所示。

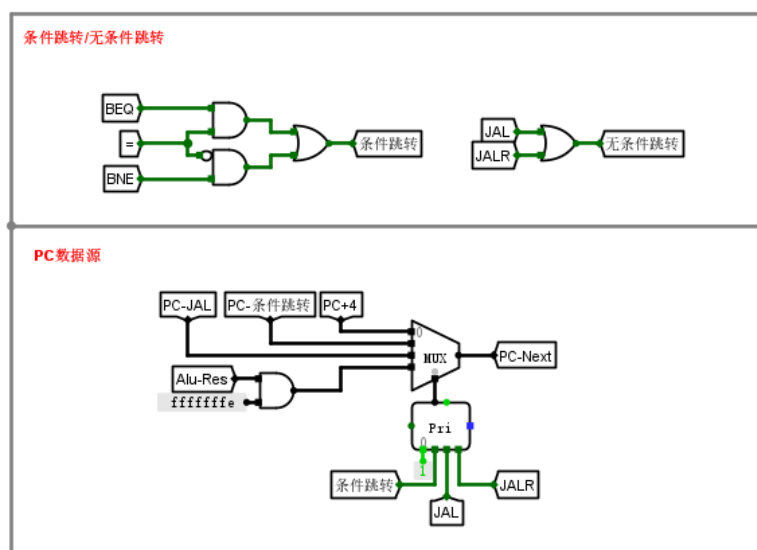


图 3.6 Branch 控制器实现

3) ECALL 控制器

ECALL 控制器主要负责当出现 ECALL 指令时，根据寄存器 \$a7 是否等于 34 决定是否停机，具体实现如图 3.7 所示。

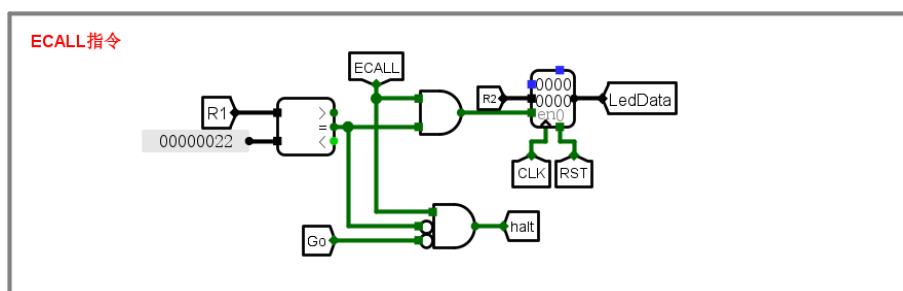


图 3.7 ECALL 控制器实现

3.2 中断机制实现

3.2.1 单级中断

1) 中断按键信号采样电路

使用提供的中断按键信号采样电路存储中断请求，并使用优先编码器输出优先级最高的中断号以及是否有中断请求，具体实现如图 3.8 所示。

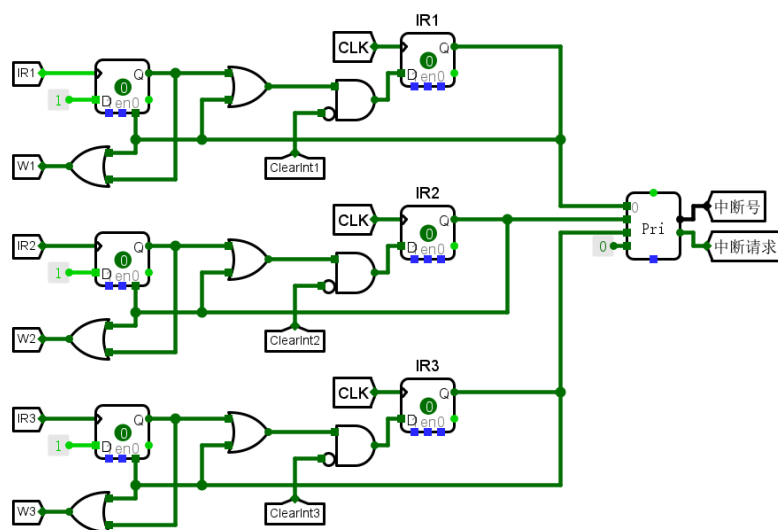


图 3.8 单级中断按键信号采样电路

2) 中断相关寄存器/硬件支持

为了实现单级中断，需要实现中断程序返回寄存器 **mEPC**、中断使能寄存器 **IE**、中断响应信号 **INT**、中断服务程序入口表。具体实现如下：当发生中断请求且中断使能时将 **PC** 送入 **mEPC** 寄存；当发生中断请求时置 **IE** 为 1，当执行 **URET** 指令时置 **IE** 为 0；当发生中断请求且中断使能时置中断响应信号 **INT** 为 1；使用多路选择器输出中断服务程序入口。具体实现如图 3.9 所示。

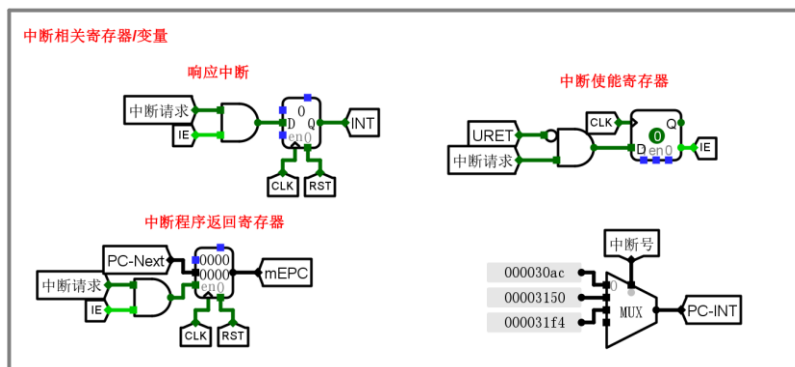


图 3.9 单级中断相关硬件支持

3) 中断相关软件支持

为了实现单级中断，需要实现中断返回指令的支持。当执行 **URET** 指令时，将 **mEPC** 寄存器保存的断点地址送入 **PC** 寄存器，具体实现如图 3.10 所示。

