华中科技大学

2024

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：	CS2207
学　　号：	U202215561
姓　　名：	瞿明睿
电　　话：	13956929603
邮　　件：	3021018823@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/层次系统设计的组织思路，有效地增强对学生的计算机系统设计与实现能力的培养”。课程设计是完成该课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有一定规模的指令系统的简单计算机系统。所设计的系统能在 LOGISIM 仿真平台和 FPGA 实验平台上正确运行，通过检查程序结果的正确性来判断所设计计算机系统正确性。

课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行中央处理器底层电路的实现、故障分析与定位、系统调试等环节的综合锻炼，进一步提高学生分析和解决问题的能力。

1.2 设计任务

本课程设计的总体目标是利用 FPGA 以及相关外围器件，设计五段流水 CPU，要求所设计的流水 CPU 系统能支持自动和单步运行方式，能正确地执行存放在主存中的程序的功能，对主要的数据流和控制流通过 LED、数码管等适时的进行显示，方便监控和调试。尽可能利用 EDA 软件或仿真软件对模型机系统中各部件进行仿真分析和功能验证。在学有余力的前提下，可进一步扩展相关功能。

1.3 设计要求

（1） 根据课程设计指导书的要求，制定出设计方案；

（2） 分析指令系统格式，指令系统功能。

（3） 根据指令系统构建基本功能部件，主要数据通路。

（4） 根据功能部件及数据通路连接，分析所需要的控制信号以及这些控制信号的有效形式；

（5） 设计出实现指令功能的硬布线控制器；

（6） 调试、数据分析、验收检查；

（7） 课程设计报告和总结。

1.4 技术指标

（8） 支持表 1.1 指令集 27 条基本 32 位 RISC-V32 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 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	MUL	乘法	
29	REMU	无符号数的余数	
30	LBU	无符号字节加载	
31	LBTU	小于时分支	

2 总体方案设计

2.1 单周期 CPU 设计

使用 LOGISIM 来创建一个 32-位 5 段流水 CPU，该 CPU 运行的是标准 RISC-V 指令集的子集，关于指令功能请仔细查找资源包中的 RISC-V 指令手册，任务书给出的五种常见指令格式如下图 2.1.1 所示：

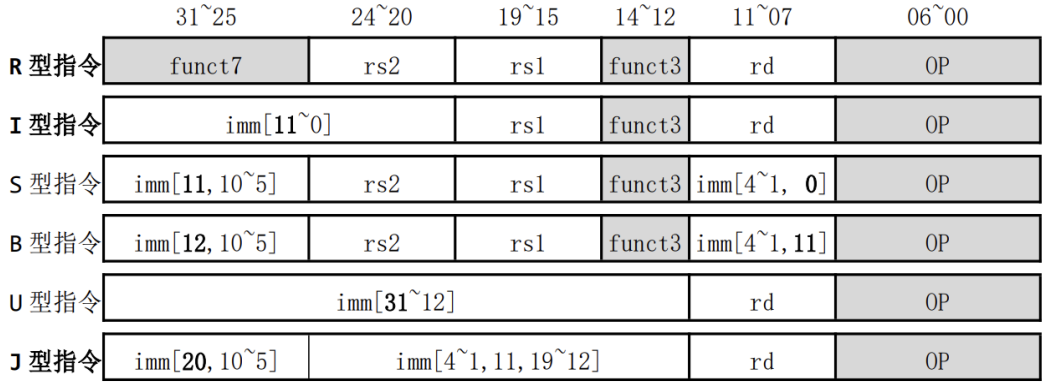

图 2.1.1 RISC-V 指令示意图

因为 CPU 架构为 RISC-V 类型，我们的设计方案是硬布线控制器法，具体来说是采用了三级时序硬布线控制器。同时采用了指令存储器和数据存储器相分离的哈佛结构来实现方案的设计。

在设计过程中，我们将硬布线控制器中的控制信号和运算信号分开生成封装再组成硬布线控制器，组成后再加辅以数据通路就可以得到 CPU 的整个电路图。阅读任务书可以得到简单思路：采用简单迭代法实现单周期处理器，只需先完成支持一条固定 R 型指令的基本数据通路，运行通过后，再在这个基础上不断增加新的数据通路，支持新的指令，直至所有指令都能正常运行。

总体结构图按照任务书的架构设计，如图 2.1.2 所示。

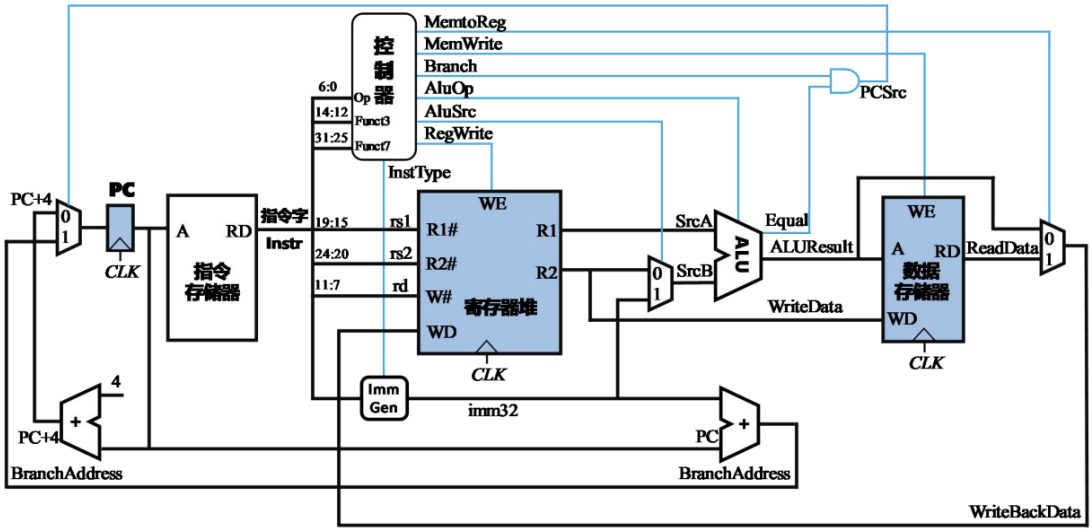

图 2.1.2 数据通路示意图

2.1.1 主要功能部件

1. 程序计数器 PC

对于 PC 的的内容，在单周期设计中，我们需要明确三种输入：ALU 计算的结果，PC+4 的结果，还有最后的指令跳转即立即数跳转。

这样我们可以采用多路选择器来实现这样的分层控制，控制源来自我们设计的，经过不同指令得到的多位信号来选择实现 PC 的值选择的设计，同时我们需要接上 halt 信号来保证 ecall 中断

2. 指令存储器 IM

指令寄存器需要读取 PC 的值对应地址空间寻找，充当我们的 RAM 的一部分，也是让指令和内存中其他数据分离，方便处理其他设计。

3. 运算器 ALU

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
A	输入	32	操作数

引脚	输入/输出	位宽	功能描述
B	输入	32	操作数
S	输入	4	运算器功能码，
Result	输出	32	ALU 运算结果
Result2	输出	32	ALU 结果第二部分
=	输出	1	操作数 A=B
<	输出	1	操作数 A<B
>=l	输出	1	操作数 A>=B

ALU 运算机器功能码如下图 2.1.3 所示

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

图 2.1.3 ALUOP 字段

4. 寄存器堆 RF

作为寄存器，寄存器堆有读和写的操作。针对读操作，需要输入两个寄存器 R1,R2 的地址，对应输出两个寄存器数据，同时需要使能信号。同理在写操作中，输入为写入的数据和寄存器的地址，同时需要使能信号。RF 还需要时钟接入。

5. 数据存储器

数据存储器有读操作和写操作。读操作输入地址，输出一个数据；写操作需要输

入地址、数据、使能信号。

2.1.2 数据通路的设计

程序计数器 PC 作为指令地址的源头，其输出连接到指令存储器 IM 的地址输入端。当 PC 输出当前指令地址后，IM 根据该地址读取相应的指令，并将指令输出到后续的数据通路中。指令在数据通路中的传递路径如下：从 IM 输出的指令，其操作码部分会被送往控制器，控制器依据操作码生成各类控制信号。同时，指令中的寄存器地址字段会被送往寄存器堆 RF，用于读写寄存器数据。

对于寄存器堆 RF，它有两个读端口和一个写端口。在读取操作时，根据指令中的 rs 和 rt 字段，从 RF 中读出相应寄存器的值，并将这两个值分别作为操作数送往算术逻辑运算单元 ALU 的 A 和 B 输入端。ALU 的运算由 ALUOP 决定。

