

# 华中科技大学

## 2017

### 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:	CS1409
学 号:	U201414797
姓 名:	张丹朱
电 话:	13297920602
邮 件:	1240864213@qq.com
完成日期:	2017-03-26 周日下午



计算机科学与技术学院

## 目 录

<b>1</b>	<b>课程设计概述 .....</b>	<b>3</b>
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
<b>2</b>	<b>总体方案设计 .....</b>	<b>6</b>
2.1	单周期 CPU 设计 .....	6
2.2	中断机制设计 .....	8
2.3	流水 CPU 设计 .....	9
2.4	数据转发流水线设计 .....	10
2.5	气泡式流水线设计 .....	10
<b>3</b>	<b>详细设计与实现 .....</b>	<b>11</b>
3.1	单周期 CPU 实现 .....	11
3.2	中断机制实现 .....	16
3.3	流水 CPU 实现 .....	21
3.4	数据转发流水线实现 .....	22
3.5	气泡式流水线实现 .....	24
<b>4</b>	<b>实验过程与调试 .....</b>	<b>25</b>
4.1	测试用例和功能测试 .....	25
4.2	流水线 FPGA 实现 .....	28
4.3	性能分析 .....	28
4.4	主要故障与调试 .....	29
4.5	实验进度 .....	32

# 华中科技大学课程设计报告

---

5 设计总结与心得.....	33
5.1 课设总结 .....	33
5.2 课设心得 .....	33
参考文献.....	35

## 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	SRAV	算术可变右移	
29	XOR	异或	
30	SH	存储半字	
31	BGTZ	大于 0 转移	

## 2 总体方案设计

### 2.1 单周期 CPU 设计

本次实验采用指令存储器和中断处理程序存储器不共用一个存储器的方式完成方案的设计。用控制器生成所有的控制信号，在一个周期内完成指令的执行。

所有电路全部在 logisim 上进行实现并测试。

总体结构图如图 2.1 所示。

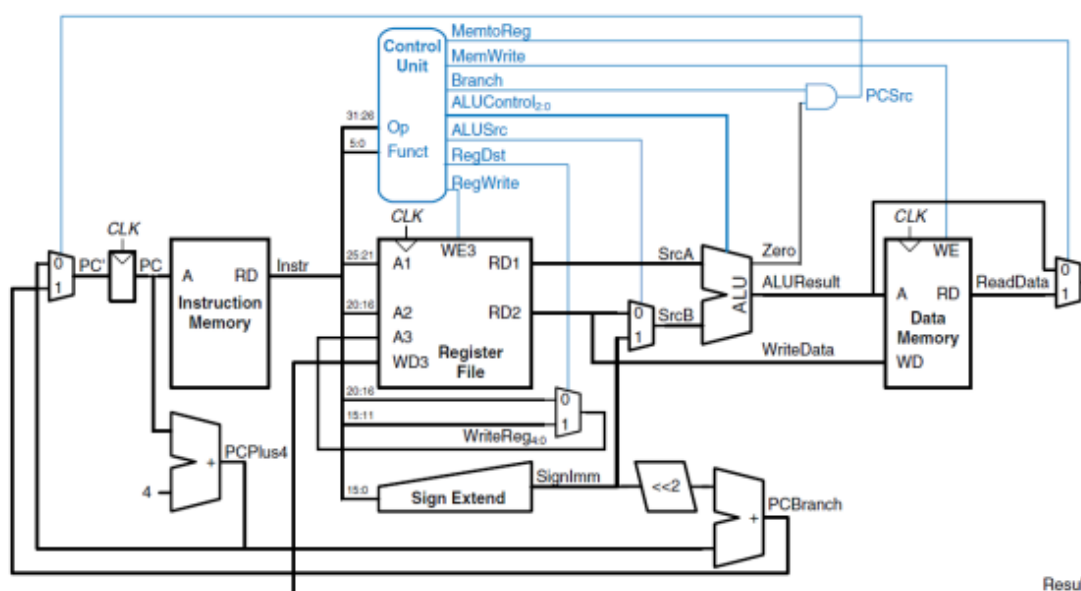


图 2.1 总体结构图

#### 2.1.1 主要功能部件

运算器部分，具体设计思路如下：采用上学期实验所完成的运算器，即利用自己设计的 32 位快速加法器以及 logisim 平台上已有的相关部件构建运算器。

##### 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，当需要进行停机时，Halt 控制信号为 0，屏蔽时钟信号，使整个电路停机。

# 华中科技大学课程设计报告

## 2. 指令存储器 IM

使用一个只读存储器 ROM 实现指令存储器 (IM)。设置该只读存储器的地址位宽为 10 位, 数据位宽为 32 位。因为 PC 中存储的指令地址有 32 位, 而 ROM 地址线宽度有限, 仅为 10 位, 故将 32 位指令地址高位部分和字节偏移部分直接屏蔽, 使用分线器只取 32 位指令地址的 2-11 位作为指令存储器的输入地址。

## 3. 运算器

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

## 4. 寄存器堆 RF

使用 32 个 32 位寄存器实现相应功能。

### 2.1.2 数据通路的设计

表 2.2 指令系统数据通路框架

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	



# 华中科技大学课程设计报告

## 2.1.3 控制器的设计

首先对于控制信号进行统计，包括各个主要部件所需要输入的控制信号，以及数据通路合并表中所示的具有多输入的主要部件需要进行输入选择的控制信号，并且对各个统计信号的各种取值情况进行定义，统计得到的控制信号以及说明如表 2.3。

表 2.3 主控制器控制信号的作用说明

控制信号	取值	说明
R1	0	寄存器堆 R1 口读取 rs 字段指示寄存器的值
	1	寄存器堆 R1 口读取 2 号寄存器的值

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	R	RW	WE	X	EXT	Y	ALUop	MemWrite	MemRead	Din	Branch	SYSCALL

## 2.2 中断机制设计

### 2.2.1 总体设计

本次实验需要设计一个支持多级嵌套的中断机制。

当中断请求产生时将请求信号暂存寄存器，通过中断屏蔽电路后获得有效的中断请求信号，当下一个时钟上升沿来临时才进行信号的输出，进入中断相应部分。

中断响应时使用硬件完成对旧 EPC 的存储和对新的屏蔽字的加载(同时使用硬件清除对应的中断请求信号)，然后使用软件依次完成对旧屏蔽字的存储，对使用

# 华中科技大学课程设计报告

---

到的寄存器进行保存，接着开始执行中断，执行完成之后恢复现场、恢复 pc 并重新回到中断点继续执行上一段程序。

由于中断屏蔽电路的存在，可以实现多级嵌套。

## 2.2.2 硬件设计

硬件电路主要实现以下功能：

- 1) 产生中断信号后由当前程序跳转到中断处理程序；
- 2) 保存 EPC；
- 3) 中断屏蔽；

中断处理机制主要依靠简易的 CP0 进行实现，简易 CP0 包含三个寄存器：EPC 保存旧的 PC 值，STATUS 的最后一位保存中断使能信号，CAUSE 寄存器保存中断号。

在产生中断请求信号之后使 EPC 寄存器使能信号有效以保存断点的 PC 值，并通过数据选择器进入到中断响应程序的起始地址。

中断屏蔽电路则是根据 CAUSE 的值（低两位）通过与、或门产生对应的屏蔽信号再与中断请求信号经过与门后获得有效的请求信号。

## 2.2.3 软件设计

通过编程，首先关中断，将 EPC 寄存器中的值存放到内存中（以栈的方式），然后保护现场，将要用到的寄存器入栈，接着存屏蔽字即中断号（此处判断是否是从主程序进入，若是则存 0，否则存上一轮中断的中断号），然后转入中断服务程序，将 CAUSE 寄存器中的值转入一个通用寄存器（存屏蔽字时存该寄存器的值），根据中断号跳转到响应的地址进行中断服务，处理完后开中断，恢复屏蔽字，恢复现场，恢复 PC，开中断，执行 eret，回到断点继续执行。

