華中科技大學

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：	CS2003
学　　号：	U202015371
姓　　名：	范辰睿
电　　话：	18171082013
邮　　件：	1415913936@qq.com

目 录

1 课程设计概述

1.1 课设目的

计算机组成原理是计算机专业的核心基础课。该课程力图以“培养学生现代计算机系统设计能力”为目标，贯彻“强调软/硬件关联与协同、以 CPU 设计为核心/层次化系统设计的组织思路，有效地增强对学生的计算机系统设计与实现能力的培养”。课程设计是完成该课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有一定规模的指令系统的简单计算机系统。所设计的系统能在 LOGISIM 仿真平台和 FPGA 实验平台上正确运行，通过检查程序结果的正确性来判断所设计计算机系统正确性。

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

1.2 设计任务

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

1.3 设计要求

（1） 根据课程设计指导书的要求，制定出设计方案；

（2） 分析指令系统格式，指令系统功能。

（3） 根据指令系统构建基本功能部件，主要数据通路。

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

（5） 设计出实现指令功能的硬布线控制器；

（6） 调试、数据分析、验收检查；

（7） 课程设计报告和总结。

1.4 技术指标

（8） 支持表 1.1 前 27 条基本 32 位 MIPS 指令；

（9） 支持教师指定的 4 条扩展指令；

（10）支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（11）支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

（12）能运行由自己所设计的指令系统构成的一段测试程序，测试程序应能涵盖所有指令，程序执行功能正确。

（13）能运行教师提供的标准测试程序，并自动统计执行周期数

（14）能自动统计各类分支指令数目，如不同种类指令的条数、冒险冲突次数、插入气泡数目、load-use 冲突次数、动态分支预测流水线能自动统计预测成功与失败次数。

表 1.1 指令集

#	指令助记符	指令类型	简单功能描述	备注
1	ADD	R	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	I	立即数加	
3	AND	R	与	
4	ANDI	I	立即数与	
5	SLLI	I	逻辑左移	
6	SRAI	I	算数右移	
7	SRLI	I	逻辑右移	
8	SUB	R	减	
9	OR	R	或	
10	ORI	I	立即数或	
11	XORI	I	或非/立即数异或	
12	LW	I	加载字	
13	SW	S	存字	

#	指令助记符	指令类型	简单功能描述	备注
14	BEQ	B	相等跳转	
15	BNE	B	不相等跳转	
16	SLT	R	小于置数	
17	SLTI	I	小于立即数置数	
18	SLTU	R	小于无符号立即数置数	
19	JAL	J	转移并链接	
20	JALR	I	转移到指定寄存器	
21	ECALL	I	无条件转移	if($a7==34)LED 输出$a0 的值 else 等待 Go 按键暂停 注意显示逻辑需要考虑如何锁存过去的数据，否则数据一闪而过
22	CSRRSI	I	转移并链接	中断相关，可化简为开中断
23	CSRRCI	I	转移到指定寄存器	中断相关，可化简为关中断
24	URET	I	系统调用	清中断，mEPC 送 PC，开中断
25	XOR	R	异或	
26	LUI	U	高位立即数加载	
27	SB	S	存字节	
28	BGEU	B	无符号大于等于时分支	

2 总体方案设计

2.1 单周期 CPU 设计

本设计采用的方案是硬布线控制的哈佛结构。

硬布线控制器为组合电路，根据指令功能和译码表，通过指令译码器生成运算器操作码，通过控制信号生成器生成单周期内相应的控制信号。

由于单周期的存取要求，此设计结构将程序指令存储与数据存储分开，每个存储器独立编址、独立访问。存储结构的并行性有效避免了指令与数据间的访存冲突，能在单个周期内完成一条指令的执行。

总体结构图如图 2.1 所示。

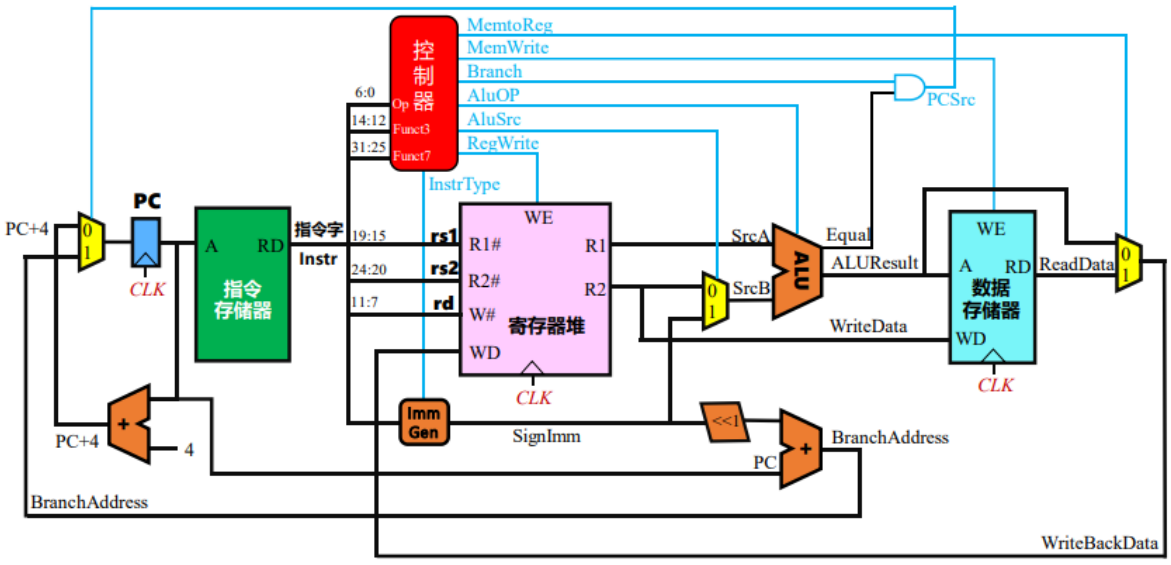

图 2.1 总体结构图

2.1.1 主要功能部件

设计的主要功能部件有程序计数器、指令存储器、寄存器堆、运算器、数据存储器与若干逻辑电路元器件。

1. 程序计数器 PC

程序计数器为一个 32 位上升沿触发的寄存器，输入数据为下一周期应当执行的程序入口地址，输出数据为本周期执行的指令地址，其 2-11 位流向指令寄存器，全部位数进入 PC+4 的加法器计算单元。其使能信号由 GO 信号与 halt 信号取反后做或操作得到，目的是使其遇到 halt 信号时停止更新执行地址从而使得 CPU 暂停运行，而遇到 GO 按钮信号时恢复运行。程序计数器的时钟信号与清零信号与电路的总时钟与总清零直接相连。

2. 指令存储器 IM

指令寄存器为一个地址位宽为 10 位，数据位宽为 32 为的只读寄存器，负责存贮程序指令并根据 PC 寄存器的输出数据读出当前周期所执行的指令数据。指令数据后续送入硬布线逻辑控制器进行信号译码，并使用分线器提取各种不同类型指令所需要的指令数据（如寄存器编号、立即数）。

3. 运算器 ALU

运算器由逻辑门、加法器、减法器、乘法器、除法器、移位器、多路选择器构成。其接受来自控制器的运算器功能码、来自寄存器堆或者符号拓展单元的操作数，输出操作数经过指定运算后的结果与操作数的比较信息。运算器的输入输出引脚与功能表述如表 2.1 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，具体功能见表 2.2
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
<	输出	1	X（有/无）符号小于 Y 时为 1，对（有/无）符号

引脚	输入/输出	位宽	功能描述
			比较操作有效
≥	输出	1	X **无符号**大于等于 Y 时为 1，对所有操作有效
Equal	输出	1	Equal=(x==y)?1:0，对所有操作有效

表 2. 2 算术运算逻辑单元功能码

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

寄存器堆由 regfile 器件（内部为$0-$31 共 32 个寄存器构成），在单周期设计下寄存器更新为上升沿触发，在流水线设计中为下降沿触发。

为解决单周期 CPU 读写冲突，寄存器堆栈能一次读出两个寄存器编号所对应寄存器内的值并同时根据写入编号更新一个寄存器的值；因此堆寄存器的读写操作能在一个周期内无冲突地完成。

寄存器堆的输入输出引脚及功能如表 2.3 所示

表 2.3 寄存器堆引脚与功能描述

引脚	输入/输出	位宽	功能描述
R1#	输入	5	寄存器 R1 的编号
R2#	输入	5	寄存器 R2 的编号
W#	输入	5	写入寄存器的编号
RDin	输入	32	写入寄存器的值
WE	输入	1	寄存器堆写使能信号
CLK	输入	1	时钟输入
*R*1	输出	32	寄存器编号 R1#对应寄存器内的值
R2	输出	32	寄存器编号 R2#对应寄存器内的值

