

华中科技大学

2023

计算机组成原理

· 实验报告 ·

专 业： 计算机科学与技术

班 级： 计卓 2101

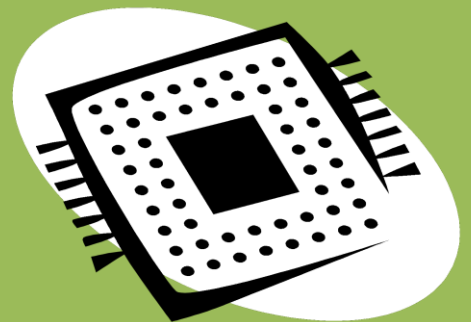
学 号： U202112071

姓 名： 王彬

电 话： 15005276201

邮 件： 2457537174@qq.com

完成日期： 2023-06-28



计算机科学与技术学院

1 CPU 设计实验

1.1 设计要求

本实验中，我们需要利用 Logisim 平台构建一个 32 位基于 MIPS 指令集的现代时序单总线 CPU，并以此为基础增加数据通路并升级控制器，编写中断服务程序，实现简单的单级中断处理机制。在完成的基础上，可进行系统联调，并在“单总线 CPU+中断（微程序）”子电路中加载 sort-int-5.hex 程序，通过时钟自动运行测试程序运行情况。

本实验的具体指令集如下表所示。

表 1-1 MIPS 指令集

序号	指令	格式	功能备注
1	Add	Add \$rd, \$rs, \$rt	指令功能及其格式参考 MIPS32 指令集
2	Add Immediate	Addi %rt, \$rs, imm	
3	Load Word	Lw \$rt, offset(\$rs)	
4	Store Word	Sw \$rt, offset(\$rs)	
5	Branch on Equal	Beq \$rs, \$rt, label	
6	Eret	Eret	停机

1.2 方案设计

MIPS 现代时序中断机制实现主要由指令译码器、微程序控制器+中断等部件组成。我们的设计思路是先对底层逻辑进行设计，诸如 MIPS 指令译码器、支持中断的微程序入口查询逻辑、条件判别测试逻辑，随后再具体地设计支持中断的微程序控制器与单总线 CPU 的设计。在完成对基于微程序控制器的单总线 CPU 的设计后，我们再对现

代时序硬布线控制器状态机和具体的控制器进行设计，以构成基于硬布线的单总线 CPU 设计。

1.2.1 现代时序中断机制底层逻辑设计

(1) 设计 MIPS 指令译码器

我们利用比较器等功能模块将 32 位 MIPS 指令字译码生成 LW、SW、BEQ、SLT、ADDI、OtherInstr 信号，分别进行比较后得到不同的指令信号。

(2) 支持中断的微程序入口查找逻辑

根据不同的指令，得到对应的微程序入口地址，并以此为微程序控制器中微程序分支实现跳转的操作。我们可以填写支持中断的微程序入口地址表（Excel 表），并生成其逻辑表达式。我们的指令变换状态图如图 1.1 所示。

