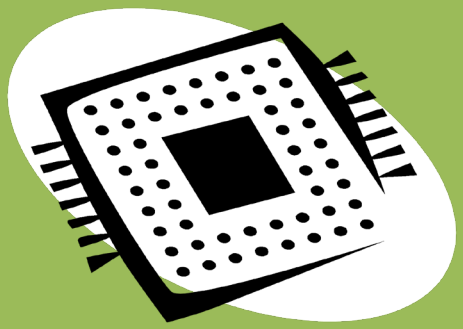
华中科技大学

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：	U202215642
姓　　名：	田清林
电　　话：	15603480922
邮　　件：	1240067633@qq.com
完成日期：	2024-06-07



计算机科学与技术学院

目录

1 CPU 设计实验

1.1 设计要求

利用 logisim 平台中现有运算部件以及框架设计一个支持中断的现代时序 MIPS 单总线 CPU，并执行简单的 sort_int.asm 程序。CPU 支持的指令及其功能如表 1-1 所示。

表 1-1 CPU 支持的指令及其功能

指令名	类型	汇编代码表示	指令功能
lw	I 型	lw rd,imm(rs1)	R[rd] ← M[R[rs1] + SignExt(imm)]
sw	I 型	sw rs2,imm(rs1)	M[R[rs1] + SignExt(imm)] ← R[rs2]
beq	I 型	beq rs1,rs2,imm	if(R[rs1] == R[rs2]) PC ← PC+4 + SignExt(imm) << 1
slt	R 型	slt rd,rs1,rs2	If (rs1 < rs2) R[rd] ← 1 else R[rd] ← 0
addi	I 型	addi rd,rs1,imm	R[rd] ← R[rs1] + SignExt(imm)
eret	--	eret	PC←EPC，IE←1

1.2 方案设计

1.2.1 MIPS 指令译码器设计

指令译码器是控制器核心功能部件，负责将指令字翻译成一根根的指令译码信号，每一根指令译码信号代表一条具体的指令。MIPS 常见指令格式如图 1.1 所示。

图 1.1 MIPS 常见指令格式

根据表 1-1 可知，LW、SW、BEQ、ADDI 是 I 型指令，需要按照高六位的 OP 字段值进行判断，SLT 为 R 型指令，需要 OP＝0 且 FUNCT 字段等于 0x2a 才可判断。各指令 OP、FUNCT 字段值如表 1-2 所示。

表 1-2 各指令 OP、FUNCT 字段值(无 eret)

指令名	OP	FUNCT
lw	100011 (0x23)	xxxxxx
sw	101011(0x2b)	xxxxxx
beq	000100(0x04)	xxxxxx
slt	000000(0x00)	101010(0x2a)
addi	001000(0x08)	xxxxxx