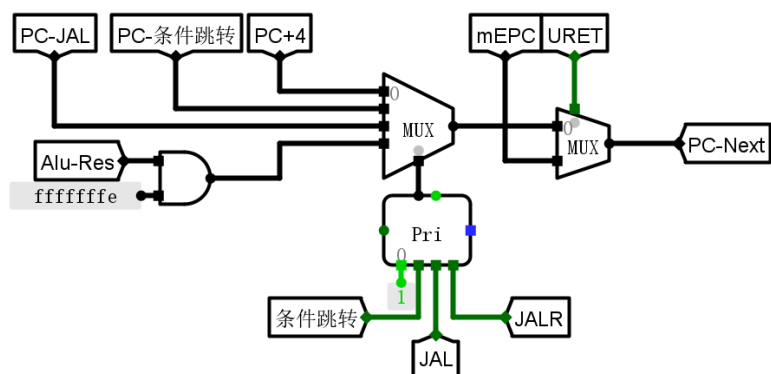


图 3.10 单级中断相关软件支持

4) 单级中断数据通路

根据中断信号采样电路、中断相关寄存器、URET 指令支持，即可在单周期 CPU 的基础上实现单级中断，具体实现如图 3.11 所示。

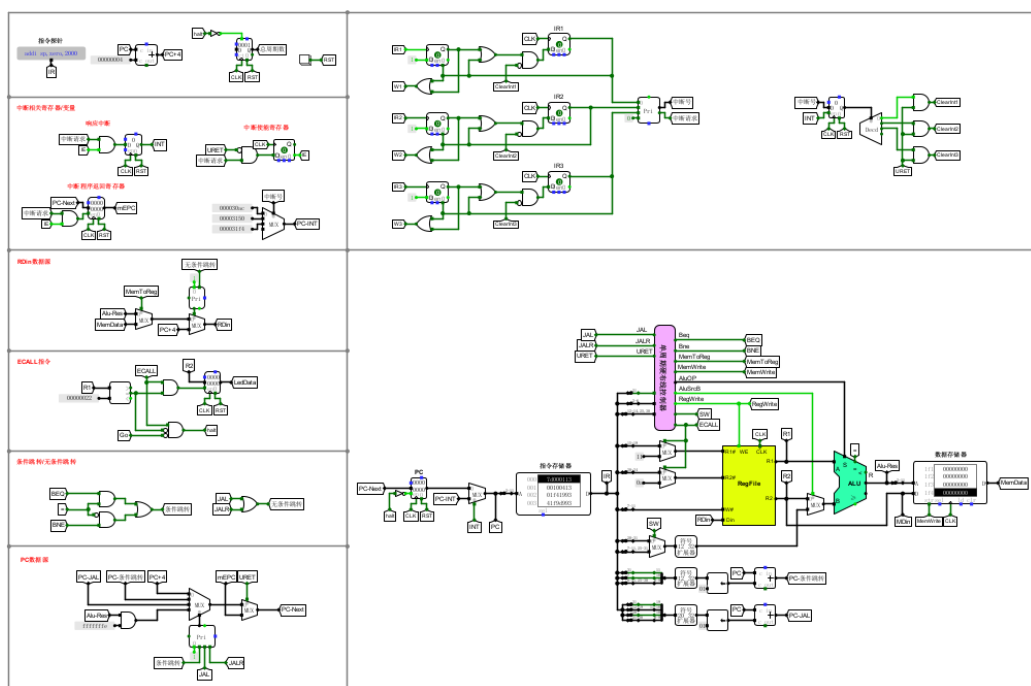


图 3.11 单级中断数据通路

3.2.2 多级中断

1) 中断按键信号采样电路

多级中断与单级中断相比，增加了寄存当前中断号的电路，如图 3.12 所示。

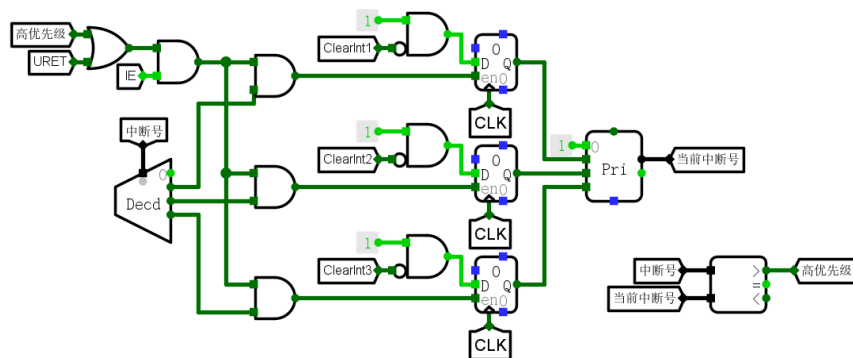


图 3.12 多级中断当前中断号寄存电路

2) 中断相关寄存器/硬件支持

为了实现多级中断，需要设置多个终端程序返回寄存器 mEPC。同时，使用 CSRRSI、CSRRSI 指令模拟开关中断，因此需要更改中断使能寄存器 IE 的相关逻辑。其余硬件支持与单级中断大致相同。具体实现如图 3.13 所示。