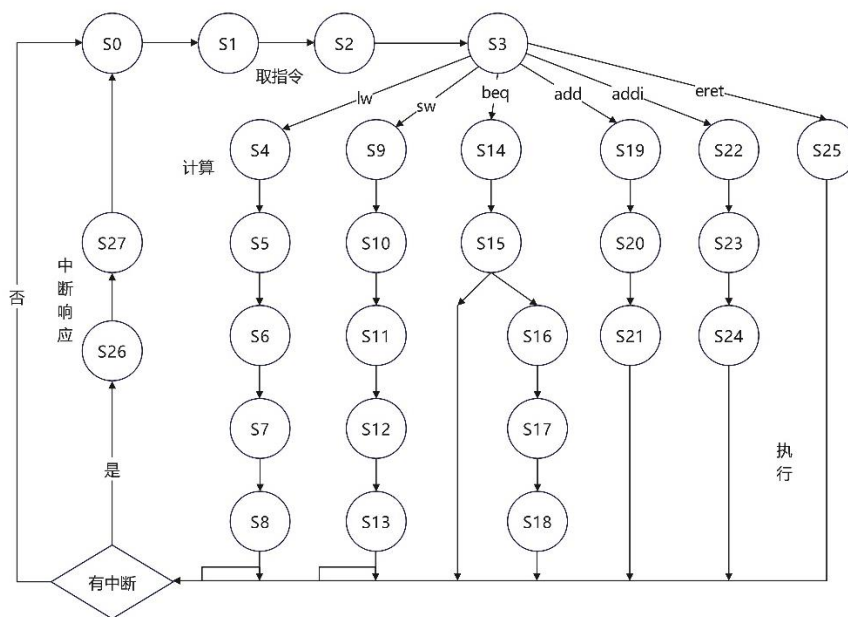


图 1.1 支持中断的现代时序状态机

因此我们根据相应指令的微程序状态入口，对微程序的入口查找逻辑进行支持。我们填写的微程序入口如图 1.2 所示。

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 1.2 微程序地址入口表

对于微程序的转移，如果无中断，状态机使用指令译码信号进行转移当且仅当状态机处于 S3 状态时，其余指令可以根据下址进行计数器累加跳转。

(3) 支持中断的微程序条件判别测试逻辑

状态机的后续状态转移需要通过一个多路选择器实现，其示意图如图 1.3 所示。为了对不同的分支地址进行控制，我们需要根据判别字段和状态条件对控制信号进行生成。

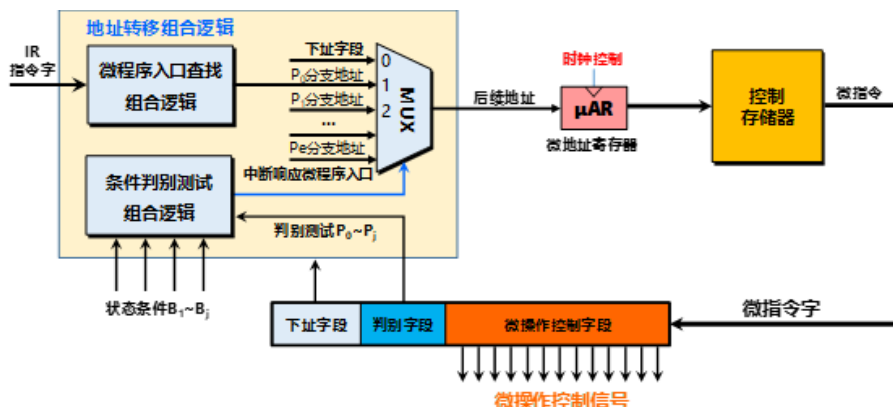


图 1.3 支持中断的微程序控制器原理图

不同的指令转移下址有着不同的控制，例如当 $P_0=1$ 时可以忽略其它信号的变化，转移地址都是下址字段。在本次设计中，我们的判别字段为 $P_0/P_1/P_2$ ，其中 P_2 位为 1 时说明当前微指令为本次指令的最后一条，则需要进行中断判断并做出相应的响应。同时具有比较状态信号 $equal/intR$ ，基于这些信号生成对应的控制信号 $S_0 \sim S_2$ 。我们对判别测试逻辑自动生成表达式表单进行填写，其结果如图 1.4 所示。

图 1.4 判别测试逻辑自动生成表达式

设计基于微程序的支持中断的单总线 CPU，我们需要对具体的微程序控制存储器进行编写，而后形成总体的单总线 CPU。

本微程序控制器的设计方式是在实验三中对普通单总线 CPU 的设计基础上，添加中断的指令转移和信号。我们需要对微程序的状态对应信号进行填写，生成微指令并输入控制器的控制存储器中，对微指令的具体实现做出支持，我们设计的微指令如图 1.5 所示。

