

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：CS1601

学号：U201614531

姓名：刘本嵩

电话：+1 (669) 266-7699

邮件：root@recolic.org

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 流水 CPU 设计	12
2.3 气泡式流水线设计	12
2.4 重定向流水线设计	13
3 详细设计与实现	14
3.1 单周期 CPU 实现	14
3.2 流水 CPU 实现	22
3.3 气泡式流水线实现	24
3.4 重定向流水线实现	25
4 实验过程与调试	28
4.1 测试用例和功能测试	28
4.2 EDUCODER 测试结果	30
4.3 性能分析	31
4.4 主要故障与调试	32
4.5 实验进度	33
5 设计总结与心得	35
5.1 课设总结	35

华 中 科 技 大 学 课 程 设 计 报 告

5.2 课设心得.....	35
参考文献.....	37

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	SUB	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	XOR	异或	
29	SLTIU	无符号比较	
30	LB	加载 Byte	
31	BGEZ	大于等于零跳转	

2 总体方案设计

2.1 单周期 CPU 设计

在此次单周期 CPU 的设计中，我们采用的方案是硬布线控制。同时，为了避免结构冲突，我们采用哈佛结构将指令存储器与数据存储器分离分别用单独的存储器表示。这个 cpu 支持如表 1.1 所示的 24 条基本指令以及 4 条拓展的指令，分别是 SLL,SRA,SRL,ADD,ADDU,SUB,AND,OR,NOR,SLT,SLTU,JR,SYSCALL,J,JAL,BEQ,BNE,ADDI,ANDI,ADDIU,SLTI,ORI,LW,SW,XOR,SLTIU,LB,BGEZ. 设计过程中主要采用了 logisim 来完成逻辑电路的设计，然后计划开学后在 FPGA 平台进行该单周期 28 条指令 CPU 的实际上板验证。

单周期硬布线 CPU 的总体结构图如图 2.1 所示。

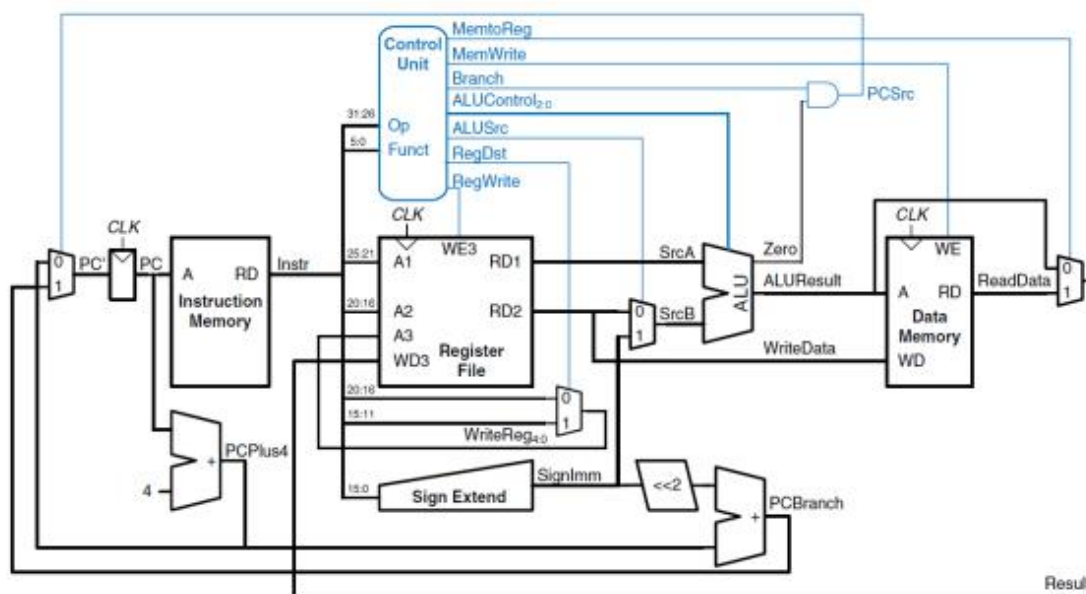


图 2.1 总体结构图

2.1.1 主要功能部件

该单周期硬布线 CPU 主要包括的部件有：程序计数器 PC，指令存储器 IM，运算器 ALU，寄存器组 RE，数据存储器 DM。其中单个部件的设计如下所述：

华中科技大学课程设计报告

1. 程序计数器 PC

程序计数器 PC 是一个 32 位的寄存器,存储将要执行的指令的地址,同时也会被跳转和分支指令修改。CPU 在每一个周期中,将 PC 寄存器中的值作为访问指令存储器 IM 的 address。

2. 指令存储器 IM

指令存储器 IM 用于存储程序的所有指令。在该 CPU 的设计中,我们使用老师提供的 Logisim 库中的硬件 RAM 来完成。它根据输入的 PC 作为 address,将对应地址的指令字取出,交给下一步 decoder。

3. 运算器

运算器 ALU 与 IM 一样,在前面的实验已经完成,因此可以直接使用。算术逻辑运算单元引脚与功能描述,以及操作码与功能的对应关系如表 2.1 和 2.1.2 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
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

华中科技大学课程设计报告

表 2.1.2 操作码与运算功能的对应关系表

ALU_OP	十进制	运算功能
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
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. 寄存器堆 RF

寄存器组 RF 在 cs3410 库中也已经提供，直接使用即可。这是一个上边缘触发的寄存器。在后面实验中，少数情况下调整了寄存器的实现用来避免竞争，将会在必要时提到。

5. 数据存储器 DM

数据存储器 DM 用于存储程序的所有数据，构成了计算机的所有主存空间。实

华中科技大学课程设计报告

现与 IM 相似。

2.1.2 数据通路的设计

根据 MIPS 文档，依次分析 28 条指令，得到如下表 2.2 所示的指令系统数据通路框架表：

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		R2	0		
SRA	PC+4	PC	rs	rt	rd	ALU		R2	1		
SRL	PC+4	PC	rs	rt	rd	ALU		R2	2		
ADD	PC+4	PC	rs	rt	rd	ALU	R1	R2	5		
ADDU	PC+4	PC	rs	rt	rd	ALU	R1	R2	5		
SUB	PC+4	PC	rs	rt	rd	ALU	R1	R2	6		
AND	PC+4	PC	rs	rt	rd	ALU	R1	R2	7		
OR	PC+4	PC	rs	rt	rd	ALU	R1	R2	8		
NOR	PC+4	PC	rs	rt	rd	ALU	R1	R2	10		
SLT	PC+4	PC	rs	rt	rd	ALU	R1	R2	11		
SLTU	PC+4	PC	rs	rt	rd	ALU	R1	R2	12		
JR	R1	PC	rs								
SYSCALL	PC+4	PC	2	4							
J	(PC+4)[31:28]] IMM26 00	PC									
JAL	(PC+4)[31:28]] IMM26 0	PC			31	PC+ 4					
BEQ	PC+4/PC+4+ IMM<<2	PC	rs	rt			R1	R2			
BNE	PC+4/PC+4+	PC	rs	rt			R1	R2			

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
	IMM<<2										
ADDI	PC+4	PC	rs		rt	ALU	R1	IMM	5		
ANDI	PC+4	PC	rs		rt	ALU	R1	IMM	7		
ADDIU	PC+4	PC	rs		rt	ALU	R1	IMM	5		
SLTI	PC+4	PC	rs		rt	ALU	R1	IMM	11		
ORI	PC+4	PC	rs		rt	ALU	R1	IMM	8		
LW	PC+4	PC	rs		rt	Dout	R1	IMM	5	ALU	
SW	PC+4	PC	rs	rt			R1	IMM	5	ALU	R2
XOR	PC+4	PC	rs	rt	rd	ALU	R1	R2	9		
SLTIU	PC+4	PC	rs		rt	ALU	R1	IMM	12		
LB	PC+4	PC	rs		rt	Dout	R1	IMM	5	ALU	
BGEZ	PC+4/PC+4+ IMM<<2	PC	rs						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	说明	取值(何时为 1)
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

华中科技大学课程设计报告

MemToR eg	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 t	立即数符号扩展	ADDI、ADDIU、SLTI 指令
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
LB	LB 指令译码信号	LB 指令
BGEZ	BGEZ 指令译码信号	BGEZ 指令, 用于有条件分支控制

对照所有控制信号, 依次分析各条指令, 分析该指令执行过程中需要哪些控制信号, 对于与本条指令无关的控制信号, 控制信号的取值一律为 0, 以简化控制器电路的设计。该控制信号表的框架如表 2.4 所示。

