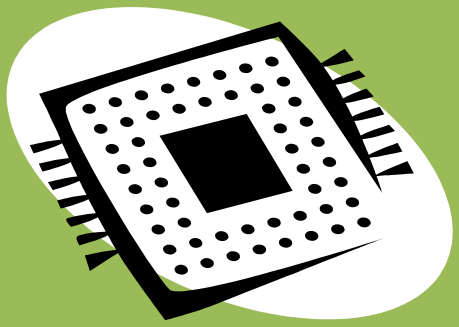
华中科技大学

2020

计算机组成原理 ·实验报告·

专　　业：	计算机科学与技术
班　　级：	CS1806
学　　号：	U201814788
姓　　名：	刘美
电　　话：	15673882367
邮　　件：	1527796339@qq.com
完成日期：	2020-12-07



计算机科学与技术学院

目 录

1 总线 CPU 设计实验

1.1 设计要求

能够结合定长指令周期三级时序系统的设计的知识以及利用该时序构造硬布线控制器，支持 5 条典型 MIPS 指令在单总线 CPU 上运行，最后 CPU 能够运行内存冒泡排序。本实验在 logisim 实验平台上基于已有的实验框架完成。

1.2 方案设计

1.2.1 MIPS 指令译码器设计

本关卡需要五条指令 lw,sw,beq,slt,addi，通过实验包给出的 MIPS 指令手册，分别查找五条指令的十六进制编码和 R 型指令的 op 及 funct 部分，利用比较器即可完成。

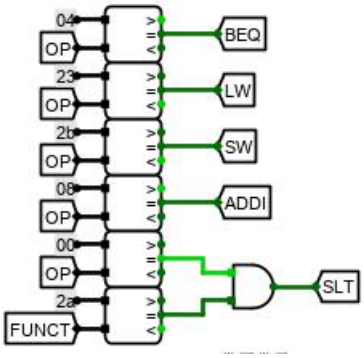


图 1-1 MIPS 指令译码器设计电路图

1.2.2 定长指令周期——时序发生器 FSM 设计

在定长指令周期设计中，所有 MIPS 指令都需要三个机器周期——取指周期 Mif、计算周期 Mcal、执行周期 Mex，每个机器周期有 4 个时钟节拍 T1、T2、T3、T4，故一共需要 12 个状态，将其按序编号为 0-11，按照当节拍脉冲到来时状态由现态转换为次态的原理可以得到 FSM 的设计，如图 1-2 所示：

图 1-2 时序发生器 FSM 图

1.2.3 定长指令周期——时序发生器输出函数设计

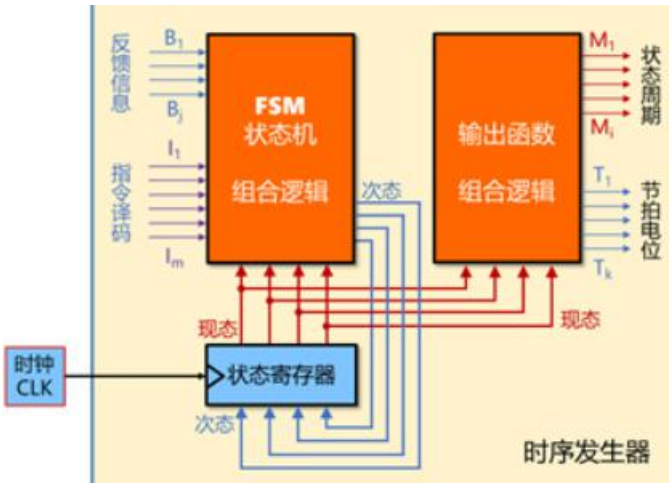


图 1-3 时序发生器逻辑图

时序发生器大体由两部分组成，一个是 FSM 状态机部分，状态机的次态输出送到状态寄存器的输入端，当时钟触发时，次态信息从状态寄存器输出端输出并送到状态机的输入作为下一时刻的现态。这样随着时钟触发到来，状态机和状态寄存器将按照 FSM 设计图一样进行状态转换。同时状态寄存器输出的次态信息还被输送到输出函数的输入端，通过输出函数器件生成机器周期信号和节拍信号。输出函数为组合逻辑，输入为状态寄存器的现态输出，输出为状态寄周期电位和节拍电位信号。填写表格时即可理解为当为某现态时，该状态对应处于三个机器周期的哪一个机器周期及机器周期中四个节拍的第几个节拍。例如，当现态为 S0 时，对应正在执行取指周期 Mif 的第一个节拍 T1，所以表格上现态为 0 时，Mif、T1=1；当现态为 S4 时，对应正在执

行计算周期 Mcal 的第一个节拍，所以表格上 Mcal、T1=1。同理可以得到其它的现态对应的输出。对应所填写的表格如下图 1-4 所示：

当前状态(现态)					输出						
S3	S2	S1	S0	现态10进制	Mif	Mcal	Mex	T1	T2	T3	T4
0	0	0	0	0	1			1			
0	0	0	1	1	1				1		
0	0	1	0	2	1					1	
0	0	1	1	3	1						1
0	1	0	0	4		1		1			
0	1	0	1	5		1			1		
0	1	1	0	6		1				1	
0	1	1	1	7		1					1
1	0	0	0	8			1	1			
1	0	0	1	9			1		1		
1	0	1	0	10			1			1	
1	0	1	1	11			1				1

