

华中科技大学

2024

计算机组成原理

· 实验报告 ·

专 业： 计算机科学与技术

班 级： CS2208

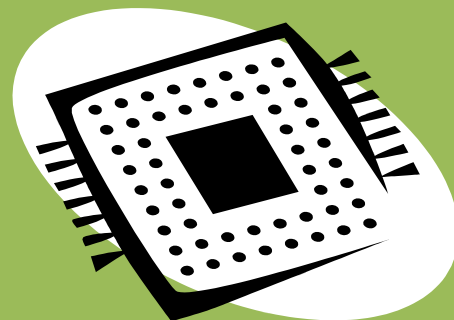
学 号： U202215628

姓 名： 方子豪

电 话： 19546890096

邮 件： 3299488768@qq.com

完成日期： 2024-06-14



计算机科学与技术学院

华中科技大学课程实验报告

目录

1 CPU 设计实验	2
1.1 设计要求	2
1.2 方案设计	3
1.2.1 MIPS 指令译码器设计	3
1.2.2 支持中断的微程序入口查找逻辑	4
1.2.3 支持中断的微程序条件判别测试逻辑	4
1.2.4 支持中断的微程序控制器设计	5
1.2.5 支持中断的微程序单总线 CPU 设计	6
1.2.6 支持中断的现代时序硬布线控制器状态机设计	8
1.2.7 支持中断的现代时序硬布线控制器设计	9
1.2.8 支持中断的硬布线单总线 CPU 设计	9
1.3 实验步骤	10
1.4 故障与调试	10
1.4.1 判别测试逻辑出现问题	10
1.4.2 寄存器翻转设置错误	10
1.4.3 中断请求设置出错	11
1.5 测试与分析	12
1.5.1 简单运行 sort-5-int.hex 测试	12
1.5.2 中断测试	13
1.6 实验总结	14
1.7 实验心得	14

1 CPU 设计实验

1.1 设计要求

要求设计支持中断机制的现代时序 RISC-V 单总线 CPU。CPU 的总体架构已经给出，需要分别完成现代时序 RISC-V 单总线 CPU 的微程序控制器和硬布线控制器。

对微程序控制器，具体任务包括设计指令译码器，设计支持中断的微程序入口查找逻辑，设计支持中断的微程序条件判别测试逻辑，支持中断的微程序控制器，最后连接各单元设计，然后运行带中断排序程序检验 CPU 能否正确运行。

对于硬布线控制器，具体任务则是设计支持中断的硬布线状态机，从而设计支持中断的硬布线控制器，完成测试后用硬布线控制器替换 cpu 中的微程序控制器进行程序测试。

要求支持的指令如下表 1-1:

表 1-1 要求 CPU 支持的指令及描述

指令	汇编代码	指令类型	功能描述
lw	lw,rt,imm(rs)	I 型	$R[rd] \leftarrow M[R[rs1] + \text{SignExt}(imm)]$
sw	sw,rt,imm(rs)	I 型	$M[R[rs1] + \text{SignExt}(imm)] \leftarrow R[rs2]$
beq	beq rs,rt,imm	I 型	if ($R[rs1] == R[rs2]$) $PC \leftarrow PC + \text{SignExt}(imm) \ll 1$
addi	addi rt,rs,imm	I 型	$R[rd] \leftarrow R[rs1] + \text{SignExt}(imm)$
slt	slt rd,rs,rt	R 型	If ($rs1 < rs2$) $R[rd] \leftarrow 1$ else $R[rd] \leftarrow 0$
eret	eret		$PC \leftarrow EPC, IE \leftarrow 1$

1.2 方案设计

1.2.1 MIPS 指令译码器设计

MIPS 指令集中共有三种类型的指令，对应的指令格式如图 1-1,1-2,1-3 所示：



图 1 R 型指令格式



图 2 I 型指令格式



图 3 J 型指令格式

其中 R 型指令的 OP 为 0，通过区分 func 来判断 R 型指令的类型，而 I 和 J 型指令则可以通过 OP 字段进行区分。

在我们设计的 CPU 中共能够支持 lw, sw, add, addi, slt, eret 共六条指令，其中前四条为 I 型指令，slt 为 R 型指令。故我们查询信号的 26~31 位即可获得 I 型和 J 型指令，对于 R 型指令 slt，我们不光要判断 OP=0，同时还要检测 func 字段是否满足 slt 要求。通过查询 MIPS 指令手册，我们获得了对应指令的指令字，因此通过 logisim 自带的比较器即可完成译码器设计：

