

# 目 录

## 第一篇 I/O 控制的剖析

第一章 微处理器的 I/O .....	2
1.1 基本的 I/O 接口 .....	3
1.2 IBM PC 的 I/O 结构 .....	4
1.3 IBM PC 的扩充槽(Slot)结构 .....	6
1.4 思考题 .....	8
第二章 I/O 适配器 .....	9
2.1 微计算机 I/O 扩充槽 .....	9
2.2 如何使用扩充槽 .....	9
2.3 I/O 适配器 .....	10
2.4 适配器的测试 .....	14
2.5 思考题 .....	19
第三章 可编程外设接口 8255A .....	20
3.1 8255A 结构 .....	20
3.2 8255A 内部结构与功能简介 .....	22
3.3 模式的编程 .....	23
3.4 8255A 的操作模式 .....	24
3.5 C 口单一位的设置/复位 .....	26
3.6 操作模式的研究 .....	27
3.7 混合模式 .....	41
3.8 特殊模式的综合考虑 .....	42
3.9 C 口的应用考虑 .....	42
3.10 各种工作模式的定义 .....	43
3.11 8255A 的应用实例 .....	43
3.12 思考题 .....	44
第四章 8255A 的基本应用示例 .....	46
4.1 8255A 模式 0 的应用, PORTA 及 PORTB 均为输入型 .....	46
4.2 8255A 模式 0 的应用, PORTA 为输入型, PORTB 为输出型 .....	48
4.3 8255A 模式 0 的应用, PORTA 为输出型, PORTB 为输入型 .....	50
4.4 8255A 模式 0 的应用, PORTA 及 PORTB 均为输出型 .....	51
4.5 8255A 模式 1 的应用, 模拟交互式数据输入 .....	53
4.6 8255A 模式 1 的应用, 模拟交互式数据输出 .....	55
4.7 8255A 模式 2 的应用, 双向式数据传输 .....	58
4.8 思考题 .....	61

第五章 系统中断 .....	62
5.1 IBM PC 的中断 .....	65
5.2 INTR 中断 .....	66
5.3 8259A 中断控制器 .....	67
5.4 如何读取 8259A 的状态 IRR, ISR 及 IMR .....	80
5.5 串连模式 .....	80
5.6 82380 简介 .....	80
5.7 思考题 .....	82
第六章 IBM PC 及 8259A 的基本应用示例 .....	83
6.1 PC 系统中断模拟 .....	83
6.2 8255A 中断交互式数据输入模拟, 8255A 模式 1 .....	87
6.3 8255A 中断交互式数据输出模拟, 8255A 模式 1 .....	90
6.4 8255A 中断交互式数据输出输入模拟, 8255A 模式 2 .....	93
6.5 思考题 .....	98
第七章 可编程计时器 8253 / 8254 .....	99
7.1 功能简介 .....	99
7.2 8254 的结构及引脚说明 .....	100
7.3 如何编程 8254 .....	103
7.4 操作模式 .....	104
7.5 操作编程 .....	113
7.6 思考题 .....	115
第八章 8253 / 8254 的基本应用示例 .....	116
8.1 8253 / 8254 工作模式 0 的应用 .....	116
8.2 8253 / 8254 工作模式 0, 观察计时器值读回情形 .....	118
8.3 8253 / 8254 工作模式 1 .....	121
8.4 8253 / 8254 工作模式 2 .....	123
8.5 8253 / 8254 工作模式 3 .....	125
8.6 8253 / 8254 工作模式 4 .....	126
8.7 8253 / 8254 工作模式 5 .....	128

## 第二篇 I/O 控制的专题研究

第九章 跑马灯专题 .....	132
9.1 工作目标 .....	132
9.2 硬件设计 .....	133
9.3 软件设计 .....	134
9.4 结论 .....	145
9.5 思考题 .....	145
第十章 红绿灯专题 .....	146

10.1	工作目标	146
10.2	硬件设计	147
10.3	软件设计	147
10.4	结论	156
10.5	思考题	156
第十一章	家电控制专题	157
11.1	工作目标	157
11.2	硬件设计	157
11.3	软件设计	159
11.4	结论	163
11.5	思考题	163
第十二章	LED 七段显示器专题	164
12.1	工作目标	164
12.2	硬件设计	164
12.3	硬件说明	165
12.4	软件设计	166
12.5	结论	173
12.6	思考题	174
第十三章	方波发生器专题	180
13.1	工作目标	180
13.2	硬件设计	181
13.3	软件设计	181
13.4	结论	188
13.5	思考题	188
第十四章	自动售货机专题	189
14.1	工作目标	189
14.2	硬件设计	190
14.3	软件设计	190
14.4	结论	204
14.5	思考题	204
第十五章	防盗器专题	205
15.1	工作目标	205
15.2	硬件设计	206
15.3	软件设计	206
15.4	结论	212
15.5	思考题	213
第十六章	数字 IC 测试器专题	214
16.1	工作目标	214
16.2	硬件设计	214

16.3	软件设计 .....	217
16.4	结论 .....	221
16.5	思考题 .....	222
<b>第十七章</b>	<b>电梯模拟专题 .....</b>	<b>223</b>
17.1	工作目标 .....	223
17.2	硬件设计 .....	223
17.3	软件设计 .....	224
17.4	结论 .....	235
17.5	思考题 .....	235
<b>第十八章</b>	<b>加油机专题 .....</b>	<b>236</b>
18.1	工作目标 .....	236
18.2	硬件设计 .....	236
18.3	软件设计 .....	237
18.4	结论 .....	244
18.5	思考题 .....	244
<b>第十九章</b>	<b>键盘模拟专题 .....</b>	<b>245</b>
19.1	工作目标 .....	245
19.2	硬件设计 .....	245
19.3	软件设计 .....	246
19.4	结论 .....	251
19.5	思考题 .....	252
<b>第二十章</b>	<b>声音控制 (一) 专题 .....</b>	<b>254</b>
20.1	工作目标 .....	254
20.2	硬件设计 .....	254
20.3	软件设计 .....	255
20.4	结论 .....	259
20.5	思考题 .....	259
<b>第二十一章</b>	<b>声音控制 (二) 专题 .....</b>	<b>261</b>
21.1	工作目标 .....	261
21.2	硬件设计 .....	262
21.3	软件设计 .....	262
21.4	结论 .....	265
21.5	思考题 .....	265

### 第三篇 I/O 控制的思考专题研究

<b>第二十二章</b>	<b>ADC 模拟 / 数字转换器 .....</b>	<b>268</b>
22.1	ADC 结构 .....	268
22.2	ADC-0804 简介 .....	268

22.3	ADC 与微型处理机的接口 .....	272
22.4	ADC 与微型处理机的应用实例 .....	273
22.5	思考题 .....	274
<b>第二十三章 DAC 数字 / 模拟转换器 .....</b>		<b>275</b>
23.1	DAC 的基本结构 .....	275
23.2	DAC-08 简介 .....	278
23.3	DAC 与微型处理机的接口 .....	279
23.4	思考题 .....	279
<b>第二十四章 数字激光音响与数字电视 .....</b>		<b>280</b>
24.1	数字音响 .....	280
24.2	数字电视 .....	282
24.3	数字音响和数字电视中有一部 IBM PC 吗 .....	283
<b>第二十五章 电话保密与数字通讯 .....</b>		<b>284</b>
25.1	模拟通讯 .....	284
25.2	数字通讯 .....	285
25.3	信号的数字化与传送 .....	285
25.4	电话保密 .....	288
25.5	思考题 .....	289
<b>第二十六章 数据通讯的接口技术 .....</b>		<b>290</b>
26.1	计算机通讯方式 .....	290
26.2	调制解调器(Modem) .....	292
26.3	创建通讯规则 .....	293
<b>第二十七章 步进电机的应用 .....</b>		<b>294</b>
27.1	步进电机简介 .....	294
27.2	步进电机与微处理器的接口 .....	296
27.3	步进电机的应用实例 .....	298
<b>第二十八章 EPROM 编程器 .....</b>		<b>300</b>
28.1	EPROM 简介 .....	300
28.2	操作模式 .....	302
28.3	编程框图 .....	304
28.4	EPROM 编程器的设计 .....	305
28.5	EPROM 编程器的硬件设计 .....	305
28.6	软件编程 .....	305
<b>第二十九章 赌博性电动玩具与警察专题 .....</b>		<b>307</b>
29.1	工作目标 .....	307
29.2	硬件设计 .....	307
29.3	软件设计 .....	307
<b>第三十章 微型计算机的其它应用 .....</b>		<b>309</b>
30.1	为什么要用微计算机 .....	309

30.2 单片机与 IBM PC .....	310
30.3 机器人(I)(利用 IBM PC) .....	310
30.4 机器人(II) .....	310
30.5 火车站台自动控制 .....	312
30.6 霹雳车与计算机 .....	313
30.7 IC 卡 .....	313
附录 A ASCII 字符和扩展字符表 .....	316
附录 B Turbo C 用于 I/O 的函数介绍 .....	324
B.1 inportb() .....	324
B.2 inport() .....	324
B.3 outportb() .....	324
B.4 outport() .....	325
B.5 简单 I/O 控制声音的应用 .....	325
附录 C 本书所使用的 Turbo C 的函数表 .....	328
C.1 clrscr() .....	328
C.2 cprintf() .....	328
C.3 delay() .....	329
C.4 getch() .....	330
C.5 getvect() .....	331
C.6 gotoxy() .....	332
C.7 kbhit() .....	333
C.8 localtime() .....	334
C.9 nosound() .....	335
C.10 pow() .....	336
C.11 printf() .....	337
C.12 random() .....	341
C.13 scanf() .....	342
C.14 setvect() .....	343
C.15 sleep() .....	344
C.16 sound() .....	345
C.17 textbackground() .....	345
C.18 textcolor() .....	346
C.19 time() .....	347
C.20 window() .....	347
附录 D IC 的基本常识 .....	349

# 第一篇

## I/O 控制的剖析

- 第一章 微处理器的I/O
- 第二章 I/O适配器
- 第三章 可编程外设接口8255A
- 第四章 8255A的基本应用示例
- 第五章 系统中断
- 第六章 IBM PC及8259A的基本应用示例
- 第七章 可编程计时器8253/8254
- 第八章 8253/8254的基本应用示例

# 第一章 微处理器的 I/O

## 本章学习目的

1. 读者可通过本章获得 IBM PC 的接口知识。
2. 读者通过 I/O 的读写时序图可以了解微处理器的 I/O 工作方式。

## 本章内容

- 1.1 基本的 I/O 接口
- 1.2 IBM PC 的 I/O 结构
- 1.3 IBM PC 的扩充槽(slot)结构
- 1.4 思考题

何谓接口？见图 1.1。微处理器的各种接口。

微处理器是计算机系统的心脏，它必须通过键盘、驱动器、屏幕、打印机等与外界组成一个完整的系统。外围设备与微处理器之间的数据传输必须依赖接口的协助，微处理器就是通过这种接口准确高速地控制着每一个外设。

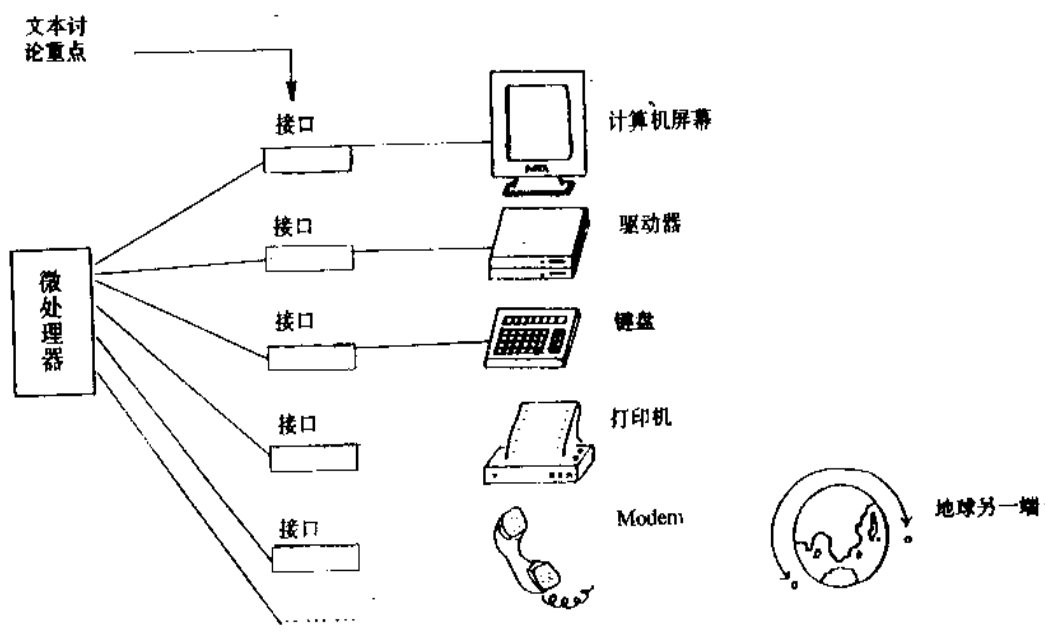


图 1.1 微处理器的接口

微处理器的 I/O 可分为以下类型：

- 输入/输出端口映射式 I/O (Input/Output Port Mapped I/O)



这种 I/O 的最大特色是它自己拥有独立的 I/O 指令，其存储器寻址和 I/O 寻址是独立的，只能以 IN, OUT, INS 与 OUTS 等指令访问，这种 I/O 为本书讨论的重点。

注：对 Turbo C 而言，IN, OUT, INS 与 OUTS 等指令相当于下列函数 *inportb()*, *outportb()*, *inport()* 和 *outport()*。有关这些函数的使用方式，可参考附录 B，本文叙述仍将以 IN, OUT 代表输入及输出，不过读者以 Turbo C 设计程序时，可将它想象成 *inport()* 和 *outport()*。

#### • 存储器映射式 I/O (Memory Mapped I/O)

这种存储器映象对象位于主存储器的地址空间内。例如屏幕上每一点都可称为是 I/O，因为这些 I/O 包含于主存内，所以称为存储器映射式 I/O，CPU 借助这些 I/O 将信息显示于屏幕上。

## 1.1 基本的 I/O 接口

典型的 IBM PC 系统有下列常用的外设元件：

- 8259A 可编程中断控制器。
- 8237 DMA 控制器，用以直接、快速的处理存储器的访问。
- 8272 软盘驱动器控制器 (Floppy disk controller)
- 82062, 82064 硬盘驱动器控制器 (Hard disk controller)
- 8274 复式串行控制器 (Multi-protocol serial controller)

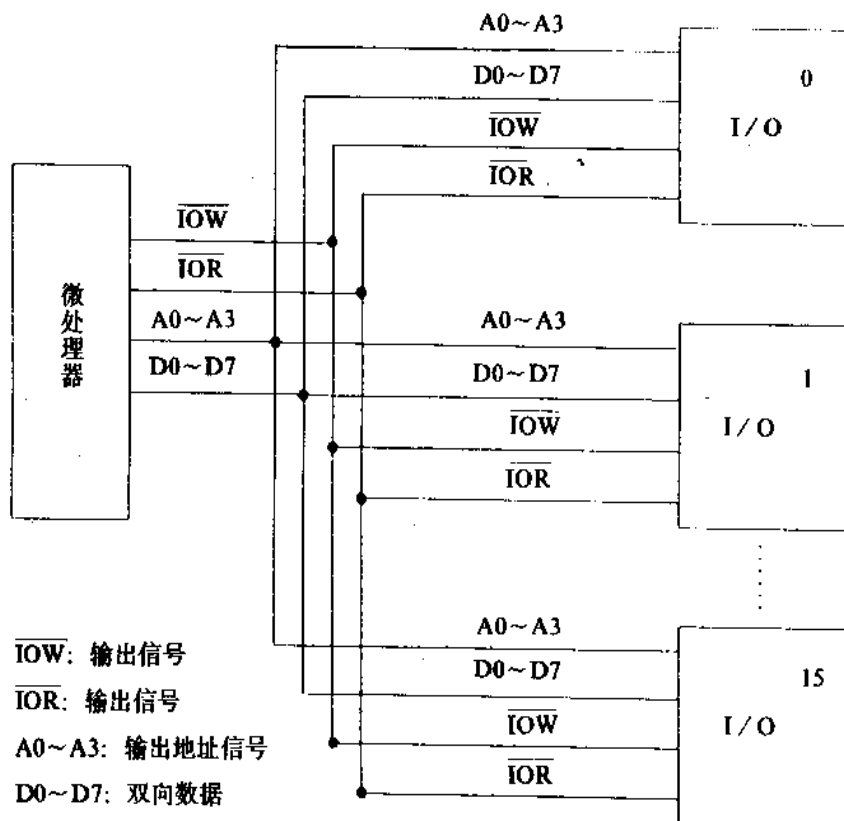


图 1.2 微处理器与 I/O 口结构方块图

- 82258 高等 DMA 控制器。
- 8255A 可编程外设接口 (Programmable peripheral interface)
- 82C53(82C54)可编程计时 / 计数器 (Programmable interval time)

本书的讨论重点在于 8259A, 8255A, 82C53(82C54)的编程, 以实例分析、探讨这些接口的 I/O 及其应用。

## 1.2 IBM PC 的 I/O 结构

IBM PC 系统中有许多 I/O 系统专用的 IC, 如系统计时的 82C54, 中断处理的 8259A 及直接存储访问的 8237 等。这些 I/O 都有特定的地址, 微处理器通过这些地址与不同的 I/O 外设通讯。读者在设计 I/O 实验时, 必须查阅所用系统的使用手册, 避免不同的 I/O 使用相同的地址, 造成系统混乱。

在图 1.2 中共有 16 个 I/O 口, 由 A0-A3 确定不同的 16 个 I/O 地址。D0-D7 为系统送出的双向数据总线 (bus)。 $\overline{IOW}$  为 I/O Write 的数据控制总线,  $\overline{IOR}$  为 I/O Read 的数据控制总线。

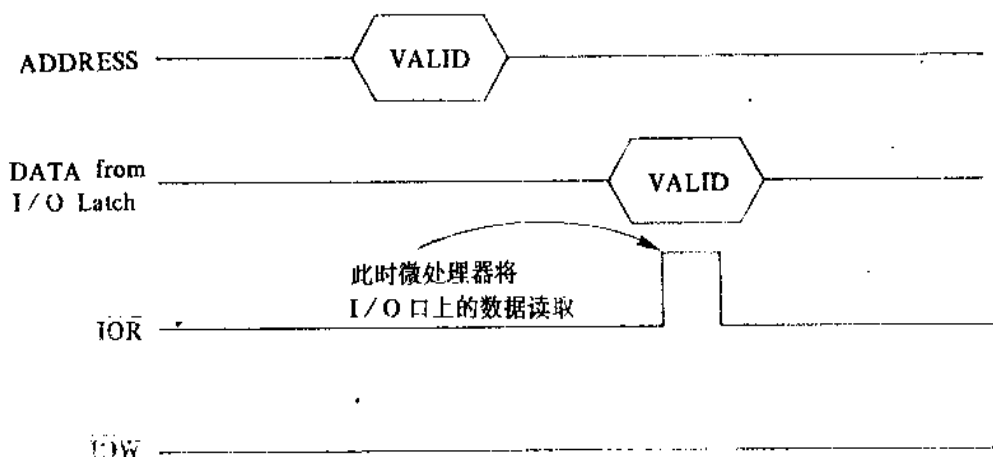


图 1.3 I/O 读取时序图

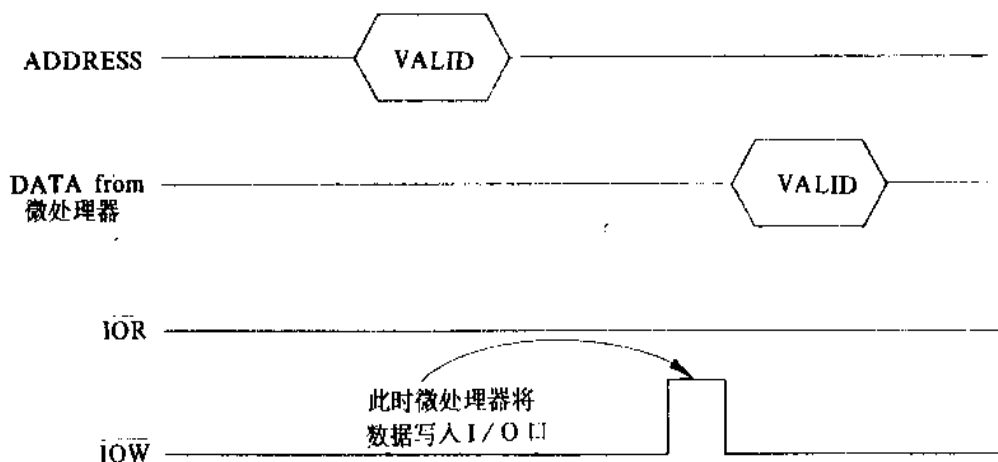
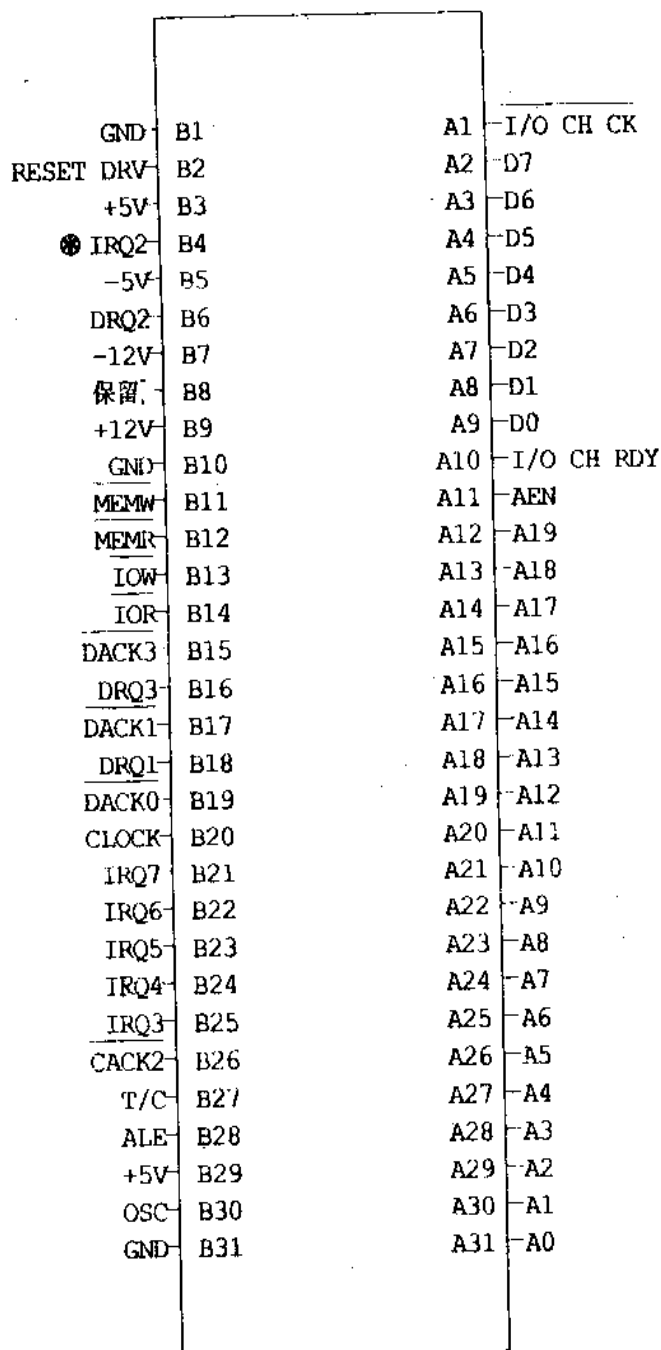


图 1.4 I/O 写入时序图

图 1.3 为系统读取 I/O 外设的时序图。当微处理器发出  $IN \times \times$  ( $\times \times$  为地址) 时, 所有的 I/O 均将自己的地址与  $\times \times$  比较。如果确定自己被系统选到时, 即将数据从 I/O 放出, 然后被系统送出的  $\overline{IOR}$  将 I/O 的数据送至系统数据总线上。

图 1.4 为系统将 DATA 写入 I/O 外设的时序图。当微处理器发出  $OUT \times \times$  ( $\times \times$  为地址) 时, 所有的 I/O 即将自己的地址与  $\times \times$  比较, 如果确定自己被选到时, 即可利用  $\overline{IOW}$  将系统数据总线上的 DATA 写入 I/O 的数据锁存器 (DATA Latch) 中。



\* 在 80386 中此信号为 IRQ9

图 1.5 扩充槽引脚 (62 脚)

### 1.3 IBM PC 的扩充槽 (Slot) 结构

IBM PC 提供了若干扩充槽供外设接口使用。譬如下列外设使用了这些扩充槽。

- 打印机
- EGA、VGA 显示卡
- 磁盘驱动器

扩充槽上有许多信号引脚, 经过这些特定的信号排列, 外设接口可以准确地和微处理器联系。在每个扩充槽中配有 62 个引脚, 各代表不同的信号, 每个槽上的外设可以配置任何标准设备, 如打印机、EGA、VGA 显示卡及驱动器。本书的讨论重点即在这个扩充槽上, 我们将设计一个硬件的接口, 插入这个扩充槽, 和微处理器沟通信息。

信号名称及扩充槽引脚请参阅图 1.5。现将各信号的代表意义说明如下:

O

表示由微处理器输出。

I

表示由外设输入至微处理器。

A0-A19(O)

地址位 0-19。这 20 个地址位用以指定存储器及接口位置。

D0-D7(I/O)

数据位 0-7。

OSC(O)

振荡器的频率, 由振荡器输出到此, 再送到外设上。

CLK(O) (CLOCK)

PC 系统振荡频率输出。

RESET DRV(O)

此复位信号用来复位(RESET)外设, 这一逻辑信号电平常为低电平。

ALE(O) (Address Latch Enable)

地址锁定允许。在一连串的系统总线上, 载有各种不同的信息, 当 ALE 产生时, 会将此时的总线状态锁定, 被锁定的信息即被视为地址区。

$\overline{\text{MEMW}}$  (O) (Memory Write)

存储器写入信号, 低电平动作的三态(Three State)信号 (高、低、高阻抗), 将要写数据写入存储器。

$\overline{\text{MEMR}}$  (O) (Memory Read)

存储器读取信号, 低电平动作的三态(Three State)信号 (高、低、高阻抗), 用来读取存储器的数据。

DRQ1~DRQ3 (I) (DMA Request)

当外界需要直接存储器访问 (DMA) 服务时, 可通过这些信号告诉 DMA。

注: DRQ1 具有较高的优先权。

- 在 DMA 应用中, 其实 DRQ0 具有较高的优先权, 但此信号已被指定用以刷新

(Refresh)动态存储器，所以不能被用户使用。

**DACK0~DACK3 (I) (DMA Acknowledge)响应信号**

当系统知道接口通过 DRQ1~DRQ3 提出服务要求之后，会产生此信号通知接口“我知道了”。

**I/O CH CK (I) (I/O Channel Check)**

I/O 通道 (I/O Channel) 的奇偶校验同步位，低电位时有效 (Active Low)，此时表示 I/O 通道同步错误。

**I/O CH RDY (I) (I/O Channel Ready)**

此信号表示 I/O 通道准备完成的意思，当外界接口配置未完成准备工作时，会将此信号设为低电平，以告诉微处理器暂缓运行，以适合低速外部设备与高速微处理器之间的信息输送。

**IRQ2~IRQ7 (I) (Interrupt Request)**

第二级至第七级的中断请求。第零级和第一级已分别被 82C53 系统计时器及键盘扫描中断使用，因此不能被其它外设使用。

**$\overline{\text{IOR}}(\text{O})$  (I/O Read)**

I/O 读取信号。为低电位时工作 (Active Low)。

**$\overline{\text{IOW}}(\text{O})$  (I/O Write)**

I/O 写入信号。为低电位时工作。

**AEN (O) (Address Enable)**

地址允许，将被锁定的地址输回系统总线。

**T/C(O) (Terminal Count)**

结束计数，表示 DMA 已经完成所要求的信息传输。

注意所有符号上端有横线（例如英文字  $\overline{\text{IOR}}$  上方的横线）表示低电位有效，其余为高电位有效。

表 1.1 介绍 I/O 扩充卡信号驱动的能力。请读者注意，以免发生计算机系统不能驱动外设，或因而产生死机的问题。

表 1.1 I/O 扩充槽推动的能力

总线信号名称	IOL	TOH
D7~D0	23.7	-15.0
A19~A16	7.2	-2.5
A15~A14	21.3	-2.5
A13	23.2	-2.6
A12~A0	23.5	-2.6
$\overline{\text{IOR}}$ , $\overline{\text{IOW}}$ , $\overline{\text{MEMR}}$ , $\overline{\text{MEMW}}$	24.0	-5.0
CLK	23.2	-15.0
AEN	24.0	-15.0

(续表)

总线信号名称	IOL	TOH
$\overline{\text{DACK0}}$	24.0	-15.0
$\overline{\text{DACK1}}$	3.2	-0.2
$\overline{\text{DACK2}}$ , $\overline{\text{DACK3}}$	2.8	-0.2
ALE	14.6	-0.9
RESET DRV	8.0	-0.4
T/C	8.2	-0.4
OSC	5.2	-1.0

注 1: 上表以毫安(mA)为单位

注 2: 不同主机有不同的驱动能力

表 1.2 介绍 I/O 扩充槽输入信号的负载。

表 1.2 I/O 扩充槽输入信号负载的能力

I/O 总线信号名称	LOAD	
	IIL	IIH
D7~D0	-0.42	0.05
$\overline{\text{I/O CH CK}}$	-0.42	0.03
$\overline{\text{I/O CK RDY}}$	-0.44	0.03
$\overline{\text{IRQ7}} \sim \overline{\text{IRQ2}}$	-0.01	0.04
$\overline{\text{DRQ3}} \sim \overline{\text{DRQ1}}$	-0.01	0.01

注: 上表以毫安(mA)为单位

## 1.4 思考题

1. 何谓接口 I/O? 并将所有您知道的 I/O 接口列出来。
2. 拆开您的计算机, 分辨出所有的 IC 及扩充槽。将所有的 IC 列出来, 并说明它们的功能。
3. 比较 IBM PC XT, AT, 386, 486, 说明它们有何不同。
4. 比较计算机外设各个驱动器的不同点, 它们之间可否互换, 如将 A 驱动器和硬盘 C 互换, 它们的适配器是否相同?

## 第二章 I/O 适配器

### 本章学习目的

1. 读者通过本章可以获得如何设计适配器的相关知识。
2. 读者经过本章亦可温习电子线路的基本概念。
3. IC 测试、适配器的测试基本知识也可以借助本章获得。

### 本章内容

- 2.1 微计算机 I/O 扩充槽
- 2.2 如何使用扩充槽
- 2.3 I/O 适配器
- 2.4 适配器的测试
- 2.5 思考题

I/O 适配器是 PC 系统和外界交换信息的外设接口。任何外设控制都必须借助这个适配器。如果读者打开计算机，将会发现在计算机背后插着多块外设卡，这些卡有些连接计算机屏幕，有些连接驱动器、键盘或打印机。这些卡又称为适配器。

### 2.1 微计算机 I/O 扩充槽

在第一章中曾经详述 PC 系统的 I/O 扩充槽，这些扩充槽有的是 62 脚，有的是 36 脚，用以提供不同的 PC 系统接口使用。62 脚的扩充槽不论在 XT 中还是 AT 中大致相同，唯一的不同点是在 80386 的 62 脚扩充槽的第 4 脚信号为 IRQ9 与 XT 的 IRQ2 不同。如果读者详细研究 80386 系统之后会发现，80386 的 PC BIOS 已将此部分处理过了，所以对用户而言，可以将 IRQ9 当成 IRQ2 来处理。在本章、第六章及研究专题中有些示例是在 80386 系统下编程的，读者可以参考并加以应用。

### 2.2 如何使用扩充槽

市面上的适配器，形式如图 2.1。

在图 2.1 中，我们可以发现 I/O 标准适配器上的 62 个信号和 PC 系统扩充槽上的 62 个信号相连接。这 62 个信号的定义完全相同，不同的扩充槽结构和信号也完全相同，所以任何一个同脚数的扩充槽均可以接受任何驱动器或打印机，甚至于将来开发出来的新卡。

在本书中，所讨论的重点为 8255A，82C53/82C54 及 8259A。针对以上的要求我们将这 3 种集成电路设计在这个 I/O 标准适配器上，通过所讨论过的 62 个标准信号和 PC 系统互传信息。

这些 PC 上的标准扩充槽已经在全世界通用，任何新的适配器必须与之配套，否则不能和任何的 IBM PC 机兼容。

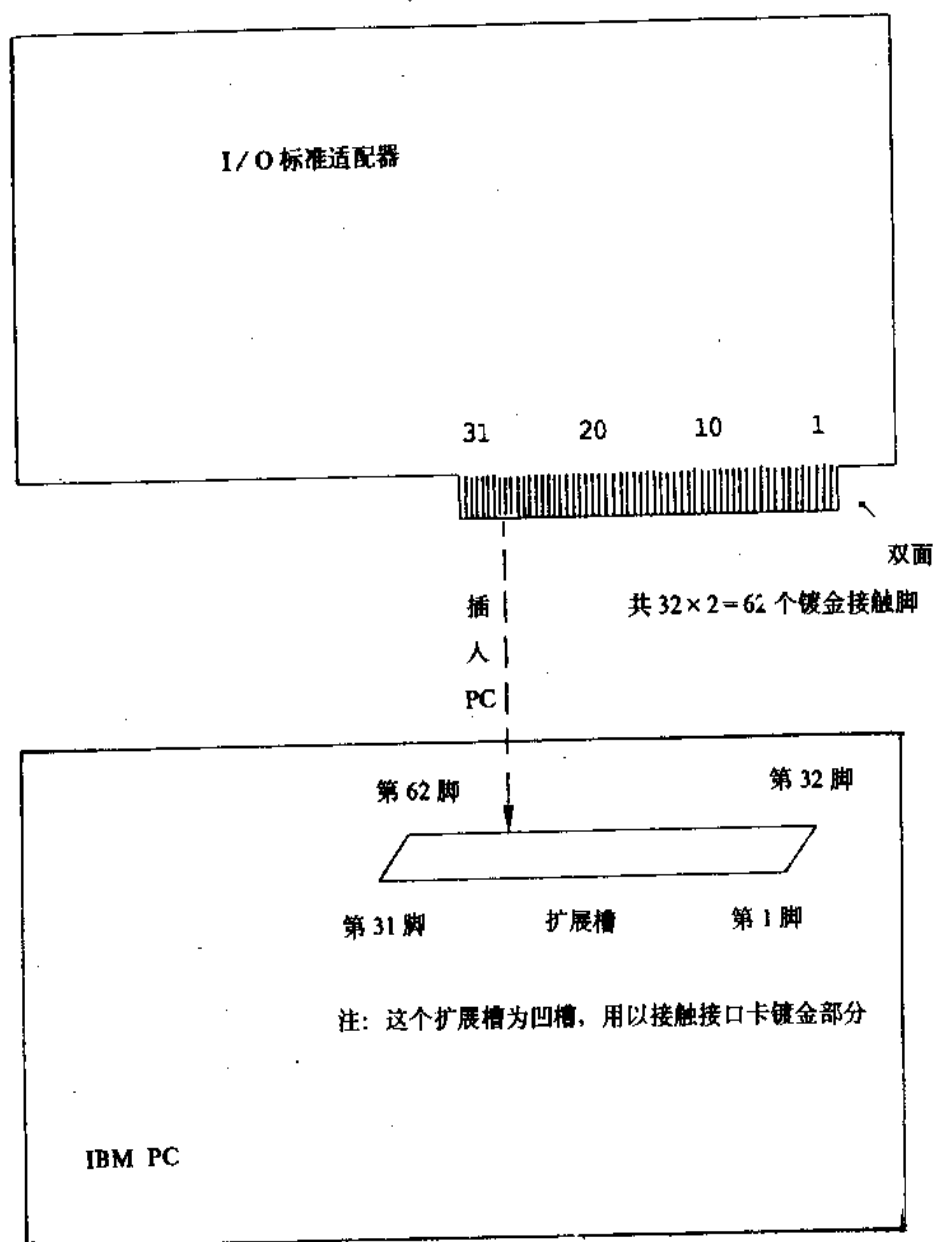


图 2.1 微计算机系统和适配器的连接结构

## 2.3 I/O 适配器

本书所谈论的重点在于 8255A, 82C53/82C54 及 8259A 三种集成电路。我们设计了一个适配器，利用这三种集成电路和外界传递信息。这三种集成电路为 I/O 接口所不可缺少的重要组件，随后的各章节中将详述它们的功能。详细研究之后，可了解 PC 系统所有接口的基本结构。以下简介这个适配器的结构。



### 2.3.1 I/O 适配器的解码电路

图 2.2 为此适配器的解码电路。A1~A9 及 AEN 可以允许(enable)8255A-1, 8255A-2, 8259A 及 82C53/82C54 等四片集成电路。

74LS138 为一个常用的解码集成电路, 它可以将 PC 系统送出来的地址解码, 用以允许其它元件。

AEN 在 74LS138 线路中最主要目的是将 74LS138 禁止 (disable), 为什么 AEN 要使 I/O 口地址解码电路失去效用呢? 因为 AEN 是主机板 8237 或同性质的集成电路送出的信号 (如 80386 系统中的 82380)。当此信号为高电平时, 表示目前正进行 DMA 的总线周期。为了避开 DMA 的总线周期, 以免数据互相冲突, 所以当 DMA 在运行时, 必须禁止 I/O 接口工作。RESET (复位) 由 IBM 的 PC 系统产生, 经过两个反向逻辑, 与 I/O 适配器的复位信号相连接, 为什么? 一般的反向逻辑可以当成简单的缓冲区, 这样可以增加 RESET 信号的驱动能力。

74LS245 为一个高速的双向传输元件。当 DIR 为低电平时, XD0~XD7 的数据会传到 D0~D7, 表示系统读取外设元件的数据。反之 D0~D7 的数据会传给 XD0~XD7, 表示系统将数据写入外设元件。

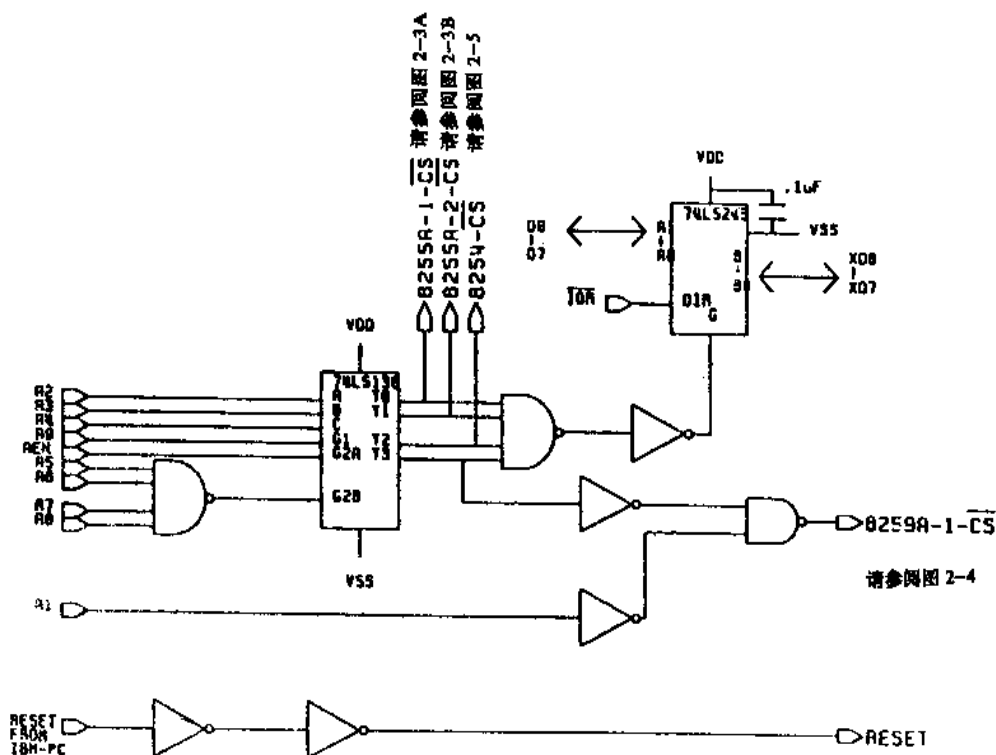
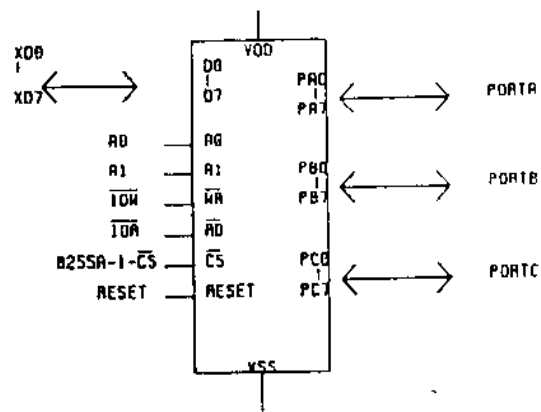


图 2.2 I/O 适配器的解码电路

### 2.3.2 8255A I/O 地址及引脚图

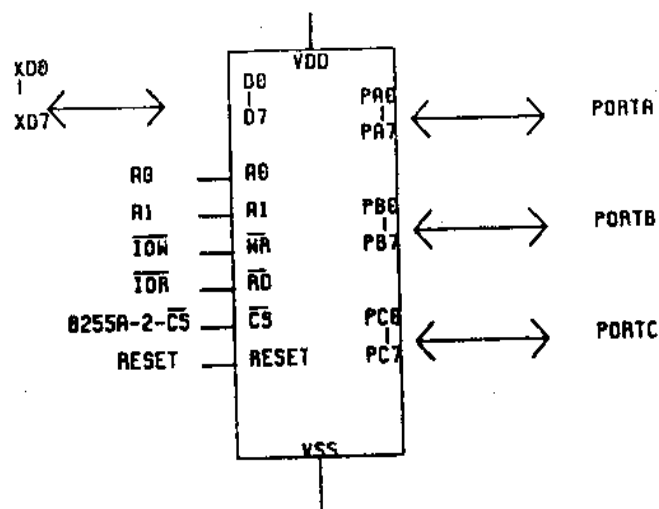
这个 I/O 适配器中包含两个 8255A, 它们的寻址地址如图 2.3。



ADDRESS	REGISTER
3E8H	PORTA
3E1H	PORTB
3E2H	PORTC
3E9H	CTR REG

8255A-1 I/O ADDRESS

图 2.3A 8255A I/O 地址及引脚图



ADDRESS	REGISTER
3E4H	PORTA
3E5H	PORTB
3E6H	PORTC
3E7H	CTR REG

8255A-2 I/O ADDRESS

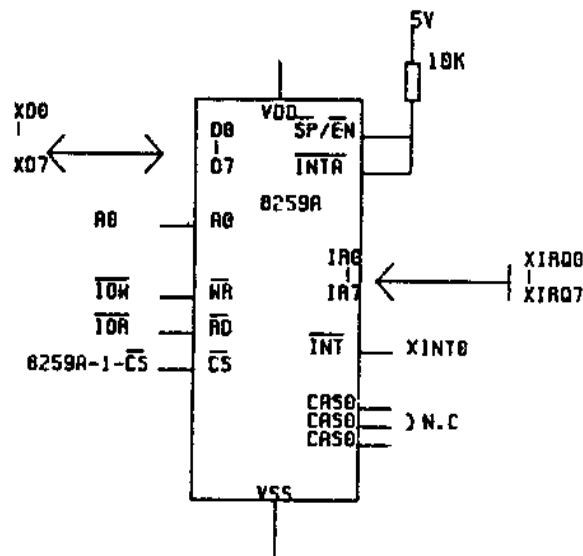
图 2.3B 8255A I/O 地址及引脚图

### 2.3.3 8259A I/O 地址及引脚图

这个 I/O 适配器提供了一个 8259A，且只在主(Master)模式下使用，所以  $\overline{SP/EN}$  都为高电平。

### 2.3.4 8253/8254 I/O 地址及引脚图

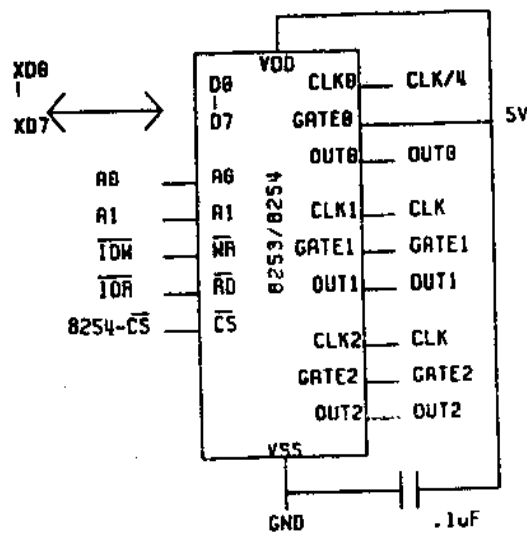
这个 I/O 适配器上提供了一个 8253/8254。其中 CLK0 由系统的时钟频率除以 4 而得到，CLK1 及 CLK2 接上系统的 CLK。



ADDRESS	REGISTER
3E0H	ICW1
3E0H	ICW2
3E0H	ICW3
3E0H	ICW4
3E0H	OCW1
3E0H	OCW2
3E0H	OCW3

8259A-1 I/O ADDRESS

图 2.4 8259A I/O 地址及引脚图



ADDRESS	REGISTER
3EBH	PORTA
3E9H	PORTB
3EAH	PORTC
3EBH	CTR REG

8253/8254 I/O ADDRESS

图 2.5 8253/8254 I/O 位置及引脚图

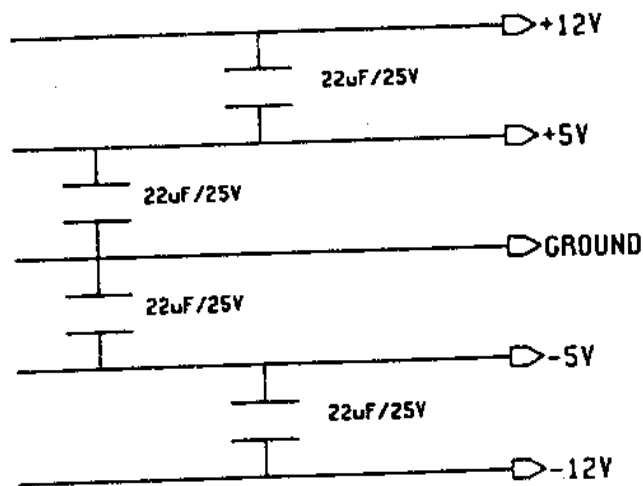


图 2.6 电源

## 2.4 适配器的测试

### 2.4.1 IC 的测试

在讨论适配器的测试前，我们须先了解集成电路在出厂前是如何测试的。

一般来说, 集成电路的测试可分为三个部分:

- DC 参数 (直流部分) 测试
- AC 参数 (交流部分) 测试
- 功能测试

DC参数: 一些与时间无关的参数

如输入电压的要求  $V_{il}, V_{ih}$

输出电压的规格  $V_{ol}, V_{oh}$

输入漏电流  $I_{il}$

输出漏电流  $I_{ol}$

电源供应电流  $I_{cc}$

AC参数: 一些与时间有关的参数

如输入波上升时间  $T_r$

输入波下降时间  $T_f$

一些模拟信号的增益测量, 或其它的复杂信号测量等也属于 AC 参数。

功能测试: 测IC的功能, 这一部分是属于最重要的部分, 也是最容易了解的部分。

一般集成电路在芯片做完之时即经过一次测试, 称为 (Pretest)。好的芯片 (Chip) 在经过封装之后又经过一次测试, 称为 (Final Test)。Final Test 的最主要的目的是测试 IC 在封装时是否有损坏。一个好的测试程序必需要能将真正有问题的集成电路挑出, 但是很少有任何测试程序能做完 100% 的测试。以一个微处理器的集成电路为例, 它含有数万个晶体管, 如何在 10 秒之内将这数万个晶体管完全测完实属不易, 而且常常有些晶体管会有动态的问题 (譬如说, 它会受另一个晶体管的影响), 这样更增加了集成电路测试的难度。几乎没有一家集成电路公司曾对外宣布他们的微处理器经过 100% 的测试。

## 2.4.2 适配器的测试

适配器上有一些集成电路, 这些集成电路基本上并不在适配器测试程序上详细测试。适配器的测试主要是测集成电路是否完好地焊在适配器上。相比之下, 这种测试就简单多了。这类的测试, 我们自己就可以书写程序测试。笔者写了一个程序用以简单地测试本书所使用的适配器, 原理如下:

8255A-1: 写入值“55H”于PORT 3E0中, 再读回。如果成功读回, 即表示这片 IC 是好的。参考思考题 1。

8255A-2: 写入值“55H”于PORT 3E3中, 再读回。如果成功读回, 即表示这片 IC 是好的。不然就是坏的, 或是适配器上并没有这一片 IC 存在。

8253 / 8254: 写入值“55H”于第0个计数器中, 在“free run”下读回其值两次, 如果不相同, 表示这片 IC 的第0个计数器有动作, 所以是好的。如果相同, 则再读一次, 如果与前两次不同, 则表示这片 IC 的第0计数器是好的。三次相同的概率为  $1/256^3 = 5.9 \times 10^{-8}$ , 程序误判的概率相当小, 所以不予考虑。参考思考题 2。

8259A: 依次写入ICWS  
ICW1: 13H

ICW2: 0

ICW3: 0

ICW4: 55H

再读回ICW4，如果为55H表示此部分是好的。

示例程序 iotest.c

I/O 适配器的测试，下图是示范测试的结果。

#### I/O Card Testing

8255A-1 O.K.

8255A-2 O.K.

8254 O.K.

8259 O.K.

```
/* ===== */
/*          Program Name : iotest.c          */
/*    Testing the I/O card.                  */
/* ===== */
#include <dos.h>
#include <conio.h>
#define PORT1 0x3e3
#define PORT2 0x3e7
#define PORT3 0x3eb
unsigned char val[] = { 0x13, 0x00, 0x00, 0x01, 0x55 };
int addr[] = { 0x3ec, 0x3ed, 0x3ed, 0x3ed, 0x3ed };

void main()
{
    unsigned char byteread, bytewrite, tmp;
    int PORT;
    int i;
    unsigned char testing = 0x55;

    clrscr();          /* clear the screen */
    gotoxy(34,2);
    printf("I/O Card Testing"); /* print the program title */

    /* Testing the 8255A-1 */
    gotoxy(30,5);
```

```

    printf("8255A-1");
/* * set PORT 3e0 for output * /
    bytewrite = 0x80;
    outportb(PORT1,bytewrite);

    bytewrite = testing;
    PORT = 0x3e0;
    outportb(PORT,bytewrite); /* * send testing data to PORT 0x3e0 * /

/* * set PORT 3e0 for input * /
    bytewrite = 0x90;
    outportb(PORT1,bytewrite);

    byteread = inportb(PORT); /* * get the data from PORT 0x3e0 * /
    gotoxy(41,5);
    if ( byteread == testing )
        printf("O.K.");
    else
        printf("%dFail or not implemented.",byteread);

/* * Testing the 8255A-2 * /
    gotoxy(30,7);
    printf("8255A-2");
/* * set PORT 3e4 for output * /
    bytewrite = 0x80;
    outportb(PORT2,bytewrite);

    bytewrite = testing;
    PORT = 0x3e4;
    outportb(PORT,bytewrite); /* * send testing data to PORT 0x3e4 * /

/* * set PORT 3e4 for input * /
    bytewrite = 0x90;
    outportb(PORT2,bytewrite);

    byteread = inportb(PORT); /* * get the data from PORT 0x3e4 * /
    gotoxy(41,7);
    if ( byteread == testing )
        printf("O.K.");
    else
        printf("Fail or not implemented.");

/* * Testing 8254 * /
    gotoxy(30,9);

```

```

printf("8254");

bytewrite = 0x14;
outportb(PORT3,bytewrite); /* send control data to PORT3 */

PORT = 0x3e8;
bytewrite = testing;
outportb(PORT,bytewrite); /* send testing output to 0x3e8 */

byteread = inportb(PORT); /* get the first input */
tmp = byteread;

byteread = inportb(PORT); /* get the second input */
if ( tmp != byteread )
{
    gotoxy(41,9);
    printf("O.K.");
}
else
{
    byteread = inport(PORT); /* get the third input */
    if ( tmp != byteread )
    {
        gotoxy(41,9);
        printf("O.K.");
    }
    else
    {
        gotoxy(41,9);
        printf("Fail or not implemented.");
    }
}

/* Testing 8259 */
gotoxy(30,11);
printf("8259");

for ( i = 0; i < 5; i++ )
{
    bytewrite = val[i];
    PORT = addr[i];
    outportb(PORT,bytewrite);
}

```



```

PORT = 0x3ed;
byteread = inportb(PORT);
gotoxy(41,11);
if ( byteread == testing )
    printf("O.K.");
else
    printf("Fail or not implemented.");
}

```

## 2.5 思考题

1. 这个程序实际上只测了PORTA，只能保证D0~D7, A0, A1,  $\overline{WR}$ ,  $\overline{RD}$ ,  $\overline{CS}$ , RESET, PA0~PA7 这些引脚的连接没有问题。PB0~PB7 及 PC0~PC7 的连接并没有真正测到。读者可以自行测试它。
2. 还有哪些部分没有测到？测出它。
3. 以其它的线路重新测试这个 I/O 适配器。

## 第三章 可编程外设接口 8255A

### 本章学习目的

1. 读者可以从本章内容中了解8255A的基本概念及应用，以便彻底研究8255A的内部结构，进而全盘了解 8255A。
2. 第四章为8255A的应用示例，请读者多研读本章内容，以准备学习下一章的应用示例。

### 本章内容

- 3.1 8255A 结构
- 3.2 8255A 内部结构与功能简介
- 3.3 模式的编程
- 3.4 8255A 的操作模式
- 3.5 C 口单一位的设置/复位
- 3.6 操作模式的研究
- 3.7 混合模式
- 3.8 特殊模式的综合考虑
- 3.9 C 口的应用考虑
- 3.10 各种工作模式的定义
- 3.11 8255A 的应用实例
- 3.12 思考题

8255A 是美国英特尔公司(Intel)为所生产的微处理器而设计出的多功能接口外设控制器。现今几乎所有的 IBM PC 中都有这一片或类似功能的 IC，如 82380。研究 IBM PC 接口必须对这片 IC 深入了解，才能开发外设的应用程序，或创作新的硬件接口。

### 3.1 8255A 结构

8255A 为 CMOS 的生产工艺所研制成的，一般而言，这一类技术的优点为省电，缺点为速度较 TTL 技术所研制的 IC 为慢。

以下为图 3.1 的引脚说明

PA0~PA7 : 8位数据输出/输入。

PB0~PB7 : 8位数据输出/输入。

$\overline{RD}$  : 微处理器产生的读取信号，当  $\overline{CS}=0$ (Chip enable)， $\overline{RD}=0$ 时微处理器会将 8255A 的数据读进微处理器中。

GND : 电源接地。

A0,A1 : 地址线, 由微处理器送出, 用以定义控制寄存器的地址。工作方式如下表 3.1 所示。

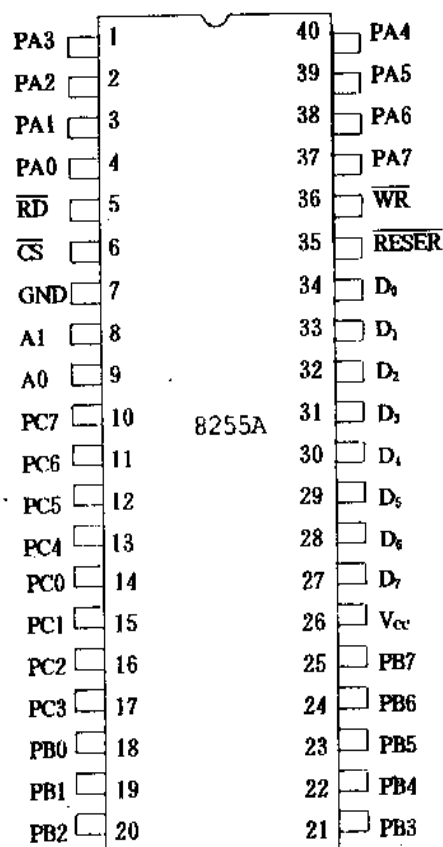


图 3.1 8255A 引脚图

表 3.1 8255A 的操作信号功能

A1	A0	RD	WR	CS	输入操作 (读取)
0	0	0	1	0	A 口→数据总线
0	1	0	1	0	B 口→数据总线
1	0	0	1	0	C 口→数据总线
					输出操作 (写入)
0	0	1	0	0	数据总线→A 口
0	1	1	0	0	数据总线→B 口
1	0	1	0	0	数据总线→C 口
1	1	1	0	0	数据总线→控制寄存器
					禁止态
x	x	x	x	1	数据总线为第三态 (高阻抗)
x	x	1	1	0	数据总线为第三态 (高阻抗)

D0~D7 : 8位三态输出/输入数据总线。

PC4~PC7 : C口的高四字节。

PC0~PC3 : C口的低四字节。PC0~PC7可由控制寄存器编程成不同的功能。  
 $V_{cc}$  : 正电源为+5V。  
 RESET : 由微处理器送出的复位信号。  
 $\overline{WR}$  : 由微处理器产生的写入信号, 当 $\overline{CS}=0, \overline{WR}=0$ 时, 微处理器会将数据写入 8255A 中。

注: ×表示 Don't Care, 其值不重要。

### 3.2 8255A 内部结构与功能简介

8255A 主要由下列的线路组成

- 数据总线缓冲区 (Data Bus Buffer)
- 读/写控制逻辑 (Read/Write Control Logic)
- A 组控制逻辑 (Group A Control)
- B 组控制逻辑 (Group B Control)
- A, B, C 口

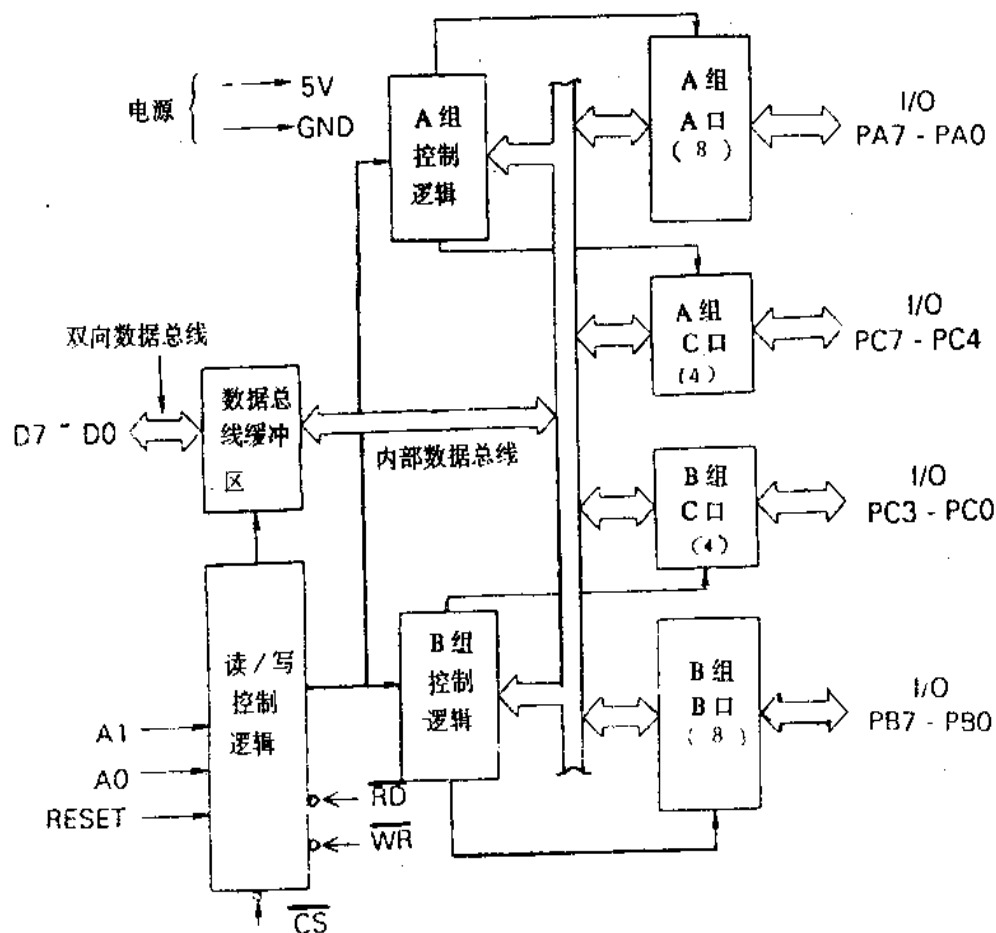


图 3.2 8255A 的结构

图 3.2 说明:

1. 从微处理器传送数据 D7~D0 至数据总线缓冲区。
2. 由读/写控制逻辑决定, 读取或写入 A, B, C 口数据。
3. A, B 组控制逻辑决定 A, B, C 口的个别不同操作模式。

A 组数据: PA0~PA7, PC4~PC7。

B 组数据: PB0~PB7, PC0~PC3。

### 3.3 模式的编程

8255A 即通过以下的控制字定义出不同的工作模式。

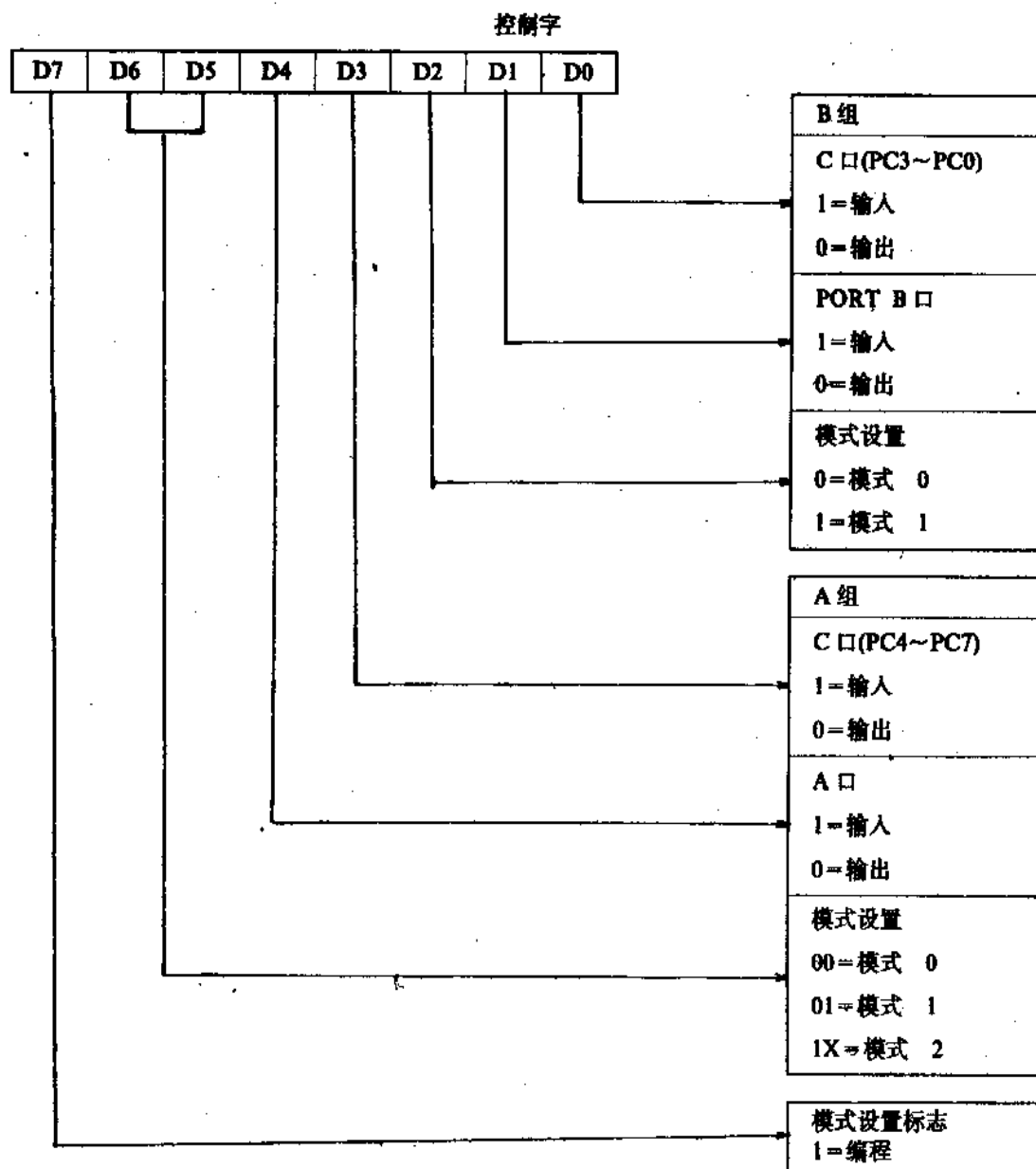


图 3.3 控制字的模式定义表

以下为图 3.3 中各位的说明:

D7: 当要编程时, 字节 D7 必须为 1.