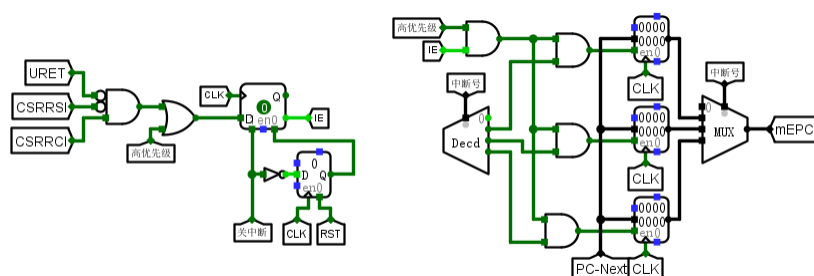


图 3.13 多级中断相关硬件支持

3) 多级中断数据通路

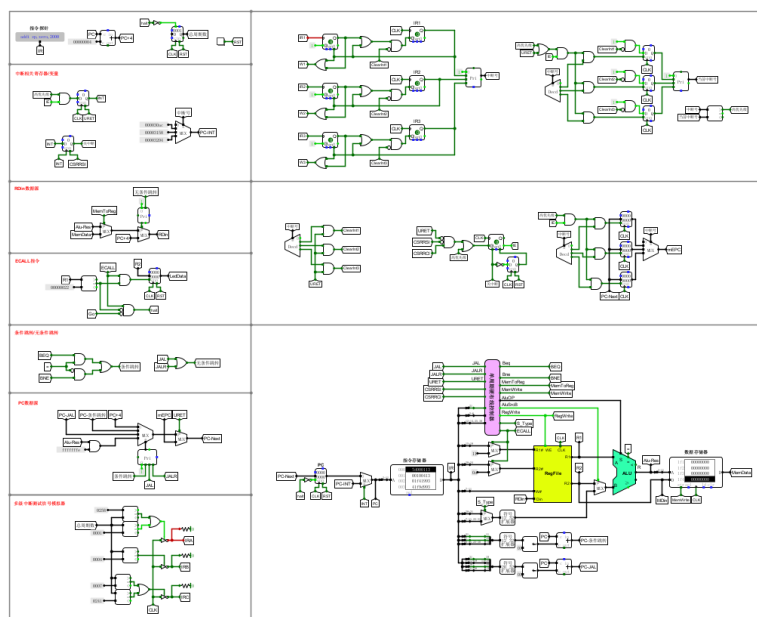


图 3.14 多级中断数据通路

3.3 理想流水线 CPU 实现

3.3.1 流水接口部件实现

流水接口部件内部采用寄存器进行锁存，寄存器均采用上升沿触发，同时需要实现同步清零端、使能端。以 ID/EX 段流水接口部件为例，具体实现如图 3.15 所示。

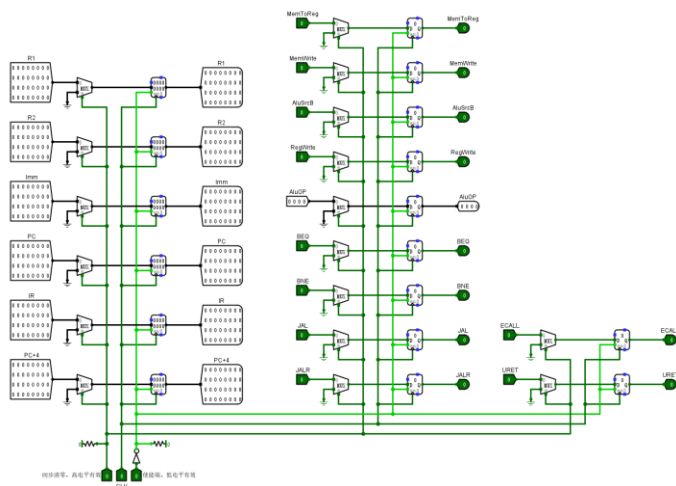


图 3.15 流水接口部件实现

封装流水接口部件时，将输入端布置在左侧，输出端布置在右侧，使能端、清零端、时钟端布置在上下侧。以 ID/EX 段流水接口部件为例，具体封装如图 3.16 所示。

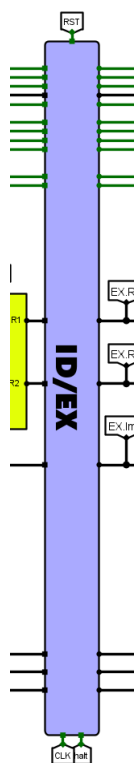


图 3.16 流水接口部件封装

3.3.2 理想流水线实现

理想流水线不涉及对分支冲突和数据冲突的处理,因此只需完成各控制信号的传递.在布线时,需要实时调整流水接口部件的封装以保证布线的美观性,同时需要考虑不同段数据的正确性,如寄存器写入时的相关信号与数据。具体实现如图 3.17 所示。

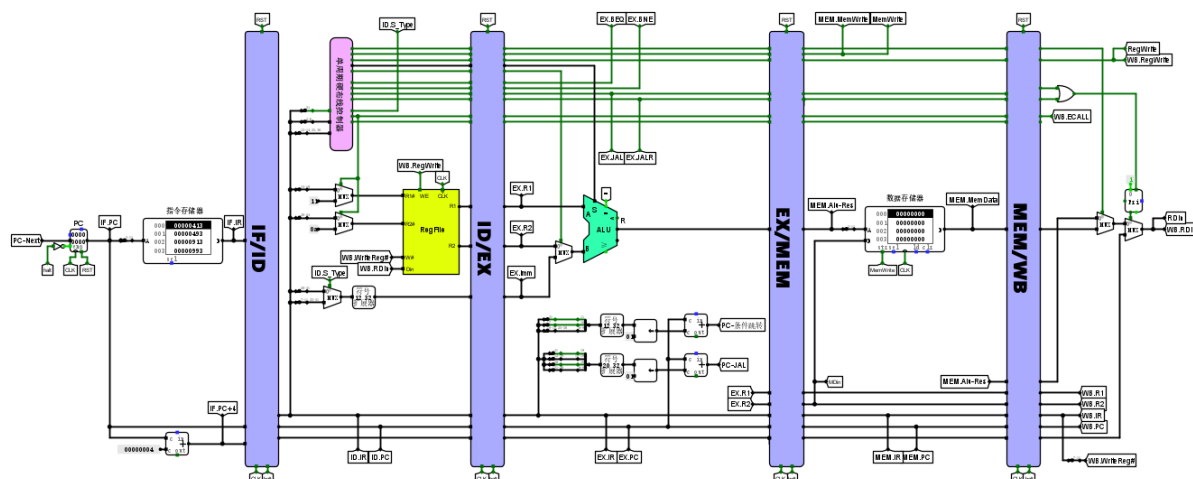
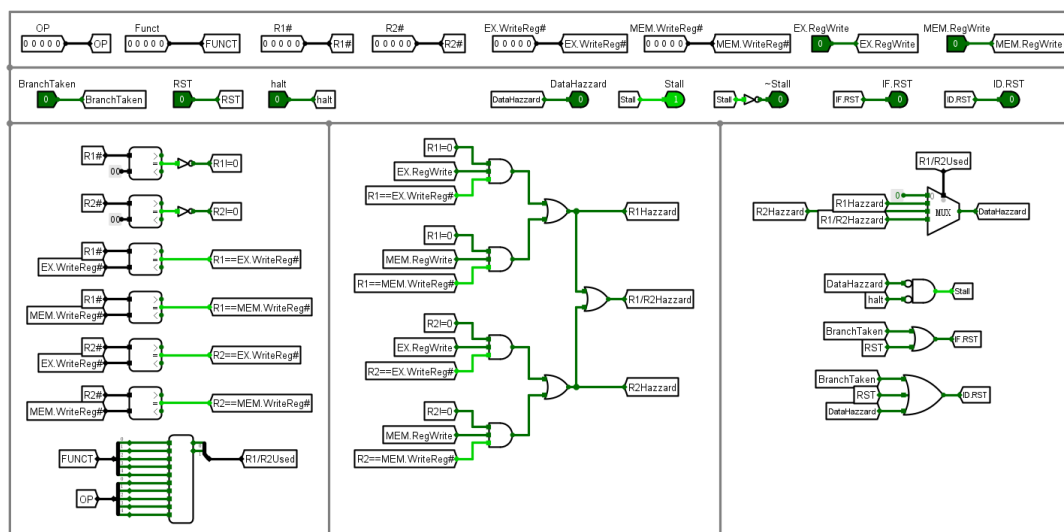


图 3.17 理想流水线实现

3.4 气泡流水线 CPU 实现

3.4.1 气泡逻辑实现

在实现插入气泡的逻辑时,需要分别实现数据相关检测逻辑、产生为了完成插入气泡操作需要送入相关部件的信号,如 IF.RST、ID.RST、Stall、~Stall 分别送入 IF/ID 流水寄存器的清零端、ID/EX 流水寄存器的清零端、PC 寄存器的使能端、IF/ID 流水寄存器的使能端。具体实现如图 3.18 所示。