对于涉及数据存储器访问的指令，ALU 的运算结果会作为数据存储器 DM 的地址输入。处理分支指令时，ALU 会对比较条件进行判断，并将比较结果输出到控制器。控制器根据比较结果和指令类型，生成相应的 PCSrc 信号，用于选择下一条指令的地址。如果分支条件成立，PCSrc 信号会使多路选择器选择分支目标地址作为下一条指令的地址；否则，选择 PC+4 作为下一条指令的地址，从而实现程序的正确跳转。

这样就完成初步的数据通路。

2.1.3 控制器的设计

我们在这里明确：控制器输出信号，针对控制信号进行统计，在对我们需要处理接受控制信号的结构加工成新的隧道从而提升可读性和健壮性，并且对各个统计信号的各种取值情况进行定义，统计得到的部分控制信号以及说明如表 2.2。

表 2.2 主控制器控制信号的作用说明

控制信号	取值	说明
Ecall	0	寄存器堆 R1、R2 口读取 rs 字段指示寄存器的值
	1	寄存器堆 R1、R2 口读取 2 号寄存器的值，控制 halt 信号
S_type	0	立即数取 I 型指令对应的字段
	1	立即数取 S 型指令对应的字段
JAL/JALR	0/1	取值为 1 时跳转，PC 取跳转地址的值

BEQ/BNE	0/1	取值为 1 且需要满足的条件成立时跳转，PC 取跳转地址的值
ALUOP	0-12	ALU 的功能
ALUSRCB	0/1	ALU 的 B 输入为 R2 还是立即数
MEMWRITE	0/1	取值为 1 时允许写入数据存储器
MEMTOREG	0/1	取值为 1 时写入寄存器的数据为读取到的数据存储器中的值
REGWRITE	0/1	取值为 1 时允许写入寄存器

首先明确我们采用硬布线控制器中的控制信号和运算信号分开生成封装再组成硬布线控制器的方法，那么我就需要根据各条指令，分析指令执行过程中需要哪些控制信号，对于无关控制信号取值为 0，以简化设计。

该控制信号表的框架如下图 2.1.4 信号框架图所示。

指令	Funct7 (十进制)	Funct3 (十进制)	OpCode (十六进制)	ALU_OP	MemtoReg	MemWrite	ALU_Src	RegWrite	ecall	S_Type	BEQ	BNE	Jal	jalr	bge	remu	RS_1	RS_2	LBU	BLTU
add	0	0	C	5				1									1	1		
sub	32	0	C	6				1									1	1		
and	0	7	C	7				1									1	1		
or	0	6	C	8				1									1	1		
slt	0	2	C	11				1									1	1		
sltu	0	3	C	12				1									1	1		
addi		0	4	5			1	1									1			
andi		7	4	7			1	1									1			
ori		6	4	8			1	1									1			
xori		4	4	9			1	1									1			
slti		2	4	11			1	1									1			
slli	0	1	4	0			1	1									1			
srli	0	5	4	2			1	1									1			
srai	32	5	4	1			1	1									1			
lw		2	0		1		1	1									1			
sw		2	8			1	1			1							1	1		
ecall	0	0	1C						1											
beq		0	18	6							1						1	1		
bne		1	18	6								1					1	1		
jal			1B					1					1							
jalr		0	19	6			1	1						1			1			
CSRRSI		6	1C																	
CSRRCI		7	1C																	
URET	2	0	1C						1											
MUL	1	0	C	3				1									1	1		
REMU	1	7	C	4				1								1	1	1		
LBU		4	0	5	1		1	1									1		1	
BLTU		6	18	12						1							1	1		1

图 2.1.4 信号框架图

2.2 流水 CPU 设计

2.2.1 总体设计

流水线 CPU 采用存储再转发的设计，我们可以在整个执行周期将其分为五个阶段，每个阶段我们需要设计由对应的功能的部件完成对应的功能，五个阶段为 IF→ID→EX→MEM→WB：

取指令 (IF)：负责从指令存储器取出指令；

指令译码 (ID)：操作控制器对指令字进行译码，同时从寄存器堆取操作数；

指令执行 (EX)：执行运算操作或计算地址；

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。

这里大概结构可以参考任务书中结构，如下图 2.2.1 流水线概览图

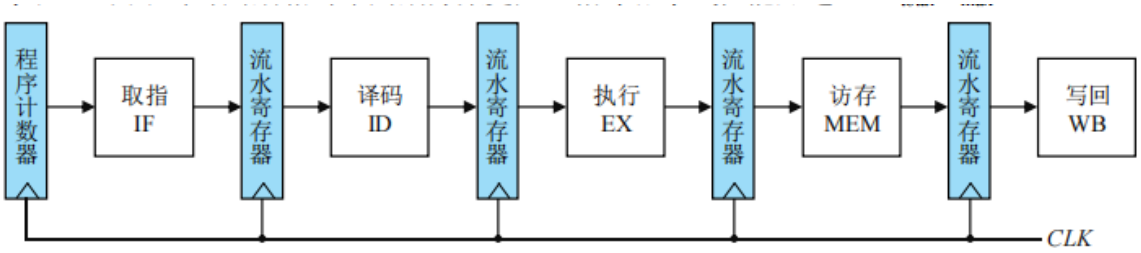

图 2.2.1 流水线概览图

2.2.2 流水接口部件设计

流水线的实现需要将 IF→ID→EX→MEM→WB 转换的具体数据保存一个周期并传递给下一个阶段，因此流水接口部件我们需要确定：时钟源实现周期性，复位能力有统一的复位信号，保存的数据接口，保存数据的寄存器，保存的数据传递出去的接口。

2.2.3 理想流水线设计

我们只需要在单周期 CPU 的基础上，加入 IF→ID→EX→MEM→WB 的改造，在这五个阶段加上对应阶段需要的流水接口部件，使得几个阶段能够实现存储再转发，而且可以看到每一个部分执行都是独立的而且独立并且串行起来。串行整体逻辑还是单周期 CPU 的逻辑，但是实际上却实现了流水线

2.3 气泡式流水线设计

2.3.1 冲突检测逻辑

对于冲突，我们需要明确：冲突有结构冲突、控制冲突、数据冲突三种类型。

针对结构冲突，我们可以采用哈佛结构将数据和指令分开存储来解决，也就是在外面的硬件上做出修改。针对控制冲突可以采用分支预测的方法解决；针对数据冲突可以采用插入气泡的方法解决。控制冲突和数据冲突需要增加冲突的检测逻辑。具体

实现来看，控制冲突需要增加清空信号将分支指令的指令清空，数据冲突需要增加气泡生成信号来增加气泡使得数据暂停。

2.3.2 气泡流水线

通冲冲突的解释，这里直接给出：气泡流水线在理想流水线的基础上加入了冲突检测逻辑，同时流水接口部件要加入新的的控制信号。在引入冲突检测逻辑之后，需要给接口部件增加同步清空信号等。示意如下图 2.3.1