5. 数据存储器 DM

为支持 SB（存字节）指令，数据存储器采用地址位宽为 10，数据位宽为 32 的 MIPS RAM，可以灵活支持向指定字的指定字节进行存储。

其地址输入连接 ALU 单元结果的第 2-11 位，数据输入来自 R2 与 SB 指令对应数据经过多路选择器后的输出结果，写使能信号来自硬布线控制器，位选择信号来自受字节地址和 SB 指令控制的逻辑单元。时钟端口直接与电路总时钟相连。

其输出结果作为 RDin 的备选结果之一。

2.1.2 数据通路的设计

表 2.4 指令系统数据通路表

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
ADD	PC+4	PC	rs1	rs2	rd	alu	r1	r2	5		
SUB	PC+4	PC	rs1	rs2	rd	alu	r1	r2	6		
AND	PC+4	PC	rs1	rs2	rd	alu	r1	r2	7		
OR	PC+4	PC	rs1	rs2	rd	alu	r1	r2	8		
SLT	PC+4	PC	rs1	rs2	rd	alu	r1	r2	11		
SLTU	PC+4	PC	rs1	rs2	rd	alu	r1	r2	12		

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
ADDI	PC+4	PC	rs1		rd	alu	r1	立即数	5		
ANDI	PC+4	PC	rs1		rd	alu	r1	立即数	7		
ORI	PC+4	PC	rs1		rd	alu	r1	立即数	8		
XORI	PC+4	PC	rs1		rd	alu	r1	立即数	9		
SLTI	PC+4	PC	rs1		rd	alu	r1	立即数	11		
SLLI	PC+4	PC	rs1		rd	alu	r1	立即数	0		
SRLI	PC+4	PC	rs1		rd	alu	r1	立即数	2		
SRAI	PC+4	PC	rs1		rd	alu	r1	立即数	1		
LW	PC+4	PC	rs1		rd	dout	r1		5	alu[11:2]	
SW	PC+4	PC	rs1	rs2			r1	立即数	5	alu[11:2]	r2
ECALL	PC+4	PC									
BEQ	PC+4/PC+SEXT(OFFSET)	PC	rs1	rs2			r1	r2			
BNE	PC+4/PC+SEXT(OFFSET)	PC	rs1	rs2			r1	r2			
JAL	PC+SEXT(OFFSET)	PC			rd	pc+4					
JALR	(R1+SEXT(OFFSET))&~1	PC			rd	pc+4					
CSRRSI	PC+4	PC									
CSRRCI	PC+4	PC									
URET	EPC	PC									
XOR	PC+4	PC	rs1	rs2	rd	alu	r1	r2	9		
LUI	PC+4	PC				ui*					
SB	PC+4	PC	rs1	rs2					5		sbdata*
BGEU	PC+4/PC+SEXT(OFFSET)	PC	rs1	rs2			r1	r2			

注：ui = sext(immediate[31:12] << 12), sbdata = sext(r2[7:0])<<(byte*8)

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.。

表 2.5 主控制器控制信号的作用说明

控制信号	取值	说明
RegWrite	0/1	寄存器写使能
MemWrite	0/1	写内存控制信号
AluOP	0-12	运算器操作控制符，详见表 2.2
MemToReg	0/1	寄存器写入数据来自存储器
S_Type	0/1	S 型指令译码信号
ALUSrcB	0/1	运算器 B 输入选择（0 表示 R2，1 表示立即数）
JALR	0/1	JALR 译码信号
JAL	0/1	JAL 译码信号
Beq	0/1	Beq 译码信号，用于有条件分支控制
Bne	0/1	Bne 译码信号，用于有条件分支控制
ecall	0/1	ecall 译码信号，根据 a7 寄存器的值，决定停机或者输出
uret	0/1	uret 译码信号
lui	0/1	lui 译码信号
sb	0/1	sb 译码信号
Bgeu	0/1	Bgeu 译码信号，用于有条件分支控制
CSRRSI	0/1	CSRRSI 译码信号，用于多级中断处理中的开中断操作
CSRRCI	0/1	CSRRCI 译码信号，用于多级中断处理中的关中断操作

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.所示。

表 2.6 主控制器控制信号框架

#	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	URET	LUI	SB	BGEU	CSRRSI	CSRRCI	CSRRW
1	add	0	0	c	5				1													
2	sub	32	0	c	6				1													
3	and	0	7	c	7				1													
4	or	0	6	c	8				1													
5	slt	0	2	c	11				1													
6	sltu	0	3	c	12				1													
7	addi		0	4	5			1	1													
8	andi		7	4	7			1	1													
9	ori		6	4	8			1	1													
10	xori		4	4	9			1	1													
11	slti		2	4	11			1	1													
12	slli	0	1	4	0			1	1													
13	srli	0	5	4	2			1	1													
14	srai	32	5	4	1			1	1													
15	lw		2	0	5	1		1	1													
16	sw		2	8	5		1	1			1											
17	ecall		0	1c						1												
18	beq		0	18								1										
19	bne		1	18									1									
20	jal			1b					1					1								
21	jalr		0	19					1						1							
22	CSRRSI		6	1c	8			1	1											1		
23	CSRRCI		7	1c	7			1	1												1	
24	URET		0	1c												1						
25	XOR	0	4	c	9				1													
26	LUI			d					1								1					
27	SB		0	8	5		1	1			1							1				
28	BGEU		7	18															1			
29	CSRRW																					1

2.2 中断机制设计

2.2.1 总体设计

本设计中与中断相关的部分分为单周期单级中断、单周期多级中断与流水线单级中断。中断信号共有 3 种，均由按键触发，其优先级随中断序号依次递增。

中断按键触发时，对应的中断指示灯亮起；当其被解决，中断信号清零时指示灯熄灭。当处理第 i 个中断时，数码为 i 的走马灯从右向左依次左移三轮，数码管其他数字为右移的轮数。

处理单级中断时，遇到中断信号的第一个周期内系统通过设置 IE 寄存器关中断，EPC 寄存器寄存下一时刻的 PC，多路选择器根据中断号以及预先设定的中断程序处理地址得出下一周期地址并送入 PC 寄存器，CPU 进入中断处理程序。当中断处理程序执行至 URET 指令时，CPU 通过将 EPC 寄存器中的值送入 PC 恢复现场并开中断。

处理多级中断时，通过比较器完成新中断与正在处理中的中断的优先级比较决定是否打断目前执行的中断处理程序，设置多个 EPC 寄存器以保护嵌套中的现场，并使用 CSRRSI 与 CSRRCI 指令避免该周期被抢占，满足 CPU 对嵌套中断的支持。

2.2.2 硬件设计

中断机制相关硬件模块分为中断按键电路、中断号生成电路、中断清零信号产生电路、中断使能逻辑电路与 EPC 寄存器控制电路。相关电路元器件为逻辑门、触发器、多路选择其和解复用器。另外，多级中断设计中另设有硬件堆栈以满足中断的嵌套与返回功能。

当按钮信号触发时，应使用触发器将中断信号锁存，对应中断指示灯亮起，直至清零信号到来将触发器状态重置，中断指示灯熄灭。

处理单级中断时，不同中断信号经过优先编码器得出此时优先级最高的中断信号并以此为多路选择器选择信号将预先设定的中断程序处理地址选送入 PC 寄存器；同时中断信号应改变中断使能寄存器 IE 的值，在 URET 指令执行之前关闭中断响应，完整地执行该中断程序；同时将下一时刻 PC 值送入 EPC 寄存器以保护现场。

当执行 URET 指令时改变 IE 寄存器的值打开中断响应，同时根据此时的中断处理程序号生成响应的中断清零信号；同时将 EPC 寄存器中的值送入 PC 以恢复现场。

处理多级中断的硬件设计与单级中断不同，为满足对嵌套程序现场的保护，设有 3 个 EPC 寄存器以及 3 个中断信号寄存器。中断按键电路与单级中断相同，中断使能寄存器可以被 CSRRSI 与 CSRRCI 信号控制进行开中断与关中断操作，从而可以实现高优先级中断打断低优先级中断处理的功能。电路设有比较器比较当前中断与当前正在处理的中断的优先级，若新触发的中断信号的优先级较高，在中断使能信号允许的条件下将进入高级中断的处理程序。在这程中原中断对应的 EPC 寄存器完成对 PC 寄存器现场的保护，新中断信号将被对应的中断信号寄存器锁存，直至对应清零信号产生。

2.2.3 软件设计

中断机制相关指令为 URET、CSRRSI、CSRRCI 指令，前者用于中断的返回与主程序运行的恢复，后两者用于多级中断中开关中断操作。

当中断处理程序功能语句执行完毕时，程序将执行 URET 语句返回中断触发时的主程序。此时系统需要恢复中断触发时的状态现场，沿原 PC 继续执行程序。URET 即中断返回指令。