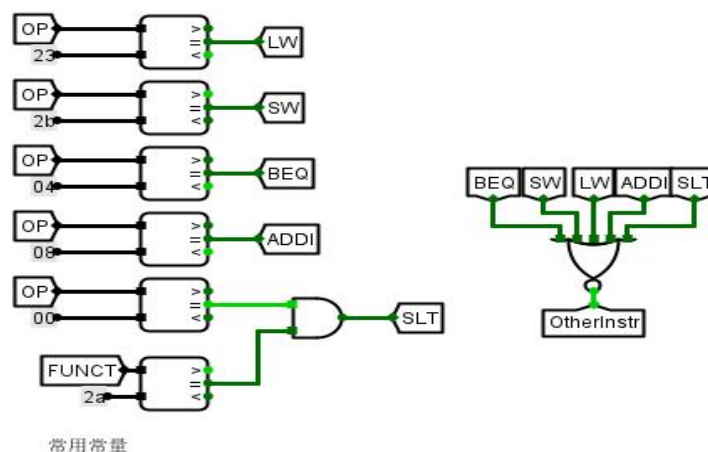


图 4 指令译码器设计图

华中科技大学课程实验报告

将对应的情况填入表格即可完成我们的条件判断逻辑设计：

输入（填1或0，不填为无关项x）							
P0	P1	P2	equal	IntR	S2	S1	S0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	1
1	0	0	1	1	0	0	1
0	1	0	1	0	0	1	0
0	1	1	1	0	0	1	0
0	1	1	1	1	0	1	0
0	1	1	0	1	0	1	1
0	1	1	0	0	1	0	0
0	0	1	0	1	0	1	1
0	0	1	1	1	0	1	1
0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	0
0	1	0	0	1	1	0	0

图 6 判别测试逻辑设计

1.2.4 支持中断的微程序控制器设计

对于每一条指令，我们都将其分解为了多条微指令，而最终的目的则是在 CPU 的同一时钟控制下，在每一个时钟周期能够输出正确的微命令并实现对应的微操作，我们需要分析表 1 中给出的六个指令对应的机器周期中会执行什么操作，并将这些操作分配到一定的时钟周期中来完成。最终实现将一个指令拆分成多个微指令，每个微指令调用一些相容性的微命令在一个时钟周期内完成一些微操作。设计如图 7：

微指令序号	PCin	Shlout	Zout	Rout	Wout	Shlout	PCin	ARin	DRin	Xin	Rin	IRin	PCin	Wout	Add	Sub	Slt	READ	Wout	PCin	Wout	STI	CLI	P1	P2	P3	微指令十六进制	微指令十六进制
取指令	0	1					1			1																10000000100100000000000000000000	20240000	
取指令	1																	1									00000000000000000000000000000000	800
取指令	2		1				1	1											1								00100000101000000000000000000000	8500200
取指令	3		1									1												1			01000000000000000000000000000000	10010004
lw	4			1							1																00010000000000000000000000000000	4040000
lw	5				1											1											00001000000000000000000000000000	2001000
lw	6			1																							00100000100000000000000000000000	8200000
lw	7											1								1							00000000010000000000000000000000	100200
lw	8		1										1														10100000000000000000000000000000	10020001
sw	9			1								1															00010000000000000000000000000000	4040000
sw	10				1																						00001000000000000000000000000000	2001000
sw	11			1																							00100000100000000000000000000000	8200000
sw	12			1																							00010000001000000000000000000000	4084000
sw	13					1																					10000001000000000000000000000000	800101
beq	14			1								1															00010000000000000000000000000000	4040000
beq	15			1																							10010000000000000000000000000000	400C003
beq	16	1										1															10000000000000000000000000000000	20040000
beq	17					1																					00000100000000000000000000000000	1001000
beq	18			1																							10010000100000000000000000000000	8400001
slt	19				1							1															00010000000000000000000000000000	4040000
slt	20				1																						00010000000000000000000000000000	4004400
slt	21			1								1															10010000000000000000000000000000	8022001
addi	22			1								1															00010000000000000000000000000000	4040000
addi	23				1																						00001000000000000000000000000000	2001000
addi	24			1								1															10010000000000000000000000000000	8020001
eret	25						1																				10000000100000000000000000000000	400091
中断响应	26	1																									10000000000000000000000000000000	20000048
中断响应	27						1																				10000000100000000000000000000000	400021