图 1-4 输出函数真值表

1.2.4 硬布线控制器组合逻辑单元设计

在硬布线控制器组合逻辑单元中，其输出为微操作控制信号 Cn，输入为指令译码信号 Im、状态周期电位 Mi、节拍电位 Tk 和由微操作控制信号构成的反馈信号 Bj。

所以得到节拍信号的组合逻辑函数：

$$C_n = \sum_{m,i,k,j}(I_m \cdot M_i \cdot T_k \cdot B_j)$$

可解释为：在机器周期 Mi 的 Tk 节拍时，如果有指令译码信号 Im（如 lw 指令信号）和微操作控制反馈信号（如 equal 信号）则仅当其均为 1 时，对应执行译码为 Im 信号的指令的微操作，并生成执行该微操作需要的控制信号 Cn。

图 1-5 输出信号真值表示例图

如上图所示，在执行指令 lw 的执行周期的第一个节拍时，Mex 和 T1 均为 1，且输入的指令译码信号为 lw 信号，不存在反馈的控制信号，此时对应执行 Z->AR 的操作，所以其对应的控制信号为：Zout，ARin，所以这些信号输出为 1，其余为 0 不填。

1.2.5 定长指令周期——硬布线控制器设计

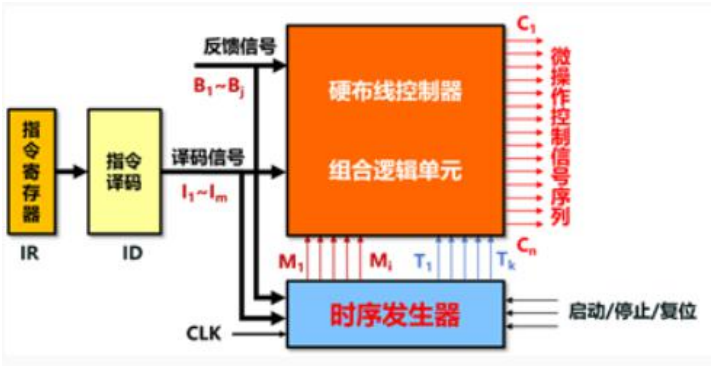


图 1-6 硬布线控制器逻辑图

整个硬布线控制器由三个部分构成，一个是指令译码器，能够将从 IR 指令寄存器中取出的指令译码得到译码信号 I1-Im，具体表现为 lw，sw，beq，addi，slt 五种指令信号以及 other 除了前面五种以外不用管的其他指令信号；该译码信号作为时序发生器的输入，通过时序发生器，在时钟的触发下可以实现状态之间的转换并同时生成与当前状态对应的机器周期和节拍电位信号。这些时钟电位信号和译码信号以及微操作控制信号作为的反馈信号一起作为硬布线控制器的输入通过硬布线控制器生成新的微操作控制信号序列。

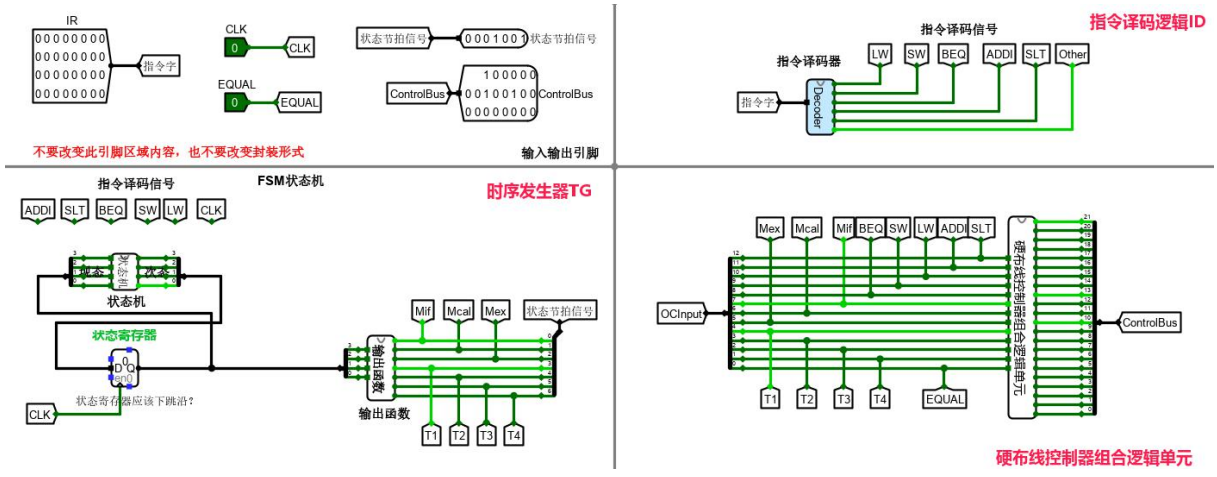


图 1-7 硬布线控制器电路图

在图 1-7 所示的电路中，时序发生器部分的状态寄存器的时钟应该选择下降沿，因为除了寄存器之外其它组件的跳变都是上升沿触发，为了让其他计算在始终跳变之前完成，防止所有触发都集中在下降沿导致。当选择上升沿时，测试样例的结果将出现大片规律性错误（间隔两个错一个）。