D6, D5: 用以选择 A 组的工作模式.

D4: 决定 A 口(PA0~PA7)为输出或输入型.

D3: 决定 C 口(PC4~PC7)为输出或输入型.

D2: 用以选择 B 组的工作模式.

D1: 决定 B 口(PB0~PB7)为输出或输入型.

D0: 决定 C 口(PC0~PC3)为输出或输入型.

控制字(Control Word)的地址线分析如下:

A7 A6 A5 A4 A3 A2 A1 A0

x x x x x x 1 1

x: 根据硬件要求编程为 0 或 1. 本书讨论的 I/O 适配器上有两个 8255A, 它们的控制组地址分别为 03E3H 及 03E7H. 其中 A0, A1 均为 1.

### 3.4 8255A 的操作模式

8255A 可由微处理器编程成三种基本工作模式. 当开机后或系统复位后, 三个口均被设置为输入型缺省.

模式 0: 请参考图 3.4, 基本的 I/O, PA, PB, PC 均可由用户自行编程成输出或输入. 此时 PC0~PC7 也可视为独立的口.

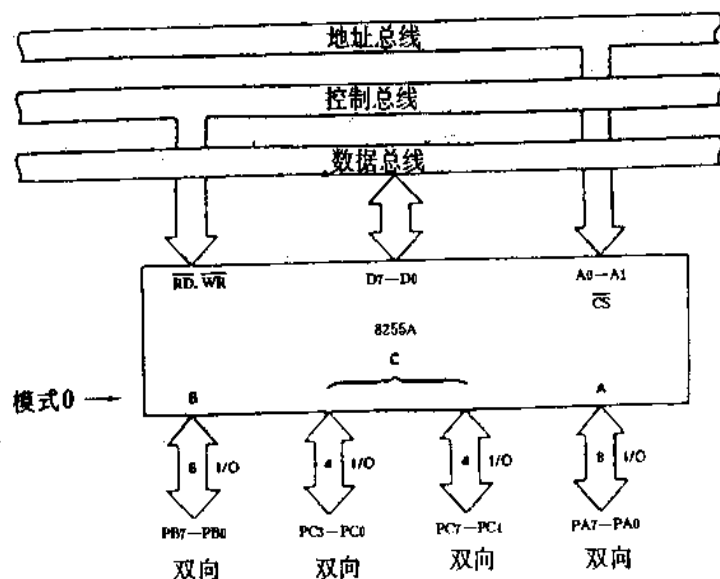
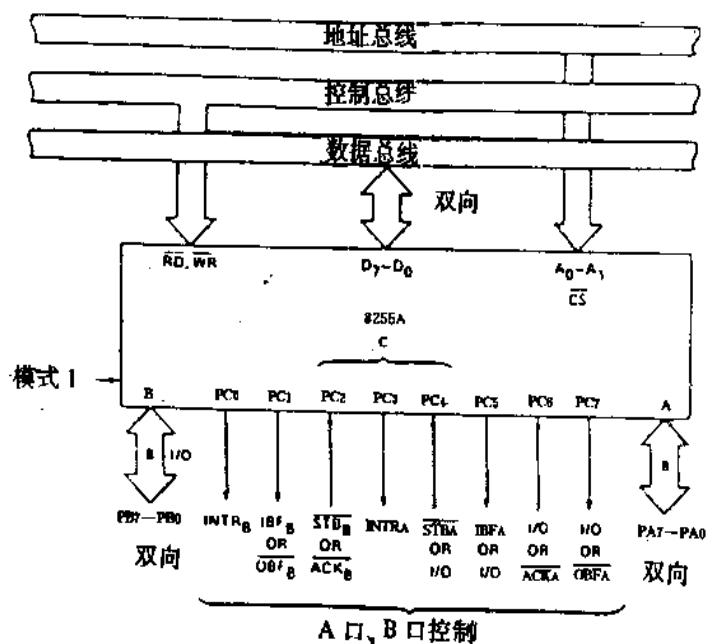


图 3.4 操作模式 0 的总线结构

模式 1: 请参考图 3.5, PA 及 PB 可由用户编程成输出或输入. PC3~PC7 伴随 PA, 当成 PA 交互式的握手控制信号(Handshake).



**图 3.5 操作模式 1 的总线结构**

**模式2:** 请参考图3.6, 在此种模式下, A口有输出及输入的功能, C口为双向控制式 I/O 的控制信号。此时 B 口仍可独立被编程为任一种模式。

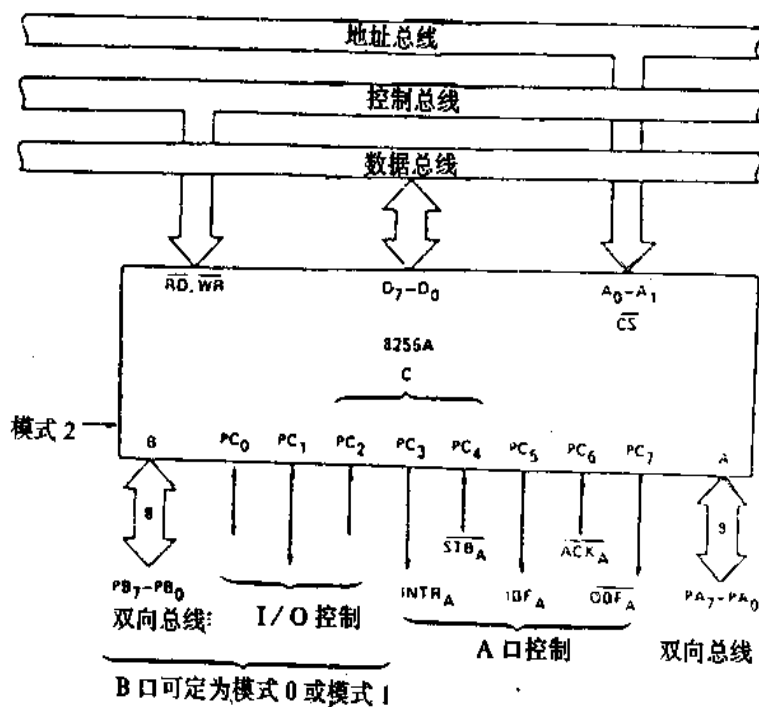


图 3.6 操作模式 2 的总线结构

### 3.5 C口单一位的设置/复位

C口除了可以配合A、B口的交互式信号使用之外，它的任何位也可被OUT命令设置或复位。这种特性可以减少软件的需求，换言之，可以缩小软件程序，而仍能达到系统所要求的功能。注意：在位元设置/复位的命令格式中，D7必须为0，否则会 and 模式设置的格式相混。图3.7为C口位设置/复位命令。

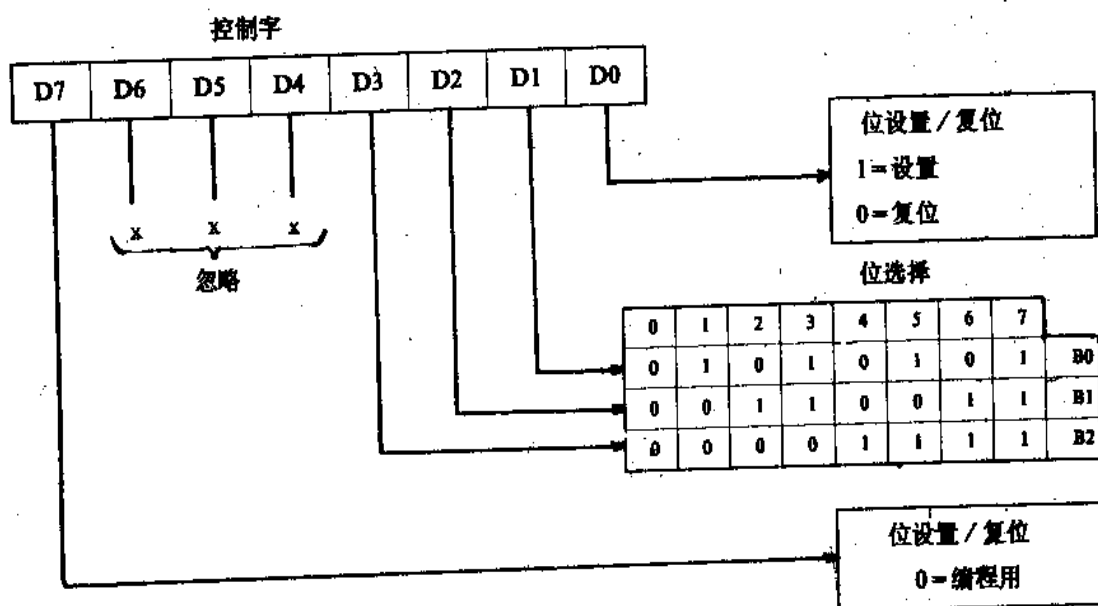


图 3.7 位设置/复位命令的格式

位选择:

D7	D6	D5	D4	D3	D2	D1	D0	
0	X	X	X	0	0	0	1	设置 PC0 为 1
0	X	X	X	0	0	0	0	设置 PC0 为 0
0	X	X	X	0	0	1	1	设置 PC1 为 1
0	X	X	X	0	0	1	0	设置 PC1 为 0
...				...				...
0	X	X	X	1	1	1	0	设置 PC7 为 0

有了这一种功能，当8255A操作于模式1或2模式时可以单独定义C口任何一个位。这样也可以随时中断微处理器，进行其它操作。



### 3.6 操作模式的研究

以下分别说明三种模式的操作研究及时序说明。

#### 3.6.1 模式 0 的研究

模式 0 为基本的 I/O 模式，它的特性如下：

- 有 2 个 8 位的口 PA 及 PB。
- 有 2 个 4 位的口 PC0~PC3, PC4~PC7。
- 任何口均可作为输出或输入。
- 输出有锁定功能，即当输出为 0 或 1 时，数据可以保持至下一次输出为止。
- 输入没有锁定功能，数据并不保存在口上。
- 16 种不同输出输入组合。

表 3.2 为 A 组及 B 组所有基本输出输入情形的选择组合摘要。

表 3.2 由 D4, D3, D1, D0 所选择的输入/输出模式

A		B		GROUP A			GROUP B	
D4	D3	D1	D0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

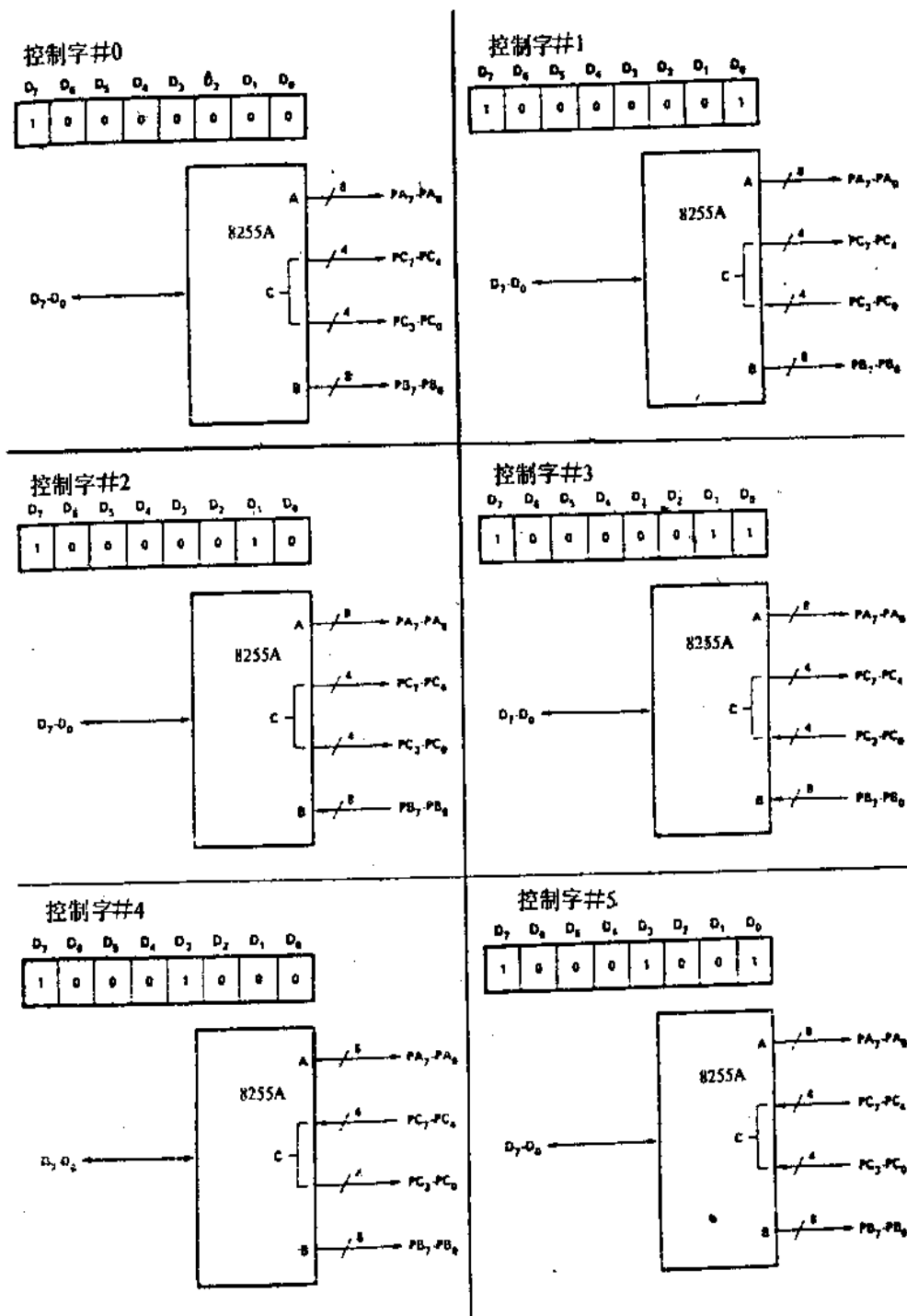


图 3.8A 模式 0 的组合

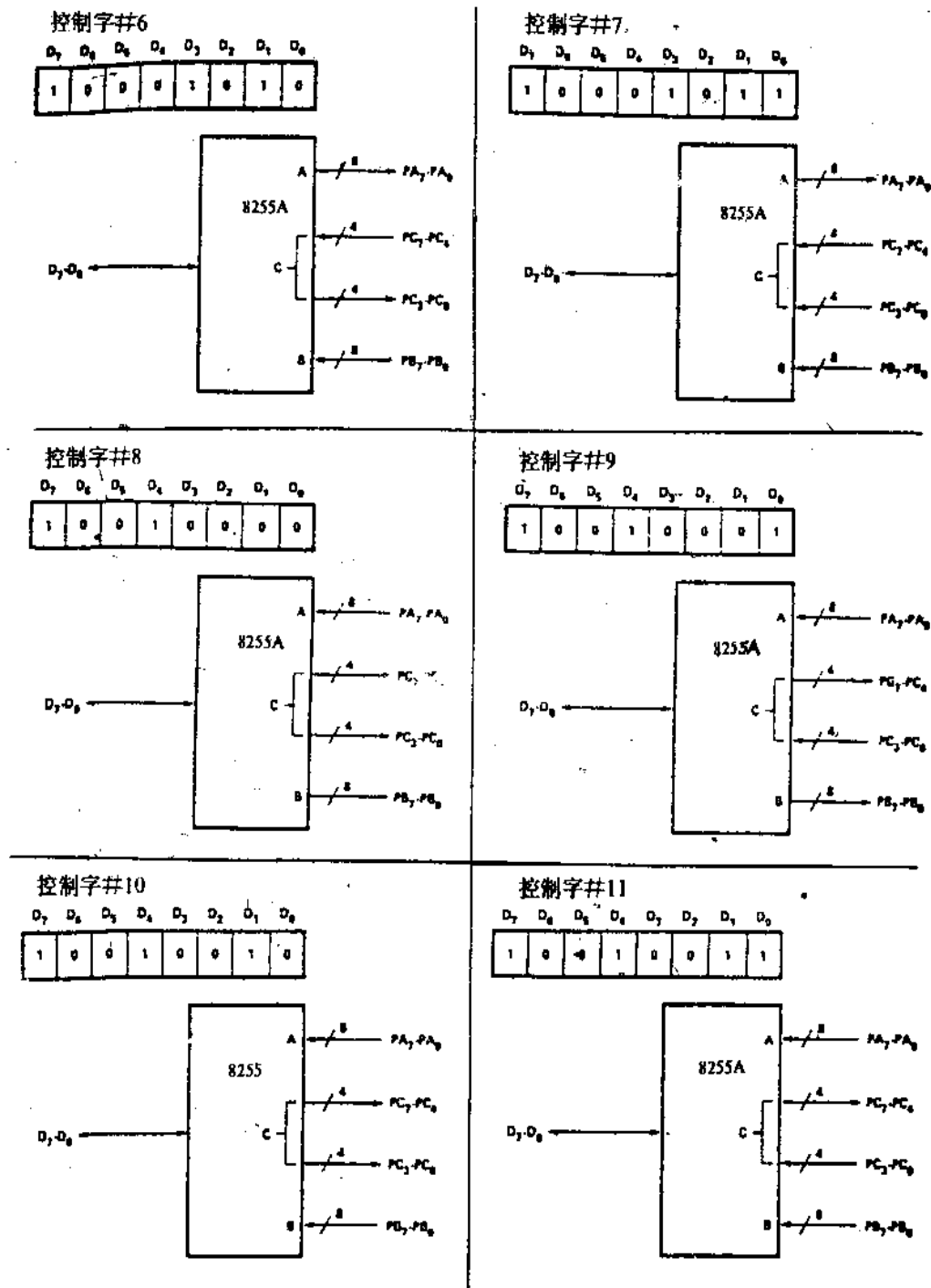
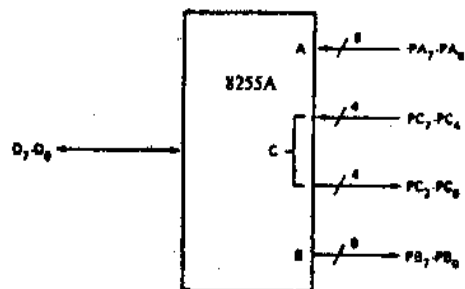


图 3.8B 模式 0 的组合

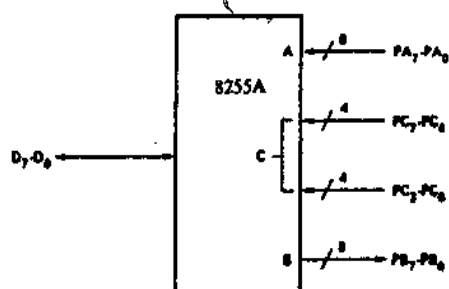
控制字#12

$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$
1	0	0	1	1	0	0	0



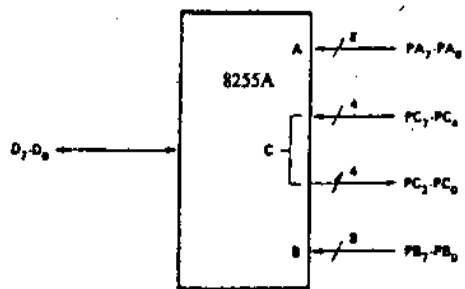
控制字#13

$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$
1	0	0	1	1	0	0	1



控制字#14

$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$
1	0	0	1	1	0	1	0



控制字#15

$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$
1	0	0	1	1	0	1	1

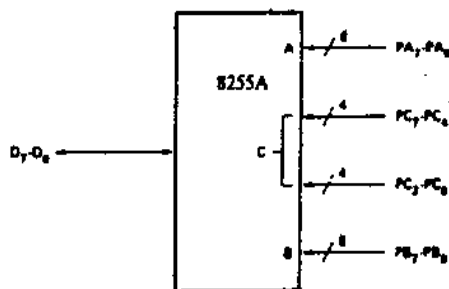


图 3.8C 模式 0 的组合

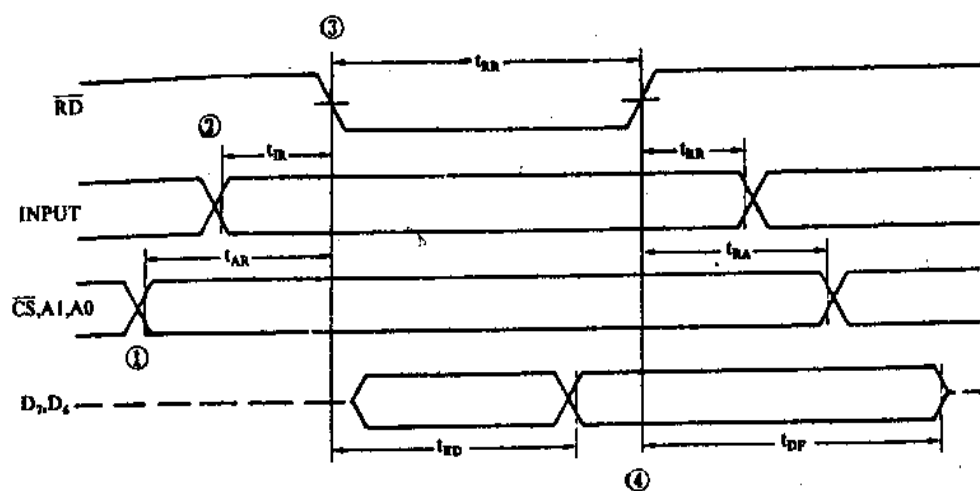


图 3.9 模式 0 (基本输入) 时序图

基本输出输入的时序说明如图 3.9、3.10，它们并不须利用交互式信号，只是单纯的输出或输入。

图 3.9 时序说明：

- ①微处理器送出地址信号，经解码后，8255A 被启动。
- ②先前控制字已完成了输入型的操作，所要读的数据此时会出现在INPUT Port上。
- ③微处理器送出读取的信号( $\overline{RD}$ )。
- ④利用此边缘，微处理器将8255A的数据由输入口读进系统。此时D7~D0的值即为INPUT 的值。

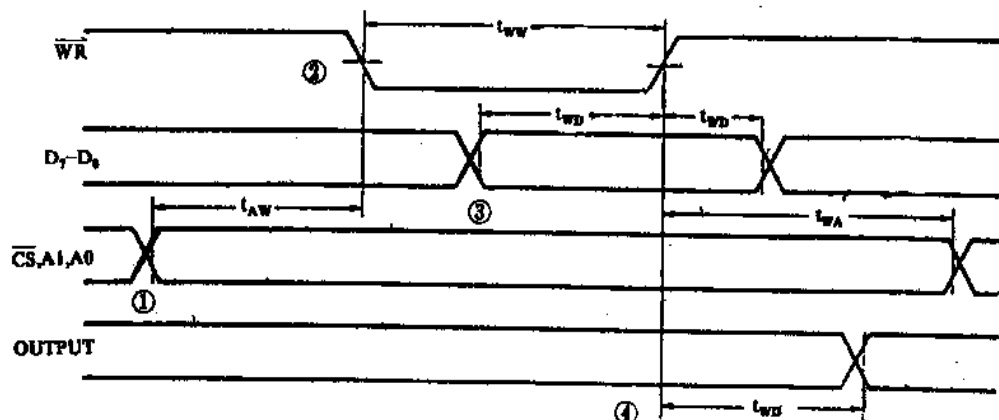


图 3.10 模式 0 (基本输出) 时序图

图 3.10 时序说明：

- ①微处理器送出地址信号，经解码后，8255A 被启动。
- ②微处理器送出写( $\overline{WR}$ )的信号。
- ③要输出的数据由微处理器送出。
- ④经过  $t_{DW}$  后，数据稳定了，微处理器利用  $\overline{WR}$  将数据写入8255A口中。此时，再经过  $t_{WB}$  时间之后，OUTPUT 即存有微处理器送出的数据。

### 3.6.2 模式 1 的研究

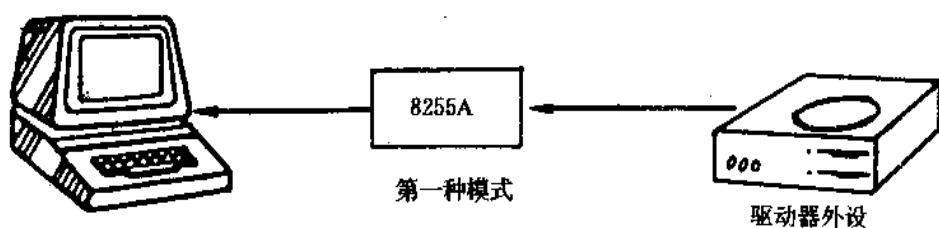
模式 1 为选通输入/输出类型。在此模式下，数据可以从口传送至外设并等待外设的响应信号(ACK)，这种数据传送称为交互式(Handshake)。我们常在操作驱动器时，因为忘记放磁盘，而出现(Retry Abort)的信息。这也是因为微处理器等不到驱动器的响应信号而拒绝继续送出数据，而将此送不出数据的错误信息告诉用户，加强了数据传送的可靠性。

模式 1 的特性为：

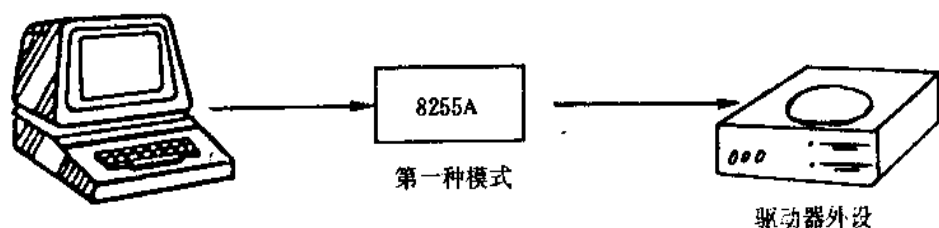
- 共分为 2 组口，A 和 B。
- 各组分别拥有 8 位的数据口及 4 位的控制口。
- 8 位的数据口可以作为输出或输入。
- 4 位的控制口用以说明及控制 8 位的数据口。
- 当编程为输入口时，外设装置传至微处理器的数据必须由外设控制电路产生选通信号(Strobe Input, STB) 锁住，此时数据即瞬间被锁住。

## 模式 1 的应用场合

1. 输入: 微计算机要从驱动器取数据。



2. 输出: 微计算机要送数据给驱动器。



模式 1 的输入结构下图 3.11 说明

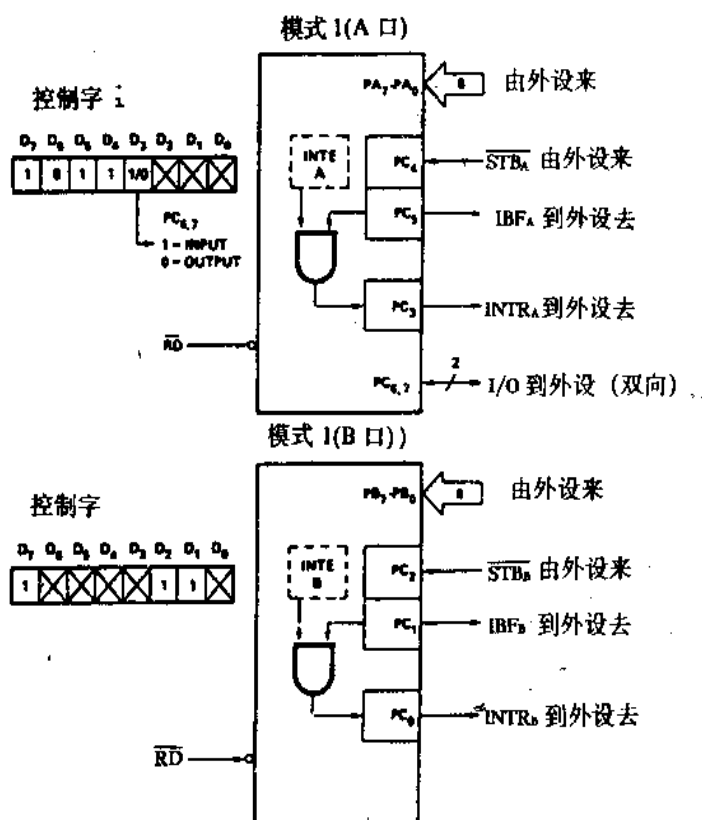


图 3.11 模式 1 的输入结构

### 3.6.2.1 A 组的编程

控制字:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1/0	X	X	X

控制字节说明:

- D7 : 当编程时D7必须为1.  
 D6,D5=0,1 : A组被编程为模式1.  
 D4=1 : A口被编程为输入型.  
 D3=1 : PC6,PC7被编程为输入口.  
 0 : PC6,PC7被编程为输出口.  
 D2,D1,D0 : 不用,可为1或0.

以下分别讨论 PC3, PC4, PC5 被编程为控制信号时所扮演的角色.

PC4:  $\overline{STB}_A$  A口的选通输入(Strobe Input). 当此信号为低电平时, 可将外设传过来的数据写入 A 口的内部输入锁存器 (Input Latch).

PC5:  $IBF_A$  输入缓冲区已满(Input Buffer Full)当数据输入A口的输入锁存器后, 此信号会被 8255A 定为高电平, 通知外设两件事:

1. 8255A正忙于处理数据.
2. 不要再送数据给8255A.

微处理器此时会将8255A的数据利用 $\overline{RD}$ 读走, 此时,  $IBF_A$ 又变为低电平, 可以再处理外设其它数据.

PC3: INTR A口的中断请求.

当 $IBF_A$ 为高电平及 $INTE_A$ 也为高电平时, INTR会变为高电平, 如果此时与 PC 系统相连接, 可以中断微处理器, 通知它任何的信息, 譬如常用的“数据已送进 8255A 的输入口, 你(微处理器)可以去取数据了.”

INTE A: 是一个内部信号, 由 PC4 控制.

INTE B: 是一个内部信号, 由 PC2 控制.

图 3.12 为整个输入过程的时序说明

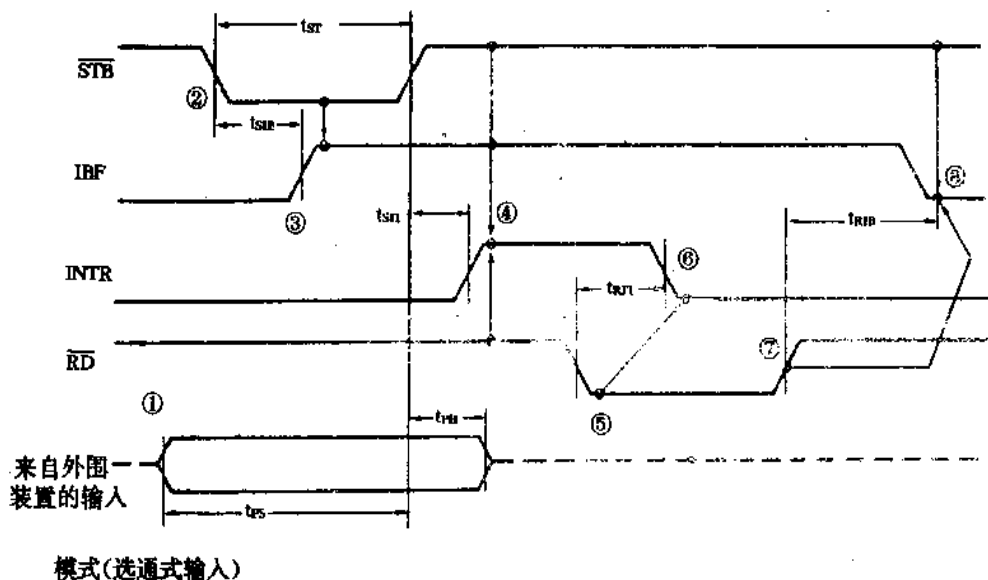


图 3.12 模式 1 的输入时序

- ①外设（如打印机）将数据送至 8255A 的口上。
- ②来自外设的 $\overline{STB}$ 会告诉8255A，数据已准备好了，并利用 $\overline{STB}$ 将数据锁入口A的输入锁存器。
- ③8255A收到数据后会产生IBF给外设，同时告诉外设，数据已满不要再送数据。
- ④8255A也送出INTR给微处理器通知微处理器，“我的数据准备好了，你可以来取走。”
- ⑤微处理器送出RD，将数据取走。
- ⑥INTR 的工作完成了，复位为 0。
- ⑦RD的工作完成了，复位为 0。
- ⑧IBF 复位为 0，准备输入由外设送过来的另一批数据。

由以上说明，我们可以知道，外设、8255A 及微处理器三者配合得完美无缺。这种数据传送，通过了一些控制信号，好像将数据传给对方手中一样安全。所以称为交互式的传输(Handshake)。

### 3.6.2.2 B 组的编程

D7	D6	D5	D4	D3	D2	D1	D0
1					1	1	

控制字说明：

D7=1: 当编程时 D7 必须为 1。

D6, D5, D4, D3: 为 A 口控制部分，不用。

D2=1: B 口为模式 1 的选择。

D1=1: B 口为输入的选择。

D0: 不使用。

B 口的操作说明：

PB :为输入口。

PC2: 为 B 口的 STB。

PC1: 为 B 口的 IBF。

PC0: 为 B 口的 INTR。

由以上的编程情形，可以知道 A 口和 B 口的编程是独立的，它们可以同时编程为不同的工作模式。以下列四种编程示例说明：

1.

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	1	1	0	1	1
	$\Delta$	$\Delta$			$\Delta$		

A 口为第 0 模式，B 口也为第 0 模式

2.

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1	1	1	
	$\Delta$	$\Delta$			$\Delta$		



A 口为第 1 模式, B 口亦为第 1 模式

3.

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	1	1	1	1	
	△	△			△		

A 口为第 0 模式, B 口为第 1 模式

4.

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1	0	1	
	△	△			△		

A 口为第 1 模式, B 口为第 0 模式  
模式 1 的输出结构以图 3.13 说明。

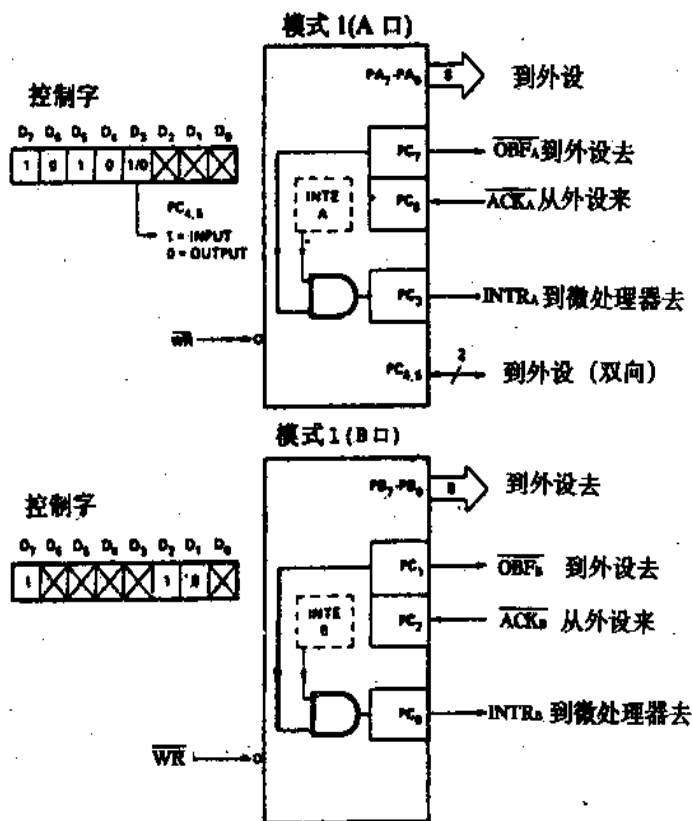


图 3.13 模式 1 的输出结构

A 口:

控制字

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	0	1/0			

D7=1: 编程时 D7 必须定为 1.

D6, D5=0, 1: A 口选择模式 1.

D4=0: 设置 A 口为输出模式.

D3=1: PC4, PB5 被编程为输入型.

0: PC4, PC5 被编程为输出型.

D2, D1, D0: 为 B 口控制部分, 不用.

以下分别讨论 PC3, PC6, PC7 被编程为控制信号后的传输情形.

PC7:  $\overline{\text{OBF}}$  (Output Buffer Full), 输出缓冲区数据已满.

当微处理器将数据传给 8255A 时, 8255A 会产生此信号告诉外设“从微处理器来的数据已到, 我准备好了, 随时可以送给你 (外设)” (低电平有效).

PC6:  $\overline{\text{ACK}}$  (Acknowledge) 确认信号. 由外设通知 8255A, “数据我已经收到了” (低电平有效).

PC3:  $\text{INTR}$  (Interrupt Request) 中断请求, 由 8255A 传给微处理器. “你传给外设的信号, 它已收到了.”

图 3.14 为整个输出过程的时序说明

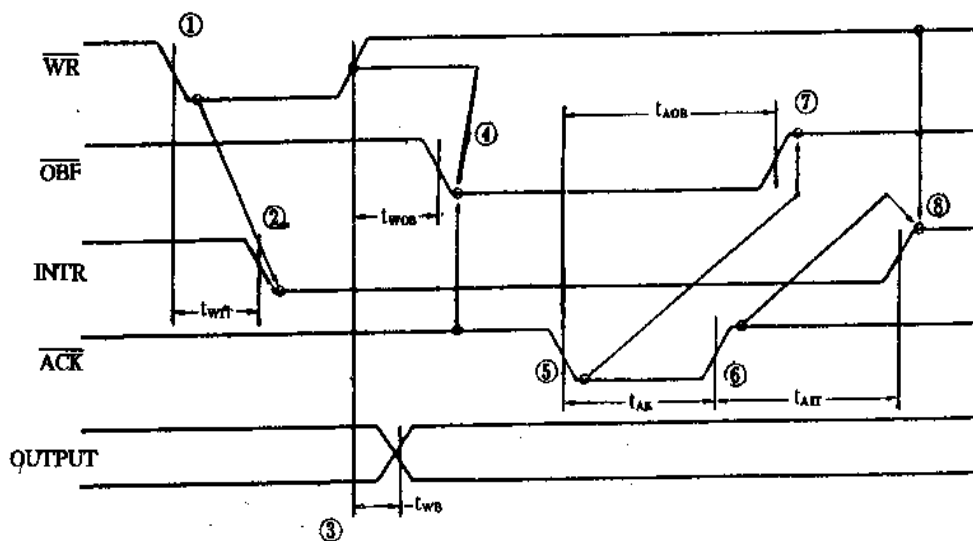


图 3.14 模式 1 的输出时序

①微处理器送出  $\overline{\text{WR}}$  给 8255A, 将所要传送给外设的信号写入 8255A 中.

②同时并将先前一次 8255A 产生的  $\text{INTR}$  信号清除.

③  $\overline{\text{WR}}$  信号结束, 此时数据已传给 8255A 了.

④ 8255A 告诉外设“微处理器的数据我收到了, 也准备好了, 可以随时传给你了.”

⑤ 外设收到数据后通知 8255A “OK, 我已收到数据了.”

⑥ 确认信号结束.

⑦由⑥产生，通知外设，我的数据缓冲区已经没数据了。

⑧由⑥产生，告诉微处理器“数据已传给外设，它们也收到了”。

模式1可以同时让PA、PB为输出型及输入型，或者输入、输出型，因为PA、PB为独立口，可以分别编程。

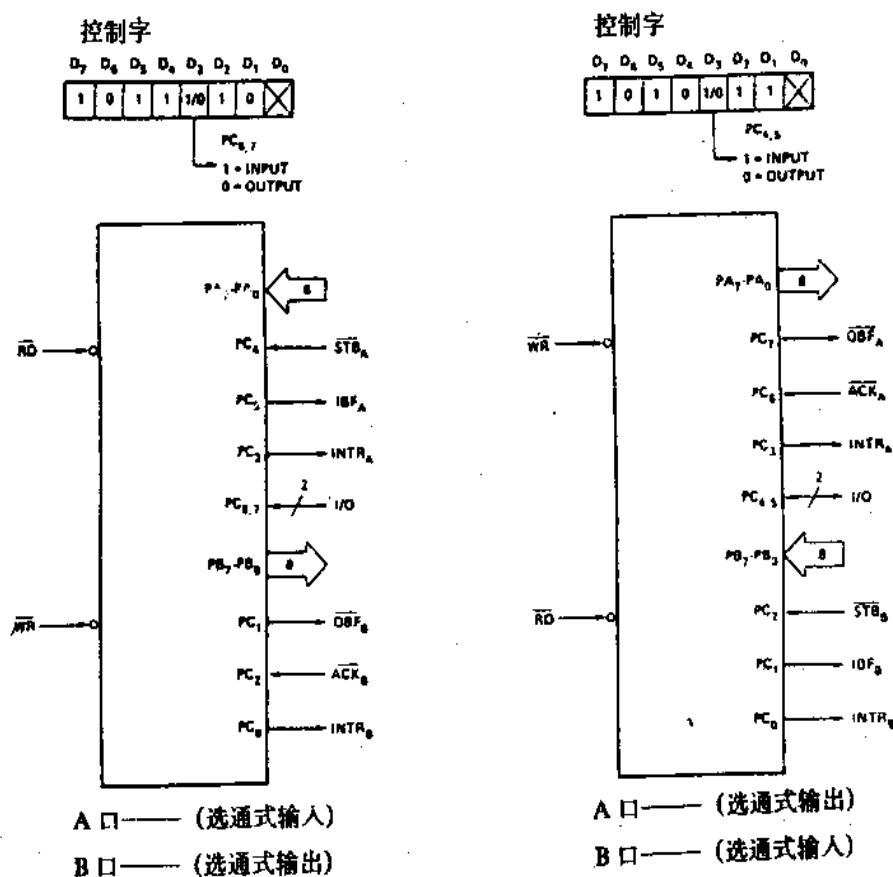


图 3.15 模式1的组态

表 3.3 为 C 口的握手信号的整理，以便读者查阅。

表 3.3 C口当成握手信号的整理

	PC0	PC1	PC2	PC3	PC4	PC5	PC6	PC7
输入	INTR <sub>B</sub>	IBF <sub>B</sub> C	STB <sub>B</sub>	INTR <sub>A</sub>	STB <sub>A</sub>	IBF <sub>B</sub>	I/O	I/O
输出	INTR <sub>A</sub>	OBF <sub>B</sub>	ACK <sub>B</sub>	INTR <sub>A</sub>	I/O	I/O	ACK <sub>A</sub>	OBF <sub>A</sub>

### 3.6.3 模式2: (Strobed Bidirection Bus I/O)双向选通式I/O总线

这种模式的特色在于A口可以同时编程成输出及输入型，注意B口无此功能。

功能基本定义:

- 只有A口可以双向传输数据。

- C 口中的 PC3, PC4, PC5, PC6, PC7 作为控制信号。
- 输出及输入均有锁存控制, 防止数据瞬间消失。
- 当 A 口定为第 2 种模式时, B 口可定为 0 或 1 模式。

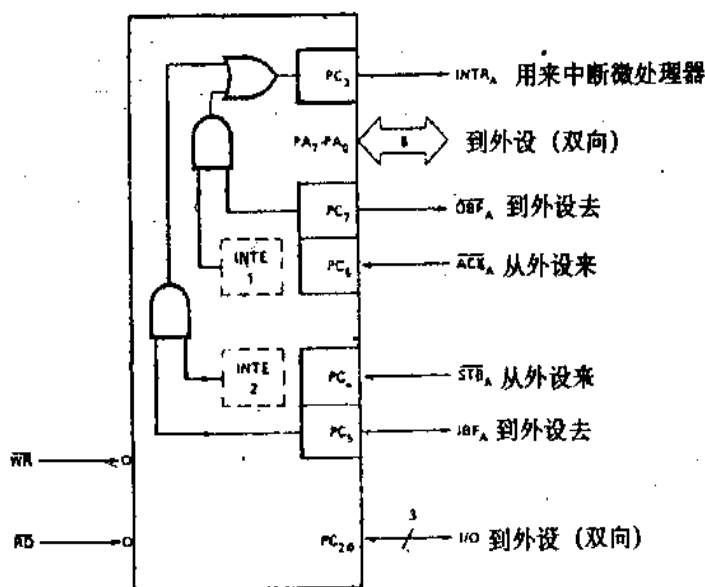
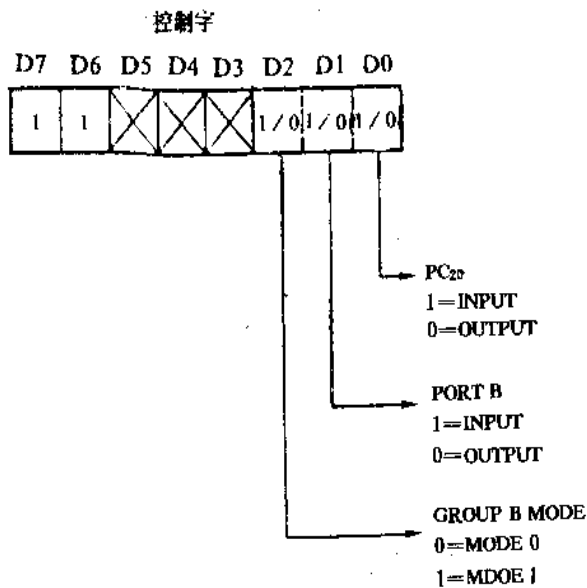


图 3.16 模式 2 的结构

以下为图 3.16 各信号的说明:

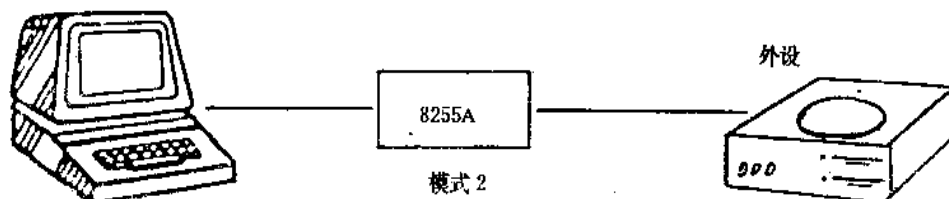
- PA: 双向传送的数据总线。
- PC3:  $\overline{INTR}_A$  (Interrupt) 中断请求。当高电平时, 用以中断微处理器。
- PC7:  $\overline{OBF}_A$  (Output Buffer Full), 输出缓冲区数据已满。当微处理器将数据传给 8255A 的输出缓冲区后, 8255A 产生此信号通知外设, “缓冲区已满”。
- PC6:  $\overline{ACK}_A$  (Acknowledge) 确认信号。当外设发出确认信号通知 8255A 后, 8255A 的 A 口将由高阻抗转为输出口, 将下一个数据 (如果有) 输出给外设。
- PC4:  $\overline{STB}_A$  (Strobe input) 选通输入, 由外设产生, 用以将外设的数据送进 A 口的输入锁存器。
- PC5:  $\overline{IBF}_A$  (Input Buffer Full) 输入缓冲区已满。当外设的输入信号送进 A 口的输入锁存器之后, 8255A 将产生此信号给外设, 表示输入缓冲区已满, 且不要再送数据进来。
- INTE1: 为 8255A 的内部信号, 由 PC6 控制。参阅 6-5。
- INTE2: 为 8255A 的内部信号, 由 PC4 控制。

模式 2 的控制字:



- D7=1 : 编程时D7必须定为1。
- D6=1,D5=× : 模式2的设置。
- D4,D3=×, × : 模式2下, A口已被设置为输出输入口了, 所以此两字位不具任何意义。
- D2=0 : 表示B组被设置为0模式。
- D2=1 : 表示B组被设置为1模式。
- D1=0 : 表示B组被设置为输出口。
- D1=1 : 表示B组被设置为输入口。
- D0=0 : 表示PC2~PC0 被定为输出口。
- D0=1 : 表示PC2~PC0 被定为输入口。

模式2的应用:



微计算机由 A 口写数据进入外设的同时, 也可以同时利用 A 口从外设读回另一批数据。

模式2的时序说明:

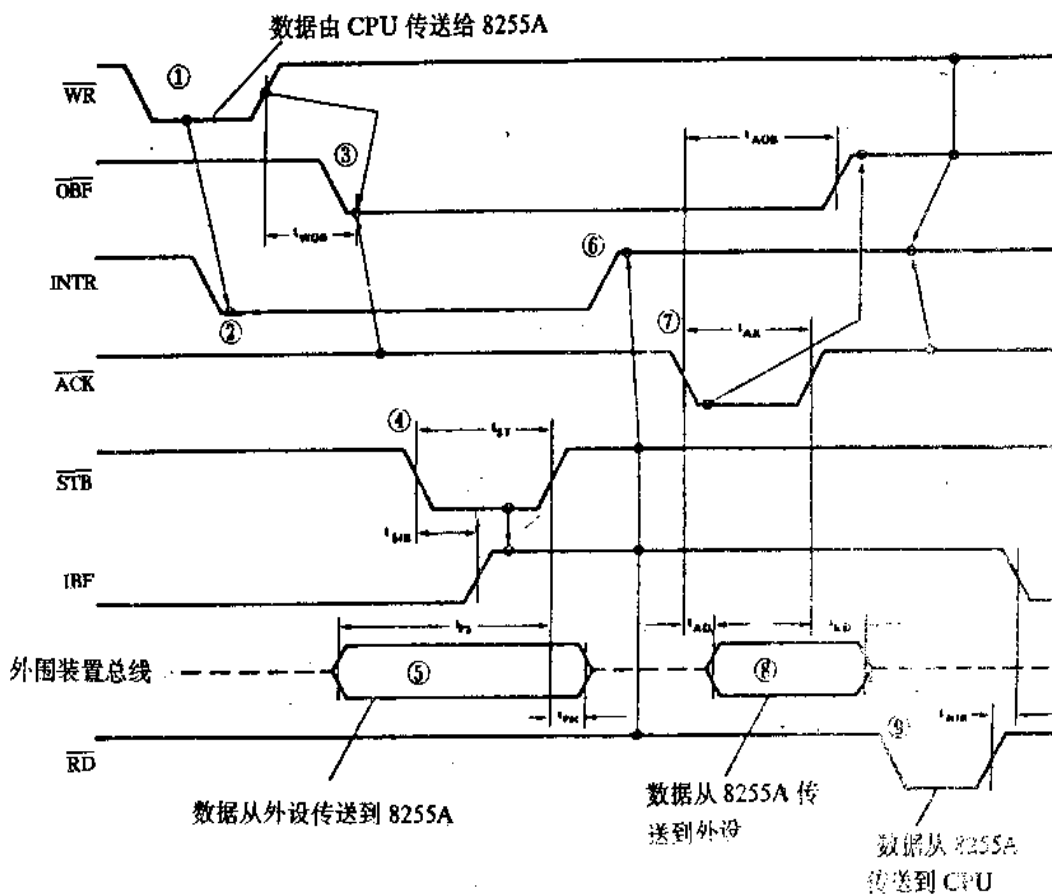


图 3.18 第二种模式的双向传输

- ①微处理器将信息利用 $\overline{\text{WR}}$ 送给 A 口的输出缓冲区。
- ②此时前一个中断申请被复位(Reset)
- ③同时OBF变为低电平，表示输出缓冲区已满。
- ④STB通知 8255A，将外设的信息送入 8255A 的输入缓冲区。
- ⑤数据从外设进入 8255A。
- ⑥8255A收到外设的数据后，利用中断申请方式通知微处理器，数据已收到了。
- ⑦外设通知 8255A，已经可以将数据送去外设了。
- ⑧数据从 8255A 的输出缓冲区进入外设。
- ⑨数据从 8255A 的输入缓冲区利用 $\overline{\text{RD}}$ 送入微处理器。