表 2.4 主控制器控制信号框架

指令	ALU_OP	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL	LB	BGEZ
SLL	0				1			1							
SRA	1				1			1							
SRL	2				1			1							
ADD	5				1			1							
ADDU	5				1			1							
SUB	6				1			1							
SUB	7				1			1							
OR	8				1			1							
NOR	10				1			1							
SLT	11				1			1							
SLTU	12				1			1							
JR	X										1	1			
SYSCALL	X					1		1							
J	X											1			
JAL	X				1							1	1		
BEQ	X								1						
BNE	X									1					
ADDI	5			1	1		1								
ANDI	7			1	1										
ADDIU	5			1	1		1								
SLTI	11			1	1		1								
ORI	8			1	1										
LW	5	1		1	1										
SW	5		1	1											
XOR	9				1			1							
SLTIU	12			1	1		1								
LB	5	1		1	1		1							1	
BGEZ	11														1

2.2 流水 CPU 设计

2.2.1 总体设计

流水 CPU 的主要设计思想是，利用时间重叠，来提高 CPU 的指令吞吐量，通过指令级并行来提高执行速度。在课程设计指导视频的指导下，我们可以根据一个指令执行的过程，可以将 MIPS 指令的执行过程分为 5 个阶段，分别是：取指令阶段 IF、译码阶段 Decode、执行阶段 EXEXUTION、访存阶段 MEMACCESS、写回阶段 WRITE_BACK。同时在不同的阶段之间添加锁存器，锁存该阶段的数据，从而达到段与段之间的分离，避免了部分数据冲突。

2.2.2 流水接口部件设计

流水接口部件的内部由很多寄存器构成。流水线接口部件用来满足上一阶段与下一阶段的连通功能，即下一阶段所需的数据需要通过流水接口部件输出，上一阶段产生的数据也在流水接口暂存。为了满足课程设计的要求，流水接口部件应包含时钟端，清零端，使能端，数据输入端，数据输出端等。

2.2.3 理想流水线设计

在单周期硬布线 CPU 的实现基础上，结合视频指导课程的指导，将理想流水线每一段分离开，并在段间插入流水接口部件(那一大堆寄存器)，重新绘制理想流水线的通路。两者的主要的区别在于分段的思想与额外寄存器的存在，而数据通路的大致结构非常相似。

2.3 气泡式流水线设计

因为理想流水线只能在理想情况下正常运行，不能处理任何实际存在的数据冲突问题。为了解决这个问题，设计了气泡式流水线。气泡式流水线的主要设计思想是，当流水线中检测到数据冲突时，就用气泡(NOP)暂停部分流水线阶段的执行，不断的插入气泡，直到数据冲突被解决。为了实现上述功能，首先需要实现一个能检测到数据冲突的电路，当数据冲突检测电路检测到数据冲突时，则插入气泡从而暂停流水线，等待数据冲突被解决。注意到 ID 段和 WB 段之间的存在险象，会导致最新写入 REGFILE 的数据在同周期内无法正确读出，将寄存器文件改为下边缘可

以解决这个问题，但不能过 educoder 的自动测试。于是我增加了一个 REGFILE_HACKED，增加了一个比较输入 W#ADDR 和 R1#/R2#的电路，当符合的时候直接将 Din 给出，解决了这个问题。

2.4 重定向流水线设计

重定向流水线的功能与气泡流水线类似，都是为了解决数据冲突而提出的解决方案。不同点在于，重定向流水线是通过将所需要读但还没写入寄存器文件的数据直接定向到需要该数据的位置，从而解决了“写后读”冲突，即 Read After Write 冲突。而气泡流水线则是通过在出现数据冲突时插入气泡的方式，使流水线“暂停”一段时间。因此重定向流水线的执行效率相比气泡流水线效率更高。

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。

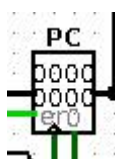


图 3.1 程序计数器 (PC)

② FPGA 实现：

程序计数器 PC 的 Verilog 代码如下：

```
always@(negedge clk,posedge clear)
begin
    if(clear)
        pc_out<=0;
    else if(!halt)
        pc_out<=pc_in;
end
```

2) 指令存储器 (IM)

① Logism 实现：

华中科技大学课程设计报告

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。

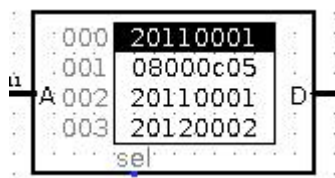


图 3.2 指令存储器（IM）

② FPGA 实现：

直接使用 Vivado 中自带的 ROM 作为指令存储器，其设置如错误!未找到引用源。所示。选择 ROM 的数据位宽为 32 位，因为该 ROM 的地址位宽为 10 位，所以选择 ROM 的大小选择为 1024。

指令存储器 IM 的 Verilog 代码如下：

```
pc pcmeml(im_in[11:2],im_out);
```

直接调用之前设置的 ROM 作为指令存储器，输入为指令地址的 2-11 位，输出为该指令。

3) 运算器（ALU）

Logism 实现：

采用课程设计中老师提供的 ALU 即可。如图 3.3 所示：

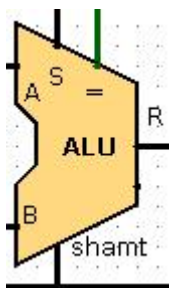


图 3.3 运算器（ALU）

4) 寄存器堆（RegFile）

Logism 实现：

华中科技大学课程设计报告

采用课程设计中老师提供的 RegFile 即可。如图 3.4 所示：

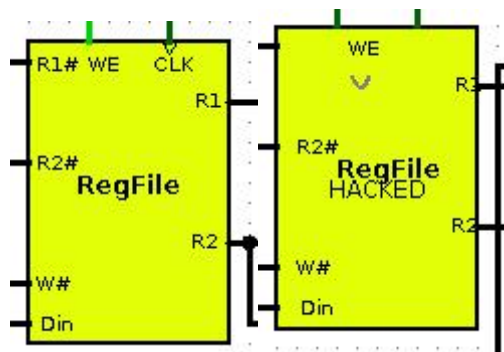


图 3.4 寄存器堆 (RegFile)

在理想/气泡/重定向流水线 CPU 中, RegFile 是经过修改的, 注意到 ID 段和 WB 段之间的存在险象, 会导致最新写入 REGFILE 的数据在同周期内无法正确读出, 将寄存器文件改为下边缘可以解决这个问题, 但不能过 educoder 的自动测试。于是我增加了一个 REGFILE_HACKED, 增加了一个比较输入 W#ADDR 和 R1#/R2# 的电路, 当符合的时候直接将 Din 给出, 解决了这个问题。

5) 数据存储器 (DM)

Logism 实现：

直接使用 logisim 中库文件中自带的 RAM 即可满足要求。如图 3.5 所示：

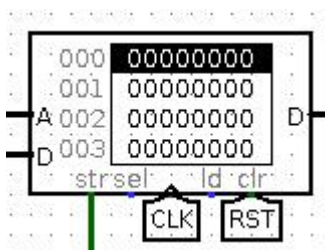


图 3.5 数据存储器 (DM)

3.1.2 数据通路的实现

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

根据总体方案设计中数据通路设计那一小节的详细内容，具体分析每一条指令

华中科技大学课程设计报告

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	I M	RF				ALU			DM		Tube
			R1 #	R2 #	W #	Din	A	B	O P	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			
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。

华中科技大学课程设计报告

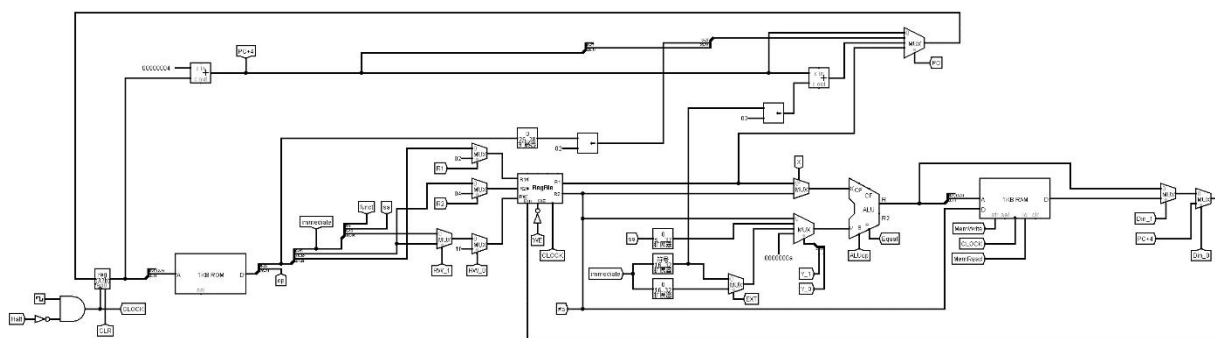


图 3.6 单周期 CPU 数据通路 (Logism)