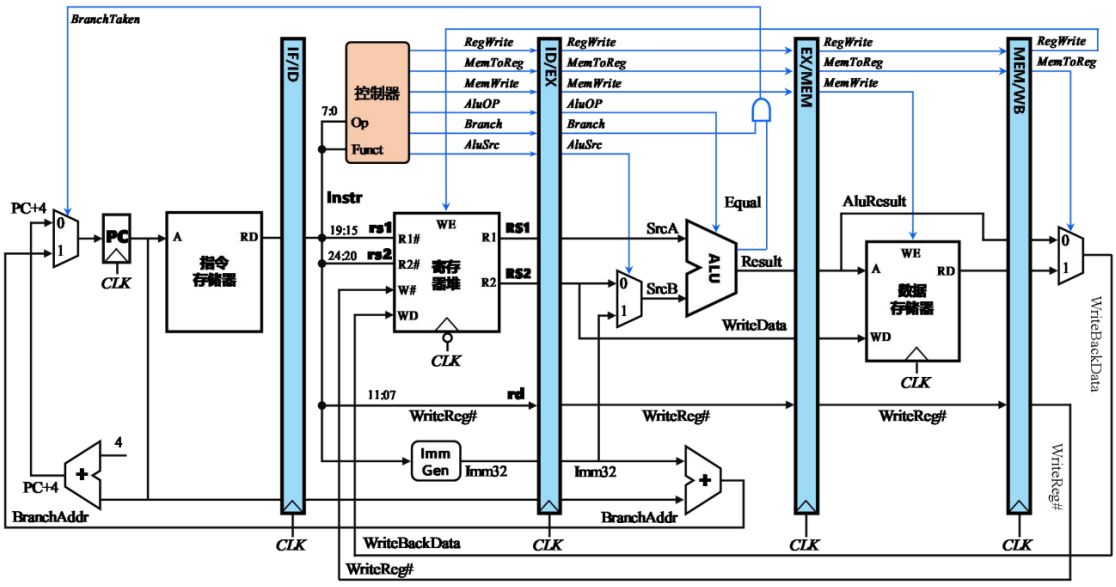

图 2.3.1 气泡流水线概览图

2.4 重定向流水线设计

重定向是气泡处理的另外一种策略，不会在冲突时直接将 ID 阶段的数据清空，而是正常的运行流水线，但在取值时采用重定向将 EX 段取出的数据进行纠正。

但是重定向对 load-use 相邻的指令不适用，传递给 EX 阶段的值可能并非写好的值会导致错误，那么在该指令下依旧需要通过增加气泡的方式来解决数据冲突。

所以重定向需要增加一个和气泡相似的冲突检测逻辑模块，该模块输入为 EX 阶段的信号，输出为重定向信号和气泡插入信号。顶层来看结构和气泡流水线相似。

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）

Logism 实现：

我们通过一个 32 位寄存器来实现程序计数器 PC 的功能，触发方式设置为下降沿触发，通过 PC 的功能这个寄存器的的输入时下一条指令的地址，输出为当前指令的地址。

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 为 1，屏蔽时钟信号，使整个电路停机。

如图 3.1.1 所示。

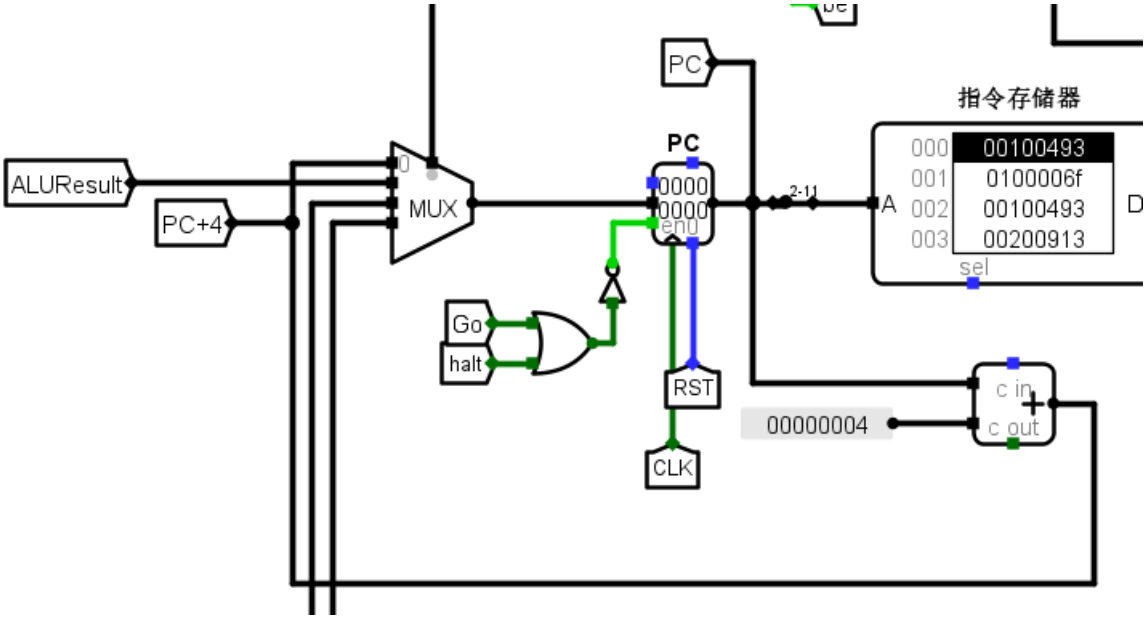

图 3.1.1 PC 的实现

2) 指令存储器（IM）

Logism 实现：

我们使用一个只读存储器 ROM 实现指令存储器。定义只读存储器的地址位宽为 10 位，数据位宽为 32 位。在 PC 地址有 32 位但是这里 ROM 地址线宽度有限（10 位），我们处理方法时直接屏蔽高位部分和字节偏移部分，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 3.1.2 所示。

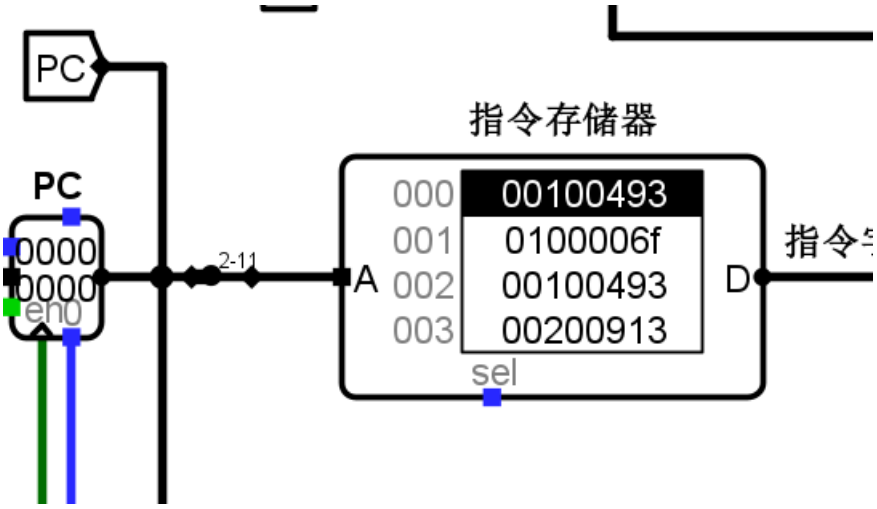

图 3.1.2 指令寄存器实现

3.1.2 数据通路的实现

我们可以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，最后将所有部件连接在一起即可。展示如下图 3.1.3。

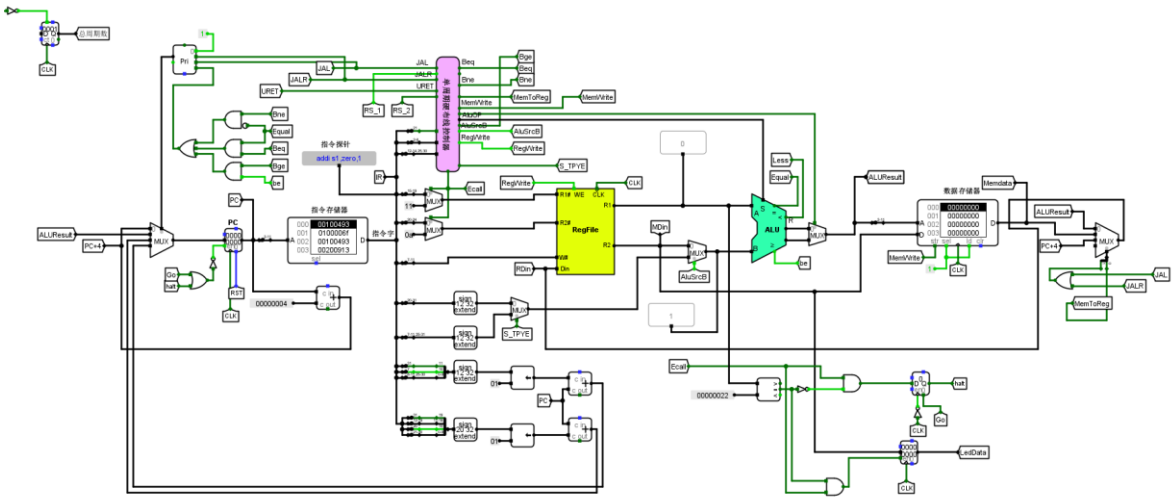

图 3.1.3 数据通路展示

3.1.3 控制器的实现

通过分析不难得到，我们的单周期硬布线控制器的输入为输入指令中的 OP_CODE、FUNCT、IR21 三个字段，输出时各个控制信号。

那在控制器的实现中，我们需要实现运算控制器和控制信号的产生，还有 uret 和 ecll 信号的输出，这里使用我们的指令的输出图，对应好我们的指令需要的输出信号，采用组合逻辑电路自动生成的功能，这里可以参考图 2.1.4。最后可得到这样一个顶层展示，如下图 3.1.4。

