

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： CS1401

学 号： U201414573

姓 名： 朱珺

电 话： 18040506495

邮 件： 412199815@qq.com

完成日期： 2017-2-26



计算机科学与技术学院

华中科技大学课程设计报告

目 录

1	课程设计概述.....	3
1.1	课设目的	3
1.2	设计任务	3
1.3	设计要求	3
1.4	技术指标	4
2	总体方案设计	6
2.1	单周期 CPU 设计	6
2.2	中断机制设计	9
2.3	流水 CPU 设计	9
2.4	气泡式流水线设计	10
2.5	数据转发流水线设计	11
3	详细设计与实现	12
3.1	流水线 CPU 实现	12
3.2	中断机制实现	20
4	实验过程与调试	23
4.1	测试用例和功能测试	23
4.2	VIVADO 仿真调试主要参数	26
4.3	性能分析	27
4.4	主要故障与调试	27
4.5	实验进度	28
5	设计总结与心得	29
5.1	课设总结	29
5.2	课设心得	29

华中科技大学课程设计报告

参考文献.....	31
-----------	----

1 课程设计概述

1.1 课设目的

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

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

1.2 设计任务

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

1.3 设计要求

- (1) 根据课程设计指导书的要求，制定出设计方案；
- (2) 分析指令系统格式，指令系统功能。
- (3) 根据指令系统构建基本功能部件，主要数据通路。
- (4) 根据功能部件及数据通路连接，分析所需要的控制信号以及这些控制信号的有效形式；
- (5) 设计出实现指令功能的硬布线控制器；

华中科技大学课程设计报告

- (6) 调试、数据分析、验收检查;
- (7) 课程设计报告和总结。

1.4 技术指标

- (8) 支持表 1.1 前 27 条基本 32 位 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	
13	NOR	或非	

华中科技大学课程设计报告

#	指令助记符	简单功能描述	备注
14	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	XORI	异或立即数	
29	LUI	立即数加载至至高位	
30	LH	加载半字	
31	BGEZ	大于等于零转移	

2 总体方案设计

2.1 单周期 CPU 设计

在实验 3 完成的 CPU 基础上进行修改，增加的指令除了要对已有的控制信号进行赋值，还要增加一个立即数加载至高位指令信号、一个加载半字指令信号、一个大于等于零跳转信号，另外还有 3 个关于中断对于 CPU 操作的信号。

总体结构图如图 2.1 所示。

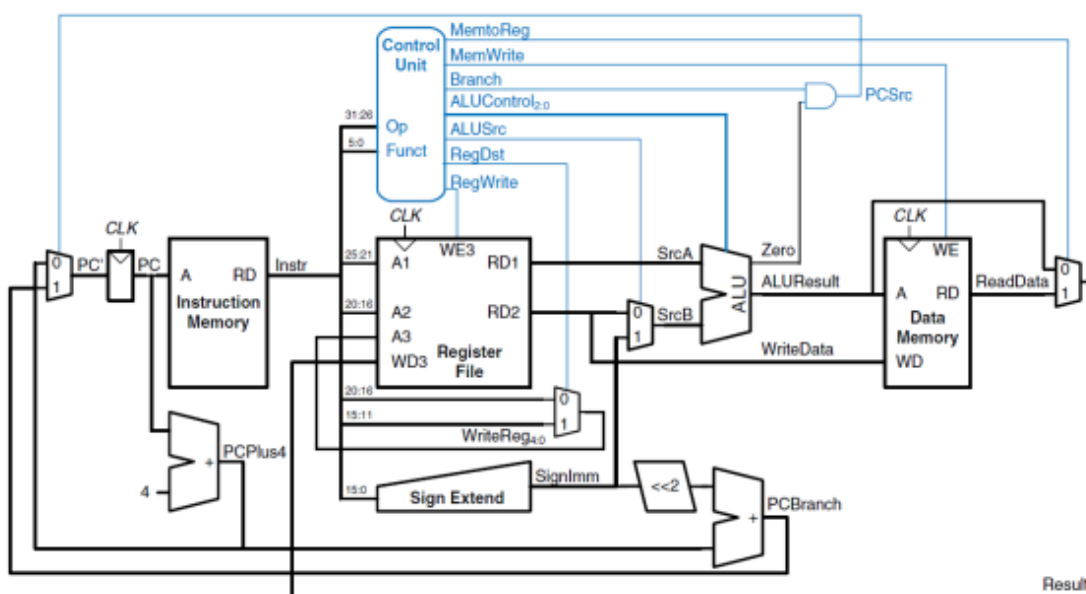


图 2.1 总体结构图

2.1.1 主要功能部件

运算器部分，详见实验 1。

寄存器部分，详见实验 2。

控制信号部分，将指令分位后进行相对应控制信号的判断。

指令寄存器，ROM。

数据寄存器，RAM。

华中科技大学课程设计报告

1. 程序计数器 PC

PC 采用下降沿触发，写使能信号恒为 1。

2. 指令存储器 IM

取计数器的 2-11 位，以 PC+4 位跳转单位，因为指令的长度为 32 位，4 个字节。

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

2.1.2 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 2.2 控制信号、表 2.3 控制信号（续）。

华中科技大学课程设计报告

表 2.2 控制信号

控制信号	C14	C13	C12	C11	C10	C9	C8
有效值	1	1	1	1	1	1	1
标签 (隧道)	Ctl14	Ctl13	Ctl12	Ctl11	Ctl10	Ctl9	Ctl8
功能	当为 1 时将 PC+8 送入寄存器	是否为停机指令	PC 是加载顺序逻辑值还是寄存器中的值	JAL 指令时选择输入进 r1 的是固定 31 号还是指令地址	选择是相等跳转还是不相等跳转	选择 Rt 还是 Rd 作为 R2 的编号值	是否允许向寄存器写数据

表 2.3 控制信号 (续)