以上流程最重要的部分为下列 2 点：

- $\overline{\text{WR}}$ 必须比ACK先到
- STB必须比RD先到

$$\text{INTR} = \text{IBF} \cdot \text{MASK} \cdot \text{STB} \cdot \overline{\text{RD}} + \text{OBF} \cdot \text{MASK} \cdot \text{ACK} \cdot \overline{\text{WR}}$$

思考：既然有了第一种模式可以作为交互式的通讯传输，为什么还要第二种模式呢？

提示:

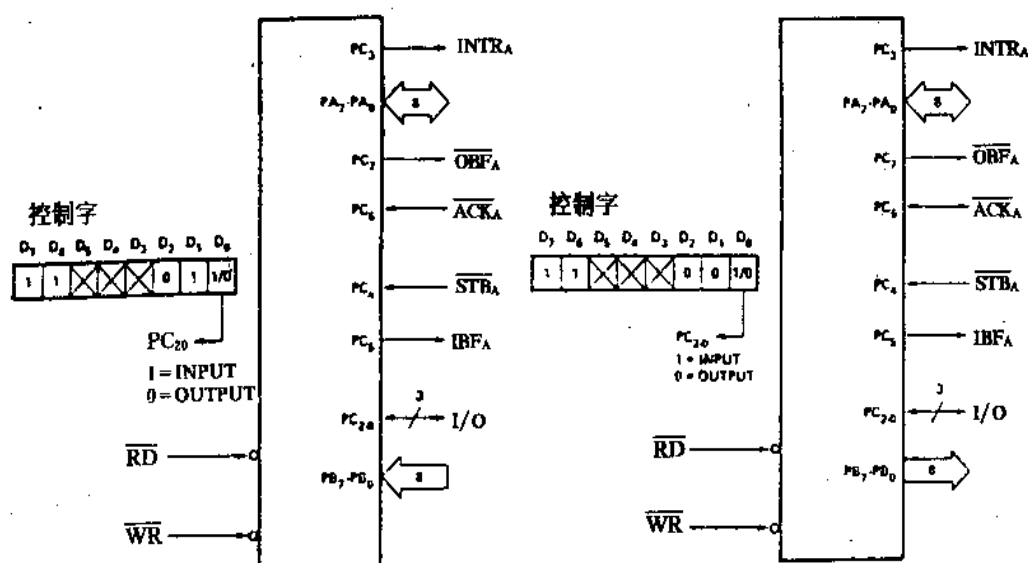
第二种模式的优点:

- 只用了 A 口即可以和外界双向通讯, 此时 B 口可以移作它用。
- 数据的传输不须利用太多的软件, 和第一种模式比较, 软件的编程较少即可达到相同的目的。

### 3.7 混合模式

模式 2 和模式 0 (输入) 组合

模式 2 和模式 0 (输出) 组合



模式 2 和模式 1 (输出) 组合

模式 2 和模式 1 (输入) 组合

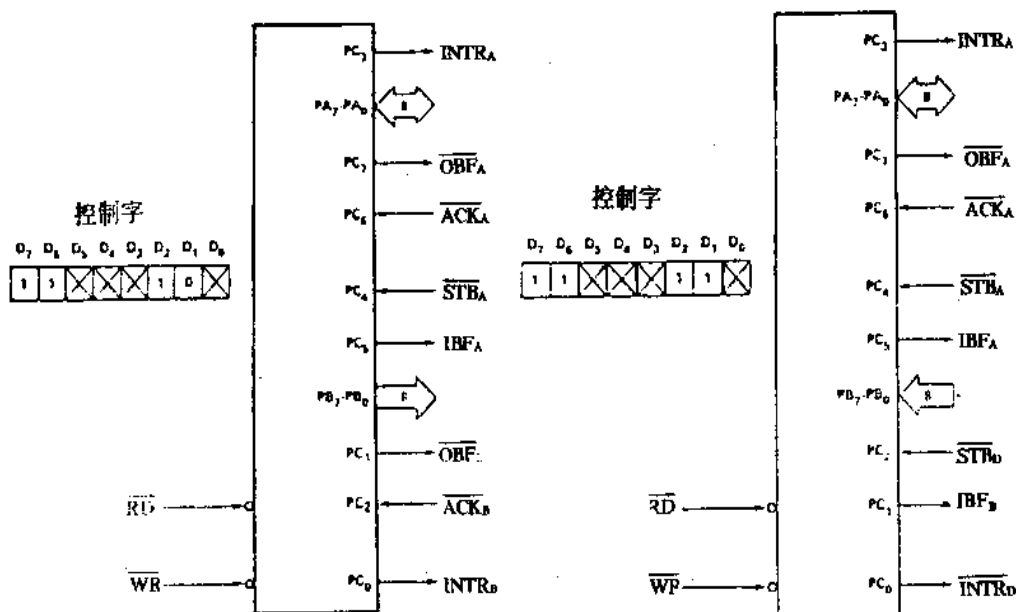


图 3.19 各种模式的混合组织

以上介绍 8255A 的三种工作模式，这三种工作模式也可以混合使用，以增强处理外设的能力。图 3.19 为各种模式的混合组态。

### 3.8 特殊模式的综合考虑

在某些应用下，C 口的位被用为控制信号，其它未被当成控制信号的部分也可被编程成输出或输入。其状态依 8255A 的操作模式而决定。

当读取 C 口时，除了  $\overline{\text{ACK}}$  和  $\overline{\text{STB}}$  所占有的 PC6、PC4 及 PC2 之外，C 口的状态应该可以被微处理器读回，因为  $\overline{\text{ACK}}$  及  $\overline{\text{STB}}$  已被外设当成输入 8255A 的控制信号后，其读回的值应为 INTE1、INTE2 及 INTEB。参阅图 3.16。

### 3.9 C 口的应用考虑

当微处理器要编程 C 口时必须考虑到 C 口此时的定义，以下有两个原则必须注意：

1. 当成输出的位，如  $\overline{\text{INTR}}$ 、IBF、OBF 均可为写入或复位。
2. 当成输入的位，如  $\overline{\text{ACK}}$ 、 $\overline{\text{STB}}$  则不能被写入或复位。

	模式 0		模式 1		模式 2
	IN	OUT	IN	OUT	A 组适合
PA <sub>0</sub>	IN	OUT	IN	OUT	↔
PA <sub>1</sub>	IN	OUT	IN	OUT	↔
PA <sub>2</sub>	IN	OUT	IN	OUT	↔
PA <sub>3</sub>	IN	OUT	IN	OUT	↔
PA <sub>4</sub>	IN	OUT	IN	OUT	↔
PA <sub>5</sub>	IN	OUT	IN	OUT	↔
PA <sub>6</sub>	IN	OUT	IN	OUT	↔
PA <sub>7</sub>	IN	OUT	IN	OUT	↔
PB <sub>0</sub>	IN	OUT	IN	OUT	—
PB <sub>1</sub>	IN	OUT	IN	OUT	—
PB <sub>2</sub>	IN	OUT	IN	OUT	—
PB <sub>3</sub>	IN	OUT	IN	OUT	—
PB <sub>4</sub>	IN	OUT	IN	OUT	—
PB <sub>5</sub>	IN	OUT	IN	OUT	—
PB <sub>6</sub>	IN	OUT	IN	OUT	—
PB <sub>7</sub>	IN	OUT	IN	OUT	—
PC <sub>0</sub>	IN	OUT	INTR <sub>B</sub>	INTR <sub>B</sub>	I/O
PC <sub>1</sub>	IN	OUT	IBF <sub>B</sub>	OBF <sub>B</sub>	I/O
PC <sub>2</sub>	IN	OUT	STB <sub>B</sub>	ACK <sub>B</sub>	I/O
PC <sub>3</sub>	IN	OUT	INTR <sub>A</sub>	INTR <sub>A</sub>	INTR <sub>A</sub>
PC <sub>4</sub>	IN	OUT	STB <sub>A</sub>	I/O	STB <sub>A</sub>
PC <sub>5</sub>	IN	OUT	IBF <sub>A</sub>	I/O	IBF <sub>A</sub>
PC <sub>6</sub>	IN	OUT	I/O	ACK <sub>A</sub>	ACK <sub>A</sub>
PC <sub>7</sub>	IN	OUT	I/O	OBF <sub>A</sub>	OBF <sub>A</sub>

只适用于  
模式 0 或  
模式 1

图 3.20 模式定义摘要



### 3.10 各种工作模式的定义

图 3.20 介绍 PA, PB, PC 在各种不同工作模式时的各个引脚使用定义。

### 3.11 8255A 的应用实例

以下介绍两种 8255A 在计算机系统中的应用。

1. 打印机接口, 如图 3.21。

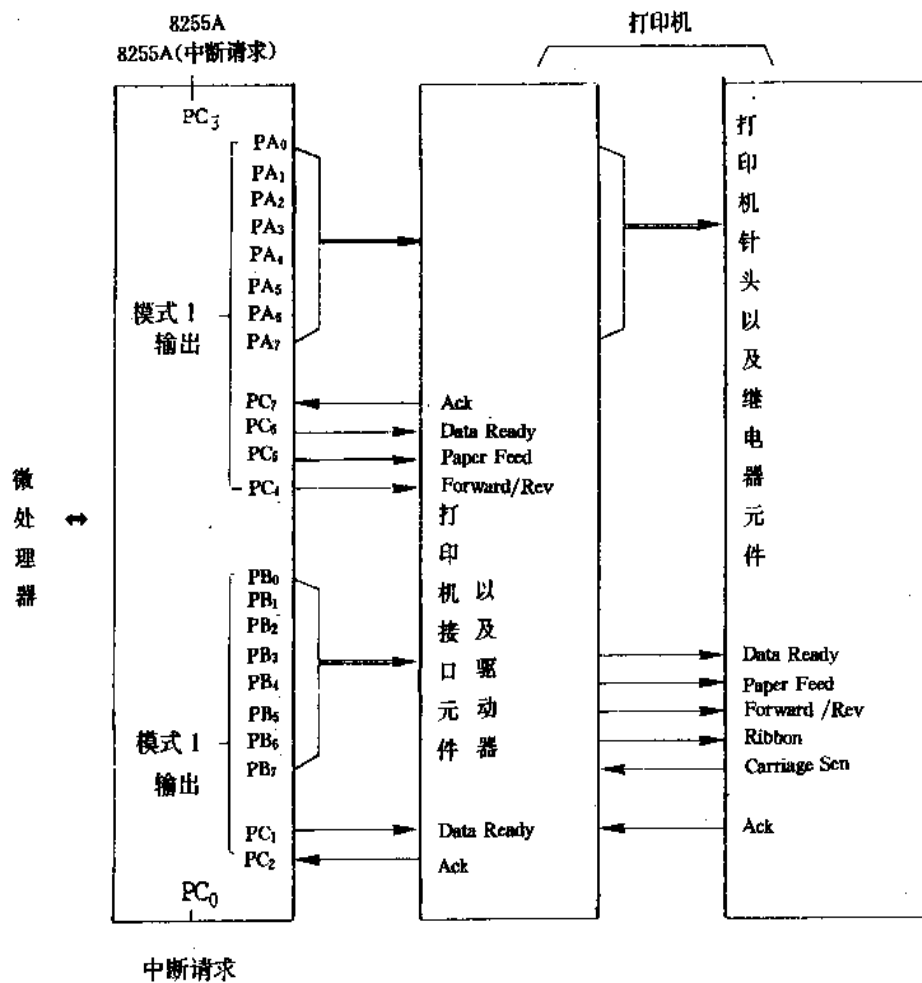


图 3.21 打印机接口

- DATA 由微处理器送给 8255A 的 PA 及 PB,再送到打印机。
- 打印机内的驱动器驱动继电器,用以控制打印机的打印针头。
- 其它信号用作握手(Handshake)的处理。
- 键盘的接口如图 3.22。

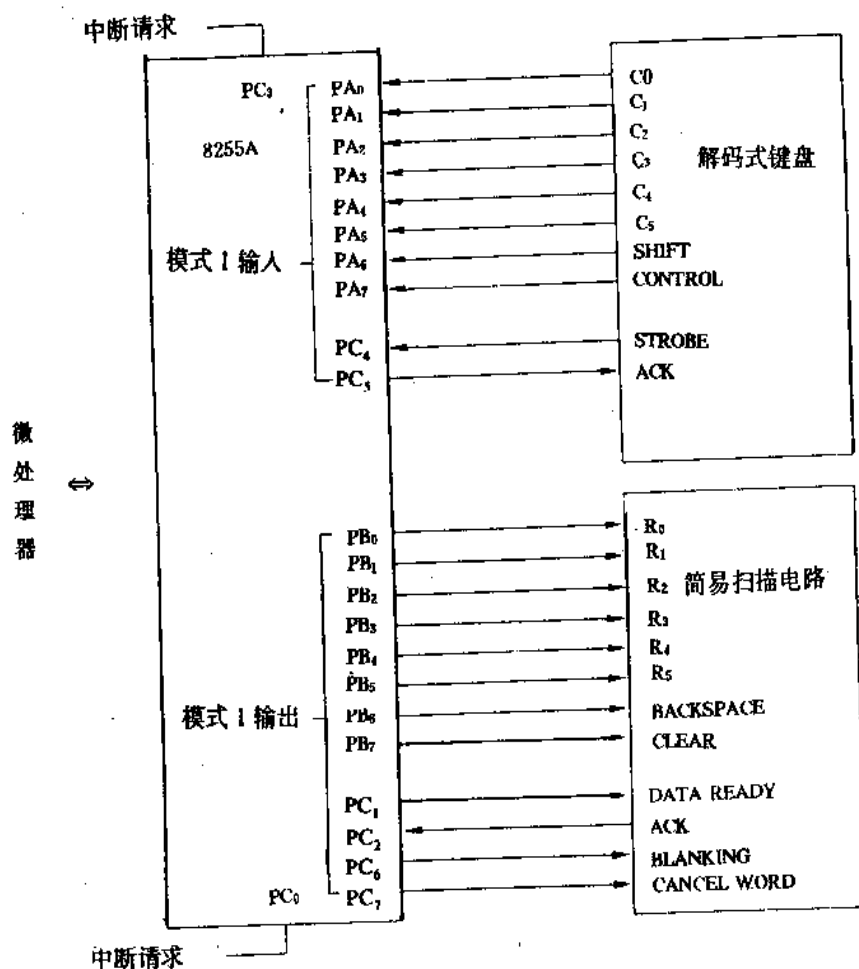


图 3.22 键盘接口

- B 口送出扫描信号。
- A 口接收 B 口的扫描信号，并检查是否有键盘被按下。
- R0~R5 为 ROW (行) 的扫描信号。
- C0~C5 为 Column (列) 的接收信号。
- 8255A 将 A 口的值传回微处理器，用以检测被按下的键码。
- 现今的复杂键盘已不再使用此种设计，取而代之的是一个微处理器，以提高键盘功能及速度。

### 3.12 思考题

1. 为什么微处理器传数据给外设时，一定要借助于8255A?
2. 微处理器为什么需要8255A用中断方式来进行数据传输?

**问题提示:**

- (1) 和外设作数据传输时一定要借助于端口 (Port)，一般微计算机的处理器没有 Port 或者 Port 太少，不足以和这么多的外设通讯。而 8255A 可以接收微处理器的指示，利用 3 个 Port 和外界通讯，替微处理器解决 Port 不足的问题。
- (2) 涉及到和外界外设的通讯，一般来说速度都不快，如果微处理器此时闲置在等待阶段，对系统而言是一种浪费，所以一般的系统工程师会让微处理器在此时作其它的事，以争取时间。当外设的数据处理完毕后，再以中断方式来通知处理器，准备下一次数据传送。

## 第四章 8255A 的基本应用示例

### 本章学习目的

1. 读者学习了8255A的基础理论之后，经过本章的示例可以对8255A的基本应用得到更深一层的了解。
2. 8255A共有三种工作模式，本章将分别以实例说明全部工作模式。

### 本章内容

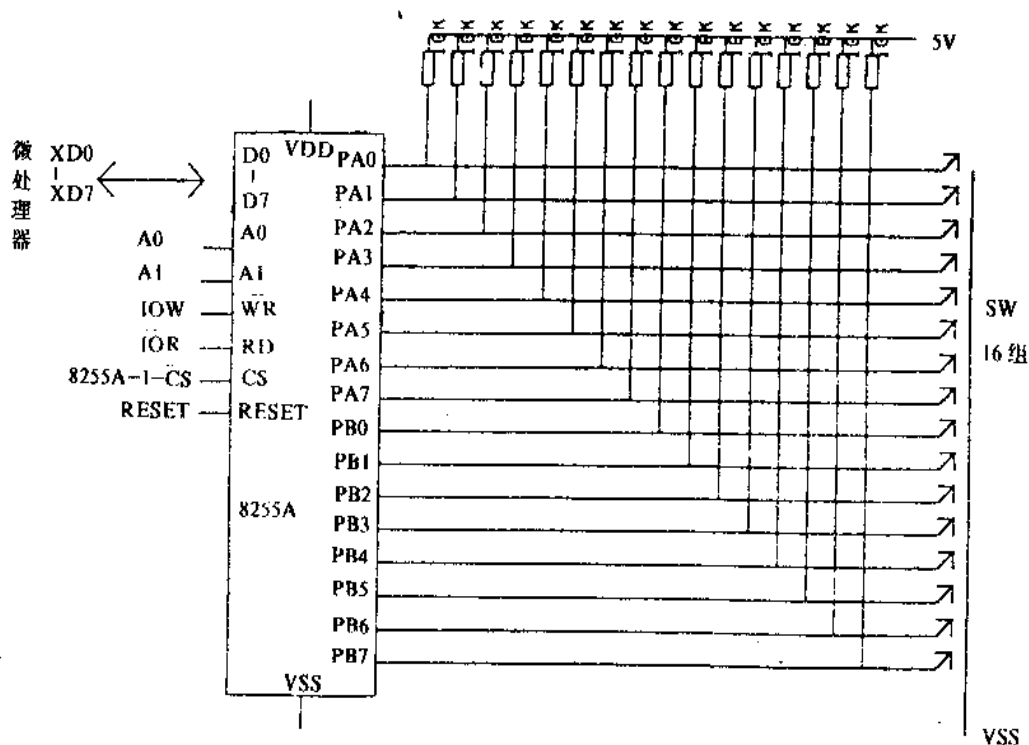
- 4.1 8255A 模式 0 的应用，PORTA 及 PORTB 均为输入型
- 4.2 8255A 模式 0 的应用，PORTA 为输入型，PORTB 为输出型
- 4.3 8255A 模式 0 的应用，PORTA 为输出型，PORTB 为输入型
- 4.4 8255A 模式 0 的应用，PORTA 和 PORTB 均为输出型
- 4.5 8255A 模式 1 的应用，模拟交互式数据输入
- 4.6 8255A 模式 1 的应用，模拟交互式数据输出
- 4.7 8255A 模式 2 的应用，双向式数据传输
- 4.8 思考题

#### 4.1 8255A 模式 0 的应用，PORTA 及 PORTB 均为输入型

##### 工作目标:

利用 8255A 模式 0，自 PORTA 及 PORTB 输入数据，然后在 IBM PC 屏幕上显示出来。

## 硬件设计:



## 程序示例 ch4\_1.C

从 PORTA 和 PORTB 输入数据，然后从屏幕显示所输入的数据。

```

/* ----- */
/*          Program Name : ch4_1.c          */
/*  Both PORTA and PORTB are input mode.    */
/*  For 8255, Mode 0 application            */
/* ----- */
#include <dos.h>
#include <conio.h>
#define SETPORT 0x3e3
#define PORTA   0x3e0
#define PORTB   0x3e1
void main()
{
    void display();
    unsigned char bytewrite, byteread;

    clrscr();      /* clear the screen */

    /* set up the PORTA and PORTB for input */
    bytewrite = 0x92;
    outportb(SETPORT, bytewrite);

```

```

/* get PORTA input and display it */
gotoxy(10,8);
printf("PORTA :");
byteread = inportb(PORTA);
gotoxy(20,8);
display(byteread);

/* get PORTB input and display it */
gotoxy(10,10);
printf("PORTB :");
byteread = inportb(PORTB);
gotoxy(20,10);
display(byteread);
}
void display(unsigned char data)
{

printf("%d",(data & 0x80) >> 7);
printf("%d",(data & 0x40) >> 6);
printf("%d",(data & 0x20) >> 5);
printf("%d",(data & 0x10) >> 4);
printf("%d",(data & 0x08) >> 3);
printf("%d",(data & 0x04) >> 2);
printf("%d",(data & 0x02) >> 1);
printf("%d",data & 0x01);
}

```

实验结果:

PORTA: 00000011

PORTB: 01010101

E:\TC\IO>

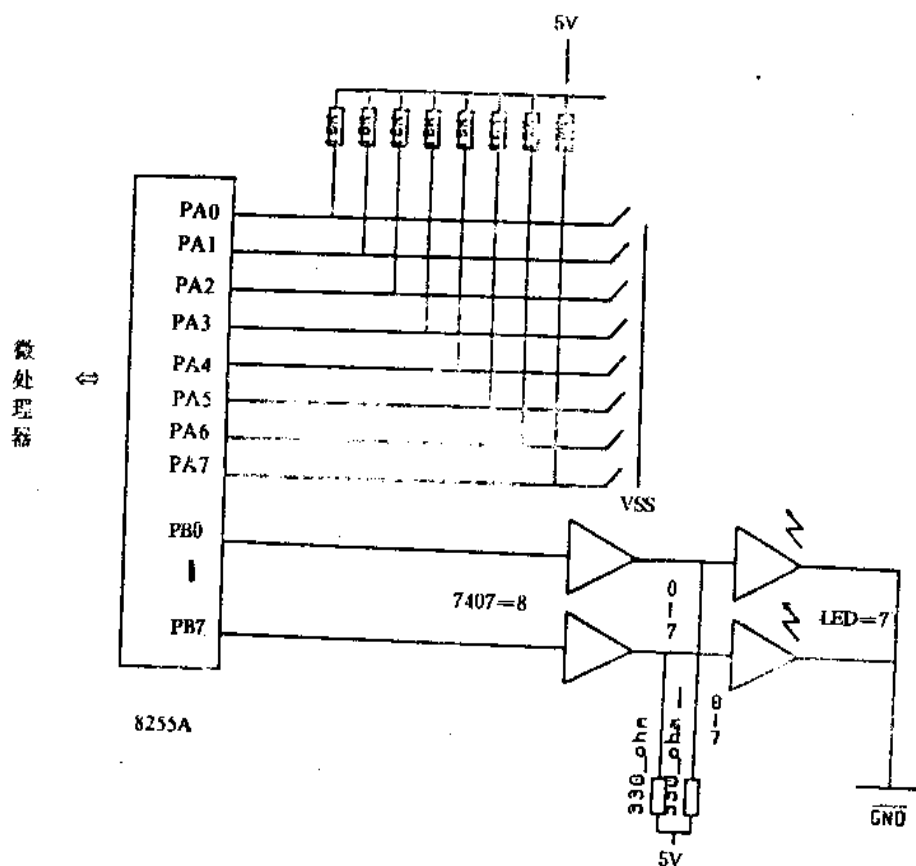
## 4.2 8255A 模式 0 的应用, PORTA

为输入型, PORTB 为输出型

工作目标:

利用 8255A 模式 0, 自 PORTA 输入数据, 于 PORTB 输出。

### 硬件设计:



**程序示例: ch4\_\_2.c**

从 PORTA 读取数据, 然后将所读取的数据送至 PORTB.

```

/*
/*          Program Name : cn4_2.c
/*  PORTA is input mode and PORTB is output mode
/*  For 8255, Mode 0 application
/*
#include <dos.h>
#define SETPORT      0x3c3
#define PORTA        0x3c0
#define PORTB        0x3c1
void main()
{
    unsigned char bytewrite, byteread;

/* set up PORTA for input and PORTB for output */
    bytewrite = 0x90;
    outportb(SETPORT,bytewrite);

```

```

/* get PORTA input */
byteread = inportb(PORTA);

/* send to PORTB */
bytewrite = byteread;
outportb(PORTB,bytewrite);
}

```

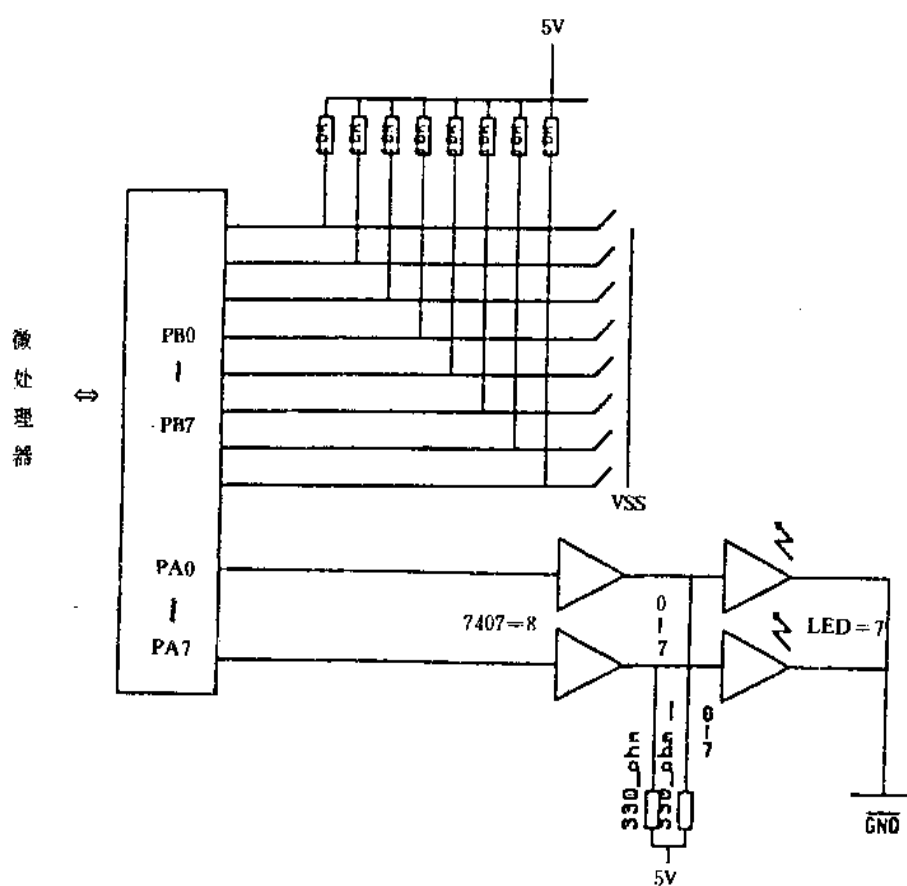
### 4.3 8255A 模式 0 的应用, PORTA

为输出型, PORTB 为输入型

工作目标:

利用 8255A 模式 0, 自 PORTB 输入数据, 于 PORTA 输出

硬件设计:





程序示例: ch4\_3.c

从 PORTB 读取数据, 然后将所读取的数据送至 PORTA.

```
/* ----- */
/*      Program Name : ch4_3.c      */
/*      PORTA is output mode and PORTB is input mode      */
/*      For 8255, Mode 0 application      */
/* ----- */
#include <dos.h>
#define SETPORT 0x3c3
#define PORTA 0x3c0
#define PORTB 0x3c1
void main()
{
    unsigned char bytewrite, byteread;

    /* set up PORTA for output and PORTB for input */
    bytewrite = 0x82;
    outportb(SETPORT,bytewrite);

    /* get PORTB input */
    byteread = inportb(PORTB);

    /* send to PORTA */
    bytewrite = byteread;
    outportb(PORTA,bytewrite);
}
```

#### 4.4 8255A 模式 0 的应用, PORTA 及 PORTB 均为输出型

工作目标:

利用 \*63\*H (或可由用户自定义) 输出至 PORTA 及 PORTB.

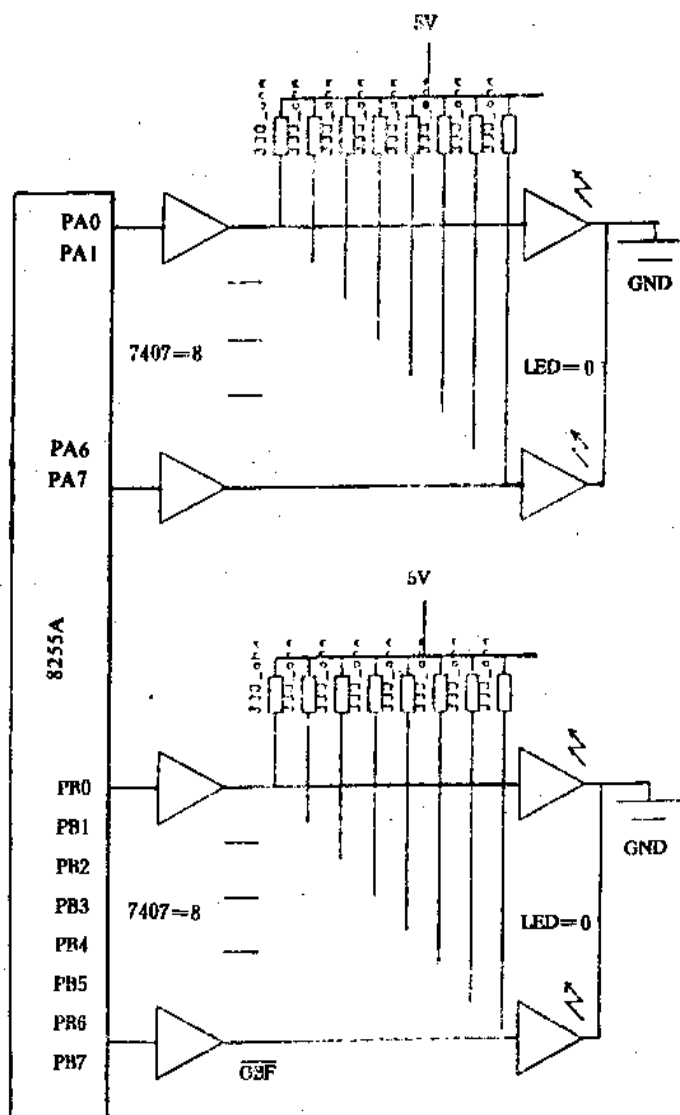
硬件设计:

程序示例: ch4\_4.c

将某个数据 (在第 10 行定义), 分别输出至 PORTA 和 PORTB.

```
/* ----- */
/*      Program Name : ch4_4.c      */
/*      Both PORTA and PORTB are output mode      */
/*      For 8255, Mode 0 application      */
/* ----- */
#include <dos.h>
#define SETPORT 0x3c3
#define PORTA 0x3c0
```

微处理器



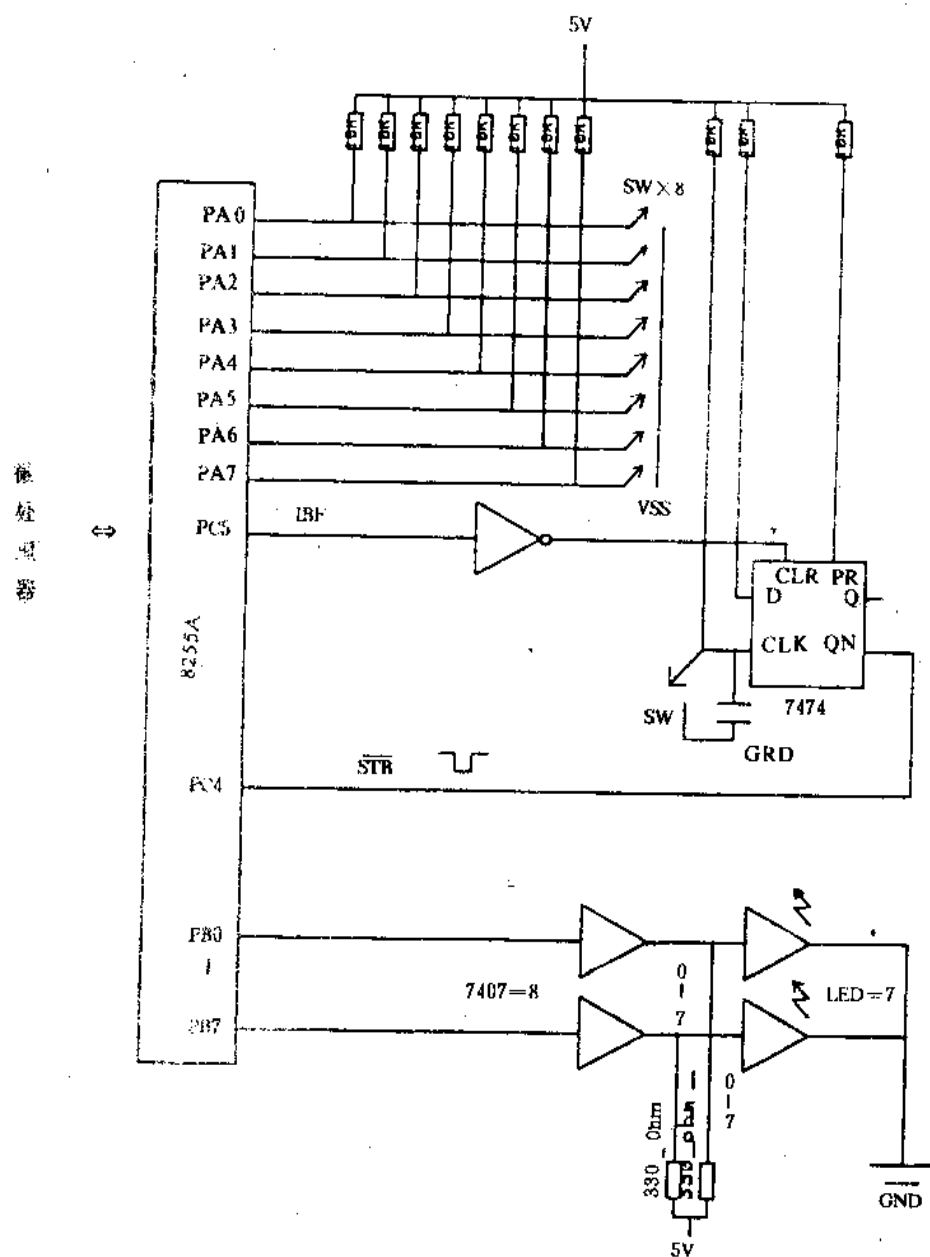
#### 4.5 8255A 模式 1 的应用, 模拟交互式数据输入

**工作目标:**


利用 8255A 第一种模式将数据由 PORTA 输入并在第 0 模式下将数据由 PORTB 输出。

本示例需要一个由外设产生的 $\overline{\text{STB}}$ 信号，我们以 7474 和一个 SW 开关来模拟产生。与 INTR 有关的交互式数据输出，请参阅第六章。

### 硬件设计:



PA0~PA7 为 8 个输入的开关, 平常接 10K $\Omega$  和 5V 相连 (即接上拉电阻)。

外部的 SW(SWITCH)由 7474D 型锁存器模拟。当 SW 由“L”到“H”的上升边缘将 D 输入锁存器, 此时 STB 转为“L”, 当 IBF 由“H”转为“L”时, D 型锁存器的数据被复位, QN 为“H”, STB 此时转为“H”, 恢复初始状态, STB 波形为  形。

PB0~PB7 为 8 个输出的端口位, 在此实验中, 将由 PA0~PA7 读入的数据输出至 PB0~PB7。


软件设计:

1. 设置控制字为第一种输入选通模式

```
SETPORT = 0x3e3; /* 8255A 的控制寄存器地址为 0x3e3 */
```

```
bytewrite = 0xb8;
```

```
outportb(SETPORT, bytewrite);
```

2. 手搬动 SW 产生  形, 用以模拟外部数据的控制信号。SW 在这个情况是模拟 STB, 表示外部数据已经准备就绪。此步骤与软件无关。

3. 软件此时检查 PC5, IBF (Input Buffer Full)。

4. 如果 IBF = 1, 表示第 2 步骤的 STB 已经被 8255A 收到, 因此产生 IBF = 1, 通知 I/O 外设。

5. 将 PORTA 读到的数据由 PORTB 输出。

程序示例: ch4\_5.c

8255A 交互式数据输入的模拟。

```
/* ----- */
/*          Program Name : ch4_5.c          */
/*      Basic handshake application.      */
/*      For 8255, Mode 1 application      */
/* ----- */

#include <dos.h>
#define SETPORT    0x3e3
#define PORTA      0x3e0
#define PORTB      0x3e1
#define PORTC      0x3e2

void main()
{
    unsigned char bytewrite, byteread, bytetest;

    /* set up control register */
    bytewrite = 0xb8;
    outportb(SETPORT, bytewrite);

    while ( !kbhit() )
    {
        byteread = inportb(PORTC);
        bytetest = byteread & 0x20;
```

```

if ( bytetest != 0x20 )      /* check PC5 */
    continue;

byteread = inportb(PORTA); /* get data from PORTA */
bytewrite = byteread;
outportb(PORTB,bytewrite); /* send data to PORTB */
}
}

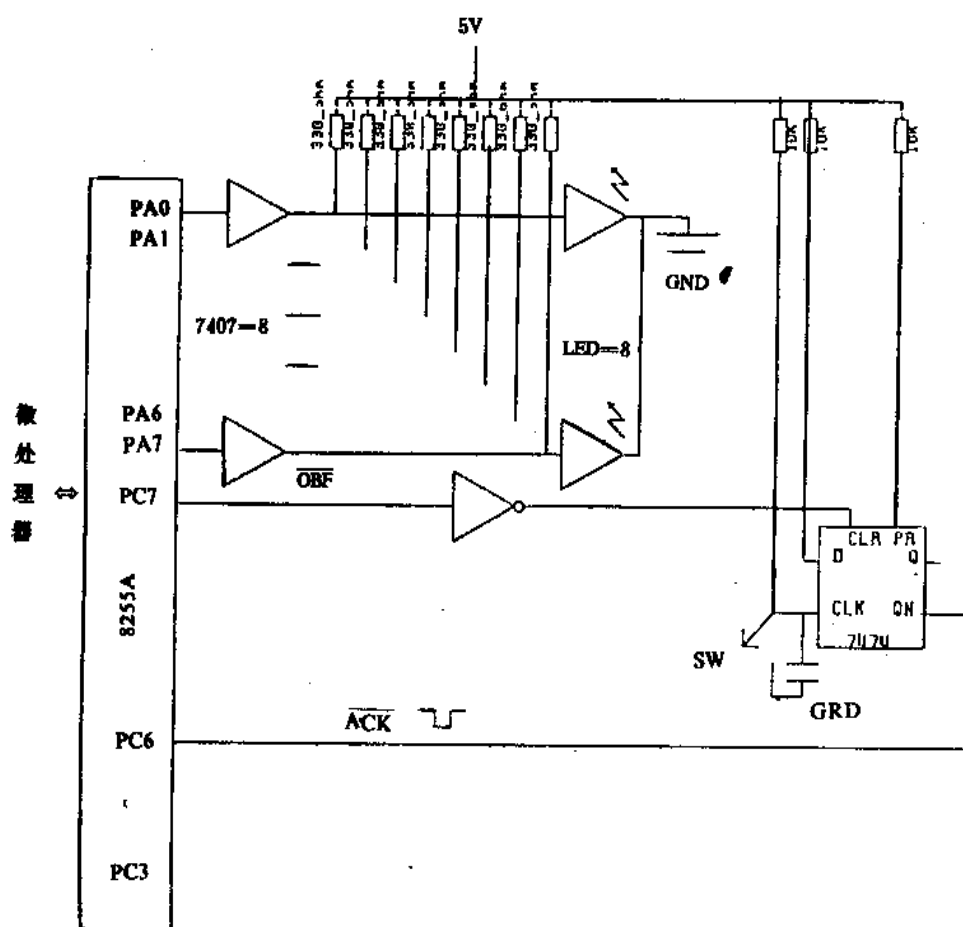
```

经过此实验，读者可以了解，在 8255A 模式 1 的输入模拟中最重要的就是由外界产生的  $\overline{STB}$  信号用以通知 8255A，外界外设数据要输入，以及由  $\overline{STB}$  激发的  $\overline{IBF}$  信号，用以通知外界，输入的数据已被接收，外界外设可以继续再送数据（如果有）。

这种交互式数据传送的缺点为，系统必须时时去检查 PC5，造成系统时间的浪费。请参阅第 6 章，系统的中断，另一种 8255A 的输入示例。

读者可以尝试不用 8255A 模式 1，而用模式 0 达到相同的工作目标。

#### 4.6 8255A 模式 1 的应用，模拟交互式数据输出



### 工作目标:

利用 8255A 第一种模式的输出类型, 将数据由 01 开始, 且每次增加 1, 由系统送到 PORTA.

本示例需要一个由外设产生的确认 (ACK) 信号, 通知微处理器送下一个数据。我们以外部 7474 和一个开关模拟产生。与中断有关的实验参照第六章。

硬件设计: 见上图

硬件说明:

PA0~PA7 为 8 个输出的位。

PC7 :  $\overline{\text{OBF}}$ (Output Buffer Full), 当输出数据送至 PA0~PA7 时,  $\overline{\text{OBF}}$  为 "L", 用以通知外部外设, 此时有数据输出, 同时 ACK 为 "H".

PC6 : 当 SW 将 ACK 由 "H" 转为 "L" 时, 表示数据已被外设取走了。

软件设计:

1. 设置控制字为第一种输出选通模式。

```
SETPORT = 0x3e3;
```

```
bytewrite = 0xa0;
```

```
outportb(SEPORT, bytewrite);
```

2. 系统将某数据 (暂定为 01H) 输出至 PortA, 且每次增加 1.

☆3. 硬件产生  $\overline{\text{OBF}}$ (Output Buffer Full), 读者可在 7474CLR 的标志灯上看出此信号已经准备就绪。

☆4. 读者拨动 SW 模拟外设产生的 ACK 确认信号。

☆5. 在 ACK 上升的边缘, 将产生中断 CPU 的动作。

6. 软件此时重复检查 PC6(ACK), 当 PC7( $\overline{\text{OBF}}$ ) 为高电平时, 表示先前送出的数据已被外界读取, 而且另一批新的数据已经送到了 PORTA 上了。

7. 屏幕上出现 "Continue?" Y (大写) 则继续, 若为其它值, 程序则结束。

☆: 表示为硬件动作。

程序示例: ch4\_6.c

8255A 交互式数据输出的模拟。

```
/* ----- */
/*          Program Name : ch4_6.c          */
/*      Basic handshake 2 application.      */
/*      For 8255, Mode 1 application        */
/* ----- */
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#define SETPORT    0x3e3
#define PORTA      0x3e0
#define PORTC      0x3e2
void main()
{
```

```

unsigned char bytewrite, byteread, bytetest;
char ch;

clrscr();          /* clear the screen */

/* set up control register */
bytewrite = 0xa0;
outportb(SETPORT,bytewrite);

bytewrite = 0x01;
outportb(PORTA,bytewrite);    /* output to PORTA */
while ( !kbhit() )
{
    gotoxy(33,10);
    printf("Waiting for OBF.");
    while ( 1 )
    {
        byteread = inportb(PORTC);
        sleep(1);    /* delay 1 second to read the PC7 bit */
        bytetest = byteread&0x80; /* check bit 7 */
        if ( bytetest != 0x80 )
            continue;
        else
            break;
    }
    bytewrite++;
    outportb(PORTA,bytewrite);
    gotoxy(33,12);
    printf("Do you like to continue?(y/n)");
    ch = getch();
    if ( ch != 'y' )
        break;
    clrscr();
}
}

```

从上面程序可知整个系统操作为：

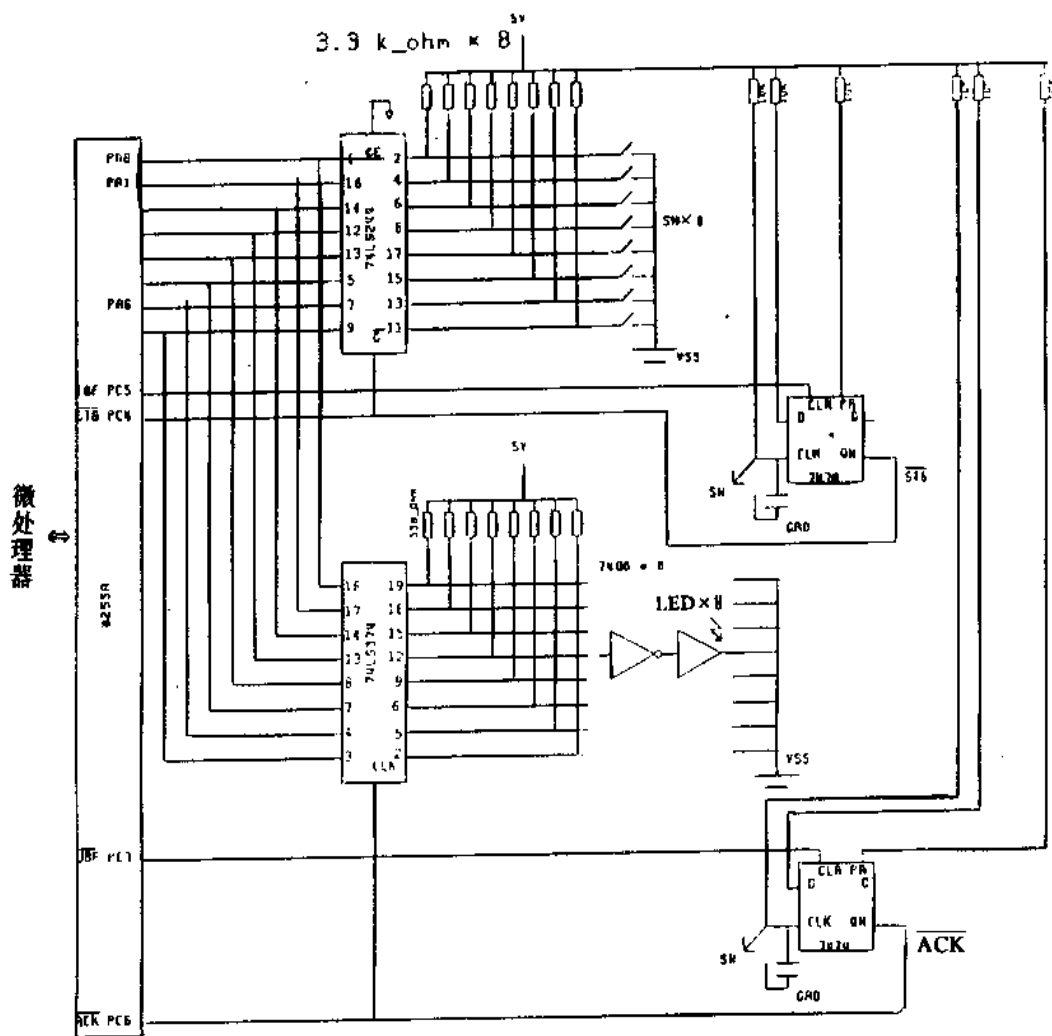
1. 系统输出数据于 PORTA，同时 8255A 将 PC7 定为“L”。
2. 外界外设以  $\overline{\text{ACK}}$  回应。
3. 当系统收到  $\overline{\text{ACK}}$  信号之后，准备下一批的数据再送出。

#### 4.7 8255A 模式 2 的应用, 双向式数据传输

### 工作目标:

利用 8255A 模式 2, 自 PORTA 输入数据, 再将它的补码输出于 PORTA。任意按一下键, 程序结束。这个示例不使用中断信号。

### 硬件设计:



### 原理说明:

8255A 在模式 2 时 A 口是双向选通的总线 I/O，只有 A 口才具有此功能。换言之，它可以当成输出，也可以当成输入。当系统对它运行输出的指令时，它即变为输出口，反之，当系统对它运行输入的指令，它又立刻为输入口。

在此模式下必须注意以下各个信号:

PC3 : INTR,用以中断系统,在本章中不使用,我们只以软件查询的方式处理。

PC6 :  $\overline{\text{ACK}}$ , 由外设产生的确认信号。

**PC4** :  $\overline{\text{STB}}$ , 由外设产生的输入控制信号。



PC5 : IBF, 由系统产生, 告诉外设输入缓冲区已满。

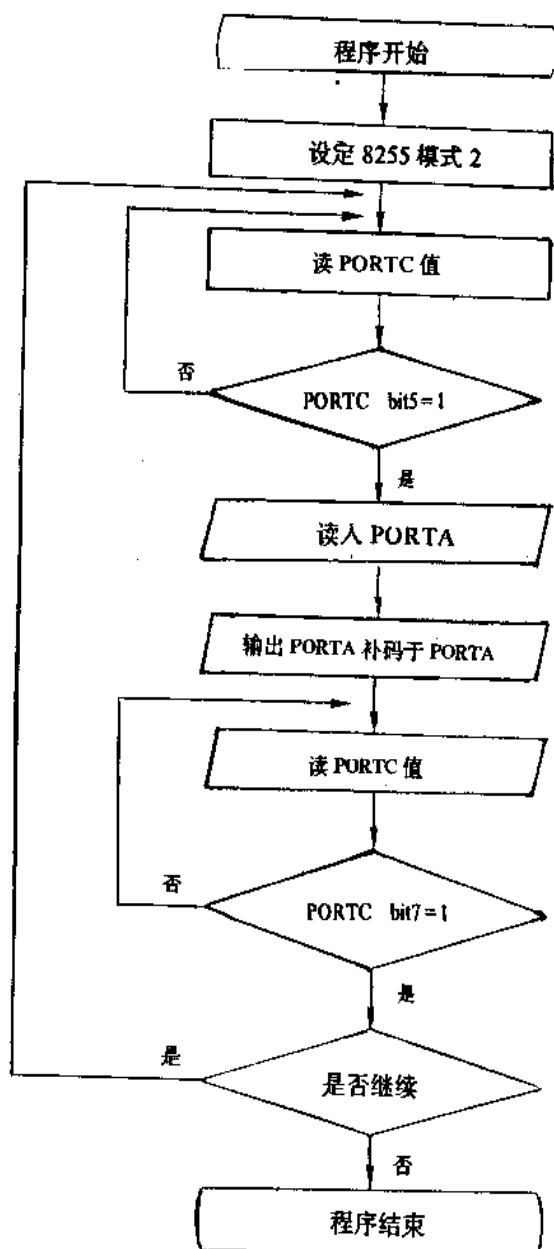
PC7 : OBF, 由系统产生, 告诉外设输出缓冲区已满。

硬件说明:

74LS244 可以控制数据流动的方向。当 $\overline{STB}$ 为低电平时,  $\overline{G}$ 也为低电压, 此时 SW 所选择的数据可以进入 8255A, 当 $\overline{G}$ 为高电压时, 由 8255A 送出的数据由左向右传送。在本例中对 SW 并无实质的影响。此时 74LS374 即可将 8255A 送出的数据送给 LED 显示器。

8255A 送出的数据送给 LED 显示器。

在 LED 前的缓冲区为负向时, LED 所显示的数据与 8255A 送出的相反, 即 8255A 送出“0”时, LED 会亮。



### 工作情形:

1. 由 A 口的 8 个小开关 SW×8 选择要输入系统的数据。
2. 手动STB开关, 模拟 A 口将数据输入。
3. 系统利用STB将数据读进输入锁存器, 并将 IBF(PC5)设为低电平。
4. 手动ACK开关, 当ACK为低电平时, 系统将数据送给 PORTA。
5. 系统查询是否继续。
6. 如果要继续, 回到 1。

由于不使用中断信号, 所以系统必须利用软件, 随时检查 PC5 及 PC7。

软件编程: 见 59 页流程图

程序示例: ch4\_7.c

```
/* ----- */
/*          Program Name : ch4_7.c          */
/*      Bidirection handshakes application      */
/*      For 8255, Mode 2 application          */
/* ----- */
#include <dos.h>
#include <conio.h>
#define SETPORT 0x3e3
#define PORTA 0x3e0
#define PORTC 0x3e2
void main()
{
    unsigned char bytewrite, byteread;

    do
    {
        clrscr();
        gotoxy(30,9);
        printf("Waiting for data. Press STB");
    /* save the original interrupt mask status */
        bytewrite = 0x0c;
        outportb(SETPORT,bytewrite);

    /* read the PORTA */
        byteread = inportb(PORTA);

    /* read the PORTC and check bit 5 */
        do
        {
            byteread = inportb(PORTC);
            byteread &= 0x20;
        } while ( byteread != 0x20 );    /* if bit 5 != 1, read again */
    }
```

```

/* read the PORTA, do the complement and output to PORTA */
    bytread = inportb(PORTA);
    bytewrite = ~bytread;
    outportb(PORTA,bytewrite);

/* read the PORTC and check bit 7 */
    do
    {
        bytread = inportb(PORTC);
        bytread &= 0x80;
    } while ( bytread != 0x80 );      /* if bit 7 != 1, read again */

    gotoxy(23,10);
    printf("Press ACK. Finally, do you like to continue?(y/n)");
    } while ( getch() == 'y' );
}

```

#### 思考:

这个示例只是第二种工作模式的简单示例。它并不表示一定要经过 A 口输入数据,并且立即输出数据。它告诉读者模式 2 是输出及输入功能的组合。

当它当成输入时,由外设产生的 $\overline{STB}$ 信号会将数据输入,8255A 以 $\overline{IBF}$ 应答。

当它当成输出时,由外设产生的 $\overline{ACK}$ 信号告诉 8255A,外设已经收到数据了。8255A 以 $\overline{OBF}$ 应答。

哪一种外设应用必须使用这种模式?