4

图 1.5 支持中断的微程序控制存储器

之后我们将生成结果输入单总线 CPU 的控制存储器的存储区即可。

（2）支持中断的微程序单总线 CPU 设计

对于具体的单总线 CPU 进行设计，除了我们上述设计的控制器外，CPU 还需要具有下列功能部件：（i）指令计数器 PC；（ii）主存储器 MEM；（iii）指令寄存器 IR，用以存放当前指令；（iv）数据寄存器 DR；（v）寄存器堆（\$0~\$31）；（vi）内部总线；（vii）暂存器 X。

同时，为了支持中断请求，我们还需要引入中断逻辑。中断寄存器 EPC 用于保存断点指令 PC，同时我们使用中断控制器对中断行为实现控制。当执行 ERET 指令时，中断控制器将开放中断，并清除中断源。

1.2.3 基于硬布线的单总线 CPU 设计

硬布线控制器使用组合逻辑推导出微操作控制信号序列，并根据 FSM 状态机通过组合逻辑进行状态转移。本节我们着重设计支持中断的现代时序硬布线控制器。

（1）支持中断的现代时序硬布线控制器状态机设计

根据我们在图 1.1 绘制的现代时序硬布线 FSM 状态机简化状态图，我们对其中的状态转移进行设计。我们根据上述状态图写出状态转移表如图 1.6 所示。

*

1	当前状态(现态)						输入信号								下一状态(次态)					
2	S4	S3	S2	S1	S0	10进制	LV	SW	BEQ	SLT	ADDI	ERET	IR	EQUAL	次态10进制	N4	N3	N2	N1	N0
3	0	0	0	0	0	0									1	0	0	0	0	1
4	0	0	0	0	1	1									2	0	0	0	1	0
5	0	0	0	1	0	2									3	0	0	0	1	1
6	0	0	0	1	1	3	1								4	0	0	1	0	0
7	0	0	0	1	1	3		1							9	0	1	0	0	1
8	0	0	0	1	1	3			1						14	0	1	1	1	0
9	0	0	0	1	1	3				1					19	1	0	0	1	1
10	0	0	0	1	1	3					1				22	1	0	1	1	0
11	0	0	0	1	1	3						1			25	1	1	0	0	1
12	0	0	1	0	0	4									5	0	0	1	0	1
13	0	0	1	0	1	5									6	0	0	1	1	0
14	0	0	1	1	0	6									7	0	0	1	1	1
15	0	0	1	1	1	7									8	0	1	0	0	0
16	0	1	0	0	0	8									0	0	0	0	0	0
17	0	1	0	0	1	9									10	0	1	0	1	0
18	0	1	0	1	0	10									11	0	1	0	1	1
19	0	1	0	1	1	11									12	0	1	1	0	0
20	0	1	1	0	0	12									13	0	1	1	0	1
21	0	1	1	0	1	13									0	0	0	0	0	0
22	0	1	1	1	0	14									15	0	1	1	1	1
23	0	1	1	1	1	15									0	0	0	0	0	0
24	0	1	1	1	1	15							1		16	1	0	0	0	0
25	1	0	0	0	0	16									17	1	0	0	0	1
26	1	0	0	0	1	17									18	1	0	0	1	0
27	1	0	0	1	0	18									0	0	0	0	0	0
28	1	0	0	1	1	19									20	1	0	1	0	0
29	1	0	1	0	0	20									21	1	0	1	0	1
30	1	0	1	0	1	21									0	0	0	0	0	0
31	1	0	1	1	0	22									23	1	0	1	1	1
32	1	0	1	1	1	23									24	1	1	0	0	0
33	1	1	0	0	0	24									0	0	0	0	0	0
34	1	1	0	0	1	25									0	0	0	0	0	0
35	1	1	0	1	0	26									27	1	1	0	1	1
36	1	1	0	1	1	27									0	0	0	0	0	0
37	0	1	0	0	0	8							1		26	1	1	0	1	0
38	0	1	1	0	1	13							1		26	1	1	0	1	0
39	0	1	1	1	1	15							1		26	1	1	0	1	0
40	0	1	1	1	1	15							1	1	16	1	0	0	0	0
41	1	0	0	1	0	18							1		26	1	1	0	1	0
42	1	0	1	0	1	21							1		26	1	1	0	1	0
43	1	1	0	0	0	24							1		26	1	1	0	1	0