图 7 微指令设计图

华中科技大学课程实验报告

最终将对应的微指令保存至控制存储器中并连线实现微程序控制器设计如图 8:

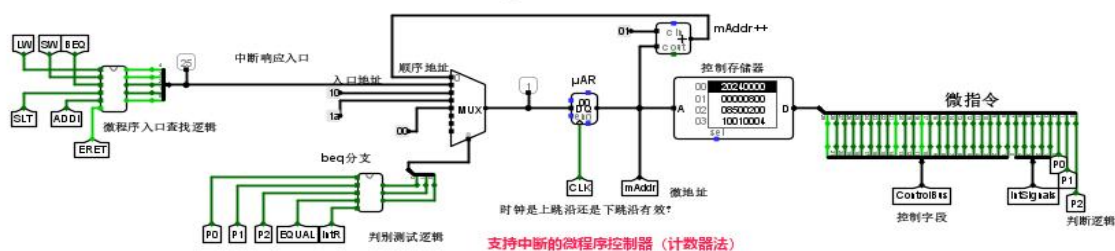


图 8 现代时序带中断微程序控制器电路图

由于实现微程序控制器使用的是计数法，所以我们的微指令设计中没有包含下址字段，同样我们在连接控制器的时候也不需要利用下址字段来进行跳转，而是通过添加一位判别位来利用判别字段和状态反馈信息来进行跳转。

对应 1.2.3 小节中的状态来反映跳转对应情况,我们这里利用多路选择器来分别进行跳转的区分。为了方便观察跳转目标可以参照下图 12,跳转的状态用 S2S1S0 表示,当 S2S1S0 等于 010 时,说明进入 beq 分支,此时对于,即为 15 状态跳入 16,因此地址为 0X10。同理当 S2S1S0 等于 011 时说明进入中断,则对应状态 26 即为 0X1a。S2S1S0 为 100 时则说明选择进入取值微程序,则地址为 0X0。

1.2.5 支持中断的微程序单总线 CPU 设计

对于 CPU 的连线大部分已经完成，需要我们自己完成的总共三步：

- (1) 将微程序控制器连入总线当中实现控制功能
- (2) 在 RAM 中加载 sort-5-int.hex 程序并最后成功运行
- (3) 在单总线数据通路中增加与中断相关的硬件模块

对于步骤一和二非常简单，不再赘述。对于步骤三，主要包括异常程序地址计数器 EPC，中断使能寄存器 IE，中断控制器等模块，需要在主电路中将**这些模块**进行有效连接，并在本关进行最终的联调，测试 CPU 是否能正常响应 2 个按键对应的中断服务程序。对于 EPC 保存现场使用一个寄存器 EPC 即可，难点在于找到对应中断处理程序的入口地址，这里我们选择用实验包里给的预装软件中的 **Mars4_5** 来查看对应入口地址：

华中科技大学课程实验报告

```
C:\Users\32994\Desktop\hustzc-master\预装软件>java -jar Mars4_5.jar
```