## 2.3 流水 CPU 设计

### 2.3.1 总体设计

采用 5 段流水线，将一条指令分为取指、译码、执行、访存、写回五个周期，每段之间增添缓冲接口（寄存器实现），通过接口传递与指令相关的数据信息、控

# 华中科技大学课程设计报告

---

制信息、反馈信息，后续部件对数据的加工处理依赖于前段传递过来的信息。故在 ID 段译码生成该指令的所有控制信号并向后传递，后续部件控制信号不再单独生成。

对于数据冲突和分支冲突，采用插气泡和重定向的方式消除。

## 2.3.2 流水接口部件设计

不同阶段之间设置缓冲接口部件，接口部件采用寄存器实现，上一阶段的数据信息、控制信息等全部传递到接口部件，在下一个时钟上升沿到来时就将数据传递到下一阶段。

## 2.3.3 理想流水线设计

理想流水线不考虑分支和数据冲突，只需在 ID 阶段将指令所需的所有控制信号全部生成并向后传递，同时将所需的数据也向后传递即可。将原单周期电路拉开分成 5 个阶段并在各段间插入缓冲接口，相应数据与控制信号通过缓冲接口传递，直到后续不再使用就停止传递。

## 2.4 数据转发流水线设计

为避免冗余的插气泡(除 load\_use 外其余的数据冲突皆可以通过数据转发解决)首先进行数据转发流水线的设计，在 ID 段译码的时候解析出该条指令是否需要读取寄存器，将所要读取的寄存器与 EX 段和 MEM 所要写入的寄存器进行比较，若是相同的（0 号寄存器除外），则给出选择信号，在下一周期 EX 段直接选择上一周期 EX 段（或 MEM 段）的数据代替从寄存器读出的值。

## 2.5 气泡式流水线设计

在 ID 段检测 load\_use 冲突，当 ID 段的指令需要读某寄存器而 EX 段正执行 LW 指令会写入该寄存器时则产生 LOADUSE 信号，将 EX 段之前的段全部暂停并在下一个时钟沿到来之时清空 EX 段，即插入一个气泡，之后便可通过重定向的方式解决冲突。

本次设计在 EX 阶段进行分支跳转，故当在 EX 阶段发现跳转时同样清空 ID 和 EX 段，删除误取指令。

## 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，输出为当前执行指令的地址。Halt 为停机信号，当需要进行停机时，Halt 控制信号为 0，屏蔽时钟信号，使整个电路停机。如图 3.1 所示。

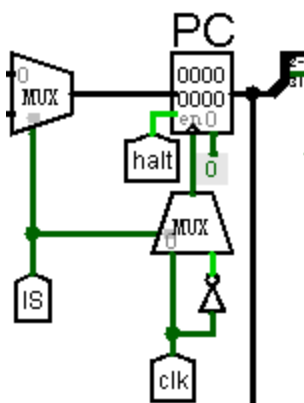


图 3.1 程序计数器 (PC)

###### ② FPGA 实现:

程序计数器 PC 的 Verilog 代码如下:

```
assign pc_4 = pc+4;
assign pc_en = halt & (~LOADUSE);
assign pc_next = E_jr?E_R1:E_addr;
reg_pos pc_reg(clk,1'b0,pc_en,pc_next,pc);
```

##### 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

# 华中科技大学课程设计报告

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。

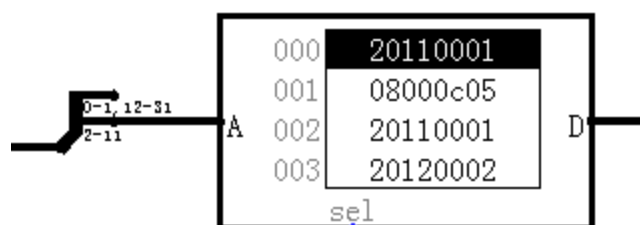


图 3.2 指令存储器 (IM)

## ② FPGA 实现:

在 Vivado 中编写 ROM。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:

```
module ROM(raddr,dout);
    parameter    DWIDTH = 32; //数据宽度
    parameter    AWIDTH = 10; //地址宽度
    input [AWIDTH-1:0] raddr;
    output [DWIDTH-1:0] dout;

    reg [DWIDTH-1:0] ROM_[0:2**AWIDTH-1];

    initial
    begin
        $readmemh("D:\\MIPS_CPU\\test\\benchmark.hex",ROM_,0,2**AWIDTH-1);
    end

    assign dout = ROM_[raddr];
endmodule
```

&

# 华中科技大学课程设计报告

ROM u\_ROM(pc[11:2],rom\_out);

直接调用自己编写的 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。

## 3.1.2 数据通路的实现

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

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

表 3.1 指令系统数据通路表

指令	PC	IM	RF				S_ext	ALU		DM	
			R1#	R2#	W#	Din		A	B	Addr	Din
add	pc+4	pc	rs	rt	rd	alu		Rf.d 1	Rf.d 2		
addi	pc+4	pc	rs		rt	alu	Imm _16	Rf.d 1	S_ext t		
addi u	pc+4	pc	rs		rt	alu	Imm _16	Rf.d 1	S_ext t		
addu	Pc+4	pc	rs	rt	rd	alu		Rf.d 1	Rf.d 2		
and	Pc+4	pc	rs	rt	rd	alu		Rf.d 1	Rf.d 2		
andi	Pc+4	pc	rs		rt	alu	Imm _16	Rf.d 1	S_ext t		
sll	Pc+4	pc		rt	rd	alu		Rf.d 2	sa		
sra	Pc+4	pc		rt	rd	alu		Rf.d 2	sa		
srl	Pc+4	pc		rt	rd	alu		Rf.d 2	sa		
sub	Pc+4	pc	rs	rt	rd	alu		Rf.d 1	Rf.d 2		
or	Pc+4	pc	rs	rt	rd	alu		Rf.d 1	Rf.d 2		
ori	Pc+4	pc	rs		rt	alu	Imm _16	Rf.d 1	S_ext t		
nor	Pc+4	pc	rs	rt	rd	alu		Rf.d 1	Rf.d 2		
lw	Pc+4	pc	rs		rt	Dm_	Imm	Rf.d	S_ext	alu	

# 华中科技大学课程设计报告

						out	_16	1	t		
sw	Pc+4	pc	rs	rt			Imm _16	Rf.d 1	S_ex t	alu	Rf.d 2
beq	Pc+4 +offs et/pc +4	pc	rs	rt			Imm _16	Rf.d 1	Rf.d 2		
bne	Pc+4 +offs et/pc +4	pc	rs	rt			Imm _16	Rf.d 1	Rf.d 2		
slt	Pc+4	pc	rs	rt	rd	alu		Rf.d 1	Rf.d 2		
slti	Pc+4	pc	rs		rt	alu	Imm _16	Rf.d 1	S_ex t		
sltu	Pc+4	pc	rs	rt	rd	alu		Rf.d 1	Rf.d 2		
j	(PC+ 4)[31 ..28], addr ess,0 ,0	pc									
jal	PC <= (PC+ 4)[31 ..28], addr ess,0 ,0				31	Pc+4					
jr	Rf.d 1	pc	rs								
sysca ll											
srav	pc+4	pc	rs	rt	rd	ALU		rf.d2	rf.d1		
xor	pc+4	pc	rs	rt	rd	ALU		rf.d1	rf.d2		
sh	pc+4	pc	rs	rt			imm _16	rf.d1	s_ext	ALU	拼出 的
bgtz	Pc+4 +offs et/pc +4	pc	rs				imm _16	0	rf.d1		
合并	4 输入	pc	rs	rt	3 输入	3 输入	Imm _16	3 输入	4 输入	alu	Rf.d 2