1.2.6 定长指令周期——单总线 CPU 设计

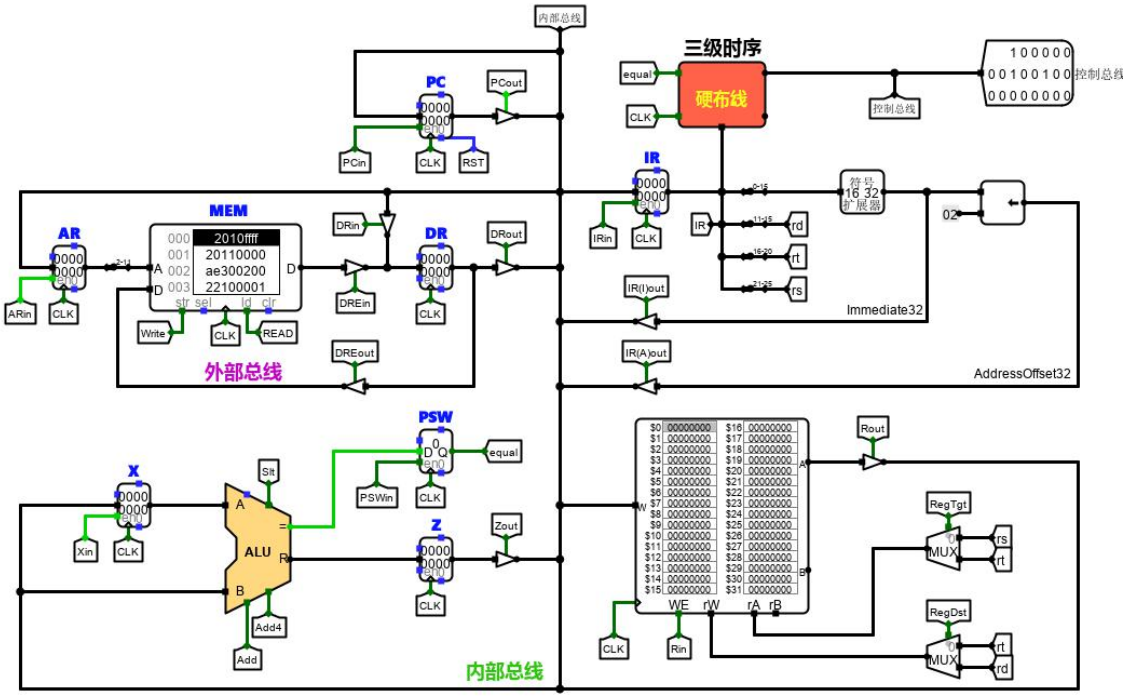


图 1-8 单总线 CPU 设计

在完成前面的实验有了译码器、状态机、控制器的部件后，对于给定的 CPU 框架，需要的功能部件硬布线控制器已经实现，将反馈信号 equal 信号和时钟信号 clk 作为硬布线部件的输入连接到器件上，并输出控制总线可以查看对应的控制信号的值。ALU 运算部件的两个操作数一个来自总线，一个来自 X 寄存器，运算结果存放在 Z 寄存器中。该运算器可以进行 add，addi，slt 三种运算，并将运算过程中产生的状态信息（如 equal）放到 PSW 状态字寄存器中。寄存器部件通过寄存器选择控制信号可以将需要的寄存器的值输送到总线上。此时要实现 sort-5 程序只需要将要测试的文件 sort-5.hex（保存了冒泡排序代码机器指令的文件）直接加载到 RAM 存储器中，这样当需要执行指令时通过取指指令从主存中获取指令送到指令寄存器再根据指令译码，通过硬布线生成的控制信号控制各个功能部件执行相应的操作。在整个执行过程中，数据都通过一根总线进行传输，数据传输到总线需要相应的控制信号打开三态门。

1.3 实验步骤

表 1-1 educoder 通关测试

关卡序号	关卡设计
1	MISP 指令译码器设计
2	定长指令周期--时序发生器 FSM 设计
3	定长指令周期--时序发生器输出函数设计
4	硬布线控制器组合逻辑单元
5	定长指令周期--硬布线控制器设计
6	定长指令周期--单总线 CPU 设计

在 educoder 上根据关卡逐步完成电路的设计。

1.4 故障与调试

1.4.1 beq 指令跳转异常

故障现象：执行 beq 指令时对比测试用例发现结果的现态不对，答案对应的 controlbus 为 400C0，但是我的为 400E0

原因分析：对比正常和实际运行结果的 controlbus 的值可以得出时 beq 的控制信号填写不对，一开始填写表格的时候因为没有仔细看视频，不知道 ALU 运算器可以自动实现减法，自认为没有 SUB 的话输出 ADD 的控制信号即可，结果反而是画蛇添足。除此以外，控制器部分对 beq 的跳转分支的地址填写不对，初始设置成了 0x0f，没有了解这里跳转的根本意思。

解决方案：我首先是根据测试用例的 controlbus 的值直接修改我对应的控制信号（因为 controlbus 是由控制信号组成的十六进制值），修改完毕后，发现问题出现在多添加了 ADD 指令。修改通过测试后，思考了一下错误的地方，明白了错误的原因。对于 beq 的跳转，因为 beq 有两个跳转分支，在控制器中需要选择跳转的入口应该是判断条件满足，即两个寄存器的值相等，equal 信号输出为 1 时跳转到下一个状态即下一条执行过程的地址 0x10。