华中科技大学课程设计报告

图 3.18 气泡逻辑实现

将气泡逻辑的相关实现封装为气泡逻辑部件，如图 3.19 所示。



图 3.19 气泡逻辑封装

3.4.2 气泡流水线实现

将气泡逻辑部件添加到原有的理想流水线中，并将各输入输出连接正确后，即可得到气泡流水线的具体实现，如图 3.20 所示。

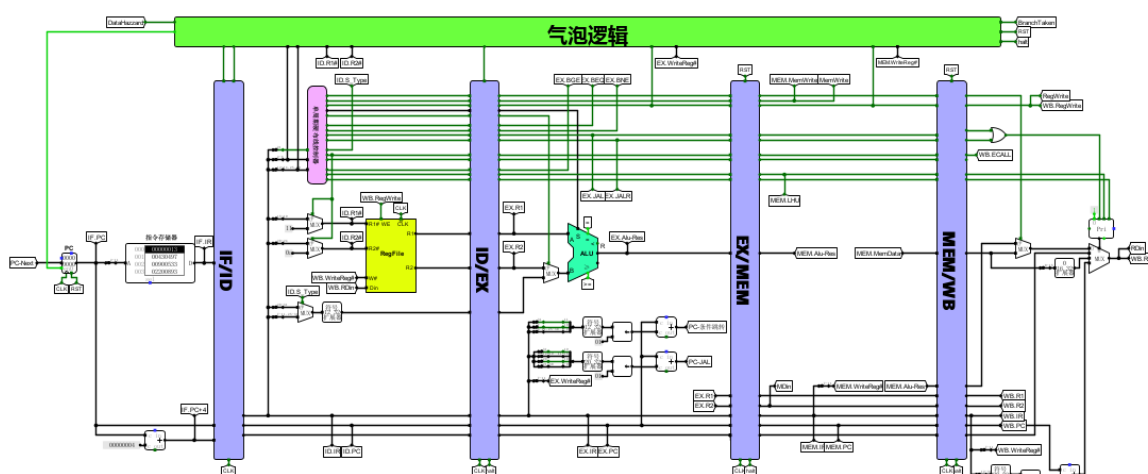


图 3.20 气泡流水线实现

3.5 重定向流水线 CPU 实现

3.5.1 重定向逻辑实现

在实现重定向逻辑时，只需要在气泡逻辑的基础上添加 LoadUse 相关的检测、Fwd1 和 Fwd2 信号的生成，并更改相关信号的逻辑。具体实现如图 3.21 所示。

华中科技大学课程设计报告

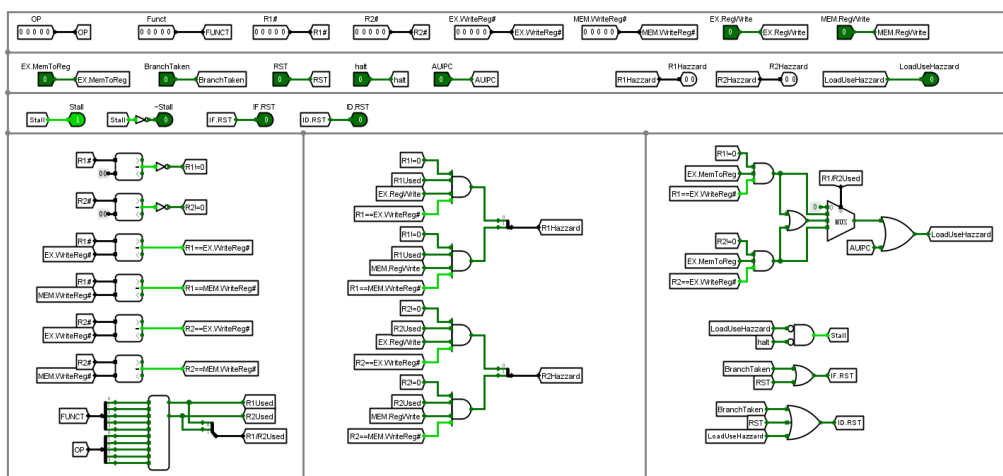


图 3.21 重定向逻辑实现

将气泡逻辑的相关实现封装为气泡逻辑部件，如图 3.22 所示。

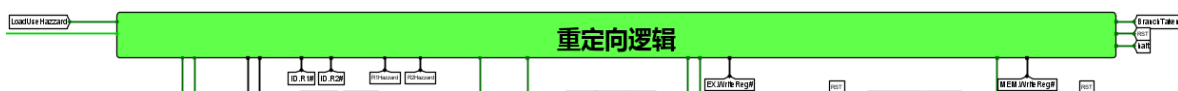


图 3.22 重定向逻辑封装

3.5.2 数据重定向实现

在 EX 段,使用 Fwd1 和 Fwd2 选择信号将正确的数据重定向到运算器的输入端,具体实现如图 3.23 所示。

3.5.3 重定向流水线实现

使用重定向逻辑部件替换气泡逻辑部件,并将各输入输出连接正确,添加数据重定向逻辑后,即可得到重定向流水线的具体实现,如图 3.24 所示。

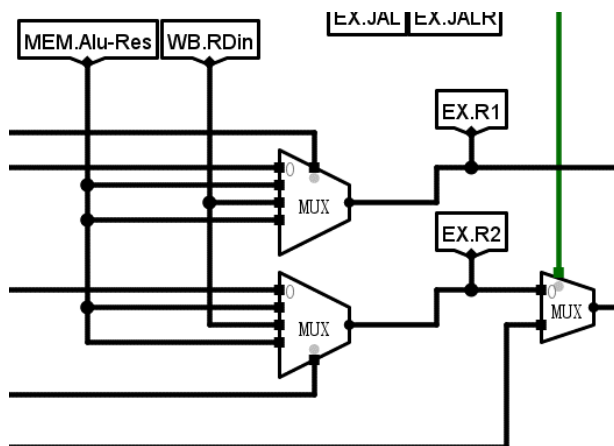


图 3.23 数据重定向实现

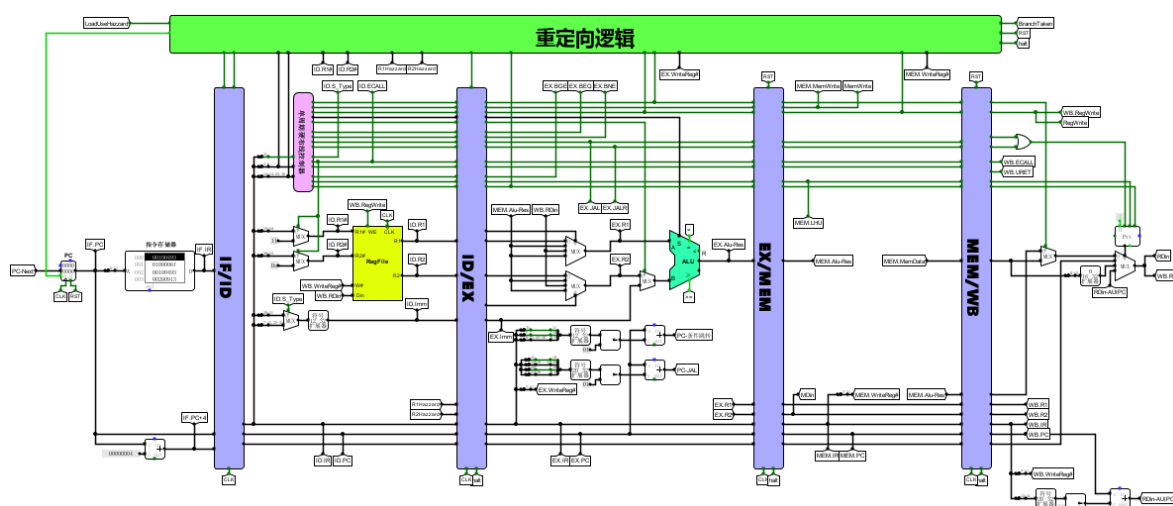


图 3.24 重定向流水线实现

3.6 动态分支预测机制实现

3.6.1 双位预测状态机实现

根据双位预测状态转换图(图 2.2),使用 Logisim 软件的分析组合逻辑电路功能,得到双位预测状态机如图 3.25 所示。

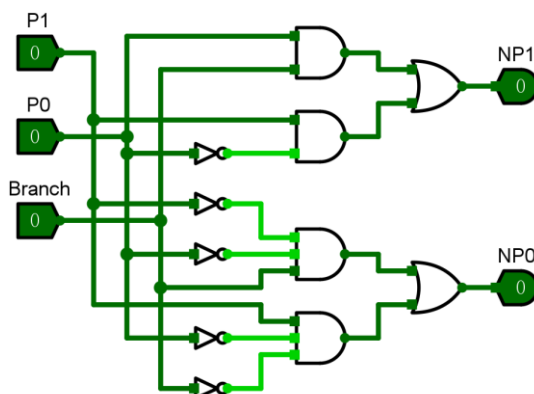


图 3.25 双位预测状态机

3.6.2 BTB 表实现

BTB 表的存储设计与全相联 Cache 基本一致,有关八路并发比较、LRU 淘汰算法等实现不再赘述,这里着重说明分支历史预测位的相关逻辑。

在存储分支历史预测位时,寄存器使用双位预测状态机输出的次态作为输入,并将现态作为状态机的输入。由于分支预测初始值也会影响预测结果,这里选择初始值为 11,将分支历史预测位的高位作为预测结果输出。总体实现如图 3.26 所示。