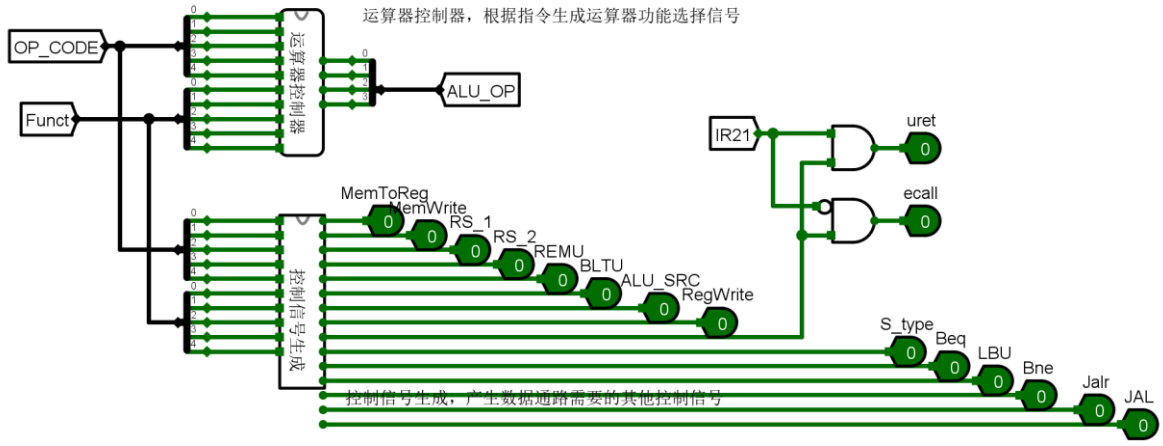

图 3.1.4 硬布线控制器顶层展示

我也可以看一下对于某一个的信号的生成样式，如图 3.1.5。

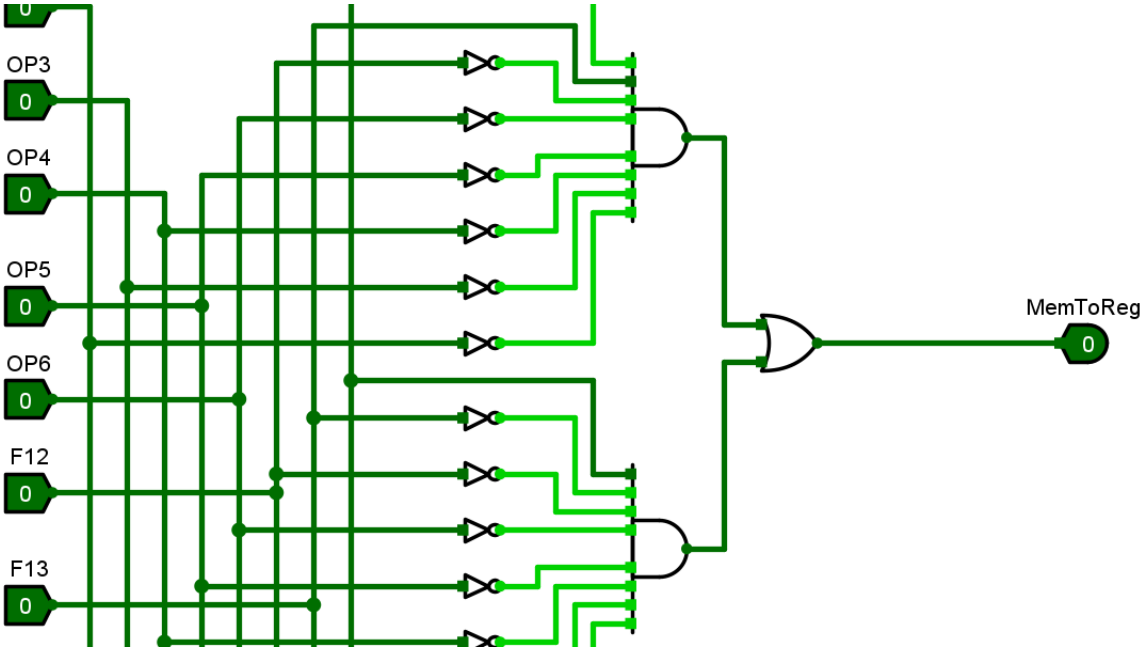

图 3.1.5 MenToReg 部分展示

3.2 流水 CPU 实现

3.2.1 流水接口部件实现

流水接口部件实现需要将 IF→ID→EX→MEM→WB 转换的具体数据保存一个周期并传递给下一个阶段。需要设计四个流水接口部件：IF/ID、ID/EX、EX/MEM、MEM/WB，由寄存器实现，采用同一个时钟周期进行保存，并且使用相同的复位信号。

那么这里给出例子如下图 3.2.1

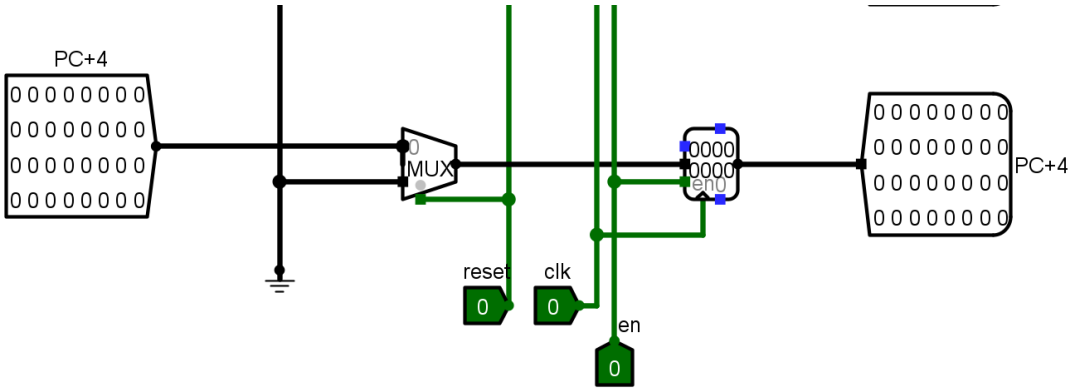

图 3.2.1 流水接口部件示意

可以看到：在这样一个部件中，有输入的部分，有输出的部分，通过一个寄存器建立一个周期内存储的需求，而且多寄存器公用时钟和使能信号，使得一个部件都可以正常工作，也同时使用用一个 reset 信号。让一个周期内输入为 0 从而能初始化。

通过这样多个输入输出接口，加上寄存器和共同使用的信号，我们就是实现了 IF/ID、ID/EX、EX/MEM、MEM/WB 这样的四个部件。

3.2.2 理想流水线实现

理想流水线不用考虑数据冲突等一系列问题，按 2.2.3 所阐述的原理将单周期 CPU 划分为五个阶段然后将这五个阶段需要的数据在流水接口部件中保存并向下传递即可，如图 3.2.2 所示。

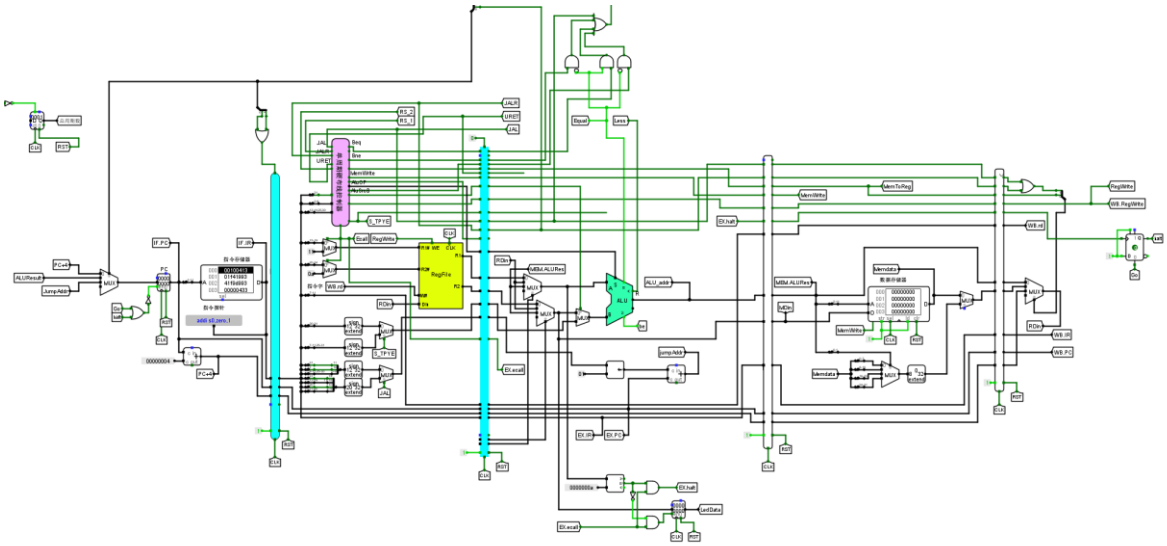

图 3.2.2 理想流水线示意图

3.3 气泡式流水线实现

3.3.1 冲突检测逻辑的实现

通过 2.3.1 的介绍，我们可以知道：冲突有结构冲突、控制冲突、数据冲突三种类型。而解决这几个问题：控制冲突需要增加清空信号将分支指令的指令清空，数据冲突需要增加气泡生成信号来增加气泡使得数据暂停。