1.4.2 slt 指令错误

故障现象：当执行到 slt 指令，输出 controlbus 与给出样例不同，样例的 slt 第一个节拍和第三个节拍的 controlbus 值分别为 40400，80220，但我的是 40404 和 80224

原因分析：不了解 slt 作为输出信号的作用，尽管老师提示 slt 是 slt 的特殊运算，还是没搞清具体意义，在 slt 指令执行的时候，将 slt 输出信号全部置 1。

解决方案：事实上 slt 作为输入信号表示执行 alu 的有符号比较运算，即小于置 1 运算，而输出的 slt 控制信号的获得是通过比较得到的，这个比较的执行在 slt 指令执行周期的第二个节拍获得，所以只在该节拍会输出为 1，其他情况下不需要输出为 1。

1.5 测试与分析

测试到节拍数为 0xbbb，指令执行条数为 251 条，程序执行完毕。最后实现有符号数的降序排序。

2 单总线 CPU（现代时序）

2.1 设计要求

能够通过本次实验理解变长周期三级时序系统的设计，能够利用该时序构造硬布线控制器，支持 5 条典型 MIPS 指令在单总线 CPU 上运行，最终 CPU 能运行内存冒泡排序。

2.2 方案设计

2.2.1 MIPS 指令译码器设计

这一部分的设计和 1.2.1 节一样。

2.2.2 单总线 CPU 微程序入口查找逻辑设计

机器指令译码信号					微程序入口地址					
LW	SW	BEQ	SLT	ADDI	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

图 2-1 入口查找逻辑真值表

因为该 CPU 在执行指令时按照一个节拍执行一次操作对应一条微指令从而对应有一个微指令地址，所以可以将指令按照 lw->sw->beq->slt->addi 的顺序，并且每条指令按照取指周期->计算周期->执行周期，且每个机器周期按照 T1->T2->T3->T4 的节拍顺序对所有的微指令进行顺序编址（编序号）。首先五条指令分别要先执行四个节拍的取指指令，然后在计算周期 Mcal 和执行周期 Mex 中，lw、sw、beq 指令共需要五个节拍，slt 和 addi 共需要三个节拍（全部设定为在执行周期），而在指令执行前的取指周期都需要 4 个节拍，所以可以得到取指指令和五条指令对应的入口地址分别为：

取指指令——00000（0），lw——00100（4），sw——01001（9），beq——01110（14），slt——10011（19），addi——10110（22）

入口查找逻辑将实现根据输入的译码信号（指令信号），输出相应的入口地址。

故可以得到对应的真值表如图 2-1 所示。

2.2.3 单总线 CPU 微程序条件判别测试逻辑设计

对于条件判别位，在后面的课程复习过程中对判别位有了进一步的理解：首先判别位只是代表一个测试，当判别位为 1 时代表进入相关的条件测试（而不是代表测试结果为真直接跳转），只有当条件判别测试的结果为真才会进行分支跳转，否则就取下址字段。P0 的优先级是最高的，只要 P0 为 1，就会选择微程序入口地址对应的端口，即 musel=1；当 P0 为 0，且 P1=1，equal=0 时意味着 beq 指令选择条件判别为错或者 P1=0 意味着尚未执行到条件判别指令的地方，则下一条指令的地址应该来自下址字段，即 muxsel=0；而当 P1=1 且 equal=1 时对这五条指令来说意味着 beq 指令的判别条件正确，下一个微操作应该执行的时 PC->X，所以对应的地址应该选择 beq 分支地址，即 muxsel=2。

2.2.4 单总线 CPU 微程序控制器设计

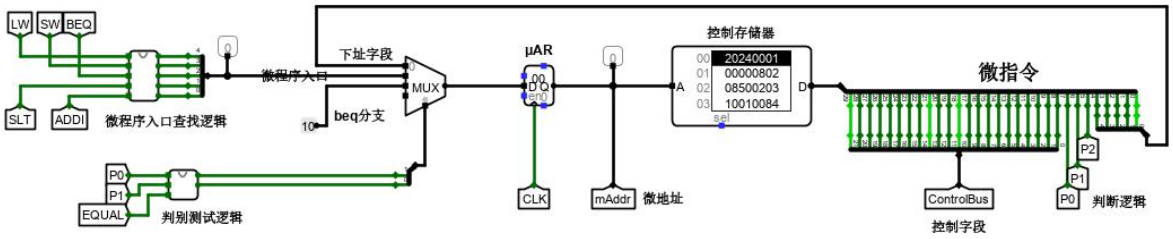


图 2-2 单总线 CPU 微程序控制器设计

在本关的设计中，指令译码信号通过微程序入口查找部件可以得到对应的微程序入口地址，同时指令的判别字段输入到判别测试部件生成微地址选择信号，选择下一条输入到微地址寄存器的到底是来自 0 号下址字段或者是 1 号微程序入口地址，或者 2 号 beq 分支，选择器的后续地址输出将送到微地址寄存器，该寄存器的输出地址将可以访问存储在控制存储器中的微指令得到下一条微指令字，循环这个过程就可以实现整个指令的控制。

按照上述原理连接好电路如图 2-2 所示，还需要将所有的微指令载入控制存储器中。微指令是由许多微命令也就是控制信号以及判别字段和下址字段构成的，每一条微指令对应了一条指令在一个节拍内的执行情况。故我们可以填写一个微程序控制信