图 9 在当前文件夹打开终端运行 Mars4_5

再依照文件开头要求对环境进行设置，接着在 settings 选项打开 show labels window 选项，就可以找到对应的中断程序入口地址：

IntProgram1	0x000030a4
IntProgram2	0x000030ec

图 10 中断程序入口地址

最终就能够完成与中断相关的硬件模块的设计：

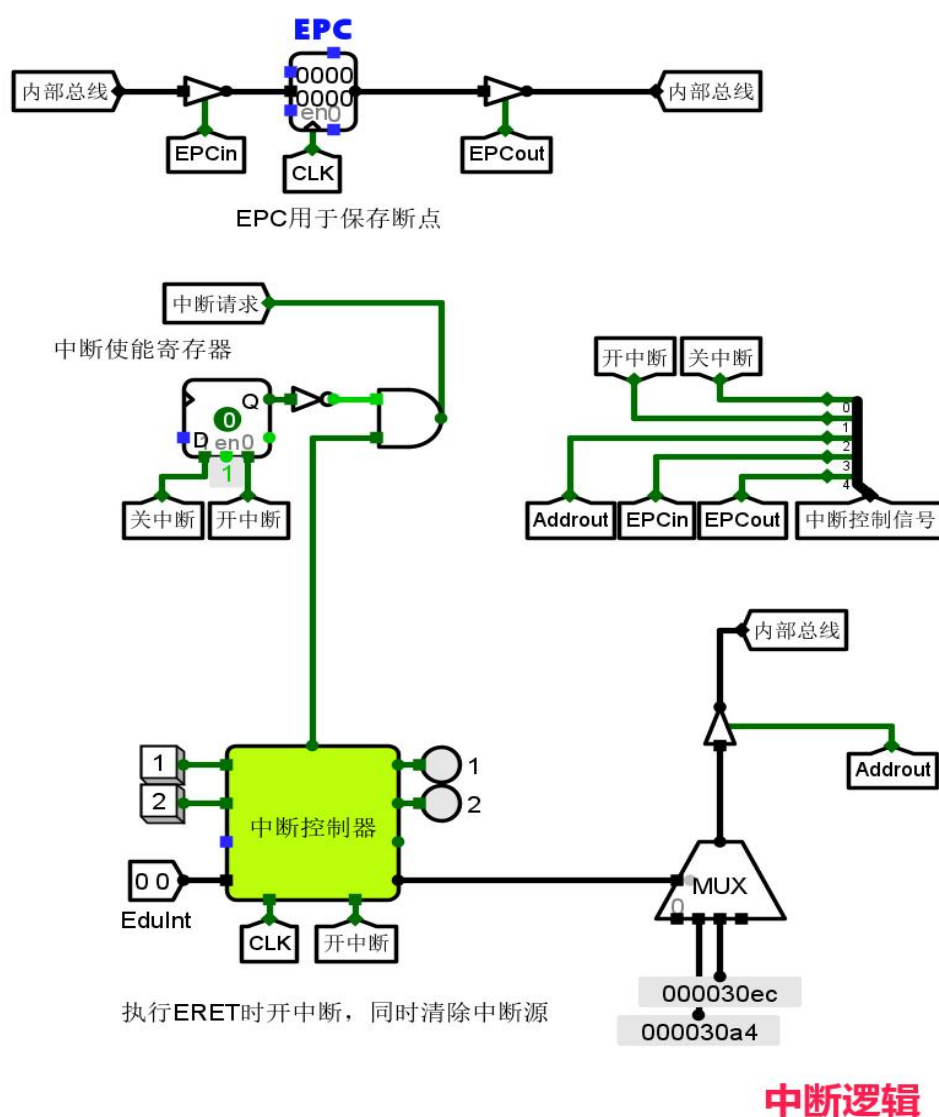


图 11 中断硬件逻辑设计

1.2.6 支持中断的现代时序硬布线控制器状态机设计

设计该支持中断的硬布线控制器状态机，我们需要首先明确状态总数和状态间的转移关系，因此我们直接通过状态图来获取信息：

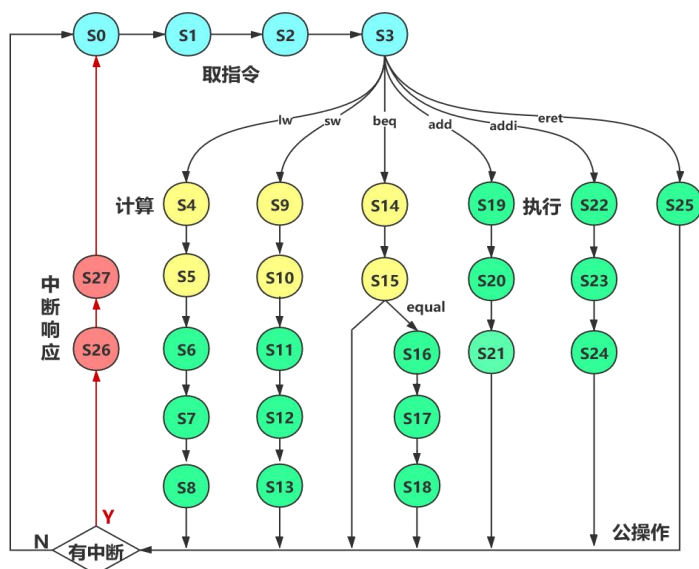


图 12 支持中断的硬布线控制器状态图

从该图中我们可以很方便知道总共有 28 个状态以及这 28 个状态之间的转移图，所以我们利用将对应信息输入至 Excel 表格中并利用其自动生成次态逻辑表达式：

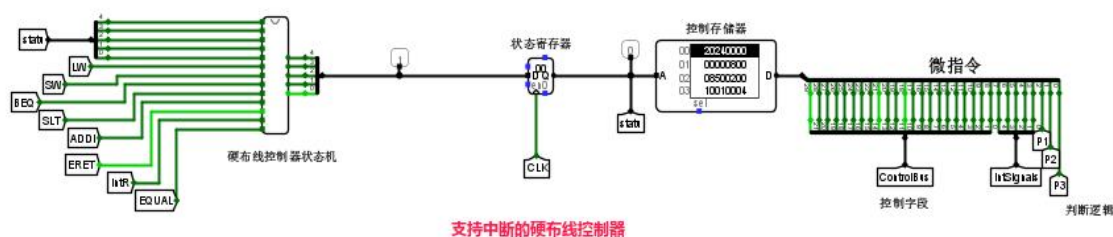
当前状态(现态)						输入信号								下一状态 (次态)					
S4	S3	S2	S1	S0	现态 10 进制	IW	SW	BEQ	SLT	ADDI	ERET	IR	EQUAL	次态 10 进制	N4	N3	N2	N1	N0
0	0	0	0	0	0									1	0	0	0	0	1
0	0	0	0	1	1									2	0	0	0	1	0
0	0	0	1	0	2									3	0	0	0	1	1
0	0	0	1	1	3	1								4	0	0	1	0	0
0	0	0	1	1	3		1							9	0	1	0	0	1
0	0	0	1	1	3			1						14	0	1	1	1	0
0	0	0	1	1	3				1					19	1	0	0	1	1
0	0	0	1	1	3					1				22	1	0	1	1	0
0	0	0	1	1	3						1			25	1	1	0	0	1
0	0	1	0	0	4							1		5	0	0	1	0	1
0	0	1	0	1	5									6	0	0	1	1	0
0	0	1	1	0	6									7	0	0	1	1	1
0	0	1	1	1	7									8	0	1	0	0	0