那么我们急需要完成这个冲突检测逻辑，我们可以知道，气泡流水线就是在理想流水线的基础之上修改，我们可以根据当前电路的状态增加 DataHazzard 逻辑判断，输出 stall 信号和 flush 信号，具体来说，我们通过 EX、MEM 段是否有数据写入内存与 ID 段的指令所取的寄存器地址相匹配得到信号，具体实现如下图 3.3.1。

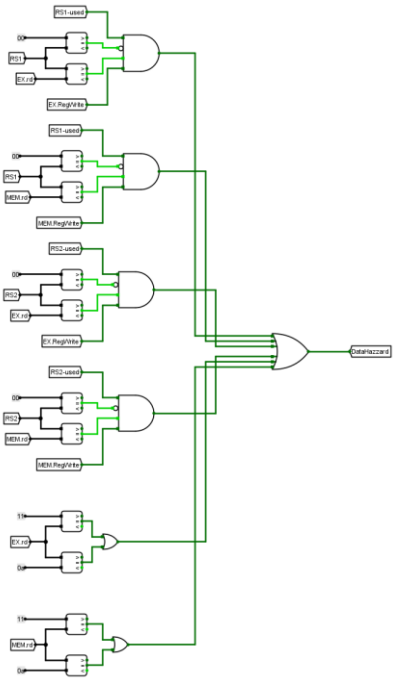

图 3.3.1 冲突处理逻辑

3.3.2 气泡流水线数据通路

正如前面冲突逻辑所言：数据冲突利用产生气泡来解决，分支需要通过清空前方指令来解决，因此需要增加分支检测和冲突检测并增加相关信号来将信息传递给流水接口部件使其保存正确的值。具体实现如下图 3.3.2

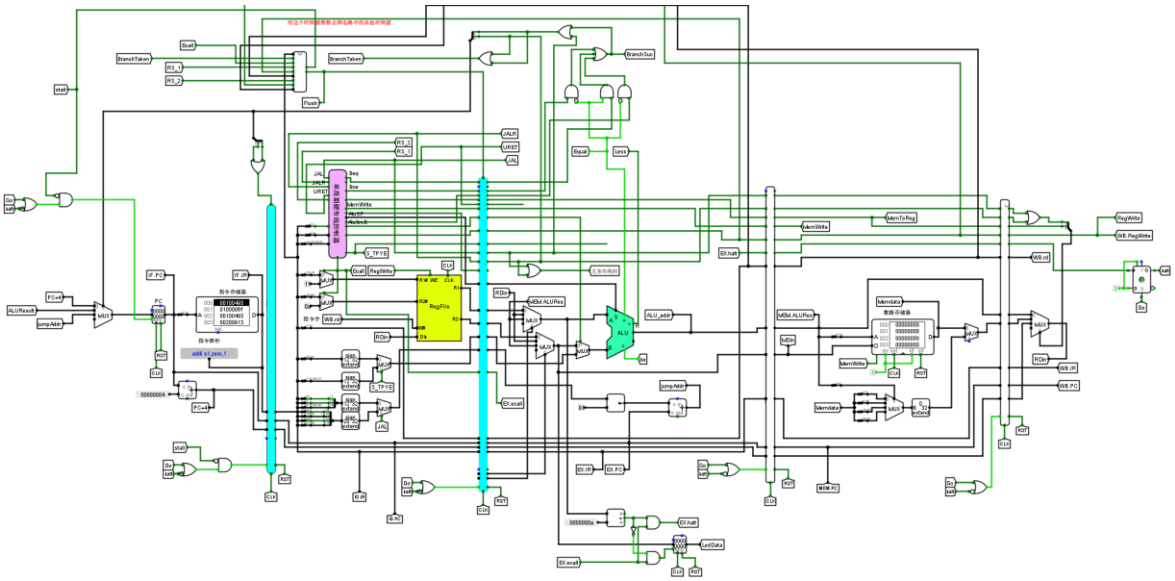

图 3.3.2 气泡流水线实现

3.4 重定向流水线实现

3.4.1 冲突检测逻辑

根据重定向不会直接在冲突时直接将前面的 ID 阶段的数据清空，而是正常的运行流水线，但在取值的时候，采用重定向将 EX 段取出的数据进行纠正的原理，增加一个和气泡流水线相似的冲突检测逻辑模块。

根据数据存储转发可以设计 RS1 和 RS2 的存储转发逻辑。

根据 Flush=BranchTaken|Load_Us 和 stall=Load_Use，就可以得到这四个信号用于后面重定向处理。

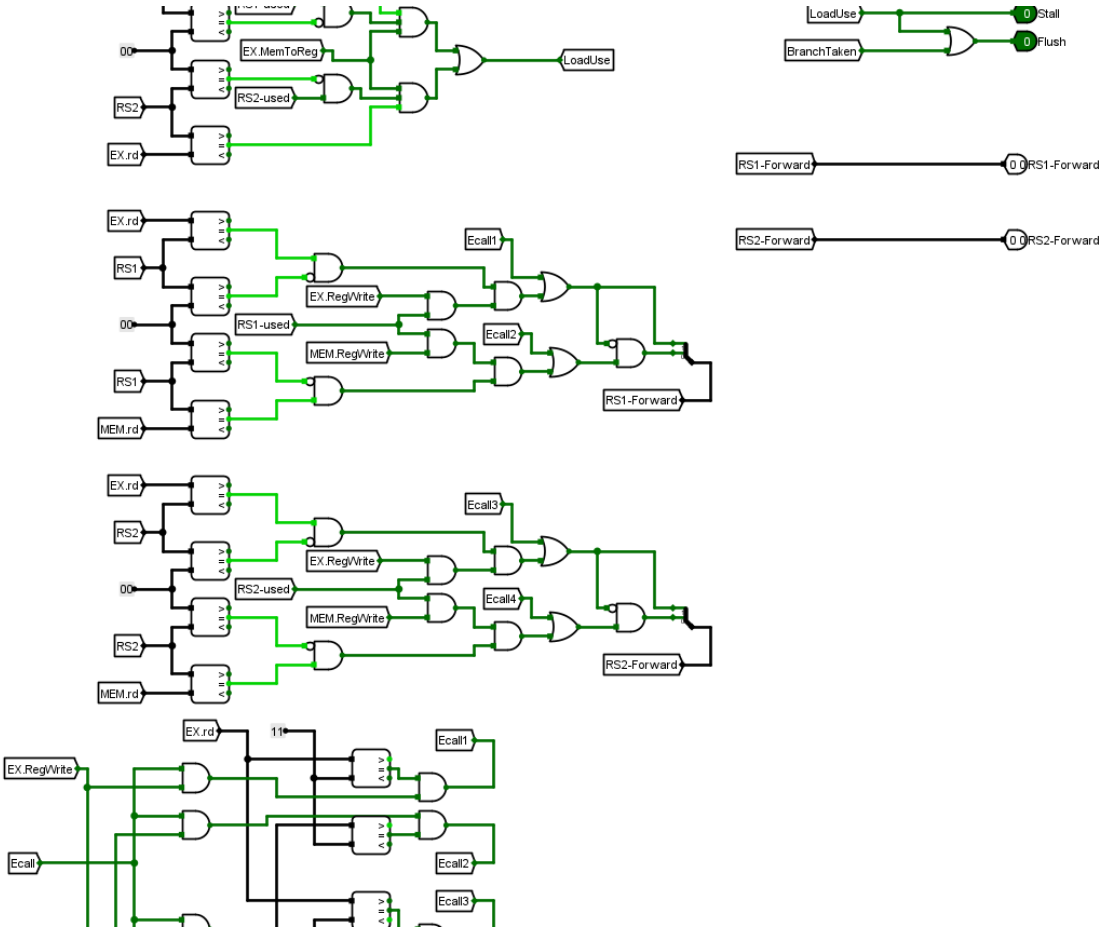

图 3.4.1 重定向处理逻辑

3.4.2 重定向流水线数据通路

对于生成的四个信号，分别是 RS1_Forward, RS2_Forward,stall 和 flush，我们可以知道重定向不会直接在冲突时直接将前面的 ID 阶段的数据清空，而是正常的运行