CSRRSI 与 CSRRCI 为支持多级中断的开中断、关中断语句。由于在多级嵌套中断中中断处理程序可以被更高级别的中断所打断，但是其保护现场与恢复现场的过程无法被打断，因此设计开中断与关中断语句维护这一过程的执行。

2.3 流水 CPU 设计

2.3.1 总体设计

流水线技术能极大地提升 CPU 运行速度，提高工作效率；本设计采用五段流水线设计，分别是实现了理想流水线、气泡流水线与重定向流水线三种流水线机制。

五段流水线设计将单条指令的执行分为取指令（ID），指令译码（IF），指令执行（EX），存储修改（MEM）与寄存器堆写回（WB）五个阶段，在流水线被充满时能同时执行五条指令，从而在不提升时钟频率的情况下提高了指令执行的效率。

然而，指令间的非线性执行与读写依赖问题严重影响了 CPU 运行的高效性与正确性，使得理想流水线只能执行相对简单受限的指令；因此本设计采用气泡流水线与重定向流水线两种机制解决指令间的数据依赖关系，后续通过动态分支预测技术进一步提高流水线执行效率。

2.3.2 流水接口部件设计

流水接口的设计目的为在流水线的不同阶段传递一条指令的控制信号与数据信息，从而实现单条指令的分阶段执行目标；同时为避免数据冲突与分支时预取指令的清空，流水线接口部件还应支持对寄存数据的阻塞与清空操作。

流水接口部件主要由逻辑门、寄存器与多路选择器构成，并根据相邻指令执行步骤间需要传递的信号与其位宽设计输入与输出。设计通过多路选择器来实现流水寄存器的同步清零操作。

2.3.3 理想流水线设计

理想流水线 CPU 将单周期 CPU 执行指令的不同阶段用流水接口部件隔开，从而实现了多条指令同时执行的并行性。由于向寄存器堆的写回操作发生在 WB 段，而寄存器堆的物理位置处于 IF 段，因此有必要将写回操作的写入寄存器编号随流水线传输，当写入操作发生时再流向寄存器堆。

理想流水线不考虑程序流水执行时的存取冲突以及分支跳转预取指令的清空问题，因此只能执行简单的顺序程序，并且相邻指令间不能引用相同的寄存器。

2.4 气泡式流水线设计

气泡式流水线在理想流水线的基础上增添了气泡逻辑控制单元，能有效解决数据冲突与分支跳转时预取指令清空的影响 CPU 执行程序正确性的问题。

当检测到 ID 段指令将要使用的寄存器编号与 EX 段或 MEM 段指令所要向寄存器堆栈访问或写入的寄存器编号相冲突时，程序将 ID/IF 流水接口的数据阻塞并将 ID/EX 端口的数据清零，待前序指令执行至写回阶段后（同时应将寄存器堆栈设置为下降沿触发）再继续执行原被阻塞在 ID/IF 端口的指令。

当 EX 段指令为无条件分支或被触发的有条件分支时，CPU 应执行的下一条指令应当是对应分支地址的指令而非预取在 IF/ID 段的顺序指令。因此，气泡逻辑控制器应当对 IF/ID 流水端口进行清空操作。

注意，如果 EX 段为成功触发的分支语句，则 IF 段指令失效，因此数据冲突检测不再成立；故当两者同时存在时分支指令对应的清空操作具有更高的优先级，即应当对 IF/ID 流水端口进行清空而非阻塞操作。

2.5 重定向流水线设计

气泡流水线设计能够满足 CPU 执行指令的正确性要求，但是频繁地向流水线中插入气泡会较大地影响处理器效率；重定向流水线通过转发流水线中的部分值来规避部分数据冲突，从而减少插入气泡的个数，提升流水效率。

若不考虑数据冲突，则 EX 段使用的 R1 与 R2 可能发生错误，原因是除从 IF 段取出的寄存器值外，可能来自 MEM 段的 ALU 计算结果、RAM 访存结果与 WB 段的写回结果。为了修正这一点，我们可以将 MEM 段的 ALU 计算结果与 WB 段的写回结果作为 EX 段寄存器引用的备选，并通过重定向逻辑控制多路选择器引用正确的寄存器值。

这里我们不将 MEM 段 RAM 的访存结果重定向而是插入气泡，原因是对访存结果重定向会增加流水线关键延迟。若执行该写汇操作，则 EX 段的计算必须等待 MEM 执行完毕后开始，将电路关键延迟增加为 EX+MEM 两端流水线运行所需时间，大大降低了流水频率，影响处理器效率。

综上，重定向流水线的控制逻辑为：当检测到 ID 段访问寄存器与 EX 段、MEM

段 regwrite 信号写入寄存器相同时将对应的 ALU 结果、RDin 结果重定向写入 ID/EX 流水接口并设置相应的多路选择器信号；当检测到 loaduse（ID 段访问寄存器与 EX 段访存指令所用寄存器冲突）时阻塞 IF/ID 流水接口，加入气泡。

重定向流水线对于分支指令的处理办法与气泡流水线相同。

2.6 动态分支预测机制

重定向流水线设计在气泡流水线的基础上优化了对于始数据冲突的处理，但是对于分支指令的处理仍略显低效：只有分支指令运行至 EX 段时才能决定是否跳转，若跳转则需要清空 IF 与 ID 段的预取指令。在实际程序的执行中，由于涉及循环条件判断等时空相关反复执行的指令，根据 PC 可以一定程度上在 IF 阶段预测程序是否跳转，从而降低预取指令被清空的概率，提高处理器运行效率。

动态分支预测 BTB 模块由一个八槽全连接 cache 和若干逻辑电路元件组成。程序运行时，BTB 使用 EX 段分支指令的 PC、跳转历史信息与跳转地址数据装填 cache；若 IF 段指令命中 cache 中的 PC，则其跳转与否与跳转地址信息可以通过 cache 行中对应数据给出。将预测结果随流水线传输，传输至 EX 段后与其真实跳转结果进行对比，若预测失败则清空 IF 与 ID 段流水数据。

动态分支预测的历史信息状态转移图由图 2.2 所示，当 EX 段命中 cache 时根据此时的跳转情况更新状态转移图中的状态；当 IF 段命中 cache 时根据记录的历史状态做出跳转与否的预测。本设计中使用 10 作为状态的初值。

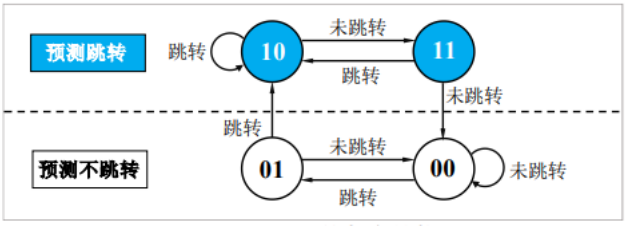

图 2.2 动态分支预测状态转移图

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）

① Logism 实现：

使用一个 32 位寄存器实现程序计数器 PC，触发方式为上升沿触发，输入为下一条将要执行的指令的地址，由多路选择器从备选地址中选出；输出为当前执行指令的地址。

Halt 为停机信号，通过非门后为 1，再与 Go 信号做或操作后作为 PC 寄存器的使能信号。当 Halt 信号为 1 且无 Go 信号时屏蔽时钟信号，使整个电路停机；停机时按压 Go 按钮电路又能重新运行。寄存器的时钟信号与清零信号与总电路的时钟与复位信号相连。具体设计如图 3.1 所示。

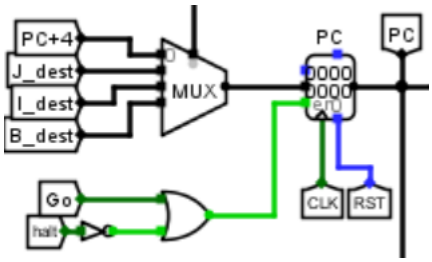

图 3.1 程序计数器（PC）

② FPGA 实现：

程序计数器 PC 的 Verilog 代码如下：

```
module register(Din,en,clk,rst,Dout);
input[31:0]Din;
input en,clk,rst;
output reg[31:0]Dout;
initial begin
    Dout=0;
end
```

```
    always@(posedge clk,posedge rst)begin
        if(rst)
            Dout<=0;
        else if(en)begin
            Dout<=Din;
        end
    end
endmodule
```

2) 指令存储器（IM）

① Logism 实现：

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：

建立 32 位，长度为 2^10-1 的寄存器变量数组用于存储指令数据，并通过 $readmemh 指令从相应的文件中加载程序数据初始化指令存储数组，并根据地址 A 在数组中寻址并输出对应的指令数据。