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

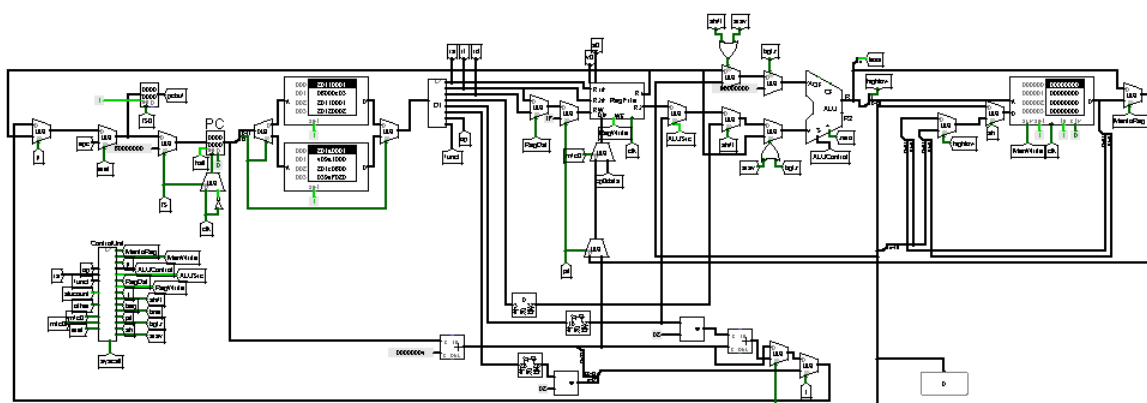


图 3.3 单周期 CPU 数据通路 (Logism)

## 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 上进行主控制器的具体实现。

主控制器

对照表 3.2 所示。

表 3.2 主控制器控制信号

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
控制信号	add	addi	addiu	addu	and	andi	sll	sra	srl	sub	or	ori	nor	lwr	sw	beq	bne	slt	slti	sltu	j	jal	jr	syscall
ADD	1	1	1	1										1	1									
SUB										1						1	1							
AND					1	1																		
OR											1	1												
NOR													1											
SLL							1																	
SRA								1																
SRL									1															
SLT																		1	1					
SLTU																				1				
jr																							1	
shift							1	1	1															
RegDst	1			1	1		1	1	1	1	1		1					1		1				
jal																						1		
RegWrite	1	1	1	1	1	1	1	1	1	1	1	1	1	1				1	1	1		1		



# 华中科技大学课程设计报告

控制信号	a d d	a d d i	a d d i u	a d d u	a n d	a n d i	s l l	s r a	s r l	s u b	o r	o r i	n o r	l w	s w	b e q	b n e	s l t	s l t i	s l t u	j	ja l	jr	sy sc all
ALUSrc		1	1			1						1		1	1				1					
MenWrite															1									
MentoReg														1										
j																					1	1		
beq																1								
bne																	1							
syscall																								1

## ① FPGA 实现

根据在 Logism 实现中得到的各个控制信号的表达式，直接使用数据流建模，首先根据 opt 与 funct 获得代表各条指令的信号，然后根据各控制信号关于各指令的表达式生成信号。

由于使用 assign 的 Verilog 代码过于冗长，故只取指令信号 srav 和控制信号的生成代码举例如下：

```
assign srav = ({op,funct}==12'b0000000000111)?1:0;
....
assign RegDst = xor_|add|addu|and_|sll|sra|srl|sub|or_|nor_|slt|sltu|srav;
```

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。

## 3.2 中断机制实现

### 3.2.1 硬件实现

中断屏蔽电路的实现由 CAUSE 寄存器的低两位引出后通过基本的与、或门获得对应的屏蔽电路。具体实现上由于优先级  $3>2>1$ ,故当中断号为 01 或 10 或 11 时中断请求信号 1 被屏蔽，即第 0 位与第 1 位取或后得 1 号中断的屏蔽信号。同理，第 1 位作为中断 2 的屏蔽信号，最低 2 位通过与门后得到 3 号中断的屏蔽信号。具体电路实现如下图 3.4 所示。

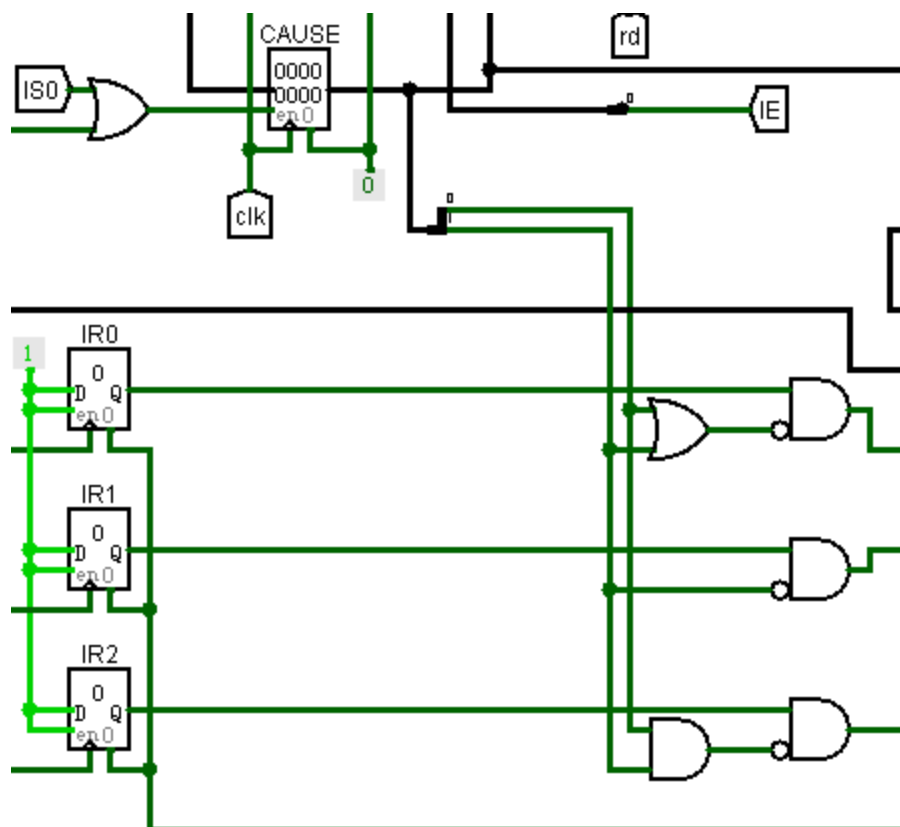


图 3.4 中断屏蔽电路

有效的中断请求信号的产生通过与门，经由中断使能信号与 `eret` 信号控制获得。该中断请求信号通过寄存器在下一个时钟上升沿来临时获得中断响应信号，该信号获得后的下一个时钟下降沿清除，同时清除对应的中断请求。具体实现如下图 3.5 所示。

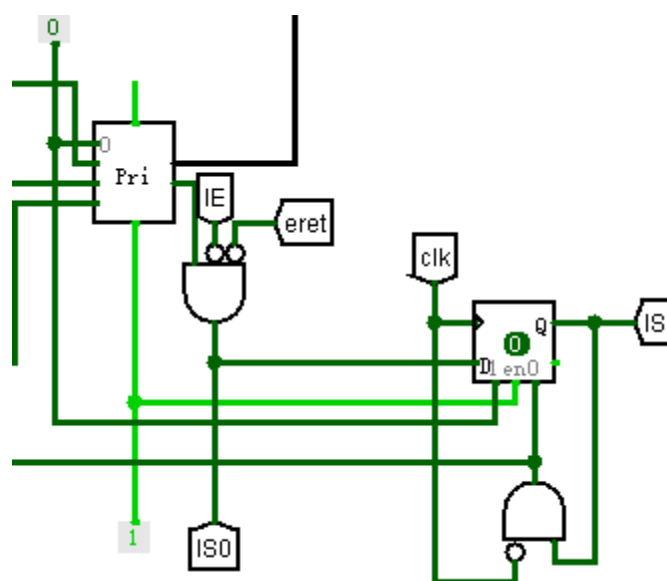


图 3.5 中断信号产生与反馈电路

# 华中科技大学课程设计报告

EPC 的保存和中断号的保存同样使用硬件电路完成，当有效的中断请求信号产生后，在下一个时钟上升沿来临时即进行相关信息的保存。具体的电路实现如下图 3.6 所示。

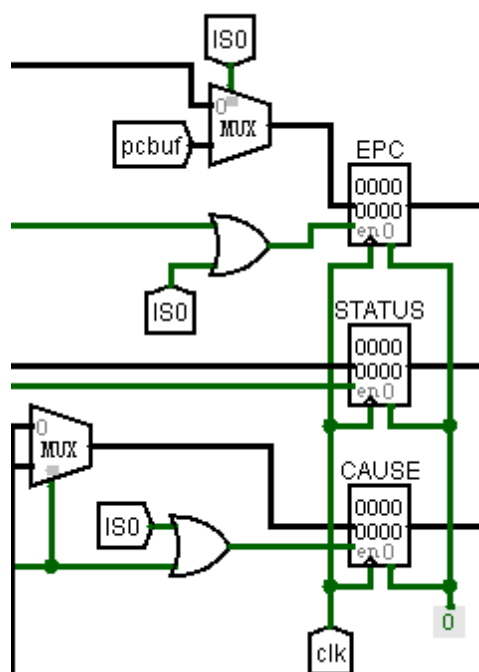


图 3.6 EPC 与 CAUSE 保存电路

中断处理程序有固定的入口地址，当中断响应信号产生后即进入中断处理程序。具体在电路上采用主指令存储器与中断处理程序存储器分开的方式，中断响应信号作为片选信号进行主程序与中断程序的选择。具体实现电路如下图 3.7 所示。