控制信号	C7	C6	C5	C4	C3	C2	C1
有效值	-	1	1	1	1	1	1
标签 (隧道)	Ctl7	Ctl6	Ctl5	Ctl4	Ctl3	Ctl2	Ctl1
功能	当指令为非 R 型的时候选择 ALU 的 op 内容	ALU 的 y 输入端是选择立即数还是寄存器	是否将数据加载入数据寄存器	是否从数据存储器中加载数据	是否从数据存储器中取数据	判断是否是 Jmp 指令	控制跳转立即数跳转

2.2 中断机制设计

2.2.1 总体设计

按下中断按钮后通过电路进行判断此时应该排队还是进行中断，若可以进行则保存当前 PC 值再将对应的中断号地址送入 PC，然后编写程序保存线程并执行中断程序。

2.2.2 硬件设计

建立一个 CP0 寄存器储存执行中断前的 PC 值与当前的屏蔽字，再在中断产生的地方进行判断按下的中断是否被屏蔽，若没有则将信号 INT 置为 1，执行硬件保存现场并修改 PC。

2.2.3 软件设计

将 CP0 的内容保存到数据寄存器中，再向写入新的 PC 与屏蔽字，执行中断，恢复屏蔽字和 PC 值，返回到源程序中。

2.3 流水 CPU 设计

2.3.1 总体设计

将 CPU 分成 5 个阶段，取指令、读指令、计算、数据储存、写回，每个阶段仅完成指令的一部分内容，以达到多条指令并发执行的效果。

2.3.2 流水接口部件设计

1. IF/ID

连接取指令与读指令的器件。

2. ID/EX

连接读指令与计算部分的部件，主要负责信号传递以及寄存器数据传递。

华中科技大学课程设计报告

3. EX/MEM

连接计算部分和数据存储部分的部件，主要负责信号传递和计算结果传递，还有分支成功后的地址传递。

4. MEM/WB

连接数据存储部分和写回部分的部件，主要负责信号的传递和可能写回的数据传递。

2.3.3 理想流水线设计

不考虑数据冲突问题，将上面 4 个寄存器组器件搭建起来，然后从控制信号处生成信号，向后传递，在需要此信号的阶段将此信号分出来，如图 2.2 信号的控制所示，大体流程如图 2.3 流水线指令流程所示。

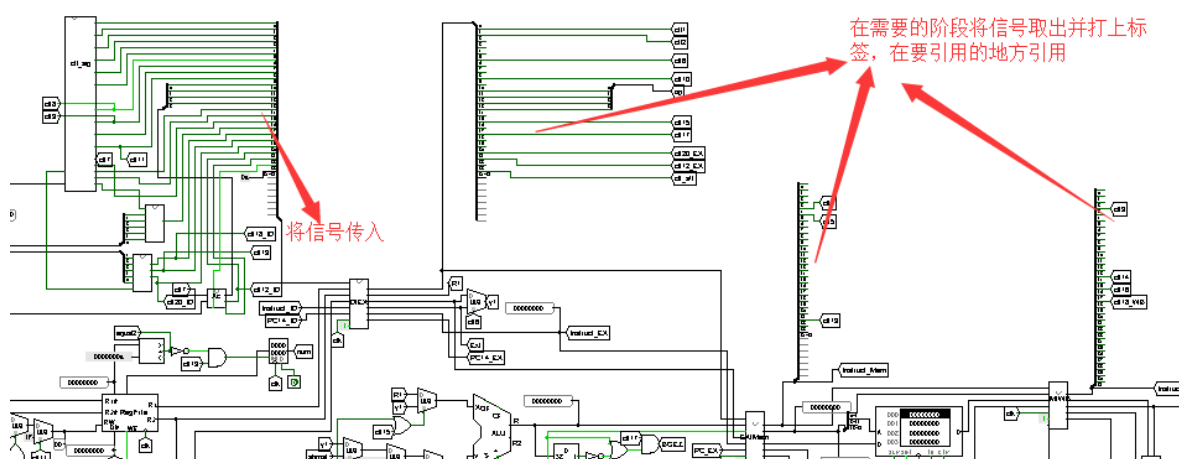


图 2.2 信号的控制

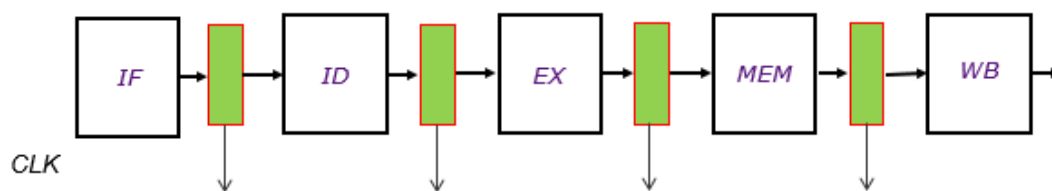


图 2.3 流水线指令流程

2.4 气泡式流水线设计

新建一个控制器，将阶段 ID 和 EX 的指令分段传入，比较两条指令数据是否相

华中科技大学课程设计报告

关(具体来说 EX 阶段的写是否与 ID 阶段的读相关)。若相关, 则生成控制信号将 ID 与 EX 间插入两段空指令, 并将 PC 值停止。这么做是为了将写指令执行完然后再进行读指令。如图 2.4 气泡流水线所示。