指令存储器 IM 的 Verilog 代码如下：

```
module inst_rom(Addr,Dout);
    input[9:0] Addr;
    output[31:0] Dout;
    reg[31:0] rom[2**10-1:0];
    initial begin
        $readmemh("D:/work/hardware_training/package/cpu21-riscv-
```

```
fpga/hardwire_controller/mips_cpu/ccab.txt",rom);
        end
        assign Dout=rom[Addr];
    endmodule
```

3) 运算器（ALU）

① Logsim 实现

使用若干逻辑门、比较器与多路选择器实现根据运算操作码对两输入不同操作后的输出。运算器的输入为 X,Y 两操作数以及运算器操作码，输出为运算结果以及相等、（有/无）符号小于、无符号大于等于标志位。如图 3.3 所示

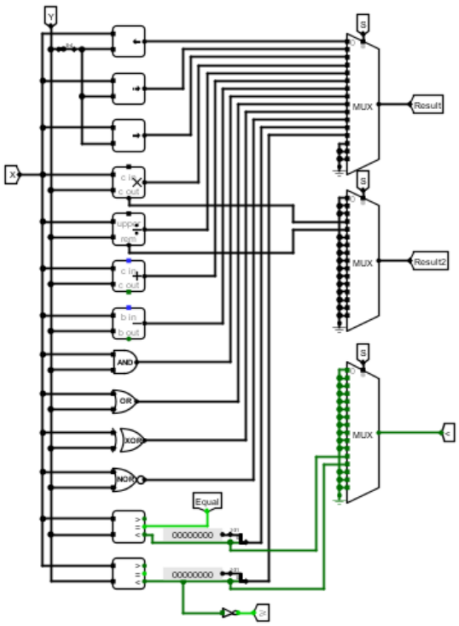

图 3.3 指令存储器（IM）

② FPGA 实现：

预设 X、Y 均为有符号数，使用 case 语句输出对应操作码下的输出结果与比较标志位；当遇到无符号操作与比较时使用$unsigned 表达式对变量进行强制类型转化，从而实现对应功能。

```
`timescale 1ns / 1ps
module ALU(X,Y,S,Result,Result2,Equal,Geq,Leq);
    input signed[31:0]X,Y;
    input[3:0]S;
    output reg[31:0]Result,Result2;
```

```
output reg Leq;
output Equal,Geq;
assign Geq=($unsigned(X)>=$unsigned(Y))?1:0;
assign Equal=(X==Y)?1:0;
initial begin
    Result=0;
    Result2=0;
    Leq=0;
end
always @(X,Y,S)begin
    case(S)
        0:begin
            Result=X<<(Y[4:0]);
            Result2=0;
            Leq=0;
        end
        1:begin
            Result=X>>>(Y[4:0]);
            Result2=0;
            Leq=0;
        end
        2:begin
            Result=X>>(Y[4:0]);
            Result2=0;
            Leq=0;
        end
        3:begin
            {Result2,Result}=X*Y;
            Leq=0;
        end
```

```
4:begin
    Result=X/Y;
    Result2=X%Y;
    Leq=0;
end
5:begin
    Result=X+Y;
    Result2=0;
    Leq=0;
end
6:begin
    Result=X-Y;
    Result2=0;
    Leq=0;
end
7:begin
    Result=X&Y;
    Result2=0;
    Leq=0;
end
8:begin
    Result=X|Y;
    Result2=0;
    Leq=0;
end
9:begin
    Result=X^Y;
    Result2=0;
    Leq=0;
end
```

```
                10:begin
                    Result=^(X|Y);
                    Result2=0;
                    Leq=0;
                end
                11:begin
                    Result={31'b0,{X<Y}};
                    Result2=0;
                    Leq=(X<Y)?1:0;
                end
                12:begin
                    Result={31'b0,{$unsigned(X)<$unsigned(Y)}};
                    Result2=0;
                    Leq=($unsigned(X)<$unsigned(Y))?1:0;
                end
                default:begin
                    Result=0;
                    Result2=0;
                    Leq=0;
                end
            endcase
        end
    endmodule
```

4) 寄存器堆（RF）

① Logsim 实现

寄存器堆直接使用 CS3410 库中 regfile 元件，其中 W 连接寄存器写入数据，向 W#对应寄存器中写入数据；R1、R2 分别输出对应编号的寄存器数据。RF 的时钟信号与写使能信号与电路的总时钟与指令译码器产生的寄存器写使能信号相连。

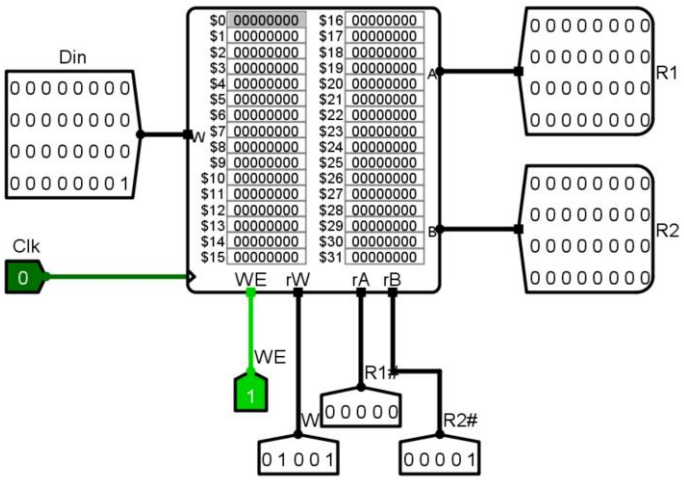

图 3.4 寄存器堆（RF）

② FPGA 实现：

寄存器堆的 FPGA 实现由 32 个长度为 21 比特的寄存器数组组成，其初始值均为 0，输出值 R1,R2 为对应编号寻址后的结果。时钟上升沿时当检测到 WE 信号且写入地址不为 0 时对相应的寄存器进行赋值操作。

```
module regfile(R1Addr,R2Addr,WAddr,Din,clk,WE,R1,R2);
    input[4:0]R1Addr,R2Addr,WAddr;
    input[31:0] Din;
    input clk,WE;
    output [31:0]R1,R2;
    reg[31:0] ram[31:0];
    initial begin
        ram[0]=32'b0;
        ……
        ram[31]=32'b0;
    end
    assign R1=ram[R1Addr];
    assign R2=ram[R2Addr];
    always@(posedge clk)begin
        if(WE&&(WAddr!=0))begin
            ram[WAddr]=Din;
```

```
            end
        end
    endmodule
```

5) 数据存储器（DM）

① Logism 实现：

数据存储器由 MIPS RAM 与若干逻辑元件构成，其 sel 接口可以实现对指定字的指定字节进行写入。为实现 SB 指令对单字节的写入功能，取地址最后两比特位对 R2 的右移位数与 sel 接口进行逻辑控制，而从灵活实现单字节访存。

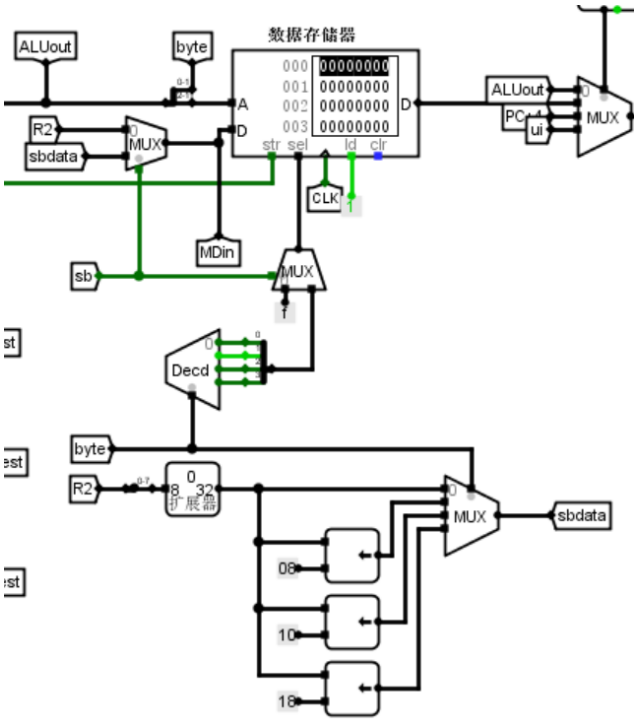

图 3.5 数据存储器（DM）

② FPGA 实现：

相比上述实现，Verilog 语句能更加灵活地实现单字节访存功能；设置访存字节输入，data_ram 通过地址的数据合并操作可以灵活地实现对单字节的存储与访问功能。