由于本实验中指令译码器只会接收到六个指令中的任何一条，因此指令不属于上述五条的指示引脚 OtherInstr 就可作为 eret 的指令引脚。

电路图如图 1.2 所示。

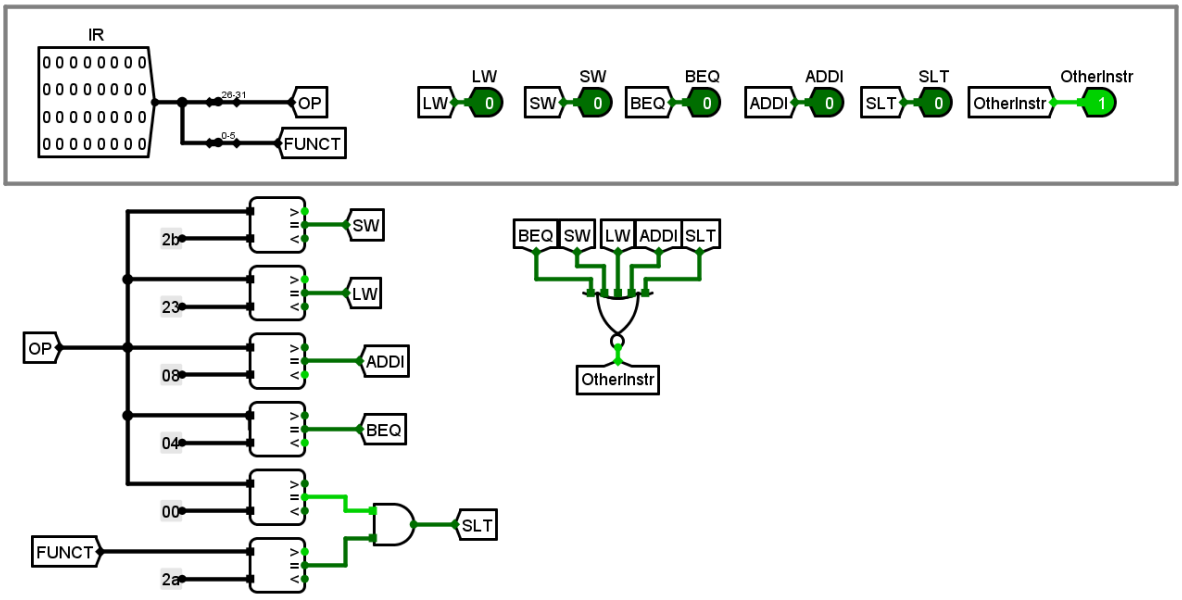


图 1.2MIPS 指令译码器电路图

1.2.2 支持中断的微程序入口查找逻辑

根据地址转移逻辑（如图 1.3 所示），在取值周期结束后需要根据译码信号进入合适的微程序继续执行，其微程序入口地址值与状态图中状态编号一致，填写相应 excel 表，如图 1.4 所示。填表后自动生成逻辑表达式，将得到的逻辑表达式输入 logsim 的

分析组合逻辑电路功能中即可自动生成逻辑电路。

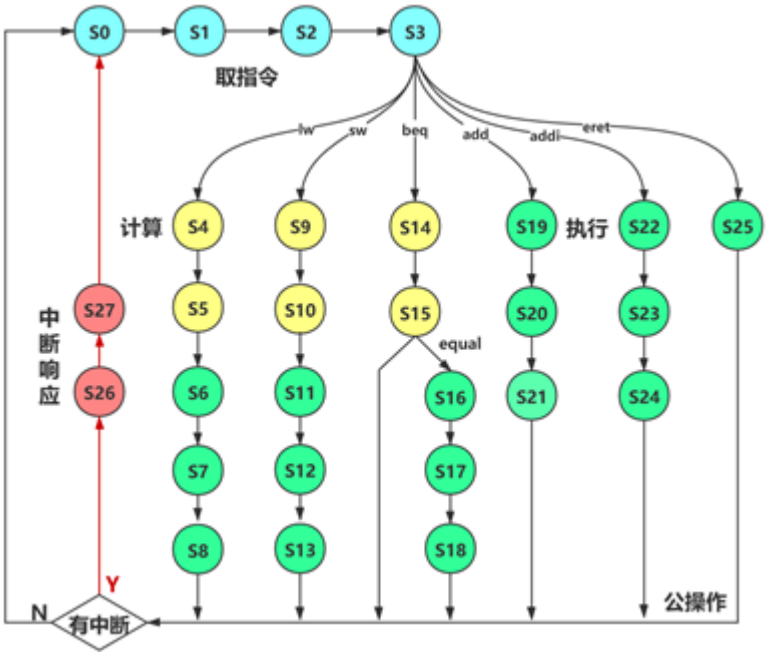


图 1.3 MIPS 指令执行状态转换图

机器指令译码信号						微程序入口地址					
LW	SW	BEQ	SLT	ADDI	ERET	入口地址10进制	S4	S3	S2	S1	S0
1						4	0	0	1	0	0
	1					9	0	1	0	0	1
		1				14	0	1	1	1	0
			1			19	1	0	0	1	1
				1		22	1	0	1	1	0
					1	25	1	1	0	0	1

图 1.4 微程序入口查找逻辑设计表

1.2.3 支持中断的微程序条件判别测试逻辑

条件判别测试主要是用来控制指令执行完毕后续地址的选择，分为计数器计算出的顺序地址、微程序入口地址、BEQ 分支、中断响应程序地址、取指微程序地址，各后续地址对应状态编码如表 1-3 所示。

表 1-3 后续地址与状态编码对应表

后续地址	状态编码 S
顺序地址	000

微程序入口地址	001
BEQ 分支	010
中断响应程序地址	011
取指微程序地址	100

当判别测试字段信号全部为 0 时后续地址为计数器计算出的顺序地址，当 P0 为 1 时跳转到微程序入口地址；在 P1 测试信号为 1 时，若 equal 为 1 则跳转到 beq 分支，否则跳转到取址微程序地址。

将对应值填入 excel 表格，如图 1.5 所示。之后按照逻辑表达式生成电路。

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	1
0	1		1		0	1	0
0	1	0	0		1	0	0
0	1	1	0	1	0	1	1
0	1	1	0	0	1	0	0
0	0	1		0	1	0	0
0	0	1		1	0	1	1