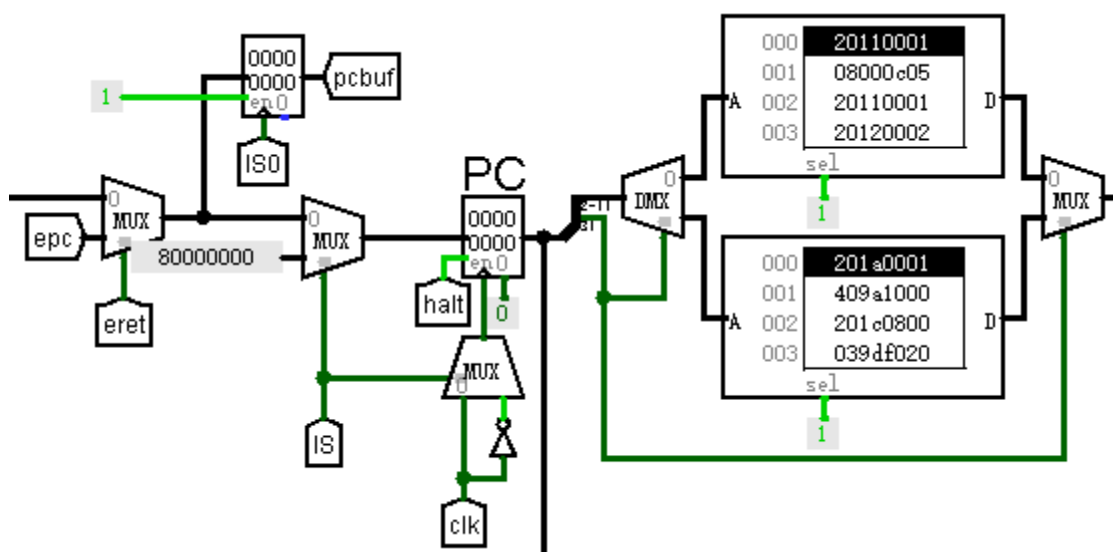


图 3.7 中断进入电路

## 3.2.2 软件实现

软件处理流程为：

### 1.关中断

由于 IE 信号即为 STATUS 的最低位，故关中断时只需将寄存器 STATUS 的值赋为 1 即可。具体代码如下：

```
addi $k0,$zero,1  
mtc0 $k0,$2
```

### 2.设置栈顶指针

```
addi $gp,$zero,0x800  
add $fp,$gp,$sp
```

### 3.保存 PC

由于断点 pc 值在 CP0 的 EPC 中，而要存入内存目前只有 SW 对通用寄存器进行操作可以实现，故要使用 mfc0 指令先将 EPC 存入通用寄存器再入栈。具体代码如下：

```
mfc0 $k0,$1  
sw $k0,($fp)  
addi $fp,$fp,-4  
addi $sp,$sp,-4
```

### 4.保护现场

将中断处理程序所需寄存器全部入栈，避免现场被破坏。以下仅以一个寄存器的保存为例：

```
sw $a0,($fp)  
addi $sp,$sp,-4  
addi $fp,$fp,-4
```

### 5.保护屏蔽字

由于屏蔽字的加载采用硬件电路实现，当中断程序执行到此处时屏蔽字已经不在 CP0 的 CAUSE 寄存器中了，又由于每次中断都会将中断号（相当于屏蔽字）存在 t0 寄存器中，故保护屏蔽字其实只需将 t0 的值保存。考虑到主程序初次进入中断这一特殊事件需要单独考虑，故程序中需要进行相应判断，若是从主程序进来

# 华中科技大学课程设计报告

---

直接保存 00 即可。而判断是否从主程序进入中断只需比较栈顶指针的值即可。具体代码如下：

```
addi $t1,$zero,-24
beq $sp,$t1,init
sw $t0,($fp)
addi $fp,$fp,-4
addi $sp,$sp,-4
bne $t1,$zero,open
init:
addi $t1,$zero,0
sw $t1,($fp)
addi $fp,$fp,-4
addi $sp,$sp,-4
```

## 6. 开中断，进入中断处理程序

中断程序中需要根据中断号进行相应的服务（1、2 号中断皆为跑马灯，3 号中断用于测试扩展指令），故在进行中断处理前需要有判断与跳转的部分。具体代码如下（省略开中断）：

```
mfc0 $t0,$3 # get cause number
addi $v0,$zero,3
beq $t0,$v0,mypro
addi $t1,$zero,8
add $a0,$zero,$t0
addi $t2,$zero,1
loop:
syscall
sll $a0,$a0,4
sub $t1,$t1,$t2
bne $t1,$zero,loop
beq $t1,$zero,close
```

```
mypro:
addi $a0 $zero 16384
addi $t1 $zero 1
addi $t2 $zero 0x804
loop1:
syscall
add $t2 $t2 2
srav $a0 $a0 $t1
sh $a0 1($t2)
bgtz $a0 loop1
addi $a0 $zero 0x00000aaa
addi $t1 $zero 0x00000555
xor $a0 $a0 $t1
syscall
addi $a0 $zero 0x00000aaa
addi $t1 $zero 0x00000aaa
xor $a0 $a0 $t1
syscall
```

## 7.恢复与返回

按照前面相反的顺序依次进行关中断，恢复屏蔽字（即重新加载中断号），恢复现场（同样以与前面相反的顺序进行），恢复 PC，开中断，返回（eret）。

## 3.3 流水 CPU 实现

### 3.3.1 流水接口部件实现

流水接口部件采用寄存器实现，有多少信号需要传递便使用多少个寄存器，另外要有清零信号与使能信号以实现插气泡与暂停的功能。以 IF/ID 为例，电路实现如下图 3.8 所示。