## 4.8 思考题

1. 修改 4-1, 将所得的 PORTA, PORTB 数据以十六进制方式打印出。
2. 修改 4-2, 将所得数据也在计算机屏幕上打印出。
3. 修改 4-3, 将所得数据以十六进制方式在计算机屏幕上打印出。
4. 修改 4-4, 让程序将 0-FF 输出至 PORTA, 另它的互补值输出至 PORTB。按任何键则程序结束。
5. 研究 4-5, 这个程序如何结束? 请修改程序, 让它在用户按下“Q”键后程序自动结束。
6. 您认为 4-6 是很好的程序吗? 将它的缺点列举出来? 应如何改进? 这种数据传输的好处在哪里? 坏处呢?
7. 修改 4-7, 将一连串的数据由 PORTA 输入, 并在屏幕上显现出来。按下任何键, 则程序停止。
8. 修改 4-7, 将一连串的数据由 PORTA 输出, 并同时在屏幕上也显现出来。按下任何键, 则程序停止。
9. 将问题 7、8 改成由第 0 种模式运行。

## 第五章 系统中断

### 本章学习目的

1. 读者可以通过本章了解 IBM PC 的中断。
2. 8259A 是 80286 系统使用的中断器，读者必须详细研读。
3. 82380 为 8259A 的替代 IC，通过本章，也可使读者了解未来的 IC 工业设计方向。
4. 为研究下一章的实例做准备。

### 本章内容

- 5.1 IBM PC 的中断
- 5.2 INTR 中断
- 5.3 8259A 中断控制器
- 5.4 如何读取 8259A 的状态 IRR, ISR 及 IMR
- 5.5 串连模式
- 5.6 82380 简介
- 5.7 思考题

微型计算机系统由一些不同的计算机外设和微处理器所组成。它们之间的配合是系统工程师所最关注的问题。微处理器是整个系统的核心，所有的外设都必须接受它的命令，分别或同时运行微处理器所分配的工作。

当这些外设需要服务时，通常有两种方式通知微处理器。

1. 等待微处理器轮询 (Polling)
2. 中断 (Interrupt)

以上两种方式，分别介绍如下：

第一种方式，微处理器每隔一段时间，就必须检查各个外设，查询是否需要服务，如果是，则停住主程序，处理外设的要求。如果不是，则继续运行主程序。这种不断的查询，浪费了微处理器的工作时间 (CPU Time)。而当外设真正要求服务时，微处理器可能正忙于其它操作，而忽略了外设。

第二种方式，当任何外设需要服务时，立即向微处理器请求中断，微处理器会依当时操作的情况判断，是否应立即提供服务，或者让它稍候。在运行外设中断的同时，如果有另一个外设要求服务时，微处理器会按我们所编程的系统中断优先次序决定，是否要立即处理突发的第二级中断。

任何中断请求会迫使微处理器运行一段系统程序。这个系统程序的起始位置，即被称为中断向量 (interrupt vector)。表 5.1 为 IBM PC 80386 中断向量使用情形。表 5.2 为 IBM PC 中断向量使用的情形。

表 5.1 IBM PC 80386 中断向量使用

口地址	访 问	寄存器说明
20H	写入	B 层的 ICW1, OCW2 或 OCW2
	读取	B 层的采样、请求、处理中状态寄存器
21H	写入	B 层的 ICW2, ICW3, ICW4, OCW1
	读取	B 层的屏蔽寄存器
22H	读取	B 层的 ICW2
28H	可读/写	IRQ8 向量寄存器
29H	可读/写	IRQ9 向量寄存器
2AH	可读/写	保留
2BH	可读/写	IRQ11 向量寄存器
2CH	可读/写	IRQ12 向量寄存器
2DH	可读/写	IRQ13 向量寄存器
2EH	可读/写	IRQ14 向量寄存器
2FH	可读/写	IRQ15 向量寄存器
A0H	写入	C 层的 ICW1, OCW2 或 OCW3
	读取	C 层的采样、请求或处理中状态寄存器
A1H	写入	C 层的 ICW2, ICW3, ICW4, OCW1
	读取	C 层的屏蔽寄存器
A2H	读取	C 层的 ICW2
A8H	可读/写	IRQ16 向量寄存器
A9H	可读/写	IRQ17 向量寄存器
AAH	可读/写	IRQ18 向量寄存器
ABH	可读/写	IRQ19 向量寄存器
ACH	可读/写	IRQ20 向量寄存器
ADH	可读/写	IRQ21 向量寄存器
AEH	可读/写	IRQ22 向量寄存器
AFH	可读/写	IRQ23 向量寄存器
30H	写入	A 层的 ICW1, OCW2 或 OCW3
	读取	A 层的采样、请求、处理中状态寄存器
31H	写入	A 层的 ICW2, ICW3, ICW4, OCW1
	读取	A 层的屏蔽寄存器
32H	读取	A 层的 ICW2
38H	可读/写	IRQ0 向量寄存器
39H	可读/写	IRQ1 向量寄存器
3AH	可读/写	IRQ2 向量寄存器
3BH	可读/写	IRQ3 向量寄存器
3CH	可读/写	IRQ4 向量寄存器
3DH	可读/写	保留
3EH	可读/写	保留
3FH	可读/写	IRQ7 向量寄存器

表 5.2 IBM PC 中断向量使用

03FCH		中断状态	255	
⋮	⋮		⋮	详细使用情形可参考
⋮	⋮		⋮	← BIOS 方面的书籍
⋮	⋮		⋮	
0040H			16	
003CH	IRQ7		15	
0038H	IRQ6		14	
0034H	IRQ5		13	
0030H	IRQ4		12	主机板上 8259A 中断控制器,
002CH	IRQ3		11	← 在 80386 / 80486 中为 82380
0028H	IRQ2		10	中断控制器
0024H	IRQ1		9	
0020H	IRQ0		8	
001CH	系统保留		7	
0018H	系统保留		6	
0014H	打印屏幕内容		5	
0010H	溢位指令 INTO		4	
000CH	中断点		3	
0008H	不可屏蔽的中断		2	特殊中断向量
0004H	单步执行		1	← 使用者无法更改
0000H	除数是零或除法溢出		0	

每一种中断会引发一组子程序的运行。例如，当除法错误时，系统会运行 0000H 的子程序。本章讨论的重点在于 8259A 的中断情形及其详细编程操作，以使读者充分掌握 8259A 的细节。

## 5.1 IBM PC 的中断

一般而言，中断可以分为两种情形，如硬件中断及软件中断。下图 5.1 为 IBM PC 的中断说明。

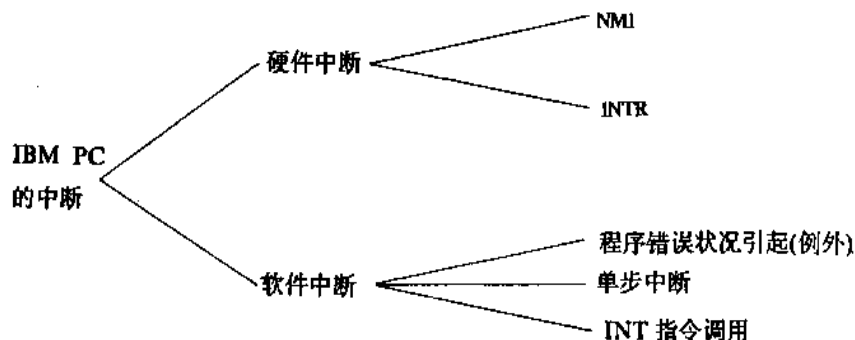


图 5.1 IBM PC 的中断

图 5.1 说明:

### • 硬件中断

—NMI(Non Maskable Interrupt)不可屏蔽式

这种中断为紧急形的，系统软件不能影响这种中断，例如在电力不足时，PC 系统可以紧急输出一个信号给电力系统，打开电力中继器，立即补充电力以免数据遗失。

—INTR 中断(Interrupt Request)

这种信号一般由外设系统产生，系统的软件可以加以管理。举例来说，82C54 的时间中断，82C54 每隔一段时间即对系统中断，当这些时间累积至 1 秒时，系统时间即增加 1 秒。

—程序错误

当作除法运算时，如果除数为 0，立即产生溢出(Overflow)的现象，通知微处理器。一般的系统工程师在处理这一类情形时，会令微处理器停住，当然我们也可以利用软件技巧，在发生这一类情形时，运行其它的工作，例如请微处理器显示 Overflow 的信息。

—单步中断

单步中断的目的是提供用户以逐步监视程序运行的情形。微处理器的作法如下：

1. 每运行一个指令后产生中断。
2. 运行中断的向量程序 0004H。
3. 0004H 会调用一个子程序，这个子程序要求程序运行暂停，相关寄存器数据继续保存。
4. 等待下一个单步运行指示，继续运行未完成的程序。

—执行 INT 指令

运行 INT 指令时，系统会立即中断，这种中断方式的最大特色在于它的中断可由用户编程。以图 5.2 为例。

当微处理器正运行某一个程序时，突然间来了一个中断请求，微处理器立即进行处

理，也就是运行这个中断的服务程序。处理完毕后，又回头继续运行原来的主程序。

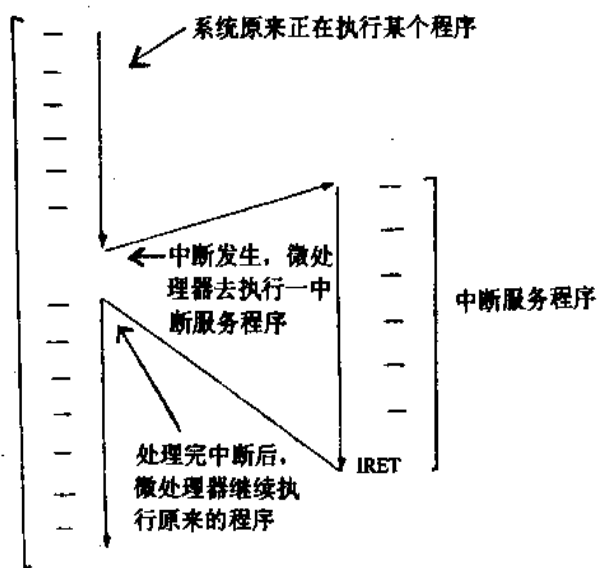


图 5.2 中断处理的方式

## 5.2 INTR 中断

本书的讨论重点在于 INTR 的中断。这一类的中断必须配合软件和硬件的编程。唯有完整的编程才能在不影响原有的操作系统之下，运行我们所需要的中断。

这一类的中断是可屏蔽的，也就是说，它的中断请求可以被系统拒绝。系统如何拒绝它的中断请求呢？在微处理器中有一个寄存器 IF(Interrupt Flag)，当它被设置为 0 时，所有的 INTR 中断都会被忽略。

以 IBM PC XT 系统为例，表 5.2 为它所具有的 INTR 中断。IRQ0~IRQ7 由系统的 8259A 提供。我们将在下一节中详细介绍 8259A 的功能。IRQ0, IRQ1, IRQ4, IRQ7 已经被系统使用了。读者在设计适配器时，须特别加以注意。

表 5.3 IBM PC 中断表

优先权	用法说明
最高优先权 NMI	主机板上 RAM 的奇偶校验位, I/O 通道检查
可屏蔽式 IRQ0	系统计时器 82C54 通道 0 的输出中断
可屏蔽式 IRQ1	键盘扫描码的中断
可屏蔽式 IRQ2	保留未使用
可屏蔽式 IRQ3	保留未使用
可屏蔽式 IRQ4	供 RS-232-C 中断使用
可屏蔽式 IRQ5	保留未使用
可屏蔽式 IRQ6	供驱动器状态中断使用
可屏蔽式 IRQ7	供并行打印机口中断使用



IBM PC AT 或 80386 系统及其它 PS-2 系列计算机与 PC XT 稍有不同, 详细的不同之处, 请读者参考使用手册。基本上这些软件可以相互通用, 所以它的差别应该不大。

表 5.3 为 IBM PC 的中断请求。在设计适配器时必须特别注意, 不是所有中断都可以使用, 因为部分中断请求已被系统本身的 I/O 及连接系统总线的适配器占据而无法使用。IRQ2, IRQ3, IRQ5 目前尚未使用。

### 5.3 8259A 中断控制器

8259A 是一个微处理器的中断控制器。它可以同时处理 8 个不同来源的中断请求。也可彼此串连 8 个 8259A, 同时处理 64 组不同的中断来源。8259A 此时可以被编程为扮演仲裁的角色, 决定哪一个中断请求可以使用 INTR 输入引脚, 用作中断微处理器。

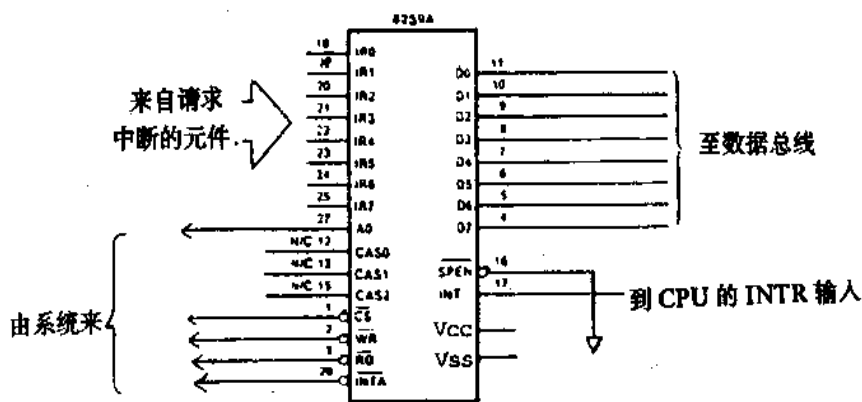


图 5.3 8259A 引脚图

图 5.3 为最基本的中断线路图, 8259A 同时可以接受不同来源的 8 个中断请求。CAS0~CAS1 在 8259A 串连时使用。

#### 5.3.1 引脚说明

- $V_{cc}$ : 5V 电源
- $V_{ss}$ : 地
- $\overline{CS}(I)$ : (Chip Select) 芯片选择。当  $\overline{CS}=0$  时表示芯片被启动了。
- $\overline{WR}(I)$ : (Write) 写。微处理器利用此信号将数据写入 8259A。
- $\overline{RD}(I)$ : (READ) 读。微处理器利用此信号将数据从 8259A 中读回系统。
- D7~D0(I/O): 双向数据总线, 可传送控制、中断、状态等信息于系统、外设和 8259A 三者之间。
- CAS0~CAS2: 串联线, 利用此信号可将其他 8259A 串联在一起。
- (I/O)
- $\overline{SP}/\overline{EN}(I/O)$ : Slave program / Enable buffer 这个信号有两个功能:
  - ① 在缓冲模式下工作时(Buffer Mode)当成控制  $\overline{EN}$  的输出。
  - ② 不在缓冲模式下时, 当成决定本身是主或从的决定信号。

SP=1 为主

• SP=0 为从

INT(O) : Interrupt 产生向微处理器中断的请求信号。

IR0~IR7(I) : Interrupt Request 由外设向8259A申请中断的请求信号。

INTA(I) : Interrupt Acknowledge 中断的确认。当微处理器接受了中断请求后向 8259A 发出此信号，表示“知道了”，以通知 8259A 继续运行其他操作。

AO(I) : 由微处理器送过来的地址线。

注：在所有的控制信号上加  $\overline{\phantom{x}}$ ，表示低电位有效。如  $\overline{\text{RD}}$ 、 $\overline{\text{CS}}$ 。

### 5.3.2 8259A 的内部构造

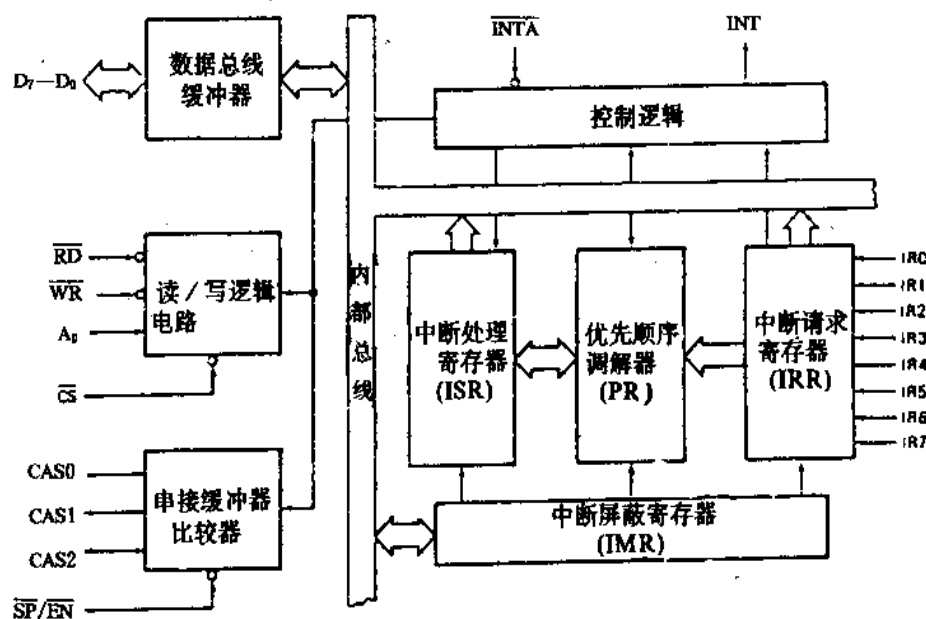


图 5.4 8259A 的内部构造方块图

8259A 内部结构主要分为四个部分，如图 5.4。

-中断请求寄存器(IRR)

-优先权调解器(Priority Resolver)

-中断处理寄存器(ISR)

-中断屏蔽寄存器(IMR)

以下分别加以说明。

**中断请求寄存器(Interrupt Request Register) (IRR)**

目的：用以记录哪些中断要求服务

IRR:

IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
-----	-----	-----	-----	-----	-----	-----	-----

例如：当外界外设IR4提出中断请求时，在这个寄存器内的IR4被设置为1。

**中断屏蔽寄存器(Interrupt Mask Register) (IMR)**

目的：用以记录哪些中断被禁止(disable)

IMR:

IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
-----	-----	-----	-----	-----	-----	-----	-----

IM7~IM0 对应于 IR7~IR0 的中断请求。若 IM4=1，即表示 IR4 被禁止，这样就不能产生中断的请求。

**中断处理寄存器(In Service Register)**

目的：用以记录哪一个中断输入正被服务，同时将中断请求相对应的IR位置复位为0。

ISR:

IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
-----	-----	-----	-----	-----	-----	-----	-----

例如：当 IR4 请求第一次中断时，8259A 会运行下列操作。

1. 将 IR4 定为 1；

2. 检查 IM4:

a. 若 IM4=0，则 IS4 为 1，同时将 IR4 复位为 0。

b. 若 IM4=1，则中断申请无效。

**优先调解器(Priority resolver)**

用以决定中断是否必须立即运行，若 IR4 要求中断，而先前已有更低优先的中断正被处理，例如 IR6（此时 IR6=1，因它正接受服务），此时 IS6 会立即被复位为 0。IS4 则为 1。通常中断服务程序内均含 STI 指令，将可再度启动中断系统。

以下两示例说明优先调解器处理中断的情形。

在主程序运行一个程序期间，来了两种不同的中断请求，以 A、B 说明如下：

A: 此时 IR1, IR4 被屏蔽，即被禁止。

B: 此时 IR1, IR4 未被屏蔽。

运行结果，如图 5.5。

A: 不运行中断。

B: 先运行 IR4 中断，再运行 IR1 中断。

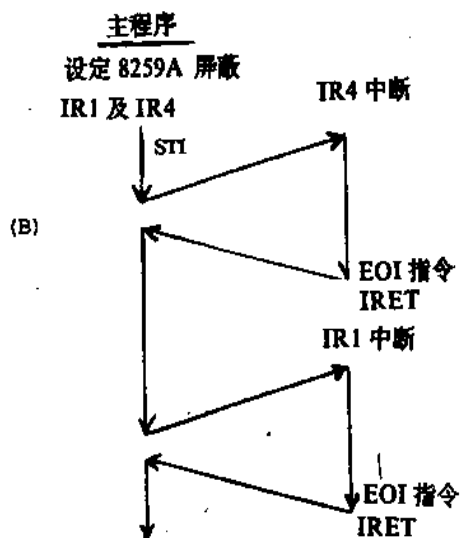
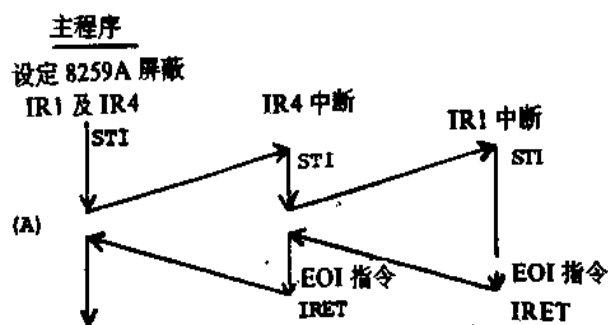
### 5.3.3 8259A 初始状态的设置

在 8259A 命令中，若 A0 为 0，D4 为 1 时，即被当成 ICW1 的初始命令。初始命令会产生下列动作。

- 中断输入 IR0~IR7 的触发方式被设置为低至高 (Low to high) 电平的边沿触发。

- IM0~IM7 被清除。

- 1-10-68



- (a) INTR 禁止时, 运行 IR4 处理例程的反应。  
(b) INTR 禁止时, 运行 IR4 处理例程的反应。

图 5.5 IR1 紧接在 IR4 后到来时, 8259A 及 CPU 的反应流程

### 5.3.4 初设命令字符组

8259A 主要是利用四个寄存器 ICW1 至 ICW4 编程出各种不同的功能。它们分别说明如下:

## 1. ICW1

**ICW1 有三个主要功能:**

- 选取 IRQ 输入的触发类型 (边缘或电平)。
- 显示中断是单独使用或处于串联形式。若是串联形式, 则中断逻辑会接受 ICW3 作

为串联形式的程序编程。不然，中断逻辑即不须接受 ICW3。

—决定是否 ICW4 会被送出。

## 2. ICW2

用以编程对应的向量寄存器，或以之作为指示器。每一个 ICW2 寄存器均分别使用不同的地址进行读取或写入操作。

## 3. ICW3

若编程为外部串联形式时，则中断逻辑将只接受一个 ICW3。ICW3 用于在串联时达成明确的编程方式，其位可显示那一中断请求输入有串联—从性控制器。这种串联方式会影响中断响应周期时中断向量的产生。

## 4. ICW4

ICW4 只有在 ICW1 的 D0=1 时才会起作用。这个命令组寄存器有两个功能：

—选择自动 EOI 模式或软件 EOI 模式。

—选择嵌套类型是否要与串联类型合用。

以下分别讨论 ICW1~ICW4 中各位所代表的意义。

ICW1:

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	A7	A6	A5	1	LTIM	ADI	SGNL	IC4

A7,A6,A5 用以设置中断向量。MCS 80/85 适用。

LTIM 用以决定中断触发方式。

当 LTIM=0 表示边缘触发(Edge trigger)

当 LTIM=1 表示电平触发(Level trigger)

ADI: 用以说明调用地址区间间隔，当 ADI=0 时表示区间间隔为 4。亦即 IR0~IR7 调用服务程序向量间隔为 4 个字节，当 ADI=1 时，则为 8 个字节。

这个位只适合 8 位的微处理器，在 8088 以后的微处理器已不适用。

SGNL: 用以说明系统中是否使用串联的中断结构。

SGNL=1 是

SGNL=0 否

IC4: 用以说明是否需要 ICW4 寄存器中的说明。

ICW4=1 是

ICW4=0 否

ICW2:

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	A15 T7	A14 T6	A13 T5	A12 T4	A11 T3	A10	A9	A8

A15~A8 表示中断向量 (MC80/85适用);

T7~T3 表示中断向量 (8086/8088系列适用);

ICW3: 这个字节可分为两种情形讨论:

A: 当成主元件时

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	S7	S6	S5	S4	S3	S2	S1	S0

每一个 8259A 的每一个 IR 输入可以再串联 8 个 8259A。这个字节中的每一个位说明相对应的 IR 是否接有从元件。SX=1 表示有从元件, SX=0 表示没有。

IRX 对应 SX (X=0~7), 即 S0~S7。

如果 S4=1, 表示 IR4 另接了一个从元件。

B: 当成从元件时, 用以决定从地址

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	ID2	ID1	ID0

从地址	0	1	2	3	4	5	6	7
D2	0	0	0	0	1	1	1	1
D1	0	0	1	1	0	0	1	1
D0	0	1	0	1	0	1	0	1

如前所述, 每一个 IR (中断信号) 可以连接 8 个从元件, 每一个从元件以这个字节的 D0, D1, D2 决定自己的地址。

ICW4 :

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	SFNM	BUF	M/S	AE01	μPM

SFNM: 特殊全嵌套类型吗? 参考下节说明

SFNM=1 是

SFNM=0 否

BUF	M/S	
0	X	非缓冲型
1	0	缓冲型 / 从元件
1	1	缓冲型 / 主元件

对于缓冲型的说明, 参考下节

AEOI: 自动 EOI 吗?

AEOI=1 是

EOI=0 否

$\mu$ PM: 何种微处理器?

$\mu$ PM=1 8086/8088 型

$\mu$ PM=0 MCS80/85 型

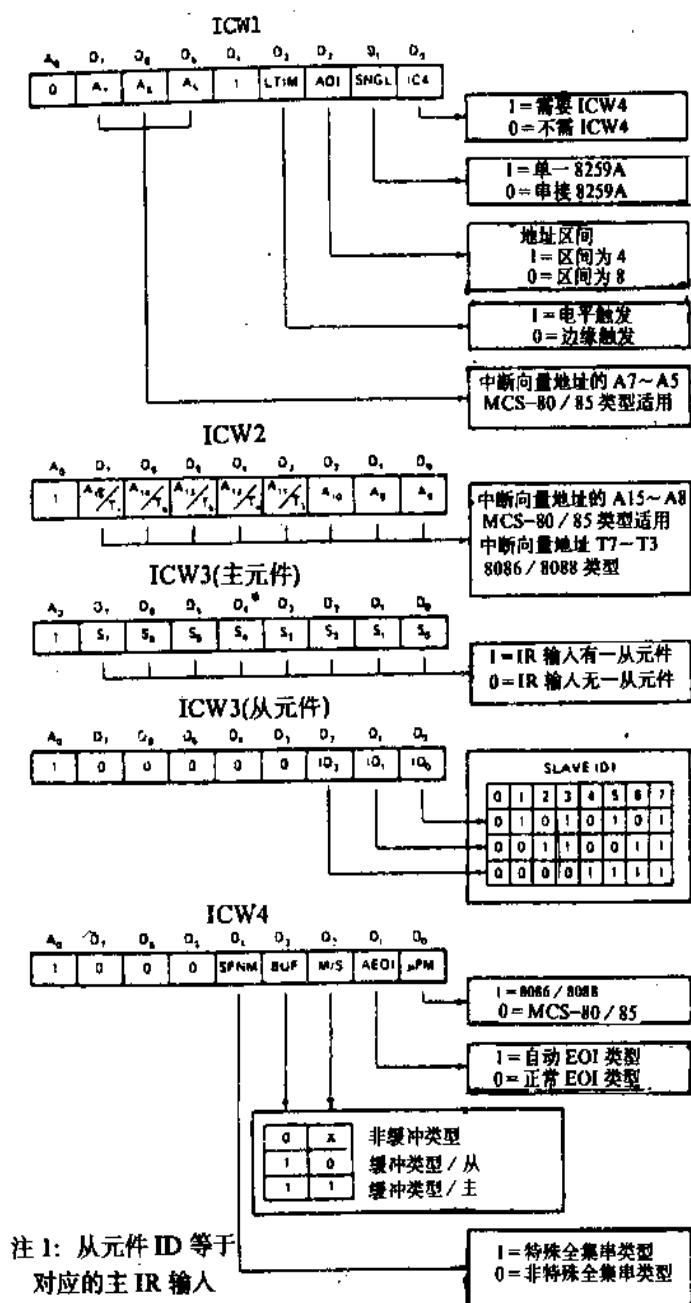


图 5.6 8259A 初值设置命令组的格式及送出顺序

初始命令字符组整理如上图 5.6 所示。

初始命令字符组的初设顺序如下图所示：

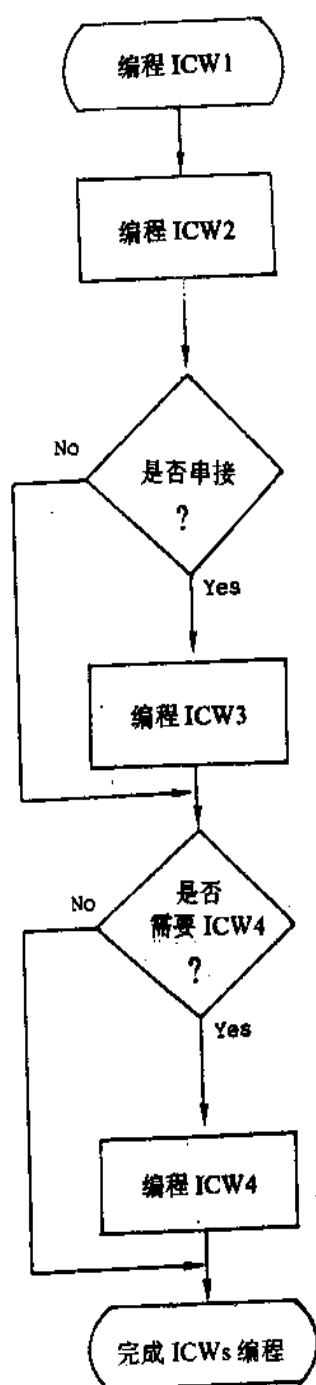


图 5.7 ICWs 编程的顺序

### 5.3.5 操作命令字(Operation Command Words, OCWS)

当我们设置了 ICWS 之后，8259A 就已经被编程完成了，准备接受其它操作命令以



进行操作。本节所讨论的就是如何下达工作指令。操作命令字共可分为3个，OCW1，OCW2及OCW3。现分别说明如下：

OCWS: 所控制的类型及操作包含

- 嵌套类型
- 循环优先权模式
- 特殊屏蔽类型
- 采样类型
- EOI指令
- 读取状态命令

OCW1: OCW1只用在屏蔽操作，它与中断屏蔽寄存器(IMR)有关。系统可以编程此字节，以将中断输入使能或禁止。

OCW2: OCW2有下列用途:

- 记录进行复位某一特别ISR位或设置一明确优先权的中断顺序。此功能可被允许或禁止。
- 选择EOI指令，即选择明确或不明确的EOI。
- 将下列其中一种优先权循环操作使能
  - a. 不明确EOI循环
  - b. 自动EOI循环
  - c. 明确EOI循环

OCW3: OCW3控制三种主要的操作类型:

- 选择及运行读取状态寄存器指令，即读取IRR或ISR。
- 送出采样指令。
- 设置或清除特殊屏蔽类型。

OCW1~OCW3 的各位说明如下:

OCW1: 用以设置或清除中断屏蔽寄存器(Interrupt Mask Register)(IMR)

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M7	M6	M5	M4	M3	M2	M1	M0

M7~M0 表示8个屏蔽位

M=1 表示禁止中断通道

M=0 表示允许中断通道

OCW2: 用以循环，改变中断信号的优先权。

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	R	SL	EOI	0	0	L2	L1	L0

R	SL	EOI	
0	0	1	不明确 EOI 指令
0	1	1	明确 EOI 指令
1	0	1	不明确 EOI 指令时的循环
1	0	0	自动 EOI 循环 (设置)
0	0	0	自动 EOI 循环 (清除)
1	1	1	明确 EOI 循环 (须使用 L2-L0)
1	1	0	设置优先权 (须使用 L2-L0)
0	1	0	无作用

OCW3 :用以允许屏蔽的模式

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	0	ESMM	SMM	0	1	P	RR	RIS

ESMM: 允许特殊屏蔽模式 (Enable Special Mode)

= 1, SMM设置特殊屏蔽

= 0, 无作用

SMM: 特殊屏蔽模式

当ESMM = 1  
SMM = 1 } 8259A进入特殊屏蔽模式

当ESMM = 1  
SMM = 0 } 8259A返回正常屏蔽模式

8259A 操作命令字整理如下图 5.8 所示。

### 5.3.6 名词解释

#### 5.3.6.1 完全嵌套模式

在系统开机后 8259A 即自动进入这一个模式。倘若此时有中断发生，以下为它的处理情形。

1. 开机后，中断顺序自动定为 IR0 最高，IR7 最低。
2. 此时有一中断 INTR 要求中断。
3. 系统运行中断请求，并送出响应信号 (ACK) 给 8259A。
4. 中断处理寄存器IS被设置。此时任何相同或较低优先权的中断要求全被禁止。
5. 当系统处理完中断服务子程序之后，立即送出“中断完毕”的信号给8259A。IS即被复位。

#### 5.3.6.2 EOI(End of Interrupt)

8259A 的 IS(In Service)位在中断开始时会被设置为 1。如何被复位呢？有 2 种方式可以将它复位。

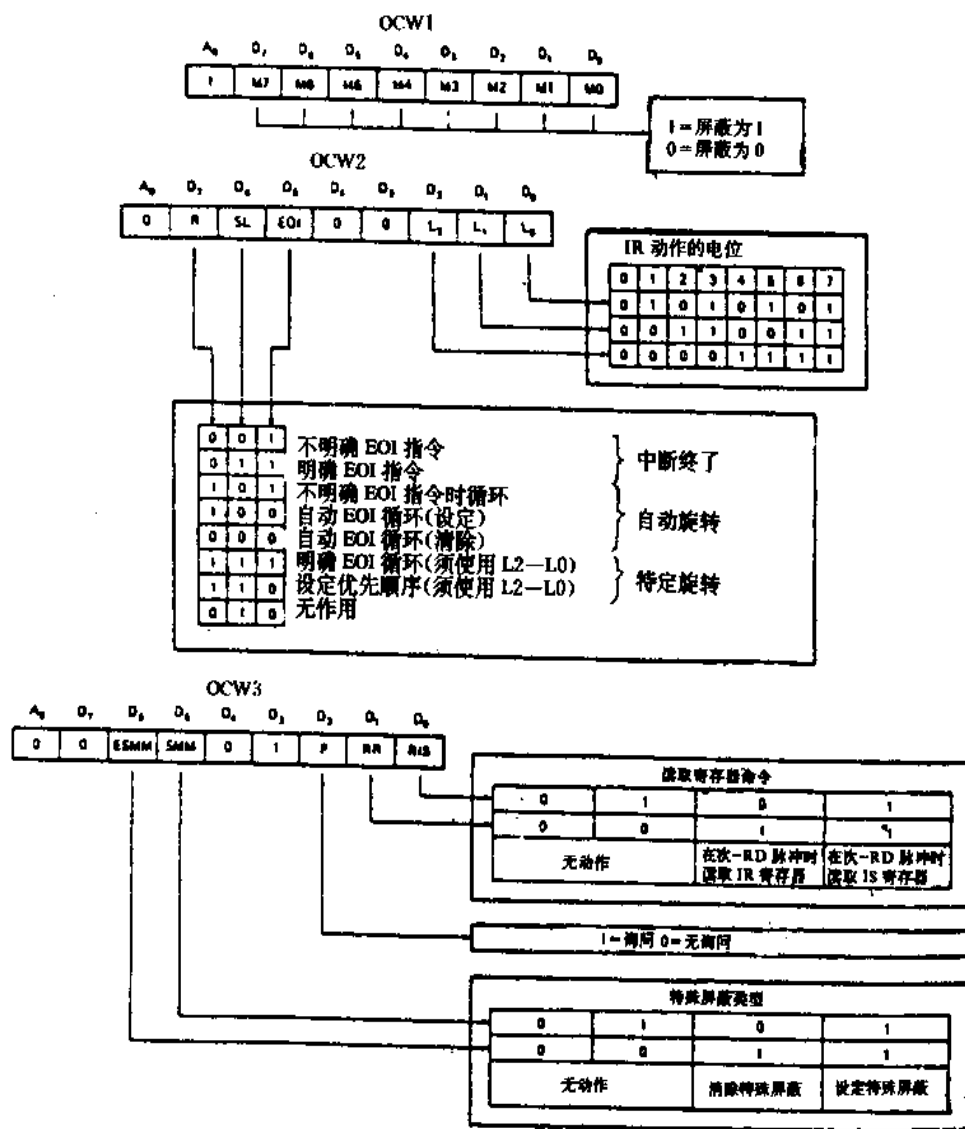


图 5.8 8259A 的操作命令字

1. 当 AEOI = 1 时 (位于 ICW4 中), 微处理器送出 INTR 信号给 8259A, 8259A 据此将 IS 复位。
2. 利用 EOI 指令, 在返回主程序之前, 系统必须送出 EOI 指令给 8259A, 通知它把 IS 复位, EOI 指令有两种, 一种为不明确的中断结束 (Non Specific EOI Command), 另一种为明确的中断结束 (Specific EOI Command)。

当 8259A 在嵌套模式下工作时, 不明确的 EOI 指令会自动清除最高优先的 IS 位, 因为在嵌套模式下, 最高优先的 IS 刚结束操作。

当 8259A 不在嵌套模式下工作时, 8259A 不能分辨哪一个 IS 位必须复位, 所以明确的 EOI 指令此时必须承担这个工作, 通知 8259A 将哪一个 IS 复位。注意当 8259A 在特殊屏蔽模式下工作时被 IMR 屏蔽的 IS 位将不受明确 EOI 指令的影响。

### 5.3.6.3 自动中断结束 (Automatic EOI)

在编程为自动 EOI 模式时，系统在运行完一个中断操作之后，不需要送出 EOI 指令通知中断系统。中断系统会在最后一个  $\overline{INTA}$  结束时运行不明确的 EOI。

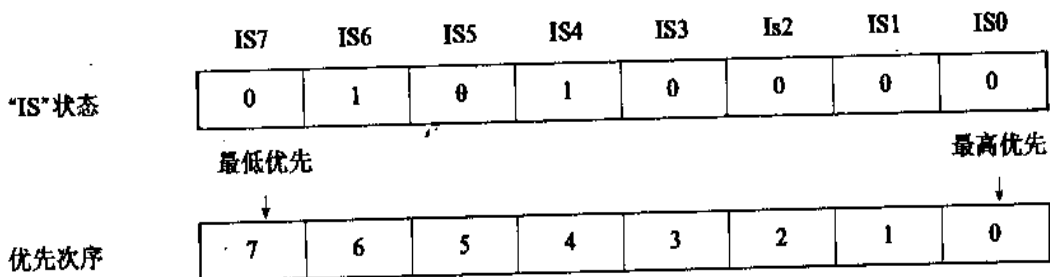
这个 EOI 模式只能用于 8259A 的主模式，而从模式并不适用。

### 5.3.6.4 自动循环式 (Automatic Rotation)

在某些应用中，有一些外设的中断具有相同优先权，当某一外设被服务完毕时，其优先权即变为最低。

图 5.9 为自动循环式优先权的说明。

循环之前 (IR4 是最高的优先)



在循环后 (IR4 被服务完，所有的优先次序相应进行循环)

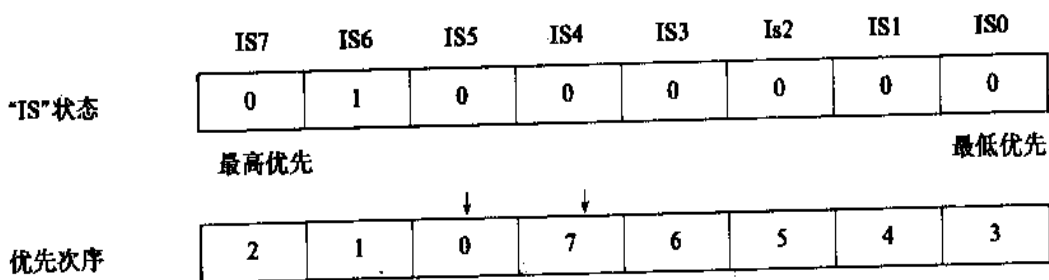


图 5.9 自动循环的优先次序说明

1. 在循环前 IR4 有最高的优先权，IS4=1 表示正接受服务。

2. IR4 被服务完后优先次序循环，IR5 变为最高优先权，而 IR4 则变为最低优先权。

有 2 种方式可以完成自动循环，一种是与不明确的 EOI 指令合用，另一种则是与 AEIOI 合用。

### 5.3.6.5 明确循环 (Spicific Rotation)

当每一次有一中断请求结束服务后，用户可以改变一个中断的优先次序。在这种模式之下，OCW2 可以用来改变内部状态。这种模式和 EOI 指令无关。

### 5.3.6.6 中断屏蔽 (Interrupt Masks)

每一个中断均可单独由中断屏蔽寄存器 (IMR) 或特殊的屏蔽形式完成。这种模式可由 OCW1 编程。

5.3.6.7 特殊屏蔽模式 (Specific Mask Mode)

为什么需要特殊屏蔽模式呢?

在运行程序时,某些场合可能须将一些中断屏蔽掉,而过一段时间又必须打开。在这种情形之下,这种模式就非常有用。它可以由 OCW3 编程进入此种工作模式。

5.3.6.8 缓冲型模式 (Buffer Mode)

为什么需要这个模式呢?

当 8259A 用于大系统时,通常需要缓冲区用以驱动总线。而当 8259A 串联使用时,这些缓冲区将造成一些实际的困难。而缓冲型模式正可以解决这个问题。方法为:每当 8259A 需要将总线驱动时,  $\overline{SP}/\overline{EN}$  将变为输出的信号,用于驱动缓冲区,8259A 可以借助编程 ICW4 完成这个目标。

5.3.6.9 采样指令 (Poll Command)

在这种模式下,INT 已不能去中断系统,此时的中断只靠 OCW3 中的 P 位,当 OCW3 中的 P 位被定为 1 时,8259A 将下一次的 RD 信号当成是由系统发出的中断确认信号,如果有中断申请时,将对应的中断服务 IS 设为 1。

5.3.7 中断优先权的类型摘要

为了使读者更易了解中断优先权的不同类型,下表为一个摘要:

表 5.4 中断优先权的类型摘要

中断优先权的类型	操作方式	在 EOI 后对优先权的影响	
		不明确/自动	明确
嵌套类型	IRQ0#最高优先 IRQ7#最低优先	优先权不变动,最高优先的 ISR 位被改变为 0	没有影响
自动循环 (元件的优先权均等)	刚被服务过的中断变最低优先,其它则顺着循环至符合嵌套形式处。	最高优先的 ISR 位被改变为 0,且其对应的 ISR 变为最低优先。	没有影响
明确循环 (明确循环顺序元件)	用户指定最低优先的顺序。其它元件的优先权则循环至符合嵌套类型为止。	没有影响	同操作方式

5.3.8 寄存器操作摘要

表 5.5 寄存器操作摘要。

表 5.5 寄存器操作摘要

操作说明	命令字	位
嵌套形式	OCW-即定	-
不明确 EOI 指令	OCW2	EOI
明确 EOI 指令	OCW2	SL,EOI,L0-L2
自动 EOI 形式	ICW1,ICW4	IC4,AEOI
不明确 EOI 循环指令	OCW2	EOI
自动 EOI 循环形式	OCW2	R,SL,EOI
设置优先权指令	OCW2	L0-L2
明确 EOI 循环指令	OCW2	R,SL, EOI
中断屏蔽寄存器	OCW1	M0-M7
特殊屏蔽形式	OCW3	ESMM,SMM
电平激活形式	ICW1	LTIM
边沿触发器形式	ICW1	LTIM
读取寄存器指令, IRR	OCW3	RR,RIS
读取寄存器指令, ISR	OCW3	RR,RIS
读取 IMR	IMR	M0-M7
采样指令	OCW3	P
特殊嵌套形式	ICW2,ICW4	IC4, SFNM

## 5.4 如何读取 8259A 的状态 IRR, ISR 及 IMR

8259A 的某些寄存器可以被系统读出。

1. IRR: 在读之前, OCW3 的 RR 设置为 1,RIS=0.
2. ISR: 在读之前, OCW3 的 RR 设置为 1,RIS=0.
3. IMR: 当 A0=1,即 (OCW1), RD=0 时, IMR 即被读出。

## 5.5 串连模式

8259A 可以很简单地被编程为 64 级的中断系统。参见图 5.10 8259A 的串联扩充。

## 5.6 82380 简介

在 80386 系统中, 已不使用 8259A, 8255A 以及 82C54。取而代之的是 82380 全局系统外设。使用这一片 IC 的好处为:

- 1.减少所使用的 IC 零件数。
- 2.减少 80386 装配时间, 降低装配成本以提高竞争力。
- 3.减少维修时间。维护工程师可以很容易的换修这一片 IC。
- 4.由于不再零散的插在电路板上, 更可以增加系统的速度。

思考: 是否有一天 82380 会和 80486 及其它的系统外设一起做在一片 IC 上? 为什么?

作者认为否, 您认为呢?

近年 IC 工业蓬勃发展, 设计能力及制造能力日新月异, 十年后的今天, 是否 80486

计算机系统会和昔日的 APPLE II 一样为现代人所遗忘?

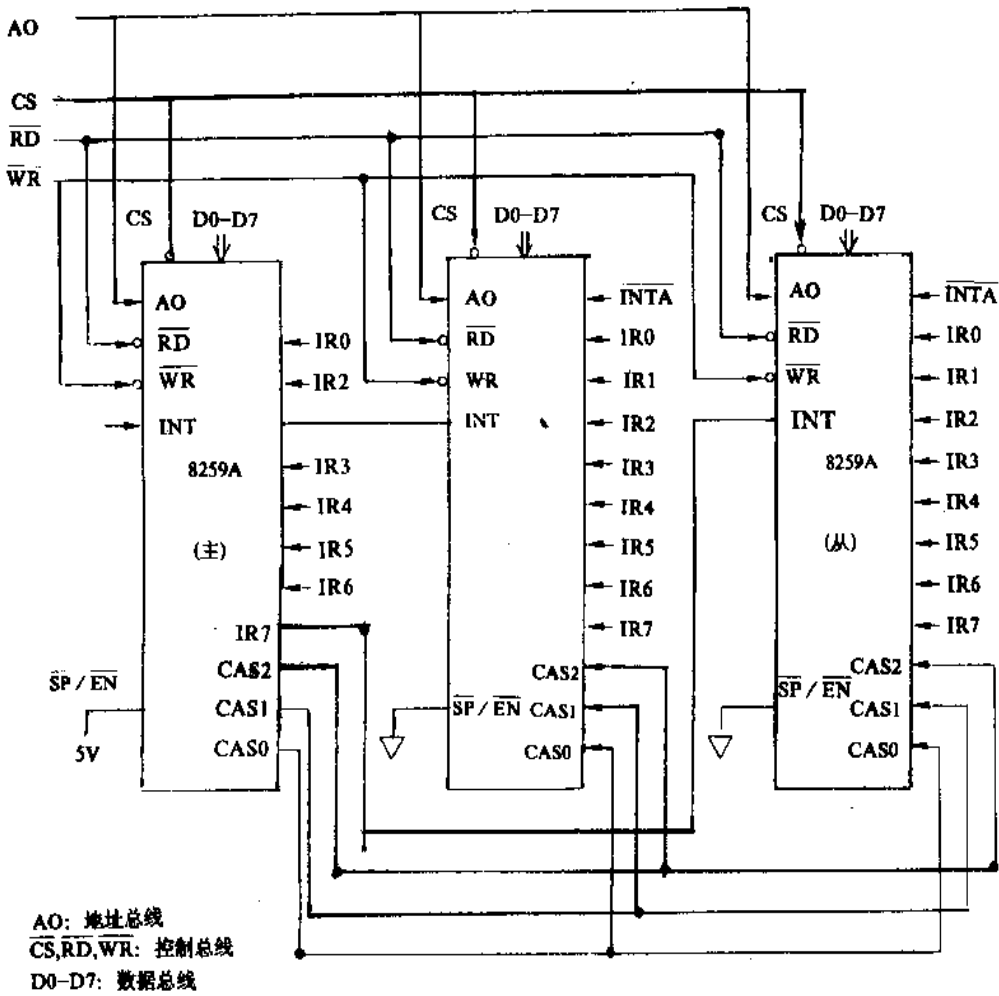


图 5.10 8259A 的串联扩充线路图

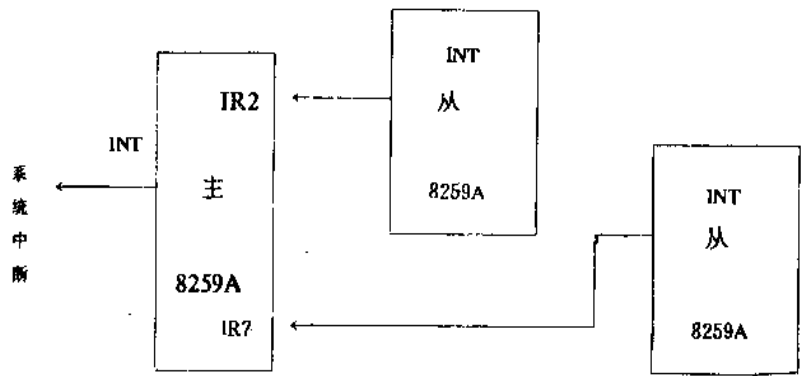


图 5.11 8259A 的主从串联

82380 是多功能的外设元件，如图 5.12 所示，主要由下列功能 IC 元件组合而成：

- DMA 控制器
- 中断控制器 (Interrupt Controller)
- 可编程间隔计时器 (Programmable Interval Timer)
- DRAM 刷新控制器 (Refresh Controller)
- 等候状态产生器 (Wait State Generator)
- 系统复位电路 (System Reset Circuit)

由于 82380 是针对 80386 系统而设计, 所以可以直接接至 80386 上, 并可在必要时控制 80386 的总线, 当在从(slave)类型时, 82380 会监视 80386 的状态, 并随时根据 80386 的指令而动作。同时它还会监视地址输送的状态。

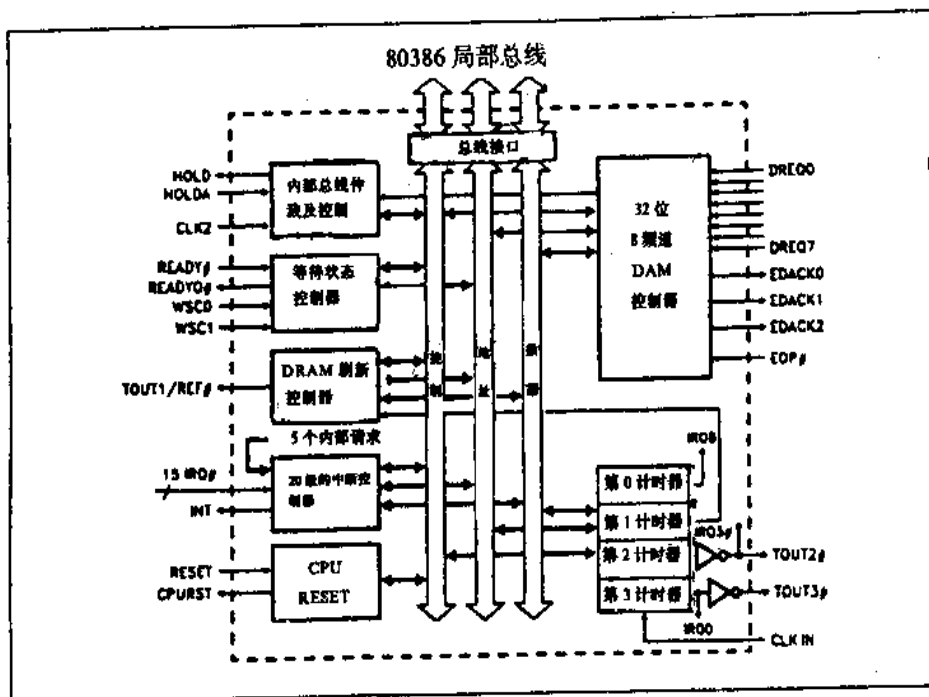


图 5.12 82380 的内部结构

由于具有以上功能, 这片 IC 可以很有效率地控制着外设和系统之间的数据传输。

## 5.7 思考题

1. 比较 IBM PC XT, 80286, 80386, 80486 的中断来源, 各有何不同?
2. 驱动器、打印机、计算机屏幕是否有中断请求? 解释说明系统时间如何产生。
3. 说明计算机键盘如何中断系统输出字符到屏幕上。如果您不了解计算机键盘的操作, 请想象编程之。
4. 如果在您忙着做计算机习题的同时, 您希望不受电话声干扰, 您说如何将电话信息传至计算机屏幕上, 显示如下信息:  
“请注意, 054-215728 正拨电话给您”



## 第六章 IBM PC 及 8259A 的基本应用示例

### 本章学习目的

1. 读者学习了8259A的理论基础之后，经过本章的实用示例可以对IBM PC的中断有更深一层的了解。
2. 在第四章中，我们学习了交互式数据传输，本章将原示例改为以中断式方法重做，让读者更明了 IBM PC 的中断应用。

### 本章内容

- 6.1 PC 系统中断模拟
- 6.2 8255A 中断交互式数据输入模拟，8255A 模式 1
- 6.3 8255A 中断交互式数据输出模拟，8255A 模式 1
- 6.4 8255A 中断交互式数据输出输入模拟，8255A 模式 2
- 6.5 思考题

在 80386,80486 中，已经没有 8259A 存在，所以本章中的中断直接通过 82380 向 80386 或 80486 申请中断。在这个前提下，任何不正确的中断将导致系统死机。建议在作这一类应用时要特别小心，熟读 80386，80486 的系统，并研读 BIOS 中的中断，这样可以避免死机。当然重新开机之后，系统仍会在您的掌握之中。

本书作者在处理这一章时遇见了一些问题，也将这些问题列出，让您避开这些同样的错误。请您修改程序以增强能力，训练您编写软件的技巧。

这些示例在 80386，80486 中均成功地运行过。所使用的系统为 DOS 5.0，如果在您的计算机上出现问题时，请先检查您的系统。

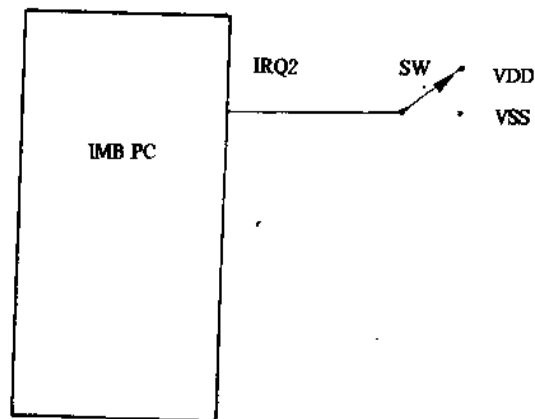
### 6.1 PC 系统中断模拟

#### 工作目标:

设置系统中断，当外接中断信号 IRQ2 启动时，屏幕上出现“2nd Interrupt happen”。

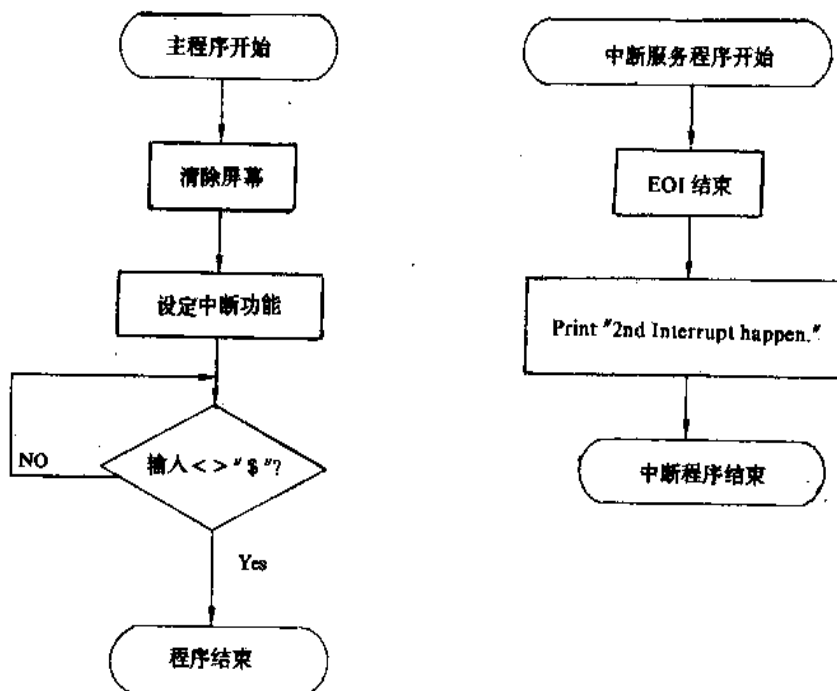
本示例给予读者一个最基本的中断概念，用以说明 PC 系统的中断如何产生及其影响。

#### 硬件设计:



当 SW 由 VSS 变 VDD 时，系统会在屏幕上出现“2nd Interrupt happen”的信息。按下任何键则程序结束。

软件设计:



\* 在此期间手动将 IRQ2 拨回 VSS.

程序示例 ch6\_1.c

基本中断应用程序设计.

```

/* ----- */
/*      Program Name : ch6_1.c      */
/*      Basic interrupt application  */
/*      For 8259                     */
/* ----- */
#include <dos.h>
#include <conio.h>
  
```

```

#define HARDINT 0x0a
void interrupt ctl_c();
void interrupt (*oldhandler) (void);
unsigned char mark;
char ch = '$';
void main()
{
    unsigned char bytewrite, byteread, PORT;

    clrscr();
    /* save the original interrupt mask status */
    PORT = 0x21;
    byteread = inportb(PORT);
    mark = byteread;
    bytewrite = 0xfc;
    outportb(PORT,bytewrite);

    /* set the Vector Address */
    oldhandler = getvect(0x0a);
    setvect(HARDINT,ctl_c);

    /* recall the interrupt mask status */
    bytewrite = mark;
    bytewrite &= 0xfb;
    outportb(PORT,bytewrite);

    gotoxy(32,10);
    printf("Waiting for Data input");
    while ( ( ch = getch() ) != '$' )
        ;
    setvect(0x0a,oldhandler);
}

void interrupt ctl_c(void)
{
    char str[] = "2nd Interrupt happen. Press any key to exit.";
    unsigned char bytewrite, PORT;
    int far *farptr;
    int i;

    clrscr();
    /* - EOI - */
    PORT = 0x20;
    bytewrite = PORT;
    outportb(PORT,bytewrite);
}

```

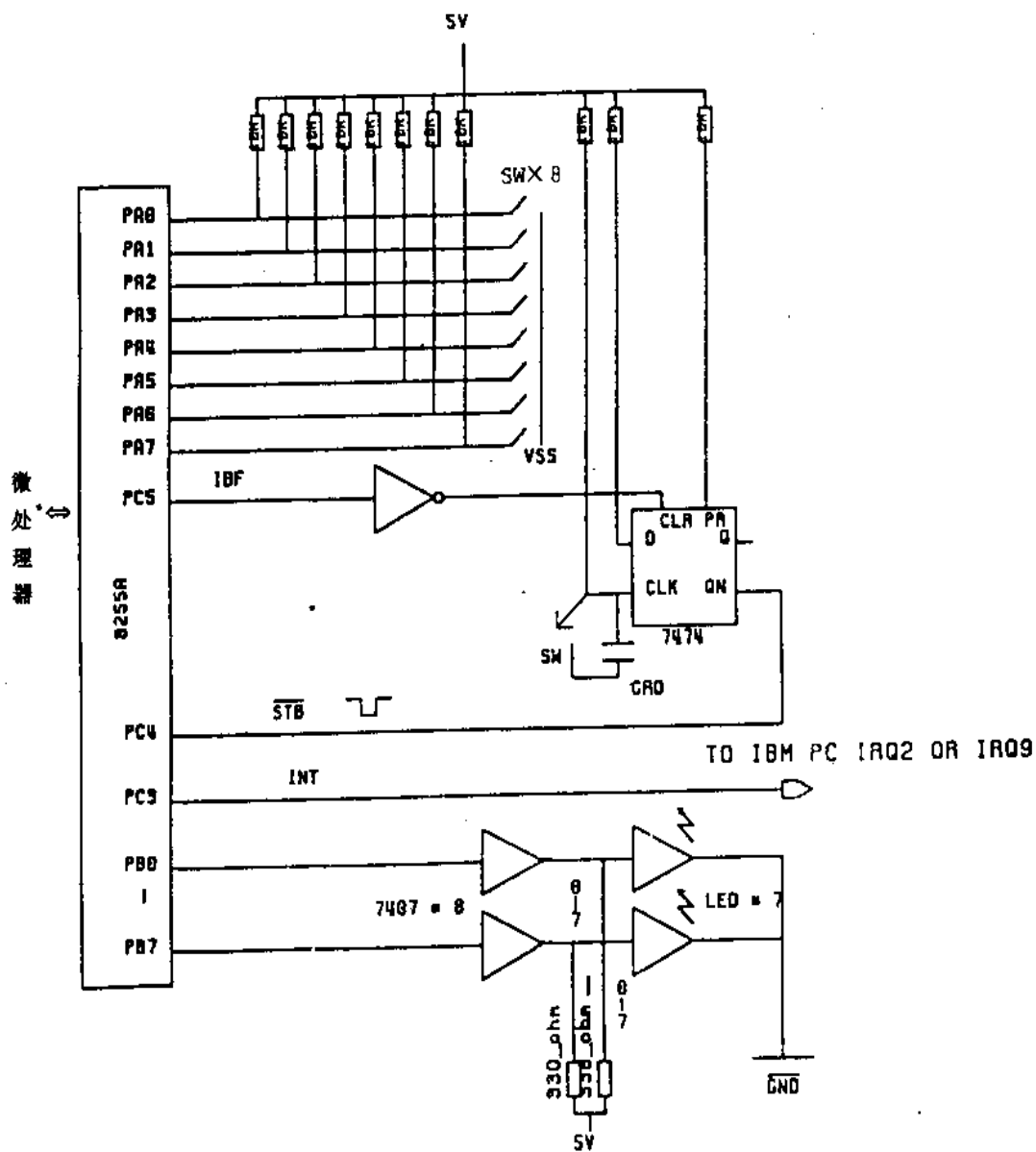
```

/* display the message */
farptr = (int far *)0xb8000000;
for ( i = 0; i <= 43; i++ )
    *(farptr + i) = str[i] | 0x0700;
}

```

思考:

1. 再接上IRQ3, 当IRQ3来时, 屏幕上出现“3rd Interrupt happen”. 此时用户有IRQ2及IRQ3的选择.
2. 将IRQ2改为IRQ0~IRQ7, 并记录它们所产生的影响, 加以解释.