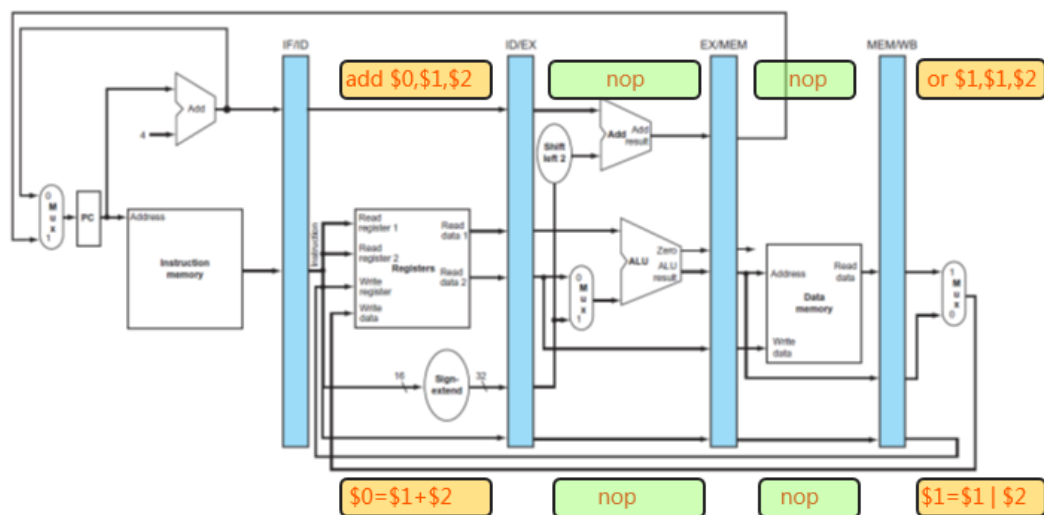


图 2.4 气泡流水线

2.5 数据转发流水线设计

后面 4 个阶段 ID 和 EX、EX 和 MEM、MEM 和 WB 进行比较, 若检测到有数据相关, 则当场把生成的数据传到读的地方去。特殊情况如图 2.5 load-use 冲突所示, 执行段路径时间过长, 再进行写回的话会延缓指令的执行时间, 因此正确做法是插入一个气泡。

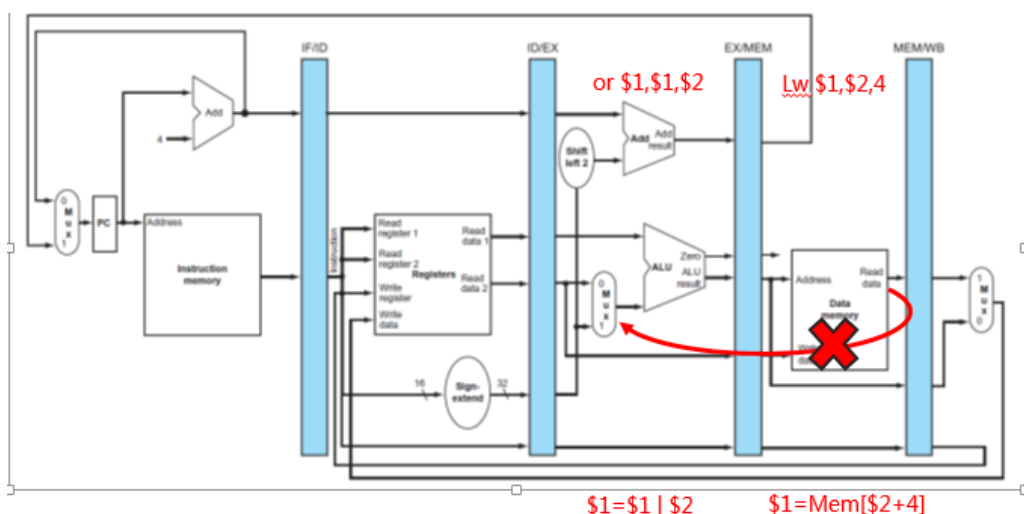


图 2.5 load-use 冲突

3 详细设计与实现

3.1 流水线 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。

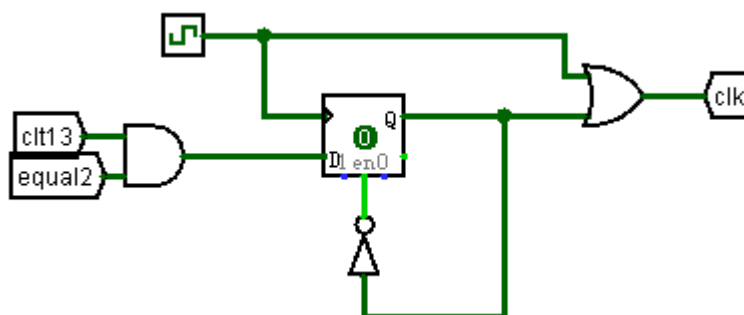


图 3.1 程序计数器 (PC)

FPGA 实现:

使用组内成员编写的寄存器进行实现。

程序计数器 Verilog 代码如下:

```
register_nbits #(32)
pc_count(~clk, ~clr, ~load_use, pc_final, pc_if),
```

图 3.2 PC 计数器

参数说明: ~clk 表示下降沿触发, ~clr 表示寄存器为 0 时清零, 由 load_use 信号确定是否可写入, 当有 load_use 信号时, 要插入气泡所以不可写入。Pc_final 代表写入 PC 寄存器的值, pc_if 为输出的 pc 值。

2) 指令存储器 (IM)

华中科技大学课程设计报告

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 3.3 所示。

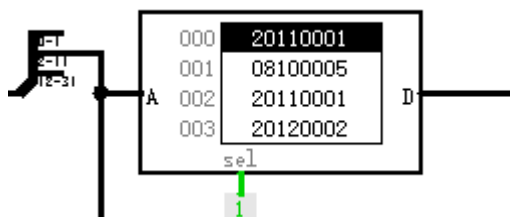


图 3.3 指令存储器 (IM)

② FPGA 实现:

使用组内成员编写的寄存器进行实现。

rom 的 Verilog 代码如下:

```
rom rml(pc_if[11:2], instruct_if[31:0]);
```

图 3.4 rom

3) 阶段寄存器 (ID/EX)

① Logism 实现:

如图 3.5 ID/EX, 类似于 regfile 寄存器组, 但写入信号与 clr 信号是同步的, 每一个寄存器都要执行的, 每当时钟到来的时候根据输入更新寄存器的值, 当清空信号来的时候异步清空内容值。