流水线，但在取值的时候，采用重定向将 EX 段取出的数据进行纠正的原理，那么我们需要在 ID/EX 阶段，处理我们的 flush 信号，清空我们的寄存器，此外，在处理 R1 和 R2 的时候，添加处理部分，增加多路选择器从而实现气泡处理，最后 stall 信号和 halt 控制 pc 寄存器的使能信号。最后电路如下图 3.4.2

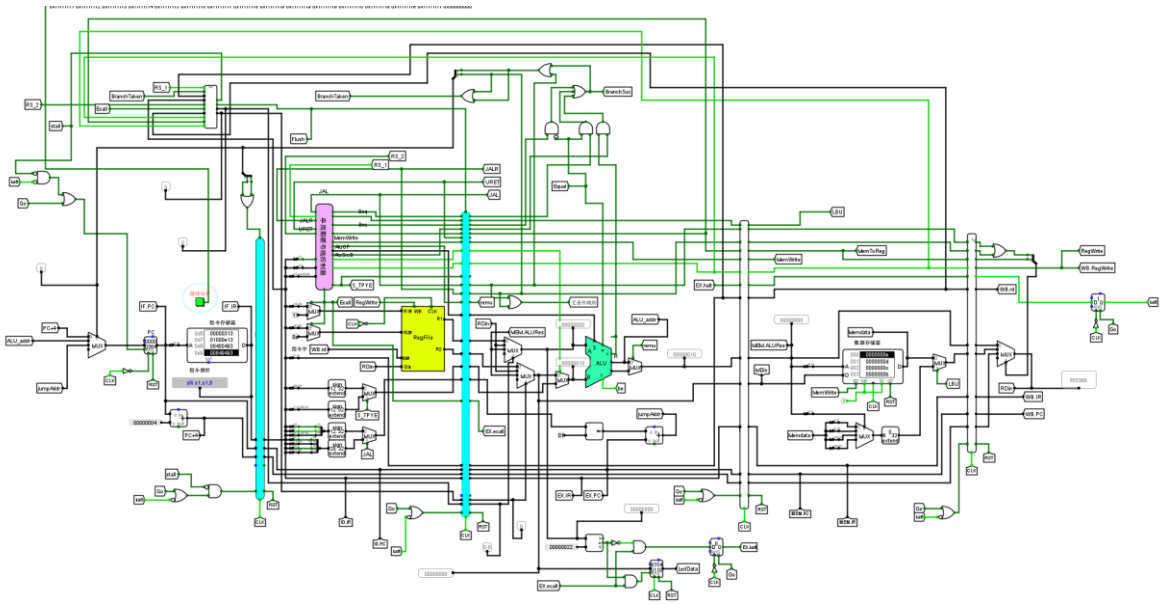

图 3.4.2 重定向实现

4 实验过程与调试

4.1 测试用例和功能测试

4.1.1 基础跑码测试

这里使用最快的重定向演示 benchmark.asm

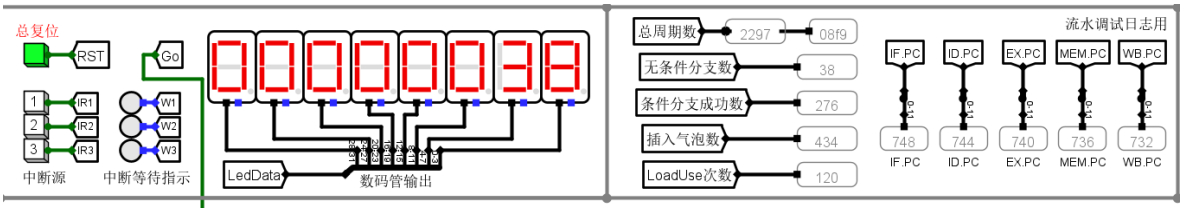

图 4.1.1 基础 bnechmark 示意图

4.1.2 CCAB 测试

这里使用 REMU 作为展示，其余指令都是动态变化难以展示，如图 4.1.2。

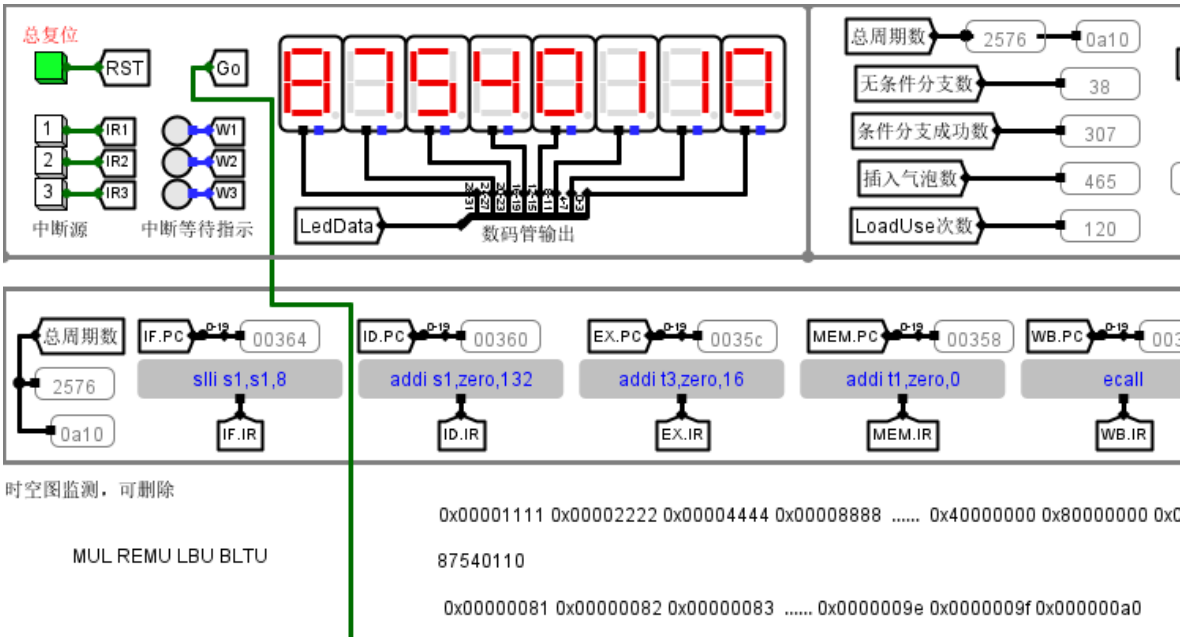

图 4.1.2 REMU 测试展示

4.2 性能分析

从实现角度来看，对比单周期，流水线的实现一定是更加快的，因为在处理单个

周期上虽然可能会多几个时钟周期，但是这样可以在每个阶段不断处理，显然是更加优化的方案。

气泡流水线在于冲突和分支都采用增加气泡的方式进行，因此需要增加气泡用掉的周期数最多，从而有了重定向解决了数据冲突问题，在寄存器取数冲突时可以不增加气泡，因此周期数相较于气泡流水线更少。比较可见表 4.1 流水线等性能比较

表 4.1 流水线等性能比较

	单周期 CPU	气泡流水线	重定向流水线
周期数	1545	3623	2297

4.3 主要故障与调试

4.3.1 CPU 运行不正确

故障现象：单周期 cpu 并没有完成预期的计算循环，在一段时间运行后直接进入 0 值指令寄存器从而没有反应。（此问题出现多次，以下给出几个原因）

原因分析：

1.并没有正确的链接顶层硬链接控制器，错误地将对应信号链接到了错误的信号输出接口上，从而导致输出的信号错位。

2.CCAB 表格上面的指令对应控制信号错误，从而输出的错误的控制信号，需要修改正确。

3.并没有成功实现气泡等流水线相关问题。

解决：

1.修改封装：将对应的值摆放好。

2.修改表格：对应指令手册和 RISC-V 官方文档，对照着书写表格。

3.检查流水线接口组件输入和输出接口是否是输入和输出不要搞反或者隧道名称写错导致气泡逻辑或者重定向逻辑出错（这些都发生了而且找不出来错误，使用一步一步单步调试对照室友的电路检查电路状态才发现控制信号有问题）