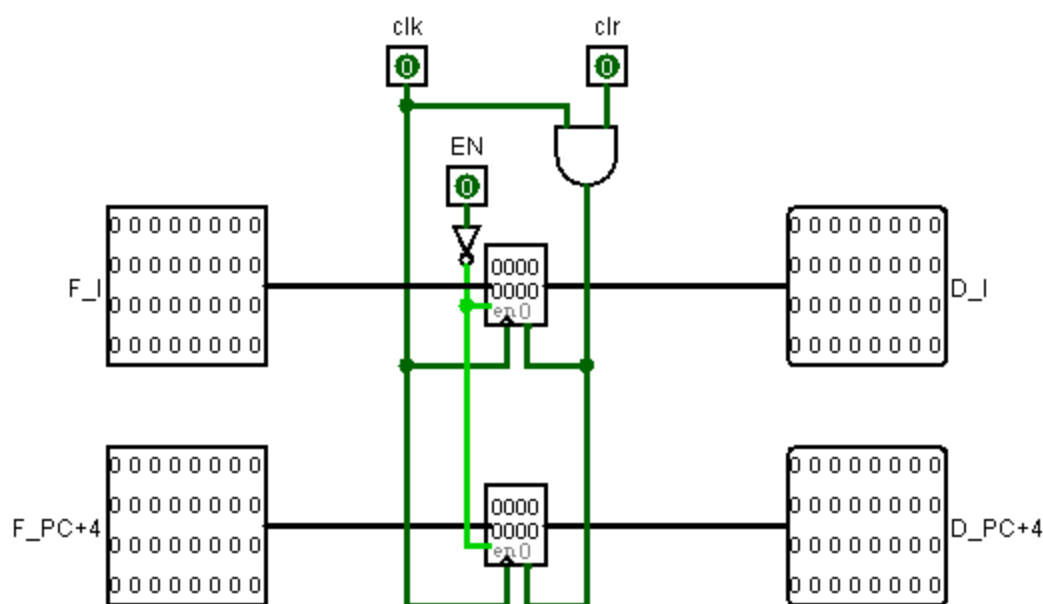


图 3.8 IF/ID 电路

## 3.3.2 理想流水线实现

由于理想流水线不需考虑数据冲突与分支冲突，故只需将原单周期流水线按分段拉开，各段之间加上缓冲接口部件完成数据与信号的传递工作即可。

## 3.4 数据转发流水线实现

在 ID 段译码的时候解析出该条指令是否需要读取寄存器，控制单元中增添 readRs 与 readRt 信号标识是否需要读相应寄存器，将所要读取的寄存器与 EX 段和 MEM 所要写入的寄存器进行比较，若是相同的（0 号寄存器除外），则给出选择信号，在下一周期 EX 段直接选择上一周期 EX 段（或 MEM 段）的数据代替从寄存器读出的值。封装一个冒险处理单元进行冲突处理，其中 Rs 与 Rt 的值是否需要重定向以及从何处重定向的信号产生电路如下图 3.9 所示。

在电路中值得注意的是 0 号寄存器的值由于一直为 0，为防止出现写入 0 号寄存器指令的出现导致错误的重定向，在此电路中使用比较器判断所需读的寄存器是否为 0 号寄存器，若是则可以直接使用，忽略重定向相关内容。

判断 EX 段与 MEM 段是否存储相应编号的寄存器时使用的信号皆是传递到 EX 与 MEM 的信号。

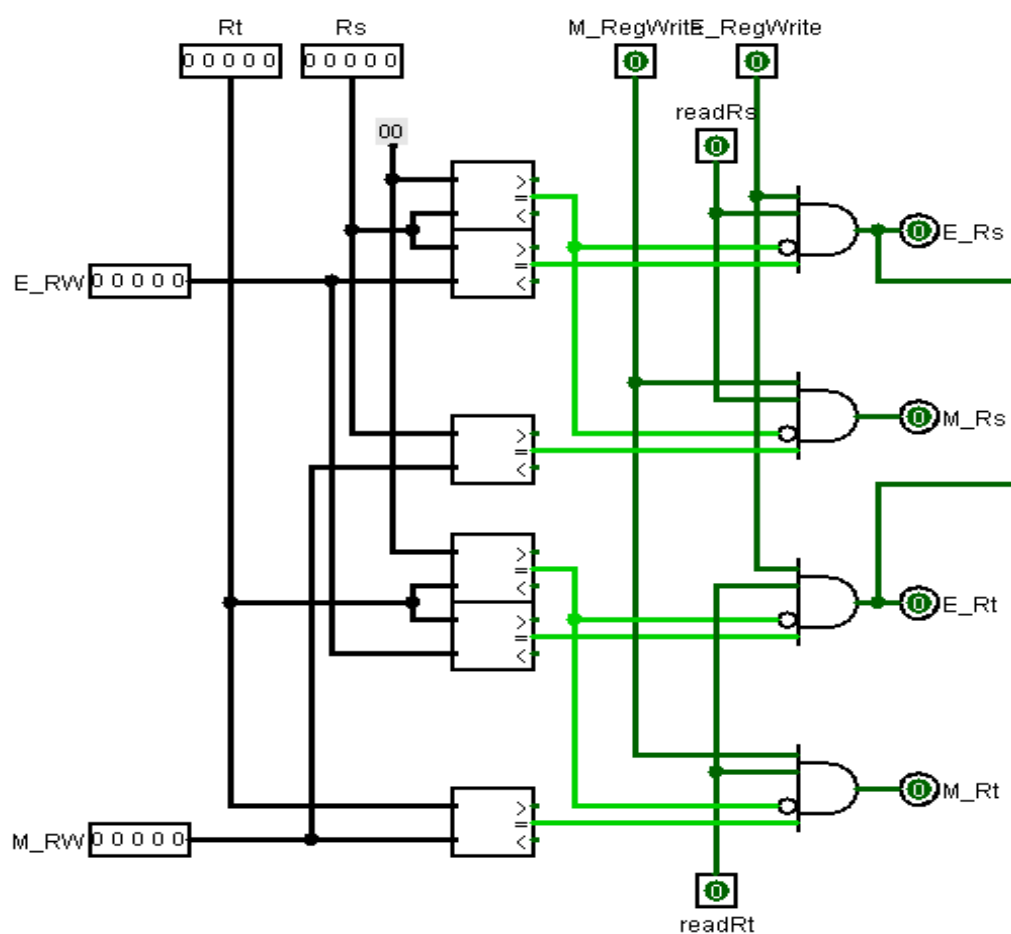


图 3.9 重定向信号产生电路

重定向信号用于在 EX 段作为数据选择器的选择端进行寄存器读数与重定向数据的选择。如下图 3.10 所示。

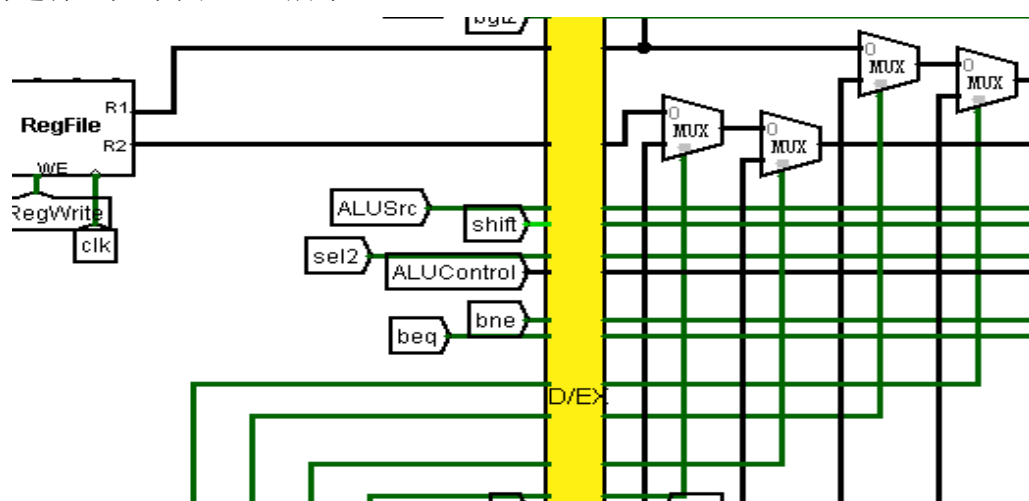


图 3.10 重定向数据选择电路



## 3.5 气泡式流水线实现

在重定向电路的基础上，解决 load\_use 与分支冲突都需要插气泡（或是清空，实现上都相同）。在冒险处理单元中增添 LOADUSE 与 BRANCH 信号的生成，通过这两个信号对相应的段进行清零或暂停操作。这两个信号的生成电路如下