0	1	0	0	1	9									10	0	1	0	1	0
0	1	0	1	0	10									11	0	1	0	1	1
0	1	0	1	1	11									12	0	1	1	0	0
0	1	1	0	0	12									13	0	1	1	0	1
0	1	1	1	0	14									15	0	1	1	1	1
1	0	0	1	1	19									20	1	0	1	0	0
1	0	1	0	0	20									21	1	0	1	0	1
1	0	1	1	0	22									23	1	0	1	1	1
1	0	1	1	1	23									24	1	1	0	0	0
0	1	0	0	0	8							1		26	1	1	0	1	0
0	1	0	0	0	8							0		0	0	0	0	0	0
0	1	1	0	1	13							1		26	1	1	0	1	0
0	1	1	1	1	15							0	0	0	0	0	0	0	0
0	1	1	1	1	15							1	0	26	1	1	0	1	0
0	1	1	1	1	15								1	16	1	0	0	0	0
1	0	0	1	0	18							0		0	0	0	0	0	0
1	0	0	1	0	18							1		26	1	1	0	1	0
1	0	1	0	1	21							0		0	0	0	0	0	0
1	0	1	0	1	21							1		26	1	1	0	1	0
1	1	0	0	0	24							0		0	0	0	0	0	0
1	1	0	0	0	24							1		26	1	1	0	1	0
1	1	0	0	1	25							0		0	0	0	0	0	0
1	1	0	0	1	25							1		26	1	1	0	1	0
1	1	0	1	0	26									27	1	1	0	1	1
1	1	0	1	1	27									0	0	0	0	0	0
1	0	0	0	0	16									17	1	0	0	0	1
1	0	0	0	1	17									18	1	0	0	1	0

图 13 硬布线状态机状态转移表

华中科技大学课程实验报告

1.2.7 支持中断的现代时序硬布线控制器设计

由于已经实现指令译码和现代时序状态机等模块，所以我们只需要将对应的指令译码信号以及状态信号准确的送入硬布线控制状态机中便可以得到下一阶段的状态，每个时钟周期进行状态的转换一次。最终连线效果如图：