在 Vivado 中使用 Verilog 语言搭建的数据通路的原理图如图 3.7 所示。

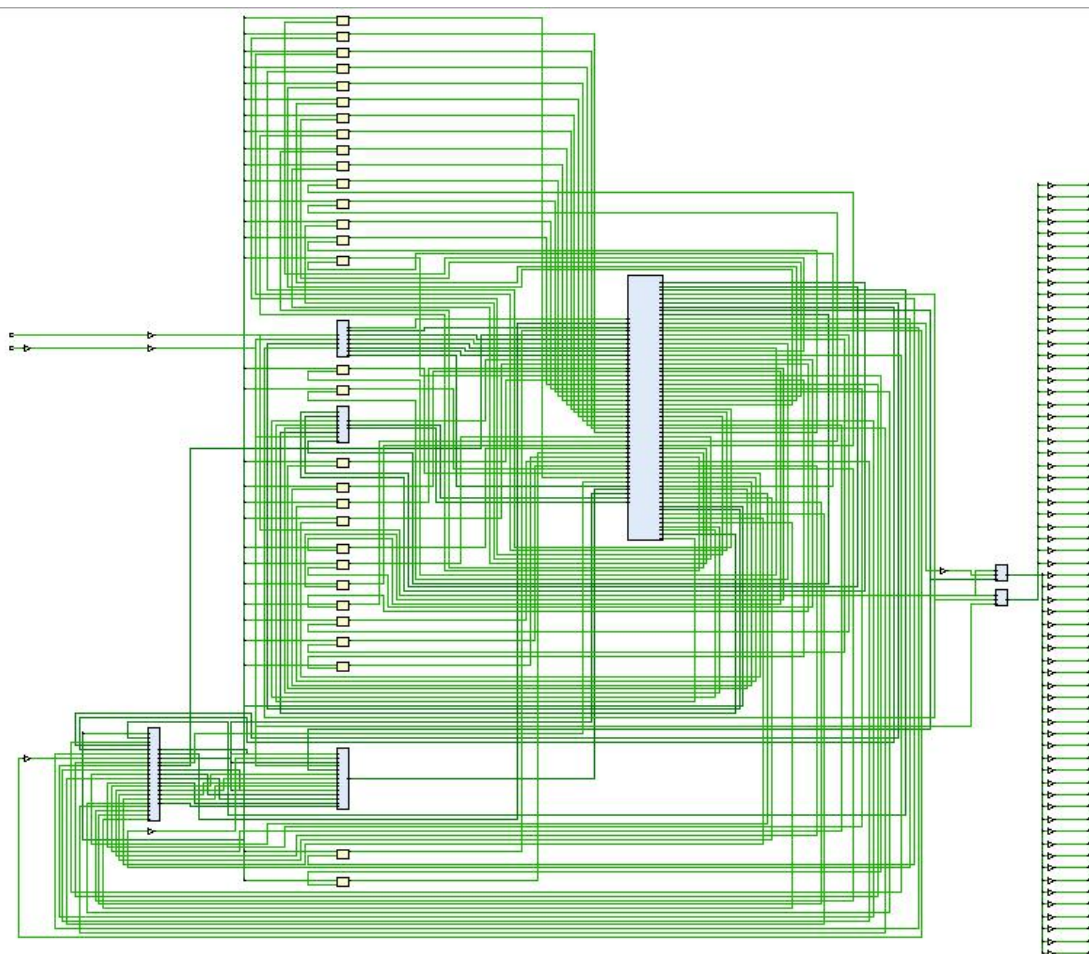


图 3.7 单周期 CPU 数据通路 (FPGA)

3.1.3 控制器的实现

根据总体方案设计中控制器的设计那一小节的相关内容，分别在 Logism 和 Vivado 上进行主控制器、Branch 控制器、SYSCALL 控制器的具体实现。

华中科技大学课程设计报告

主控制器对照表 3.2 所示控制信号表进行生成。

表 3.2 主控制器控制信号

指令	R	R W	W E	X	EXT	Y	ALUOp	MemWrite	MemRead	Din	Branch	SYSCALL
ADD	0 0	00	1	0	0	0 0	0101	0	0	00	00	0
ADDI	0 0	10	1	0	0	1 0	0101	0	0	00	00	0
ADDIU	0 0	10	1	0	0	1 0	0101	0	0	00	00	0
ADDU	0 0	00	1	0	0	0 0	0101	0	0	00	00	0
AND	0 0	00	1	0	0	0 0	0111	0	0	00	00	0
ANDI	0 0	10	1	0	1	1 0	0111	0	0	00	00	0
SLL	0 0	00	1	1	0	0 1	0000	0	0	00	00	0
SRA	0 0	00	1	1	0	0 1	0001	0	0	00	00	0
SRL	0 0	00	1	1	0	0 1	0010	0	0	00	00	0
SUB	0 0	00	1	0	0	0 0	0110	0	0	00	00	0
OR	0 0	00	1	0	0	0 0	1000	0	0	00	00	0
ORI	0 0	10	1	0	1	1 0	1000	0	0	00	00	0
NOR	0	00	1	0	0	0	1010	0	0	00	00	0

华 中 科 技 大 学 课 程 设 计 报 告

	0					0						
LW	0 0	10	1	0	0	1 0	0000	0	1	10	00	0
SW	0 0	00	0	0	0	1 0	0000	1	0	00	00	0
BEQ	0 0	00	0	0	0	0 0	0000	0	0	00	01	0
BNE	0 0	00	0	0	0	0 0	0000	0	0	00	01	0
SLT	0 0	00	1	0	0	0 0	1011	0	0	00	00	0
SLTI	0 0	10	1	0	0	1 0	1011	0	0	00	00	0
SLTU	0 0	00	1	0	0	0 0	1100	0	0	00	00	0
J	0 0	00	0	0	0	0 0	0000	0	0	00	10	0
JL	0 0	01	1	0	0	0 0	0000	0	0	01	10	0
JR	0 0	00	0	0	0	0 0	0000	0	0	00	11	0
SYSCAL L	1 1	00	0	0	0	1 1	0000	0	0	00	00	1

① FPGA 实现

根据在 Logism 实现中得到的各个一位控制信号的表达式，直接使用数据流建模，使用 assign 分的 Verilog 代码过于冗长，故只取对于控制信号 X 的生成代码举例如下：

华中科技大学课程设计报告

assign

```
X=(~OP[5]&~OP[4]&~OP[3]&~OP[2]&~OP[1]&~OP[0]&~F[5]&~F[4]&~F[3]&~F[2]&~F[1]&~F[0])|(~OP[5]&~OP[4]&~OP[3]&~OP[2]&~OP[1]&~OP[0]&~F[5]&~F[4]&~F[3]&~F[2]&F[1]&F[0])|(~OP[5]&~OP[4]&~OP[3]&~OP[2]&~OP[1]&~OP[0]&~F[5]&~F[4]&~F[3]&~F[2]&F[1]&~F[0]);
```