图 1.6 支持中断的现代时序硬布线状态机状态关系

我们将通过该表格生成的逻辑表达式填写入分析逻辑电路板块，并对电路进行自动生成即可。

(2) 支持中断的现代时序硬布线控制器设计

在实现指令译码、现代时序状态机模块后，最终实现硬布线控制器的集成，在下图中完成硬布线控制器框架连接，注意硬布线控制器组合逻辑不需要实现直接采用微程序控制器的控制存储器代替即可。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。

1.3 实验步骤

(1) 指令解析

考虑到 MIPS 指令集的定长码格式，我们将 32 位操作码用分线器接出并取得其最高六位操作码 OP，并分别对操作码进行比较器的等值比较，判别操作码对应的指令。我们实现的电路图如图 1.7 所示。

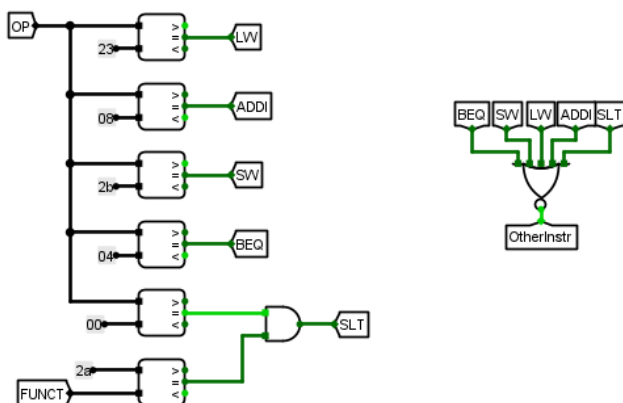