图 1.5 条件判别逻辑组合逻辑设计表

1.2.4 支持中断的微程序控制器设计

微指令功能		PCout	DRout	Zout	Rout	IR[Jdest]	IR[Adest]	DRdest	PCin	ARin	DREin	DRin	Xin	Rin	IRin	PSWin	Rs/Rt	RegDst	Add	Add4	Slt	READ	WRITE	EPCout	EPCin	Addrsrc	STI	CLI	P1	P2	P3	微指令	微指令十六进制
取指令	0	1								1			1																		10000000100100000000000000000000	20240000
取指令	1																			1											00000000000000000100000000000	800
取指令	2			1					1		1											1									00100001010000000000010000000000	8500200
取指令	3		1												1															1		01000000000001000000000000000100	10010004
lw	4				1								1																			00010000001000000000000000000000	4040000
lw	5					1													1													00001000000000000100000000000000	2001000
lw	6			1						1																						00100000100000000000000000000000	8200000
lw	7										1											1										00000000001000000000010000000000	100200
lw	8		1											1																	1	01000000000010000000000000000001	10020001
sw	9				1								1																			00010000001000000000000000000000	4040000
sw	10					1													1													00001000000000000100000000000000	2001000
sw	11			1						1																						00100000100000000000000000000000	8200000
sw	12				1							1					1															00010000001000010000000000000000	4084000
sw	13							1															1								1	00000010000000000000010000000001	800101
beq	14				1								1																			00010000001000000000000000000000	4040000
beq	15				1											1	1														1	1	00010000000000110000000000000011	400C003
beq	16	1											1																			10000000001000000000000000000000	20040000
beq	17						1												1													00000100000000001000000000000000	1001000
beq	18			1					1																						1	00100001000000000000000000000001	8400001
slt	19				1								1																			00010000001000000000000000000000	4040000
slt	20				1												1				1											00010000000000010001000000000000	4004400
slt	21			1										1				1													1	00100000000010001000000000000001	8022001
addi	22				1								1																			00010000001000000000000000000000	4040000
addi	23					1													1													00001000000000000100000000000000	2001000
addi	24			1										1																	1	00100000000010000000000000000001	8020001
eret	25								1															1			1				1	00000001000000000000000010010001	400091
中断响应	26	1																							1			1				10000000000000000000000001001000	20000048
中断响应	27								1																	1					1	00000001000000000000000000100001	400021

图 1.6 微指令设计表

首先需要进行微程序的设计。本次实验使用直接表示法和计数器法对微指令进行编码。根据表 1-1 给出的 6 个指令功能，分析每个微程序在每个时钟周期中需给出的操作信号以及顺序控制字段的值，填写 excel 表如图 1.6 所示，生成微指令后，将其填入控制存储器。

之后需要正确连接控制器电路。判别测试逻辑的输出作为入口地址多路选择器的选择端。电路图如图 1.7 所示。

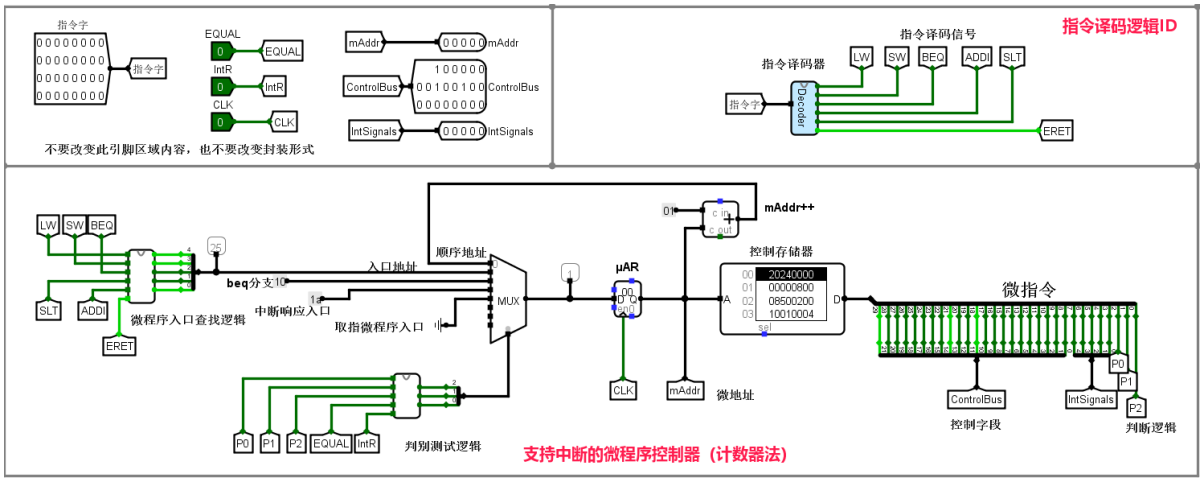


图 1.7 现代时序带中断微程序控制器电路图

多路选择器的各输入中，0 号代表顺序地址，是当前微地址 mAddr 加 1 后的值；1 号是微程序入口地址，由微程序入口查找逻辑给出各指令微程序入口地址；2 号是 beq 分支，是状态转换图（如图 1.3 所示）中 S16 对应微程序地址，当处于 S15 状态时且 equal 等于 1 时，会进入 S16 状态；3 号是中断响应入口地址，是状态转换图（如图 1.3 所示）中 S26 对应微程序地址，当指令执行完毕且有中断时会跳转到的位置；4 号是取址微程序入口地址，是状态 S0 的对应微程序地址。