以此类推，最终便可以实现整个主控制器中所有控制信号的生成。在 Vivado 中使用 Verilog 语言构成的主控制器原理图如图 3.8 所示。

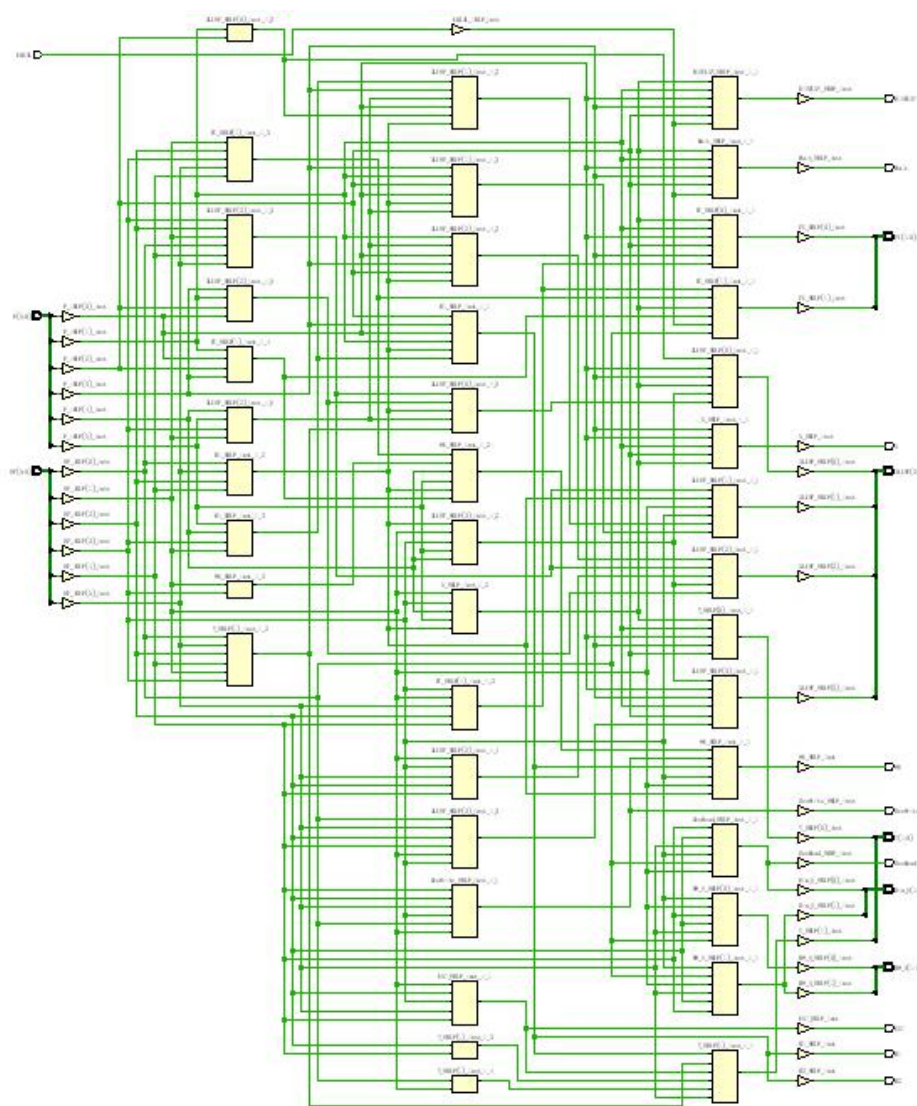


图 3.8 主控制器原理图

3.2 流水 CPU 实现

3.2.1 流水接口部件实现

流水接口包含时钟输入 CLK，同步清零端 CLR，使能端 ENABLE，气泡输入端，数据输入和输出端。流水接口部件的内部由很多寄存器构成，用来满足上一阶段与下一阶段的连通功能，即下一阶段所需的数据需要通过流水接口部件输出，上一阶段产生的数据也在流水接口暂存。现以 IF/ID 流水接口部件与 EX/MEM 流水接口部件为例，其 logisim 实现分别如下图 3.9 与 3.10 所示：

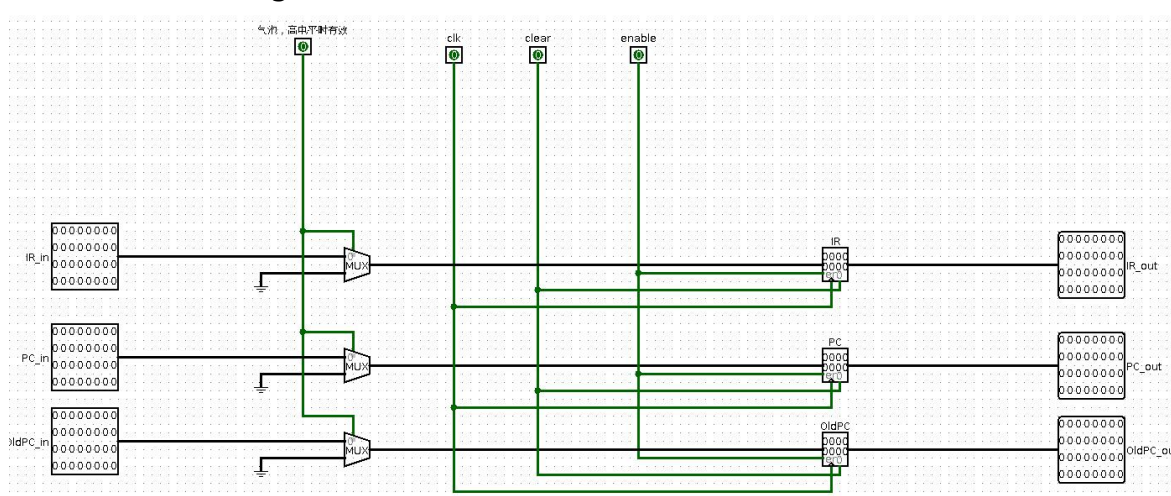


图 3.9 IF/ID 流水接口部件 logisim 实现

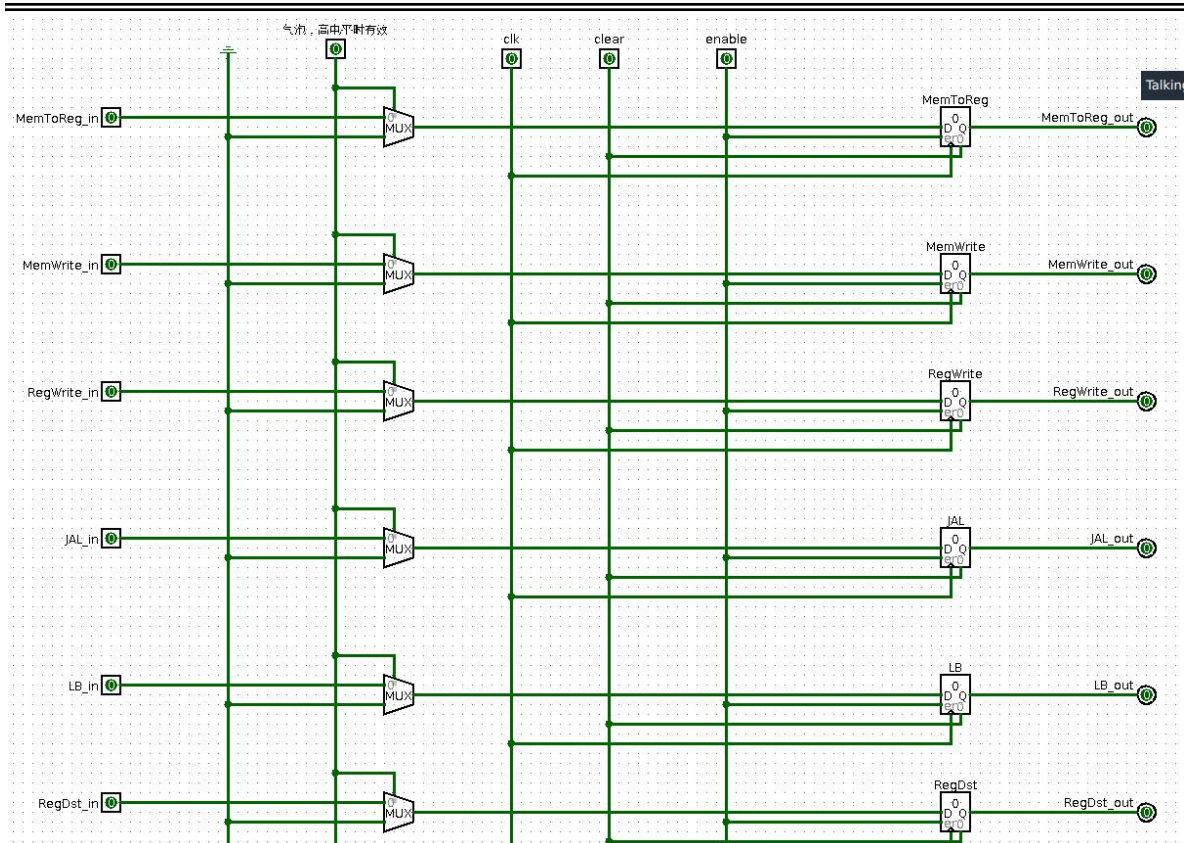


图 3.10 EX/MEM 流水接口部件 logisim 实现

3.2.2 理想流水线实现

在单周期硬布线 CPU 的实现基础上，结合视频指导课程的指导，将理想流水线每一段分离开，并在段间插入流水接口部件(那一堆寄存器)，重新绘制理想流水线的通路。两者的主要的区别在于分段的思想与额外寄存器的存在，而数据通路的大致结构非常相似。基于此设计思想以及已实现的流水接口部件，类比单周期硬布线 CPU 的数据通路进行连接即可。如下图 3.11 所示：