4.3.2 周期数不正确

故障现象：CPU 运行 benchmark 时的周期数不正确。

原因分析：主要是对于寄存器等需要定义好到底是上升沿触发还是下降沿触发的部件可能是错开设置导致毛刺的出现，我对电路的部件统一调整后一切都正常了。

4.3.3 跳转指令地址错误问题

故障现象：跳转的指令无法正确运行到指定地址

原因分析：跳转错误，就是在处理 PC 输入值的问题，在流水线中，我们需要考虑：以下几点 1.PC+4 一定要在 IF 阶段完成 2.跳转指令改变的 PC 要在 EX 阶段完成这样起码总是有指令跑在 cpu 上的不会空转

不要把 PC+4 放在最后而是在 IF 阶段直接处理，和单周期的处理不一样

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。完成部分 Logism 单周期 CPU 故障报告。
第三天	完成 Logism 单周期 CPU 的故障报告，并且通过了 Logism 单周期 CPU 的检查。使用 Verilog 实现了部分单周期 CPU 的重要部件，并通过仿真检查。
第四天	继续使用 Verilog 进行实现单周期 CPU 的工作，完成了所有部件的编写、控制器的编写，以及所有部件以及控制器的仿真测试，正在进行数据通路的拼接。
第五天	使用 Verilog 完成单周期 CPU 数据通路的连接，并且通过仿真测试。使用 Verilog 完成时钟分频以及七段数码管的代码编写，正在调试。
第六天	完成 CPU 电路的功能仿真和时序仿真，并成功将生成 bit 流烧入 FPGA 板内实现预计功能。
第七天	复习关于指令流水线的知识点，完成理想流水线的 verilog 代码，正在调试。
第八天	调试成功理想流水线 verilog 代码，并成功将 bit 流烧至 FPGA 板中。完成冒险处理中的数据冲突处理和分支处理代码编写，正在调试。
第九天	完成冒险处理中的数据冲突和分支处理，并成功烧入 FPGA 板内。完成数据

时间	进度
	重定向的 Verilog 代码的编写，正在调试。
第十天	完成数据重定向的 Verilog 代码并成功烧入 FPGA 板内。成功实现动态分支预测，预测成功率显著提高，并成功将代码烧入 FPGA 板内。

5 团队任务

5.1 团队选题与介绍

本次团队任务中，我们小组使用 logisim 软件，利用之前实验中已经完成的 risc-v 单周期 CPU 和添加的其他硬件电路，实现一个简单的推箱子小游戏。

推箱子游戏作为一款经典的智力挑战类游戏，其基本构成要素涵盖墙体、地面、玩家角色、箱子以及目标点。游戏的核心玩法在于玩家操控角色，通过上下左右移动指令，将箱子逐一推送至预设的目标点位置，以达成游戏胜利条件。

在本项目中，我们以 Logisim 软件为开发平台，利用其模拟出 RISC-V 单周期 CPU 作为游戏的运算控制核心。同时，选用 LCD 组件承担显示输出任务，能够现游戏的视觉场景，包括墙体的布局、地面的分布、玩家角色的实时位置、箱子的摆放状态以及目标点的精准标识。

在实现玩家角色移动控制方面，我们采用了中断机制作为输入手段。当玩家触发移动方向按钮时，系统即时生成中断信号，该信号迅速传递至 CPU。CPU 接收到中断信号后，依据预设的指令集精准解析并执行相应的移动操作，从而实现玩家角色在游戏场景中的移动。

5.2 设计与个人分工

5.2.1 软件设计

鉴于推箱子游戏所蕴含的复杂逻辑判断特性，我们团队在开发过程中选择了编写融合内联汇编的 C 语言代码的方案，再通过 RISC-V 32 位编译器与汇编器转变为机器码，并且 CODE 和 DATA 实现分离，这种分离式设计不仅使得代码结构更加清晰、易于维护与管理，而且极大地提升了游戏地图添加与更新的便捷性。当需要引入新的游戏地图时，我们只需在 DATA 段中进行相应的数据更新与配置，而无需对 CODE 段中的逻辑代码进行任何修改，从而实现了地图的轻松扩展与灵活定制，为游戏的持续迭代与丰富拓展提供了有力支撑。

软件设计首先需要处理该程序的逻辑功能，先从一次移动的流程分析具体的逻辑，接收到移动信号后，CPU 应关中断不再接受其他移动信号。然后计算这次移动是否可行，若不行则直接开中断；若可行，应判断新位置是否是箱子。若新位置不是箱子则

直接更新人物位置即可；若新位置是箱子，则继续判断箱子是否可以移动。若箱子不可移动，则同样直接开中断不更新；若箱子可以移动，则人物与箱子均更新，并判断游戏是否胜利，若未胜利则开中断游戏继续，否则游戏胜利并待机，具体流程图如下图 5.1 所示。故而软件设计具体要实现的函数经流程图即可知。

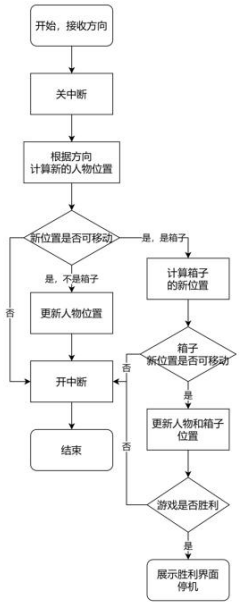

图 5.2.1 一次有效移动信号的逻辑流程图

5.2.2 硬件设计

由刚才的一次流程分析可知，至少要实现的硬件有开关中断，与存储各个像素点的信息的存储器，以及需要中断时屏蔽其他中断信号，具体分析可知要完成以下部分：

扩充 riscv 指令集：AUIPC, LUI, BLT, BGE。

更改 ecall 指令：当 a7 = 34 时，读取 s1, s2, s4 的值作为 LCD 屏幕的 y, x, rgb 值，并且为 LCD 屏幕提供使能信号。读取寄存器方式：额外设置三个寄存器，当对应寄存器写入时，同时向额外的寄存器写入，读取时读取这三个寄存器即可。

更改中断方式：中断状态时屏蔽中断信号，即在一个中断的进行过程中再次移动是完全无效的，不会导致接下来的移动。

增加硬件 DMA：原版 CPU 的 RAM 地址空间小，无法存储过多地图，而且每次都要加载镜像十分不便，所以额外设置一个 ROM 存放地图数据，每次选择关卡后通过 DMA 将 ROM 中对应关卡的地图数据写入到 RAM 中，速度十分快。

5.2.3 个人分工

我主要负责设计美术、地图，设计 DMA 实现地图数据加载，主要是硬件设计中的第四条以及地图设计

美术和地图设计不多做赘述，主要是设计问题既是个人审美也是主观判断，这里多做篇幅讲解一下“DMA”

这里将这个过程称作 DMA 其实不太好，因为 DMA 技术的核心思想是让外设直接访问系统内存，而不需要通过 CPU 进行数据中转，但是这里因为没有所谓‘cpu’的概念，因为这些数据都太小了，现代 cpu 缓存足够大，完全可以设计成和上世纪游戏机一样写死的固件实现。无论怎么样，我们就直接对这一部分叫‘DMA’了。