图 1.7 指令解析电路图

(2) 实现中断微程序入口查找逻辑

根据图 1.2 所示的表格，我们对相应的逻辑表达式进行生成并依次填入分析逻辑电路，实现对逻辑电路的生成。如图 1.8 所示，我们的入口查找逻辑电路如下。

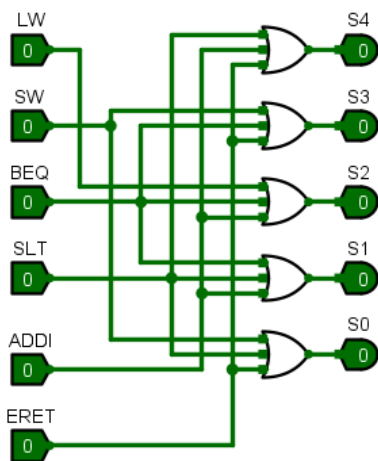


图 1.8 中断微程序入口查找逻辑电路

(3) 实现中断条件判别测试逻辑

我们填写了表格如图 1.4 所示，我们基于生成的逻辑表达式生成的逻辑电路图如图 1.9 所示。

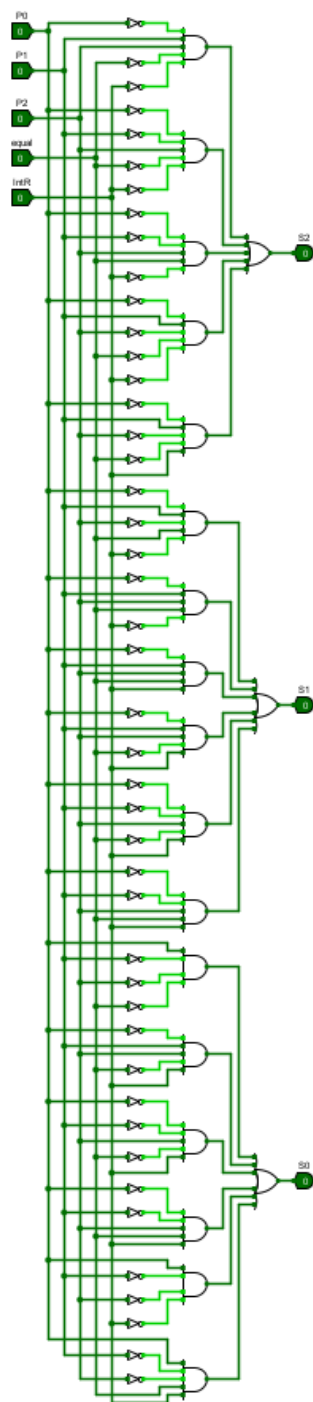
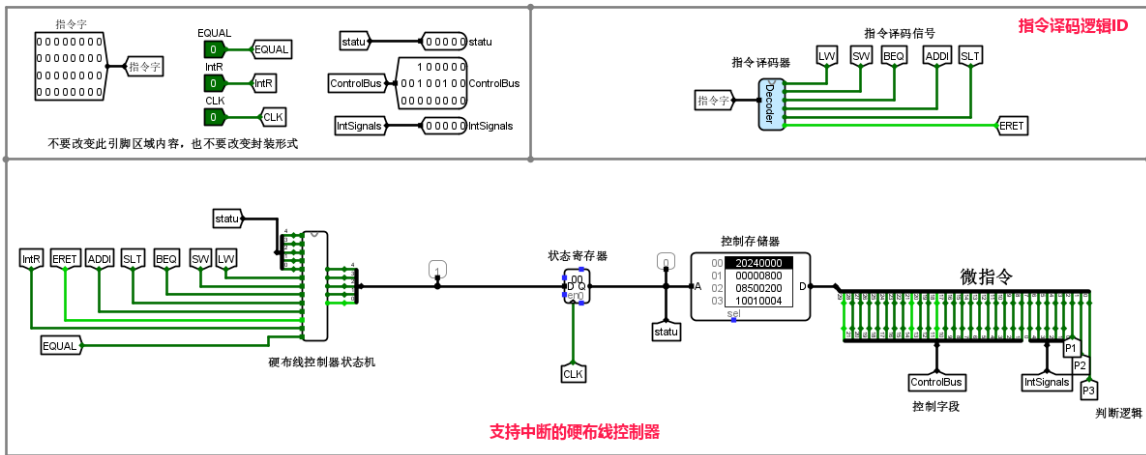
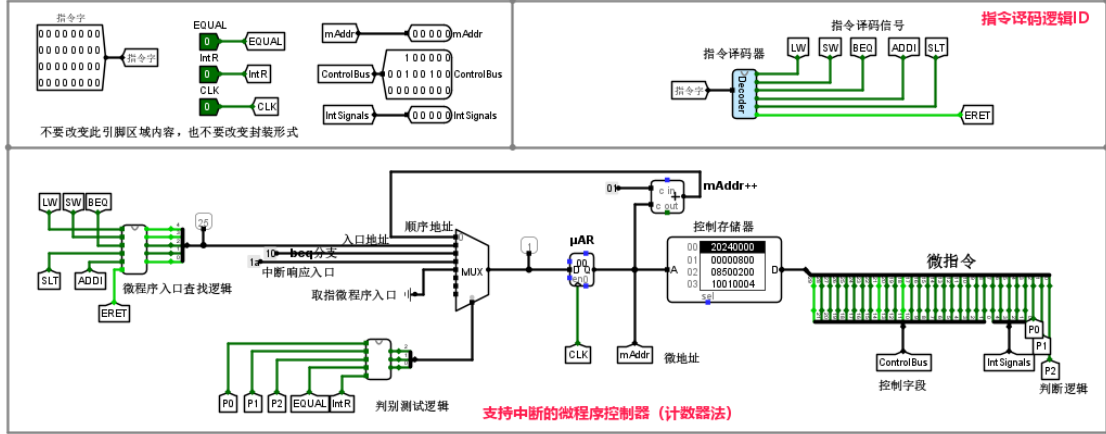