号真值表，再由该真值表得到电路图。

在电路的真值表中，输入是不同的态序号，该序号的得来是按照 2.2.2 节的编码思想对五条指令的节拍状态进行编序号，一个节拍对应一个状态序号，也对应了一条微指令一次微操作，这个微操作的执行需要一些控制信号，根据该指令的具体译码情况，可能还需要进行一些判别测试，所以输出就是与之相对应的控制信号和判别字段、下址字段。对于下址字段则就是执行完这条微指令接下来要执行的下一条微指令对应的状态序号，若该条微指令是取指指令的最后一条微指令，则因为执行完这句话之后需要按照具体的译码情况进行跳转，也就是选择微程序入口地址作为下一条执行的微指令的地址，所以下址字段其实是无关的（在这里我们可以填序号 4）；在每一条指令执行完毕后都需要返回到取指指令，故每一条指令的最后一条微指令的下址字段一定是取指指令的入口地址 0；否则一般而言，下址字段就是当前微指令的状态序号的下一个序号（即顺序执行）除非还有由于判别测试成立而要跳转的情况则根据具体情况进行跳转。写判别字段时，若该节拍微指令需要根据微程序入口地址进行跳转，则判别位 P0 需要置 1；若执行完某条指令后需要进行 equal 判别测试判断下一步跳转的分支，需要将 P1 和 equal 置 1。下图 2-3 列出了状态为 S4 时，对应的执行 lw 指令在计算周期 Mcal 的 T1 节拍，此时执行微操作 R[rs]->X，所以需要的该输出的控制信号为 Rout、Xin，这两个信号位置填 1，其余不填，而此时尚未执行到该指令的最后 一个节拍且不需要进行其它的判别测试（equal），故判别字段中 P0，P1，equal 都不用填，下址字段就是下一个状态的序号 5。其他状态序号的输出也可按照该原理得到。

图 2-3 lw 的 S4 状态的真值表填写示例

填完真值表后，再将得到的微指令十六进制数载入控制存储器中即可实现设计。

2.2.5 采用微程序的单总线 CPU 设计

设计思路和 1.2.6 节一样，只不过是将硬布线控制器替换为了微程序控制。

2.2.6 现代时序硬布线控制器状态机设计

将所有的状态按照 2.2.2 的思想进行编号后，可以设计一个真值表填写状态之间的

转换。真值表的输入为现态，输出为次态。状态之间转换的原理应该是这样：每一条指令当执行完最后一条取指微指令后，根据具体的译码信号（lw，sw 等输入信号），跳转到对应的指令信号的第一个状态——lw 的 S4，sw 的 S9，beq 的 S14，add 的 S19，slt 的 S22；当 beq 指令在 S15 状态时判别测试为真即输入 equal 信号为 1 时，应当跳转到 S16 状态继续执行，否则和所有已经执行完毕的指令一样返回到取指指令的第一个状态即 S0 状态，若当前状态执行时不存在判别测试或输入译码信号导致的跳转，则将按照顺序转换，即转换到下一个节拍对应的序号如 S5->S6。故真值表填写如图 2-4 所示：

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	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	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	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								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	1	1	1	1	15								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	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

图 2-4 现代时序硬布线控制器状态机真值表

2.2.7 现代时序硬布线控制器设计

在该设计中，大体的思路和 1.2.5 一致，由指令译码部件和由硬布线状态机和状态寄存器一起构成的时序发生器和控制存储器构成，但此时我们不再需要输出函数部件和硬布线组合逻辑部分，因为一条微指令对应的一个节拍，所以状态寄存器的输出将得到当前的微指令所处的状态，根据 2.2.3 节的思想由这个状态我们将得到唯一的微指令，这条微指令对应的十六进制数由 2.2.3 节填写的表格得到并被存储在控制存储器中，这样访问控制存储器就可以得到这条微指令。电路如图 2-5 所示：

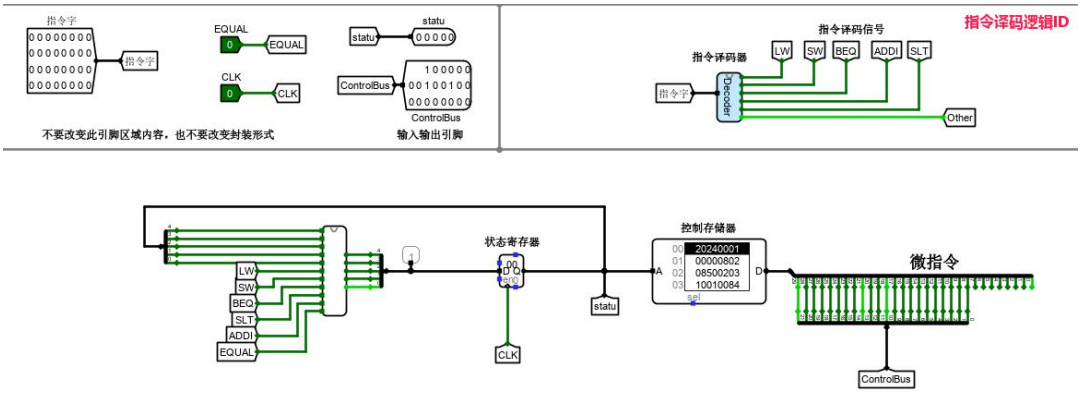


图 2-5 现代时序硬布线控制器电路图

2.3 实验步骤

表 2-1 educoder 通关测试

关卡序号	关卡设计
1	MISP 指令译码器设计
2	单总线 CPU 微程序入口查找逻辑
3	单总线 CPU 微程序条件判别测试逻辑
4	单总线 CPU 微程序控制器设计
5	采用微程序的单总线 CPU 设计
6	现代时序硬布线状态机设计
7	现代时序硬布线控制器设计