1.2.5 支持中断的微程序单总线 CPU 设计

单总线 CPU 的其他部件已在框架中给出，这里只需完成中断逻辑相关电路（如图 1.8 所示）。在不支持中断的单总线 CPU 中加入中断，需要加入开/关中断、保存断点和中断识别功能。

（1） 开/关中断

开/关中断的操作需增加 IE 寄存器。图 1.8 中的 D 触发器相当于 IE 寄存器，当输出值为 1 时才可处理中断，开中断、关中断时输出分别为 1 和 0，接异步置 0 和置 1 端

（触发器输出是~Q）。将 IE 寄存器输出与中断控制器的中断信号接入与门的输入，输出即是中断请求信号。

（2） 保存断点

EPC 用来保存断点地址，是一个寄存器，输入与输出都接到内部总线，并接入三态门，由 EPCin 和 EPCout 信号分别控制写入和输出，如图 1.8 所示

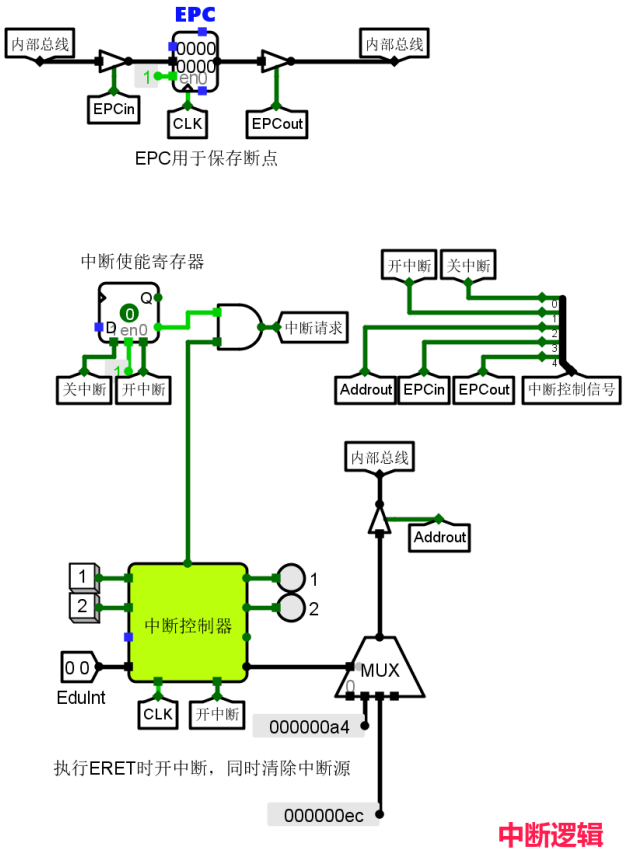


图 1.8 CPU 中断逻辑电路图

（3） 中断识别

中断控制器会检测中断并输出中断号，将中断号接入多路选择器的选择端，中断 1 和 2 地址分别接入多路选择器的 1、2 端口，中断入口地址需要用 Mars4_5.jar 程序查看。

使用 Mars4_5.jar 打开 Sort-5-int-riscv.asm，在 Settings 勾选对应选项,如图 1.9 所示。

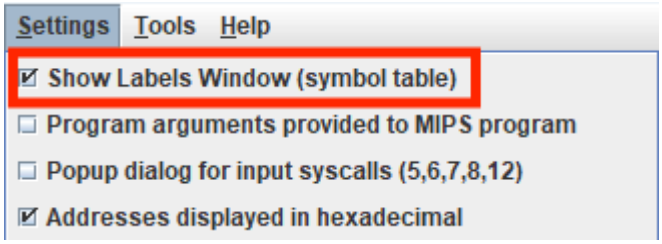


图 1.9 Settings 中勾选对应选项示意图

在 Run 菜单中选择 Assemble，即可查看对应中断程序的入口地址，如图 1.10 所示。注意这里给出的 IntProgram1 和 IntProgram2 的地址需要减去 sort_init 值，结果分别为 0xa4 和 0xec。将对应地址用常量输入多路选择器的端口 1 和 2。多路选择器的输出接入一个三态门连接到内部总线，由 Addout 信号控制数据的输出。