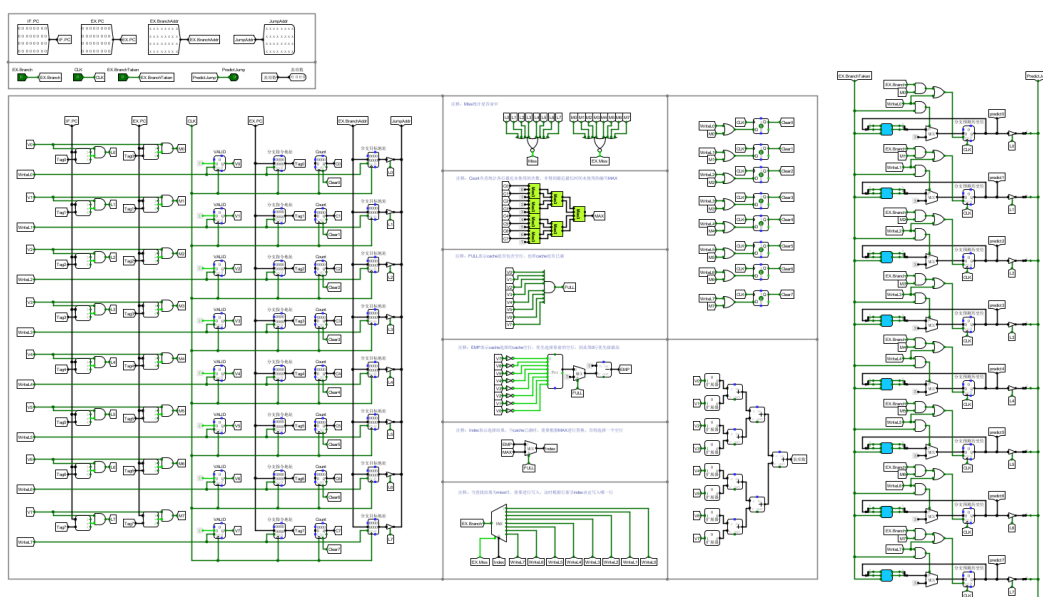


图 3.26 BTB 表总体实现

3.6.3 基于重定向流水线的动态分支预测机制实现

在重定向流水线的基础上，添加图 3.27 结构，使用 PredictError 代替 BranchTaken 送入重定向逻辑，即当分支预测失败时插入气泡。

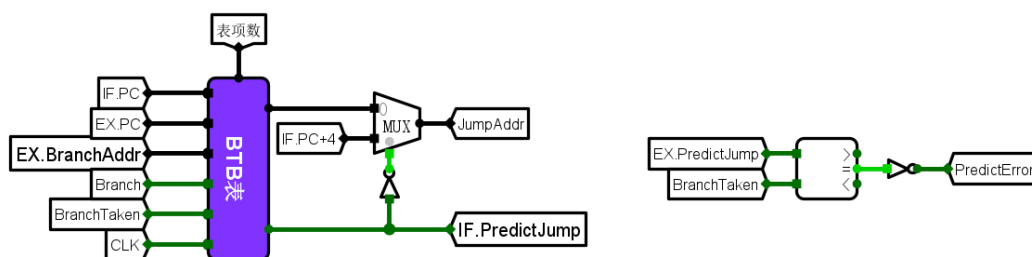


图 3.27 BTB 表的使用

在添加在流水接口上增加 PredictJump 接口、修改总周期数统计逻辑等细节后，可以得到动态分支预测的具体实现。

4 实验过程与调试

4.1 测试用例和功能测试

4.1.1 CCAB 指令测试

1) AUIPC 指令

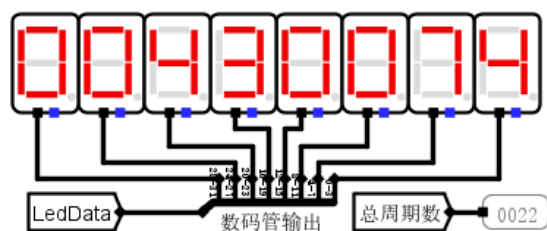


图 4.1 AUIPC 指令测试结果

2) SLTIU 指令

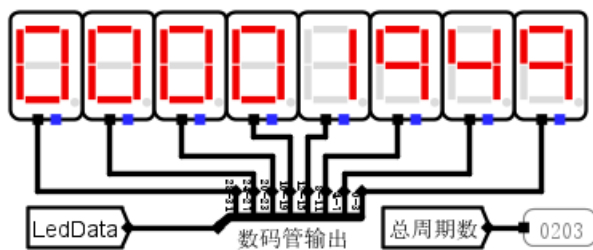


图 4.2 SLTIU 指令测试结果

3) LHU 指令

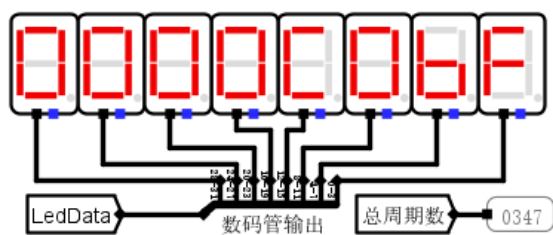


图 4.3 LHU 指令测试结果

4) BGE 指令

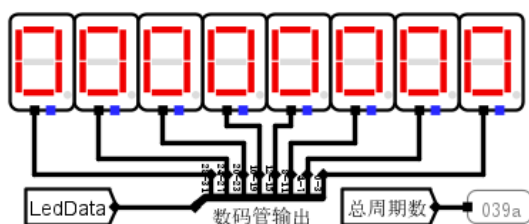


图 4.4 BGE 指令测试结果

华中科技大学课程设计报告

4.1.2 流水线测试

1) 气泡流水线

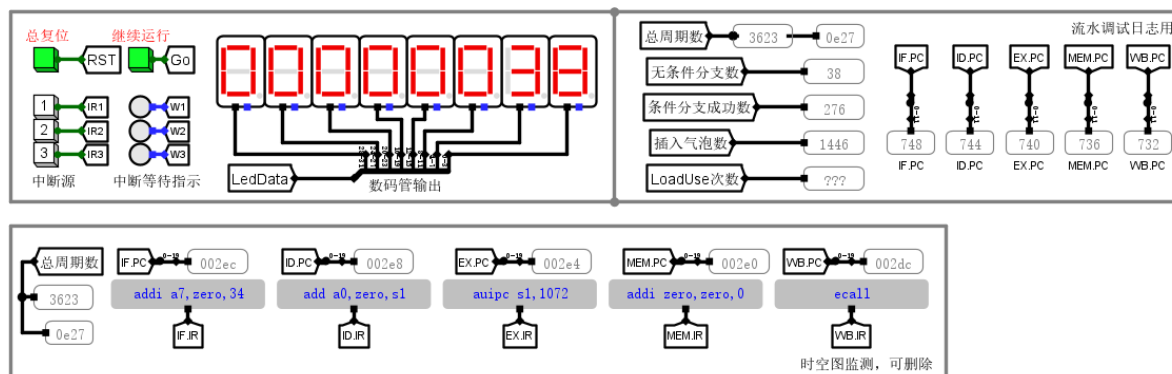


图 4.5 气泡流水线测试结果

2) 重定向流水线

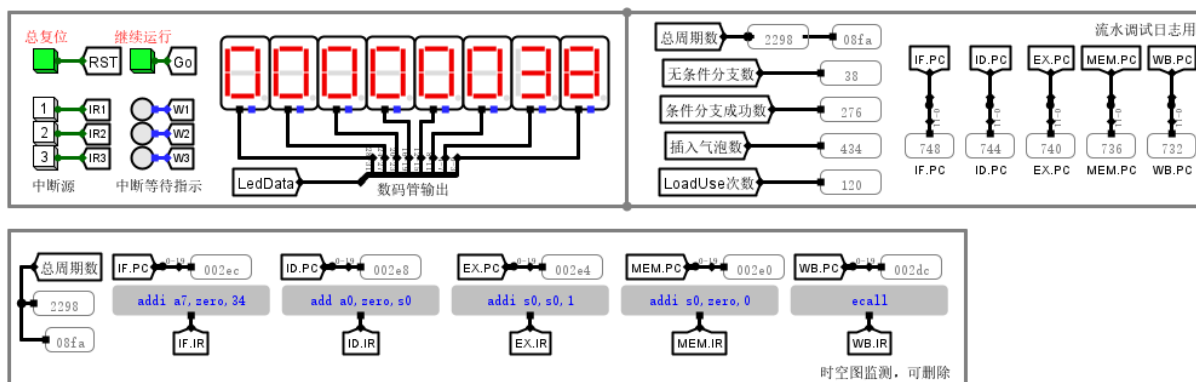


图 4.6 重定向流水线测试结果

3) 动态分支预测

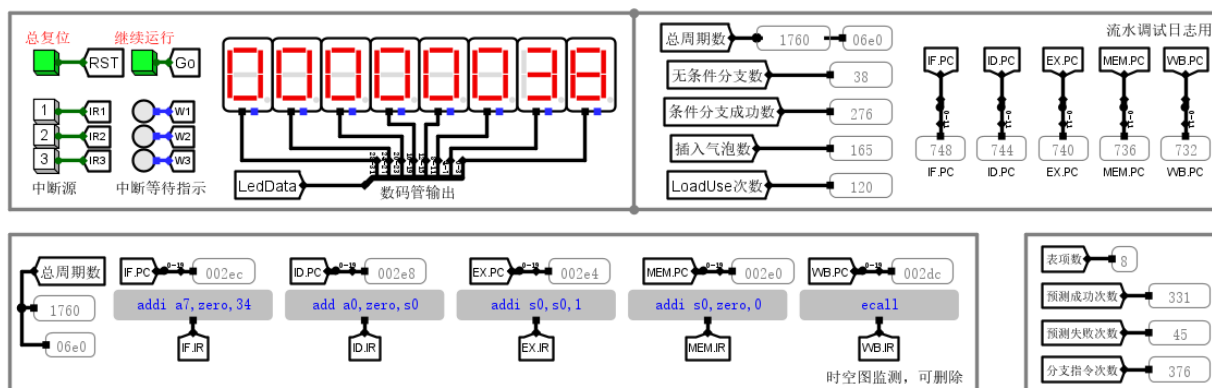


图 4.7 动态分支预测测试结果