按照 educoder 上的关卡逐关完成电路设计即可。

2.4 故障与调试

2.4.1 现代时序硬布线状态转换出现错误

故障现象：educoder 现代时序硬布线状态转换测试时，当执行 beq 指令处于 S15 状态，且 equal 为 1 时，次态应该是 S16，但实际跳转的次态为 S0

原因分析：对 equal 作为条件判别测试输入信号疏忽了，表格中不填代表的是无关而不是为 0。

解决方案：在 beq 指令跳转时 equal 为 0 必须填写。

3 现代时序中断机制实现

3.1 设计要求

能够理解现代时序控制器中断机制的实现原理，能为采用现代时序单总线结构的MIPS CPU 增加中断处理机制，可实现多个外部按键中断事件的随机处理，本实验需要完成现代时序微程序控制器的基础上完成，需要增加硬件数据通路，增加中断返回指令 eret 的支持，需要中断服务程序配合。

3.2 方案设计

3.2.1 MIPS 指令译码器设计

设计方案同 1.2.1。

3.2.2 支持中断的微程序入口查找逻辑

按照 2.2.2 的编序号思想，在 lw，sw，beq，addi 和 slt 已经编号的基础上，新增一条 eret 指令，该指令照样需要先经过取指指令的四个节拍，但在计算周期和执行周期，只需要执行一个节拍，而前一条指令 addi 的入口地址为 22 且执行三个节拍拥有三条微指令地址 22，23，24，所以排在其后的 eret 的入口地址为 25。

3.2.3 支持中断的微程序条件判别测试逻辑

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

图 3-1 判别测试逻辑真值表

同 2.2.3 的条件判别测试设计类似，我们可以通过填写一个条件判别测试的真值表获得输入为判别位和测试信号时对应的输出选择。在本次实验中，由于使用计数器法代替了下址字段法，还加上了中断，这时候判别条件应该在微程序入口地址和 beq 分支地的基础上还要增加两个——中断响应入口地址和取指微程序入口地址。中断响应地址在可以执行中断处理时实现跳转，而设置取指微程序入口地址是因为在下址字段法时我们用下址字段为 0（取指微指令入口地址）表示而实现跳转，但在计数器法中，由于计数器是由加法器对操作数加一构成的，当我们执行完指令的最后一条微指令想要跳转到取指微程序时，计数器无法将当前状态的序号变到 0，所以必须添加一个取指微程序入口地址判别。因此我们可以设置输出为 0，1，2，3 分别对应顺序地址，微程序入口地址，beq 分支地址，中断响应入口地址，取指微程序入口。

在 2.2.3 中我们对微指令设置了 2 个判别位，P0 为判别是否选择微程序入口地址（在本次实验中用于取指指令），P1 用于判别 beq 的 equal 测试，当增加了中断之后，根据中断是在指令执行完毕后执行的特点，我们还需要额外增加一个判别位 P2，用于判别正在执行的微指令是否为指令的最后一条微指令，如果是，应该将 P2 置为 1，这样，就可以接着进行中断测试。对中断来说，除了 P2 为 1 表示允许有中断以外，要产生中断还需要有中断请求信号，设置为 INTR=1，除此以外还要没有别的分支需要执行时才会进入中断处理程序，故我们可以得到进入中断处理分支的条件为 P1&IntR&~(P2&*equal*)，根据该条件可以罗列出对应输出为 3 时输入取值情况。

在本实验中，我们不再采用下址字段法而是通过计数器得到下一条微指令的入口地址，当微指令可以顺序执行也就是所有判别位为 0 且测试信号和中断信号都为 0 时，可以直接由当前状态顺序执行到下一个状态，对应真值表我们可以认为当输出为 0 时，P0,P1,P2,equal,IntR 均为 0。

由 2.2.3 我们知道 P0 代表取微程序入口地址，且其优先级最高，所以选择微程序入口地址的条件为 P0，对应真值表输出为 1 时，P0 为 1，其余位不填（表示无关）。

何时执行 beq 的分支呢，我们知道要让 beq 跳转到分支语句，首先要进行 equal 测试，所以 P1,equal 为 1，而此时若测试成立，即便有中断信号到来，也不会转入中断程序，因为成立意味着当前微指令不是 beq 指令的最后一条执行的微指令，所以和 Intr 输入信号无关。同时如果成立，应该跳转到 beq 分支而不是取微程序入口地址，

所以此时 P0 应该置为 0.因此，P0=0,P1=equal=1，Intr=0/1 时，判别为 beq 分支，输出为 2。

对于返回取指微程序，可以存在以下几种条件：第一，对非 beq 的其他五条指令，若 P2=1，Intr=0，意味着是最后一条微指令且没有中断，则返回取值微程序；第二，对于 beq 指令，若判别测试不成立，即 P1=1，equal=0，这时肯定是 beq 的最后一条微指令，此时，只要不满足中断的条件即 P2=1&Intr=1，就不会进入中断而是跳转到取指微程序。