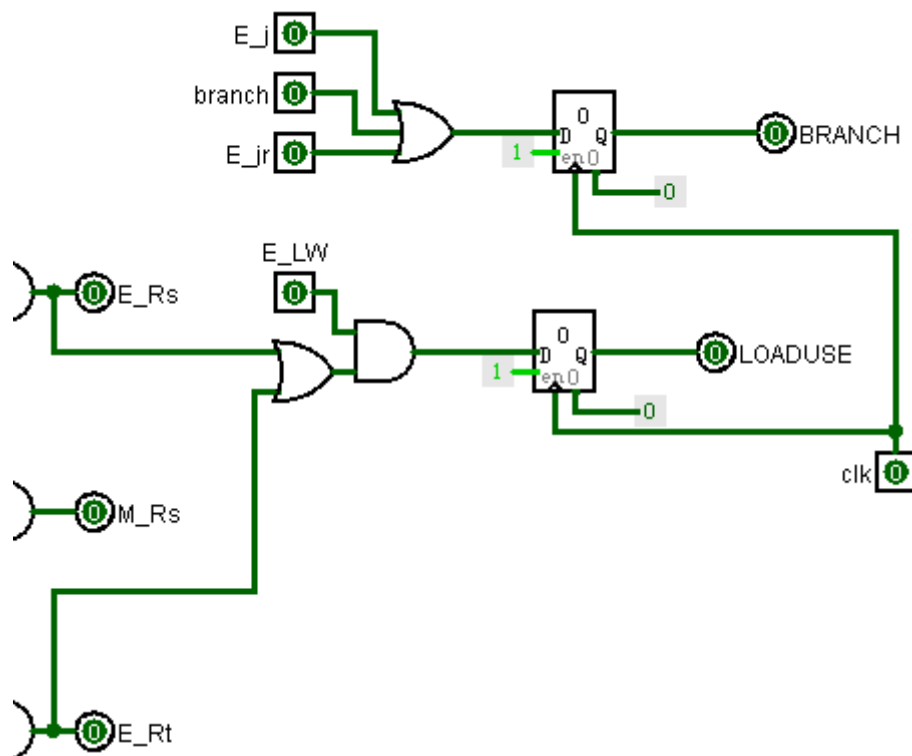


图 3.11 LOADUSE 与 BRANCH 信号产生电路

经过分析可得，当 LOADUSE 信号来时暂停 IF、ID 段，清空 EX 段，当 BRANCH 信号来时清空 ID 与 EX 段。故相应地给出指令寄存器、缓冲接口 IF/ID 与缓冲接口 ID/EX 的使能信号与清零信号即可。以缓冲接口 IF/ID 为例，电路如下

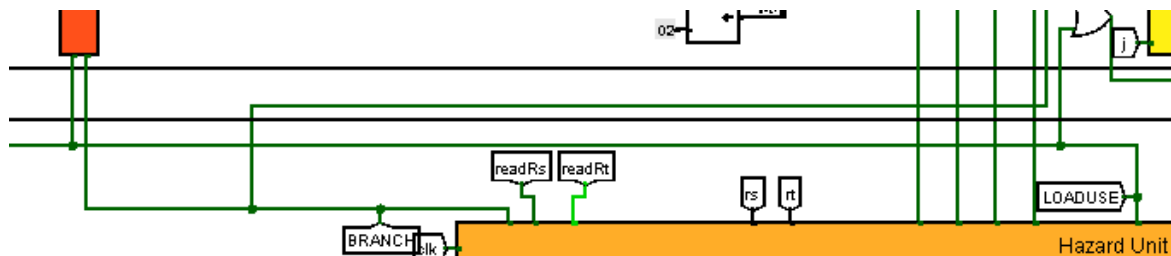


图 3.12 LOADUSE 与 BRANCH 信号作用电路

## 4 实验过程与调试

### 4.1 测试用例和功能测试

本次测试主要包括多级嵌套中断的测试、理想流水线的测试以及解决各类冒险后的流水线测试。

#### 4.1.1 多级嵌套中断测试

在此只给出一种中断测试用例的说明：首先在执行主程序时按 2 号中断，接着在 2 号中断执行过程中按 3 号中断，最后在 3 号中断执行过程中按 1 号中断。

预期会先响应 2 号中断，当 3 号中断来临后执行 3 号中断，当其执行完后回到 2 号中断的断点处继续执行，直到 2 号中断执行完后才执行 1 号中断并在结束后返回主程序的断点处继续执行。