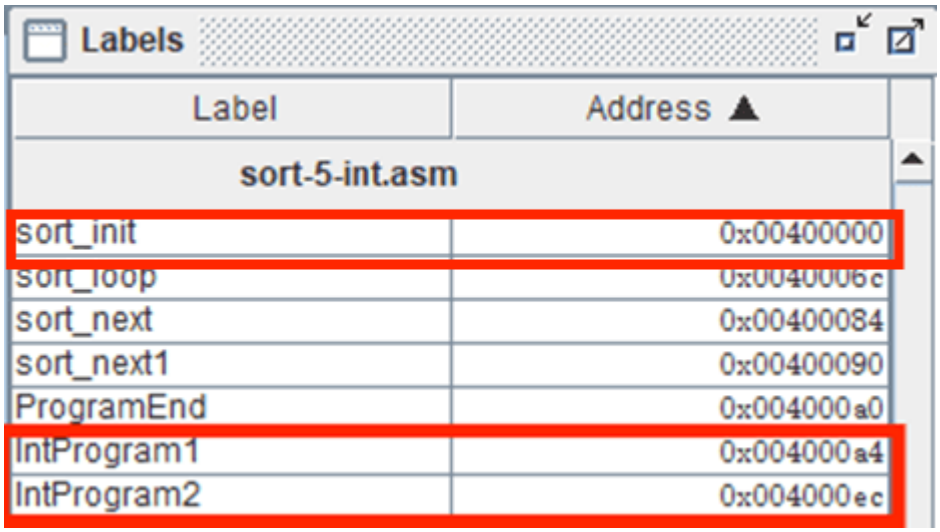


图 1.10 对应中断程序的入口地址

1.2.6 支持中断的现代时序硬布线控制器状态机设计

根据状态机的状态转换条件（如图 1.3 所示），填写 excel 表，如图 1.11 所示。

对于取址指令的 S3 状态，之后需要根据输入的 LW、SW、BEQ、SLT、ADDI、ERET 等译码信号进入对应的微程序入口。

对于指令的最后一个状态，根据 IR 值来判断是否处理中断，若 IR 为 1 则进入中断处理周期，IR 为 0 则回到取指程序。

对于 BEQ 指令的 S15 状态，若 EQUAL 为 1 则进入 S16 状态，若 EQUAL 为 0 该状态为指令的最后一个状态，需要根据 IR 值判断处理中断还是回到取指程序。

填写完后，将自动生成的逻辑表达式输入 logisim 自动生成电路即可。

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	LW	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	1	0	0	4									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	0	8							0		0	0	0	0	0	0
0	0	0	1	1	3		1							9	0	1	0	0	1
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	0	1	13							0		0	0	0	0	0	0
0	0	0	1	1	3			1						14	0	1	1	1	0
0	1	1	1	0	14									15	0	1	1	1	1
0	1	1	1	1	15							0	0	0	0	0	0	0	0
0	1	1	1	1	15								1	16	1	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
1	0	0	1	0	18							0		0	0	0	0	0	0
0	0	0	1	1	3				1					19	1	0	0	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	0	1	21							0		0	0	0	0	0	0
0	0	0	1	1	3					1				22	1	0	1	1	0
1	0	1	1	0	22									23	1	0	1	1	1
1	0	1	1	1	23									24	1	1	0	0	0
1	1	0	0	0	24							0		0	0	0	0	0	0
0	0	0	1	1	3						1			25	1	1	0	0	1
1	1	0	0	1	25							0		0	0	0	0	0	0
0	1	0	0	0	8							1		26	1	1	0	1	0
0	1	1	0	1	13							1		26	1	1	0	1	0
0	1	1	1	1	15							1	0	26	1	1	0	1	0
1	0	0	1	0	18							1		26	1	1	0	1	0
1	0	1	0	1	21							1		26	1	1	0	1	0
1	1	0	0	0	24							1		26	1	1	0	1	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

图 1.11 硬布线控制器状态机状态转换表

1.2.7 支持中断的现代时序硬布线控制器设计

将当前状态信息 statu 与各译码信号和控制信号作为硬布线控制器状态机的输入，将它的输出（次态）作为状态寄存器的输入，状态寄存器的输出为现态。由于没有编写各控制信号的组合逻辑表示，在控制存储器中放入先前的微程序，来控制各控制信号和顺序控制字段的值。电路图如图 1.12 所示。