## 6.2 8255A 中断交互式数据输入模拟, 8255A 模式 1

### 工作目标:

利用 8255A 第一种工作模式, 将数据由 PORTA 输入, 在这种模式下 A 组将有 INTR(PC3)中断信号产生, 利用此信号中断系统。此时系统将读取 PORTA 的数据, 再将它输出到 PORTB。

注: 本示例需要一个由外设产生的 $\overline{\text{STB}}$ 信号, 由于我们并没有真正的外设, 所以由外部 7474 以及一个开关模拟。

本示例与示例 4-5 类似, 唯一不同点在于示例 4-5 不用 INTR, 系统必须每隔一段时间查询外设是否需要服务, 本示例所采用的是 INTR 中断方式。

硬件设计: 见上图所示

所要输出的数据由用户改变  $\text{SW} \times 8$  而定。这些开关平常接上拉电阻, 所以为高电平, 当 Switch 接至 VSS 时数据才变为 0。

### 工作方式:

- ① 8 个在 PORTA 上的开关模拟外面外设要送给 8255A 的数据。
- ② 7474 CLK 开关模拟出  $\overline{\text{STB}}$ (strobe), 用以通知 8255A, “外设的数据准备好了”。并利用  $\overline{\text{STB}}$  将数据锁入 PORTA 的输入锁存器。
- ③ 8255A 送出 INTR 给微处理器, 通知微处理器“我已经准备好了, 你可以来将数据取走了”。
- ④ 微处理器将所获得的数据送给 PORTB, 在 PORTB 上可以见到与 PORTA 相同的数据。
- ⑤ 若按下任一键, 则程序结束。
- ⑥ 否则重返①; 继续运行。

软件设计: 见 89 页流程图

### 程序示例 ch6\_2.c

```
/* ----- */
/*      Program Name : ch6_2.c      */
/*      Basic handshake and interrupt application      */
/*      For 8255 and 8259      */
/* ----- */

#include <dos.h>
#include <conio.h>
#define HARDINT    0x0a
#define SETPORT    0x3c3
#define PORTA      0x3e0
#define PORTB      0x3e1
void interrupt ctl_c();
void interrupt (* oldhandler) (void);
char ch = '$';
```

```

void main()
{
    unsigned char bytewrite, byteread;

    clrscr();
    /* set the Vector Address */
    oldhandler = getvect(0x0a);
    setvect(HARDINT,ctl_c);

    bytewrite = 0xb8;
    outportb(SETPORT,bytewrite);

    /* enable interrupt */
    bytewrite = 0x09;
    outportb(SETPORT,bytewrite);

    gotoxy(32,10);
    printf("Waiting for interrupt.");

    while ( ( ch = getch() ) == '$' )
        ;
    setvect(0x0a,oldhandler);
}

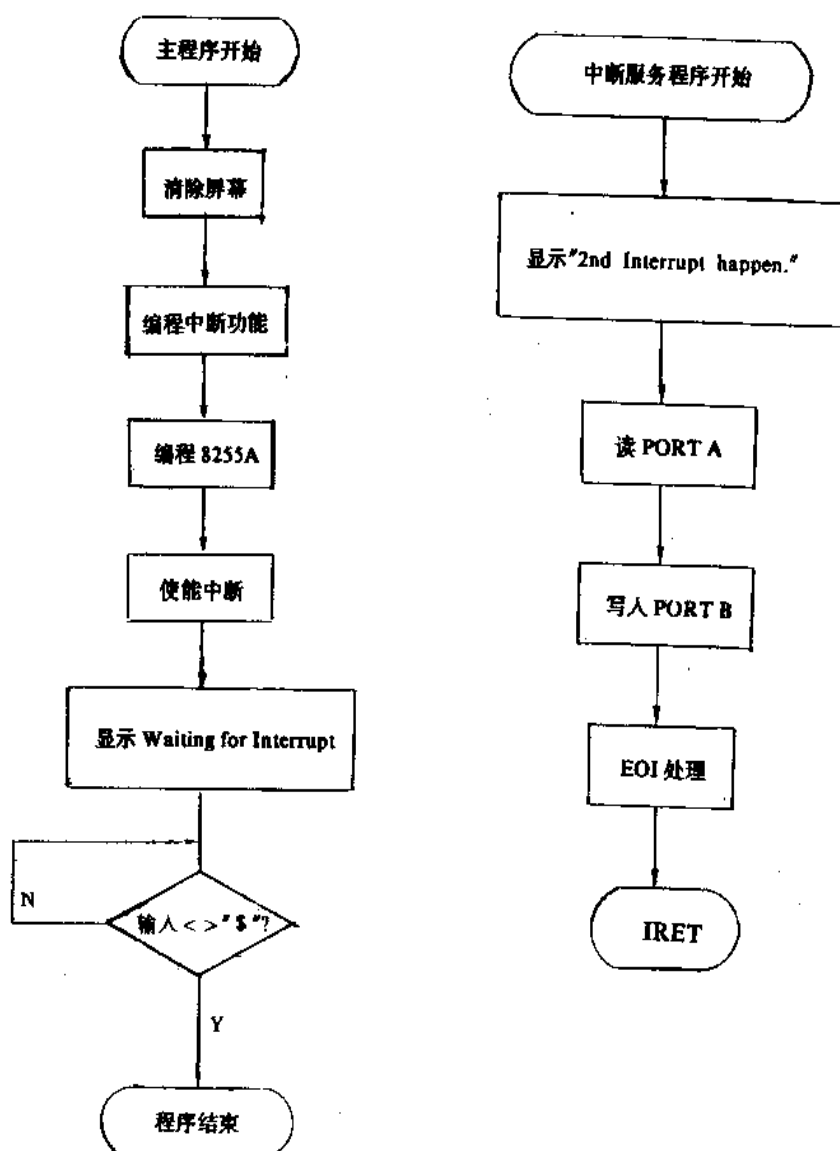
void interrupt ctl_c(void)
{
    char Str[] = "2nd Interrupt happen. Press any key to exit.";
    unsigned char bytewrite, byteread, PORT;
    int far *farptr;
    int i;

    clrscr();
    /* display the message */
    farptr = (int far *)0xb8000000;
    for ( i = 0; i <= 43; i++ )
        *(farptr + i) = str[i] | 0x0700;

    byteread = inportb(PORTA); /* read from PORTA */
    bytewrite = byteread;
    outportb(PORTB,bytewrite); /* output to PORTB */

    /* EOI */
    PORT = 0x20;
    bytewrite = 0x20;
    outportb(PORT,bytewrite);
}

```



思考:

1. 如果您没有7474这片IC, 您应如何利用单一的开关去模拟 $\overline{STB}$ 的信号? 软件应如何配合?
2. 如果不接PC3, 软件应如何配合才可以达到相同的目的?
3. 查看您的PC, INT应该接IRQ2还是IRQ9?
4. 这个程序是如何停止的? 研究一下, 并说出它的优缺点? 有无改进的方式?

### 6.3 8255A 中断交互式数据输出模拟, 8255A 模式 1

### 工作目标:

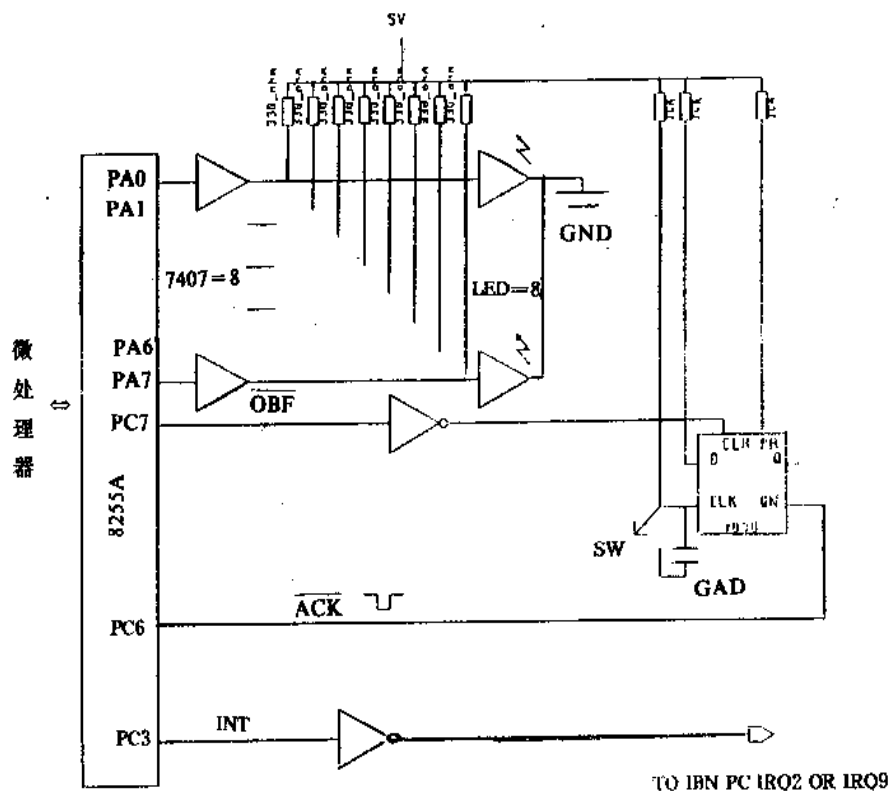
利用 8255A 第一种工作模式，将数据由系统一个一个地输出至 PORTA。在 PORTA 上将见到 1, 2, 3.....由系统输出的数据。按下 Q 键，则程序结束。

**注:**

本示例需要一个由外设产生的ACK信号，用以通知8255A，“由系统送过来的数据已由外设收到了”。由于我们并没有真正的外设，所以由外部7474以及一个开关模拟。

本示例与示例 4-6 类似，唯一不同点在于示例 4-6 不用 INTR，系统必须每隔一段时间询问外设是否要求服务，本示例所采用的则是 INTR 中断方式。

### 硬件设计:

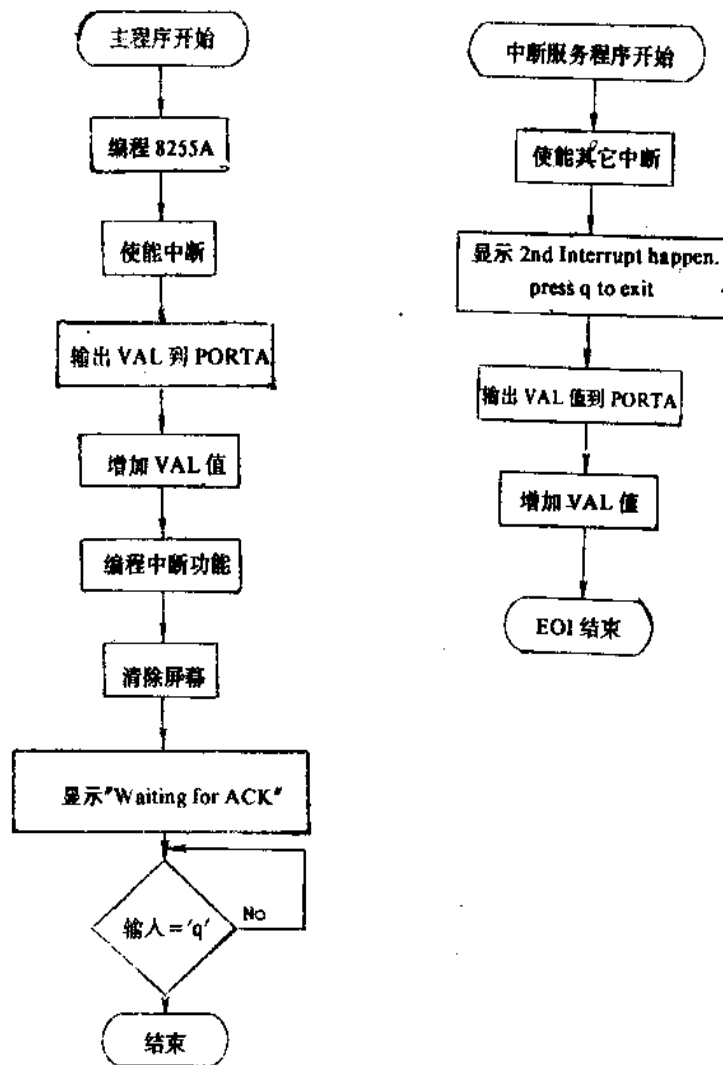


**工作方式:**

- ①微处理器发送数据“1”（数据每次增加1）给 PORTA。
- ②由 7474 的 SW 模拟ACK的信号，通知 8255A，“数据已经收到了”。
- ③微处理器发送数据“2”给 PORTA。
- ④重复②-④。



## 软件设计:



## 程序示例 ch6\_3.c

```

/* ----- */
/*      Program Name : ch6_3.c      */
/*      Basic handshake and interrupt application 2      */
/*      For 8255 and 8259      */
/* ----- */
#include <dos.h>
#include <conio.h>
#define HARDINT    0x0a
#define SETPORT    0x3e3
#define PORTA      0x3e0
void interrupt ctl_c();
void interrupt (*oldhandler) (void);
  
```

```

char ch;
int val = 1;
unsigned char flag;

void main()
{
    unsigned char bytewrite, byteread, PORT;

    /* set the control register */
    bytewrite = 0xa0;
    outportb(SETPORT,bytewrite);

    /* enable interrupt */
    bytewrite = 0x0d;
    outportb(SETPORT,bytewrite);

    /* output 1 to PORTA, which is simulated as a data output */
    bytewrite = val;
    outportb(PORTA,bytewrite);

    val++;

    /* save original register */
    PORT = 0x21;
    byteread = inportb(PORT);
    flag = byteread;
    bytewrite = 0xfc;
    outportb(PORT,bytewrite);

    /* set the Vector Address */
    oldhandler = getvect(0x0a);
    setvect(HARDINT,ctl_c);

    bytewrite = flag & 0xfb;
    outportb(PORT,bytewrite);

    clrscr();
    gotoxy(32,10);
    printf("Waiting for ACK.");

    while ( ch != 'q' )
    {
        ;
        setvect(0x0a,oldhandler);
    }
}

```

```

void interrupt ctl_c(void)
{
    char str[] = "2nd Interrupt happen. Press q to exit.";
    unsigned char bytewrite, byteread, PORT;
    int far *farptr;
    int i, j;

    clrscr();

    /* display the message */
    farptr = (int far *)0xb8000000;
    for ( i = 0; i <= 37; i++ )
        *(farptr + i) = str[i] | 0x0700;

    bytewrite = val;
    outportb(PORTA,bytewrite);

    val++;

    ch = getch();

    /* EOI */
    PORT = 0x20;
    bytewrite = 0x20;
    outportb(PORT,bytewrite);
}

```

思考:

1. 如果您不方便找到 7474, 应如何利用单一开关模拟  $\overline{\text{ACK}}$ ?
2. 这种模式可能应用在哪些外设中?
3. 请重新制定这个流程图, 并设计程序, 它必须具有相同的功能。

提示: 可使用三态输入输出的开关。

## 6.4 8255A 中断交互式数据输出输入模拟, 8255A 模式 2

工作目标:

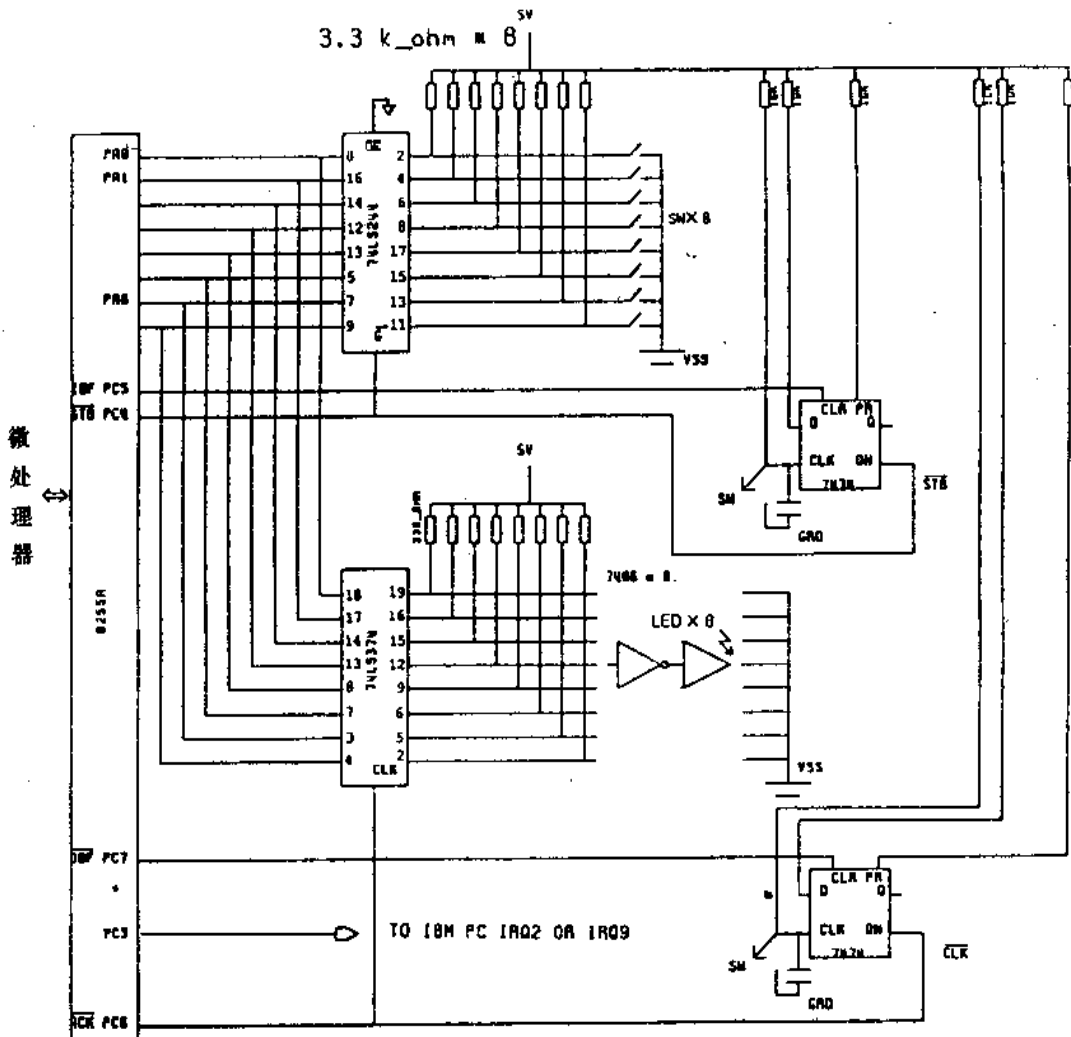
利用 8255A 模式 2, 自 PORTA 输入数据, 再将它的补码输出于 PORTA。任意按下一键, 程序结束。这个示例使用中断信号。

原理说明:

8255A 在模式 2 时 A 口是双向选通的总线 I/O, 只有 A 口才具有此功能。换言之, 它可以当成输出, 也可以当成输入。当系统对它运行输出的指令时, 它即变为输出口, 当系统对它运行输入的指令时, 它立刻又变为输入口。

**PC7 :  $\overline{\text{OBF}}$** , 由系统产生, 告诉外设输出缓冲区已满。

### 硬件设计:



**工作情形:**

1. 由 A 口的 8 个小开关  $SW \times 8$  选择要输入系统的数据。
2. 手动  $\overline{STB}$  开关, 模拟数据输入 A 口。
3. 系统利用  $\overline{STB}$  将数据读进输入锁存器, 并将  $IBF(PC5)$  置于低电平。
4. 手动  $\overline{ACK}$  开关, 当  $\overline{ACK}$  为低电平时, 系统将数据给  $PORTA$ 。
5. 程序结束。

在本实验中只利用了一次中断即将数据自 A 口输入而且补码自 A 口输出, INTR 编

程只有在 $\overline{STB}$ 发生时才产生。

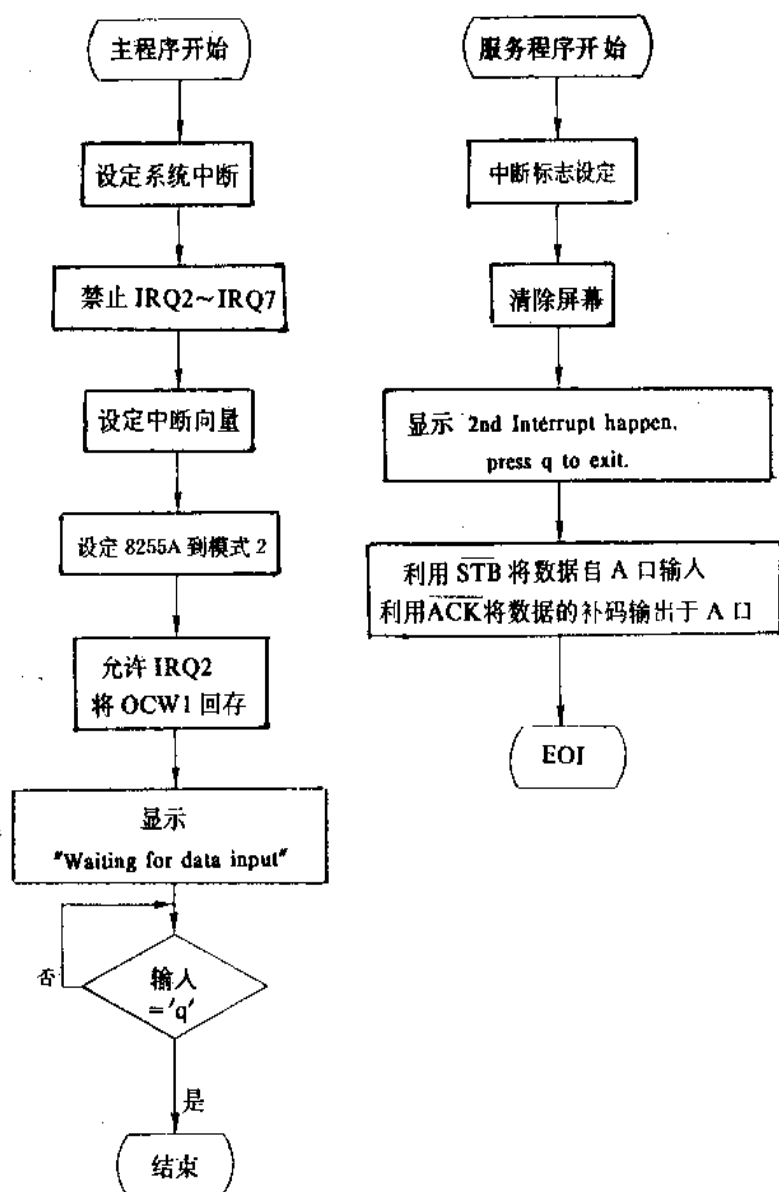
疑问：如何编程与 INTR 有关的软件呢？

图 3.26 中有 INTE1 及 INTE2。它的编程方式如 CH6.4.ASM 的第 91~94 行，本示例不使用 INTE1。

注意

74LS244 可以控制数据流动的方向，当 $\overline{STB}$ 为低电平时， $\overline{G}$ 也为低电平，此时 SW 所选择的数据可以进入 8255A。当 $\overline{G}$ 为高电平时，由 8255A 送出的数据自左向右传送。在本示例中对 SW 并无功能上的影响，此时 74LS374 即可将 8255A 送出的数据送给 LED 显示器。

软件设计：



## 程序示例 ch6\_4.c

中断交互式数据输出输入模拟

```
/* ----- */
/*      Program Name : ch6_4.c      */
/*      Basic handshake and interrupt application 3      */
/*      For 8255 and 8259, 8255 Mode 2 application      */
/* ----- */
#include <dos.h>
#include <conio.h>
#define SETPORT    0x3c3
#define PORTA      0x3c0
#define HARDINT    0x0a
unsigned char flag;
char    answer;
void    interrupt (*oldhandler)(void);
void    interrupt ctl_c();
void main()
{
    unsigned char bytewrite, byteread, PORT;

    clrscr();

    /* save the original interrupt mask status */
    PORT = 0x21;
    byteread = inportb(PORT);
    flag = byteread;
    bytewrite = 0xfc;
    outportb(PORT,bytewrite);

    /* set the Vector Address */
    oldhandler = getvect(0x0a);
    setvect(HARDINT,ctl_c);

    /* set control register to mode 2 */
    bytewrite = 0xc0;
    outportb(SETPORT,bytewrite);

    /* set INTE1
    bytewrite = 0x0d;
    outportb(SETPORT,bytewrite);
    */
    /* set INTE2 */
    bytewrite = 0x09;
```

```

    outportb(SETPORT,bytewrite);

/* recall the interrupt mask status */
    bytewrite = flag;
    bytewrite &= 0xfb;
    outportb(PORT,bytewrite);

    gotoxy(32,10);
    printf("Waiting for Data input");
    while ( getch() != 'q' )
        ;
    setvect(0x0a,oldhandler);
    return 0;
}

void interrupt ctl_c(void)
{
    char str[] = "2nd Interrupt happen. Press q to exit.";
    unsigned bytread, bytewrite, PORT;
    int far * farptr;
    int i;

    clrscr();
/* display the message */
    farptr = (int far *)0xb8000000;
    for ( i = 0; i <= 37; i++ )
        *(farptr + i) = str[i] | 0x0700;

    bytread = inportb(PORTA); /* read PORTA */
    bytewrite = ~bytread;
    outportb(PORTA,bytewrite); /* write PORTA */
/* EOI instruction */
    PORT = 0x20;
    outportb(PORT,0x20);
}

```

#### 思考问题:

1. 怎样将程序改为继续运行。
2. 本示例使用的中断只有一次。请修改程序，编程中断方式如下：

#### 目标:

- A. 若 $\overline{STB}$ 产生则将 A 口的数据输入。
  - B. 若 $\overline{ACK}$ 产生则将 A 口输入锁存器的数据，取其补码输出于 A 口。
- A, B 并无绝对顺序。
3. 将上题的流程图划出，并且依次详细解释。

提示: 1.由 PC7 可以判断 8255A 是应该输入数据还是输出数据。

## 6.5 思考题

1. 在有及没有中断时的工作方式下, 比较8255A的工作模式1及2.
2. 比较8255A的三种工作模式. 计算机实际应用中何种工作模式最有用?
3. 您想成为计算机硬件工程师吗? 如何使用8255A及8259A去设计新产品? 以调制解调器 (Modem) 想象并编程实现.



## 第七章 可编程计时器 8253 / 8254

### 本章学习目的

1. 8254是一个微型计算机系统中应用最广的计时器，经过本章的学习可以彻底的对它有所认识，进而熟练地应用它。
2. 配合下一章的应用实例可以增进应用的技巧。

### 本章内容

- 7.1 功能简介
- 7.2 8254 的结构及引脚说明
- 7.3 如何编程 8254
- 7.4 操作模式
- 7.5 操作编程
- 7.6 思考题

### 7.1 功能简介

每一部计算机都有一个准确的计时器，在 80286 以前使用 8253 / 8254 作为计时器 / 计数器，80386 微处理器则使用 82380。这些计数器的功能完全相同，只是 82380 拥有四组计时器，而 8253 / 8254 则只有三组。

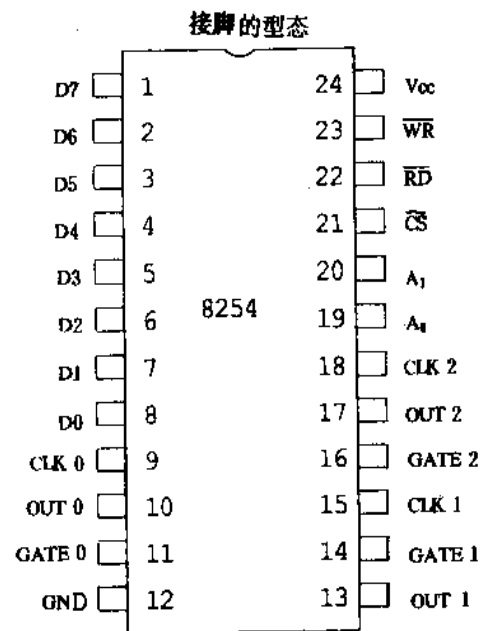
8253 和 8254 的引脚是兼容的，其功能也几乎完全相同，最主要的差别在于：

1. 8253 的最高工作频率为 2.6MHz，8254 为 8MHz，8254-2 则为 10MHz。
2. 8254 有回读的功能，可于任何时刻锁定计数器的值，8253 则无此功能。

下列这些功能均可以用 8254 来编程：

- 准确的实时时钟计时(Real Time Clock)
- 事件计数器(Event Counter)
- 数字触发器(Digital One-shoot)
- 可编程比率产生器(Programmable Rate Generator)
- 方波产生器(Square Wave Generator)
- 二进制乘法器(Binary Rate Multiplier)
- 复合式波形产生器

## 7.2 8254 的结构及引脚说明



引脚符号	引脚意义
D7~D0	双向数据总线
CLK <sub>n</sub>	计数器 n 时钟输入 n=0, 1, 2
GATE <sub>n</sub>	计数器 n 使能输入 n=0, 1, 2
OUT <sub>n</sub>	计数器 n 输出 n=0, 1, 2
$\overline{RD}$	读取计数器信号
$\overline{WR}$	写入命令或数据信号
$\overline{CS}$	芯片选择信号
A <sub>0</sub> ,A <sub>1</sub>	地址线选择信号
Vcc	+5V
GND	接地

图 7.1 8254 引脚图

### 信号说明:

- D7~D0      双向数据总线，与微处理器互相沟通
- CLK<sub>n</sub>      计数器时钟输入

**OUT<sub>n</sub>** 计时器输出  
**GATE** 计时器使能输入  
 **$\overline{\text{RD}}$**  由微处理器读取8254的信号  
 **$\overline{\text{WR}}$**  由微处理器编程8254的写入信号  
 **$\overline{\text{CS}}$**  芯片选取, 低电位时配合 $\overline{\text{RD}}$ , 或 $\overline{\text{WR}}$ 读取或将数据写入8254  
**V<sub>cc</sub>** 正电源  
**A1,A0** 用以选择以下三个计时器及控制位寄存器

A1 A0 选择  
 0 0 计时器#0  
 0 1 计时器#1  
 1 0 计时器#2  
 1 1 控制字寄存器

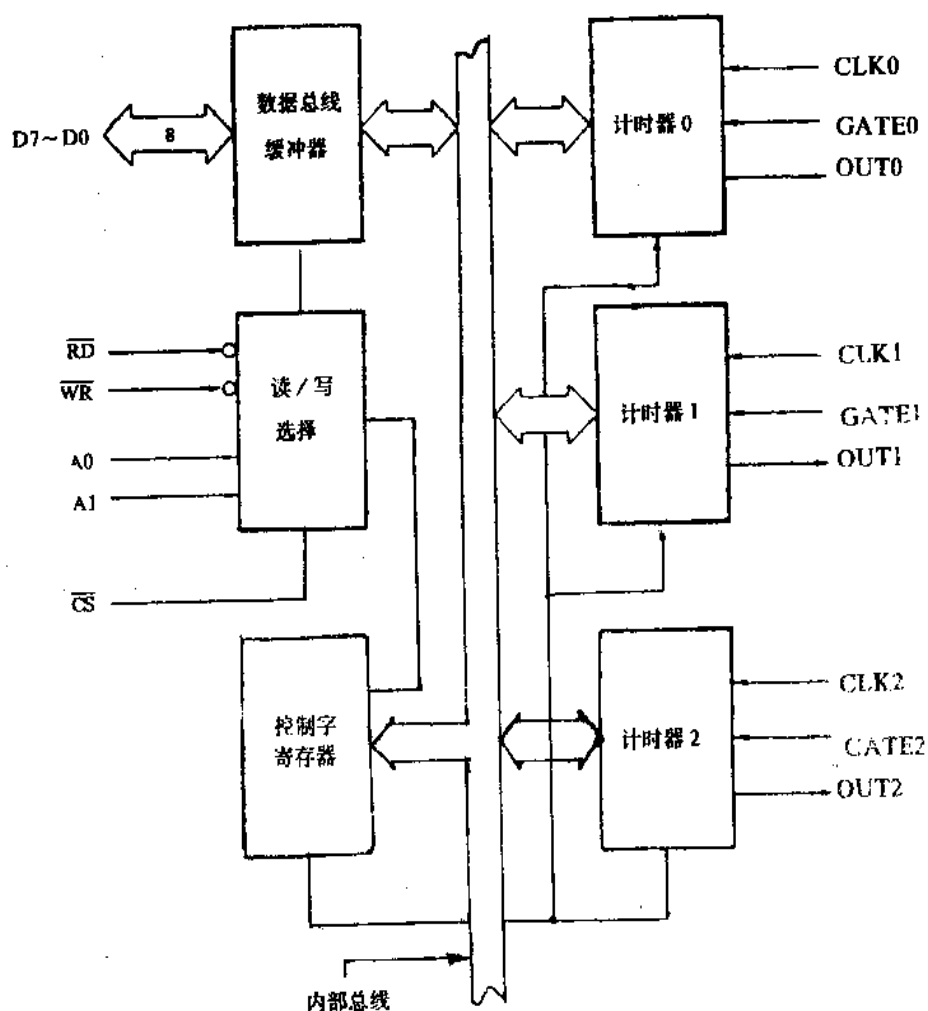


图 7.2 8254 内部结构方块图

表 7.1 为各个寄存器的地址及功能简介。图 7.3 为 8254 与系统总线的结构。

表 7.1 寄存器地址 功能简表

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A1	A0	功能
0	1	0	0	0	写入计时器 0
0	1	0	0	0	写入计时器 1
0	1	0	1	0	写入计时器 2
0	1	0	1	1	写入控制字
0	0	1	0	0	读取计时器 0
0	0	1	0	1	读取计时器 1
0	0	1	1	0	读取计时器 2
0	0	1	1	1	无动作
1	x	x	x	x	无动作
0	1	1	x	x	无动作

内部结构方块 (图 7.2):

以下分别介绍 8254 的内部方块:

- 读写逻辑
- 控制字寄存器
- 计时器 0,1,2

读写逻辑: 读写逻辑由微处理器的  $\overline{RD}$ ,  $\overline{WR}$  控制, A1,A0 用以选择 3 个计数器中的其中之一, 如表 7.1.

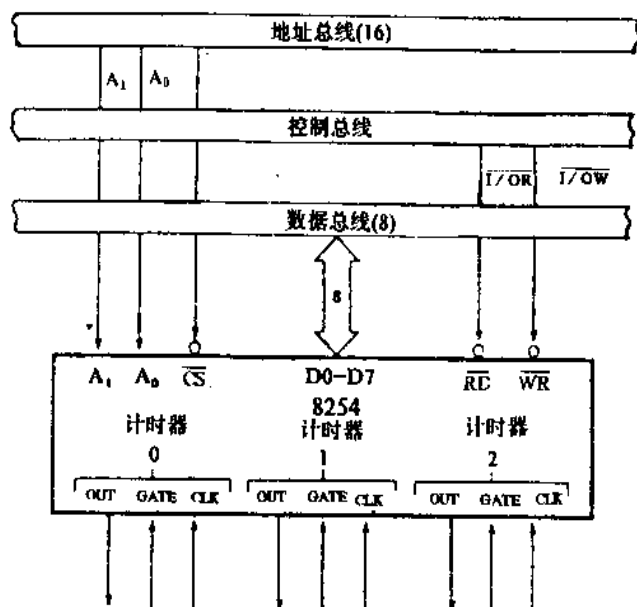


图 7.3 8254 与系统总线的结构

控制字寄存器：控制字寄存器用以编程各种不同的工作模式。另外可利用读的指令，读回这个寄存器的状态。

计时器 0,1,2: 这 3 组计时器功能完全相同，而且完全独立，可以分别编程成不同的工作模式。

### 7.3 如何编程 8254

8254 的控制字如表 7.2。当  $A1, A0 = 1, \overline{CS} = 0, \overline{RD} = 1, \overline{WR} = 0$  时，这个控制字可以编程出不同的工作模式。

控制字格式：

表 7.2 控制字格式

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

控制的定义：

SC—Select Counter (计时器的选择)

0	0	选择计时器 0
0	1	选择计时器 1
1	0	选择计时器 2
1	1	读回指令

RW—Read / Write (读取 / 写入)：

RL1 PL0

0	0	计时器锁定指令 (见读回指令)
0	1	只读 / 写低字节
1	0	只读 / 写高字节
1	1	先读 / 写低字节，然后读 / 写高字节

M—Mode (模式)：

M2 M1 M0

0	0	0	模式 0
0	0	1	模式 1
×	1	0	模式 2
×	1	1	模式 3
1	0	0	模式 4
1	0	1	模式 5

BCD:

0	16 位二进制计时器
1	BCD(Binary Code Decimal) 二进制十进位计时器 (4 位)

### BCD 格式:

BCD(Binary Code Decimal)是计算机系统的另外一种数字的表示方法。这种格式对于处理十进制小数点非常好用。

所谓 BCD 格式就是用四个二进制位,来表示一个十进制数字。这也是为什么它会被称为“二进码十进制”。BCD 格式又可分为压缩式(Packed)和非压缩式(Unpacked) 两种基本模式。

压缩式 BCD 格式,就是以一连串四位为一组的二进制来代表一个十进制数字。例如,十进制的 9502 可以写成

1001	0101	0000	0010
9	5	0	2

至于非压缩式的 BCD 格式为每一个十进位数字存放在字节的低四位,至于高四位放什么东西就不重要了。

uuuu1001	uuuu0101	uuuu0000	uuuu0010
9	5	0	2

u:表示我们不关心所放的内容。

## 7.4 操作模式

每一个计时器可分别编程成六种不同形式的操作模式。计时器编程的方式是将一控制字写入控制字寄存器,然后再写入一个初始计数值。

名词定义:

CLK 脉冲 : 由CLK的上升缘至下降缘。

触发 : 计时器的GATE输入上升缘。

计时/计数器的取入 : 将计数值由计数寄存器传至计数元件。

研究操作模式的最好方式即研读时序图。

### 7.4.1 第0种形式: 终止计数时产生中断 (Interrupt on Terminal Count)

第 0 种形式用于事件的计数 0,当控制字写入寄存器之后,OUT 信号变为低电平,然后计时器开始倒数,而且一直维持到计数值为零为止。此时 OUT 信号会变为高电平。新的计数值或新的第 0 形式的设置可以将 OUT 的输出改变为低电平,再重新计数。

在这一形式下,当 GATE 为高电平时会使计数器使能,在低电平则禁止。

在控制字及起始计数值写入计时器之后,起始计数值会在下一个 CLK 脉冲来时才被写入。这一 CLK 并不会影响计数值,因此若起始计数值为 N,则在第 N+1 个脉冲时,OUT 信号才会变为高电平。

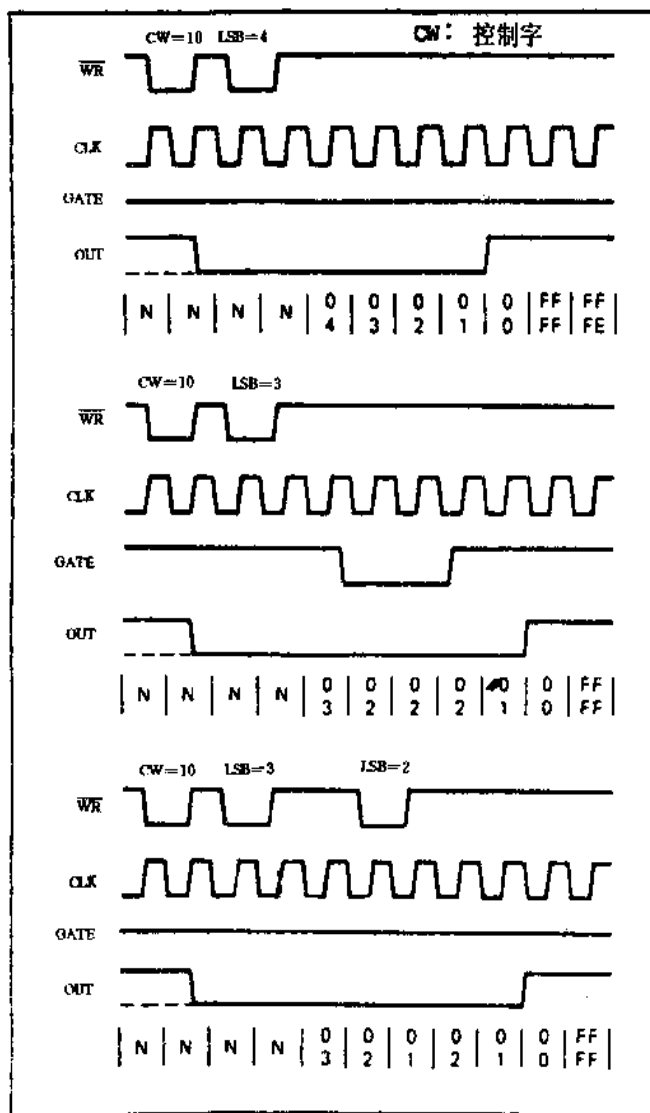
若在计数的过程中,初始计数值被改变,新值要在下一个 CLK 来之时被写,然后即由此新的计数值重新开始计数。如果写入的是双字节的 16 位计数值,下列情形会发生。

1. 写入第一个字节时会使计数值禁止,OUT 信号此时立即变为低电平。
2. 写入第二个字节时将使新的计数值在下一个 CLK 脉冲来时才被写入。

这样计数和软件可以同步。

若起始值在 GATE 为低电平时写入计数器，则计数值将在下一个 CLK 脉冲来时才被写入，当 GATE 变为高电平后，OUT 在第 N 个 CLK 之后变为高电平。N 为读者自定义的计数值。

图 7.4 为模式 0 的时序图。



注：下面规则适用于所有模式的时序图：

1. 计时器是被编程成二进制（非BCD）和仅读/写低字节。
2. CS一定是在低电位被选取。
3. CW的意义是控制字（Control Word），CW=10，表示10H控制字被写入计时器内。
4. LSB表示计时器的低字节。
5. 在时序图下方的数目表示计时值。

图 7.4 模式 0 的时序图

#### 7.4.2 第1种形式：硬件可触发单触发器 (Hardware Retriggerable One-shoot)

工作方式：

1. OUT 状态刚开始为高电平。
2. 在一触发启动(One Shoot)后，OUT 即会在脉冲来时变成低电平。
3. OUT 信号一直维持到计时器至零之前才改变。
4. 计时器变为 0 时，OUT 状态立即变为 1。直至下一次触发为止。

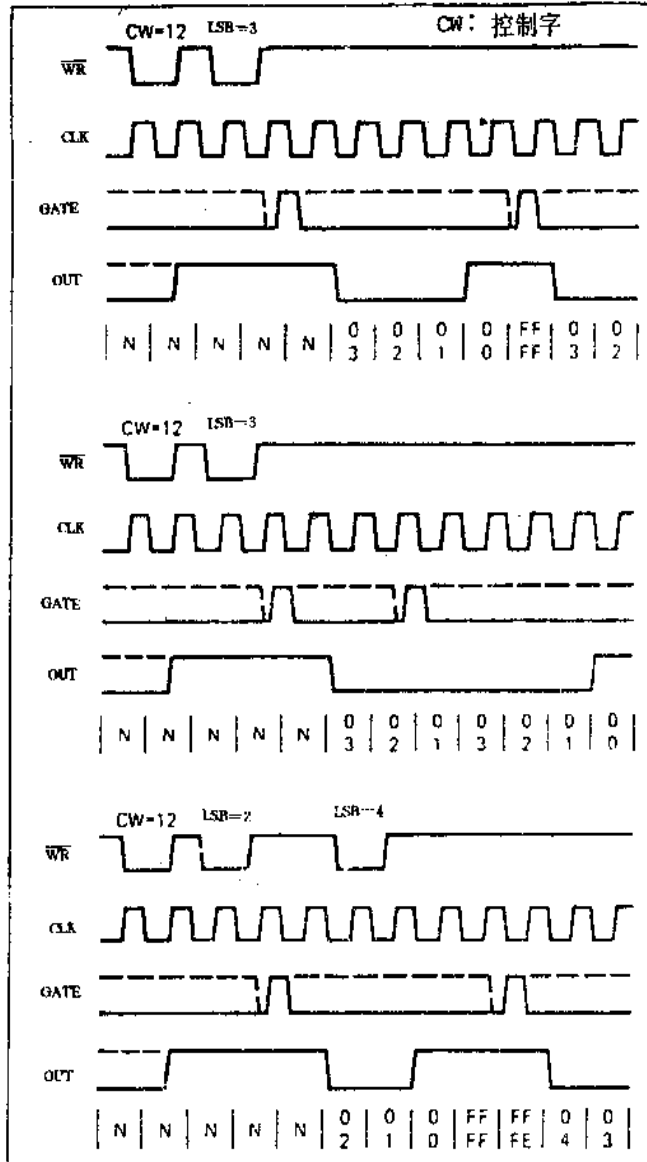


图 7.5 模式 1 的时序图

在写入控制字及起始计数值之后，计时器即等待触发脉冲来临，将 OUT 变为低电



平。因此起始计数值  $N$  将产生一个宽度为  $N$  个脉冲长的单触发脉冲。这种单触发操作可以重触发(retriggerable)。也就是说,在每一次触发之后,OUT 信号会停留在低电平状态  $N$  个脉冲之久。此种模式之下不需要再次将计数值写入计时器,单触发器即可重复操作。

在单个触发操作期间,若有新的计数值写入计时器,此时必须在计时器重新触发之后,现有的单触发脉冲宽度才受影响。

图 7.5 为模式 1 的时序图。

#### 7.4.3 第 2 种形式: 比率产生器 (Rate Generator)

这一形式其实就是除以  $N$  的计时器,其应用实例如:实时时钟中断等。刚开始时 Out 输出信号为高电平,当计时器递减为 1 时 Out 才会变为低电平。一个 CLK 后再变为高电平。此时计时器会再重新将初始计数值写入,然后又继续计数。因此这一形式周而复始。

GATE 为高电平时,会令计时器使能,而 GATE 为低电平时,计时器则无动作,当 GATE 恢复为高电平时,计时器又继续动作。若 GATE 在 OUT 为低电平时变为低电平,OUT 会立即变为高电平。GATE 的上升沿会在下一个 CLK 脉冲来时将起始计数值重新写入计数器内,然后在  $N$  个 CLK 脉冲后,OUT 输出为一个 CLK 脉冲长的低电平,因此 GATE 可以使计数器同步。

在写入控制字及起始计数值之后,计时器会在下一个 CLK 脉冲来时输入新值。而在起始计数值写入的第  $N$  个 CLK 脉冲之后,OUT 会变为低电平。这是计时器另一种利用软件而达到同步的方式。

在计数过程中写入一个新的初始计数值并不会影响现有的计数状态,因为新的初始计数值只会在现有的计数周期结束后才被写入。若在写入新的计数值之后而现有的周期尚未结束之前又有触发,必须等到触发过后的下一个 CLK 脉冲时,新的计数值即会被写入计数器内,同时将由新的计数值开始计数。这种计数形式,初始计数值不可设为 1。

图 7.6 为模式 2 的时序图。

#### 7.4.4 第 3 种形式: 方波产生器 (Square Wave Mode)

第三种形式主要应用于波特率(Baud Rate)的产生。除了 OUT 输出工作周期(Duty Cycle)不同之外,它的功能与第 2 种形式类似。在第 3 种形式时,OUT 刚开始为高电平,在起始计数值过了一半之后,OUT 在另一半周期变为低电平,此一计数过程自动重复,因此可以周期性的产生  $N$  个 CLK 脉冲长的方波。

GATE 为高电平时令计数使能,低电平时则令计数禁止。若 GATE 在 OUT 为低电平时变为低电平,则 OUT 会立即变为高电平。在下一个 CLK 脉冲时,起始计数值将被重新写入计数器内。

写入控制字与起始计数值之后的下一个 CLK 时,计时器才写入新值,这样新的计时器能以软件方式达到同步的效果。

在计数过程中如果写入新的计数值并不会影响原有的计数状态,但若在写入新计数值后,且在方波的现有半周期结束之前又有触发信号,则在下一个 CLK 脉冲时,新的计数

值即被写入，此时计数会由新的计数值重新开始。

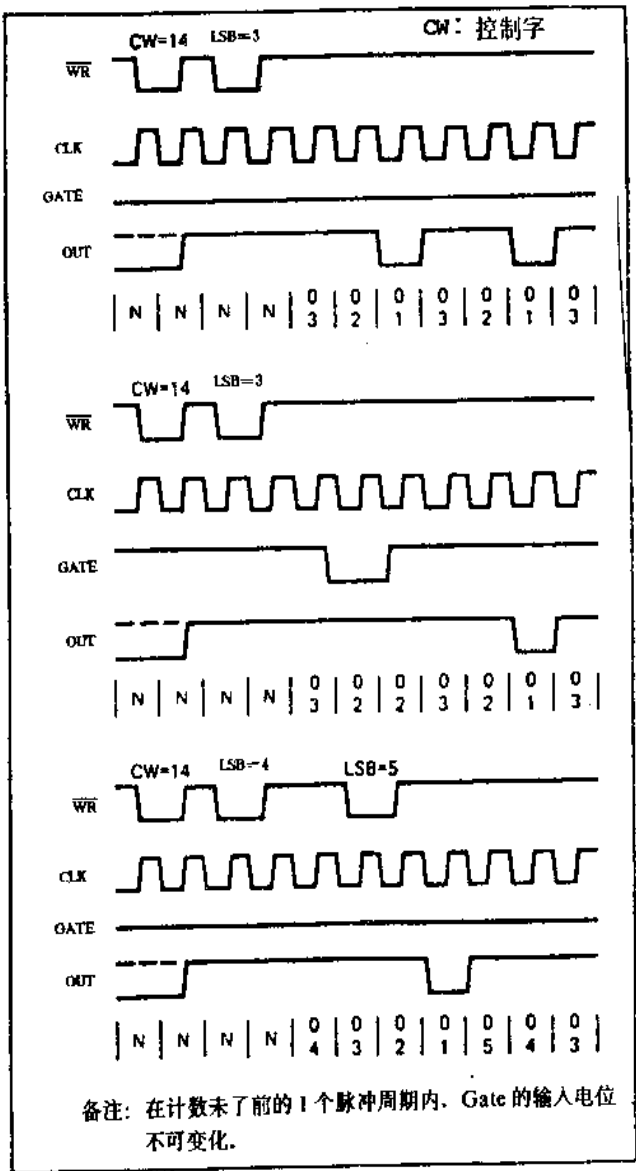


图 7.6 模式 2 的时序图

起始计数值为奇数或为偶数时，它的工作方式稍有不同。

偶数时，OUT 刚开始为高电平，起始计数值在下一个 CLK 脉冲时写入计时器，然后在紧接着的 CLK 脉冲时减 2。在计时器的值减为 2 时，OUT 变为低电平，且计时器重新输入起始计数值。此过程重复不断。

奇数时，OUT 刚开始为高电平，起始计数值先减 1，它的结果（新的计数值）在 CLK 脉冲时写入计数器，并在随后的 CLK 脉冲到来时每次减 2，在计时值到期(expired)后的一个脉冲时，OUT 变为低电平，且计时器再次写入初始计数值减了 1 后的新值。紧接着的 CLK 脉冲也每次将计数值减 2。在计数值到期后(expired)，OUT 立即变为高电

平, 计数器又存入初始值减一后的新值, 重复不断。

在起始计数值为奇数时, OUT 将处于高电平达  $(N+1)/2$  个脉冲之久, 而处于低电平达  $N/2$  个脉冲之久。

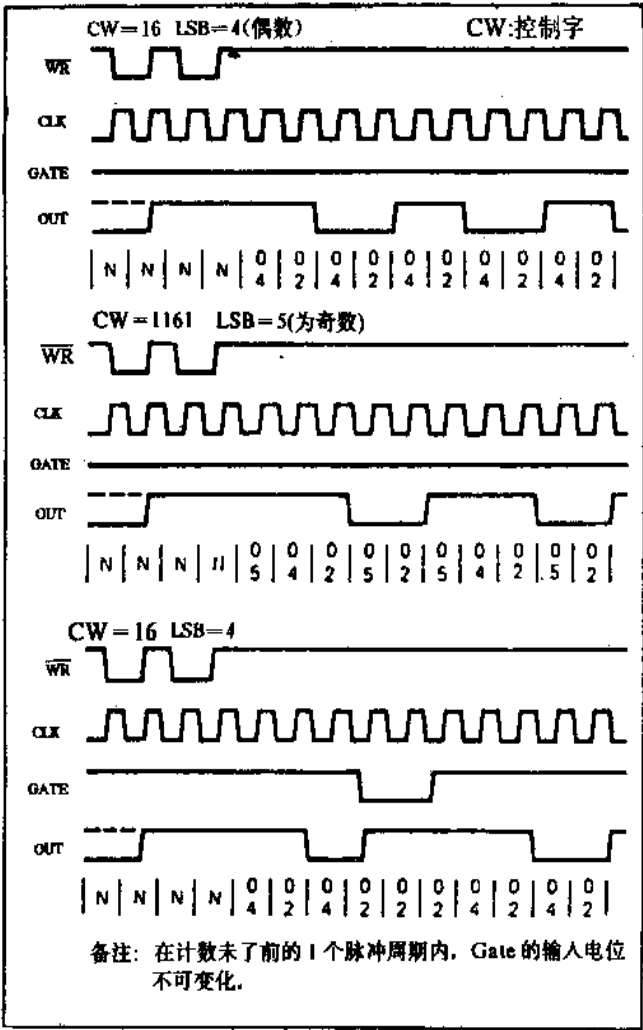


图 7.7 模式 3 的时序图

#### 7.4.5 第 4 种形式: 软件触发式 (Software Triggered Strobe)

这个模式和模式 0 非常类似, 只是 OUT 输出不同而已, 且模式 0 为电位输出, 而模式 4 为负脉冲输出形式。

它的工作方式如下:

1. 起始计数值写入计数器, OUT 刚开始为高电平。
2. 计数器开始启动。
3. 计数器的值递减为 1 后, OUT 则变为低电平达一个 CLK 脉冲之久。
4. OUT 又变为高电平。

GATE 为高电平时计数器处于使能状态, 为低电平时, 则处于禁止状态。在写入控

制字与起始计时值后，计时器将在下一个 CLK 脉冲时写入起始计数值，但这一 CLK 脉冲并不影响计数器计数，因此若起始计时值为 N，则 OUT 会在起始计数值写入后的第 N+1 个 CLK 脉冲，才变为低电平。若初始计数值在计数过程中改变，则下一个 CLK 脉冲时新的计数值被写入，且计数会由新的计数值开始。若写入的是双字节，则写入第一个位时对计数过程无影响，写入第二个字节时将使新的计数值在下一个 CLK 脉冲时写入。这种方式使得这种计数可以被软件重复触发。

图 7.8 为模式 4 的时序图。

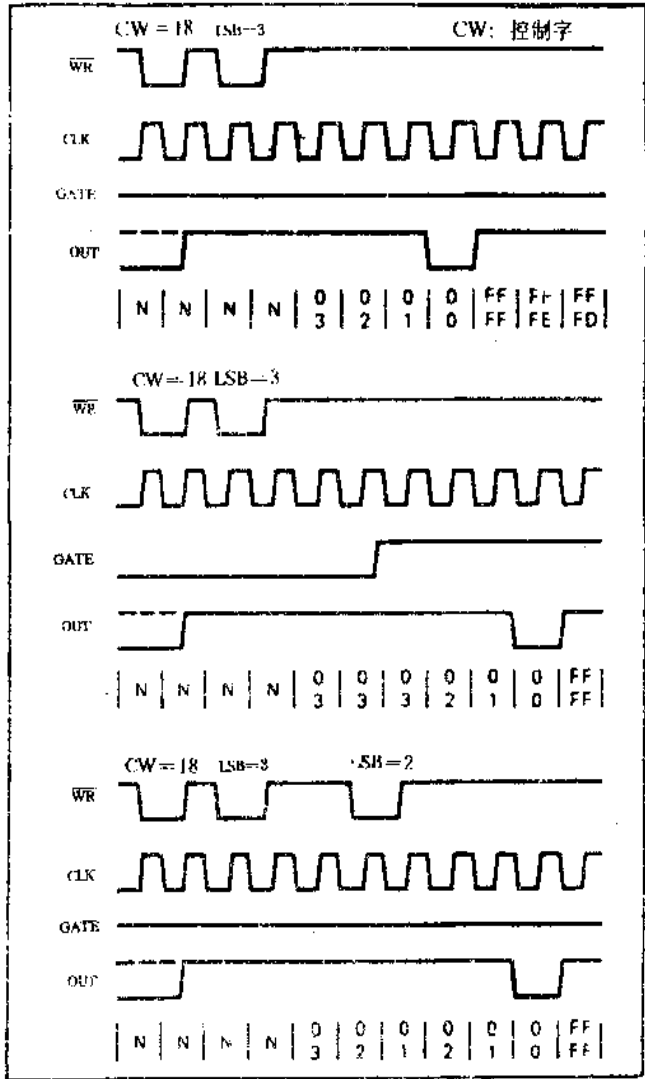


图 7.8 模式 4 的时序图

#### 7.4.6 第 5 形式：硬件可重复触发式 (Hardware Triggered Strobe)

除了计时器由 GATE 信号触发而不由写入起始值立即触发之外，第五种形式和第四种形式类似。刚开始时 OUT 为高电平，直到 GATE 的上升沿启动计时器后，计数才开始，在计时器递减为 1 时，OUT 会变为低电平，过一个 CLK 脉冲之后变为高电平。

在写入控制字与起始计数值后，计数元件会一直等到触发之后的 CLK 脉冲时才将计数值写入，但此 CLK 脉冲并不启动计时器，因此若起始计数值为 N，则 OUT 将在触发之后的第 N+1 个 CLK 脉冲时变为低电平。

整个计数过程是可以重新触发的。每一次触发后起始计数值会在下一个 CLK 脉冲时才写入计时器。若起始计数值在计时器计数时写入，现有的计数过程将不受影响。若触发在新计数值写入之后，且在现有计数值计满(expired)之前发生，新的计数值即在下次 CLK 脉冲时被写入计时器内，新的计时即由此开始。

图 7.9 为模式 5 的时序图。

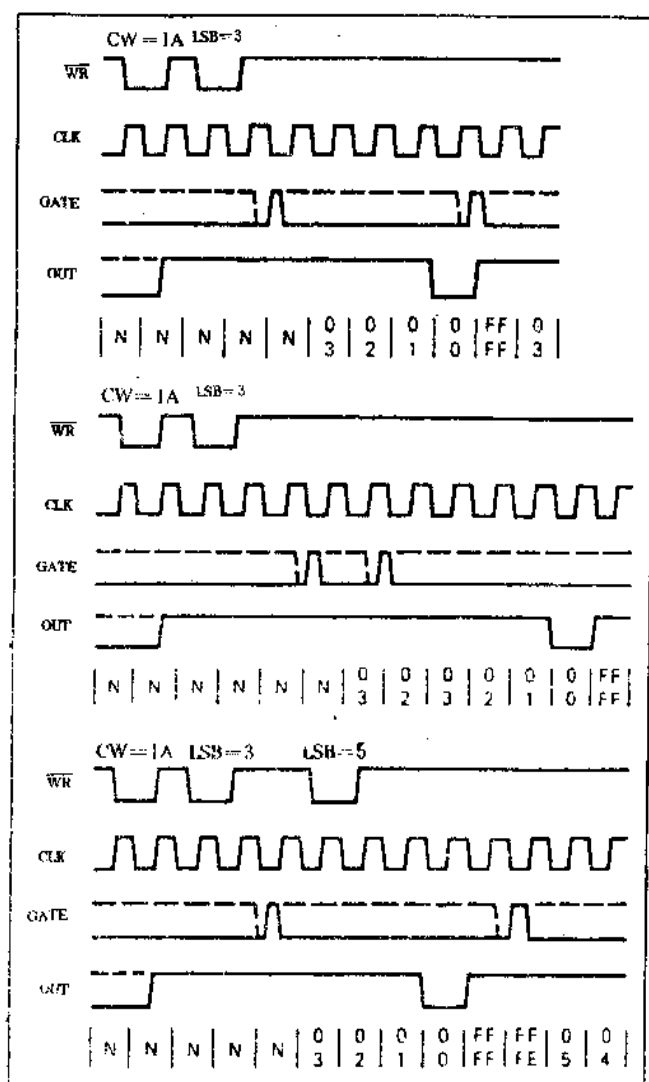


图 7.9 模式 5 的时序图

#### 7.4.7 所有形式都相同的操作方式

编程:

当计时器被重新编程时，也就是新的控制字写入计时器之时，所有的计时器逻辑线路

会被复位(RESET)。OUT 输出信号的状态依所编程的模式编程而决定。

**GATE:**

在这六种模式时 GATE 的输入均在 CLK 上升缘时采样(sampled)。在第 0,2,3 与第 4 形式时, GATE 输入为电位触发。也就是逻辑电位(Logic level)在 CLK 信号的上升缘被采样(sampled)。在第 1,2,3 和 5 时, GARE 的上升缘会使计时器内一边缘触发的正反器的值设置为 1。此触发器的值会在 GATE 信号被采样之后立刻复位。这样无论触发何时发生, 都会被检测到。也就是说, GATE 的高电平电位不需一直维持至 CLK 的下一次上升缘为止。

表 7.3 为 GATE 信号的摘要整理。

表 7.3 GATE 信号的摘要表

形式	GATE 低电平或变为低电平	GATE 上升沿	GATE 高电平
0	计数禁止	—	计数使能
1	—	1. 计数值重新写入 2. 在下一时序后令输出清除	—
2	1. 计数禁止 2. 输出定为高电平	计数重新开始	计数使能
3	1. 计数禁止 2. 输出定为高电平	计数重新开始	计数使能
4	计数禁止	—	计数使能
5	—	计数重新开始	—

**计时器:**

新的计数值在 CLK 下降沿时才可被写入计时器及启动。计时器的值最大可被设置为 0, 亦即  $2^{16}$  或 BCD 计数器  $10^4$ 。在计时器递减为 0 之后, 计数并不停止, 此时它会随工作形式有所不同。在第 0, 1, 4, 5 形式时, 计时器的值会由 0 变为 16 进制的 FFFF 或十进位的 9999, 且继续计时下去。第 2 及第 3 形式则为周期性的, 计时器会再次写入起始计数值并继续计数。每一计时器所能写入的最大及最小值与工作形式有关, 以表 7.4 说明。

表 7.4 各个工作形式的最大及最小的计数初始值

形式	最小值	最大值
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

思考: 为什么最大值是 0? 0 代表什么意义?