华中科技大学课程设计报告

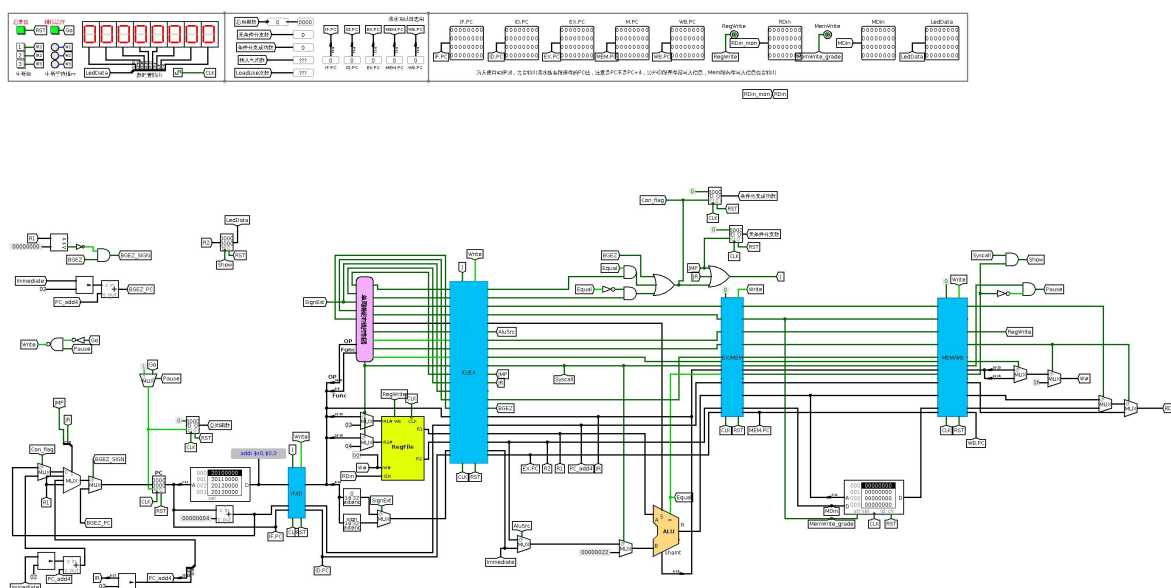


图 3.11 理想流水线 logisim 实现

3.3 气泡式流水线实现

1. 源寄存器使用情况电路实现

这是数据冲突检测电路的一个部分，只需要填写 Excel 表格，利用此工具来自动生成逻辑表达式即可，生成的电路如下图 3.12 所示：

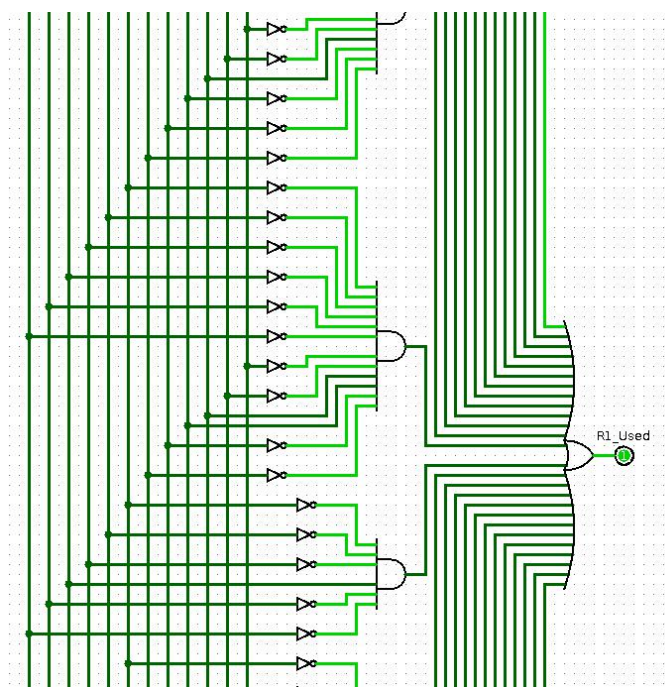


图 3.12 源寄存器使用情况部分 logisim 实现

2. 数据冲突检测电路实现

利用上面已经实现的源寄存器使用情况检测电路，可以实现数据冲突检测电路。它能够检测 ID, EX, MEM 段的数据冲突，具体实现如下图 3.13 所示。

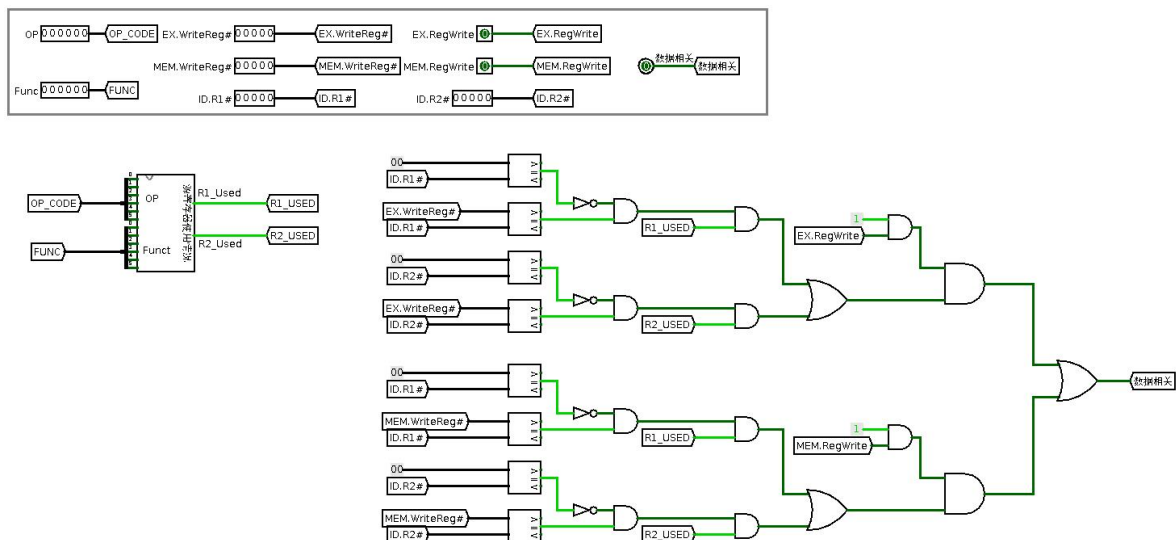


图 3.16 数据冲突检测逻辑电路 logisim 实现

3. 总体数据通路实现

在实现了数据相关检测电路后，在理想流水线电路的基础上，利用产生的数据冲突信号控制气泡的插入和流水线的暂停，即可成功实现所需的功能。最终 Logism 数据通路如下图 3.17 所示：

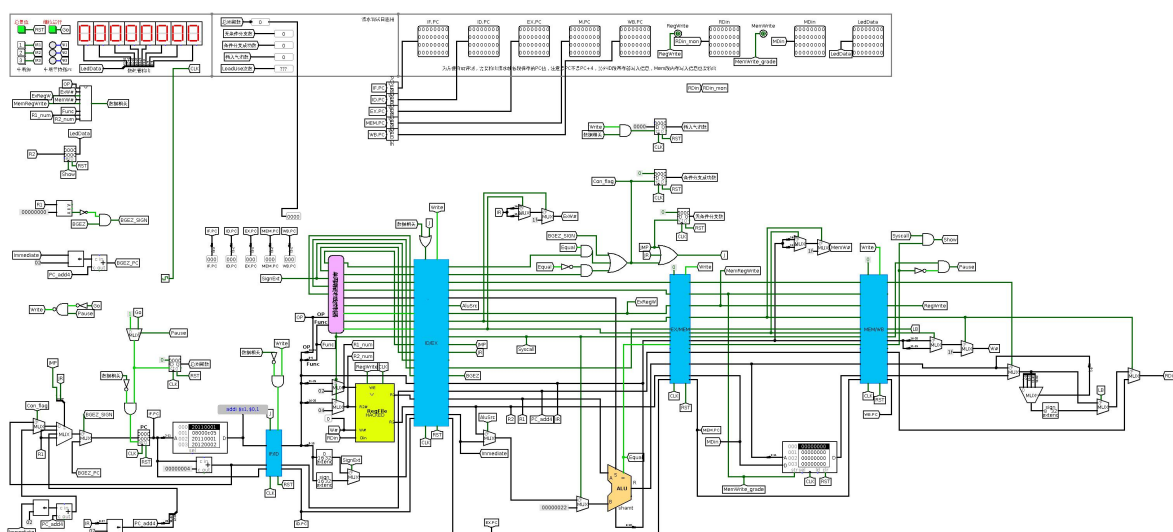


图 3.17 气泡流水线的数据通路 logisim 实现

3.4 重定向流水线实现

1. Helper 电路

华中科技大学课程设计报告

为了保持流水线主电路的简洁性，有必要把部分便于封装的电路抽出。这里也利用到了气泡流水线的源寄存器检测电路，用来检测 load-use 重定向检测相关逻辑。依照总体设计的思路，通过对各个阶段输入数据的比较，产生数据选择的控制信号以及 load-use 信号。其具体实现如下图 3.18 所示：