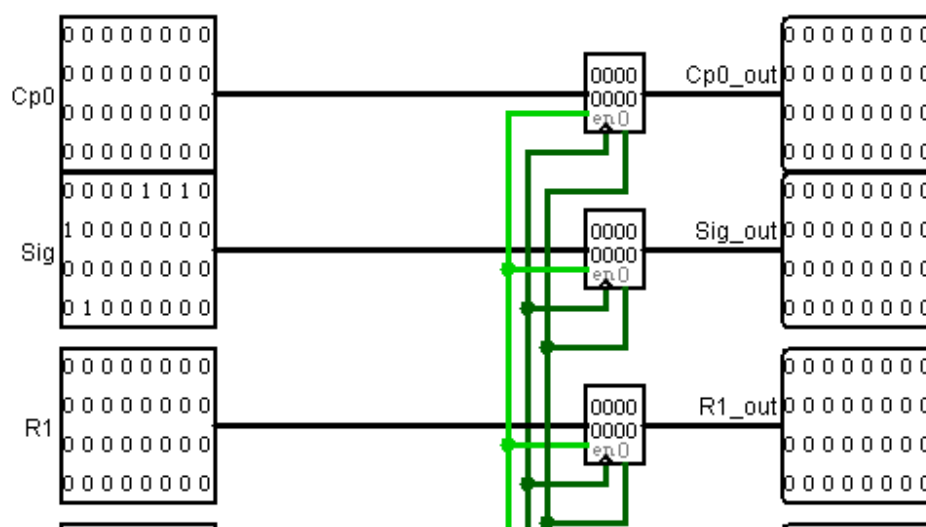


图 3.5 ID/EX

② FPGA 实现:

ID/EX Verilog 代码如下:

```

register_nbits #(1) d_flip3(clk, Load_use, 1'b1, Load_use, Q3);

or          o1(bubble, Q3, J);

register_nbits #(32) sig(clk, ~bubble, WE, Sig, Sig_out),
                r1(clk, ~bubble, WE, R1, R1_out),
                r2(clk, ~bubble, WE, R2, R2_out),
                ext(clk, ~bubble, WE, Ext, Ext_out),
                instruction(clk, ~bubble, WE, Instruction, Instruction_out),
                pc(clk, ~bubble, WE, PC, PC_out);
    
```

图 3.6 ID/EX 的 verilog 实现

其他的 3 个阶段寄存器的原理与此一样，在此不再赘述。

4) 数据相关检测器

① Logism 实现:

如图 3.7 数据相关检查，主要分为两个部分，一个是解析指令，判断指令是否是读写指令以及需要读或者需要写哪一条指令，另一个是比较器，比较需要比较的编号内容是否相等。参数为：1 条读指令，3 条写指令，输出为 3 条写指令相关与 1 条读指令的相关性。如表 3.1 所示为每条指令的读写情况。

华中科技大学课程设计报告

表 3.1 指令读写寄存器

指令	读寄存器	写寄存器
Add	Rs rt	Rd
Addi	Rs	rt
Sll	Rt	Rd
Sra	Rt	Rd
Srl	Rt	Rd
Sub	Rs Rt	Rd
Or	Rs Rt	Rd
Ori	Rs	Rt
NOR	Rs Rt	Rd
LW	M[base+offset]	Rt
SW	Rt	M[base+offset]
BEQ	Rs Rt	\\
BNE	Rs Rt	\\
SLT	Rs Rt	Rd
SLTI	Rs	Rt
SLTU	Rs Rt	Rd