其中，1 号与 2 号中断皆是跑马灯，3 号中断是测试扩展指令（具体代码见 3.2.2 节）。测试结果部分结果见下图 4.1、图 4.2 和图 4.3 所示

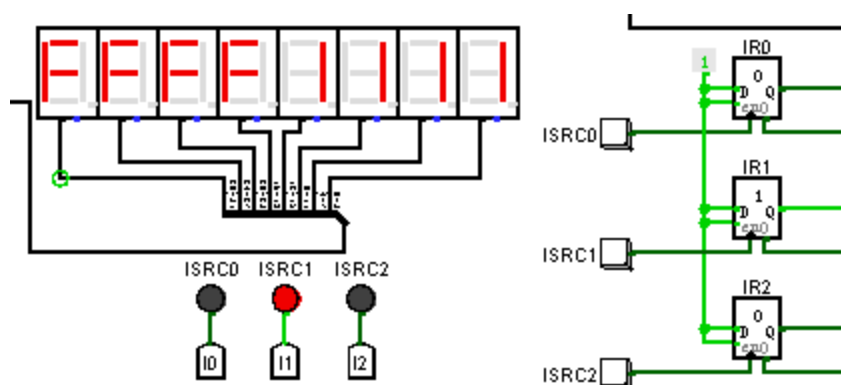


图 4.1 执行主程序时中断来临

如图 4.1 所示，在执行主程序时中断 2 发出请求信号，此时主程序转到中断处理程序，符合预期。

在执行 3 号中断时 1 号中断请求到来，但由于优先级的限制，1 号中断一直等待，图 4.2 显示的是在从 3 号中断返回 2 号中断后 1 仍然在等待。

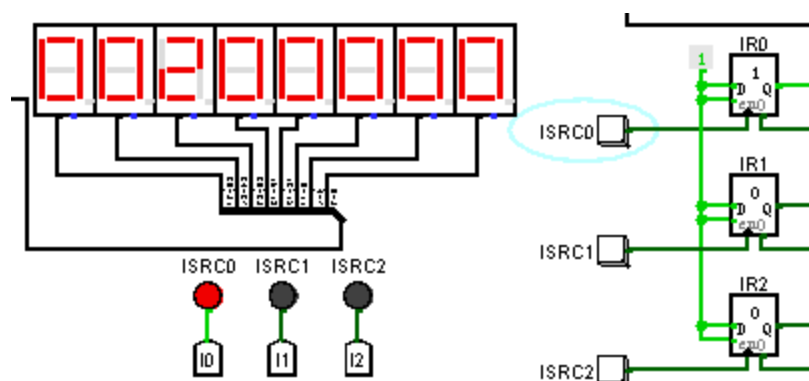


图 4.2 中断 1 等待

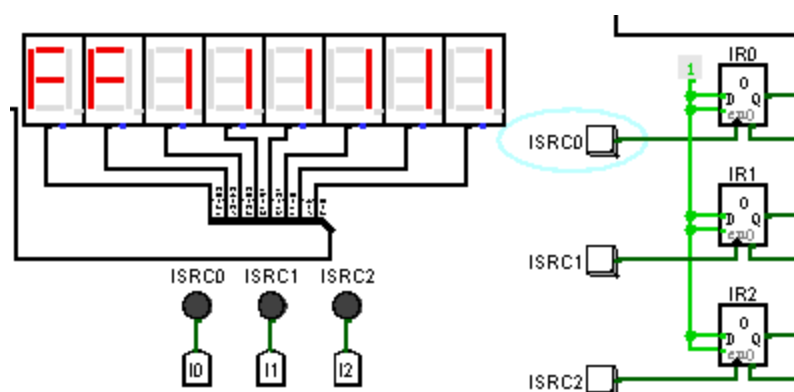


图 4.3 中断结束回到主程序

由以上，可见测试成功，符合预期。

## 4.1.2 理想流水线测试

直接采用老师所给的测试文件进行测试，得到如下图 4.4 结果统计。

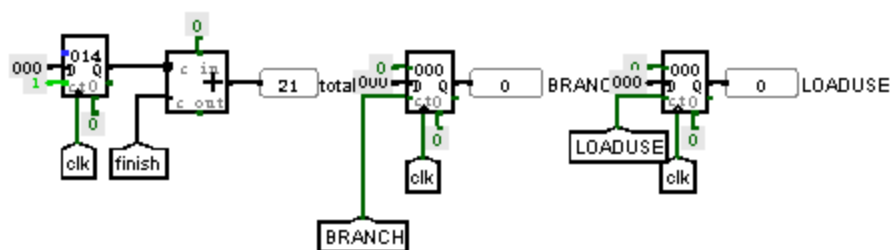


图 4.4 理想流水线测试统计结果

与标准比较后发现正确，符合预期。

## 4.1.3 冲突解决后流水线测试

直接采用老师给出的测试文件，测试结果依次见下

# 华中科技大学课程设计报告

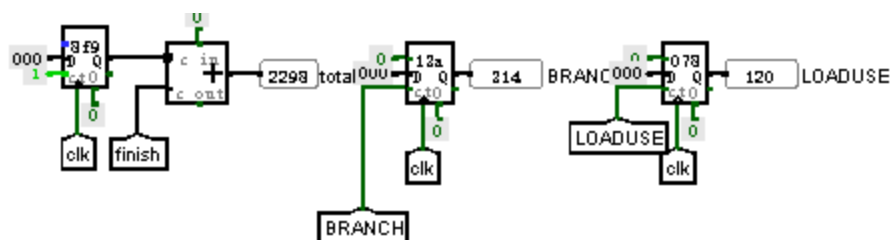


图 4.5 benchmark 测试结果

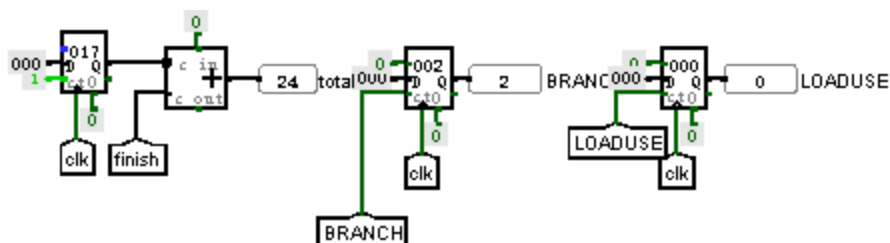


图 4.6 B 指令测试结果

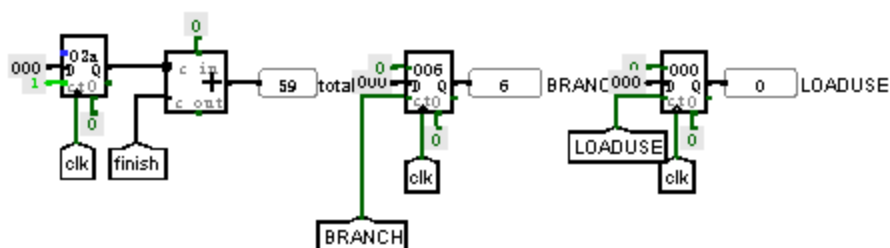


图 4.7 J 指令测试结果

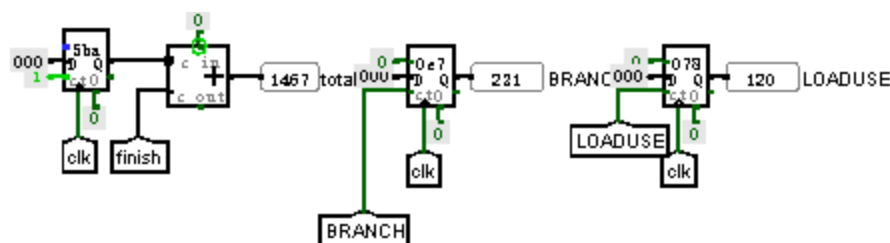


图 4.8 排序测试结果

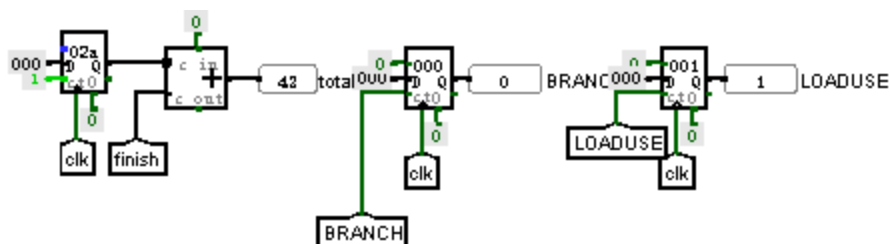


图 4.9 数据相关测试结果

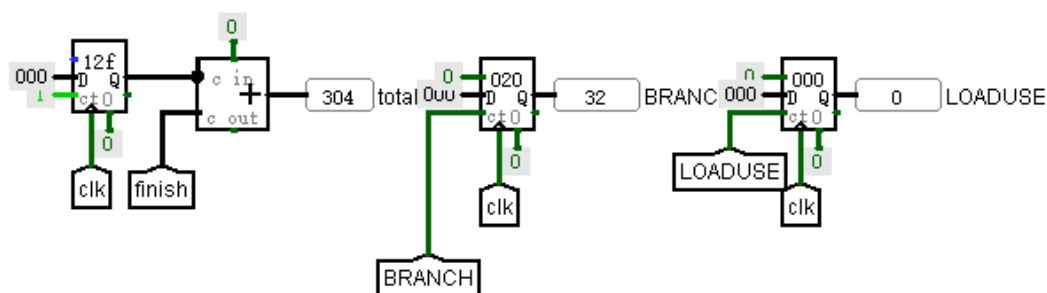


图 4.10 移位测试结果

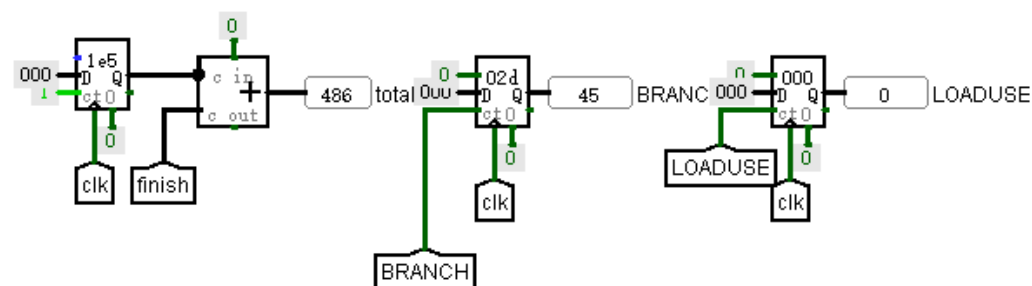


图 4.11 走马灯测试结果

另外，排序之后内存数据如下

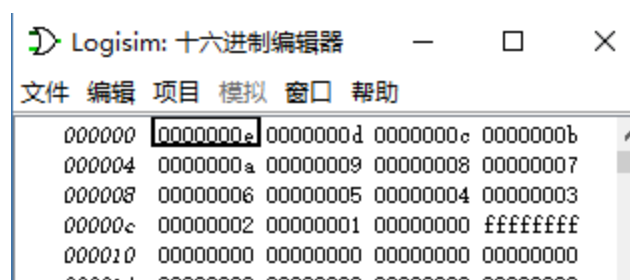


图 4.12 排序后内存数据

由以上结果可见，测试成功，符合预期。

## 4.2 流水线 FPGA 实现

将 logisim 上的电路图用 verilog 语言重写，通过仿真后生成比特流并烧录到板子上。

上板测试通过，可以正确执行 benchmark 及扩展指令。

## 4.3 性能分析

由于指令本身的特点，单周期方案无疑是时钟周期数最少的，但其时钟周期受用时最长的那条指令限制，实际执行时间反而会长。

采用 5 段流水的设计方案，若是各段的延迟平衡合理，则能够有效解决上述问题，

# 华中科技大学课程设计报告

虽然总的时钟周期数会增加但实际用时可能会得到降低。

在 5 段流水线的设计中影响比较大的就属在哪一段进行分支跳转，因为在 ID 段跳转，误取深度为 1，EX 段误取深度为 2，MEM 段误取深度为 3。故在 ID 段进行分支跳转性能较高，但是考虑到在 ID 段跳转就需提前进行比较运算获得相关运算，会增大电路实现的难度，同时分段的清晰性会降低，故选择折中的 EX 段进行分支跳转是一个比较好的选择。