1.2.8 支持中断的硬布线单总线 CPU 设计

将 1.2.5 中的微程序控制器换成 1.2.7 中设计的硬布线控制器，其余部分依照原本的连线逻辑和查找逻辑进行连接即可。

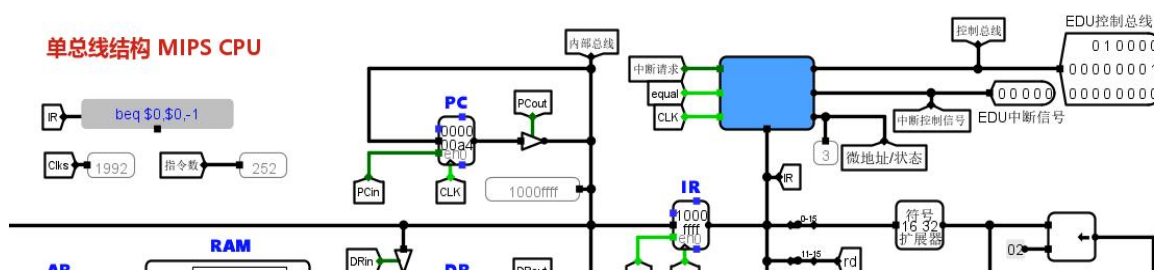


图 14 微程序控制器单总线 CPU

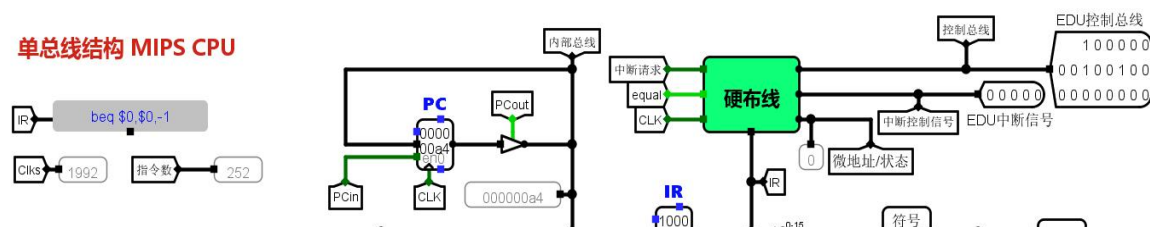


图 15 硬布线控制器单总线 CPU

1.3 实验步骤

- (1) 分析总体任务要求，将总体任务拆分成多个微小模块。
- (2) 针对每个任务方面进行针对性设计，对于复杂的状态设计借助 Excel 表格的功能自动生成表达式辅助完成设计，对并进行子电路封装使其能被总电路引用。
- (3) 将多个子电路进行组装联调形成最终电路并运行简单的排序程序 `sort-5-int.hex` 检测 CPU 能否正常工作。
- (4) 将 CPU 文件提交到头歌平台上进行测试。

1.4 故障与调试

1.4.1 判别测试逻辑出现问题

故障现象：在设计条件判别测试的时候，仅仅只考虑了部分情况，没有估计全体，比如在设计之初没有考虑到 P2, P3, equal, Inter 同时出现的问题，导致即使已经确定的情况的优先级，在这种情况下应当执行 beq 分支指令跳转到 16，但是实际运行情况会显示跳转错误，进入中断周期分支。

原因分析：在设计判别测试逻辑时，为了优化部分行，选择将类似的判别结果综合起来放在一起，但是由于没有经过严格的计算和分析，最终导致出错。

解决方案：检查所有对应状态的转移是否均出现在我们的判别测试逻辑设计表中，增添对应缺少的项即可，最终修改完毕之后即为图 6 所示。

1.4.2 寄存器翻转设置错误

故障现象：在本地 logisim 完成微程序控制器设置之后，将文件提交到头歌进行测评，反馈发生错误，并且在每一个时钟周期输出两次，且第二次输出均为错误的值。

具体现象如图 16：

Cnt	Instr	equal	IntR	mAddr	cBus	ErrBit	Isigs	
00	2010ffff	0	0	00	202400	xx	00	
00	2010ffff	0	0	01	000008	15	00	Error!
01	2010ffff	0	0	01	000008	xx	00	
01	2010ffff	0	0	02	085002	13	00	Error!
02	2010ffff	0	0	02	085002	xx	00	
02	2010ffff	0	0	03	100100	14	00	Error!
03	2010ffff	0	0	03	100100	xx	00	
03	2010ffff	0	0	16	040400	14	00	Error!