图 1.9 中断条件判别测试逻辑电路

(4) 用计数器法和硬布线法实现控制器

控制器的实现主要有两种方式，分别为计数器法+微程序和硬布线控制的方法。在

1.3 节我们已经讨论过这两种方法的逻辑转移关系和大致的结构，因此我们下面给出这两种方法连接的电路结果。



(5) 硬布线控制器状态机的逻辑生成

我们根据图 1.6 现代时序硬布线状态机状态关系，生成状态机的逻辑表达式，作为状态转移的条件。我们得到的部分逻辑表达式为：

$$\begin{aligned}
 N2 = & \sim S4 \& \sim S3 \& \sim S2 \& S1 \& S0 \& LW + \sim S4 \& \sim S3 \& \sim S2 \& S1 \& S0 \& BEQ + \\
 & \sim S4 \& \sim S3 \& \sim S2 \& S1 \& S0 \& ADDI + \sim S4 \& \sim S3 \& S2 \& \sim S1 \& \sim S0 + \\
 & \sim S4 \& \sim S3 \& S2 \& \sim S1 \& S0 + \sim S4 \& \sim S3 \& S2 \& S1 \& \sim S0 + \sim S4 \& S3 \& \sim S2 \& S1 \& S0 + \\
 & \sim S4 \& S3 \& S2 \& \sim S1 \& \sim S0 + \sim S4 \& S3 \& S2 \& S1 \& \sim S0 + S4 \& \sim S3 \& \sim S2 \& S1 \& S0 + \\
 & S4 \& \sim S3 \& S2 \& \sim S1 \& \sim S0 + S4 \& \sim S3 \& S2 \& S1 \& \sim S0
 \end{aligned}$$

由于生成的逻辑表达式及逻辑电路过于庞大，我们在下面给出该逻辑表达式组生成的硬布线控制器状态转移逻辑的部分电路。

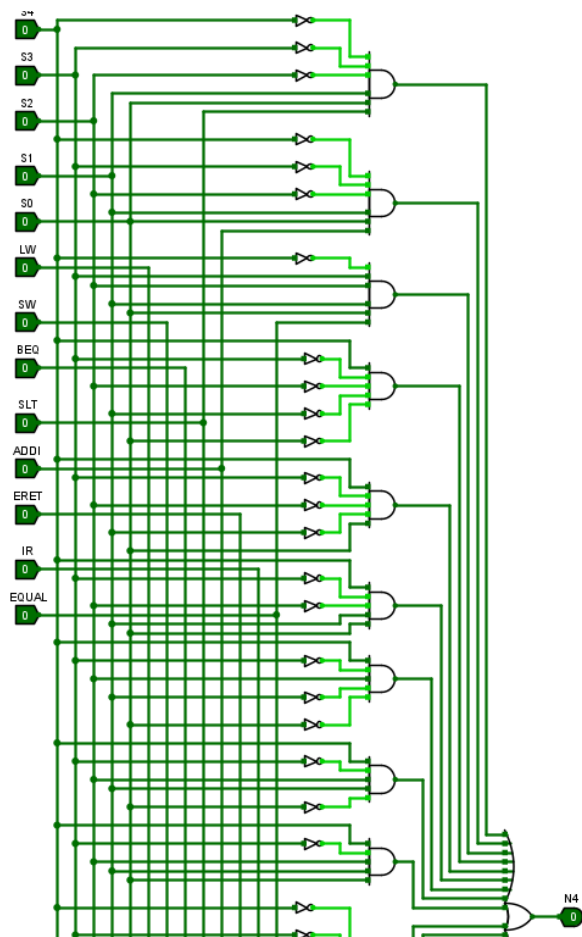


图 1.12 硬布线控制器状态转移逻辑电路（部分）

（6）支持中断的微程序单总线 CPU 设计

我们需要对原先的数据通路进行升级，在单总线数据通路中加入中断逻辑模块如图 1.13 所示。对中断模块进行连接后，我们将图 1.5 所设计的微指令填充入控制存储器中，并对各个器件进行连结。