4.2 性能分析

表 4.1 不同方案时钟周期数对比

方案	时钟周期数
单周期 CPU	1545
气泡流水线	3624
重定向流水线	2297
动态分支预测	1759

单周期硬布线的时钟周期最短，但是每条指令的平均执行时间最长，效率最差；

气泡流水线时钟周期最多，主要是由于处理数据冲突和分支冲突时均采用插入气泡的方式，从而导致插入了大量气泡；

重定向流水线采用重定向方式处理数据冲突，减少了插入气泡数；

动态分支预测则在重定向的基础上优化了分支冲突的处理方式，效率最高。

4.3 主要故障与调试

4.3.1 多级中断中中断返回后无法响应其他中断请求

多级中断中，当某个中断服务程序执行完毕返回时，无法继续响应其他中断请求。

故障现象：当执行中断号为 3 的中断服务程序的过程中，发生了中断号为 1 的中断请求，由于中断优先级：3>1，因此继续执行当前中断服务程序。当当前中断服务程序返回时，却无法响应中断号为 1 的中断请求。

原因分析：在多级中断中，新中断发生时未能将中断号正确保存在相应寄存器中，导致当前中断执行结束时无法继续响应新中断。

解决方案：使用三个寄存器将每次的中断请求保存下来，利用 IE 寄存器的状态和是否发生新中断信号作为使能端，更新历史记录，根据最近历史记录和当前中断号进行判断决定应该进行的中断号。具体实现如图 4.8 所示。

4.3.2 动态分支预测优化失败

动态分支预测优化失败，减少的气泡数未达到预期。

故障现象：正确完成动态分支预测时，理想的气泡数为 1800 左右，而当我完成动态分支预测时，经过屡次调整，插入气泡数也未达到理想水平。

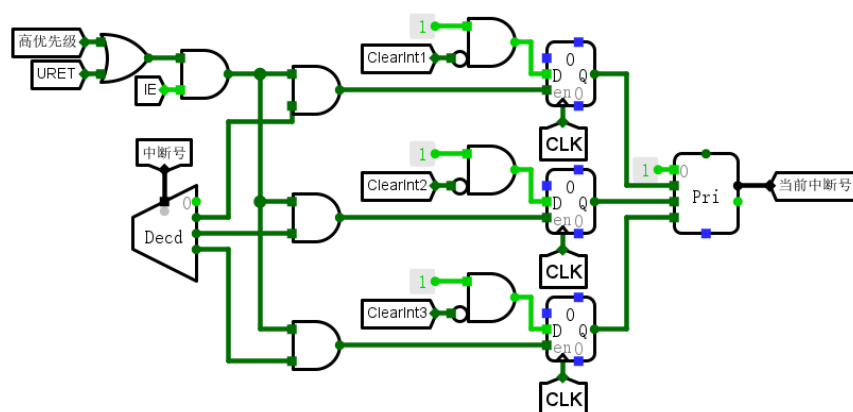


图 4.8 多级中断中断请求信号寄存器图

原因分析：由于初期布线设计 BTB 表时，写入 BTB 表的逻辑有误，分支预测历史位的更新逻辑有误，将 BTB 表预测结果与重定向流水线整合时有误，导致屡次修改调整，减少的气泡数未达到预期。