图 16 控制器出错

原因分析：经过检查 CPU 总线的总体部分发现，在除了控制器之外的 CPU 的部分的所有寄存器，全部都是上升沿改变并锁存。故当我们设置微指令寄存器 μAR 为上升沿改变并锁存时，当时钟上升沿来到后，首先我们的微指令寄存器 μAR 会改变，经过组合逻辑之后导致对应的微指令改变，而此时还在上升沿，微指令的改变会导致 CPU 中其他寄存器的一同改变锁存，这一改变又会导致微指令寄存器的输入可能改变，我们无法保证当改变之后的值传递到当前的微指令寄存器时我们的微指令寄存器 μAR 是否已经完成锁存，所以有可能发生一个周期翻转两次的错误。

解决方案：将微指令寄存器改为下降沿触发，避开数据锁存和状态切换的过程，从而使得状态切换时其他寄存器的值暂时不变，而当其他寄存器的值在上升沿锁存时也不会影响微指令寄存器的状态切换。

1.4.3 中断请求设置出错

故障现象：中断的关闭和开启并不按照逻辑运行，无法实现中断控制，中断请求反馈出错。

原因分析：在单总线 CPU 中断实现逻辑中，寄存器 IE 值为 0 时代表关中断，值为 1 时代表开中断。而在我一开始的想法中是 IE 为 1 代表关中断，IE 为 0 代表开中断，这就导致中断请求反馈发生错误，如图 17，此时应当是关中断，但是中断请求却置 1 了。

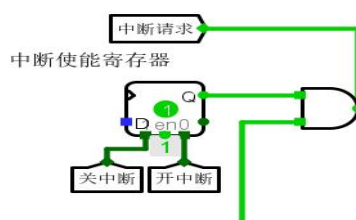


图 17 中断请求发生错误

华中科技大学课程实验报告

解决方案：纠正思维的逻辑错误和实际的单总线 CPU 实现之间的区别即可，即确保中断让寄存器置 0，开中断让寄存器置 1，且寄存器置 0 时能够反馈中断请求，所以在寄存器输出端 Q 加一个非门即可实现置 0 开中断的需求。

1.5 测试与分析

1.5.1 简单运行 sort-5-int.hex 测试

将两种控制器分别替换至已经连接好的单总线数据通路中，加载 sort-5-int.hex 程序镜像至 ROM 中并运行测试，结果如下：

```
000 23bd0400 2010fff 20110000ae300200 2210000122310004ae30020022100001 22310004ae3002002210000122310004 ae3002002210000122310004ae300200
010 2210000122310004ae30020022100001 22310004ae3002002210000122310004 ae300200201000002011001c8e130200 8e3402000274402a11000002ae330200
020 ae140200 2231fff 12110001 1000fff7 221000042011001c12110001 1000fff3 1000fff 23bd0008afb00000 afb10004 203102408e30000022100001ae300000
030 ae300004ae300008ae30000cae300010 ae300014ae300018ae30001c8fb10004 8fb00000 23bdfff8 4200001823bd0008 afb00000 afb10004 203102808e300000
040 2210fff ae300000ae300004ae300008 ae30000cae300010ae300014ae300018 ae30001c8fb10004 8fb00000 23bdfff8 42000018000000000000000000000000
050 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
060 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
070 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
080 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
090 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
```

图 18 原始数据内容

将微程序控制器连接至单总线数据通路中并运行程序，不人为添加任何中断，最终运行完毕共执行 252 条指令，再次查看 ROM 中数据结果如图 19：

```
000 23bd0400 2010fff 20110000ae300200 2210000122310004ae30020022100001 22310004ae3002002210000122310004 ae3002002210000122310004ae300200
010 2210000122310004ae30020022100001 22310004ae3002002210000122310004 ae300200201000002011001c8e130200 8e3402000274402a11000002ae330200
020 ae140200 2231fff 12110001 1000fff7 221000042011001c12110001 1000fff3 1000fff 23bd0008afb00000 afb10004 203102408e30000022100001ae300000
030 ae300004ae300008ae30000cae300010 ae300014ae300018ae30001c8fb10004 8fb00000 23bdfff8 4200001823bd0008 afb00000 afb10004 203102808e300000
040 2210fff ae300000ae300004ae300008 ae30000cae300010ae300014ae300018 ae30001c8fb10004 8fb00000 23bdfff8 42000018000000000000000000000000
050 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
060 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
070 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
080 00000006000000050000000400000003 000000020000001000000000 ffffffff 00000000000000000000000000000000 00000000000000000000000000000000
```