数据存储器 DM 的 Verilog 代码如下：

```
module data_ram(Addr,MDin,str,sel,clk,clr,Dout);
    input[9:0] Addr;
    input[31:0] MDin;
    input str,clk,clr;
    input[3:0] sel;
```

```
    output [31:0] Dout;
    reg[7:0] ram[2**12-1:0];
    integer i;
    initial begin
        for(i=0;i<2**12;i=i+1)begin
            ram[i]=8'b0;
        end
    end
    assign Dout[7:0]=ram[{Addr,2'b00}];
    assign Dout[15:8]=ram[{Addr,2'b01}];
    assign Dout[23:16]=ram[{Addr,2'b10}];
    assign Dout[31:24]=ram[{Addr,2'b11}];
    always@(posedge clk,posedge clr)begin
        if(clr)begin
        end
        else if(str)begin
            if(sel[3])
                ram[{Addr,2'b11}]<=MDin[31:24];
            if(sel[2])
                ram[{Addr,2'b10}]<=MDin[23:16];
             if(sel[1])
                ram[{Addr,2'b01}]<=MDin[15:8];
             if(sel[0])
                ram[{Addr,2'b00}]<=MDin[7:0];
        end
    end
endmodule
```

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**错误!未找到引用源。**所示。

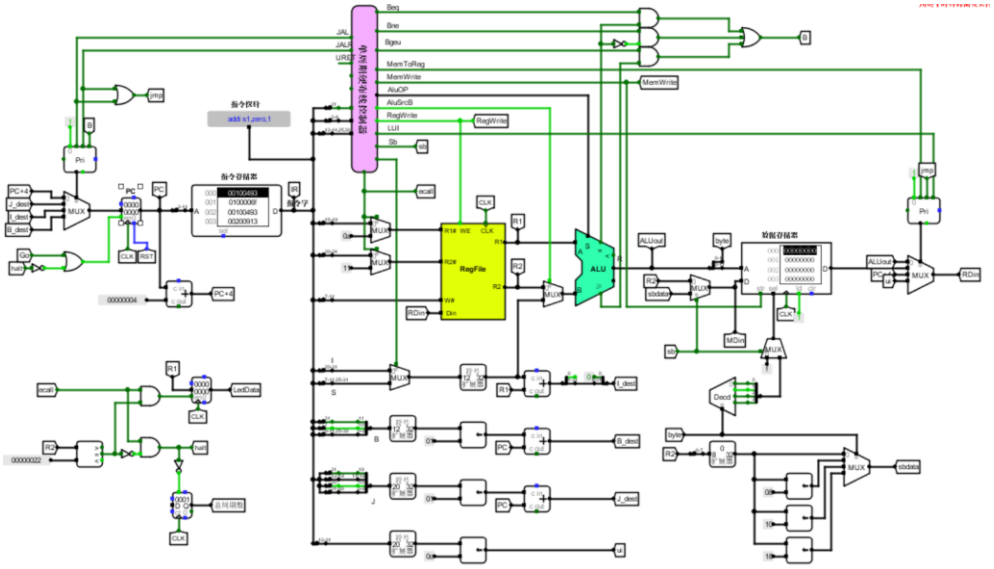

图 3.3 单周期 CPU 数据通路（Logism）

在 Vivado 中使用 Verilog 语言搭建的数据通路的原理图如图 3.4 所示。

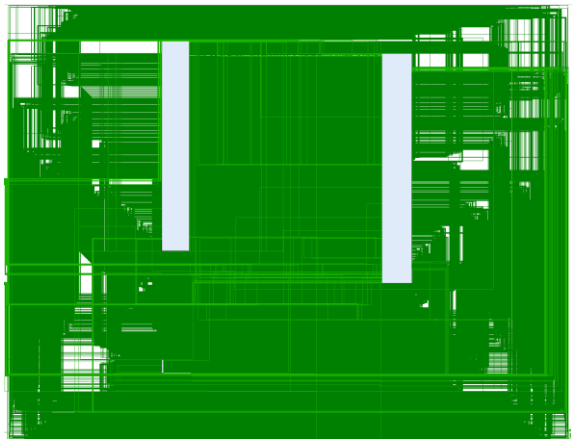

图 3.4 单周期 CPU 数据通路（FPGA）

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。

主控制器信号生成逻辑对照表 2.6。

将表 2.6 中的控制信号生成逻辑输入 Logism 中自动逻辑电路生成器便得到了其 Logism 实现。在此基础上使用 Logism 至 Verilog 程序转换器，生成对应的 Verilog 代码及以其封装的电路元件。

3.2 中断机制实现

3.2.1 单周期+单级中断

依照 2.2 中的软硬件设计逻辑设计中断按键电路、中断使能寄存器 IE、现场保护寄存器 EPC，并通过多路选择器与解复用器完成中断地址的选择与清零，最终设计如图 3.5 所示。

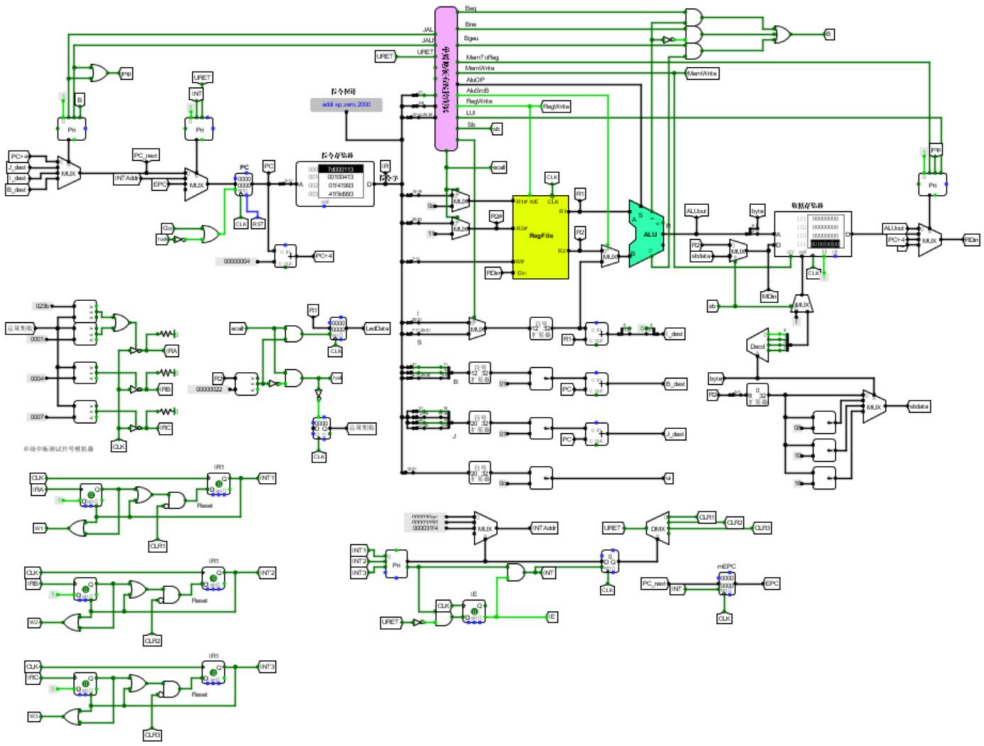

图 3.5 单周期+单级中断电路

3.2.2 单周期+多级中断

依照 2.2 节中的多级中断设计逻辑设计中断按键电路、中断使能寄存器、3 个 EPC 与 INT 寄存器模拟的中断硬件堆栈。另设优先编码器求解当前时刻优先级别最高的中断并与当前时刻运行的中断程序优先级进行比较，若大于当前优先级且 IE 寄存器处于开中断状态则更新中断 INT 寄存器与当前中断，保存当前 PC 现场进入更高优先级的中断。为实现保存、恢复现场等程序过程不被打断，设有开关中断信号辅助 IE 寄存器锁存状态。最终设计的 PC 相关部分与图 3.5 中相同，多级中断的逻辑处理电路如图 3.6 所示。

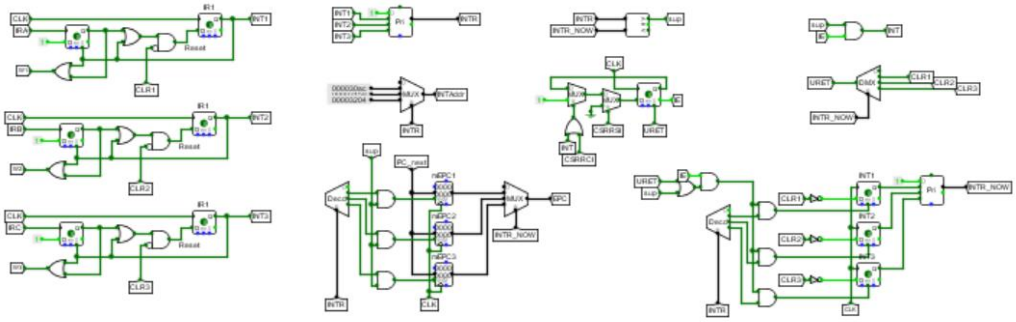

图 3.5 单周期+多级中断逻辑模块

3.2.3 重定向流水线+单级中断

重定向流水线的单级中断设计与单周期处理器的单级中断处理非常类似，在设计时选择将 EX 段的程序执行完毕，因此只需要将 EPC 保存的 PC 值替换为 EX 段指令的下一条指令（分支条件为选择信号从 EX.PC+4 等备选指令地址中选择）。中断模块如图 3.6 所示。