根据上述思路，我们可以得到填写的真值表如图 3-1 所示。

3.2.4 支持中断的微程序控制器设计

该部分的设计包括地址转移组合逻辑部件，微地址寄存器和控制存储器以及指令译码部件。对地址转移组合逻辑部分，和 2.2.4 的设计思想类似，不过由于计数器法和中断的设计，入口选择设置了四个。按照 2.2.4 的设计思想和 3.2.3 的原理可以完成选择器部分的连线。根据 2.2.2 的编号思想，由 3.2.1 我们知道 eret 指令的入口地址为 25，而中断处理程序除了 eret 这条返回指令执行一个节拍以外，还包括关中断、保护现场一个节拍和取中断程序入口地址一个节拍将其编号为状态 S26,S27，对应微指令地址为 26，27，且这些微指令的执行都应该在其他指令执行完毕之后，并且取完中断入口程序地址后应该进入中断程序并执行，所以 S27 之后应该跳转到取指指令，所以我们可以得到如图 3-2 所示的状态转换图（在这里出现了 add 指令，但实际上它和 slt 指令的执行节拍数是一样的所以不影响整体的编号）。由图 3-2 我们即可得知 beq 分支地址为 0x10，中断响应入口地址为 0x1a.因为这里采用的是计数器法而不是下址字段法，所以可以使用一个加法器，加法器的一个当前微指令地址，另一个操作数为 1，因而实现顺序执行，加法器的输出结果送到选择器的 0 号端口。控制器中的控制存储器需要加载指令的十六进制数，该十六进制数的获得同 2.2.4 设计一样。得到电路如图 3-3 所示

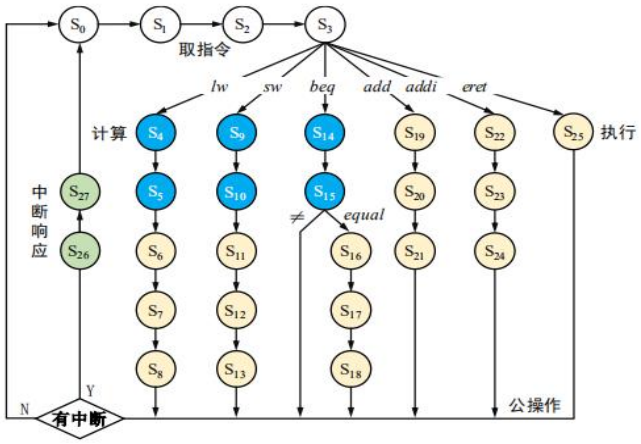


图 3.2 支持中断的微程序状态转换图

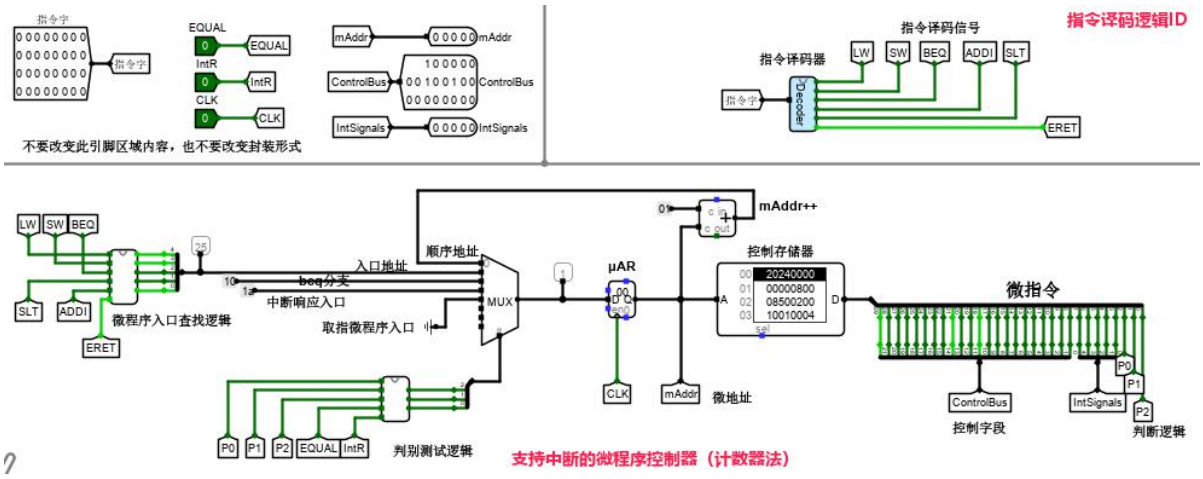


图 3-3 支持中断的微程序状态转换电路图

。

3.2.5 支持中断的微程序单总线 CPU 设计

该部分需要完成中断逻辑电路的设计。为了实现该设计，需要增加一些辅助部件，包括:为了保护现场需要一个 EPC 寄存器；为了实现中断控制需要一个中断控制器，用于生成中断请求信号和中断请求，还要设计一个选择部分，可以实现对产生的多个中断请求选择优先级最高的那个，并且获取其程序入口地址以及为了实现开关中断和中断使能信号需要一个中断使能寄存器。