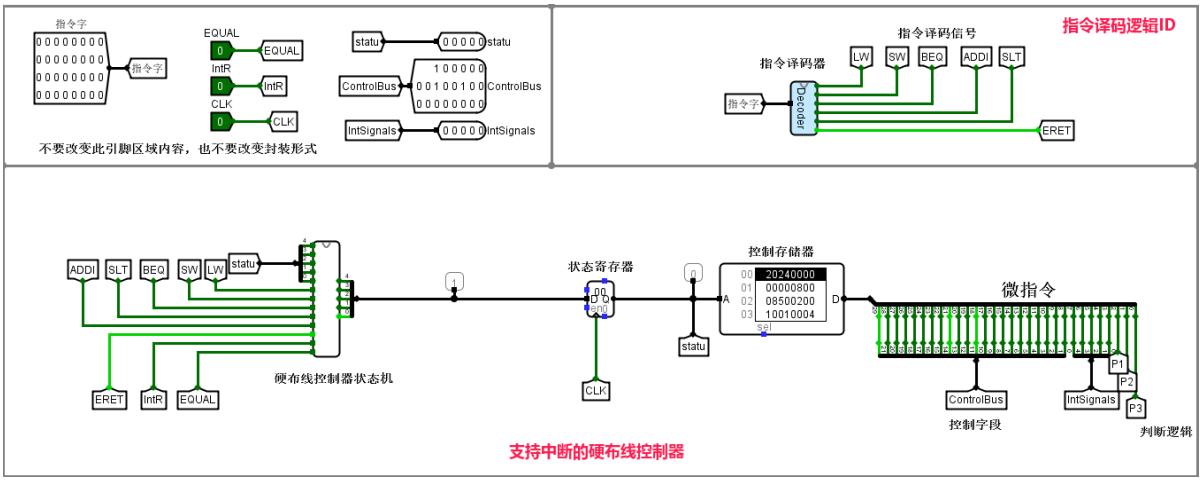


图 1.12 支持中断的现代时序硬布线控制器电路图

1.3 实验步骤

（1） 分析实验各子模块的作用及功能，理解支持中断的单总线 CPU 的结构和运行原理。

（2） 完成各个子模块的实现，可以借助实验附带的表格生成逻辑表达式，根据自动生成的表达式自动生成电路。

（3） 将各个子电路进行组装联调，构建最终的 CPU 电路。

（4） 将 logisim 代码复制到 Educoder 进行测试，若未通过则寻找故障并调试。

1.4 故障与调试

1.4.1 微程序条件判别测试无法通过

故障现象： 在 Educoder 测试后，当输入依次为 01101 时，输出状态为 4，错误。

P0	P1	P2	equal	IntR	S2	S1	S0
0	0	0			0	0	0
1					0	0	1
	1		1		0	1	0
	1		0		1	0	0
		1		0	1	0	0
		1		1	0	1	1

图 1.13 条件判别逻辑组合逻辑设计表（错误）

原因分析： 首先，该 excel 表格空格处为无关项 x，当编码出现二义性时，以靠前的行结果为准。判别逻辑组合逻辑设计表填写错误，没有考虑到 P1 和 P2 都为 1 时的优先级问题。以输入依次为 01101 为例，此时 P1 和 P2 都为 1，说明既要参考 equal 也要参考 IntR， equal 为 0 而 IntR 为 1，说明不会进入 beq 分支而是进入中断响应周期（状态编码为 011），但按照填写的设计表，会进入取指微程序（状态编码为 100）。

解决方案： 改正条件判别逻辑组合逻辑设计表。如图 1.5 所示。

1.4.2 寄存器时钟上升沿/下降沿的选择

故障现象：微程序控制器在 Educoder 测评时输出结果大面积混乱，cnt 值每个值都会出现两次。电路图如图 1.14 所示，Educoder 测试故障如图 1.15 所示。