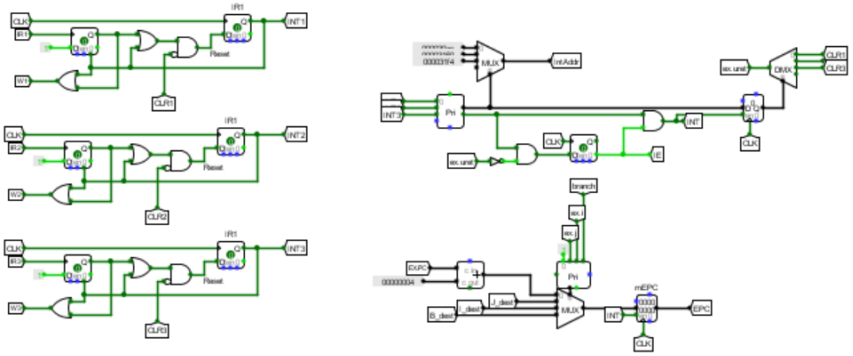

图 3.5 重定向流水线+单级中断逻辑模块

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水接口部件由逻辑门、寄存器与多路选择器实现。多路选择器的作用是实现寄存器的同步清零操作，这一过程采用其他的实现方法大多会因毛刺而失败。流水接口的阻塞与放行功能由逻辑门与寄存器的使能端共同实现。具体电路以图 3.6 中打断 IF/ID 流水接口为例。

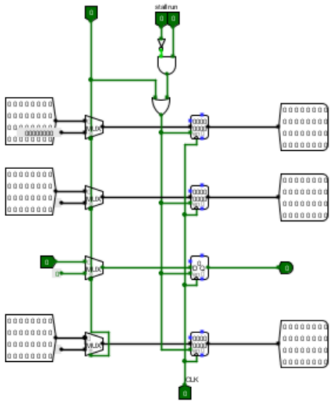

图 3.6 IF/ID 流水接口模块

3.3.2 理想流水线实现

依照 2.3 中设计思路，将单周期 CPU 按指令执行的步骤划分为五段，并通过流水接口传递段与段间的信号与数据，实现了如图 3.7 的理想流水线设计。

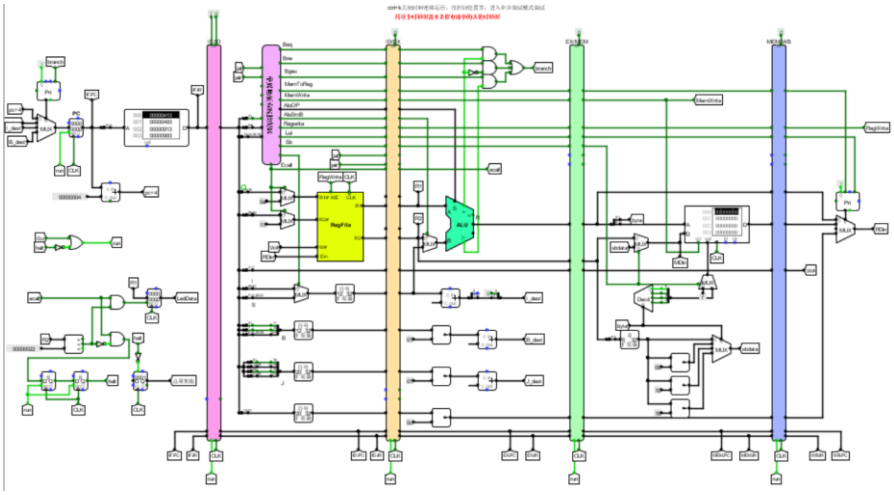

图 3.7 理想流水线实现

3.4 气泡式流水线实现

依据 2.4 中的设计方案，对分支跳转信号与 IF/EX/MEM 段中使用的寄存器信号进行统计，根据分支跳转情况与寄存器使用冲突检测由图 3.9 所示的气泡控制逻辑产生流水接口的清零与阻塞信号。

如图 3.8，将电路中气泡判断逻辑所需信号传入逻辑控制器，并将生成的清零、阻塞控制信号连接至 PC 及各流水控制器控制端口。

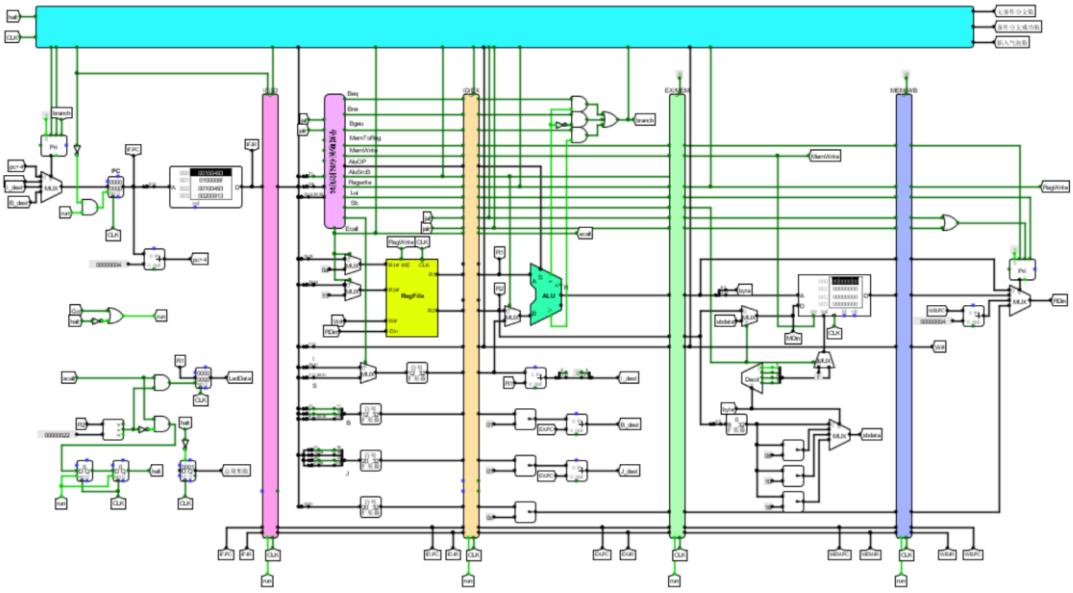

图 3.8 气泡流水线实现

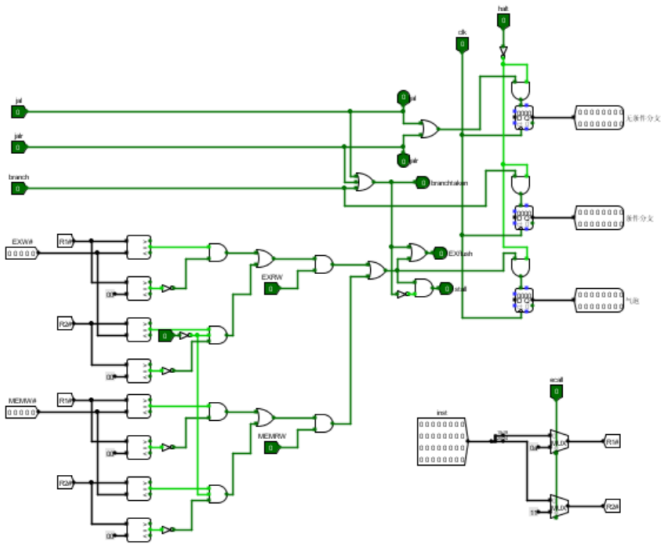

图 3.9 气泡控制逻辑控制单元

3.5 重定向流水线实现

依据 2.5 节中的重定向流水线设计思路，实现了图 3.10 所示的重定向逻辑控制电路，通过比对 IF/EX/MEM 段的寄存器使用情况和 Regwrite, MemWrite 等控制信号产生 Loaduse 信号；再结合分支跳转逻辑确定了流水接口的清零与阻塞逻辑。