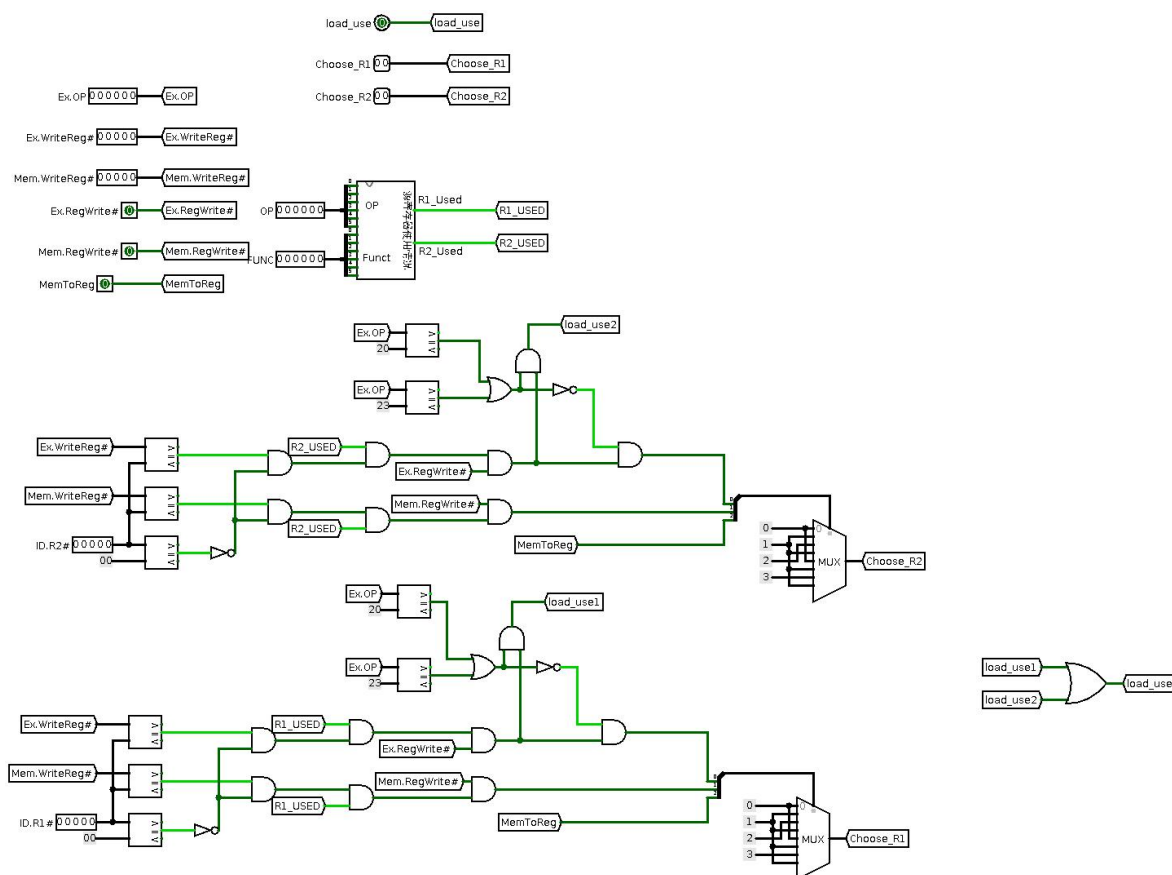


图 3.18 重定向相关电路 logisim 实现

2. 总体数据通路实现

利用已经实现的重定向相关检测电路，基于之前实现的气泡流水线，参照总体设计中的思路，为逻辑电路图增加重定向相关功能，即可完成重定向流水线的总体数据通路。主要修改的部分在于 ALU 的输入端的位置，需要利用控制信号选择可能的多个输入数据中的一种。总体数据通路的实现如下图 3.19 所示：

华中科技大学课程设计报告

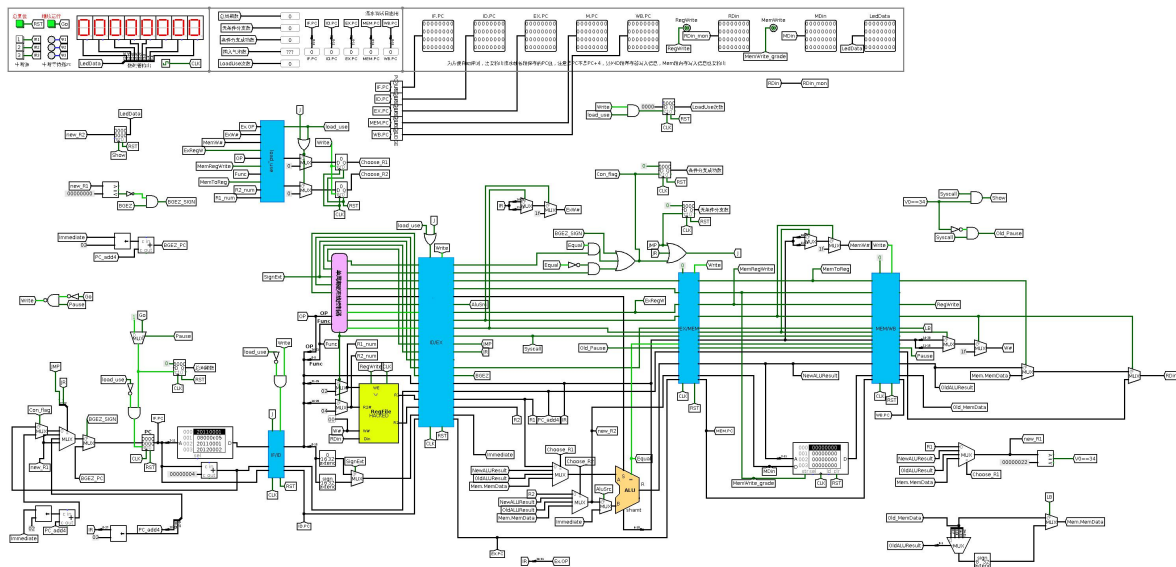


图 3.19 重定向流水线的数据通路 logisim 实现

4 实验过程与调试

4.1 测试用例和功能测试

测试用例使用与 educoder 平台相同的用例进行测试。除理想流水线使用特定的测试样例外，其余 CPU 均使用老师提供的 benchmark.hex 进行测试。

4.1.1 单周期硬布线 CPU 测试

此测试中使用 benchmark.hex 进行测试，最终总周期数为 1545，与标准答案相同。观察 LED 的变化情况，与程序预期现象一致。结果如图 4.1:

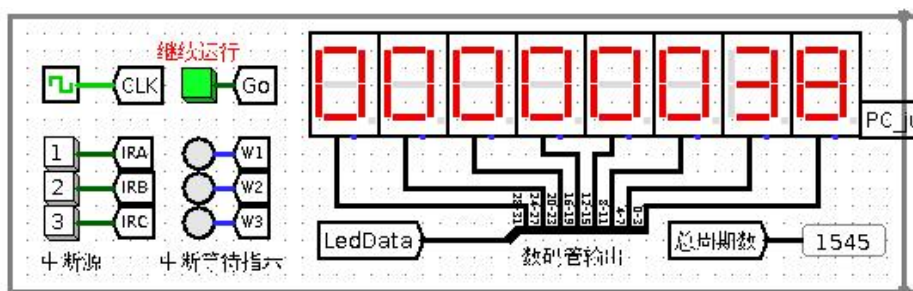


图 4.1 单周期硬布线 CPU 测试图

打开数据存储器，观察数据的排列方式，发现已经成功按降序排序，正确实现功能。测试截图如下图 4.2 所示：

```
000 0000000e 0000000d 0000000c 0000000b
004 0000000a 00000009 00000008 00000007
008 00000006 00000005 00000004 00000003
00c 00000002 00000001 00000000  ffffffff
010 00000000 00000000 00000000 00000000
```

图 4.2 单周期硬布线 CPU 测试图 RAM

4.1.2 理想流水线 CPU 测试

由于理想流水线 CPU 不能运行常规的 MIPS 指令，必须保证没有数据冲突。因此此测试中使用理想流水线专用测试样例进行测试。这个测试样例很短，只有 20 个时钟周期。检查最终的数据存储器，发现内容正确。周期数也与期望相同。测试截图如图 4.3 所示。