## 7.5 操作编程

在开机后, 8254 处于未定的状态, 工作模式、计时器的状态在使用前都必须加以编程。

### 7.5.1 初值设置

计时器的编程包含了初值的设置及计时器工作模式的选取两部分。一般来说, 工作的编程程序非常灵活, 但必须记住两个原则。

1. 每一个计时器的控制字都必须在起始计数值之前写入。
2. 16 位的起始计数值必须遵照控制字中的格式编程。

由于每个控制字及计数寄存器都有个别不同的地址, 因此每一个计时器均可由适当的控制字寄存器个别选取, 不需有任何特定的顺序。

若计时器已先编程成读/写双字节的计数值, 则在写入第一及第二字节之间, 程序控制即不应转移至另一个写入同一计数器的程序上。否则读/写操作即造成不正确的结果。

每次在所有控制字写入计时器时, 被写入的计时器的所有控制电路即会被复位为 0 (不须有任何 CLK 脉冲)。同时对应的输出也会处于已知的起始状态。

### 7.5.2 如何读取计数器的值

当计时器正在工作时, 计数值一直减少, 我们可以用简单的三种方式读取工作中的计数器值及状态。一、读取计数寄存器, 二、计数器锁住命令, 三、回读指令。分别说明如下:

1. 读取计数寄存器。计时器的计数值可经过读取其计数寄存器而获得。仅这一读取操作有一限制: 它须利用其它线路将计时器的 CLK 禁止。不然, 在读取时可能此时的计数值也正在改变, 这样读出的值将不确定。如果 3 个计时器共使用一个 CLK 信号, 令 CLK 禁止的结果将使其它两个计时器也停止。

2. 计数器锁住指令。这个类似于控制字的指令可以写入控制字寄存器中, 用以锁住计数器。此时 D5 及 D4 必须为 0, 下表为这个指令的编程操作形式。

$A1, A0 = 1, 1$ ;  $\overline{CS} = 0$ ;  $RD = 1$ ;  $WR = 0$

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	×	×	×	×

SC1, SC0 : 用以选择计数器

SC1	SC0	计数器
0	0	0
0	1	1
1	0	2
1	1	读回指令

注意: ×表示可为0或1(Don't care)

[例] 利用计数器锁住指令读取第二计数器的值。此操作须由三个步骤完成。

步骤 1.

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

表示第二计时器被选定

步骤 2.

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	0	0	0	0	0

表示第二计时器的值被锁住

步骤 3.

系统读取计时器的值

在执行计数器锁住指令时,被选取的计时器的输出锁存器(OL)会在计时器计数到此指令时将计数值锁住。然后该计数值即被存在锁存器内,直至它被系统读取或重新编程为止。在系统将被锁住的值读取之后,紧接着计数值即会自动变为未锁住,且OL又去追踪其它计数元件的内容。若要锁住一个以上的计时器时,必须使用多个计时器锁住命令。

若计时器被锁定,在锁定的值尚未被读取之前又被锁定一次,则第二次的锁定即属无效。若计时器编程为双字节,则两个字节均须被读取,不过两个字节可以不须连续被系统读取,在二次读取之间系统可以插入一些其它的读/写程序编程。

计时器锁住指令的另一特色:同一计时器的读取及写入可以交错进行。譬如,若计时器编程为双字节,则下行的读取/写入可以如下列方式:

- (1) 读取低位字节
- (2) 写入新的低位字节
- (3) 读取高位字节
- (4) 写入新的高位字节

3. 回读指令。是一种能读取某一计时器的计数数值及状态的指令,它的指令形式如

图 7.10。

$A1, A0 = 1; CS = 0; RD = 1; WR = 0$

D7	D6	D5	D4	D3	D2	D1	D0
1	0	COUNT	STATUS	CNT2	CNT1	CNT0	0

D5 :0 锁住计时器的计数值

D4 :0 锁住计时器的状态

D3 :1 选择第 2 计时器

D2 :1 选择第 1 计时器

D1 :1 选择第 0 计时器

D0 :0 必须为 0, 保留供未来使用

图 7.10 回读指令格式



回读指令利用 D5=0 可以同时锁住 0~2 个不同计时器的计数值。计时器的锁定由 D1, D2, D3 决定。锁定值由输出锁存器(OL)锁住, 直到 OL 值被系统读取之后才会消失。回读指令与锁住命令最大的差别是, 锁住命令一次只能锁住一个计时器, 而回读指令可以同时锁住一个, 两个或全部计时器的内容, 若连续下达多次的回读指令, 只有第一次的命令是有效的, 也就是说, 每下达一次回读命令就必须读一次。

回读指令也可锁住状态 (利用 D4=0), 图 7.11 为回读指令的状态格式。

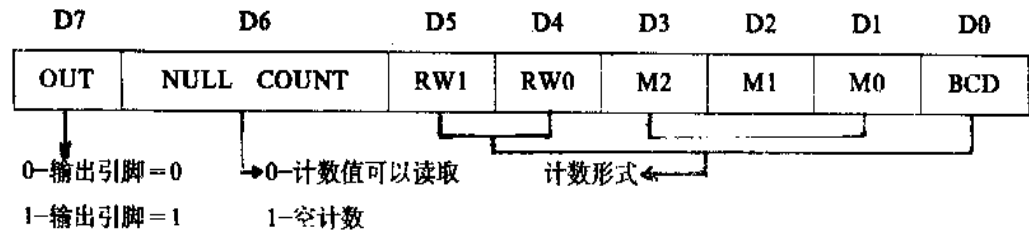


图 7.11 回读指令的状态格式

- D7: OUT=0 表示OUT=0  
 =1 表示OUT=1
- D6: Null Count=0 表示计数值可以读取  
 =1 表示空计数
- 以下三种动作可以影响 Null Count:
1. 写至控制寄存器, 则 Null Count=1
  2. 写至计数寄存器 CR, 则 Null Count=1
  3. 新的计数值存入计数元件(CR→CE), 则 Null Count=0
- 回读指令有不同的读回方式, 参考图 7.12.

D7	D6	D5	D4	D3	D2	D1	D0	操作描述	操作运行结果
1	1	0	0	0	0	1	0	读回第 0 计时器的值及状态	第 0 计时器的值及状态被锁住
1	1	1	0	0	1	0	0	读回第 1 计时器状态	第 1 计时器的状态被锁住
1	1	1	0	1	1	0	0	读回第 1,2 计时器的状态	第 1,2 计时器的状态被锁住
1	1	0	1	1	0	0	0	读回在第 2 计时器的值	第 2 计时器的值被锁住
1	1	0	0	0	1	0	0	读回第 1 计时器的值及状态	第 1 计时器的值及状态被锁住
1	1	1	0	0	0	1	0	读回第 1 计时器的状态	第 1 计时器的状态被锁住

图 7.12

### 7.6 思考题

1. 8254 有哪六种工作形式, 各有何特色? 各适用于何处?
2. 打开您的 PC, 指出何处为 8254 或 82380。检查它们的引脚, 各走向何处。

## 第八章 8253 / 8254 的基本应用示例

### 本章学习目的

1. 读者学习了8253 / 8254的基本理论之后, 经过本章的示例可以对8253 / 8254的应用得到更深一层的了解。
2. 8253 / 8254 共有六种工作模式, 本章将分别以实例说明全部的工作模式。

### 本章内容

- 8.1 8253 / 8254 工作模式 0 的应用
- 8.2 8253 / 8254 工作模式 0, 观察计时器值读回情形
- 8.3 8253 / 8254 工作模式 1
- 8.4 8253 / 8254 工作模式 2
- 8.5 8253 / 8254 工作模式 3
- 8.6 8253 / 8254 工作模式 4
- 8.7 8253 / 8254 工作模式 5

### 8.1 8253 / 8254 工作模式 0 的应用

工作目标:

将计时器 1, 依第 0 种工作模式编程。初始计数值存 0820H。

硬件设计:

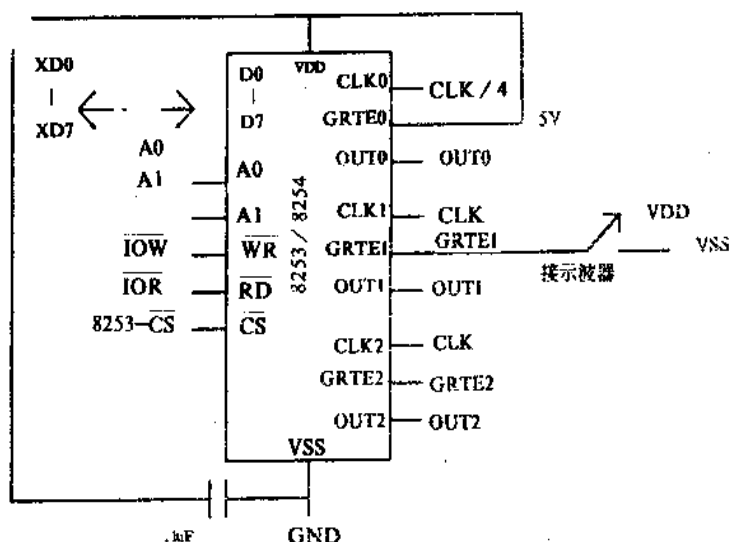


图 8.1 示例 8.1 接线图

### 硬件说明:

此种工作模式中, GATE1 必须先接高电平, 在示波器中观察 OUT1 的波形输出情形。

### 软件设计:

#### 1. 编程控制字

D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	1	0	0	0	0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

```
SETPORT=0x3eb;           /* 设置控制寄存器的地址 */
bytewrite=0x70;          /* 第0种模式的选择 */
outportb(setport,bytewrite); /* 同上 */
```

#### 2. 存入计时器的初始值

```
PORTA=0x3c9;             /* 设置第1计时器的地址 */
bytewrite=0x20;
outportb(PORTA,bytewrite); /* 设置计数值低字节 */
bytewrite=0x80;
outportb(PORTA,bytewrite); /* 设置计数值高字节 */
```

3. 完成了计数值的设置, 下一个时序脉冲的下降边沿, 计时器开始倒数计时。

#### 程序示例 ch8\_1.c

8254 模式 0 的基本应用。

```
/* ----- */
/*      Program Name : ch8_1.c      */
/*      For 8254, Mode 0 application */
/* ----- */
#include <dos.h>
#include <conio.h>
#define SETPORT 0x3eb
#define PORTA 0x3c9
void main()
{
    unsigned char bytewrite;

    /* set up 8254 mode 0 */
    bytewrite = 0x70;
    outportb(SETPORT,bytewrite);

    /* write initial value to COUNTER 1 */
```

[. 8255A 的 PORTA 为 8255A]

```

bytewrite = 0x20;
outputb(PORTA,bytewrite); /* output lower byte */
bytewrite = 0x08;
outputb(PORTA,bytewrite); /* output upper byte */
}

```

思考:

1. 改变计时器 1 及 2 的选择。
2. 改变 RL1、RL0 的选择, 观察其影响。
3. 将 BCD 位改为 1, 观察结果。
4. 在倒数计数期间将 GATE 值由 VDD 改为 VSS, 观察结果。

## 8.2 8253/8254 工作模式 0, 观察计时器值读回情形

工作目标: 观察计时器值读回系统的情形

1. 将计时器依第 0 种模式倒数计时。
2. 编程计时器, 将计时器值读回系统。
3. 将此值输出至 8255A 的 PORTA, 以便于观察。

硬件设计:

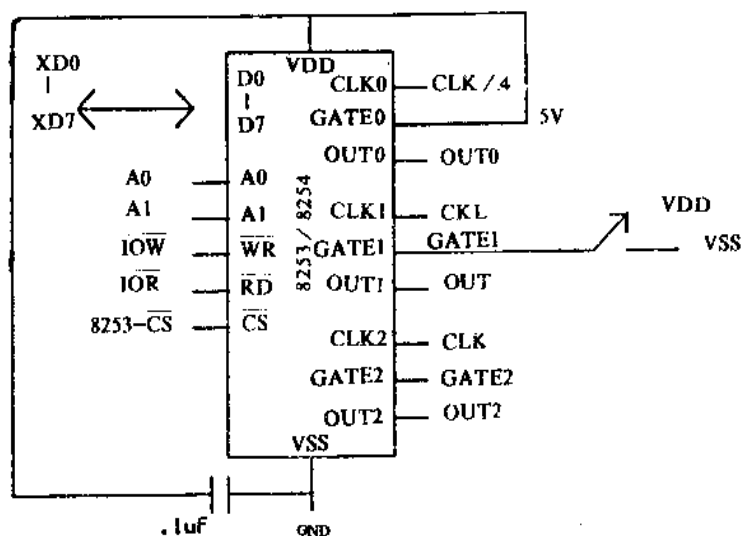


图 8.2 示例 8.2 接线图

硬件说明:

在 8253/8254 中, CLK1 接 PC 的系统, 由于各种机型的 PC 系统时钟不同, 请读者以示波器测量所使用系统的周期。GATE1 为计时器倒数计时的开关, 当 GATE1 = V<sub>DD</sub> 时, 系统进行倒数计时工作, 反之系统则停止计时。

在 8255A 中, 系统将所读取的计数值输出至 PORTA 上。

软件设计:

1. 编程控制字

```

SETPORT=0x3eb;      /* 设置控制寄存器的地址 */
bytewrite=0x70;     /* 第0种模式的选择 */
outportb(SETPORT,bytewrite);

```

## 2. 存入计时器的初始值

```

PORTA=0x3e9;        /* 设置第1计时器地址 */
bytewrite=0xff;     /* 设置计时器低字节 */
outportb(PORTA,bytewrite);
bytewrite=0xff;     /* 设置计时器高字节 */
outportb(PORTA,bytewrite);

```

下一个时序脉冲的下降边缘，计时器开始倒数计时。

## 3. 编程 8255A

```

PORT=0x3c3;         /* 编程输出口 */
bytewrite=0x80;
outportb(PORT,bytewrite);

```

## 4. 读回计时器的计数值

```

PORTA=0x3e9;
byteread=inportb(PORTA)

```

## 5. 输出至 PORTA

```

PORT=0x3c1;
bytewrite=byteread;
outportb(PORT,bytewrite);

```

## 程序示例 ch8\_2.c

观察计时器值读回的基本应用。

```

/* ----- */
/*      Program Name : ch8_2.c      */
/*      For 8254, Mode 0 application */
/* ----- */
#include <dos.h>
#include <conio.h>
#define SETPORT 0x3cb
#define PORTA 0x3e9
void main()
{
    unsigned char byteread, bytewrite;
    int PORT;

    /* set up 8254 mode 0 */

```

```

    bytewrite = 0x70;
    outportb(SETPORT,bytewrite);

/* write initial value to COUNTER 1 */
    bytewrite = 0xff;
    outportb(PORTA,bytewrite); /* output lower byte */
    bytewrite = 0xff;
    outportb(PORTA,bytewrite); /* output upper byte */

/* Set up 8255 */
    bytewrite = 0x80;
    PORT = 0x3e3;
    outportb(PORT,bytewrite);

/* read data from counter */
    while ( !kbhit() )
    {
        byteread = inportb(PORTA);

        PORT = 0x3e1;
        bytewrite = byteread;
        outportb(PORT,bytewrite);
    }
}

```

思考:

是否一定要将 GATE1 接成“0”才可读取计时器? 可否有其它方式?

提示: 本书前一章曾提及回读指令。

D7	D6	D5	D4	D3	2	D1	D0
SC1	SC0	0	0	x	x	x	x

将 D5, D4 定为 0,即为回读指令形式。如以下说明:

```

SETPORT=0x3eb;          /* 设置控制寄存器 */
bytewrite=0x00;         /* 回读指令 */
outportb(SETPORT,bytewrite);

PORT=0x3e9;             /* 设置第1计时器 */
byteread = inportb(PORT); /* 读回数据 */

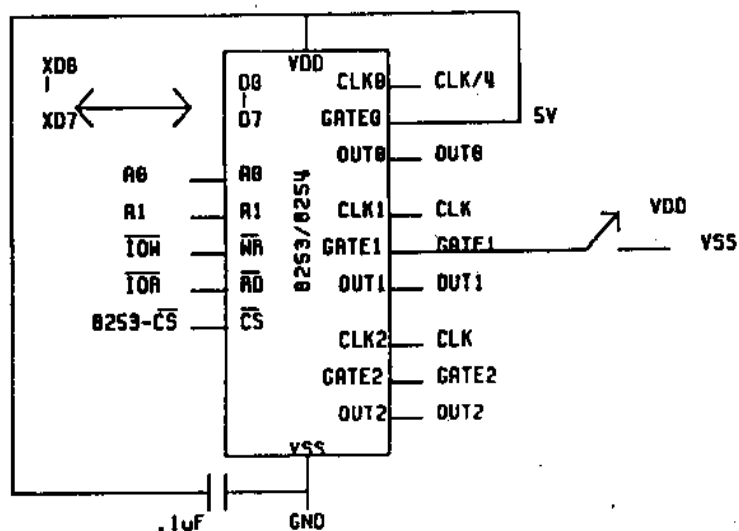
```

### 8.3 8253/8254 工作模式 1

### 工作目标:

**将计时器 1 依第一种工作模式编程，计数值存 1959H。**

### 硬件设计:



**图 8.3 示例 8.3 接线图**

读者接示波器观察输出波形变化情形。

在这种模式之下 GATE1 必须先接为 VSS.

### 软件设计:

### 1. 编程控制字

D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	1	0	0	1	0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

```
SETPORT=0x3cb;          /* 设置控制寄存器的地址 */
bytewrite=0x72;          /* 第1种模式的选择 */
outportb(SETPORT,bytewrite);
```

## 2. 写入计时器的初始值

```
PORTA=0x3c9;          /* 设置第1计时器的地址 */
bytewrite=0x59;        /* 低字节值 */
outportb(PORTA,bytewrite);
bytewrite=0x19;         /* 高字节值 */
outportb(PORTA,bytewrite);
```

3. 将GATE1由VSS接至VDD, 再返回VSS, 此动作称为单触发(One-shoot), OUT1 会在 CLK1 脉冲时变为低电平, 直至计时器值变为 0。

4. 以示波器观察 OUT1 的输出情形。

程序示例 ch8\_3.c

基本 8254 工作模式 1 的应用。

```

/* ----- */
/*          Program Name : ch8_3.c          */
/*      For 8254, Mode 1 application      */
/* ----- */

#include <dos.h>
#include <conio.h>
#define SETPORT    0x3eb
#define PORTA      0x3c9
void main()
{
    unsigned char bytewrite, PORT;

    /* set up 8254 mode 1 */
    bytewrite = 0x72;
    outportb(SETPORT, bytewrite);

    /* write initial value to COUNTER 1 */
    bytewrite = 0x59;
    outportb(PORTA, bytewrite); /* output lower byte */
    bytewrite = 0x19;
    outportb(PORTA, bytewrite); /* output upper byte */
}

```

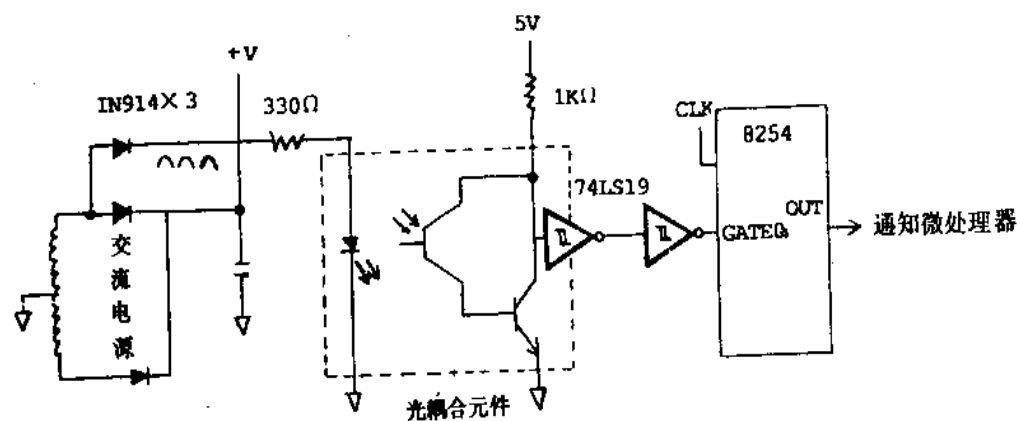


图 8.4 模式 1 的应用实例 (断电检测器)

这种模式最大的特色即是在每一次触发完后, 计数器的值会重新写入原始值。换句话



说, 如果我们不断地将 GATE 由低电平改变为高电平, 再变低电平, 高电平……, 此时原始计数值会不停地写入计数器内, OUT 的状态也变不了“1”。

读完上述讨论, 我们应如何应用呢? 最常见的就是“断电检测线路了”。

图 8.4 为模式 1 的应用实例: 断电检测器。

工作原理:

1. 在未断电时, 60Hz 的周期波经整形后会产生一系列的周期波, 这个周期波会送入 GATE0。
2. 82C54 因为一直收到连续的波, 起始值一直不断地重新写入计时器, 而 OUT 的状态不会变成“1”, 表示电流不断。
3. 倘若断电了, GATE0 变为 0, 82C54 开始正常工作, 一段时间后 OUT 变为 1, 通知微处理器。

综合以上说明, 82C54 可以准确地在交流电源断电时立即测知。

思考:

1. 如何为以上的检测器设计一个软件。

## 8.4 8253/8254 工作模式 2

工作目标:

将计时器 2, 依第 2 种工作模式编程计数值放 03H。

硬件设计:

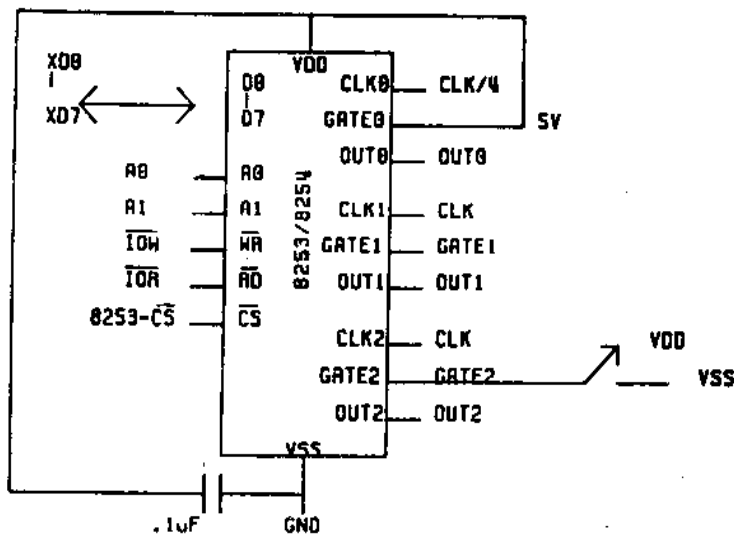


图 8.5 示例 8.4 接线图

在这种模式下, GATE 必须先接至 VDD。

软件设计:

1. 编程控制字

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	1	X	1	0	0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

M2:放 1 或 0 均可

```

SETPORT=0x3eb;          /* 设置控制寄存器的地址 */
bytewrite=0x94;          /* 第2种模式的选择 */
outportb(SETPORT,bytewrite);

```

## 2. 写入初始值

```

PORTC=0x3ea;            /* 设置第2计时器的地址 */
bytewrite=0x03;          /* 计算值为3 */
outportb(PORTC,bytewrite);

```

## 3. 在示波器观察 OUT 情形

理论上 OUT 会在一段时间后输出长达一个波长宽度的波。

这种模式最常见的应用即是 PC 系统的时间中断。譬如 8253 / 8254 定时（如 1 / 20 秒）中断系统，当中断 20 次之后，PC 系统即据此修正系统时间。这种模式下 OUT 的输出为连续不断的定时输出，所以只需要编程一次，即可永不间断提供系统时间。在 PC 系统中通常配有电池，所以系统时间不会因为计算机开机而受影响。

当 GATE 为低电平时，计时器暂停计时，此时可以校正系统时间。当 GATE 为高电平时，又继续计时。

思考：

参考使用 PC 系统的手册，找出系统周期波的频率，编程 8253 / 8254，计算出如果要每秒中断一次系统，计时器的初始值应存入何值？

### 程序示例 ch8\_4.c

8254 工作模式 2 的基本应用。

```

/* ----- */
/*          Program Name : ch8_4.c          */
/*      For 8254, Mode 2 application          */
/* ----- */
#include <dos.h>
#include <conio.h>
#define SETPORT    0x3eb
#define PORTC      0x3ea
void main()
{
    unsigned char bytewrite, PORT;

    /* set up 8254 mode 2 */

```

```

    bytewrite = 0x94;
    outportb(SETPORT,bytewrite);

/* write initial value to COUNTER 2 */
    bytewrite = 0x03;
    outportb(PORTC,bytewrite);
}

```

## 8.5 8253/8254 工作模式 3

工作目标:

将计时器 2, 依第 3 种工作编程, 计数值放 0188H.

硬件设计:

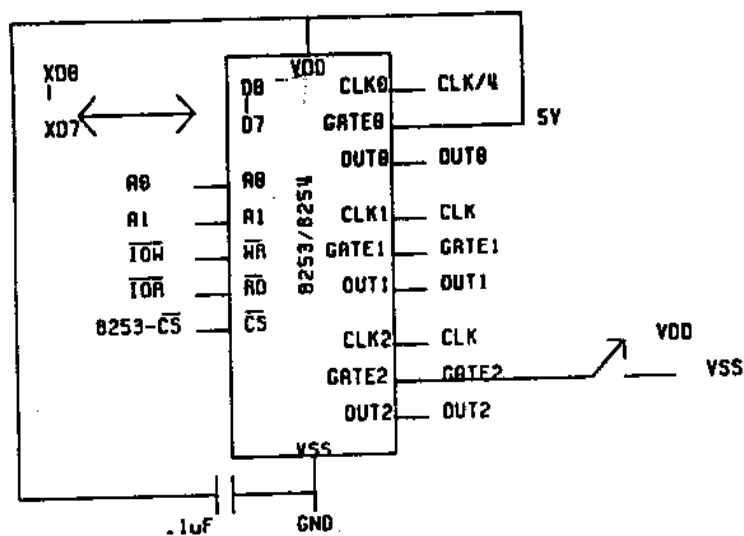


图 8.6 示例 8.5 接线图

软件设计:

### 1. 编程控制字

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	X	1	1	0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

M2:放 1 或 2 均可.

```

SETPORT=0x3eb;          /* 设置控制寄存器的地址 */
bytewrite=0xb6;          /* 第3种模式的选择 */
outportb(SETPORT,bytewrite);

```

### 2. 写入初始值

```

PORTC=0x3ea;          /* 设置第0计时器的地址 */
bytewrite=0x88;        /* 低字节值 */
outportb(PORTC,bytewrite);
bytewrite=0x01;        /* 高字节值 */
outportb(PORTC,bytewrite);

```

### 3. 在示波器上观察 OUT

理论上我们可以看到方波。

这种模式可检查波特率(Baud Rate)。

思考:

1. 在前一章中讨论到起始数值是偶数及奇数操作方式有所不同, 请读者输入偶数及奇数值在示波器上观察不同波形, 并与前一章的内容比较。
2. GATE在高电平时可使计时器使能, 在低电平时使计时器禁止, 请证明。

#### 程序示例 ch8\_5.c

8254 工作模式 3 的基本应用。

```

/* ----- */
/*          Program Name : ch8_5.c          */
/*      For 8254, Mode 3 application      */
/* ----- */

#include <dos.h>
#include <conio.h>
#define SETPORT    0x3eb
#define PORTC      0x3ea
void main()
{
    unsigned char bytewrite, PORT;

    /* set up 8254 mode 3 */
    bytewrite = 0xb6;
    outportb(SETPORT,bytewrite);

    /* write initial value to COUNTER 2 */
    bytewrite = 0x88;
    outportb(PORTC,bytewrite); /* output lower byte */
    bytewrite = 0x01;
    outportb(PORTC,bytewrite); /* output higher byte */
}

```

## 8.6 8253/8254 工作模式 4

工作目标:

将计时器 2, 依第 4 种工作模式编程, 初始计数值放 1212H。

### 硬件设计:

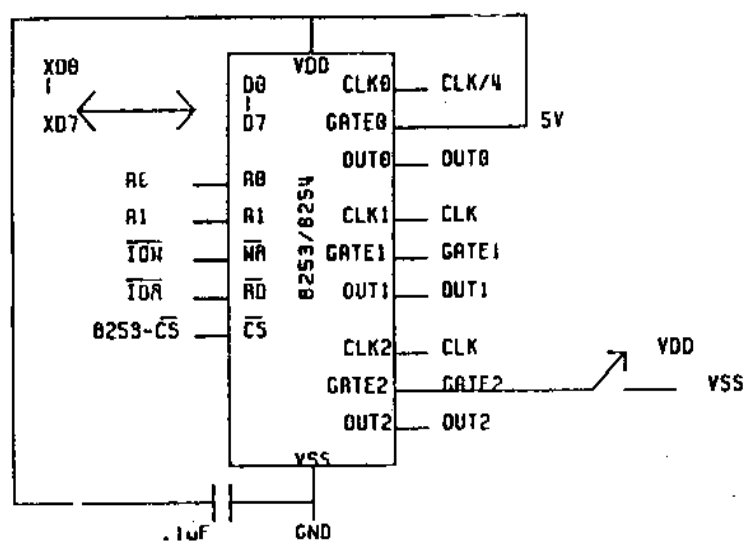


图 8.7 示例 8.6 接线图

请读者接示波器观察输出波形变化情形。在这种模式下，GATE 必须先接 VDD。

**软件设计:**

## 1. 编程控制字

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1	0	0	0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

```
SETPORT=0x3cb; /* 设置控制寄存器的地址 */
```

```
bytwrite=0xb8; /* 第4种模式的选择 */
```

```
outportb(SETPORT,bytewrite);
```

## 2. 写入计算器的初始值

```
PORTC=0x3ea; /* 设置第2计时器的地址 */
```

```
bytewrite=0x12; /* 低字节值 */
```

```
outportb(PORTC,bytewrite);
```

```
bytestrite=0x12; /* 高字节值 */
```

```
outportb(PORTC,bytewrite);
```

### 3. 观察 OUT 波形

### 思考题:

1. 将起始值如模式4的时序图定为3(LSB=3), 在纸上根据示波器的结果画出, 并与时序图比较。
2. 这种工作方式在何种应用下可以派上用场?

### 程序示例 ch8\_6.c

### 8254 工作模式 4 的基本应用:



请读者接示波器观察输出波形变化情形。在这种模式下，GATE 必须先接 VSS。

软件设计：

### 1. 编程控制字

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1	0	1	0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

```

SETPORT=0x3eb;          /* 设置控制寄存器的地址 */
bytewrite=0xba;         /* 第5种模式的选择 */
outportb(SETPORT,bytewrite);
PORTC=0x3ca;            /* 设置第2计时器的地址 */
bytewrite=0x12;         /* 低字节值 */
outportb(PORTC,bytewrite);
bytewrite=0x12;         /* 高字节值 */
outportb(PORTC,bytewrite);

```

### 2. 写入计算器的初始值

```

PORTC=0x3ca;            /* 设置第2计时器的地址 */
bytewrite=0x12;         /* 低字节值 */
outportb(PORTC,bytewrite);
bytewrite=0x12;         /* 高字节值 */
outportb(PORTC,bytewrite);

```

3. 以手动调 GATE 启动计时器，用示波器观察它。

思考题：

1. 在纸上划出示波器的结果（示波器两个频率，一个接GATE0,另一个接OUT0）。

与时序图比较。

2. 有哪一个应用可以用上此种模式？

程序示例 ch8\_7.c

8254 工作模式 5 的基本应用。

```

/* ----- */
/*          Program Name : ch8_7.c          */
/*      For 8254 Mode 5 application          */
/* ----- */
#include <dos.h>
#include <conio.h>
#define SETPORT 0x3eb
#define PORTC 0x3ca
void main()

```

```

{
    unsigned char bytewrite, PORT;

    /* set up 8254 mode 5 */
    bytewrite = 0xba;
    outportb(SETPORT,bytewrite);

    /* write initial value to COUNTER 2 */
    bytewrite = 0x12;
    outportb(PORTC,bytewrite); /* output lower byte */
    bytewrite = 0x12;
    outportb(PORTC,bytewrite); /* output higher byte */
}

```



## 第二篇

# I/O 控制的专题研究

- 第九章 跑马灯专题
- 第十章 红绿灯专题
- 第十一章 家电控制专题
- 第十二章 LED七段显示器专题
- 第十三章 方波发生器专题
- 第十四章 自动售货机专题
- 第十五章 防盗器专题
- 第十六章 数字IC测试器专题
- 第十七章 电梯模拟专题
- 第十八章 加油机专题
- 第十九章 键盘模拟专题
- 第二十章 声音控制 (一) 专题
- 第二十一章 声音控制 (二) 专题

## 第九章 跑马灯专题

### 本章实验目的

1. 通过 8255A 跑马灯的应用，读者可以了解下列知识：

- IBM PC 接口控制
- IBM PC 屏幕位置控制
- 跑马灯的意义及未来的应用与扩充

### 本章内容

- 9.1 工作目标
- 9.2 硬件设计
- 9.3 软件设计
- 9.4 结论
- 9.5 思考题

### 9.1 工作目标

让 LED 按下图的顺序发亮。

T=0    ● ○ ○ ○ ○ ○ ○ ○  
T=t    ○ ● ○ ○ ○ ○ ○ ○  
T=2t    ○ ○ ● ○ ○ ○ ○ ○  
T=3t    ○ ○ ○ ● ○ ○ ○ ○  
T=4t    ○ ○ ○ ○ ● ○ ○ ○  
T=5t    ○ ○ ○ ○ ○ ● ○ ○  
T=6t    ○ ○ ○ ○ ○ ○ ● ○  
T=7t    ○ ○ ○ ○ ○ ○ ○ ●

●：发亮

○：不亮

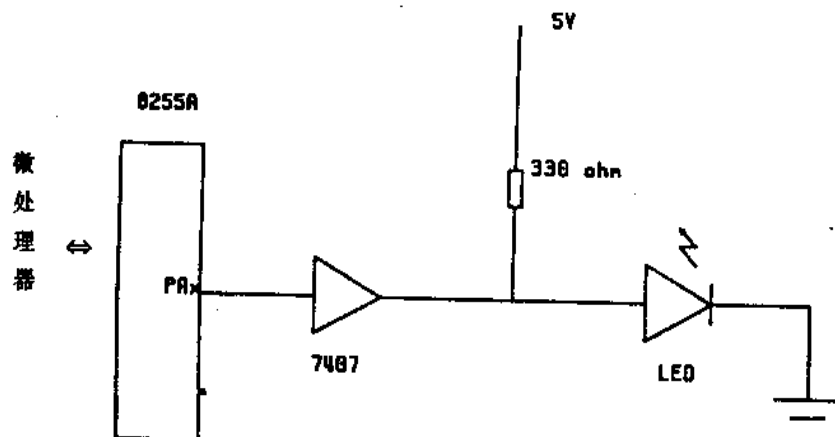
同时读者也可以在计算机屏幕上看到相同的变化。

**LED 工作原理：**

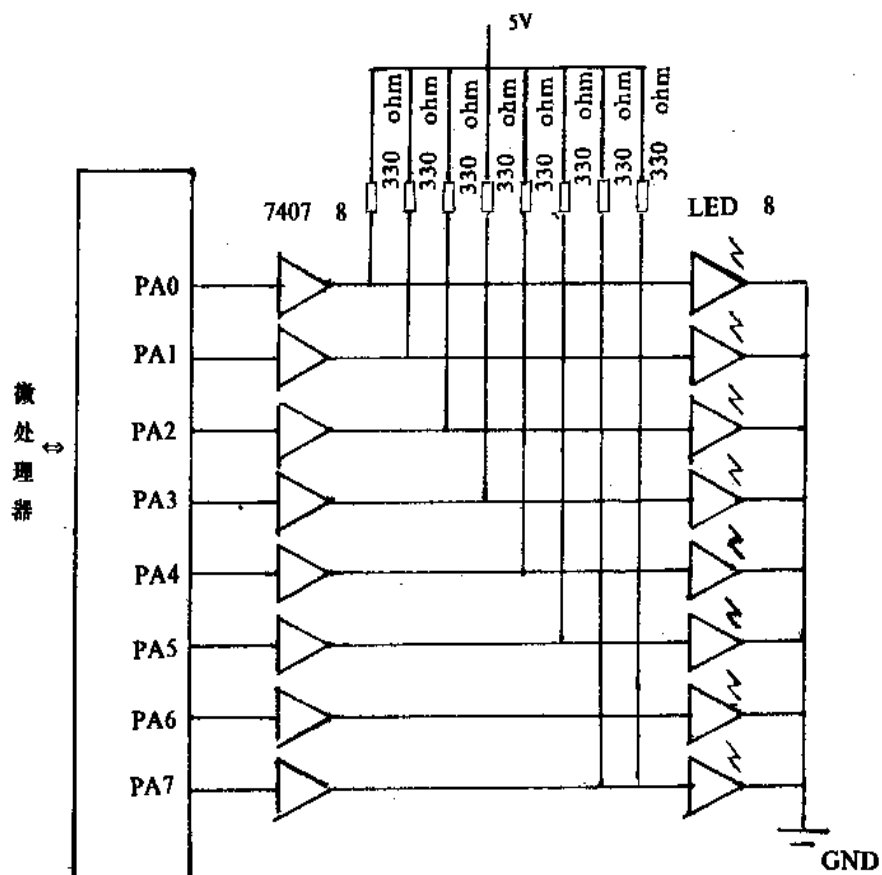
LED 为电子实验中最常见的显示器，常用的有红、绿、黄三种颜色。以下为一个 LED 应用电路（见下页）。

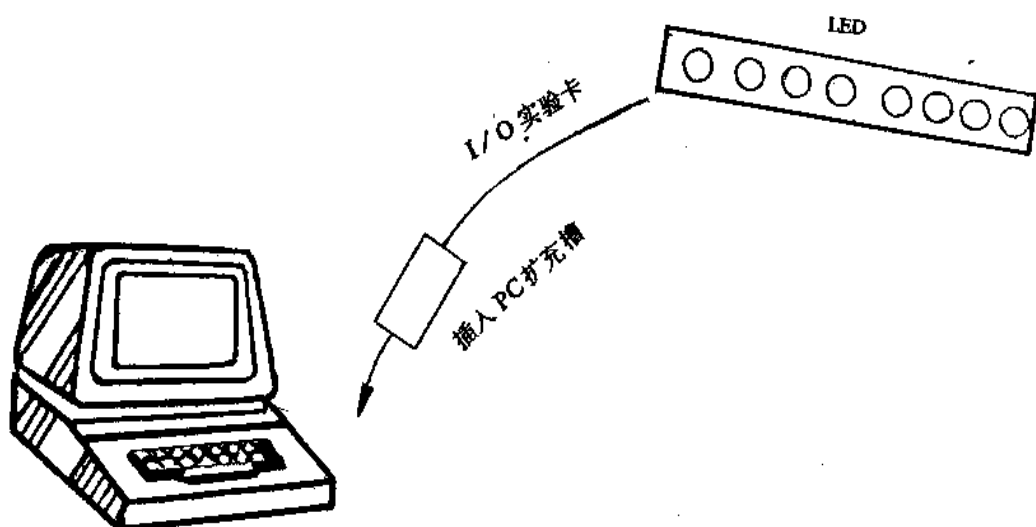
LED 通常需要 10mA 左右的电流才可以驱动，而 8255A 的驱动力有限，所以我们必须加上一个缓冲区 7407 当作驱动的媒介。

当 PAX 为 1 时，LED 即被驱动。



## 9.2 硬件设计





### 9.3 软件设计

#### 程序示例 rlight1.c

简单跑马灯左旋转的应用，本程序在运行时，首先跑马灯的最右边灯会亮，如下所示：



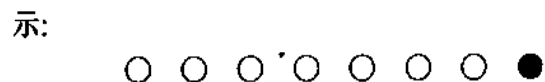
过一秒之后，右边第 2 个灯会发亮，如下所示：



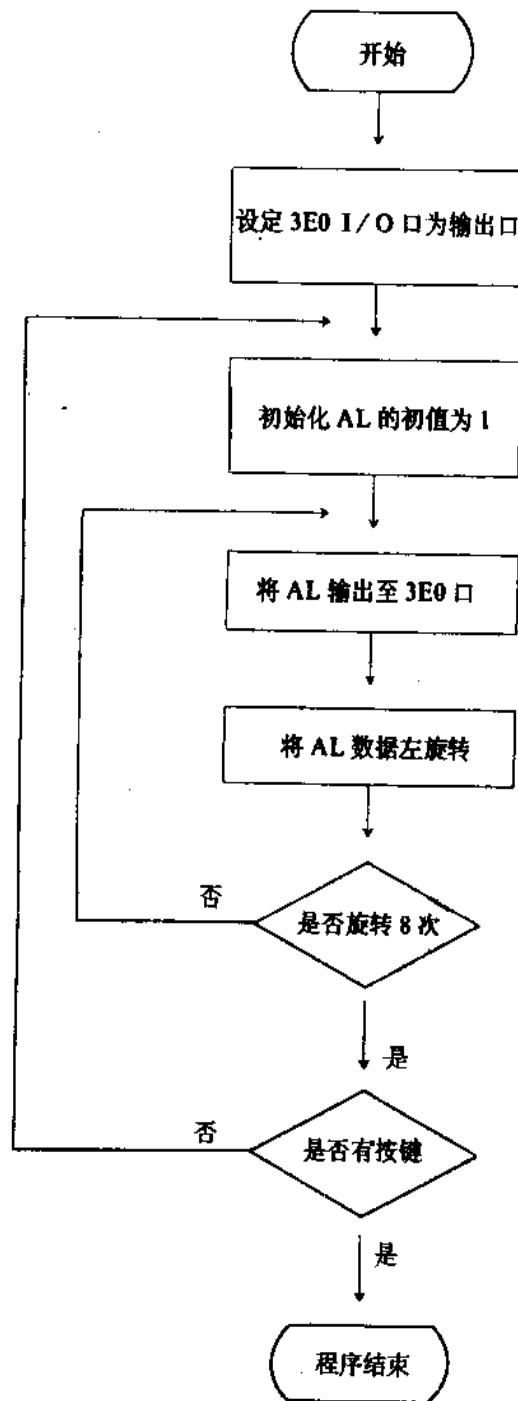
当最左边的灯发亮之后，如下所示：



此时程序检查是否有按键，如果有则程序显示，否则跑马灯最右边的灯会亮，如下所示：



本程序将这样持续运行，本程序流程如下：



```

01 /* ----- */
02 /*      Program Name : right1.c      */
03 /*      Running light application 1    */
04 /*      For 8255A,   Mode 0 application */
05 /* ----- */

```

```

06 #include <dos.h>
07 #define SETPORT      0x3e3
08 #define PORTA        0x3e0
09 void main( )
10 {
11     unsigned char bytewrite;
12     unsigned char light;
13     int i;
14
15     /* set up the output port */
16     bytewrite = 0x80;
17     outportb(SETPORT,bytewrite);
18
19     /* running the light */
20     while ( !kbhit( ) )
21     {
22         light = 1;
23         for ( i = 1; i <= 8; i++ )
24         {
25             outportb(PORTA,light);
26             light = light << 1;
27             sleep(1);          /* delay 1 second */
28         }
29     }
30 }

```

程序示例 RLIGHT1.ASM 解释:

1. 第 7 行是用于设置 SETPORT 值为控制寄存器的口, 其值为 0x3e3.
2. 第 8 行是用于设置 PORTA 口值是 0x3e0.
3. 第 16 行和 17 行是设置 PORTA 口专供输出使用.
4. 第 25 行是将跑马灯的值输出至 PORTA.
5. 第 26 行是向左旋转跑马灯的值.
6. 第 27 行是令时间延迟 1 秒.
7. 第 23 行至 28 行是将跑马灯值输出至 PORTA 的循环.
8. 第 20 行至 29 行是检查是否有键盘输入的循环, 如果有键盘输入发生则程序结束.

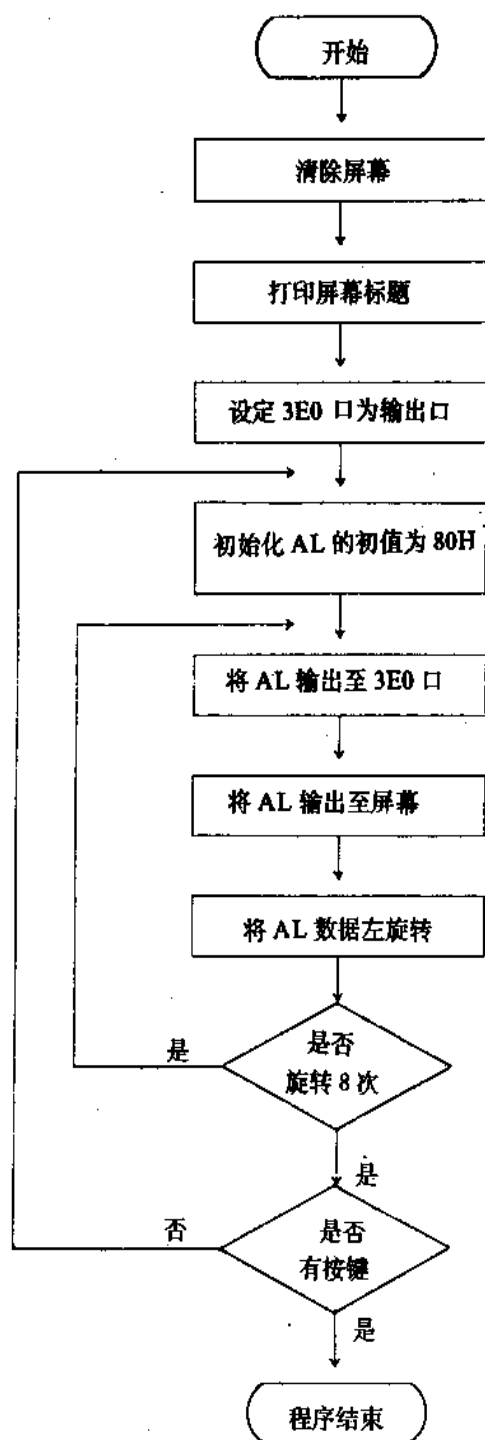
程序示例 rlight2.c

重复前一个程序示例 rlight1.c, 但是本程序的跑马灯在运行时, 各位可同时在屏幕上看到跑马灯左旋转的情形.

本程序在运行时, 除了原先 LED 跑马灯会依次运行外, 首先屏幕将如下所示:

Running Light Application2

00000001



过一秒之后，屏幕将如下所示：

Running Light Application2  
00000010

当最左边的灯发亮之后，屏幕如下所示：

Running Light Application2

10000000

此时程序检查是否有按键，如果有则程序结束，否则跑马灯最右的灯会亮，如下所示：

Running Light Application2

00000001

本程序流程如上图所示。

```
01  /* ----- */
02  /*      Program Name : rlight2.c      */
03  /*      Running light application 2 with screen output      */
04  /*      For 8255A,      Mode 0 application.      */
05  /* ----- */
06  #include <dos.h>
07  #include <conio.h>
08  #define SETPORT 0x3e3
09  #define PORTA 0x3e0
10  void main( )
11  {
12      unsigned char bytewrite;
13      unsigned char light;
14      int i;
15      void display( );
16
17      /* set up the output port */
18      bytewrite = 0x80;
19      outportb(SETPORT,bytewrite);
20
21      clrscr( );
22      gotoxy(1,1);
23      printf("Running Light Application 2");
24
25      /* running the light */
26      while ( !kbhit( ) )
27      {
```



```

28     light = 1;
29     for ( i = 1; i <= 8; i++ )
30     {
31         outportb(PORTA,light);
32         display(light);
33         light = light << 1;
34         sleep(1);          /* delay 1 second */
35     }
36 }
37 }
38 void display(unsigned char light)
39 {
40
41     gotoxy(10,3);
42     printf("%d",(light & 0x80) >> 7);
43     printf("%d",(light & 0x40) >> 6);
44     printf("%d",(light & 0x20) >> 5);
45     printf("%d",(light & 0x10) >> 4);
46     printf("%d",(light & 0x08) >> 3);
47     printf("%d",(light & 0x04) >> 2);
48     printf("%d",(light & 0x02) >> 1);
49     printf("%d",light & 0x01);
50 }

```

程序示例 rlight2.c 解释:

1. 第 8 行是用于设置 SETPORT 值为控制寄存器的口，其值是 0x3e3。
2. 第 9 行是设置 PORTA 口值是 0x3e0。
3. 第 18 行至第 19 行是设置 PORTA 口专供输出使用。
4. 第 21 行是清除屏幕内容。
5. 第 22 行是将光标移至 (1,1) 位置。
6. 第 23 行是输出 "Running Light Application 2" 字符串。
7. 第 31 行是将跑马灯值输出至 PORTA。
8. 第 32 行是调用 display( ) 函数将跑马灯值在屏幕上显示。
9. 第 33 行是将跑马灯的值向左旋转。
10. 第 34 行是令时间延迟 1 秒钟。
11. 第 29 行是将跑马灯值输出至 PORTA 的循环。
12. 第 26 行至 36 行是检查是否有键盘输入的循环，如果有键盘输入则程序结束。

函数 display( ) 解释:

1. 第 41 行是将光标移至 (10,3) 位置。
2. 第 42 行是显示跑马灯的第 7 位。
3. 第 43 行是显示跑马灯的第 6 位。
4. 第 44 行是显示跑马灯的第 5 位。

5. 第 45 行是显示跑马灯的第 4 位。
6. 第 46 行是显示跑马灯的第 3 位。
7. 第 47 行是显示跑马灯的第 2 位。
8. 第 48 行是显示跑马灯的第 1 位。
9. 第 49 行是显示跑马灯的第 0 位。

#### 程序示例 rlight3.c

简单跑马灯右旋转的应用，本程序基本上和程序示例 rlight1.c 相同，只不过 rlight1.c 会造成跑马灯左旋转，本程序会造成跑马灯右旋转。

首先跑马灯最左边灯会亮，如下所示：

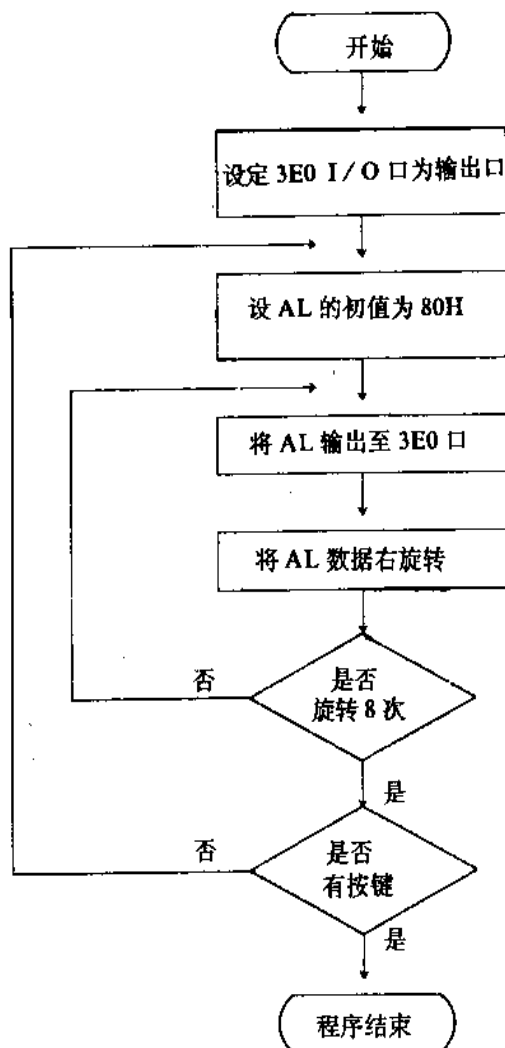
● ○ ○ ○ ○ ○ ○ ○

↑

发亮

过一秒之后，左边第 2 个灯会发亮，如下所示：

○ ● ○ ○ ○ ○ ○ ○



当最右边的灯亮之后, 如下所示:



此时程序检查是否有按键, 如果有则程序结束, 否则跑马灯最左边的灯会亮, 如下所

示:



本程序流程如上页图所示。

程序示例 rlight3.c 解释:

```
01  /* ----- */
02  /*      Program Name : rlight3.c      */
03  /*      Running light application 3    */
04  /*      For 8255A,   Mode 0 application */
05  /* ----- */
06  #include <dos.h>
07  #define  SETPORT    0x3c3
08  #define  PORTA      0x3c0
09  void main( )
10  {
11      unsigned char bytewrite;
12      unsigned char light;
13      int i;
14
15      /* set up the output port */
16      bytewrite = 0x80;
17      outportb(SETPORT,bytewrite);
18
19      /* running the light */
20      while ( !kbhit( ) )
21      {
22          light = 0x80;
23          for ( i = 1; i <= 8; i++ )
24          {
25              outportb(PORTA,light);
26              light = light >> 1; /* right rotate */
27              sleep(1);          /* delay 1 second */
28          }
29      }
30 }
```

1. 第 7 行是用于设置 SETPORT 值为控制寄存器的口, 其值为 0x3c3.
2. 第 8 行是用于设置 PORTA 口值是 0x3c0.
3. 第 16 行至 17 行是设置 PORT 口专供输出使用.
4. 第 25 行是将跑马灯的值输出至 PORTA.