如图 3.10 所示，将电路中的相关信号接入重定向控制逻辑，并将生成的流水接口控制信号输入流水接口，从而实现了重定向流水线的设计工作。

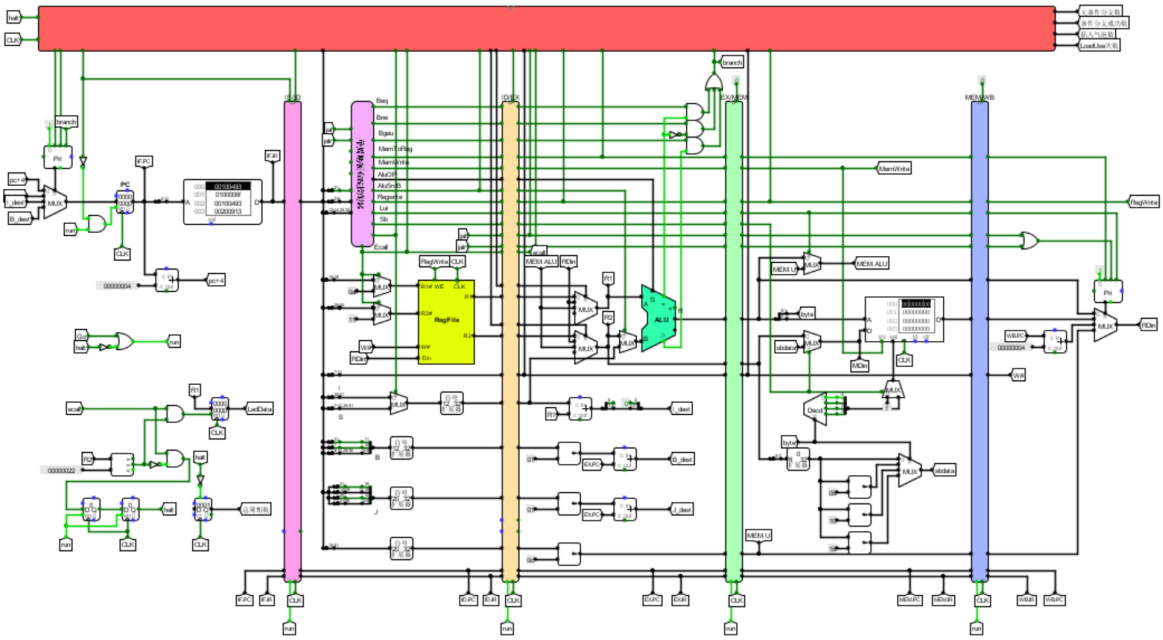

图 3.10 重定向流水线实现

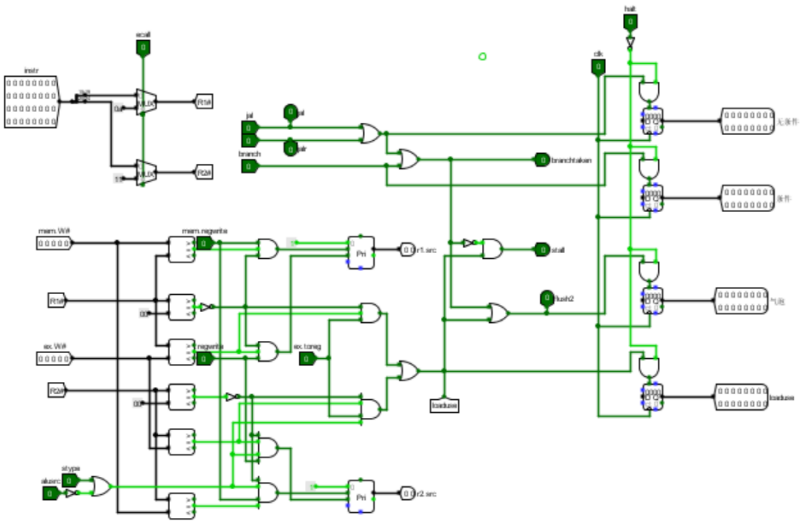

图 3.11 重定向逻辑控制单元

3.6 动态分支预测机制实现

由 2.6 节中动态分支预测的设计思路实现了图 3.13 中基于 8 槽 cache 的动态分支预测模块。模块中包含如图 2.2 实现的预测状态转移电路，当 EX 段命中时更新转移状态，当 IF 段命中时给出相应的跳转预测与跳转地址。预测结果随流水线传递至 EX 段并于真实结果进行比对，若预测正确则继续执行，否则清空预取指令的流水接口。

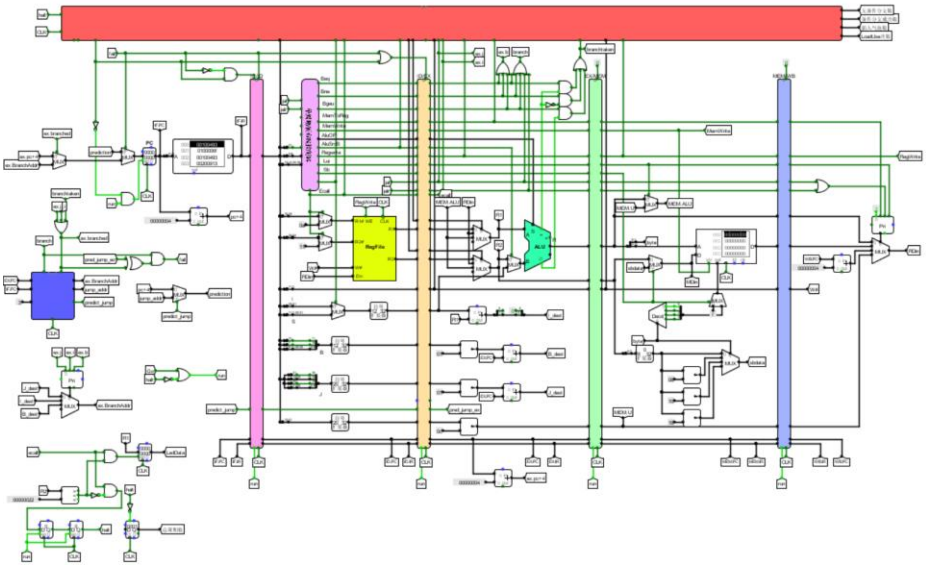

图 3.12 带动态分支预测功能的重定向流水线电路

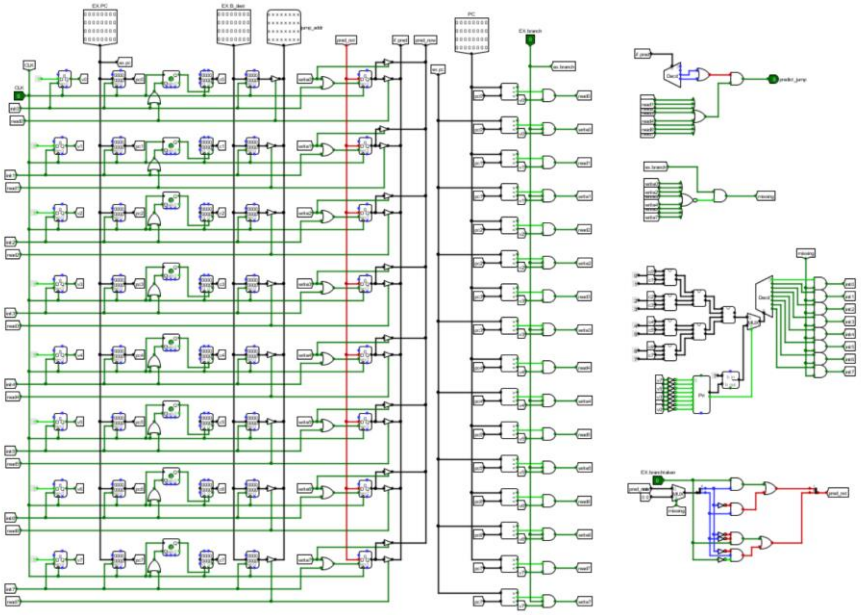

图 3.13 动态分支预测模块

4 实验过程与调试

4.1 测试用例和功能测试

单周期处理器、气泡流水线、重定向流水线以及带有分支预测功能的重定向流水线测试用例来自 risc-v-benchmark.asm 文件以及对应 CCAB 指令的汇编文件合并后通过 rars 工具编译的 hex 程序。将 hex 文件加载进指令寄存器后即可运行，当运行终止后触发 Go 按钮即可进入第一个附加指令测试程序；benchmark 测试与附加指令测试程序的预期结果见相应程序的注释段说明。

中断功能的测试用例来自 risc-v 单级中断测试程序与 risc-v 多级中断测试（EPC 硬件堆栈保护）程序。在自动测试中提前人为设置中断节拍并与标准程序的信号进行比对，从而验证程序正确性；在人工测试中通过触发中断按键并观察数码管输出结果判断程序运行的正确性。

4.1.1 测试用例 1

该测试用例来自 risc-v-benchmark.asm 文件与 CCAB 指令对应汇编文件通过 rars 编译器编译后的 hex 文件。该用例覆盖了设计需求中所要求的所有指令功能测试，测试结果包含流水灯、计数等运算结果，具体预期结果可见其汇编文件的注释段与 CCAB 输出汇总中的详细说明。