解决方案：

对于写入 BTB 表，应当在 EX 段中，EX.PC 未命中且 EX.Branch 为 1 时写入；

对于分支预测历史位的更新逻辑，应当在 EX 段进行更新，因此需要使用 EX.Branch 信号及 EX.PC 的命中情况作为寄存器的使能端；

在与重定向流水线整合时，需要注意当预测失败时，送入 PC 寄存器的内容应为 EX.PC+4，而非 IF.PC+4。

在经过以上调整，并且置分支预测历史位初始值为 11 后，最终气泡数达到预期。

4.3.3 数据存储器按半字访问错误

数据存储器按半字访问得到的数据错误。

故障现象：在测试 CCAB 指令中的 LHU 指令时，理想的 LED 输出为 0x00008281 0x00008483 0x00008685 0x00008887 ……序列，而实际输出为 0x00008281 0x00008685 ……序列，产生数据丢失。

原因分析：在布线时，未考虑到按半字或按字节访问数据存储器情况，而仅仅使用了一个数据存储器，默认为按字访问。

解决方案：利用之前在计算机组成原理实验中完成的可以按字、半字、字节访问的存储器兼容 LHU 指令，具体实现可见图 3.3。

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和 RISC-V 指令手册。
第二天	完成数据通路，实现单周期 CPU 并且通过测试。
第三天	阅读指令流水线内容，开始尝试搭建理想流水线 CPU。
第四天	重写流水接口部件，调试理想流水线 CPU 并通过测试。
第五天	阅读气泡流水线对于冲突的处理方法，尝试气泡流水线 CPU。
第六天	重写气泡逻辑部件，完成气泡流水线并通过测试。
第七天	构建重定向逻辑，完成重定向流水线，通过测试。
第八天	完成单级中断并通过测试，尝试多级中断。
第九天	调试多级中断的 bug，通过测试。开始构建 BTB 表。
第十天	解决分支预测中的诸多 bug，完成分支预测功能。

5 团队任务

5.1 选题介绍

本团队以 Risc-V CPU 为基础，结合 LCD Video 组件，实现简易的数独游戏，其中包含数独游戏的显示、填写功能。在查看往届团队的选题时，发现选题多有重复，而我们又曾在程序综合课程设计中设计过数独游戏，因而确定了数独游戏的选题。

5.2 任务分工

我主要负责电路逻辑，刘鉴之主要负责代码编写。具体分工如表 5.1 所示。

表 5.1 团队任务分工

徐瑞达	刘鉴之
<ul style="list-style-type: none">● 完成主要电路部分● 录制 Logisim 演示视频● 完成 PPT 制作	<ul style="list-style-type: none">● 编写数独 C 语言代码并汇编为 Risc-V 代码● 修改 Risc-V CPU 以支持上述代码所需要的指令● 编写代码生成数字字库文件、数据存储器文件● 录制成果汇报视频

5.3 问题与解决方案

5.3.1 如何产生数独

问题说明：当使用程序综合课程设计中的 Las Vegas 算法+挖洞法生成数独时，由于存在大量递归过程，转换为 Risc-V 代码运行后会发生堆栈溢出，不得不弃用。

解决方案：最终不得不放弃挖洞法生成数独，转而采用生成随机数选取已有数独的方案。

如代码块 5.1 所示，rand 函数根据 seed 函数产生的随机数种子产生随机数，确保每次产生随机数的随机性。

在 Risc-V 中，可以使用 rdtm 指令取 time 寄存器作为随机数种子，然而 rdtm 指令将被汇编为 csrrs 指令，需要访问 CSR 寄存器，因此只能采用硬件支持的方法。

在 Logisim 平台实现 CPU 时，当执行至 csrrs 指令时，将 Logisim 中的随机数寄存器产生的随机数送入 a0 寄存器作为随机数种子。

华中科技大学课程设计报告

代码块 5.1 产生数独的 Risc-V 代码

```
1.  seed:
2.      addi    sp,sp,-16
3.      sw      s0,12(sp)
4.      addi    s0,sp,16
5.      nop
6.      rdttime a0
7.      lw      s0,12(sp)
8.      addi    sp,sp,16
9.      jr      ra
10.
11.  rand:
12.      addi    sp,sp,-32
13.      sw      ra,28(sp)
14.      sw      s0,24(sp)
15.      addi    s0,sp,32
16.      call    seed
17.      sw      a0,-20(s0)
18.      lw      a4,-20(s0)
19.      li      a5,1103515648
20.      addi    a5,a5,-403
21.      mul     a4,a4,a5
22.      li      a5,12288
23.      addi    a5,a5,57
24.      add     a5,a4,a5
25.      sw      a5,-20(s0)
26.      lw      a5,-20(s0)
27.      srai    a3,a5,31
28.      li      a4,65536
29.      addi    a4,a4,-1
30.      and     a4,a3,a4
31.      add     a5,a4,a5
32.      srai    a5,a5,16
33.      mv      a4,a5
34.      li      a5,32768
35.      addi    a5,a5,-1
36.      and     a5,a4,a5
37.      mv      a0,a5
38.      lw      ra,28(sp)
39.      lw      s0,24(sp)
40.      addi    sp,sp,32
41.      jr      ra
```

5.3.2 如何绘制数字

问题说明：使用 LED 数码管显示每个宫格内的数字时体验性差，且不具有挑战性；如果使用 LCD Video 显示每位数字时，如何送入数据？

解决方案：如图 5.1 所示，使用 9 个 LCD Video 组件分别绘制每个宫格内的数字，每个 LCD Video 绘制 9 个格子 and 分隔线；初盘中的数字将被绘制为白色，而用户输入的数字将被绘制为蓝色。通过在 ROM 中分别存储每个数字和分隔线的点阵信息，使用到时依次输出至对应 LCD Video 控件即可。

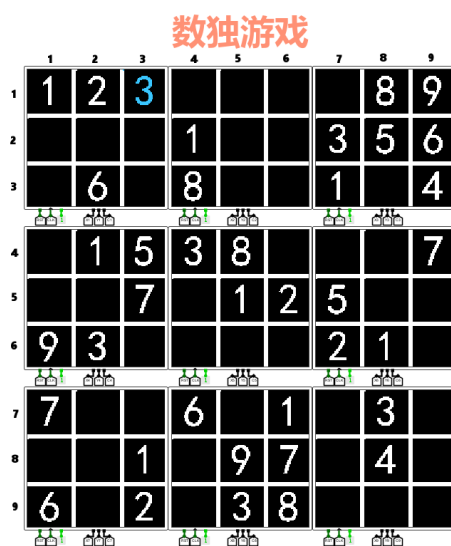


图 5.1 绘制数独的解决方案

5.3.3 如何向 LCD Video 送入数据

问题说明：如何配合时钟向 LCD Video 送入数据？如何将数独初盘和用户输入的数字绘制为不同颜色？如何在用户修改数字时刷新显示？

解决方案：经过团队商讨后，决定将“数字点阵库”封装为单个部件，并封装“LCD 数据源”部件。在 LCD 数据源部件内，通过逐个输出 9 个格子内的每个格子对应的点阵序列，达成在 LCD Video 内逐个绘制格子的目的。同时，由是否选中的信号决定“数字点阵库”输出的颜色常量值。在用户修改数字时，通过使用覆盖整个格子的点阵库并送入黑色以达到清空的目的，然后送入用户填入的数字以刷新显示。