这里展示具体实现，如图 5.2.2

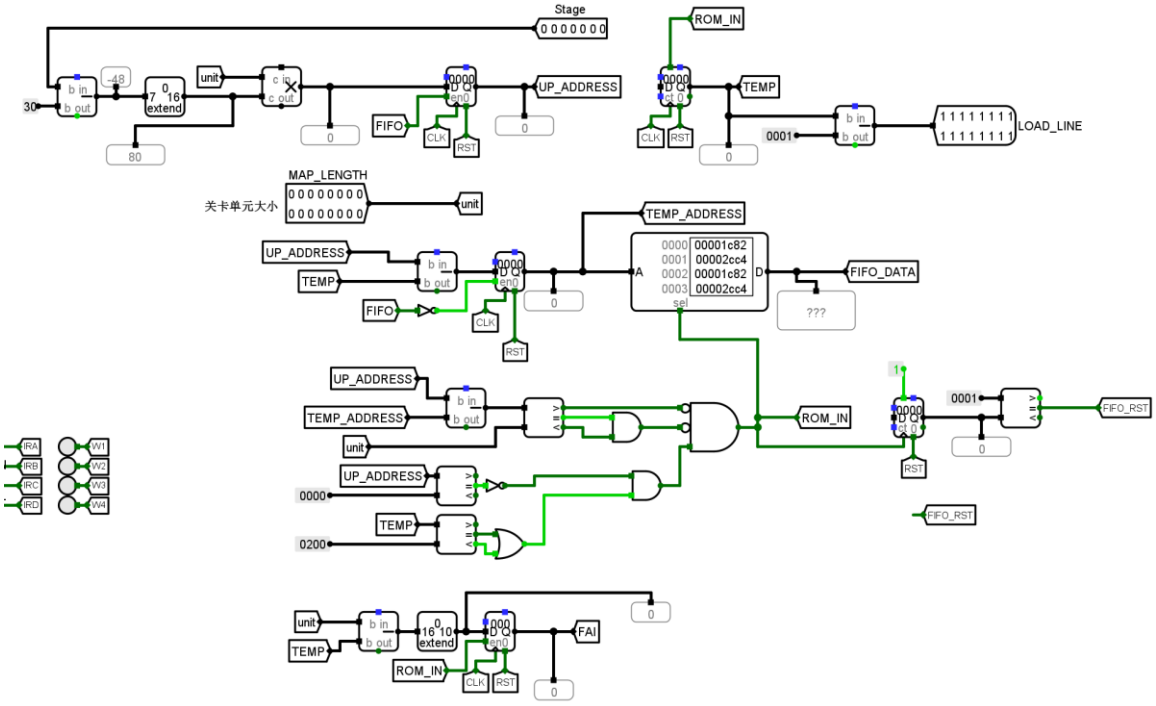

图 5.2.2 ‘DMA’示意图一

这里我重新设计了一个 rom 用来存储文件设计为 16 位地址空间，在外部调用中，输入的 keyborad 键值 Stage 为 ASCII 编码为 7 位，通过拓展得到我们的基址，再通过设定好的地图大小，做一次循环减即可。

在这里对于 RAM 的输入地址，我们采用每一关都加载到 RAM 中 0x0-0x??(输入的地图大小)也就是顶层空间的方式，正巧的时我们将需要使用的 temp 寄存器值处理一下正好适合当作 RAM 的输入地址。

这里在展示出对于原电路的修改，如图 5.2.3 ‘DMA’修改示意图二

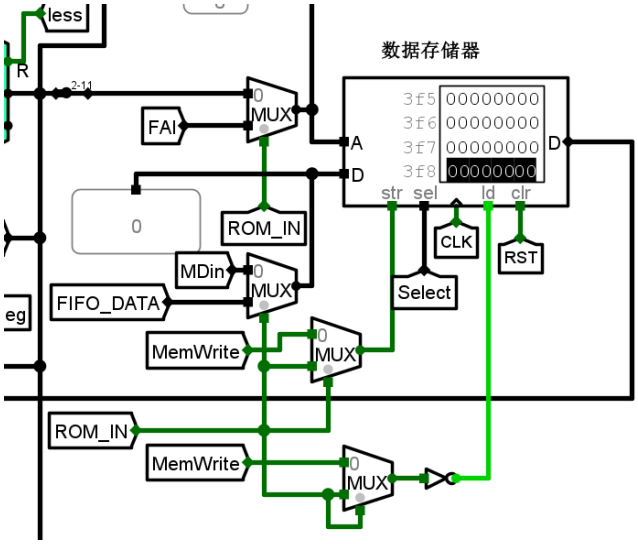

图 5.2.3 ‘DMA’修改示意图二

可以发现，我使用一个 ROM_IN 信号当作选择器输入来选择输入的地址，传入的数据也正是 ROM 中的数据，当停止读时，一切都和原电路的数据通路保持一致从而实现脱离。

这里有一个 bug：即实际上这里读取文件会多读一个字节，但是这并不影响实际游玩，因为我们实际的地图大小为并没有 200B 放置的时候也不会因为多读一个字从而出现问题，所以问题不大。

对于完成版本的修复：只需要使用 temp 隧道作为输入即可而不需要 FAI 隧道，这是调试的产物。

5.3 成果展示

游戏开始时 LCD 上没有任何东西，需要在 Keyboard 上输入数字选择关卡（例如 1、2），如图 5.3.1 所示。

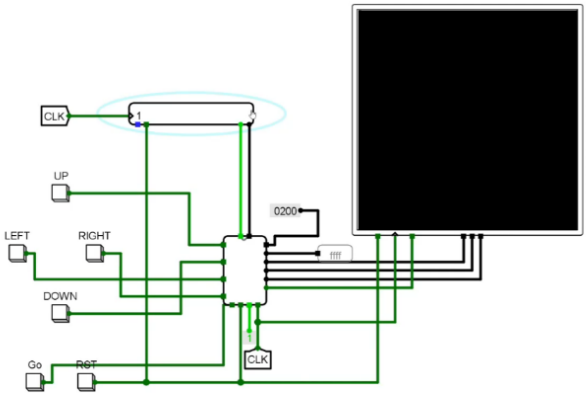

图 5.3.1 游戏开始

输入完成后启动电路仿真等待地图绘制完成如图 5.3.2 所示。

图 5.3.2 地图绘制

地图绘制完成即可点按左侧的移动按钮直至将所有的箱子推至目标点，此时游戏胜利，如下图 5.3.3 所示。若游戏死局需手动重启。

图 5.3.3 游戏胜利 win

6 设计总结与心得

6.1 课设总结

本次课程设计，设计了单周期 CPU、流水 CPU、气泡式流水线 CPU 和重定向流水线 CPU，并参与了团队任务。

在单周期 CPU 设计中，实现了哈佛结构与三级时序硬布线控制器，结合指令与数据存储器分离的方式以及程序计数器、指令存储器、运算器、寄存器堆和数据存储器等为核心部件的数据通路，并依据指令执行需求生成相应控制信号

在流水 CPU 设计中，实现了基于存储再转发的理念，将执行流程划分为 IF，ID，EX，MEM 和 WB 五个阶段，通过设计流水接口部件实现各阶段数据的周期传递与衔接。

在气泡式流水线 CPU 设计中，实现了针对结构、控制和数据冲突，分别采用哈佛结构存储分离、分支预测及插入气泡法的解决方法

在重定向流水线 CPU 设计中，实现了作为气泡处理的替代策略，在冲突时正常运行流水线，借助重定向对 EX 段数据进行校正，但对 load - use 相邻指令仍需采用气泡法处理。

最后我们完成了以上的设计和团队任务

6.2 课设心得

本次课程设计是迄今为止最具挑战性的实验，我并没有完成加分的目标却觉得这个任务我可以一个人完成是一个奇迹，从组原实验开始我就对设计部分并不熟悉，从完成单周期 CPU 的开心到完成重定向流水线的 benchmark 但是 CCAB 有问题的失落，我觉的我一步步能够实现重定向流水线，还是很有成就感的，既使我没有完成加分项目。

在设计的过程中室友给予了很大的帮助，我也要感谢我的室友，我和他们也一起参与了推箱子这个团队任务的设计。

在团队任务中我因为没有做中断，我去做了地图加载和设计的相关工作，虽然不是很核心的工作但是也是参与了团队任务，我们的推箱子游戏也是经历几个版本更新，从我录制演示视频突然临时找到 bug，到最后修改完成，也是很有趣的体验。

也许我上个学期并不是很理解所谓 cpu,所谓流水线，但是经过这一次课程设计，我

真切的理解了上个学期组原的内容，收获还是有的。

如果对于重定向和气泡流水线能够在组原的部分教学我感觉会更好，不然只能有两个方法：一是学习一下学长是如何做的，理解学长思路做自己的东西，二是查阅文章，但是读论文还是有难度的，但好在现在有很多 AI 工具，帮我读了不少相关的文章，还有国外社区的翻译和总结，在我设计的路上给了不少帮助。

其次就是感谢萝卜能在 1.10 号还能检查救我一条小命。（

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 5 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，周健，周游.计算机组成原理实验指导（基于 RISC-V 在线实训）. 北京：人民邮电出版社，2024 年.

[5] 曹强，施展.计算机系统结构（微课版）. 北京：人民邮电出版社，2024 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：瞿明睿