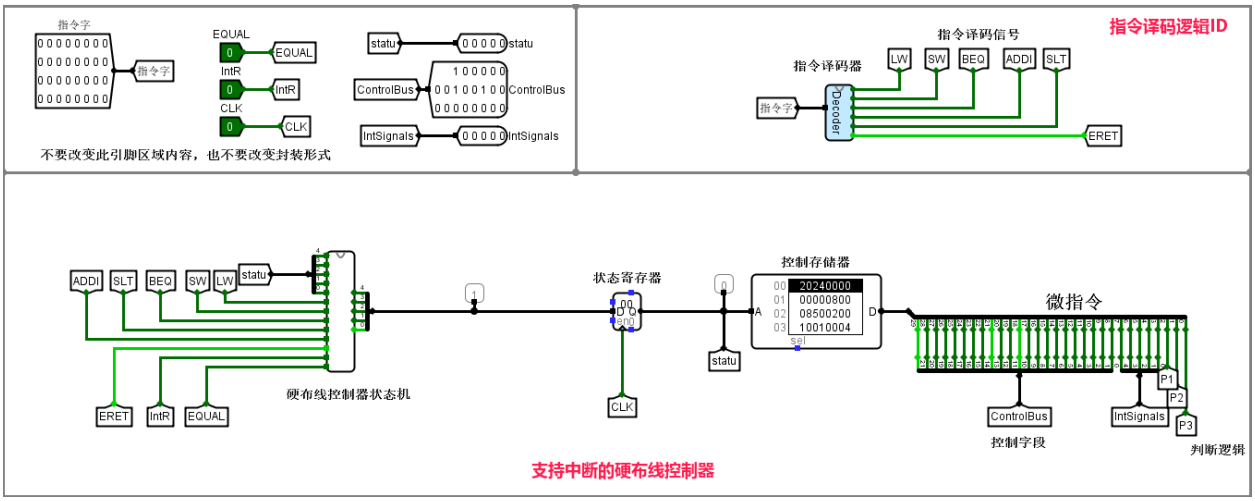


图 1.14 微程序控制器电路图

—— 预期输出 ——

Cnt	Instr	equal	IntR	mAddr	cBus
00	2010ffff	0	0	00	202400
01	2010ffff	0	0	01	000008
02	2010ffff	0	0	02	085002
03	2010ffff	0	0	03	100100
04	2010ffff	0	0	16	040400
05	2010ffff	0	0	17	020010
06	2010ffff	0	0	18	080200
07	0274402a	0	0	00	202400
08	0274402a	0	0	01	000008
09	0274402a	0	0	02	085002

—— 实际输出 ——　　展示原始输出

Cnt	Instr	equal	IntR	mAddr	cBus	
00	2010ffff	0	0	00	202400	
00	2010ffff	0	0	01	000008	15
01	2010ffff	0	0	01	000008	
01	2010ffff	0	0	02	085002	13
02	2010ffff	0	0	02	085002	
02	2010ffff	0	0	03	100100	14
03	2010ffff	0	0	03	100100	
03	2010ffff	0	0	16	040400	14
04	2010ffff	0	0	16	040400	
04	2010ffff	0	0	17	020010	12

图 1.15 微程序控制器 Educoder 测试结果

原因分析：微指令寄存器 μAR 设置成了上升沿触发。在时钟的上升沿，CPU 进行各个寄存器的锁存操作，在触发器翻转前后这段时间内，触发器的输入应该保持稳定。然而，一旦微指令寄存器设置为上升沿触发，微指令也在时钟上升沿发生变化；而微指令中的控制信号直接管制着各锁存器的输入，因而使得触发器不能可靠翻转。

解决方案：将微指令寄存器设置为下降沿触发，从而错开数据锁存和状态切换的过程，保证触发器可靠翻转。

1.5 测试与分析

Educoder 的评测关全部通过。对于支持中断的微程序单总线 CPU，将 sort-5-int.hex 放入内存并运行，无中断情况下运行结果如图 1.16 所示，可以看到从内存 0x80 开始的降序序列。

```
000 23bd0400 2010ffff 20110000 ae300200 22100001 22310004 ae300200 22100001
008 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200
010 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004
018 ae300200 20100000 2011001c 8e130200 8e340200 0274402a 11000002 ae330200
020 ae140200 2231fffc 12110001 1000fff7 22100004 2011001c 12110001 1000fff3
028 1000ffff 23bd0008 afb00000 afb10004 20310240 8e300000 22100001 ae300000
030 ae300004 ae300008 ae30000c ae300010 ae300014 ae300018 ae30001c 8fb10004
038 8fb00000 23bdfff8 42000018 23bd0008 afb00000 afb10004 20310280 8e300000
040 2210ffff ae300000 ae300004 ae300008 ae30000c ae300010 ae300014 ae300018
048 ae30001c 8fb10004 8fb00000 23bdfff8 42000018 00000000 00000000 00000000
050 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
058 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
060 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
068 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
070 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
078 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
080 00000006 00000005 00000004 00000003 00000002 00000001 00000000 ffffffff
```

图 1.16 现代时序 CPU 无中断运行结果

支持中断的现代时序 CPU 在一次按键 1 中断的情况下的测试结果如图 1.17 所示，可以看到 0x90 开始的八个字节内容全部加 1。

```
080 00000006 00000005 00000004 00000003 00000002 00000001 00000000 ffffffff
088 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
090 00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
098 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

图 1.17 现代时序 CPU 运行 1 次 1 号中断结果

两次按键 1 号中断和两次 2 号中断的情况下测试结果如图 1.18 所示，可以看到 0x90 开始的八个字节内容全部加 2，从 0xa0 开始的八个字节全部减 2（补码表示为 0xfffffffe）。可见该 CPU 可正确支持中断响应。