图 1.13 中断逻辑模块

通过上述分析，我们得到了能够支持中断的单总线 CPU 的设计。同时根据我们设计的所有指令及其数据通路，将这些部件连接为即可得到单周期 CPU 的结构图。

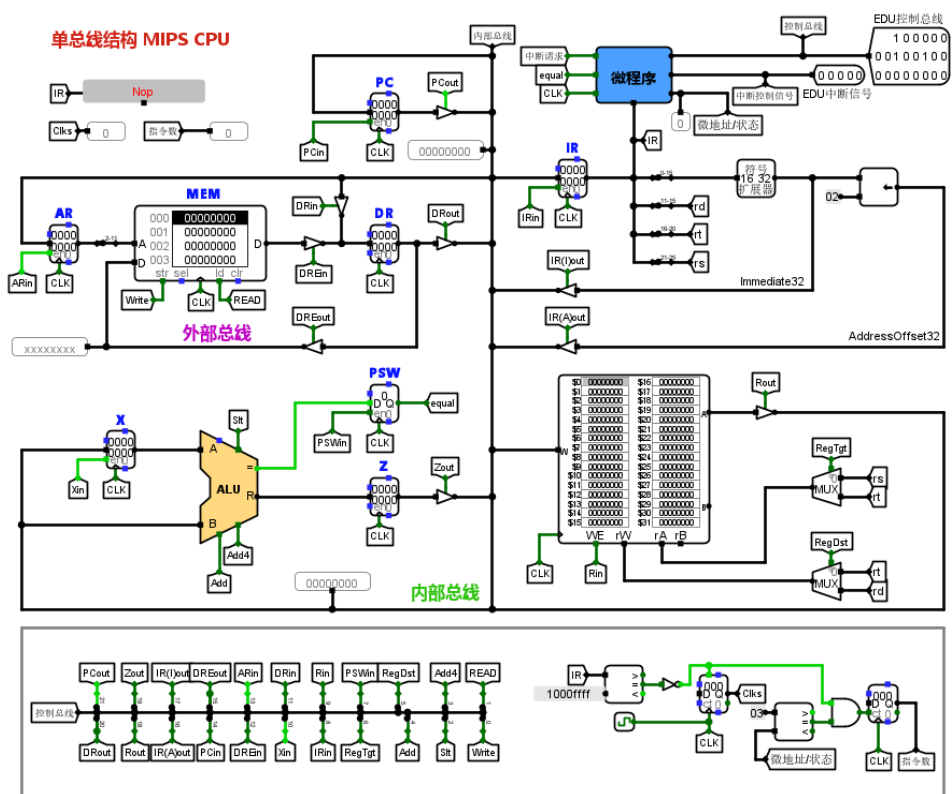


图 1.14 支持中断的单总线 CPU 设计结构图

1.4 故障与调试

1.4.1 取指令存储器地址错误

故障现象：程序每执行完一条指令后跳过 3 个指令直接执行第 4 条。

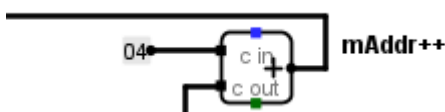


图 1.15 计算下址计数器错误

原因分析：PC+4 是对存储器加上 4 字节的偏移后进行取指。但在本题中存储器使用的是 32 比特的数据位宽，只需要对 PC+1 即可对应下一个四字节指令，因此不必对 PC 的值实际增加 4 以实现下一条指令的获取。我们实际上只需要对 mAddr 增加 1 即可。

解决方案：在控制器的计数器侧只需要对 PC 增加 1 即可，随后将计数器增加结果连至多路选择器下址字段（即 0 位选通路）。

1.5 测试与分析

我们将 sort-5-int.hex 中存储的 16 进制数据转存至我们设计的 CPU 的主存储器 MEM 中，并将其进行运行。我们在对源程序直接运行的过程中可以发现，如果未开中断，那么程序不会产生任何结果。如果我们开启 1 号中断，如图 1.16 所示，可以见到程序正常运行。