5.3.4 如何判别坐标

问题说明：如何在游戏初始化时，聚焦于首个空格？如何根据当前坐标，处理上

华中科技大学课程设计报告

下左右移动事件，聚焦于合适的空格？

解决方案：图 5.2 是部分坐标判别逻辑的电路实现图。以 NextLeft 为例，在初盘寄存器电路中，将当前 X 坐标与该行中所有空格的坐标进行比较，并送入优先编码器中生成符合要求空格的坐标；在其余电路中，还实现了当前坐标的左侧无其他空格时，跳转到上一行末空的功能。依照 NextLeft 信号的生成逻辑，可以得到 NextRight、NextUp、NextDown 以及首个空格坐标信号的生成逻辑。

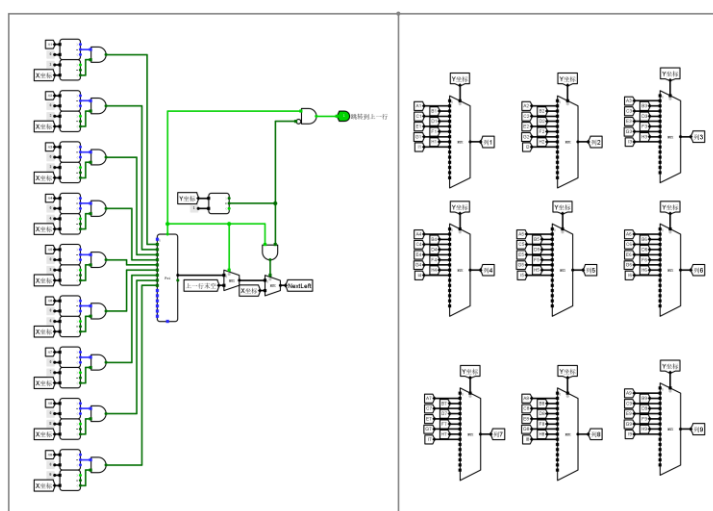


图 5.2 部分坐标判别逻辑实现

5.4 版本提交历史

表 5.2 版本提交历史

提交时间	提交者	提交内容
2022-09-17 00:07	徐瑞达	First Commit
2022-09-17 15:49	刘鉴之	完成选择数独代码部分
2022-09-17 18:06	徐瑞达	完成数独初盘、终盘寄存器
2022-09-17 21:22	刘鉴之	完成对 Risc-V CPU 的修改
2022-09-17 22:00	徐瑞达	完成 LCD 数据源体系构建
2022-09-18 12:13	刘鉴之	完成字库文件生成代码部分
2022-09-19 00:03	徐瑞达	初步达成显示数独功能，构建当前格局寄存器
2022-09-19 23:39	徐瑞达	完成基本功能，可填写及判断是否填写正确
2022-09-20 12:22	徐瑞达	完成更加方便的坐标选择逻辑
2022-09-20 16:30	团队	联调测试，修改若干小 bug，完成项目

6 设计总结与心得

6.1 课设总结

本次课程设计主要作了如下几点工作：

- 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、重定向流水线 CPU、支持单级中断的单周期 CPU、支持多级中断的单周期 CPU、支持动态分支预测机制的重定向流水线 CPU。
- 2) 功能总结：单周期 CPU、气泡流水线 CPU、重定向流水线 CPU、支持动态分支预测机制的重定向流水线 CPU 都实现了包括 CCAB 指令的指令集；气泡流水线通过插入气泡的方式实现了处理数据冲突和分支冲突；重定向流水线通过重定向数据的方式减少了插入气泡数；支持动态分支预测机制的重定向流水线通过预测分支跳转的方式进一步减少了插入气泡数；支持多级中断的单周期 CPU 实现了三级嵌套外部中断。
- 3) 团队任务总结：实现了使用随机数随机选择数独游戏的功能，实现了在 LCD Video 中绘制数独游戏的功能，实现了通过键盘输入选择空格子的功能，实现了动态绘制用户输入数字的功能，实现了数独填写正确时响起提示音的功能。

6.2 课设心得

本次课程设计是对上学期组成原理实验的拔高，从个人任务的完成到团队任务的选题、设计与完成，总共经历了大约半个月的时间。

课程设计刚开始时，由于对整个实验及其难度没有一个大致的把握，有种畏怯心理，导致看了一天的任务书和相关文档，也没有头绪。在对照着数据通路进行布线连接后，才开始慢慢进入状态，第二天便完成了单周期 CPU 的设计。

接着设计理想流水线的过程也不是很顺利，在使用了上学期组成原理实验的乘法流水线接口部件并且仔细阅读课本掌握流水线的原理后，才在第四天完成理想流水线的设计。然后是气泡流水线的设计，由于先前对数据冲突和分支冲突的概念了解不透彻，只能迷迷糊糊地按照课本上讲述的逻辑设计。在设计重定向流水线时，我方才认识到，理想流水线没有处理冲突，气泡流水线采用插入空气泡的方式处理冲突，而重

华中科技大学课程设计报告

定向流水线则是在气泡流水线的基础上改进了数据冲突的处理方式。对于最后具有一定难度的动态分支预测机制，则是在重定向流水线的基础上改进了分支冲突的处理方式。从理想流水线到气泡流水线，从重定向流水线再到支持动态分支预测机制的流水线，逐步提高 CPU 的运行效率，在逐步提高难度的过程中也获得了更强烈的自豪感。

在团队任务中，我的体悟也更为深刻。从确定选题后的毫无头绪到数独格局的成功读取，从 LCD 成功显示第一个数字到 9 个 LCD 组成的阵列成功显示整个数独，从成功完成键盘事件的处理到最终数独游戏的顺利完成，我都无法想象那些复杂部件的设计是我们在几天内完成的。虽然受限于 LCD 的绘制速度，数独游戏的绘制需要漫长的等待，但是当我想到我可以自己画出的电路玩一盘数独游戏时，心情是无比激动的，澎湃的自豪感淹没了这半个月来遇到的所有崎岖坎坷。

最后十分感谢课程组老师们的帮助，也感谢团队伙伴刘鉴之同学的支持与帮助，我相信这是我大学生涯中最有成就感的事情之一。

参考文献

- [1] DAVID A. PATTERSON(美). 计算机组成与设计硬件/软件接口(原书第 4 版). 北京: 机械工业出版社.
- [2] David Money Harris(美). 数字设计和计算机体系结构(第二版). 机械工业出版社
- [3] 谭志虎, 秦磊华, 吴非, 肖亮. 计算机组成原理. 北京: 人民邮电出版社, 2021 年.
- [4] 谭志虎, 秦磊华, 胡迪青. 计算机组成原理实践教程. 北京: 清华大学出版社, 2018.
- [5] 袁春风编著. 计算机组成与系统结构. 北京: 清华大学出版社, 2011 年.
- [6] 张晨曦, 王志英. 计算机系统结构. 高等教育出版社, 2008 年.

• 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：徐瑞达 徐瑞达