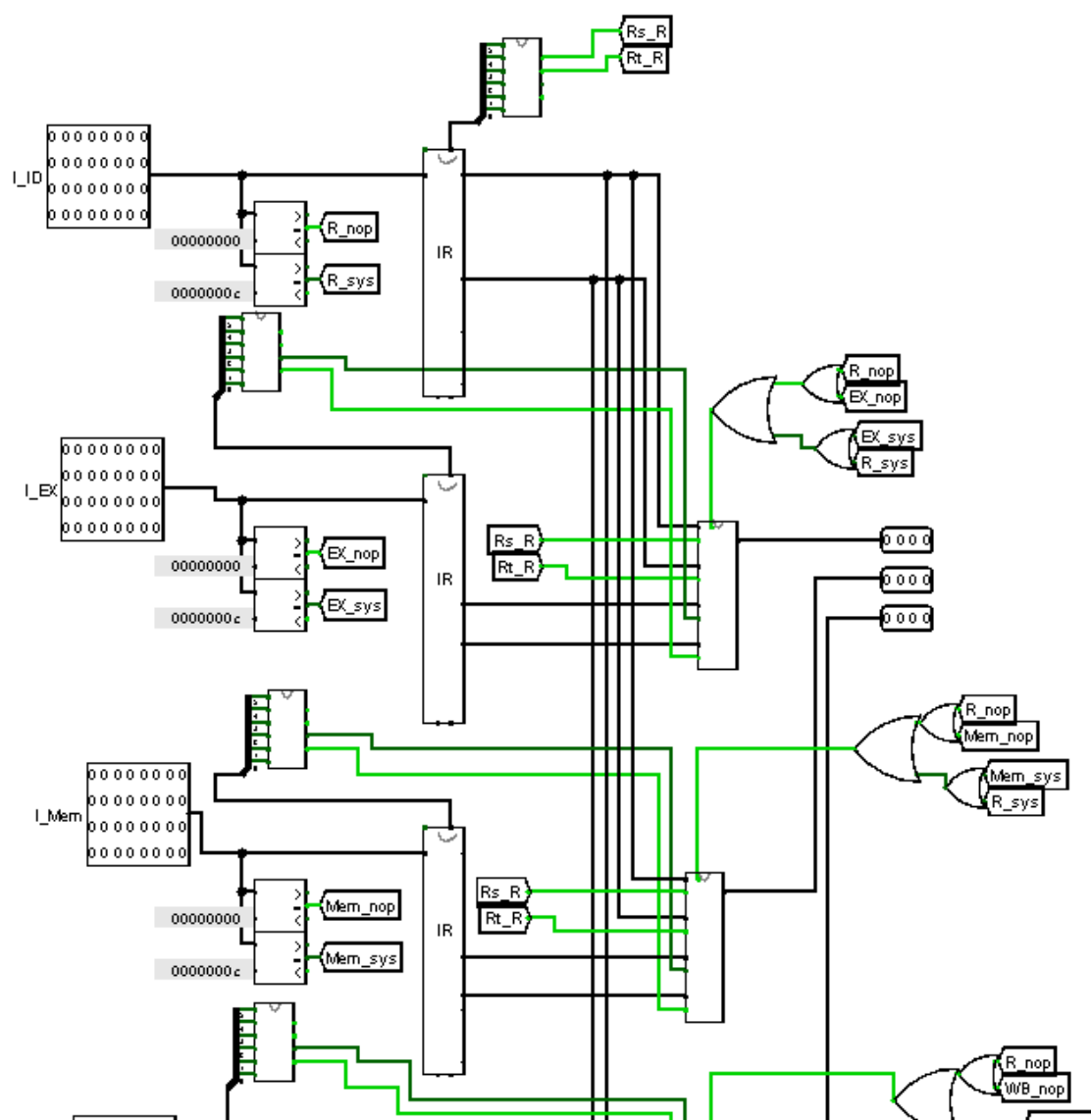


图 3.7 数据相关检查

② FPGA 实现:

数据相关检测 Verilog 代码如下:

```
Relative_Check2 rc21(I_R, I_W1, R1),
                rc22(I_R, I_W2, R2),
                rc23(I_R, I_W3, R3);
```

图 3.8 数据相关检测的 verilog 实现

Relative_Check2 为封装的电路，包含比较与判断是读写哪个寄存器两个功能，但只输入一个读一个写，读都为 I_R，后面 3 个 I_W 是不同的阶段所代表的指令。

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

华中科技大学课程设计报告

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 3.2 所示。

表 3.2 指令系统数据通路表

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

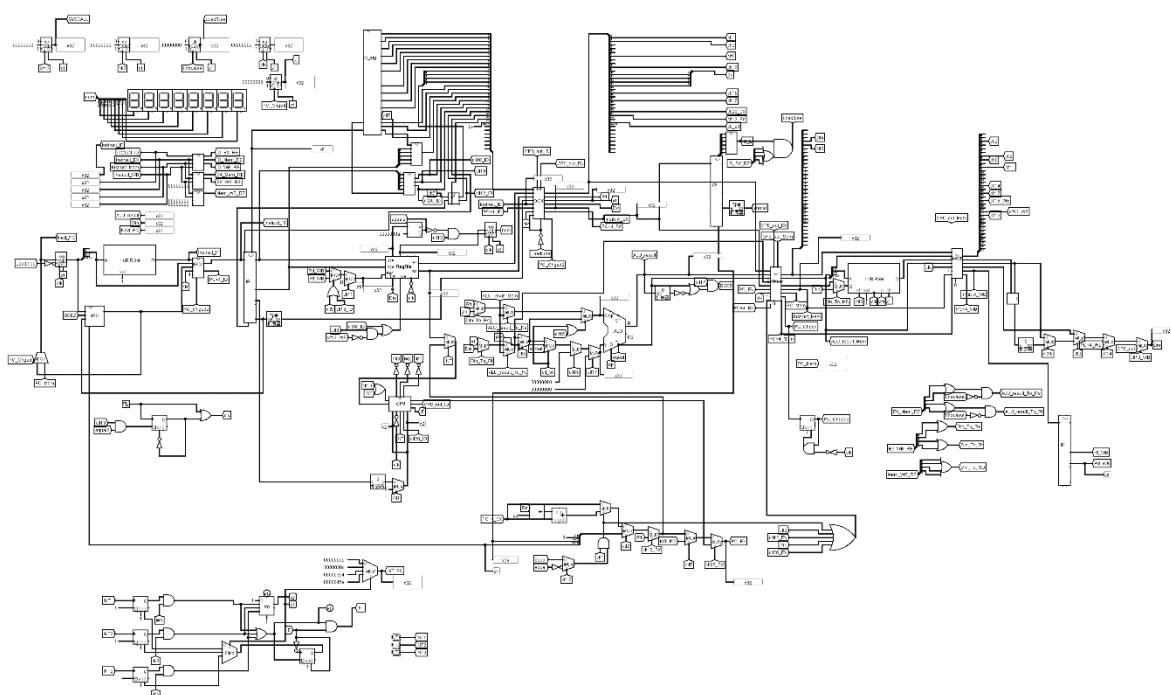


图 3.9 流水线 CPU 数据通路 (Logism, 可放大观看)

在 Vivado 中使用 Verilog 语言搭建的数据通路的原理图如图 3.10 所示。

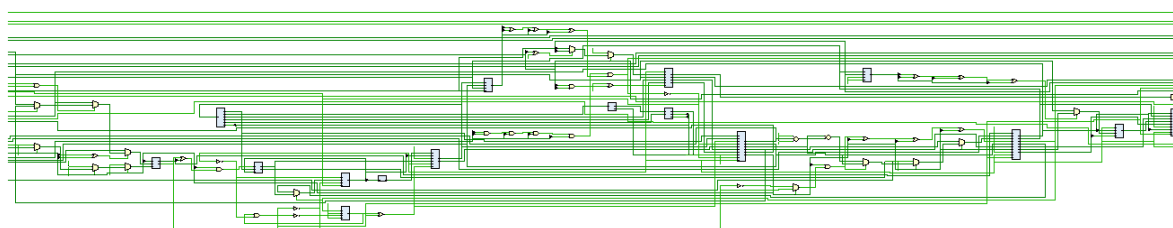


图 3.10 单周期 CPU 数据通路 (FPGA)