4.1.2 测试用例 2

该测试用例针对单级中断设计，目的为测试单级中断响应功能的正确性。在头歌平台自动测试中使用预先设定的中断节拍并与标准程序的信号进行单步比对验证，人工测试可使用中断按键。按键触发后的运行结果为对应数字由右向左的三轮走马灯，且中断处理不可被打断。

4.1.3 测试用例 3

该测试用例针对使用硬件堆栈保护的多级中断设计，使用 CSRRSI 与 CSRRCI 指令支持中断响应寄存器的开关中断操作。中断按键触发后的运行结果为对应数字的走

马灯，且在开中断的情况下低级中断可以被高级中断所打断。

4.2 性能分析

由于 Logsim 软件对电路的并行处理能力有限，不能通过分析相同程序的运行时间比较不同设计方案间的运行性能。由于总时间 $=\frac{\text{总周期}}{\text{处理器频率}}$，而处理器频率受限于电路的关键路径延迟，因此我们可以通过分析不同方案下处理器运行的总周期数与关键路径延迟做出方案间的性能比较，结果如表 4.1 所示。

表 4.1 不同方案处理器运行 benchmark 程序性能分析

电路	总周期数	关键路径延迟
单周期 CPU	1545	IF+ID+EX+MEM+WB
气泡流水线 CPU	3624	max{IF,ID,EX,MEM,WB}*
重定向流水线 CPU	2297	max{IF,ID,EX,MEM,WB}*
带有动态分支预测的重定向流水线 CPU	1764	max{IF,ID,EX,MEM,WB}*

注：分析关键路径延迟时忽略了流水线接口的路径延迟

由表可见单周期处理器的总周期数最少，但是关键路径延迟远大于流水线处理器。在流水线处理器当中，气泡流水线所需总周期数最多；由于重定向计数优化了气泡产生的条件，因此总周期数减少，性能提高；在此基础上动态分支预测功能降低了分支预取指令清空的概率，因此进一步对总周期数进行了优化，达到了几种方案中的最佳性能。

4.3 主要故障与调试

4.3.1 流水线接口同步清零故障

流水线接口： 流水寄存器同步清零恢复错误

故障现象：流水寄存器同步清零后无法在下一周期恢复。如图 4.1（左），在按下时钟信号后 out 信号能被成功同步清零，若下一时刻清零信号消失（如图 4.1（中）），则下一时刻 out 信号仍然为零，未能恢复为 1，如图 4.1（右）。

原因分析：如图 4.1，原计划采用 cache 实验中相同的解决方案，在寄存器异步清零端口增设 D 触发器将其改造为同步清零端口。

但是在实际实验中发现这样存在问题，原因可能是由于数据通路长度不同，来自 D 触发器的信号到达寄存器异步清零端口的时间迟于触发器更新的时间。清零过程可以顺利完成，因为虽然清零信号到达较晚，但由于该端口的异步性最终仍能将输出信号置为零；但是在恢复过程中，由于寄存器同步更新时清零端口信号仍为 1，所以寄存器不能及时更新为 1，当异步清零信号到来时已经为时过晚，所以输出信号不能恢复为 1.

解决方案：使用多路选择器控制寄存器输入从而实现同步清零

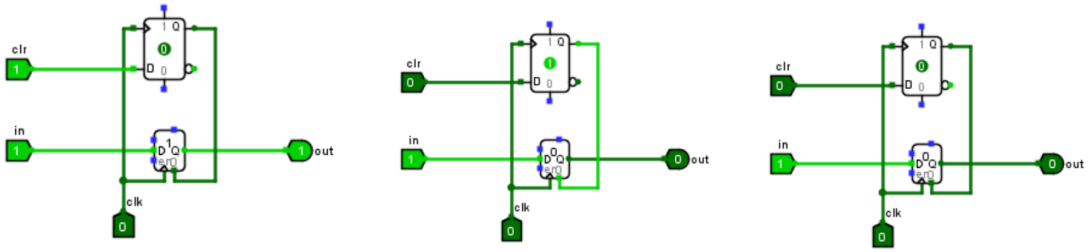

图 4.1 流水寄存器同步清零后无法在下一周期恢复示意图

4.3.2 BGEU 指令故障

ALU 故障：比较器有无符号设置出错。

故障现象：Logism 与 FPGA 在检查时 BGEU 指令对应程序运行结果均出错

原因分析：直接使用给定 ALU 的大于等于端口，忽略了 BGEU 指令所需无符号比较与 ALU 大于等于端口默认有符号比较之间的冲突。

解决方案：重新修改 Logism 给定 ALU 的内部比较逻辑，修改 Verilog 代码，使用无符号数强制类型转化表达式，将 Geq 端口设置为两操作数的无符号比较结果。

4.3.3 Verilog 运行结果出错

RegFile 故障：忽略了 risc-v 指令中零号寄存器初值为 0 且不能修改

故障现象：FPGA 电路板运行结果与预期不符

原因分析：未对 RegFile 中寄存器数组做初始化，在 Verilog 中未初始化的变量均为 X 形式的三值逻辑。而在 risc-v 指令集中零号寄存器的值固定且无法修改，在执行后续指令时寄存器归零操作要求使用零号寄存器完成，若未对其进行初始化则会引发错误

解决方案：使用 initial 语句对寄存器数组进行初始化，并约束寄存器的修改条件

4.4 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 RISC-V 指令手册，并列出 CPU 各部件的数据通路表，并完成数据通路的基本构建。
第二天	完成单周期 CPU 的控制信号表，使用 Logisim 搭建控制器，实现了单周期 CPU 并且通过了测试。
第三天	阅读关于指令流水线的相关知识，完成理想流水线的 Logisim 实现。
第四天	学习气泡流水线的相关内容，部分完成气泡流水线逻辑控制器的搭建。
第五天	完成气泡流水线设计并通过自动测试，阅读重定向流水线的相关章节，理解 Loaduse 等有关内容。
第六天	完成重定向流水线的实现与调试，复习中断处理逻辑与相关软硬件知识。
第七天	完成单周期 CPU 的单级中断设计与调试，通过头歌平台自动测试；进一步阅读复习关于多级嵌套中断的软件与硬件保护机制。
第八天	完成单周期 CPU 的硬件堆栈保护多级中断设计与调试，完成重定向流水线的单级中断实现与测试。
第九天	阅读动态分支预测相关内容，搭建 8 槽 cache 并实现 BTB 预测单元；将其加入重定向流水线并调试成功。
第十天	复习 Verilog 相关知识，并对照 Logisim 电路完成单周期 CPU 的 Verilog 程序编写工作。
第十一天	Debug Verilog 程序，最终完成 FPGA 版的烧制，实现单周期上版功能。

5 设计总结与心得

5.1 课设总结

基于 RISC-V 指令集的五段流水线 CPU 是具有一定规模指令系统的简单计算机中央处理器系统；经过多步优化后，处理器能较高效地完成由部分 RISC-V 汇编代码编写的程序。本设计在课程设计概述的指导下作了如下几点工作：

1） 完成了基于 RISC-V 指令集的单周期、理想流水线、气泡流水线、重定向流水线处理器，实现了单级/多级嵌套/流水中断、动态分支预测的处理逻辑，并实现了单周期的开发板实现。

2） 处理器能够执行技术指标中规定的 24+4 条 RISC-V 指令，并支持单周期嵌套中断、流水线单级中断，同时通过分支预测模块提高了流水线性能。

3） 完成了 Logisim 电路图到 Verilog 程序的跨平台编写，使用 Vivado 完成了 FPGA 板的烧制工作。

5.2 课设心得

本次课设是我首次完成如此大规模的硬件设计与跨平台实现工作。最初面对如此复杂度的工程问题时，我感到一头雾水、一团乱麻，不知从何开始；但是在实验指导书和头歌关卡的一步步引导下，逐渐能够将复杂问题拆分成子问题，并从指令格式、数据通路等方面开始，从单周期向流水线一步一步地走向最终设计的实现。**对我而言，这个将复杂问题拆分为简单的、可逐步实现的任务的过程就是极大的锻炼与进步。**最终实现的设计与结果在一开始是很难想象的。

本次课程设计也是对数字逻辑、计算机组成原理、Verilog 程序设计课程所学知识的综合运用，当课程所学知识转化为解决实际问题时的生产力，并逐地呈现出威力与效用时，我真心享受其中学以致用的过程。同时，在实际检验中也对所学知识进行了提升与完善，例如 4.3.1 节中的故障唤起了当年数字逻辑课上关于电路“竞争”的讨论，帮助我在实验中加深了理论认识。

在最终的跨平台代码编写中，我体会到了不同平台的特点与优势，Verilog 在实现上虽然抽象但更为灵活。相信该经历为后续的硬件开发加深了基础。

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.

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：范辰睿