华中科技大学课程设计报告

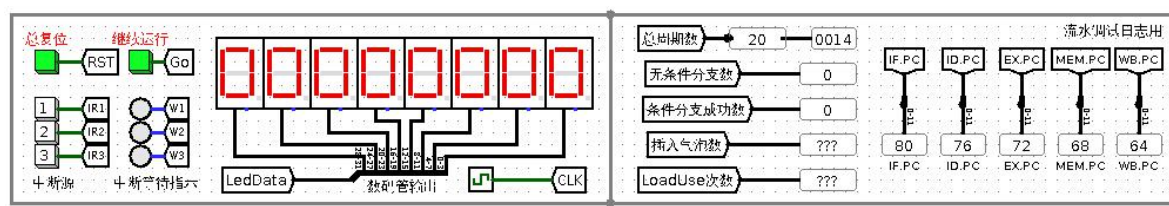


图 4.3 理想流水线 CPU 测试图

观察数据存储器，数据存储器在 0 位置依次写入了 0,1,2,3 四个整数，结果与预期一致。如下图 4.4 所示：

```
000 00000000 00000001 00000002 00000003
004 00000000 00000000 00000000 00000000
008 00000000 00000000 00000000 00000000
00c 00000000 00000000 00000000 00000000
```

图 4.4 理想流水线 CPU 测试图 RAM

在这里对流水线每一阶段的 PC 值进行了检查，确保理想流水线没有错误，继续进行后面的开发。

4.1.3 气泡流水线 CPU 测试

此测试中使用 benchmark.hex 进行测试，最终总周期数为 3623，与标准答案相同。观察 LED 的变化情况，与程序预期现象一致。结果如图 4.5：

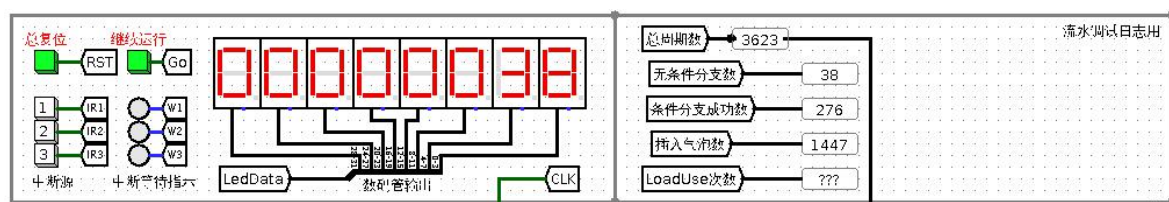


图 4.5 气泡流水线 CPU 测试图

打开数据存储器，观察数据的排列方式，发现已经成功按降序排序，正确实现功能。测试截图如下图 4.6 所示：

```
000 0000000e 0000000d 0000000c 0000000b
004 0000000a 00000009 00000008 00000007
008 00000006 00000005 00000004 00000003
00c 00000002 00000001 00000000  ffffffff
```

图 4.6 气泡流水线 CPU 测试图 RAM

华中科技大学课程设计报告

4.1.4 重定向流水线 CPU 测试

此测试中使用 benchmark.hex 进行测试, 最终总周期数为 2297, 与标准答案相同. 观察 LED 的变化情况, 与程序预期现象一致. 结果如图 4.7:

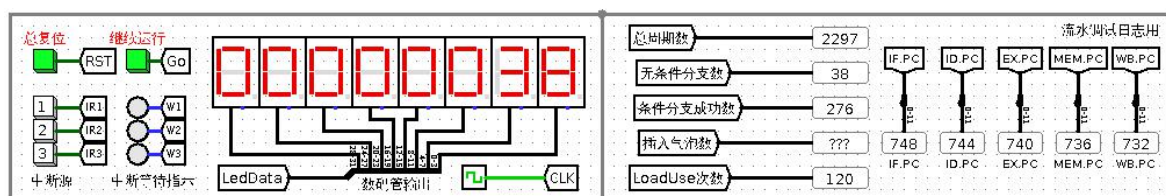


图 4.7 重定向流水线 CPU 测试图

打开数据存储器, 观察数据的排列方式, 发现已经成功按降序排序, 正确实现功能. 测试截图如下图 4.8 所示:

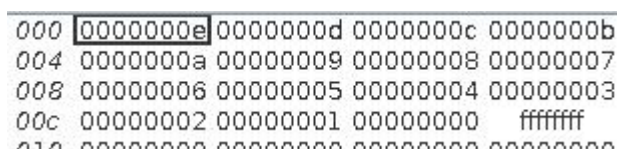


图 4.8 重定向流水线 CPU 测试图 RAM

4.2 educoder 测试结果

对电路的以下更改是仅为了通过 educoder 的测试而做的:

1. 对 PC 输出结果的高 16 位设置为 0 后再输出. PC_JUDGE 电路负责这件事.
2. 使用 Hacked 的 Regfile. 在 2.3 节 气泡流水线设计中, 提到了这个问题. 这里不在赘述. 上边缘, 下边缘, 和 Hacked 这三种 Regfile 的实现都被我测试过, 测试结果如下图 4.9 所示. 根据此测试结果, 我为 2 种含有气泡的流水线 CPU 都使用了 HACKED REGFILE.

P means "passing educoder.net", W means "working, runs benchmark correctly", N means "can not run benchmark correctly", X means "untested".

	default REGFILE (RisingEdge)	FallingEdge REGFILE	HACKED REGFILE (RisingEdge)
单周期MIPS	PW	X	N
理想流水线	PW	X	X
气泡流水线	N	W	PW
重定向流水线	W	W	PW

华中科技大学课程设计报告

图 4.9 不同 REGFILE 的险象问题测试结果

下图 4.10 展示了 educoder 平台提供的总体测试结果.



图 4.10 educoder 测试平台结果图

4.3 性能分析

对于不同实现方式的 CPU 的周期数差异的问题, 通过运行 benchmark 测试程序, 测试不同的 CPU 逻辑电路。可以观察到, 单周期 CPU 总周期数 1545, 气泡流水线总周期数为 3623, 重定向流水线总周期数为 2297。虽然单周期 CPU 总周期数最少, 但我们创造流水线 CPU 就是为了进一步缩短每个硬件时钟周期时间, 把 CPU 的频率提升到更高。单周期 CPU 虽然周期数少, 但由于它非常长的数据通路, 门延迟导致单个周期的耗时很长, 导致最终性能上, 气泡流水线和重定向流水线都要比单周期 CPU 优异的多。

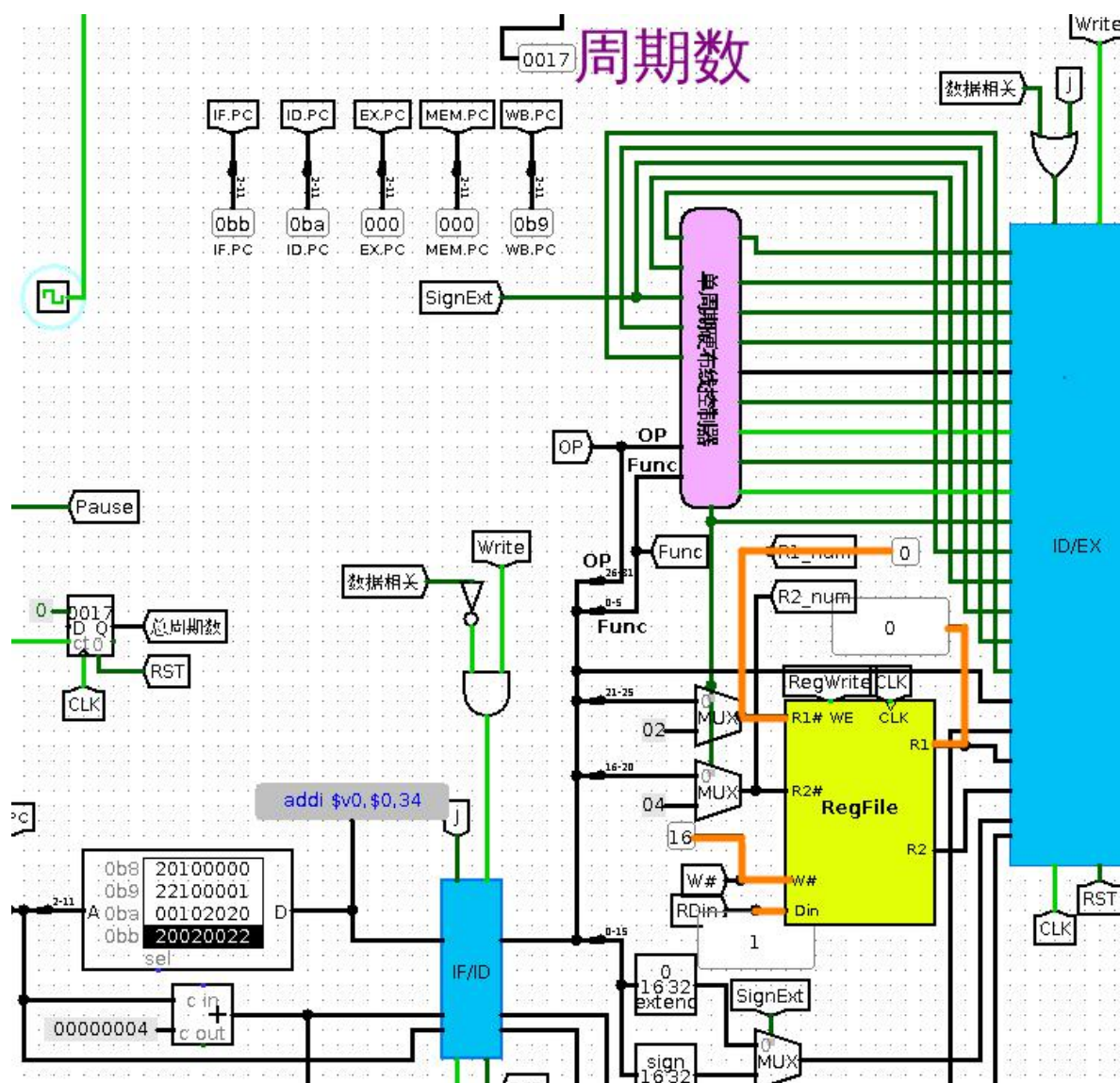
而重定向流水线与气泡流水线相比, 因为采用重定向计数, 导致插入的气泡数大大减少, 因此大大降低 CPU 流水线运行的停顿, 在气泡流水线的基础上额外获得了将近 60% 的性能提升。