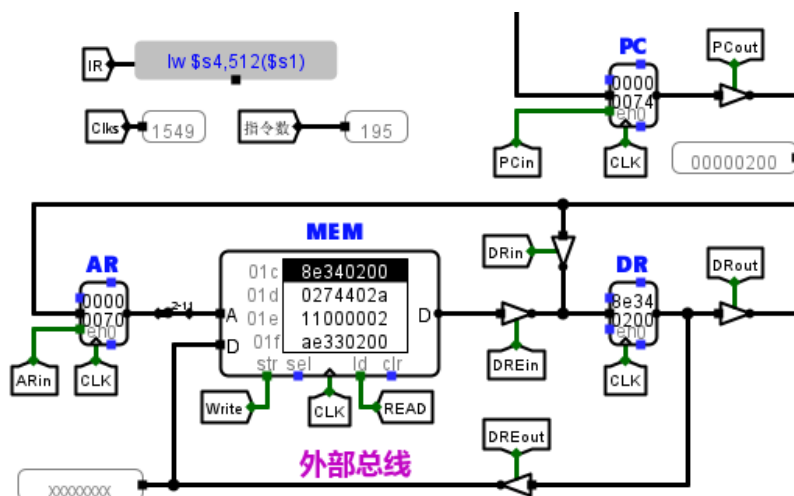


图 1.16 开中断时程序进行中断响应

如果我们没有按键行为，程序应该运行至 0x7c8 节拍停下，指令计数为 252，注意最后一条指令是一条 beq 分支指令，会跳回当前指令继续执行成为死循环。在让程序终止后，我们查看主存储器以地址 0x80 起始的数据段。如图 1.17 所示，程序的运行得到了一串已经排好顺序的数据，因此可见我们的单总线 CPU 在中断的条件下正常运行。

```

070 00000000000000000000000000000000 00000000000000000000000000000000
080 00000006000000050000000400000002 000000030000000100000000 ffffffff
090 00000001000000010000000100000001 00000001000000010000000100000001
0a0 00000000000000000000000000000000 00000000000000000000000000000000

```

图 1.17 程序 sort-5-int.hex 成功将各个数字进行排序

1.6 实验总结

本次实验主要完成了如下几点工作：

- 1) 实现了对 MIPS 定长指令的解析；
- 2) 实现了对数据寄存器 DR、指令寄存器 IR 等暂寄存器的读写和连接；
- 3) 实现了对微程序的编码和微程序控制器的实现；
- 4) 实现了对硬布线状态转移逻辑电路的生成；
- 5) 实现了可支持中断的 MIPS 单总线 CPU 总体结构设计；
- 6) 实现了对中断操作的结构支撑；
- 7) 最终实现了对目标中断程序的运行。

1.7 实验心得

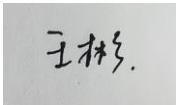
- 1) 通过本次组成原理实验, 我利用 Logisim 对于 CPU 及相关的 Cache、存储器等进行了设计, 使我对计算机的运算器、控制器的底层架构有了更深刻的了解。我逐渐熟悉了 MIPS 的指令架构, 利用定长指令的特性对于指令进行识别, 同时转换为信号。通过实验, 我对于课堂上的理论知识有了更加深刻的了解, 对课上讲过的知识和架构有了基于实践的生动认识, 这要求我们需要对计算机组成原理有更加立体的知觉。
- 2) 在完成实验任务的过程中遇到了一些困难, 如对于 Cache 的命中、对 CPU 设计一开始的茫然无措。这些在调试与试错的过程中, 逐步纠正自己的认知, 逐渐将零散的认识汇聚为一个具体的实在, 在这个过程中我对计算机的底层架构相关知识有了更加良好的把握。
- 3) 建议实验可以提供一份完整的需求文档, 让同学们除了在头歌平台上可以知道每一个小关卡需要做的任务, 还能够对实验的总体需求有宏观而立体的认识。

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：_____