## 4.4 主要故障与调试

### 4.4.1 srav 与 xor 指令故障

**故障现象：**指令执行完后 rd 对应的寄存器值未改变

**原因分析：**此两条指令都需要更改寄存器的值，故此时的 RegWrite 信号应为 1 使得时钟上升沿来临时更改寄存器的值。

**解决方案：**更改 RegWrite 控制信号的产生电路使得执行 srav 与 xor 指令时 RegWrite 为 1，即对该产生信号的或门增添输入（srav 与 xor 的产生信号）。

### 4.4.2 中断返回故障

**故障现象：**高优先级的中断处理程序在恢复屏蔽字并开中断后且返回之前，若此时被中断则无法正确返回到该返回的位置

**原因分析：**中断处理程序先获得 epc 再开中断，之后已经无法再恢复 epc 故无法跳转到正确位置。

**解决方案：**

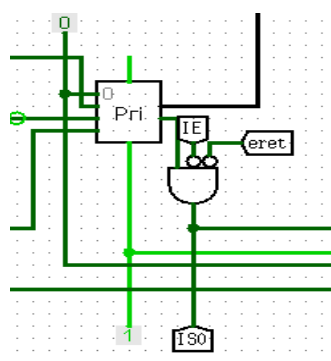


图 4.13eret 故障解决图

开中断后先将 `eret` 执行完再进行下一次中断，即在 `IE` 之外再增加 `eret` 控制信号来控制是否响应中断。如上图 4.13 所示

## 4.4.3 屏蔽字恢复故障

**故障现象：**屏蔽字未恢复，如下图 4.14 所示。

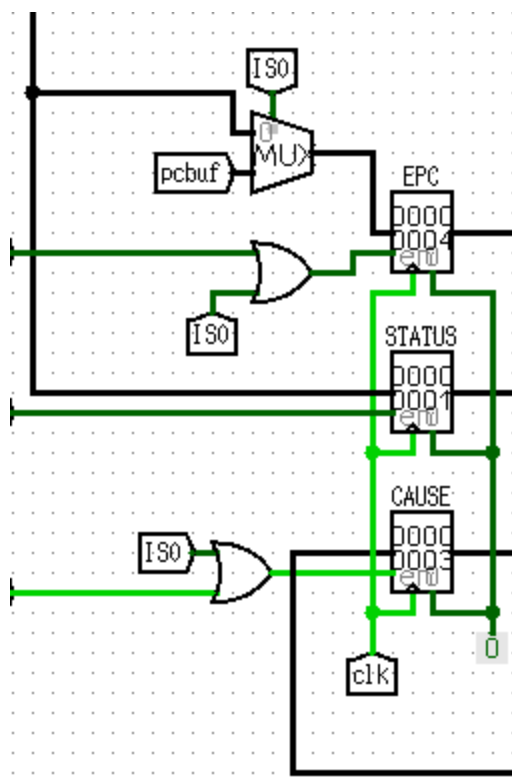


图 4.14 无法恢复屏蔽字示意图

**原因分析：**`CAUSE` 寄存器输入端漏加一个输入信号，只能保存一次中断类型而无法恢复。

**解决方案：**增添输入数据 `GPR[rt]`，并使用数据选择器选择，当需要对 `CAUSE` 寄存器进行改写时则选择 `GPR[rt]`

## 4.4.4 屏蔽字故障

**中断：**屏蔽字恢复问题。

**故障现象：**从中断程序返回时屏蔽字并没有正确恢复为上一次的而是保留在刚结束的中断的屏蔽字。。

**原因分析：**用软件保存屏蔽字时保存了当前要进入的中断的屏蔽字而不是上一

次的屏蔽字，故恢复的时候错误。

错误代码：

```
# get cause number
mfc0 $t0,$3
# store cause
sw $t0,($fp)
addi $fp,$fp,-4
addi $sp,$sp,-4
```

**解决方案：**修改中断处理程序，保存上一次的屏蔽字(若上一段程序为主程序则保存 0)。

修改代码：

```
# store cause
addi $t1,$zero,-20
beq $sp,$t1,init
sw $t0,($fp)
addi $fp,$fp,-4
addi $sp,$sp,-4
bne $t1,$zero,open
init:
addi $t1,$zero,0
sw $t1,($fp)
addi $fp,$fp,-4
addi $sp,$sp,-4
```

#### 4.4.5 仿真故障

**故障现象：**仿真图多处出现与预期不符的现象

**原因分析：**有一些变量的某个字母写错，但 vivado 自动生成了一个一位的线网变量，导致与设计不符。

**解决方案：**查看 warnings，找出写错的变量并改正



# 华中科技大学课程设计报告

## 4.5 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	阅读任务书，完成了 4 条指令的扩充与测试
第二天	查阅资料完成中断机制并着手实现
第三天	初步设计中断电路，编写中断处理程序
第四天	实现多级嵌套中断机制，完善中断处理程序并进行测试，记录故障及修正，开始研究流水线
第五天	初步完成理想流水线并进行简单测试
第六天	完成重定向部分，正处理分支和 load_use
第七天	实现 logisim 平台上全冒险处理机制的 CPU 并进行测试，开始进行 FPGA 平台设计
第八天	进行 FPGA 平台各模块的编写
第九天	大体完成 verilog 的编写
第十天	调试 verilog，完成 5 段流水线上板实现

## 5 设计总结与心得

### 5.1 课设总结

本次组成原理课程设计，主要作了如下几点工作：

- 1) 实现了 4 条指令的扩充，完成了扩充指令的测试。
- 2) 实现了简易的协处理器 CP0，实现了与中断相关 3 条指令。
- 3) 实现了支持多级嵌套的中断机制。
- 4) 完成了 5 段流水线的设计。
- 5) 实现了插气泡、重定向等方式解决数据冲突与分支冲突。
- 6) 完成了 5 段流水线的 FPGA 实现。

### 5.2 课设心得

本次课程设计是继上学期实验之后的一次更深层次的实践。

通过本次课设，我加深了对单周期 CPU 实现机制的理解，实践了单周期 CPU 的中断机制，对于中断机制的原理、中断屏蔽的硬件实现和中断服务程序的编写从原来的纯理论印象逐步做到能够进行实践。

将单周期 CPU 改为五段流水线时，首先考虑的是如何分段问题，弄清哪些事情在哪个段做，发现有些部件（如选择器）既可放在 ID 段也可以放在 EX 段，初次想的时候感觉放在哪里都无所谓并不影响功能，细想之后联系到五段流水的原理，明白了需要尽量平衡各段的延迟，不让某一段成为瓶颈。最后经历了重定向设计部分之后综合考虑各段延迟与电路实现的复杂性做出最终的分段方案。由此觉得无论做什么都应当考虑后续的扩展与修改，否则后面的更改会变得比较麻烦。

Verilog 编程后仿真出现了大量错误，一查才发现有许多的变量不小心写错了一个字母但软件自动生成了一个一位的新的变量导致与原计划不符，也正因如此，这些问题并不会报错只是单纯的 warning，而我完全没有看 warning 的习惯，导致了这次的重大错误，浪费了大量时间进行不断地仿真以查找错误变量，后来才发现 warning 中都有记录，自此懂得了 vivado 编程一定要好好注意 warning，排除所有可能的错误。此次经历纠正了一直以来在 vivado 编程中的不足之处，同时也为别的编

# 华中科技大学课程设计报告

---

程提了醒，需要多留意 warning

最后还是比较遗憾只做到了流水线上板，流水线中断和分支预测都未进行实现，需要更好地安排时间，提升效率，这在以后的学习与生活中都应该注意与改进。

总的来说，这次课程设计难度适中，我觉得这样的动手实践无论哪一门课都应当多多安排，最好是能够在实践过程中进行理论知识的传授，这样学习效果会得到极大的提升。

然而对于本次课程设计，我还有一些小小的建议和改进。本次课程设计的分组机制略显不合理，因为本次课程设计主要是单人实现不允许共享代码，这样分组的意义更多是在于讨论互助，而这即使不分组大家也会这么做。

最后在这里也感谢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] 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.
- [5] 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 •

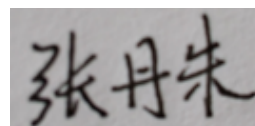
---

### 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：

A rectangular box containing a handwritten signature in black ink. The signature is written in a cursive style and appears to read '张丹朱' (Zhang Danzhu).