图 19 微程序控制器排序结果

将硬布线控制器替换微程序控制器至数据通路中并运行程序，不人为添加任何中断，最终运行完毕共执行 252 条指令，结果如图 20：

```
000 23bd0400 2010fff 20110000ae300200 2210000122310004ae30020022100001 22310004ae3002002210000122310004 ae3002002210000122310004ae300200
010 2210000122310004ae30020022100001 22310004ae3002002210000122310004 ae300200201000002011001c8e130200 8e3402000274402a11000002ae330200
020 ae140200 2231fff 12110001 1000fff7 221000042011001c12110001 1000fff3 1000fff 23bd0008afb00000 afb10004 203102408e30000022100001ae300000
030 ae300004ae300008ae30000cae300010 ae300014ae300018ae30001c8fb10004 8fb00000 23bdfff8 4200001823bd0008 afb00000 afb10004 203102808e300000
040 2210fff ae300000ae300004ae300008 ae30000cae300010ae300014ae300018 ae30001c8fb10004 8fb00000 23bdfff8 42000018000000000000000000000000
050 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
060 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
070 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
080 00000006000000050000000400000003 000000020000001000000000 ffffffff 00000000000000000000000000000000 00000000000000000000000000000000
```

图 20 硬布线控制器排序结果

在以上测试中，均同时测试过硬布线控制器和微程序控制器，二者执行结果完全一致，足以证明二者设计在一定程度上符合要求。

1.6 实验总结

- 1) 完成了 MIPS 指令译码器设计，使其能够支持识别六条所需要执行的指令并进行判断区分。
- 2) 完成了支持中断的微程序入口查找逻辑设计和微程序条件判别测试逻辑。
- 3) 完成了支持中断的微程序控制器设计。
- 4) 完成了利用支持中断的微程序控制器实现的单总线 CPU 设计。
- 5) 实现了支持中断的现代时序硬布线控制器状态机设计。
- 6) 实现了支持中断的现代时序硬布线控制器设计。
- 7) 完成了利用支持中断的现代时序硬布线控制器的单总线 CPU 设计。
- 8) 成功在 CPU 上运行一个冒泡排序程序。
- 9) 实现了 CPU 对中断的处理。

1.7 实验心得

(1) 实验的核心任务是设计一个支持中断机制的现代时序 RISC-V 单总线 CPU，这对我来说既是挑战也是机遇。通过设计微程序控制器和硬布线控制器，并将它们分别应用于 CPU，我学会了如何将抽象的理论知识转化为实际的电路设计。这种理论与实践的结合使我对计算机硬件有了更直观的认识，深刻理解了指令译码、状态转移和中断处理等核心概念。

(2) 对 logisim 的使用更加熟悉，能够利用 logisim 自带的各个组件进行组装联调从而完成一个制定的功能部件。同时对 logisim 中自带的各种逻辑门和各种运算器如选择器译码器等有了更加深刻的理解。

(3) 在这次计算机组成原理实验中，我收获颇丰，不仅提高了理论知识的掌握程度，更增强了动手实践能力。通过对现代时序 RISC-V 单总线 CPU 的设计和实现，使我对计算机组成原理有了更加深入的理解。

华中科技大学课程实验报告

(4) 实验过程中,难免会遇到各种问题。例如,在设计条件判别测试逻辑时,由于初期设计不够完善,导致了优先级判断错误,从而引发了跳转错误。通过仔细检查和分析,我逐一找出了问题的根源并加以修正。这一过程不仅锻炼了我的逻辑思维能力,也培养了我解决问题的耐心和细致程度。

(5) 通过这次实验,我不仅巩固了计算机组成原理的相关知识,还提升了动手实践能力和解决问题的能力。更重要的是,我学会了如何在面对复杂问题时,通过细致的分析和查找来找到解决方案。这些收获将对我今后的学习和职业发展产生深远的影响。

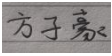
(6) 这个实验让我亲手搭建了简单的 CPU 模型,通过手动操作模拟了指令的取指、译码等阶段。我深刻理解了指令在 CPU 中是如何一步步被处理的,也体会到了 CPU 设计的复杂性和精妙之处。

• 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: 

二、对课程实验的学术评语（教师填写）

三、对课程实验的评分（教师填写）

评分项目 (分值)	课程目标 1 工具应用 (10 分)	课程目标 2 设计实现 (70 分)	课程目标 3 验收与报告 (20 分)	最终评定 (100 分)
得分				

指导教师签字: _____