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

对照表 3.3 所示。

华中科技大学课程设计报告

表 3.3 指令信号控制

指令	R	RW	WE	X	EXT	Y	ALUop	MemWrite	MemRead	Din	Branch	SYSCALL
ADD	00	00	1	0	0	00	0101	0	0	00	00	0
ADDI	00	10	1	0	0	10	0101	0	0	00	00	0
ADDIU	00	10	1	0	0	10	0101	0	0	00	00	0
ADDU	00	00	1	0	0	00	0101	0	0	00	00	0
AND	00	00	1	0	0	00	0111	0	0	00	00	0
ANDI	00	10	1	0	1	10	0111	0	0	00	00	0
SLL	00	00	1	1	0	01	0000	0	0	00	00	0
SRA	00	00	1	1	0	01	0001	0	0	00	00	0
SRL	00	00	1	1	0	01	0010	0	0	00	00	0
SUB	00	00	1	0	0	00	0110	0	0	00	00	0
OR	00	00	1	0	0	00	1000	0	0	00	00	0
ORI	00	10	1	0	1	10	1000	0	0	00	00	0
NOR	00	00	1	0	0	00	1010	0	0	00	00	0
LW	00	10	1	0	0	10	0000	0	1	10	00	0
SW	00	00	0	0	0	10	0000	1	0	00	00	0
BEQ	00	00	0	0	0	00	0000	0	0	00	01	0
BNE	00	00	0	0	0	00	0000	0	0	00	01	0
SLT	00	00	1	0	0	00	1011	0	0	00	00	0
SLTI	00	10	1	0	0	10	1011	0	0	00	00	0
SLTU	00	00	1	0	0	00	1100	0	0	00	00	0
J	00	00	0	0	0	00	0000	0	0	00	10	0
JL	00	01	1	0	0	00	0000	0	0	01	10	0
JR	00	00	0	0	0	00	0000	0	0	00	11	0
SYSCALL	11	00	0	0	0	11	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:

```
assign #1 ctl[15] = ~op[31]&~op[30]&op[29]&op[28]&op[27]&op[26];
```

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错误!未找到引用源。所示。

3.2 中断机制实现

3.2.1 中断生成电路

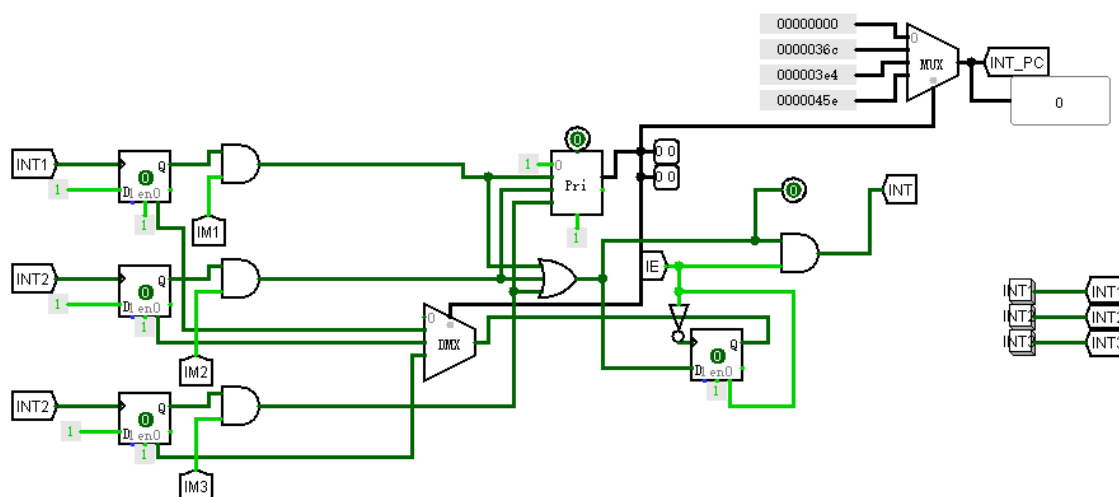


图 3.11 中断信号生成电路

生成逻辑: 当按下按钮, 首先判断是否被屏蔽 (屏蔽字由 CP0 读出), 若被屏蔽, 则保持当前值不变, 等待其屏蔽字变为 0, 若不被屏蔽, 则将此寄存器清零并生成中断信号并传入中断地址。

3.2.2 中断执行电路

当接受到中断信号的时候, 将 PC 值保存至 CP0 寄存器中, 并在分支跳转电路中加上条件: 当中断来时将中断产生电路送过来的地址值作为 PC 的下个值进行跳转。并编写信号与中断指令使得软件可以对 CP0 进行读写操作。

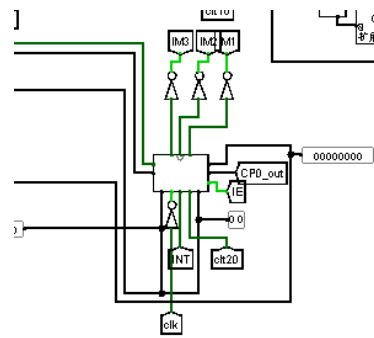


图 3.12 CP0 图

3.2.3 中断程序

中断程序的任务：1) 保存现场，将 CP0 中的 PC 值和屏蔽字取出送入数据寄存器保存，修改屏蔽字，令其为当前中断的对应屏蔽字。2) 执行对应中断程序。3) 恢复现场与屏蔽字。如下为中断程序 1 的内容，显示灯的内容为数字 1 从右向左移动。

```
inter_1_pro:
mfc0 $a0, $2

sw $s1, 0($zero)

sw $s2, 4($zero)

sw $s3, 8($zero)

sw $a0, 12($zero)

sw $v0, 16($zero)

sw $1, 20($zero)

li $1, 1

mtc0 $1, $0

li $a0, 1

mtc0 $a0, $1

addi $v0, $zero, 0

addi $s1, $zero, 1

addi $s2, $zero, 0

addi $s3, $zero, 8

inter_1_loop:

addi $a0, $s1, 0
```