5. 第 26 行是向右旋转跑马灯的值。
6. 第 27 行是令时间延迟 1 秒钟。
7. 第 23 行至 28 行是将跑马灯的值输出至 PORTA 的循环。
8. 第 20 行至 29 行是检查是否有键盘输入的循环，如果有键盘输入发生则程序结束。

#### 程序示例 rlight4.c

重复前一个程序示例 rlight3.c，但是本程序的跑马灯在运行时，各位可同时在屏幕上看到跑马灯右旋转的情形。

本程序在运行时，除了原先 LED 跑马灯会依次运行外，首先屏幕将如下所示：

```
Running Light Application 4
10000000
```

过一秒之后，屏幕将如下所示：

```
Running Light Application 4
01000000
```

当最右边的灯亮之后，屏幕如下所示：

```
Running Light Application 4

00000001
```

此时程序检查是否有按键，如果有则程序结束，否则跑马灯最左边的灯会亮。如下所示：

```
Running Light Application 4
10000000
```

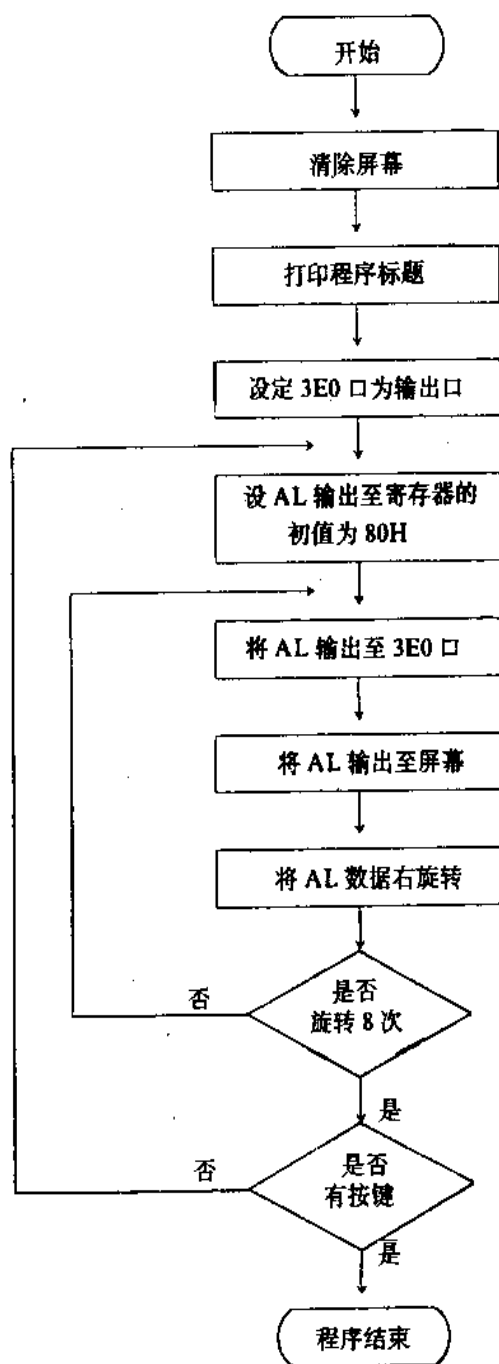
```
01  /* ----- */
02  /*      Program Name : rlight4.c      */
03  /*      Running light application 4 with screen output      */
04  /*      For 8255A,      Mode 0 application      */
05  /* ----- */
06  #include <dos.h>
07  #include <conio.h>
08  #define  SETPORT      0x3c3
09  #define  PORTA        0x3c0
10  void main( )
```

```

11 {
12     unsigned char bytewrite;
13     unsigned char light;
14     int i;
15     void display( );
16
17     /* set up the output port */
18     bytewrite = 0x80;
19     outportb(SETPORT,bytewrite);
20
21     clrscr( );
22     gotoxy(1,1);
23     printf("Running Light Application 4");
24
25     /* running the light */
26     while ( !kbhit( ) )
27     {
28         light = 0x80;
29         for ( i = 1; i <= 8; i++ )
30         {
31             outportb(PORTA,light);
32             display(light);
33             light = light >> 1;          /* right rotate */
34             sleep(1);                    /* delay 1 second */
35         }
36     }
37 }
38 void display(unsigned char light)
39 {
40
41     gotoxy(10,3);
42     printf("%d",(light & 0x80) >> 7);
43     printf("%d",(light & 0x40) >> 6);
44     printf("%d",(light & 0x20) >> 5);
45     printf("%d",(light & 0x10) >> 4);
46     printf("%d",(light & 0x08) >> 3);
47     printf("%d",(light & 0x04) >> 2);
48     printf("%d",(light & 0x02) >> 1);
49     printf("%d",light & 0x01);
50 }

```

本程序流程图如下所示:



程序示例 rlight4.c 解释:

1. 第 8 行是用于设置 SETPORT 值为控制寄存器的口, 其值是 0x3e3.
2. 第 9 行设置 PORTA 口值是 0x3e0.
3. 第 18 行至 19 行是设置 PORTA 口专供输出使用.
4. 第 21 行是清除屏幕内容.

5. 第 22 行是将光标移至 (1,1) 位置。
6. 第 23 行是输出 "Running Light Application 4" 字符串。
7. 第 31 行是将跑马灯值输出至 PORTA。
8. 第 32 行是调用 display( ) 函数将跑马灯值在屏幕上显示。
9. 第 33 行是将跑马灯的值向右旋转。
10. 第 34 行是令时间延迟 1 秒钟。
11. 第 29 行至 35 行是将跑马灯值输出至 PORTA 的循环。
12. 第 26 行至 36 行是检查是否有键盘输入的循环, 如果有键盘输入发生则程序结束。

函数 display( ) 解释:

1. 第 41 行是将光标移至 (10,3) 位置。
2. 第 42 行是显示跑马灯的第 7 位。
3. 第 43 行是显示跑马灯的第 6 位。
4. 第 44 行是显示跑马灯的第 5 位。
5. 第 45 行是显示跑马灯的第 4 位。
6. 第 46 行是显示跑马灯的第 3 位。
7. 第 47 行是显示跑马灯的第 2 位。
8. 第 48 行是显示跑马灯的第 1 位。
9. 第 49 行是显示跑马灯的第 0 位。

## 9.4 结 论

通过以上实例, 我们也可以了解大规模的显示板是如何控制的。电视墙是由数百个大规模的灯所组成的, 它的基本原理如同这一章的描述。最大的不同即是它有数种颜色, 且需要更多的 I/O 去控制。

## 9.5 思考题

1. 如何设计一个程序可以同时左旋及右旋, 亦即在左旋转完毕后, 立即右旋转。
2. 如何设计一个程序以间隔式跳动如下:

$T=0$     ● ○ ● ○ ● ○ ● ○  
 $T=t$     ○ ● ○ ● ○ ● ○ ●  
 $T=2t$     ● ○ ● ○ ● ○ ● ○  
 $T=3t$     ○ ● ○ ● ○ ● ○ ●  
 $T=4t$     ● ○ ● ○ ● ○ ● ○  
 $T=5t$     ○ ● ○ ● ○ ● ○ ●  
 $T=6t$     ● ○ ● ○ ● ○ ● ○  
 $T=7t$     ○ ● ○ ● ○ ● ○ ●

## 第十章 红绿灯专题

### 本章实验目的

1. 通过8255A红绿灯的应用,可更深一步了解复杂的I/O应用,并借此了解下列知识:

■IBM PC I/O 接口控制

■红绿灯的工作原理及更复杂的红绿灯控制知识

### 本章内容

- 10.1 工作目标
- 10.2 硬件设计
- 10.3 软件设计
- 10.4 结论
- 10.5 思考题

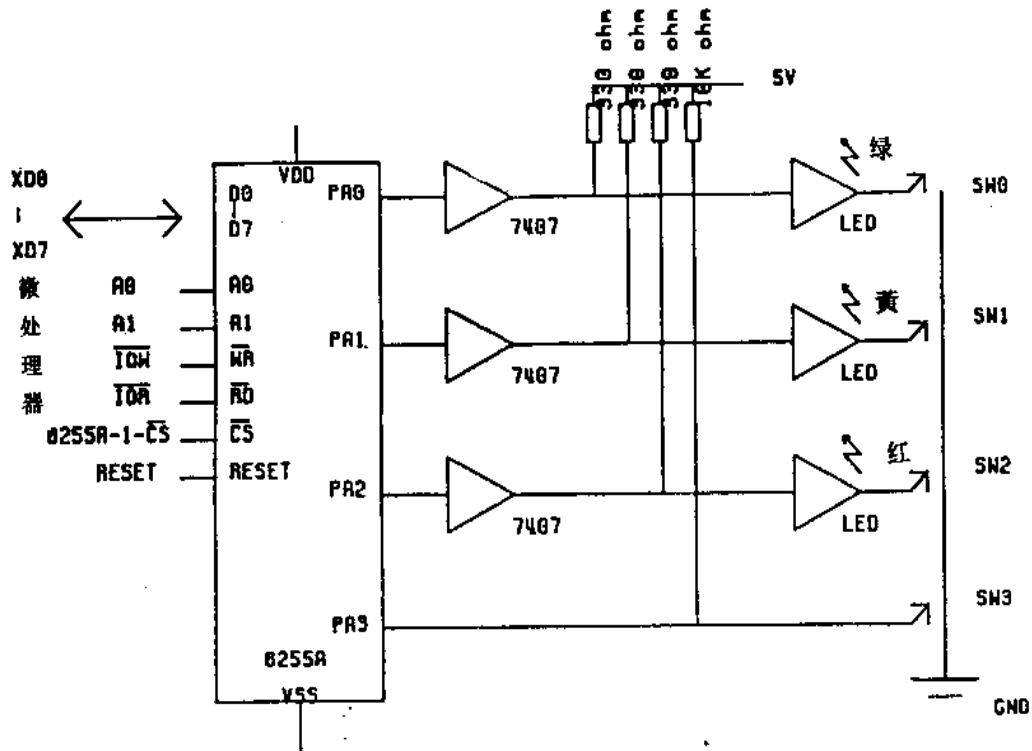
### 10.1 工作目标

十字路口的红绿灯模拟。红绿灯的闪动可以分为两种情形:

1. 由警察控制: 当  $PA3=1$  时, 警察可以随时控制三种灯号的亮暗。
2. 完全自动循环变换: 当  $PA3=0$  时, 绿、黄、红灯号以5秒、2秒、5秒连续动作。



## 10.2 硬件设计



## 10.3 软件设计

程序示例 `rgyl.c`

红绿灯的控制。本程序在运行时，若不控制任何开关，则 PORTA 的 PA0~PA2 将如下所示。首先绿灯亮 5 秒钟。

PA2 PA1 PA0

○ ○ ●

↑

绿灯亮 5 秒钟

然后黄灯亮 2 秒钟，如下所示：

○ ● ○

↑

黄灯亮 2 秒钟

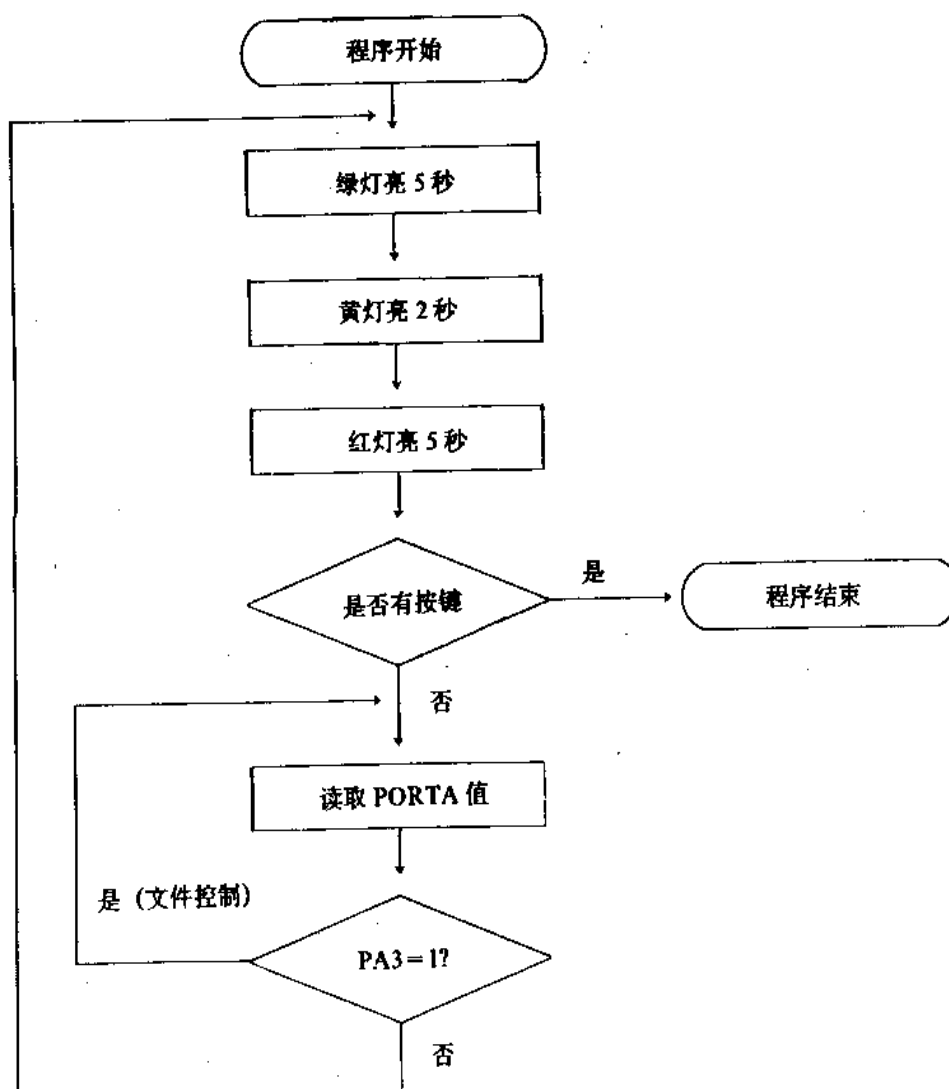
然后红灯亮 5 秒钟，如下所示：

● ○ ○

↑

红灯亮 5 秒钟

上述循环后，则检查交通警察是否控制 PA3 为 1，如果有表示交警控制了红绿灯，此时哪一个灯亮，完全由交警控制，直至 PA3 为 0 才改成自动控制。  
本程序的流程如下所示：



```

01 /* ----- */
02 /*      Program Name : rgyl.c      */
03 /*  Red, Green and Yellow light control.  */
04 /*  For 8255A,   Mode 0 application  */
05 /* ----- */
06 #include <dos.h>
07 #define  SETPORT    0x3c3
08 #define  PORTA      0x3c0
09
10 void main( )
  
```

```

11 {
12     unsigned char bytewrite, byteread;
13     unsigned char light;
14
15     while ( !kbhit( ) )
16     {
17         /* set up the output port */
18         bytewrite = 0x80;
19         outportb(SETPORT,bytewrite);
20
21         /* Green light on */
22         bytewrite = 1;
23         outportb(PORTA,bytewrite);
24         sleep(5);
25
26         /* Yellow light on */
27         bytewrite = 2;
28         outportb(PORTA,bytewrite);
29         sleep(2);
30
31         /* Red light on */
32         bytewrite = 4;
33         outportb(PORTA,bytewrite);
34         sleep(5);
35
36         /* check if police control the light */
37         while ( 1 )
38         {
39             bytewrite = 0x90;
40             outportb(SETPORT,bytewrite); /* set INPUT MODE */
41
42             byteread = inportb(PORTA); /* get value from PORTA */
43             byteread &= 0x08;
44             if ( byteread != 0x08 )
45                 break;
46         }
47     }
48 }

```

程序示例 rgyl.c 解释:

1. 第 7 行是用于设置 SETPORT 值为控制寄存器的口, 其值是 0x3e3.
2. 第 8 行是用于设置 PORTA 口值是 0x3e0.
3. 第 18 行至 19 行是设置 PORTA 口专供输出使用.
4. 第 22 行至第 23 行是设置绿灯亮.

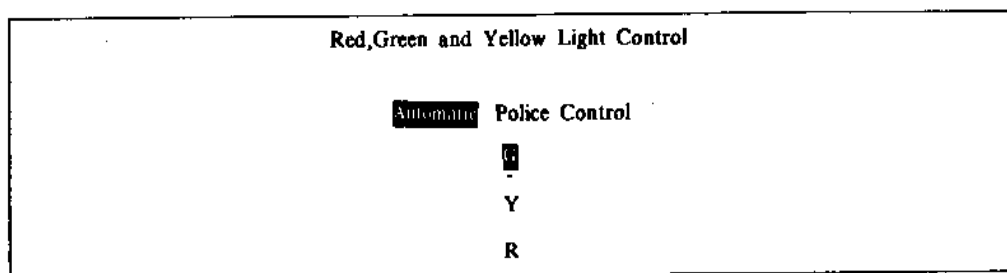
5. 第 24 行是令时间暂停 5 秒，也就是绿灯亮的时间是 5 秒。
6. 第 27 行至第 28 行是设置黄灯亮。
7. 第 29 行是令黄灯亮的时间是 2 秒。
8. 第 32 行至第 33 行是设置红灯亮。
9. 第 34 行是令红灯亮的时间是 5 秒。
10. 第 39 行至第 40 行是设置 PORTA 口专供输入使用。
11. 第 42 行是读取 PORTA 的值。
12. 第 43 行至第 45 行是检查交通警察是否有控制灯号，如果没有则跳回第 15 行，如果有则在第 37 至 46 行之间的循环持续运行。
13. 第 37 行至第 46 行是一个无限循环，如果交通警察持续控制灯号则此循环一直持续，直至交通警察放弃控制灯号才结束。
14. 第 15 行至第 47 行是程序循环，如果有键盘输入，则循环结束。

#### 程序示例 rgy2.c

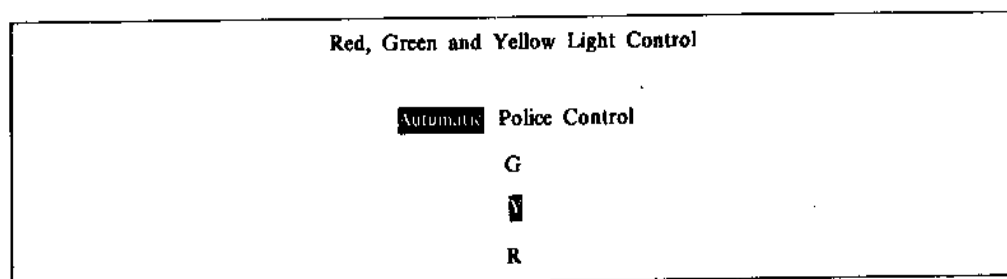
本程序基本上是前一个程序的扩充，即除了 LED 的各灯亮之外，屏幕也将以 G (Green 的缩写代表绿灯)，且以反白方式代表某特定亮的灯。

同时，如果目前交通信号是处于自动控制时，Automatic 将以反白方式显示。如果目前交通信号是处于交通警察控制时，Police Control 将以反白方式显示。

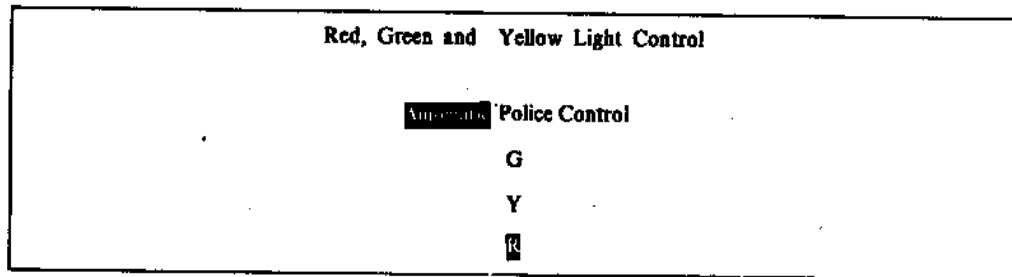
例如，当程序开始运行绿灯亮 5 秒时，首先屏幕将如下所示：表示红绿灯自动控制，此时为绿灯亮。



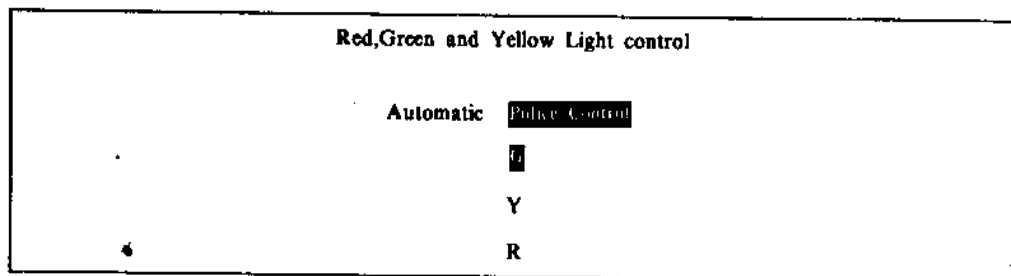
然后黄灯亮 2 秒钟时，屏幕将如下所示：



然后红灯亮 5 秒钟时，屏幕将如下所示：



当交通警察控制信号时，Police Control 字符串将以反白方式显示，至于此时的 G，Y，R 字符，是哪一个字符以反白显示，则完全由哪一个灯是交通警察控制的亮灯决定。例如，交通警察控制绿灯亮时，屏幕内容将如下所示：



本程序的流程如154 页图所示。

```

001  /* ----- */
002  /*      Program Name : rgy2.c      */
003  /*      Red, Green and Yellow light control with screen      */
004  /*      output.      */
005  /*      For 8255A,   Mode 0 application      */
006  /* ----- */
007  #include <dos.h>
008  #include <conio.h>
009  #define  SETPORT    0x3e3
010  #define  PORTA      0x3e0
011
012  void main( )
013  {
014      void displaylight( );
015      void analysislight( );
016      unsigned char bytewrite, bytread;
017      unsigned char light;
018
019      clrscr( );
020      window(1,1,80,25); /* set the screen coordinate as window */
021      gotoxy(23,1);
022      printf("Red, Green and Yellow Light Control");
023      gotoxy(27,4);
024      textcolor(BLACK);

```

```

25     textbackground(WHITE);
26     cprintf("Automatic");
27     textcolor(WHITE);
28     textbackground(BLACK);
29     gotoxy(41,4);
30     cprintf("Police Control");
31     while ( !kbhit( ) )
32     {
33     /* set up the output port */
34         bytewrite = 0x80;
35         outportb(SETPORT,bytewrite);
36
37     /* Green light on */
38         bytewrite = 1;
39         outportb(PORTA,bytewrite);
40         displaylight(6,7,8);
41         sleep(5);
42
43     /* Yellow light on */
44         bytewrite = 2;
45         outportb(PORTA,bytewrite);
46         displaylight(7,8,6);
47         sleep(2);
48
49     /* Red light on */
50         bytewrite = 4;
51         outportb(PORTA,bytewrite);
52         displaylight(8,6,7);
53         sleep(5);
54
55     /* check if police control the light */
56         while ( 1 )
57         {
58             bytewrite = 0x90;
59             outportb(SETPORT,bytewrite); /* set INPUT MODE */
60
61             byteread = inportb(PORTA); /* get value from PORTA */
62             byteread &= 0x08;
63             if ( byteread != 0x08 )
64                 break;
65             textcolor(BLACK);
66             textbackground(WHITE);
67             gotoxy(41,4);
68             cprintf("Police Control");
69             textcolor(WHITE);

```

```

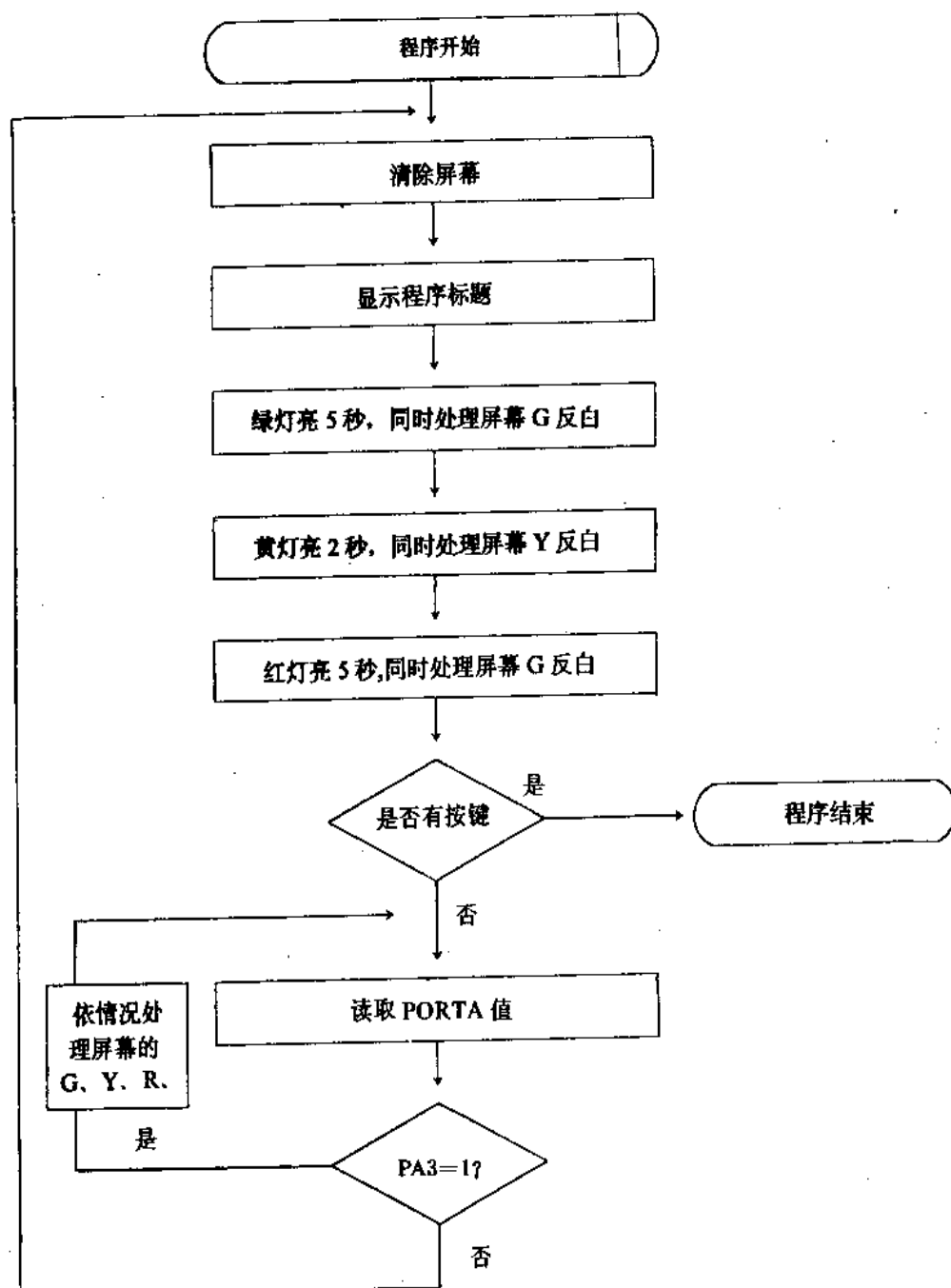
70     textbackground(BLACK);
71     gotoxy(27,4);
72     cprintf("Automatic");
73     analysislight( );
74 }
75 textcolor(BLACK);
76 textbackground(WHITE);
77 gotoxy(27,4);
78 cprintf("Automatic");
79 textcolor(WHITE);
80 textbackground(BLACK);
81 gotoxy(41,4);
82 cprintf("Police Control");
83 }
84 }
85 void analysislight( )
86 {
87     void displaylight( );
88     unsigned char byteread;
89
90     byteread = inportb(PORTA);
91     if ( ( byteread & 0x01 ) == 1 )      /* Green light on */
92         displaylight(6,7,8);
93     else if ( ( byteread & 0x02 ) == 2 ) /* Yellow light on */
94         displaylight(7,8,6);
95     else if ( ( byteread & 0x04 ) == 4 ) /* Red light on */
96         displaylight(8,6,7);
97 }
98 void displaylight(int i, int j, int k)
99 {
100     void printchar( );
101
102     gotoxy(38,i);
103     textcolor(BLACK);
104     textbackground(WHITE);
105     printchar(i);
106     textcolor(WHITE);
107     textbackground(BLACK);
108     gotoxy(38,j);
109     printchar(j);
110     gotoxy(38,k);
111     printchar(k);
112 }
113 void printchar(int ch)
114 {

```

```

115  if ( ch == 6 )
116      cprintf("G");
117  else if ( ch == 7 )
118      cprintf("Y");
119  else
120      cprintf("R");
121  }

```





程序示例 rgy2.c 解释:

1. 第 9 行是用于设置 SETPORT 值为控制寄存器的口, 其值是 0x3e3.
2. 第 10 行用于设置 PORTA 口值是 0x3e0.
3. 第 19 行是清除屏幕内容.
4. 第 20 行是设置整个屏幕成一个窗口, 这样便可以使用 cprintf( ) 函数达到控制某些字符串反白效果.
5. 第 21 行是将光标移至 (23,1) 位置.
6. 第 24 行至第 26 行是设置以反白显示 "Automatic".
7. 第 27 行至第 30 行是设置将光标移至 (41,4) 位置, 然后以正常方式显示 "Police Control".
8. 第 34 行至第 35 行是设置 PORTA 口专供输出使用.
9. 第 38 行至第 39 行是设置绿灯亮.
10. 第 40 行是调用 displaylight( ) 函数, 令屏幕显示绿灯亮.
11. 第 41 行是令时间延迟 5 秒, 也就是令绿灯亮 5 秒.
12. 第 44 行至第 47 行是令黄灯亮 2 秒, 同时也将它反映在屏幕上.
13. 第 50 行至第 53 行是令红灯亮 5 秒, 同时也将它反映在屏幕上.
14. 第 58 行至第 59 行是设置 PORTA 口专供输入使用.
15. 第 61 行是读取 PORTA 的值.
16. 第 62 行至第 64 行是检查交通警察是否控制灯号, 如果没有则跳至程序第 75 行, 如果有则往下运行.
17. 第 65 行至第 72 行主要是控制以正常方式显示字符串 "Automatic", 以反白方式显示字符串 "Police Control".
18. 第 73 行是调用 analysislight( ) 函数, 主要是分析目前交通警察是控制哪一个灯在亮, 且将它反映在屏幕上.
19. 第 56 行至第 74 行是一个循环, 如果交通警察控制灯号则此循环将继续, 否则跳至第 75 行.
20. 第 75 行至第 82 行是设定以反白方式显示字符串 "Automatic", 以正常方式显示字符串 "Police Control".
21. 第 81 行至第 83 行是一个循环, 此循环将检查是否有键盘输入, 如果有则程序结束.

函数 analysislight 解释:

1. 第 90 行是读取 PORTA 值.
2. 第 91 行至第 96 行是分析所读取 PORTA 的值, 然后决定哪一个灯会亮, 同时调用 displaylight( ) 函数将亮灯部分反映在屏幕上.

函数 displaylight( ) 解释:

1. 此函数有 3 个输入参数分别是 i, j, k.
2. 第 102 行是将光标移至 (38,j) 位置.
3. 第 103 行至第 105 行是以反白显示 i.
4. 第 106 行至 111 行是以正常方式显示 i 及 k, 当然在显示前需将光标移至 (38,j) 和

(38,k) 位置。

5. 至于 i, j, k 的值代表什么灯号, 则要参考 printchar( )函数。

函数 printchar( )解释:

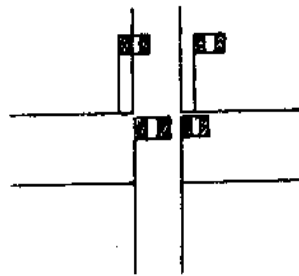
1. 此函数有一个输入参数。
2. 如果输入值等于 6 则显示字符 G。
3. 如果输入值等于 7 则显示字符 Y。
4. 如果输入值等于其他值 (应该是 8) 则显示字符 R。

## 10.4 结 论

通过以上实例, 我们可以了解 I/O 的输入及输出方式。

## 10.5 思考题

1. 以上实验交警要取得控制权必须等红灯闪动完毕之后才可。如何设计一程序可以让交警随时取得控制权?
2. 如何设计修改程序, 以避免交警同时将红灯及绿灯打开, 造成交通大乱?
3. 此程序只设计了一个红绿灯, 请设计出十字路口的四个红绿灯。



4. 笔者于1985年夏天在纽约开车曾连续通过76个红绿灯路口没有碰上一个红灯, 请思考纽约的红绿灯是如何设计的, 有无规则可循。

# 第十一章 家电控制专题

## 本章实验目的

1. 通过8255A的输出应用, 借以控制家电产品, 通过此实验我们可以了解下列知识:

■ IBM PC I/O 控制

■ 继电器的工作原理

■ 如何将 I/O 控制思想应用在家电产品

## 本章内容

11.1 工作目标

11.2 硬件设计

11.3 软件设计

11.4 结论

11.5 思考题

在日常生活中, 我们常常需要在某段时间内开关电器。譬如说在夏天, 我们希望下班回到家时, 已经有一个很清爽的屋子, 而不麻烦他人在我们回家前半小时将冷气打开。我们也希望回到家时饭已煮好了。这一类的工作可借用个别的定时器。这一章中我们将介绍如何利用 IBM PC 将所有的工作和家电用品连接在一起。

### 11.1 工作目标

利用 IBM PC 将

电锅在 17:00-18:00 打开

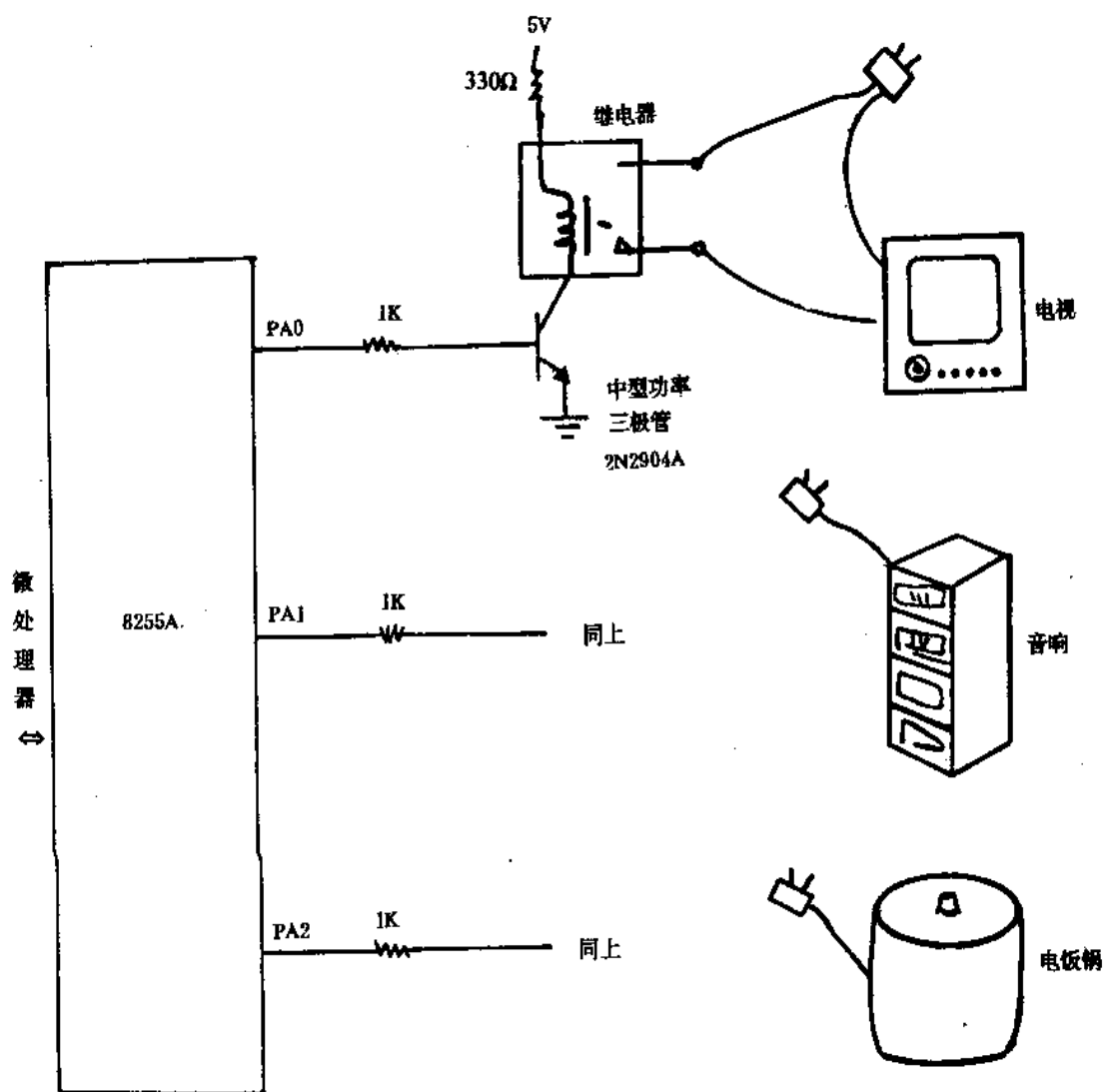
音响在 18:00-19:00 打开

电视在 19:00-22:00 打开

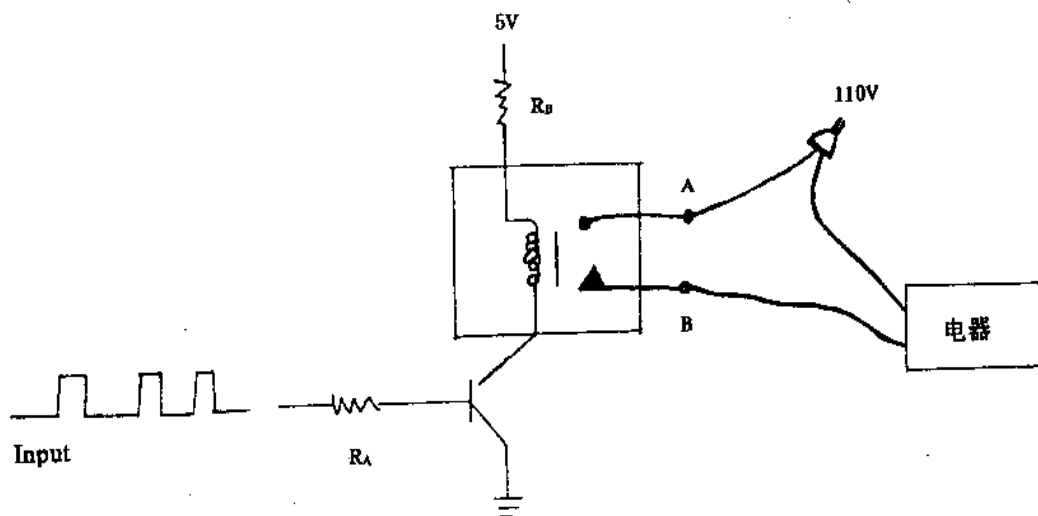
### 11.2 硬件设计

当 Input 为 1 时 A 点和 B 点导通。

$R_A$  和  $R_B$  按所选取的晶体管  $\beta$  (放大率) 而定。



继电器说明



### 11.3 软件设计

#### 程序示例 elect.c

本程序在运行时，系统会不断地读取系统时间，然后运行下列动作。

1. 在 17:00-18:00 让 PA2=1，相当于让电锅 ON。
2. 在 18:00-19:00 让 PA1=1，相当于让音响 ON。
3. 在 19:00-22:00 让 PA0=1，相当于让电视 ON。

同时本程序在运行时，若是上述电器产品没有一项是 ON 状态，则屏幕显示情形如下：

NOTHING ON

若是电视 ON，则屏幕显示如下所示：

TV ON

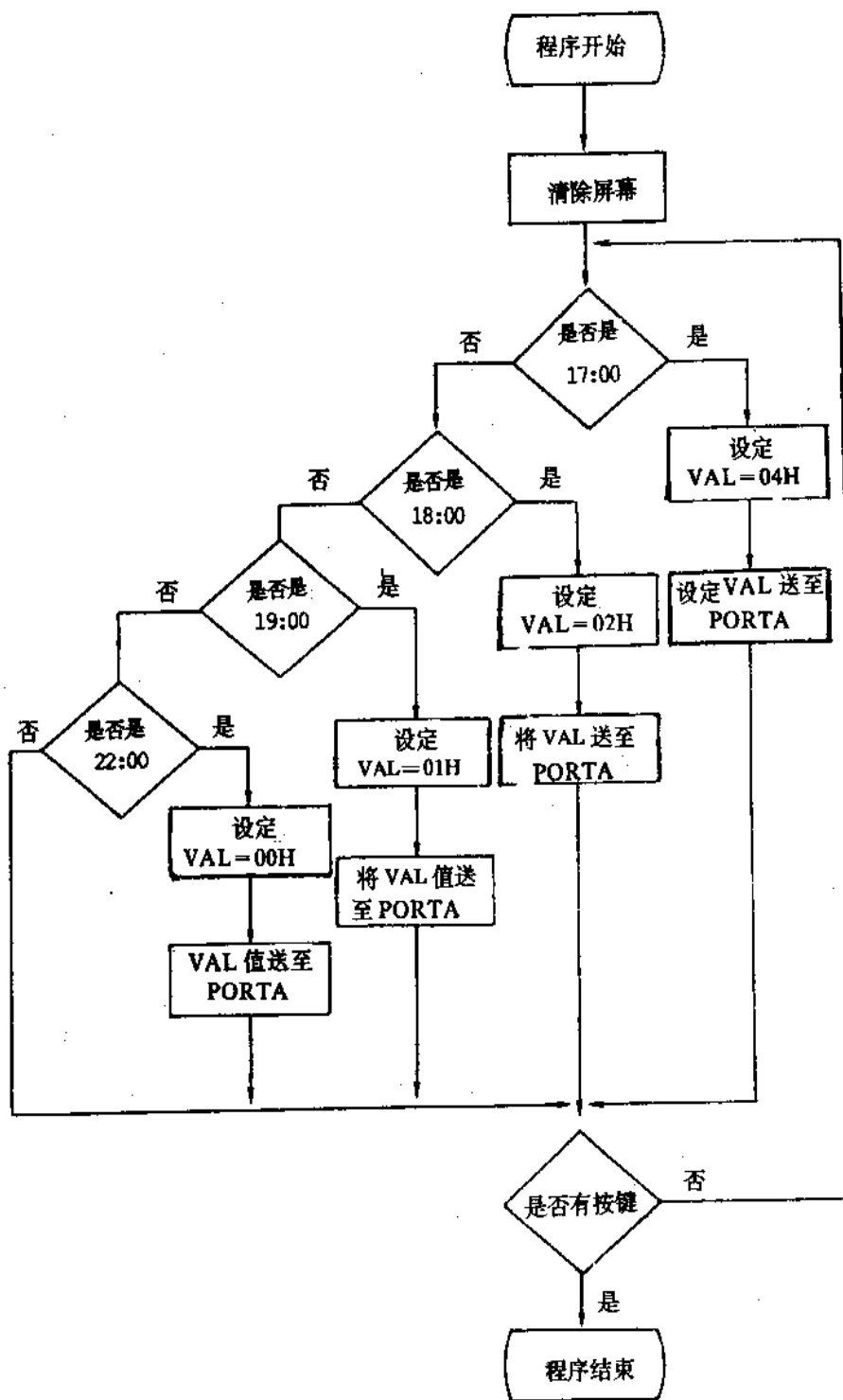
若是音响 ON，则屏幕显示如下所示：

STEREO ON

若是电锅 ON，则屏幕显示如下所示：

COOKER ON

本程序的设计流程如下所示：



```

01  / * ----- */
02  / *      Program Name : elect.c      */
03  / *      Electronic Appliances control.  */

```

```

04 /* PA0 control the TV      1 = ON,  0 = OFF      */
05 /* PA1 control the Stereo  1 = ON,  0 = OFF      */
06 /* PA2 control the Cooker  1 = ON,  0 = OFF      */
07 /* For 8255                                                         */
08 /* ----- */
09 #include <dos.h>
10 #include <time.h>
11 #include <stdio.h>
12 #include <conio.h>
13 #define SETPORT    0x3e3
14 #define PORTA      0x3e0
15 void main( )
16 {
17     unsigned char bytewrite;
18     unsigned char light;
19     struct tm * ptr;
20     time_t    var;
21
22
23     clrscr( );          /* clear the screen */
24
25     /* set up the output port */
26     bytewrite = 0x80;
27     outportb(SETPORT,bytewrite);
28
29     /* print NOTHING ON */
30     gotoxy(36,10);
31     printf("NOTHING ON");
32
33     /* running the main loop */
34     while ( !kbhit( ) )
35     {
36         var = time(NULL);
37         ptr = localtime( &var);
38
39         /* if 17:00, TURN on Cooker */
40         if ( ptr->tm_hour == 17 )
41             if ( ptr->tm_min == 0 )
42                 if ( ptr->tm_sec == 0 )
43                     {
44                         bytewrite = 0x04;
45                         outportb(PORTA,bytewrite); /* PA2 = 1 */
46                         gotoxy(36,10);
47                         printf("COOKER ON");

```

```

48         continue;
49     }
50
51     /* if 18:00, TURN on Stereo and TURN off Cooker */
52     if ( ptr->tm_hour == 18 )
53         if ( ptr->tm_min == 0 )
54             if ( ptr->tm_sec == 0 )
55                 {
56                     bytewrite = 0x02;
57                     outputb(PORTA,bytewrite); /* PA1 = 1 */
58                     gotoxy(36,10);
59                     printf("STEREO ON");
60                     continue;
61                 }
62
63     /* if 19:00, TURN on TV and TURN off Stereo */
64     if ( ptr->tm_hour == 19 )
65         if ( ptr->tm_min == 0 )
66             if ( ptr->tm_sec == 0 )
67                 {
68                     bytewrite = 0x01;
69                     outputb(PORTA,bytewrite); /* PA0 = 1 */
70                     gotoxy(36,10);
71                     printf("TV ON");
72                     continue;
73                 }
74
75     /* if 22:00, print NOTHING ON */
76     if ( ptr->tm_hour == 22 )
77         if ( ptr->tm_min == 0 )
78             if ( ptr->tm_sec == 0 )
79                 {
80                     bytewrite = 0x00;
81                     outputb(PORTA,bytewrite); /* PA0 - PA7 = 0 */
82                     gotoxy(36,10);
83                     printf("NOTHING ON");
84                 }
85     }
86 }

```

程序示例 elect.c 解释:

1. 第 13 行是用于设置 SETPORT 值为控制寄存器的口, 其值是 0x3e3.
2. 第 14 行用于设置 PORTA 口值是 0x3e0.
3. 第 23 行是清除屏幕内容.



4. 第26行至第27行是设置PORTA口专供输出使用。
5. 第30行至第31行是设置在屏幕(36,10)位置, 显示字符串“NOTHING ON”。
6. 第36行至第37行是读取系统时间。
7. 第40行至第49行是检查系统时间是不是17:00, 如果是则令PA2值是1, 且将光标移至(36,10)位置, 同时显示字符串“COOKER ON”。
8. 第52行至第61行是检查系统时间是不是18:00, 如果是则令PA1值是1, 且将光标移至(36,10)位置, 同时显示字符串“STEREO ON”。
9. 第64行至第73行是检查系统时间是不是19:00, 如果是则令PA0值为1, 且将光标移至(36,10)位置, 同时显示字符串“TV ON”。
10. 第76行至第84行是检查系统时间是不是22:00, 如果是则令PORTA值为0(PA0~PA7=0), 且将光标移至(36,10)位置, 同时显示字符串“NOTHING ON”。
11. 第34行和第85行是一个循环, 此循环将检查是否有键盘输入, 如果有则离开此循环, 相当于令程序结束。

## 11.4 结 论

本实验只是讲解简单家电控制功能, 读者应尝试去控制其它家电产品等。

## 11.5 思考题

1. 当您把所有的电器如音响、卡拉OK伴唱机、电视、冰箱、录像机等都和计算机连接时, 您的保险丝可能会烧掉, 这时候您怎么办? 请修改软件, 防范这些问题的出现。
2. 修改本程序, 增加下列功能:  
18:00-21:30 打开录像机
3. 修改程序, 明晨令收音机将您吵醒, 6:30 微波炉将早餐准备好。

### 小常识

现今的技术已经可能将所有电器产品通过电话线由用户在外地遥控, 请思考IBM PC在这种技术中可以扮演何种角色?

## 第十二章 LED 七段显示器专题

### 本章实验目的

1. 由本实验读者可以了解

■七段显示器的工作原理

■模拟数字钟的显示

■系统时间的读取

■更省电的显示方式, 共阳极、共阴极、 $5\times 7$  的多路显示器

### 本章内容

12.1 工作目标

12.2 硬件设计

12.3 硬件说明

12.4 软件设计

12.5 结论

12.6 思考题

### 12.1 工作目标

本专题包含两个实验:

1. 在键盘上按下一个数字键, 则在LED七段显示器上显示出来, 3秒 (由程序示例 LED71.ASM 106 行设置) 之后可再按另一个数字键。
2. 读取系统时间, 将秒数在 LED 七段显示器上显示出来。

### 12.2 硬件设计

本实验硬件线路设计如下图所示:

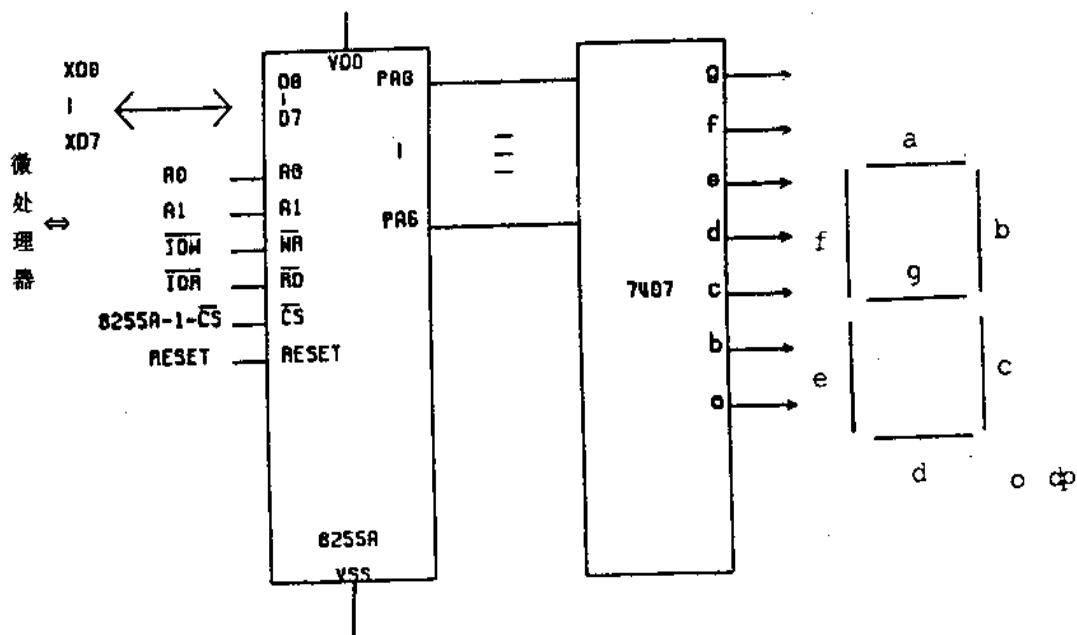


图 12.1 本实验专题所用硬件线路

### 12.3 硬件说明

利用 8255A 的 PA0~PA6 驱动 7407 缓冲器。a, b, c, d, e, f, g 为 7407 的输出，用以驱动七段显示器的七个显示段 (Segment)。

表 12.1 共阴极七段显示器数据对照表

字体	显示数字	a	b	c	d	e	f	g	HEX 码
	0	1	1	1	1	1	1	0	7EH
	1	0	1	1	0	0	0	0	30H
	2	1	1	0	1	1	0	1	6DH
	3	1	1	1	1	0	0	1	79H
	4	0	1	1	0	0	1	1	33H
	5	1	0	1	1	0	1	1	5BH
	6	1	0	1	1	1	1	1	5FH
	7	1	1	1	0	0	0	0	70H
	8	1	1	1	1	1	1	1	7FH
	9	1	1	1	0	1	1	1	7BH
	A	1	1	1	0	1	1	1	77H
	B	0	0	1	1	1	1	1	1FH
	C	1	0	0	1	1	1	0	4EH
	D	0	1	1	1	1	0	1	3DH
	E	1	0	0	1	1	1	1	4FH
	F	1	0	0	0	1	1	1	47H

dp 为 LED 的光点，一般是为了美观，本实验中不使用。

本专题所使用的是共阴极七段显示器，表 12.1 为共阴极七段显示字型数据对照表。在输出任何字型时，a, b, c, d, e, f, g 的必要状态及 HEX 码。

#### 电流计算

因为这一类的显示器所消耗的电流相当大，所以必须特别考虑输出的问题。

一般而言，LED 的压降约为 2V，而输出为 0 的电压值约为 0.8V，则此时流经 LED 的电流为：

$$I_D = \frac{5 - 2 - 0.8}{R}$$

当  $R = 330\Omega$  时

$$I_D \approx 10mA$$

8255A 不能驱动 LED 的原因就是因为它不能供给 10mA 的电流到端口上，所以必须靠大规模的缓冲器当作缓冲媒介，借以驱动七段显示器。如果我们要驱动字型“8”，它的字段要求如下：

a:1

b:1

c:1

d:1

e:1

f:1

g:1

HEX:7F

七个段都必须亮，所以消耗电流为 70mA。如果要驱动字型“F”则须 40mA。一个微型计算机 IC 的正常消耗电流为 5mA~7mA。所以根本不可能去驱动 LED，8255A 的推动能力也有限。7407 则是设计用以驱动其它元件用的 IC，它的驱动能力经过特别的设计。我们常用它驱动 LED。

## 12.4 软件设计

#### 程序示例 led71.c

七段显示器设计。本程序在运行时会把你所输入的字在七段显示器上显示出来。本程序运行时，屏幕将如下所示：

Seven Section LED application
INPUT NUMBER:

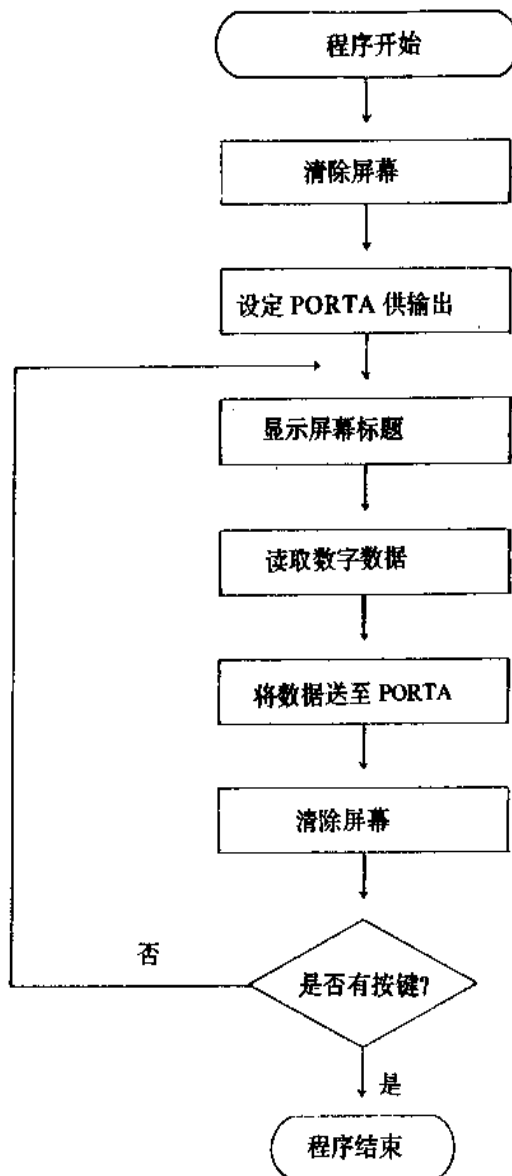
假设你所输入是 7，则七段显示器将显示 7，同时屏幕将如下所示：

Seven Section LED application

INPUT NUMBER:7

Press any to exit the program

此时，若是你随意按一个键可以使程序立即结束，本程序工作流程如下所示：



```
01 /* ----- */
02 /*      Program Name : led71.c      */
03 /*      Seven section LED application */
```

```

04  /* For 8255A, Mode 0, PORTA output.          */
05  /* ----- */
06  #include <dos.h>
07  #include <conio.h>
08  #define SETPORT    0x3e3
09  #define PORTA      0x3e0
10  unsigned char val[10] = {
11      0x7c, 0x30, 0x6d, 0x79, 0x33,
12      0x5b, 0x5f, 0x70, 0x7f, 0x7b };
13
14  void main( )
15  {
16      void sendport( );
17      void printmsg( );
18      void printtitle( );
19      unsigned char bytewrite;
20      unsigned char light;
21      int data;
22
23      /* set up the output port */
24      bytewrite = 0x80;
25      outportb(SETPORT,bytewrite);
26
27      /* Running the SEVEN LED application */
28      while ( !kbhit( ) )
29      {
30          clrscr( );          /* clear the screen          */
31          printtitle( );      /* print the program title */
32          printmsg( );        /* print the get data msg   */
33          scanf("%d",&data); /* get the input from screen */
34          sendport(data);     /* send the output to PORTA */
35          gotoxy(26,12);
36          printf("Press any to exit the program.");
37          sleep(3);
38      }
39  }
40  /* ----- */
41  /* send the screen input and convert it then send */
42  /* it to PORTA */
43  /* ----- */
44  void sendport(int data)
45  {
46      unsigned char bytewrite;
47
48      bytewrite = val[data];

```

```

49     outportb(PORTA,bytewrite);
50 }
51 /* ----- */
52 /*  print the get data msg                      */
53 /* ----- */
54 void printmsg( )
55 {
56     gotoxy(34,4);
57     printf("INPUT  NUMBER : ");
58 }
59 /* ----- */
60 /*  print the program title                      */
61 /* ----- */
62 void printtitle( )
63 {
64     gotoxy(27,1);
65     printf("Seven Section LED application");
66 }

```

程序示例 LED71.ASM 说明:

1. 第 8 行是用于设置 SETPORT 值为控制寄存器的口, 其值是 0x3c3.
2. 第 9 行是设置 PORTA 值是 0x3c0.
3. 第 24 行至第 25 行是设置 PORTA 口专供输出使用.
4. 第 30 行是清除屏幕内容.
5. 第 31 行是调用 printtitle( ) 函数显示程序标题.
6. 第 32 行是调用 printmsg( ) 函数显示 "INPUT NUMBER" 字符串.
7. 第 33 行是读取键盘输入值.
8. 第 34 行是调用 sendport( ) 函数将键盘输入转换成七段显示器的 HEX 码及将此 HEX 值送至 PORTA 控制七段显示器输出.
9. 第 35 行是将光标移至 (26,12).
10. 第 36 行是显示字符串 "Press any to exit the program".
11. 第 37 行是令时间暂停 3 秒.
12. 第 28 行至第 34 行是一循环, 用于检查是否有键盘按键, 如果有则程序结束.