```
080 00000006 00000005 00000004 00000003 00000002 00000001 00000000 ffffffff
088 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
090 00000002 00000002 00000002 00000002 00000002 00000002 00000002 00000002
098 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0a0 fffffffe fffffffe fffffffe fffffffe fffffffe fffffffe fffffffe fffffffe
```

图 1.18 现代时序 CPU 运行 2 次 1 号中断和 2 次 2 号中断结果

注意到中断请求 2 发出后，中断控制器右侧对应中断号 2 的红灯亮了一段时间，如图 1.19 所示。说明中断 1 正在被处理。此时应有 IE＝0，这个单级中断系统在这段时间内将屏蔽任何中断请求；因此在中断请求未响应完成前再次触发中断 1 或 2 都不会处理新的中断请求。

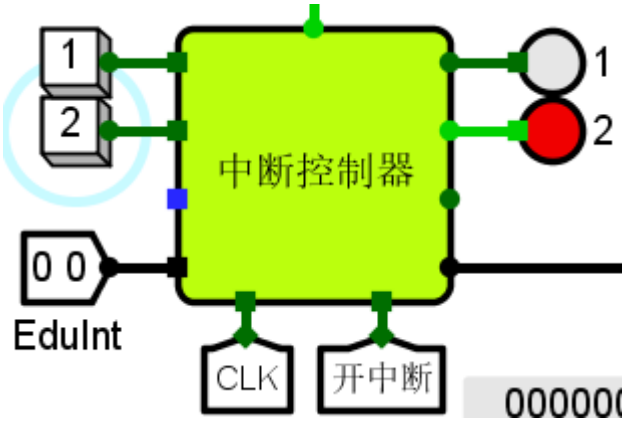


图 1.19 处理请求时的中断控制器

1.6 实验总结

本次实验主要完成了如下几点工作：

1） 设计了支持 6 条简单指令的 MIPS 定长指令集的指令译码器、微程序入口查找逻辑电路、条件判别测试逻辑电路以及保存微程序的控制存储器。

2） 设计了微程序的编码。

3） 设计了基于现代时序的硬布线状态机。

4） 完成了现代时序硬布线控制器、微程序控制器的设计。

5） 实现了基于硬布线控制器、微程序控制器的单总线 CPU。

6） 实现了 CPU 对中断的处理。

7） 成功运行了一个简易的冒泡排序程序。

1.7 实验心得

1） 通过本次实验，我了解并逐渐熟练应用 logisim 中各个部件，也熟悉了之前从未使用过的陌生器件，如多路选择器、译码器、三态门等，并复习了上个学期中数字电路和逻辑设计的知识。

2） 加深了对单总线 CPU 的结构的理解。CPU 是进行数据加工、程序控制、操作控制、时序控制、异常控制的部件，其中所有的控制操作都由控制器完成。当课上讲执行指令过程中 CPU 的数据流和控制流时，我一边感叹 CPU 控制器的精妙和复杂，一边苦恼如何实现能执行那么多指令的 CPU。本次实验虽然只实现了支持 6 条指令的 CPU，但引导我找到了合适的设计方法和策略，对于各部件的作用和他们整体的结构也更加得心应手了。

3） 加深了对单总线 CPU 程序运行方式的理解。指令执行一般包括取指令、指令译码、操作数地址计算、取操作数、执行指令、操作数地址计算、存操作数，对于本次实验支持中断的 CPU 来说，还要根据是否有中断决定是否需要进入中断响应周期。这一部分在课程学习过程中一直觉得理解不到位，直到在实验中通过实践才加深了理解。

4） 计算机组成原理作为计算机科学与技术专业核心课程之一，在学习过程中我对计算机有了更深的认识与理解，每每看到精妙的计算机设计不禁被前人的智慧折服。平时的理论学习为我的实验打基础，实验更是在实践中对理论知识的强化，也暴露了我很多理解不到位的地方。“知者行之始，行者知之成。”大概说的就是这样吧。

5） 本次实验引导较为细致，大大减少了实验所需的时间，但也具有一定的局限性。实验中体力劳动内容不少而思考内容不够多，希望多添加一些思考问题，辅助理解；内容上只涉及了单总线的 CPU，而多总线 CPU 在如今计算机结构中也具有广泛的应用场景和较高的地位，希望可以加入部分相关内容。

• 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：田清林

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：_____