当有中断产生时，EPC 寄存器在 EPCin 控制信号下保存 PC 值，中断控制器产生中断请求信号，中断使能寄存器关中断并和中断请求信号一起生成中断使能信号表示可以中断，同时控制器产生的多个中断经过选择器选择后选定一个中断，并获取入口

地址并输送到总线。当中断结束时，由相应的中断控制信号，中断使能寄存器在开中断控制信号下开中断，EPCout 控制 EPC 寄存器将 PC 的值输送到总线。

按照该思路，可以得到中断部分实现电路如图 3-4 所示

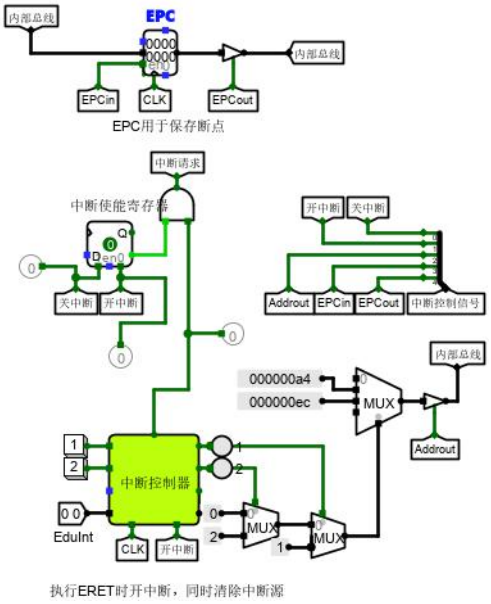


图 3-4 中断部分电路图

实现了中断部分后，其余设计和 2.2.5 一样。

3.2.6 支持中断的现代时序硬布线控制器状态机设计

经过 3.2.4 的分析我们已经得到了支持中断的状态转换，采取和 2.2.6 一样的设计思想只需要在 2.2.6 的表格的基础上，按照图 3-1 的状态图填写即可得到支持中断的状态机真值表，从而由该表自动生成电路。

3.2.7 支持中断的现代时序硬布线控制器

该设计原理和思想和 2.2.7 一样，电路图也如 2-5 所示。

3.3 实验步骤

表 2-1 educoder 通关测试

关卡序号	关卡设计
1	MISP 指令译码器设计
2	支持中断的微程序入口查找逻辑
3	支持中断的微程序条件判别测试逻辑
4	支持中断的微程序控制器设计
5	支持中断的单总线 CPU 设计
6	支持中断的现代时序硬布线状态机设计
7	支持中断的现代时序硬布线控制器设计

按照 educoder 上的关卡逐关完成电路设计即可。

3.4 故障与调试

3.4.1 对添加中断的状态图不够理解

故障现象：支持终端的微程序控制器测试时多个答案的 cbus 和实际的 cbus 不同

原因分析：基于之前学过的内容，我自动把中断程序的执行过程：关中断、保护现场——进入中断处理程序——中断返回作为状态转换的顺序，即把关中断指令作为 S25 放在取指之后跳转，进入处理程序作为 S26，eret 作为 S27，即便书上给出了状态转换图也没搞清楚状态执行的实际流程。

解决方案：复习后，我对这个状态图有了更深的理解，首先 eret 是一条无操作码的指令，其执行应该是在取指完成之后执行，而关中断和进入中断对应的应该是一个节拍要执行的，且是在一条指令执行完毕后才有可能执行的。整个中断的执行过程应该是：先执行完正常指令之后收到中断请求进入中断处理程序，此时中断处理程序也按照取指——执行指令——判断中断（此时若还发生中断则为嵌套中断），中断执行完毕后，取指——中断返回。

4 总结与心得

4.1 实验总结

本次实验主要完成了如下几点工作：

1） 设计了计算机数据表示实验、运算器设计实验、存储系统设计实验以及单总线 CPU 定长指令周期 3 级时序和现代时序以及添加中断机制后的现代时序实验

2） 实现了汉字获取、偶校验和海明校验的编码解码，4/16/32 快速加法器、补码一位乘法、阵列乘法器、MIPS 运算器、MIPS 寄存器和 RAM 以及四路组相连的 cache、单总线 CPU 添加/不添加中断的设计

4.2 实验心得

通过本课程的实验，我对计算机的一些基础部件——运算器，CPU，存储器等有了更深的理解。数据表示部分我主要收获了海明校验对一位错和二位错的区分（增加总偶校验），运算器部分我主要收获了并行技术，并结合 PPT 的时延计算了解到并行相比于串行的快速。存储器部分我主要收获了如何涉及 cache 槽，工作量虽然大但是理解了整个流程之后设计其实并不复杂。重点部分的涉及在于淘汰计数部分淘汰组中的哪一路（优先编码器的使用）。CPU 实验我主要收获了 CPU 对指令的执行的流程和各个部分到底如何具体实现，尤其是添加中断后还需要添加哪些部件（中断控制、IR，MUX 等）。

实验设计的很好，老师很负责，在做中断的时候遇到第三关中断判别测试样例给的不够充分导致面对样例通关的实验选手第四关迟迟过不了，向老师反映之后老师也立马完善了样例，但最终问题还是在于自己思考不透彻。

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 5 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,胡迪青.计算机组成原理实践教程.北京:清华大学出版社，2018 年.

[4] 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.

[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： _____

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）	报告撰写 （30 分）	课设过程 （70 分）	最终评定 （100 分）
得分			

指导教师签字： _____