华 中 科 技 大 学 课 程 设 计 报 告

```
syscall  
  
sll $s1,$s1,4  
  
addi $s2,$s2,1  
  
bne $s2,$s3,inter_1_loop  
  
li $a0,0  
  
mtc0 $a0,$0  
  
li $a0,0  
  
mtc0 $a0,$1  
  
lw $s1,0($zero)  
  
lw $s2,4($zero)  
  
lw $s3,8($zero)  
  
lw $a0,12($zero)  
  
lw $v0,16($zero)  
  
lw $1,20($zero)  
  
mtc0 $a0,$2  
  
eret
```

4 实验过程与调试

4.1 测试用例和功能测试

测试用例内容：1) misp2.asm：测试扩展指令集。2) benchmark.hex:检测正常指令是否可执行。3) benchmark1.hex：检测中断指令是否正常执行。

4.1.1 测试用例 1

扩展指令集测试文件：

```
start:
add $v0,$0,$0
lui $v0,0xffff
xori $v0,0xffff
addi $s0,$0,15
sw $s0,0($0)
lh $s0,3($0)
add $a0,$0,$s0
syscall
bgez $s0,next
j start
next:
addi $v0,$0,0xa
syscall
```

4.1.2 测试用例 2

Benchmark:

```
#####
#走马灯测试, 测试addi, andi, sll, srl, sra, or, ori, nor, syscall LED按走马灯方式来回显示数据
#####
```


华中科技大学课程设计报告

```
.text

addi $s0,$zero,1

sll $s3, $s0, 31      # $s3=0x80000000

sra $s3, $s3, 31      # $s3=0xFFFFFFFF

addu $s0,$zero,$zero  # $s0=0

addi $s2,$zero,12

addiu $s6,$0,3  #走马灯计数

zmd_loop:

addiu $s0, $s0, 1    #计算下一个走马灯的数据

andi $s0, $s0, 15

#####

addi $t0,$0,8

addi $t1,$0,1

left:

sll $s3, $s3, 4  #走马灯左移

or $s3, $s3, $s0

add    $a0,$0,$s3      # display $s3

addi   $v0,$0,34        # system call for LED display

syscall                # display

sub $t0,$t0,$t1

bne $t0,$0,left

#####

addi $s0, $s0, 1  #计算下一个走马灯的数据
```

华中科技大学课程设计报告

```
addi $t8,$0,15

and $s0, $s0, $t8

sll $s0, $s0, 28


addi $t0,$0,8
addi $t1,$0,1


zmd_right:


srl $s3, $s3, 4 #走马灯右移

or $s3, $s3, $s0


addu    $a0,$0,$s3      # display $s3
addi    $v0,$0,34       # system call for LED display
syscall                # display


sub $t0,$t0,$t1

bne $t0,$0,zmd_right

srl $s0, $s0, 28

#####

sub $s6,$s6,$t1

beq $s6,$0, exit

j zmd_loop


exit:


add $t0,$0,$0

nor $t0,$t0,$t0      #test nor ori

sll $t0,$t0,16
```

华中科技大学课程设计报告

```
ori $t0,$t0,0xffff

addu $a0,$0,$t0      # display $t0

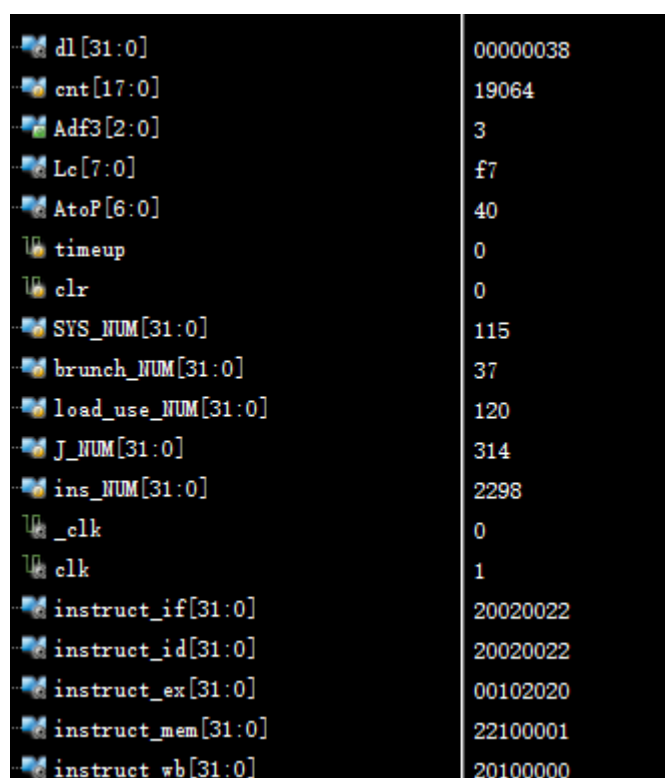
addi $v0,$0,34        # system call for LED display

syscall              # display

addi $v0,$zero,10     # system call for exit

syscall              # we are out of here.
```

4.2 Vivado 仿真调试主要参数



dl[31:0]	00000038
cnt[17:0]	19064
Adf3[2:0]	3
Lc[7:0]	f7
AtoP[6:0]	40
timeup	0
clr	0
SYS_NUM[31:0]	115
brunch_NUM[31:0]	37
load_use_NUM[31:0]	120
J_NUM[31:0]	314
ins_NUM[31:0]	2298
_clk	0
clk	1
instruct_if[31:0]	20020022
instruct_id[31:0]	20020022
instruct_ex[31:0]	00102020
instruct_mem[31:0]	22100001
instruct_wb[31:0]	20100000