4.4 主要故障与调试

4.4.1 RegFile 险象故障

理想流水线：接口处数据传输问题。

故障现象：气泡流水线 0x0017 周期，上边缘出现之后，RegFile 没能把本周期内写入的数据输出。如图 4.11 所示的橙色线，标出了有问题的 R1 输出，本来应该输出 Din=1，实际输出 0。



华中科技大学课程设计报告

输出的 R1 是旧值. 经分析, 我尝试了将寄存器设置为下边缘触发, 这确实解决了问题(见图 4.9), 但仍然不能通过 educoder 的自动测试. 因此, 我编写了 Hacked Regfile, 结构如图 4.12 所示. 此 Regfile 封装(如图 3.4)与默认的上边缘 Regfile 完全相同, 只需要更换上去, 就可以完全解决问题了.

解决方案:我编写了 Hacked Regfile, 结构如图 4.12 所示. 此 Regfile 封装(如图 3.4)与默认的上边缘 Regfile 完全相同, 只需要更换上去, 就可以完全解决问题了.

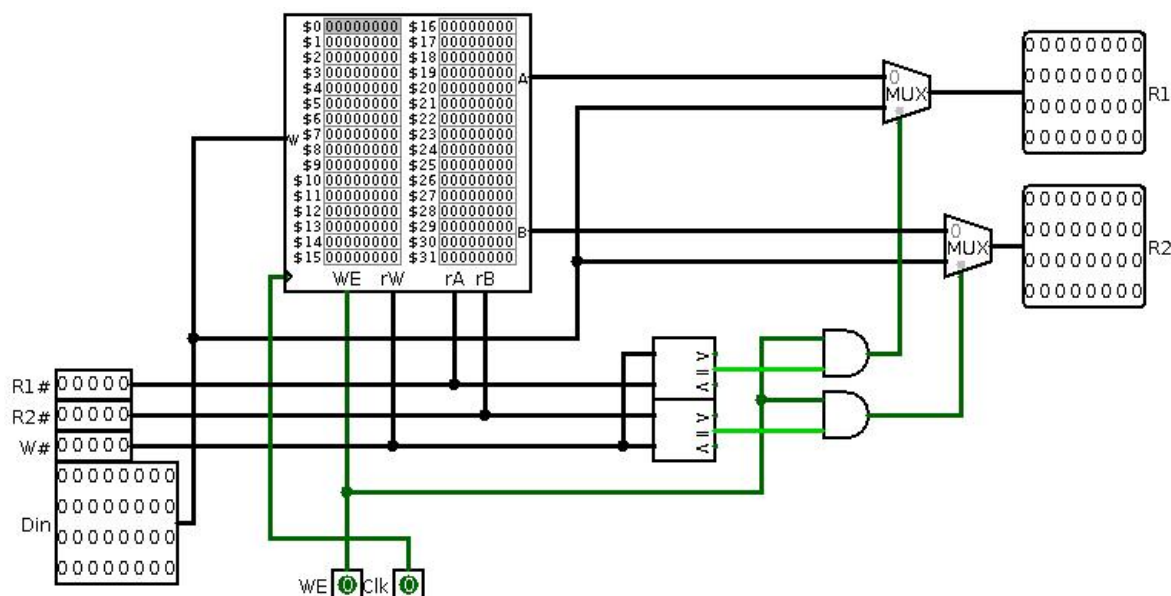


图 4.12 解决问题的 Hacked Regfile 电路图

4.5 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 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 Logisim 单周期 CPU 报告。
第三天	完成 Logisim 单周期 CPU 的故障报告, 并且调试单周期 CPU, 使其通过了 Logisim 单周期 CPU 的 educoder 测试。
第四天	学习理想流水线的相关 MOOC 视频, 画好了各级流水接口部件。
第五天	完成理想流水线的大致数据通路, 开始进行总体调试。借助 educoder 进行

华中科技大学课程设计报告

时间	进度
	调试工作.
第六天	理想流水线的logisim已经调试通过, 通过了educoder的在线检查. 开始进行后面分支相关和气泡流水线的工作.
第七天	复习关于分支相关/气泡流水线的知识点, 学习MOOC, 完成流水线的分支功能, 仍然在进行气泡流水线相关工作。
第八天	完成数据冲突和分支处理, 气泡流水线最终解决了所有问题, 用下边缘寄存器能够正确运行, 但无法通过educoder的测试. 转而寻找其他办法解决这个问题.
第九天	气泡流水线成功通过了educoder的测试, 完成了重定向关键逻辑, 没有进行测试.
第十天	完成了重定向流水线的全部数据通路. 最终把重定向流水线通过了educoder测试.

5 设计总结与心得

5.1 课设总结

在这次为期两周的课设中，我共完成了以下工作内容：

- 1) 设计并绘制了单周期 MIPS CPU 的数据通路，能够运行 24 条基础指令加 4 条拓展指令。
- 2) 设计并实现了理想流水线与分支相关流水线的逻辑电路图。
- 3) 设计并实现了气泡流水线的逻辑电路图。
- 4) 设计并实现了重定向流水线的逻辑电路图。

5.2 课设心得

计算机组成原理课程设计是所有实验以及课程设计中耗时相对较长的一门。主要是作为一个对硬件不熟悉的软件工程师，有很多工作都需要对照 MOOC 视频进行学习，在学习的过程中完成实验。因为疫情的原因，进行开发所需要的几乎所有设备都在学校宿舍里，因此在家中完成课设的过程是非常痛苦的，经过两个星期的工作才终于完成了课程设计的主要设计任务。现在再来回顾整个课程设计的整个过程，CPU 正常运行和成功通过 educoder 测试的成就感自是不用说，而且其中也有不少细节值得我去深思与体会。

课程设计一开始的时候，第一个任务是要用 Logism 设计单周期硬布线 CPU，该任务和以前实验课已完成的任务非常相似，所以整个过程还算比较迅速，把电路移植过来就可以了。主要是修改一些接口，使其能够通过完整的测试，在这里没有遇到什么问题。

紧接着，理想流水线 CPU 的设计难度也不大，只需要在单周期硬布线 CPU 的基础上分离线路，插入每一段间的寄存器组就可以。但是使用插入气泡、数据重定向技术对于流水线 CPU 增加新特性时，因为这些方法书本上并没有，老师提供的 MOOC 上也只有简单的一些描述，这就要求我不断地在网上搜索相关的知识内容，同时自己思考一些部件的实现方法，一点一点解决调试中遇到的问题。最终成功解

华中科技大学课程设计报告

决了所有的问题，通过了测试。

老师在这次实验提供了 educoder 平台，虽然有一些小问题，我为了让好的电路通过 educoder 平台的测试而做了一些额外的努力。但总体上说，这个平台对于调试是由非常非常大的帮助的。在出问题是，这个平台能在错误刚发生时就发现错误，而不是自己调试时过了好多个时钟周期才发现执行结果错误，节约了大量的定位错误的 debug 时间。

在这个实验中，我也认识到了，封装与模块化不但对于软件开发至关重要，对于简化硬件设计也有很大意义。在设计阶段，如果将部分常用的检测电路，RAM，寄存器等电子元件封装并有效复用，就能起到简化设计的作用。在 AMD 的 Ryzen 架构中，更高层面的模块化使得 CPU 能够实现低成本的大量核心堆叠，正是模块化设计思想的体现。

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成为我整个大学生涯中一段无比难忘的回忆。

参考文献

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

·指导教师评定意见·

一、原创性声明

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

特此声明！

作者签字：

刘本嵩
Apr 22, 2020.