图 4.1 参数列表

主要参数：数字灯 dl，sys_num 系统调用次数，brunch_num 分支指令次数，load_use_num，load_use 数，J_num 跳转指令数，clk 实际控制的时钟，instruction_if，if 阶段的指令内容，instruction_id，id 阶段的指令内容，instruction_ex，ex 阶段的指令内容，instruction_mem，mem 阶段的指令内容，instruction_wb，wb 阶段的指令内容。

4.3 性能分析

单周期 CPU 分析：有多少条指令就有多少个周期，但为了每一周期将指令执行完毕，这一周期所需时间很长，因为要读两次存储器，还要经过一次计算，最后可能还要写存储器。

冒泡流水线性能分析：总周期数为跳转指令*误区深度+数据相关指令*3(插入两个气泡)+普通指令。虽然每次周期的时间大大降低，但周期个数大幅度增加。

重定向流水线性能分析：总周期数为跳转指令*误区深度+load_use 指令*2(插入两个气泡)+普通指令。

单周期 CPU 总周期：1546，冒泡流水线总周期：4005，重定向流水线总周期：2298，误区深度为 2。

4.4 主要故障与调试

4.4.1 中断返回地址故障

故障现象：（中断 2 按下直接从零开始 benchmark ，起始地址错误）

原因分析：高优先级中断无法打断低优先级中断，原因：1.程序写错，IE 没有被改写。2.没有设置屏蔽字。

解决方案：在汇编指令中加上把 PC 读回的指令。并在中断程序刚开始的时候加上修改屏蔽字的指令。

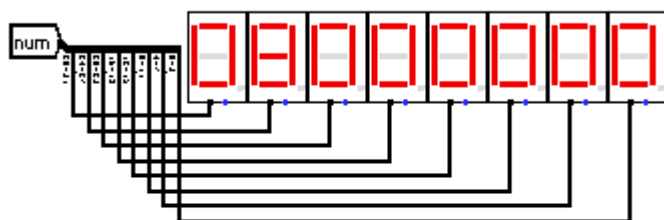


图 4.2 中断返回地址错误示意图

4.4.2 数据相关故障

故障现象：数据相关指令执行结果出错。如图 4.3 程序故障图 为指令内容。

```
6      .text
7      sort_init:
8          addi $s0,$0,-1
9          addi $s1,$0,0
10
11      sw $s0,0($s1)
```

图 4.3 程序故障图

原因分析：第三条和前两条相关，不小心将结果送入了 ALU 的操作数中(本应送入 EX/Mem 中)

解决方案：修改线路让其指向正确的方向。

4.5 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	Logisim 四条指令的扩展
第二天	3 条中断指令的实现，完成中断实现的设计方案
第三天	中断硬件的实现，中断程序编写
第四天	实现多级中断
第五天	实现了理想流水线
第六天	实现了重定向流水线
第七天	开始写 verilog 各个模块
第八天	整合模块
第九天	仿真排 bug
第十天	继续排 bug

5 设计总结与心得

5.1 课设总结

本次课程设计共完成了以下的任务：

- 1) 单周期 CPU4 条指令的扩展。
- 2) 单周期 CPU 多级嵌套中断的实现。
- 3) 理想流水线的实现。
- 4) 气泡流水线的实现。
- 5) 重定向流水线的实现。
- 6) FPGA 板上实现重定向流水线。

5.2 课设心得

这次的课程设计过程充满了艰辛与曲折，有一些任务是很难，但大多数的任务需要的是细心的连线和清晰的思路，事实上本次实验的大多数时间是被用去排除 bug 了，光上 FPGA 板之前的仿真排 bug 就整整花了我 3 天的时间，如果一开始就设计并仔细写好，我想就不会出这么多的 bug 了吧。

首先是中断的实现，刚开始跟据任务书上的提示我做了 CP0 以及相关操作，但后来发现事实上若只做中断的话有更好的解决方案，就是用硬件去存当前 PC 值，有多少中断存多少个，这样一一对应的关系使中断过程中复杂的 PC 读写关系变得简单，但由于知道这个方法的时候 CP0 已经做完了，所以只能硬着头皮写下去，再来是屏蔽字，当初觉得屏蔽字是可以用逻辑门生成的，但这样很容易导致逻辑错误以至于中断优先级不对，最稳妥的办法是用程序实现，mtc0 修改 CP0 中屏蔽字对应的寄存器。

再来是流水线的实现，首先是理想流水线，关键点是要把五个阶段分出来并将信号传递过去，然后再需要此信号的时候将其引出来即可。然后是气泡流水线，这个就是做一个逻辑判断电路判断两条指令是否有数据冲突，如果有就插入气泡，与其说是难点，不如说是麻烦点，因为逻辑判断电路要把每一种指令都判断一遍。最后是重定向流水线，这个其实跟气泡比没有太大的区别，还是要比较数据相关性，只是比较的成员多了，然后根据对应相关性将数据重定向到对应的位置即可，load-use 指令还是

华中科技大学课程设计报告

插入气泡不变，所以其实完成了气泡，重定向也不难改。

本次的实验我应该算是运气比较好，班上有一个非常热心的同学在我们组里，所以有什么不会的也可以问他，他都会非常热心地解答。之前的子模块共享也是由他写的，但由于子器件我根据自己的需求有过修改(logisim 上)，直接用他的出了不少 bug，这里还是长个记性，以后还是用自己已经都透彻的模块会比较稳。

最后要感谢老师们对我们的进度的关心，可以看出这次的课设难度和其他的课程设计难度不同，老师们也怕我们写不完就一直在给助攻，经常一语点醒梦中人，让我能有新的进展。

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.

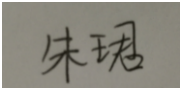
• 指导教师评定意见 •

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, which appears to read '朱瑞' (Zhu Rui).