函数 sendport( ) 解释:

1. 第 48 行是将传至 sendport( ) 函数的值转换成七段显示器的 HEX 码.
2. 第 49 行是将七段显示器的 HEX 码送至 PORTA, 以便显示结果.

函数 printmsg( ) 解释:

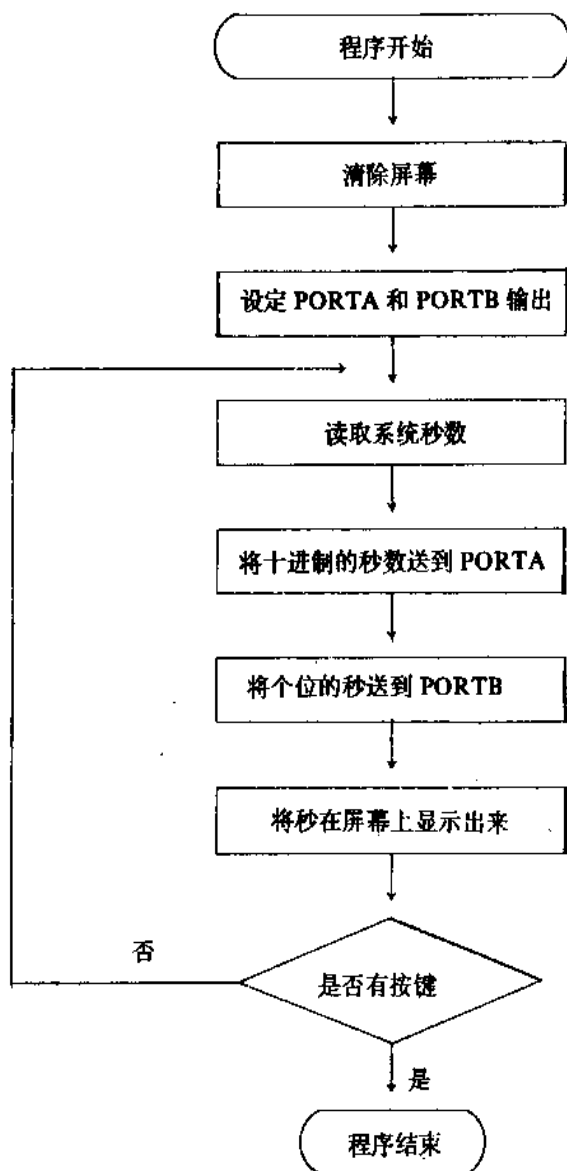
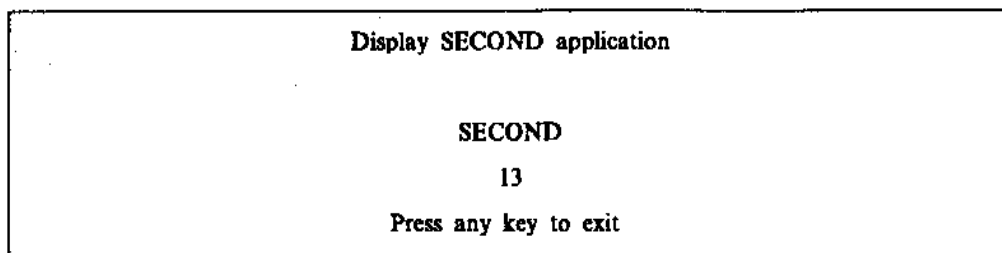
1. 第 56 行是将光标移至 (34,4) 位置.
2. 第 57 行是显示字符串 "INPUT NUMBER".

函数 printtitle( ) 解释:

1. 第 64 行是将光标移至 (27,1) 位置.
2. 第 65 行是显示字符串 "Seven Section LED application".

### 程序示例 led72.c

本程序在运行时屏幕将如下图所示：



显示系统的秒数。本程序运行时会采用不断读取系统时间的方式，且将系统时间的秒



数送至 PORTA 和 PORTB 的 7 段显示器内。其中 PORTA 所显示的是 10 进位的秒数，而 PORTB 所显示的则是个位数的秒数。同时上述的秒数也将不断的在屏幕某固定位置显示出来。

上述程序在运行过程中，只要随意按一个键，程序将立即结束。

本程序流程图如上页图所示。

```

01  /* ----- */
02  /*      Program Name : led72.c      */
03  /*      LED application      */
04  /*      For 8255A, Mode 0, output code to PORTA and      */
05  /*      PORTB.      */
06  /* ----- */
07  #include <dos.h>
08  #include <time.h>
09  #include <stdio.h>
10  #include <conio.h>
11  #define SETPORT    0x3e3
12  #define PORTA      0x3e0
13  #define PORTB      0x3e1
14  unsigned char val[10] = {
15      0x7e, 0x30, 0x6d, 0x79, 0x33,
16      0x5b, 0x5f, 0x70, 0x7f, 0x7b };
17
18  void main( )
19  {
20      void sendporta( );
21      void sendportb( );
22      void printtitle( );
23      struct tm * ptr;
24      time_t    var;
25      unsigned char bytewrite;
26      int  sec, sec10;
27
28      clrscr( );          /* clear the screen      */
29      gotoxy(30,8);
30      printf("Press any key to exit.");
31
32  /* set up the PORTA and PORTB as output port */
33      bytewrite = 0x80;
34      outportb(SETPORT,bytewrite);
35
36  /* Running the SEVEN LED application */
37      while ( !kbhit( ) )
38      {

```

```

39     printtitle( );          /* print the program title */
40     var = time(NULL);
41     ptr = localtime( &var);
42     sec10 = ptr->tm_sec / 10;
43     sendporta(sec10);        /* send second to PORTA base 10 */
44     sec = ptr->tm_sec % 10;
45     sendportb(sec);          /* send second to PORTB base 1 */
46     gotoxy(38,4);
47     printf("SECOND");
48     gotoxy(40,6);
49     printf("%d",ptr->tm_sec); /* display second on the screen */
50 }
51 }
52 /* ----- */
53 /* send the screen input and convert it then send */
54 /* it to PORTA ( base 10 ) */
55 /* ----- */
56 void sendporta(int data)
57 {
58     unsigned char bytewrite;
59
60     bytewrite = val[data];
61     outportb(PORTA,bytewrite);
62 }
63 /* ----- */
64 /* send the screen input and convert it then send */
65 /* it to PORTB ( base 1 ) */
66 /* ----- */
67 void sendportb(int data)
68 {
69     unsigned char bytewrite;
70
71     bytewrite = val[data];
72     outportb(PORTB,bytewrite);
73 }
74 /* ----- */
75 /* print the program title */
76 /* ----- */
77 void printtitle( )
78 {
79     gotoxy(28,1);
80     printf("Display SECOND application");
81 }

```

程序示例 led72.c 解释:

1. 第 11 行是用于设置 SETPORT 值为控制寄存器的口, 其值是 0x3e3.
2. 第 12 行是设置 PORTA 口值是 0x3e0.
3. 第 13 行是设置 PORTB 口值是 0x3e1.
4. 第 28 行是用于清除屏幕内容.
5. 第 29 行是将光标移至 (30,8).
6. 第 30 行是显示字符串 "Press any Key to exit".
7. 第 33 行至第 34 行是用于设置 PORTA 和 PORTB 专供输出使用.
8. 第 39 行是调用 printtitle() 函数显示程序标题.
9. 第 40 行至第 41 行是读取系统时间.
10. 第 42 行是计算 10 进位的秒值.
11. 第 43 行是调用 sendporta() 函数将 10 进位的秒值转换成七段显示器的 HEX 码, 再将它送至 PORTA.
12. 第 44 行是计算个位数的秒值.
13. 第 45 行是调用 sendportb() 函数将个位数秒值转换成七段显示器的 HEX 码, 再将它送至 PORTB.
14. 第 46 行是将光标移至 (38,4) 位置.
15. 第 47 行是显示字符串 "SECOND".
16. 第 48 行是将光标移至 (40,6) 位置.
17. 第 49 行是在屏幕上显示秒数.
18. 第 37 行是一个循环, 检查是否有键盘输入, 如果有则程序结束.

函数 sendporta() 解释:

1. 第 60 行是将传至 sendporta() 函数的值转换成七段显示器的 HEX 码.
2. 第 61 行是将 HEX 码值送至 PORTA.

函数 sendportb() 解释:

1. 第 71 行是将传送到 sendportb() 函数的值转换成七段显示器的 HEX 码.
2. 第 72 行是将 HEX 码值送至 PORTB.







函数 printtitle() 解释:

1. 第 79 行是将光标移至 (28,1) 位置.
2. 第 80 行是显示字符串 "Display SECOND application".

## 12.5 结 论

一般的七段显示器, 除了共阴极外, 还有共阳极的显示器, 其对照表如下:

表 12.2 共阳极七段显示字型数据对照表

字体	显示数字	a	b	c	d	e	f	g	HEX 码
	0	0	0	0	0	0	0	1	01H
	1	1	0	0	1	1	1	1	4FH
	2	0	0	1	0	0	1	0	12H
	3	0	0	0	0	1	1	0	06H
	4	1	0	0	1	1	0	0	4CH
	5	0	1	0	0	1	0	0	24H

(续表)

字体	显示数字	a	b	c	d	e	f	g	HEX 码
	6	0	1	0	0	0	0	0	20H
	7	0	0	0	1	1	1	1	0FH
	8	0	0	0	0	0	0	0	00H
	9	0	0	0	0	1	0	0	04H
	A	0	0	0	1	0	0	0	08H
	B	1	1	0	0	0	0	0	60H
	C	0	1	1	0	0	0	1	31H
	D	1	0	0	0	0	1	0	42H
	E	0	1	1	0	0	0	0	30H
	F	0	1	1	1	0	0	1	38H

## 12.6 思考题

如何用共阳极、共阴极、 $5 \times 7$ 的多路显示器做更省电的显示器?

1. 0~F共16个字型，我们一定要PA0~PA6，7个位去表示吗？7个位应可以表示 $2^7=128$ 个字型，我们应如何只用PA0~PA3 4个位去表示0~F 16个字型？
2. 如果我们要驱动字符“8”，必需要 $10\text{mA} \times 8 = 0.08\text{A}$ 电流，这个电流在电子应用中不怎么经济，是否有其它方式达到省电的效果？
3. LED的显示相当耗电，而LED的点阵一般为 $5 \times 7$ ，如何利用这35个点达到最经济的显示？

提示 1:

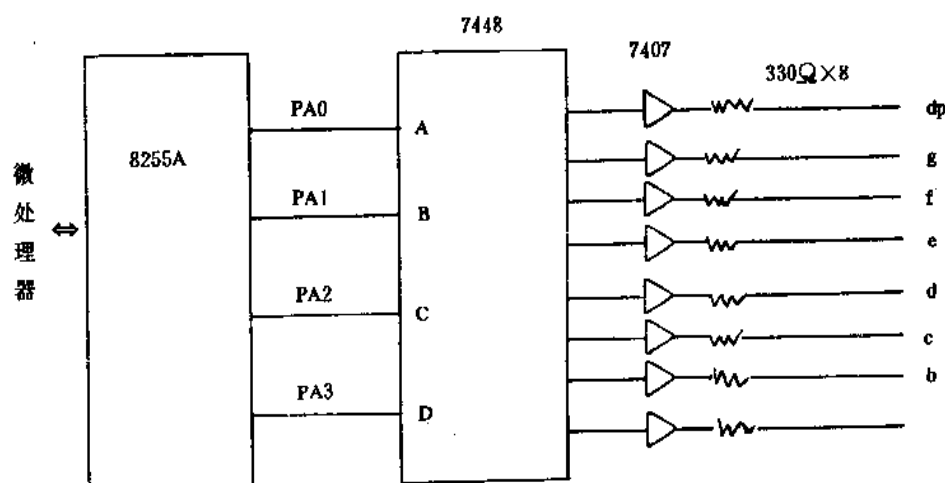


图 12.2 7448 译码器电路

7448 为 BCD 的译码器，7448 的驱动能力有限所以必须加驱动器，请读者设计它。在这种情况下，我们利用 7448 的硬件来译码，省略了 PA4~PA6 3 个位。

提示 2:

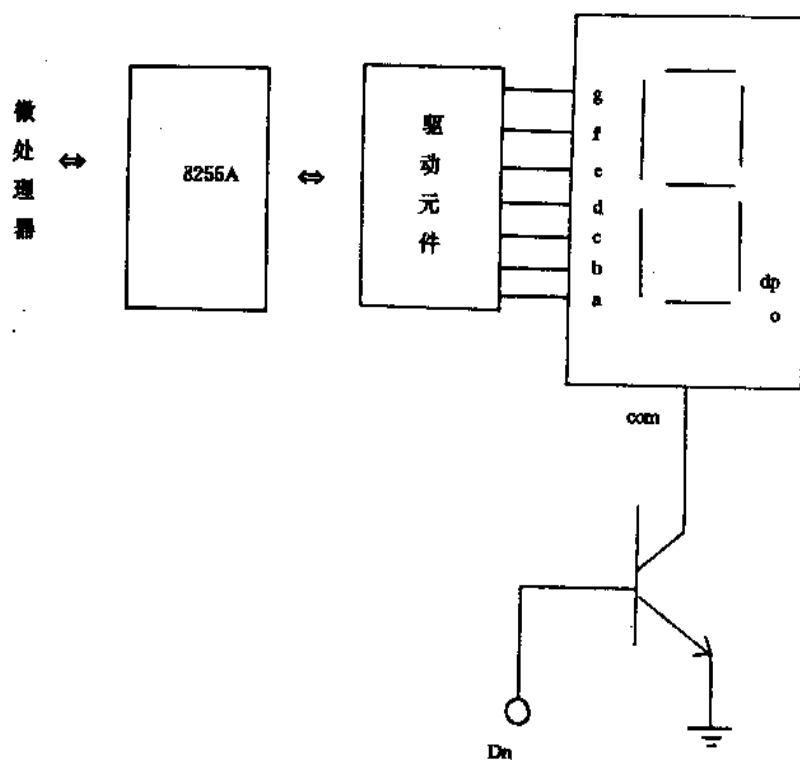


图 12.3 共阴极 7 段 LED 显示器

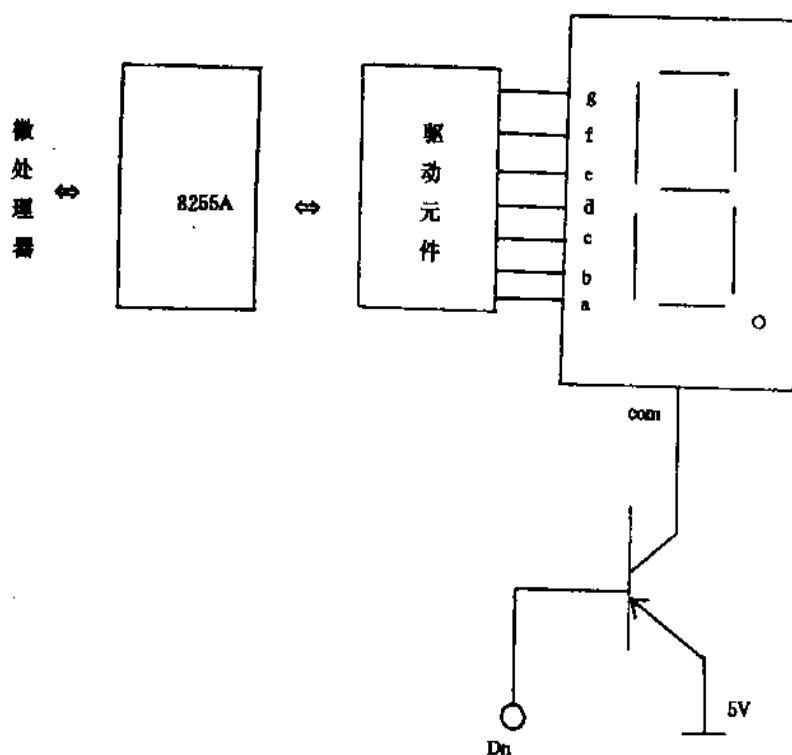


图 12.4 共阳极 7 段显示器

人类的眼睛通常有视觉暂留的现象，如果我们让 LED 一亮一暗，只要速度快一点，让肉眼不能察觉，即可达到亮的效果又可以省电。市面上有几种 LED 即是配合这种目的而设计的。

如图 12.3 共阴极 7 段 LED 显示器和图 12.4 共阳极 7 段 LED 显示器。

图 12.3 及 12.4 中的 a~g 为七个显示段的读入数据，Dn 则适时打开让显示器正常工作，随后又关掉、又打开，实际上它是一明一暗的，只是肉眼无法察觉。Dn 的工作情形如图 12.5，当 Dn 为低电平时可以启动共阳极显示器。

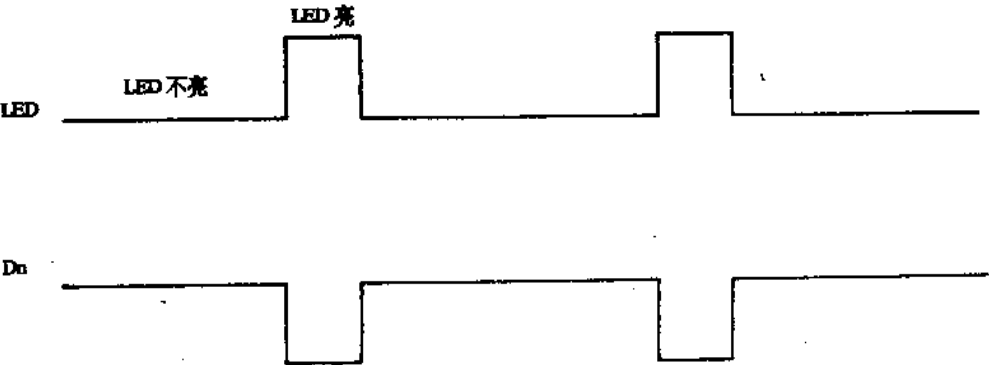


图 12.5 共阳极多路式七段 LED 显示器时序图

当 Dn 为 0 时才可以启动共阳极显示器。换句话说，只有在 Dn 为低电平时，LED 才起作用。请读者设计软件配合以上功能，此时 Dn 可以由一个端口位控制，如图 12.6。

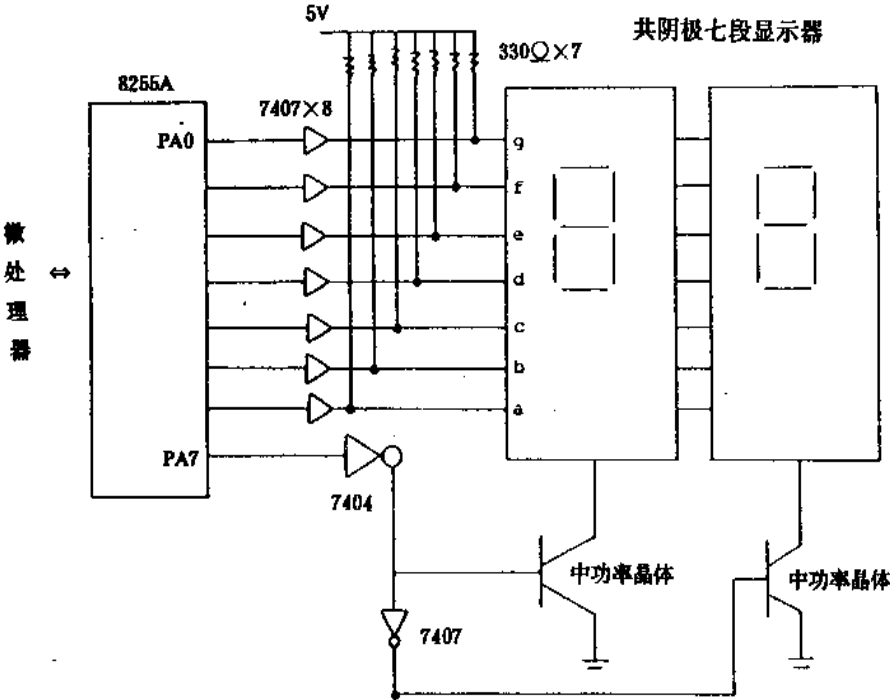


图 12.6 共阴极七段显示器接线图

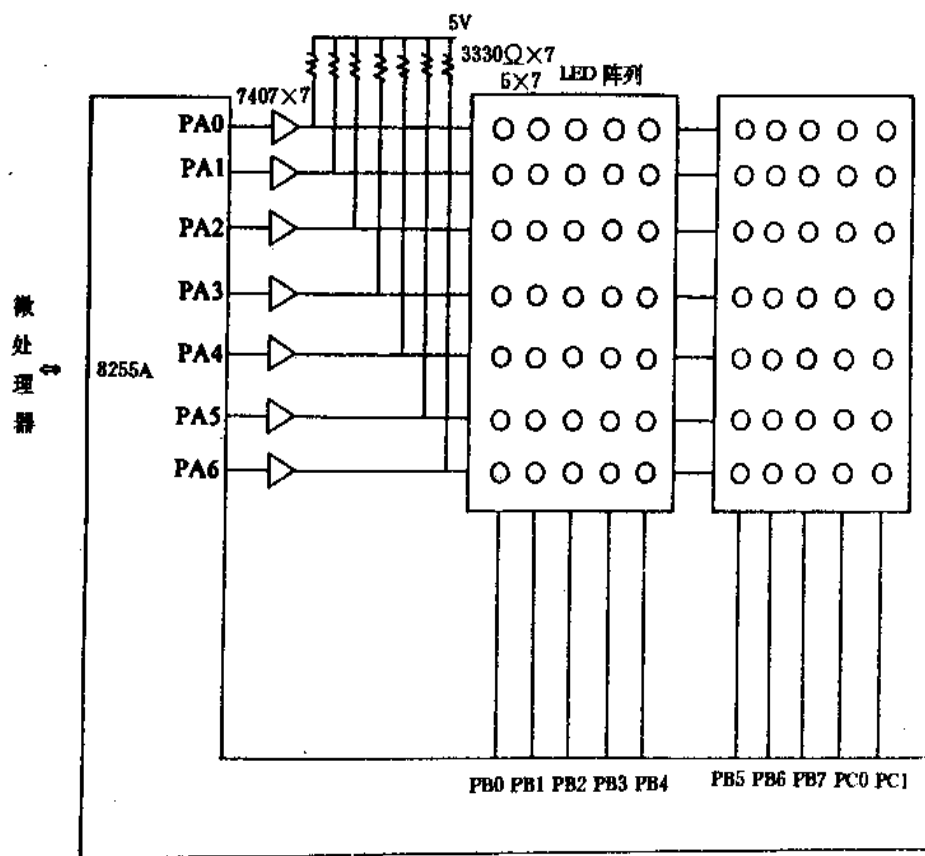
这种线路接法有几个好处:

1. 只利用 PA0~PA7 即可显示出两个字型。

2. 当 PA7=1 时, PA0~PA6 送出的数据在右边的 LED 上显示出来。而当 PA7=0 时, 另一批数据也由 PA0~PA6 在左边的 LED 中显示出来。利用改变 PA7 的方式我们可以送出两组数据, 而利用视觉暂留的原理可让肉眼当成有两种数据同时由微型处理机送出。

提示 3:

图 12.7 为 5×7 的点阵显示器, PA0~PA6 负责送出数据, PB0~PB7 及 PC0、PC1 则分别扫描。当扫描信号为“1”时, 相对应的 LED 才会亮。这种显示器和多路显示器非常类似, 只不过它具有更多的扫描线。



★注: 某些5×7LED方阵须加入缓冲区(如7407)用以驱动PB0~PB7及PC0、PC1。

图 12.7 5×7 点阵显示图线路

图 12.8 为  $5 \times 7$  点阵的驱动电路，图 12.9 即为它的等价电路。

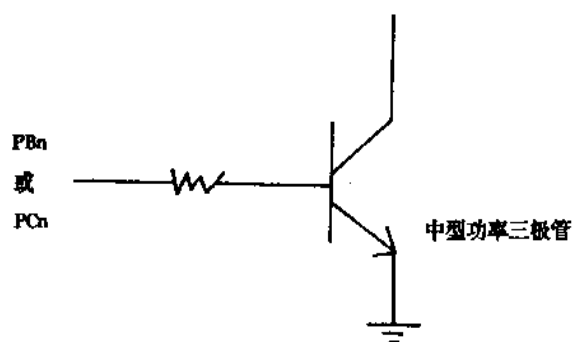


图 12.8 驱动电路

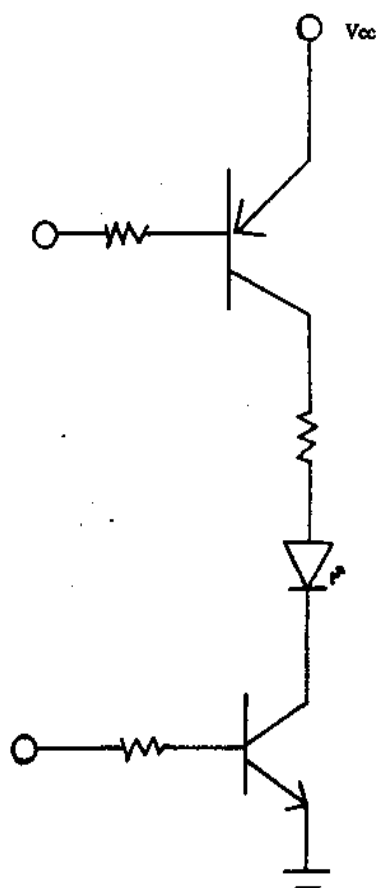


图 12.9 一个点的等价电路

PA0~PA6 送出数据信号，配合扫描信号点亮 LED。

PB0~PB7 及 PC0~PC1 送出扫描信号以利省电。



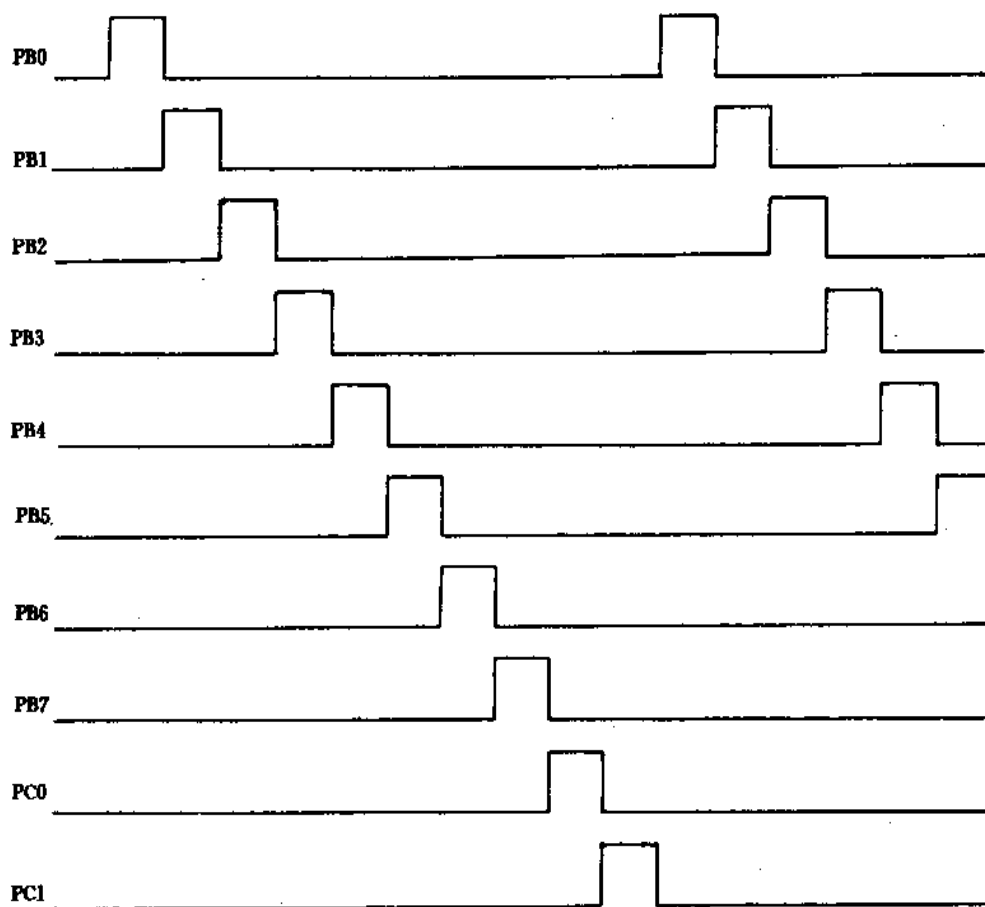


图 12.10 扫描信号时序图

请读者为这种显示器写出应用软件。

## 第十三章 方波发生器专题

### 本章实验目的

1. 通过此实验，读者可以了解：

■ IBM PC I/O 控制

■ 方波发生方式的处理

### 本章内容

13.1 工作目标

13.2 硬件设计

13.3 软件设计

13.4 结论

13.5 思考题

### 13.1 工作目标

在 Port 上显示出方波波形，并以示波器测量它，同时也在计算机屏幕显示出来。表 13.1 为由 PA0~PA7 所产生的真值表，图 13.1 为它的时序图。

表 13.1 PA0~PA7 的真值表

	PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7
T=0	0	0	0	0	0	0	0	0
T=t	1	0	0	0	0	0	0	0
T=2t	0	1	0	0	0	0	0	0
T=3t			...					
...			...					
...			...					
...	1	0	1	1	1	1	1	1
...	0	1	1	1	1	1	1	1
T=255t	1	1	1	1	1	1	1	1

图 13.1 为 PA0~PA7 在 IBM PC 屏幕上及示波器上所显示的图形。

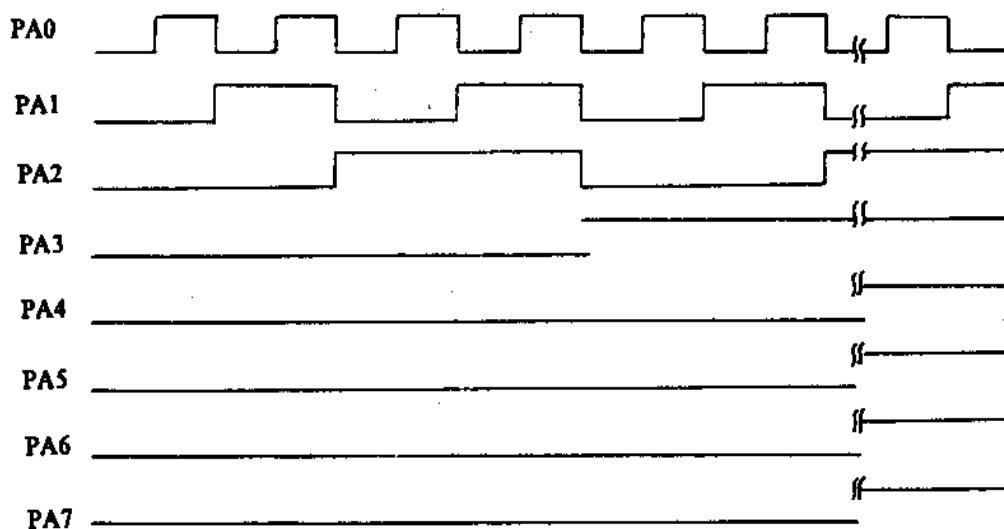


图 13.1 PA0~PA7的时序图

## 13.2 硬件设计

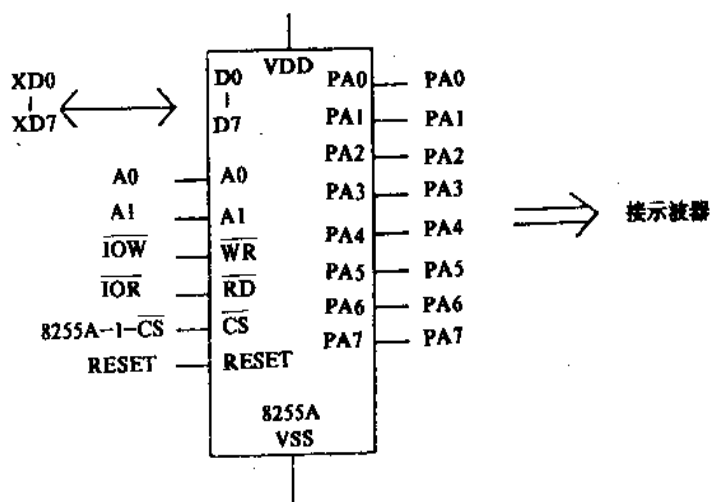


图 13.2 此专题所使用的硬件线路

在PA0~PA7上分别用示波器观察它的波形。

## 13.3 软件设计

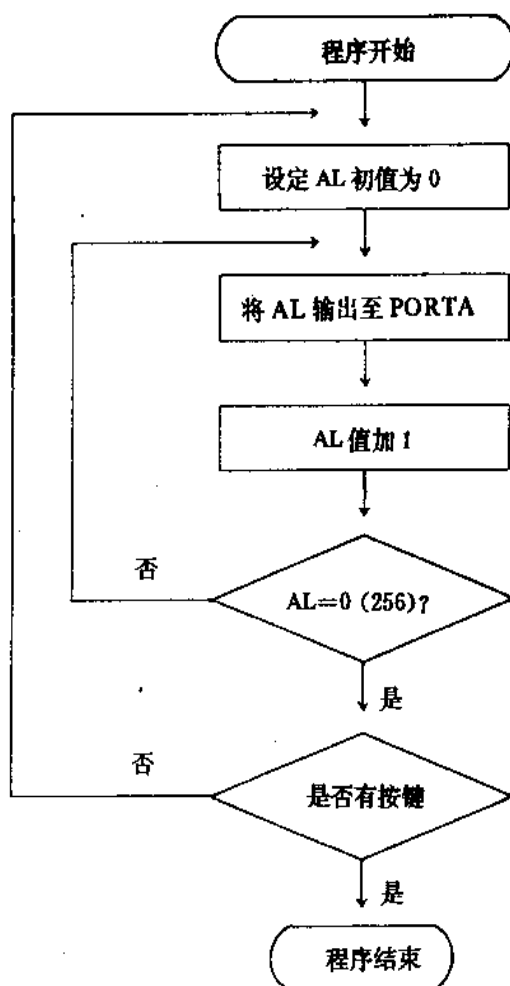
此部分包含了两个软件:

1. SQUARE1.ASM 单为 PORT 的输出。
2. SQUARE2.ASM 则另包含了屏幕上的显示。

示例程序 square1.c

本程序主要是将 0 至 255 的值依次输出 PORTA，这样对 PA0~PA7 (若值是 1 表示高电平，值是 0 表示低电平) 而言，便可达到方波的效果。

本程序的设计流程如下图所示：



```

01 /* ----- */
02 /*      Program Name : square1.c      */
03 /*      Square Wave Generation 1.      */
04 /*      For 8255A, Mode 0 application.  */
05 /* ----- */
06 #include <dos.h>
07 #include <conio.h>
08 #define SETPORT 0x3e3
09 #define PORTA 0x3e0
10 void main( )
11 {
12     unsigned int bytewrite;
13

```

```

14 /* set up the PORTA as output PORT */
15     bytewrite = 0x80;
16     outportb(SETPORT,bytewrite);
17
18 /* generate the square wave */
19     while ( !kbhit( ) )
20     {
21         for ( bytewrite = 0; bytewrite <= 255; bytewrite++ )
22             outportb(PORTA,bytewrite);
23     }
24 }

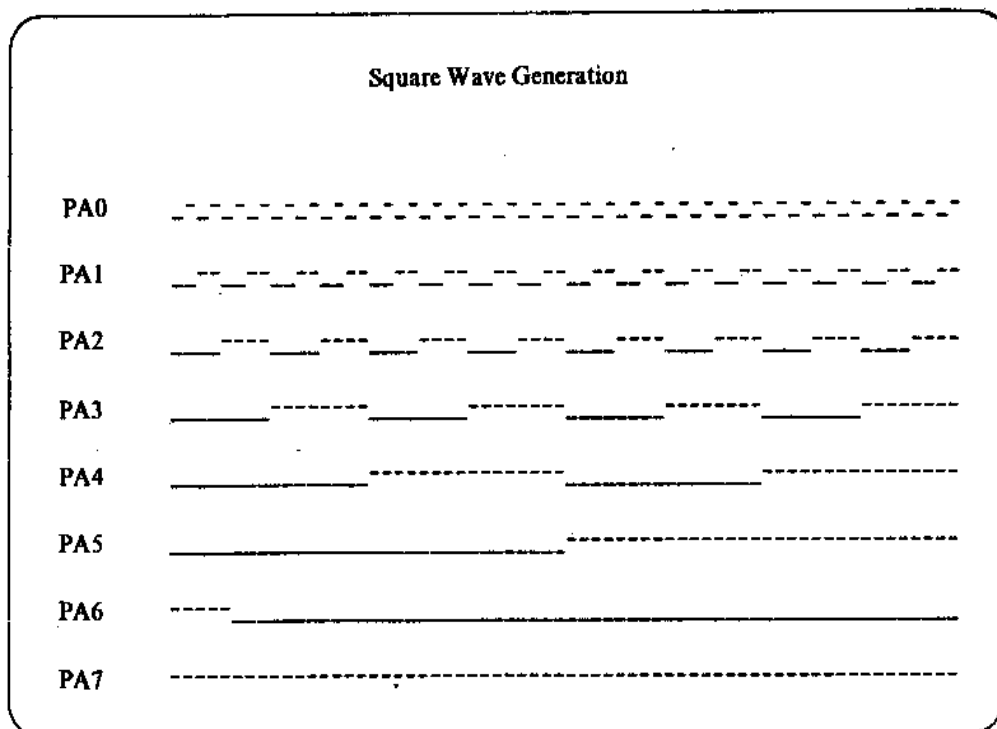
```

#### 示例程序 square1.c 解释

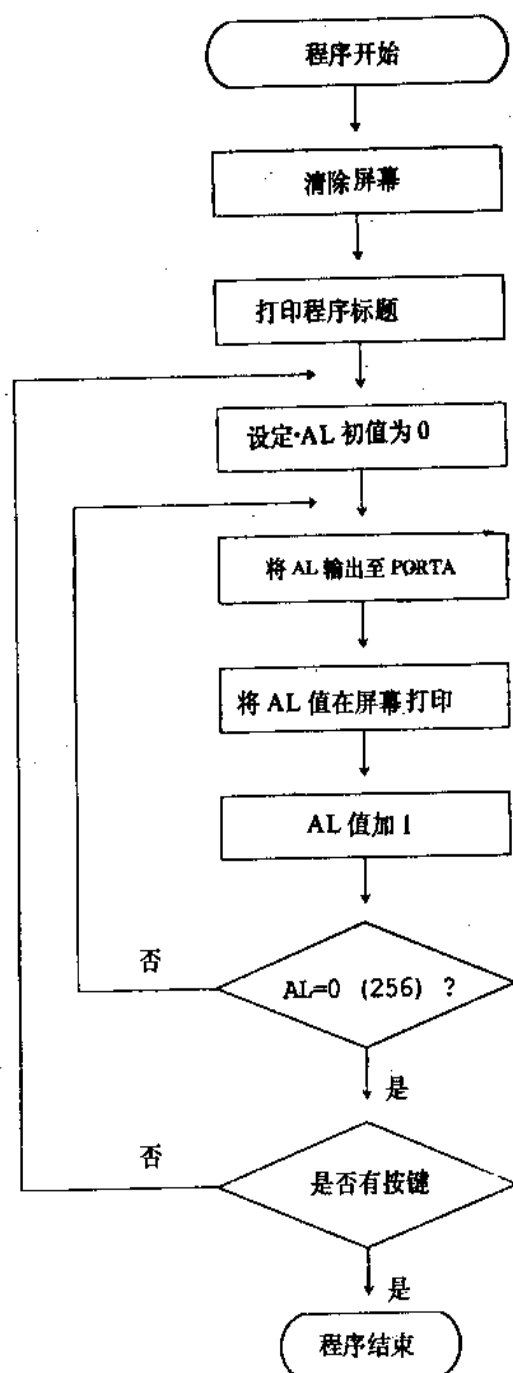
1. 第 8 行是用于设置 SETPORT 值为控制寄存器口，其值是 0x3e3。
2. 第 9 行是设置 PORTA 口，值是 0x3e0。
3. 第 15 行至第 16 行是设置 PORTA 供输出使用。
4. 第 21 行至第 22 行是一个循环，此循环会将 0 至 255 间的值依次送至 PORTA，便可产生方波。
5. 第 19 行至第 23 行是一个循环，循环主要是检查是否有键盘输入，如果有则离开此循环，程序将结束。

#### 示例程序 square2.c

本程序基本上是上一个程序的扩充，和前一个程序最大的不同在于，本程序多了将方波显示在屏幕的功能。



上图是方波在屏幕显示的图形。  
本程序的工作流程如下所示：



```

01 /* ----- */
02 /*      Program Name : square2.c      */
03 /*      Square Wave Generation with screen output. */
04 /*      For 8255A, Mode 0 application. */

```

```

05  / * ----- * /
06  #include <dos.h>
07  #include <conio.h>
08  #define SETPORT 0x3c3
09  #define PORTA 0x3c0
10  void main( )
11  {
12      void displaybit( );
13      unsigned int bytewrite;
14
15      clrscr( );          / * clear the screen * /
16
17      / * print the screen documentation * /
18      gotoxy(26,1);
19      printf("Square Wave Generation");
20      gotoxy(1,5);
21      printf("PA0");
22      gotoxy(1,7);
23      printf("PA1");
24      gotoxy(1,9);
25      printf("PA2");
26      gotoxy(1,11);
27      printf("PA3");
28      gotoxy(1,13);
29      printf("PA4");
30      gotoxy(1,15);
31      printf("PA5");
32      gotoxy(1,17);
33      printf("PA6");
34      gotoxy(1,19);
35      printf("PA7");
36      / * set up the PORTA as output PORT * /
37      bytewrite = 0x80;
38      outportb(SETPORT,bytewrite);
39
40      / * generate the square wave * /
41      while ( !kbhit( ) )
42      {
43          for ( bytewrite = 0; bytewrite <= 255; bytewrite++ )
44          {
45              outportb(PORTA,bytewrite);
46              displaybit(bytewrite);
47          }
48      }

```

```

49 }
50 / * _____ */
51 / * Display the bit */
52 / * _____ */
53 void displaybit(int bytedata)
54 {
55     int col,val;
56
57     col = bytedata % 64;
58     col += 10;
59
60     / * print bit 7 * /
61     gotoxy(col,19);
62     val = 0x80 & bytedata;
63     if ( val == 0x80 )
64         printf(" ");      / * print upper * /
65     else
66         printf("_");      / * print lower * /
67
68     / * print bit 6 * /
69     gotoxy(col,17);
70     val = 0x40 & bytedata;
71     if ( val == 0x40 )
72         printf(" ");      / * print upper * /
73     else
74         printf("_");      / * print lower * /
75
76     / * print bit 5 * /
77     gotoxy(col,15);
78     val = 0x20 & bytedata;
79     if ( val == 0x20 )
80         printf(" ");      / * print upper * /
81     else
82         printf("_");      / * print lower * /
83
84     / * print bit 4 * /
85     gotoxy(col,13);
86     val = 0x10 & bytedata;
87     if ( val == 0x10 )
88         printf(" ");      / * print upper * /
89     else
90         printf("_");      / * print lower * /
91
92     / * print bit 3 * /

```



```

93     gotoxy(col,11);
94     val = 0x08 & bytedata;
95     if ( val == 0x08 )
96         printf("#-");          /* print upper */
97     else
98         printf("#_");          /* print lower */
99
100  /* print bit 2 */
101     gotoxy(col,9);
102     val = 0x04 & bytedata;
103     if ( val == 0x04 )
104         printf("#-");          /* print upper */
105     else
106         printf("#_");          /* print lower */
107
108  /* print bit 1 */
109     gotoxy(col,7);
110     val = 0x02 & bytedata;
111     if ( val == 0x02 )
112         printf("#-");          /* print upper */
113     else
114         printf("#_");          /* print lower */
115
116  /* print bit 0 */
117     gotoxy(col,5);
118     val = 0x01 & bytedata;
119     if ( val == 0x01 )
120         printf("#-");          /* print upper */
121     else
122         printf("#_");          /* print lower */
123 }

```

示例程序 square2.c 解释

1. 第 8 行是用于设置 SETPORT 值为控制寄存器的口，其值是 0x3e3。
2. 第 9 行是设置 PORTA 口值是 0x3e0。
3. 第 15 行是清除屏幕内容。
4. 第 16 行是将光标移至 (26,1) 位置。
5. 第 17 行是显示字符串 "Square Wave Generation"。
6. 第 20 行至第 21 行是将光标移至 (1,5)，然后显示字符串 "PA0"。
7. 第 22 行至第 23 行是将光标移至 (1,7)，然后显示字符串 "PA1"。
8. 第 24 行至第 25 行是将光标移至 (1,9)，然后显示字符串 "PA2"。
9. 第 26 行至第 27 行是将光标移至 (1,11)，然后显示字符串 "PA3"。
10. 第 28 行至第 29 行是将光标移至 (1,13)，然后显示字符串 "PA4"。

11. 第 30 行至第 31 行是将光标移至 (1,15), 然后显示字符串“PA5”.
12. 第 32 行至第 33 行是将光标移至 (1,17), 然后显示字符串“PA6”.
13. 第 34 行至第 35 行是将光标移至 (1,19), 然后显示字符串“PA7”.
14. 第 45 行是将方波值送至 PORTA.
15. 第 46 行是方波调用 `displaybit()` 函数显示此方波值.
16. 第 43 行至第 47 行是一个循环. 此循环会将 0 至 255 的值分别送至 PORTA 和屏幕上.
17. 第 41 行到第 48 行是一个循环. 此循环会检查是否有键盘按键, 如果有则离开此循环.

函数 `displaybit()` 解释:

1. 第 57 行是用于计算应在屏幕第几列 (变量名称是 `col`) 显示方波.
2. 第 61 行至第 66 行是将光标移至 (`col`,19) 位置显示第 7 位.
3. 第 69 行至第 74 行是将光标移至 (`col`,17) 位置显示第 6 位.
4. 第 77 行至第 82 行是将光标移至 (`col`,15) 位置显示第 5 位.
5. 第 85 行至第 90 行是将光标移至 (`col`,13) 位置显示第 4 位.
6. 第 93 行至第 98 行是将光标移至 (`col`,11) 位置显示第 3 位.
7. 第 101 行至第 106 行是将光标移至 (`col`,9) 位置显示第 2 位.
8. 第 109 行至第 114 行是将光标移至 (`col`,7) 位置显示第 1 位.
9. 第 117 行至第 122 行是将光标移至 (`col`,5) 位置显示第 0 位.
10. 在本函数中, 位 1 以负号(-)表示, 位 0 以下划线(\_)表示.

## 13.4 结 论

通过以上实例, 读者可以了解如何将 DATA 送到端口上, 以便测量出方波频率. 这种波形的送出可以进行任何仪器的控制, 如步进电机 CNC 车床, 以及有关微型计算机的其它应用.

## 13.5 思考题

1. 请算出 PA0~PA7 在示波器上的频率?
  2. 按这种模式, 可以送出的最高频率是多少?
  3. 如果 PA0=PA1, 让 PA2=0, 请修改程序.
  4. 如果 PA0=PA1=0, 让 PA2=1, 请修改程序.
  5. 如何利用 8253 产生相同的效果.
  6. 利用 8253 / 8254 做出同一效果的情形.
- 问题 3, 4 为一种数学模式的运算. 将运算结果显示在屏幕上.

## 第十四章 自动售货机专题

### 本实验目的

1. 通过此实验，读者可以了解

■ IBM PC I/O 控制

■ 更进一步的 C 语言软件设计技巧

### 本章内容

14.1 工作目标

14.2 硬件设计

14.3 软件设计

14.4 结论

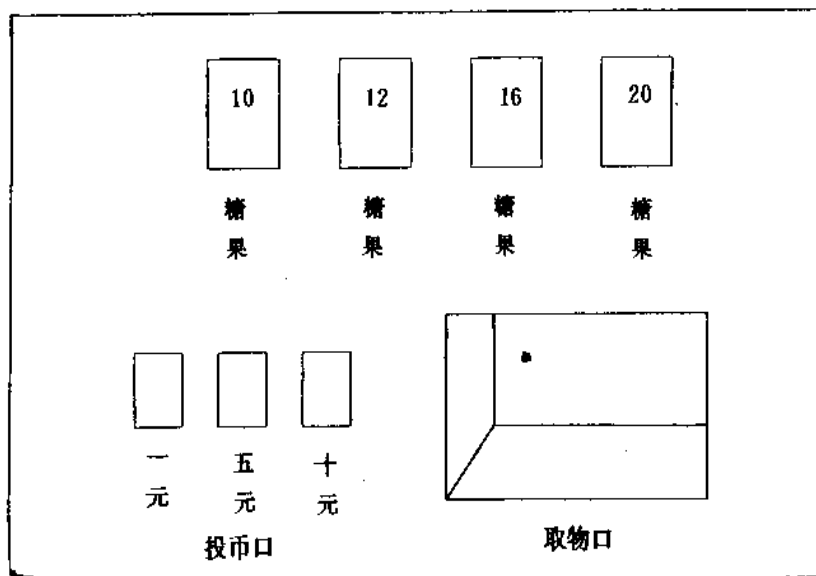
14.5 思考题

### 14.1 工作目标

市面上有很多的自动售货机都是用微型计算机控制的。本章将以简单方式让读者了解它们的原理。

本实验专题使用方式如下：

1. 用户投币。此项工作由SW5~SW7模拟投币的动作。当SW5~SW7被设置为1时，分别表示投入1，5，10元的硬币。如果在2秒钟之内SW5~SW7不被复位，则表示系统模拟成继续投币。



## 2. 选择糖果

- 若投入钱不足以买该物品，则计算机显示钱不够。
- 若钱足够则 PB0 所启动的 LED 亮，表示所选择的糖果掉出。

## 14.2 硬件设计

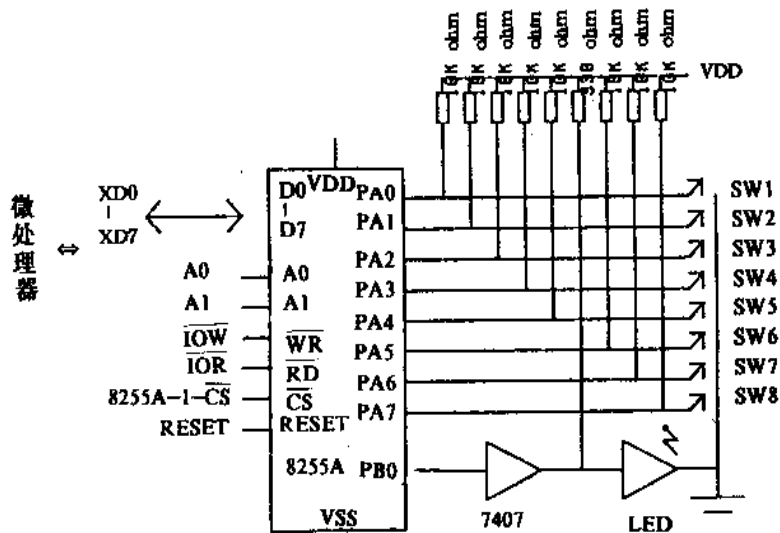


图 14.1 本专题所使用的自动售货机硬件

- SW1: 选择 10 元的糖果
- SW2: 选择 12 元的糖果
- SW3: 选择 16 元的糖果
- SW4: 选择 20 元的糖果
- SW5: 投 1 元硬币
- SW6: 投 5 元硬币
- SW7: 投 10 元硬币
- PB0 LED 亮表示所选择物品掉落。

## 14.3 软件设计

### 程序示例 seller.c

自动售货机程序设计。本程序在运行时屏幕将如下所示:

**Automatic Seller Machine**

**PA0 : CANDY 1 --- 10 dollars 0**

**PA1 : CANDY 2 --- 12 dollars 0**

**PA2 : CANDY 3 --- 16 dollars 0**

**PA3 : CANDY 4 --- 20 dollars 0**

**PA4 : CANDY 5 --- 22 dollars 0**

**PA5 : insert 1 dollar 0**

**PA6 : insert 5 dollar 0**

**PA7 : insert 10 dollar 0**

**Total : 0**

从上图可以很容易得知 PA0~PA7 选择项的意义:

PA0=1, 选择单价 10 元的糖果 1

PA1=1, 选择单价 12 元的糖果 2

PA2=1, 选择单价 16 元的糖果 3

PA3=1, 选择单价 20 元的糖果 4

PA4=1, 选择单价 22 元的糖果 5

PA5=1, 投入一元硬币

PA6=1, 投入五元硬币

PA7=1, 投入十元硬币

同时本程序在运行过程中, 不管你是选择那一项, 其右边的笑脸均会以黑色显示一秒钟。例如当你投入一个 10 元硬币之后, 首先黑色笑脸会闪一次, 然后屏幕将如下所示:

**Automatic Seller Machine**

**PA0 : CANDY 1 --- 10 dollars 0**

**PA1 : CANDY 2 --- 12 dollars 0**

PA2 : CANDY 3 --- 16 dollars 0

PA3 : CANDY 4 --- 20 dollars 0

PA4 : CANDY 5 --- 22 dollars 0

PA5 : Insert 1 dollar 0

PA6 : Insert 5 dollar 0

PA7 : Insert 10 dollar 0

Total : 10

若再投入一个 5 元硬币之后，屏幕将如下所示：

#### Automatic Seller Machine

PA0 : CANDY 1 --- 10 dollars 0

PA1 : CANDY 2 --- 12 dollars 0

PA2 : CANDY 3 --- 16 dollars 0

PA3 : CANDY 4 --- 20 dollars 0

PA4 : CANDY 5 --- 22 dollars 0

PA5 : Insert 1 dollar 0

PA6 : Insert 5 dollar 0

PA7 : Insert 10 dollar 0

Total : 15

此时若是你选糖果 2，则因总钱数大于糖果 2 的单价，因此屏幕将如下所示：

Automatic Seller Machine

PA0 : CANDY 1 --- 18 dollars 0

PA1 : CANDY 2 --- 12 dollars 0

PA2 : CANDY 3 --- 16 dollars 0

PA3 : CANDY 4 --- 28 dollars 0

PA4 : CANDY 5 --- 22 dollars 0

PA5 : Insert 1 dollar 0

PA6 : Insert 5 dollar 0

PA7 : Insert 10 dollar 0

You get the candy.

return 3

同时 PB0 的灯会亮，如果你所投入的钱数小于糖果的单价数，则屏幕将如下所示：

Automatic Seller Machine

PA0 : CANDY 1 --- 18 dollars 0

PA1 : CANDY 2 --- 12 dollars 0

PA2 : CANDY 3 --- 16 dollars 0

PA3 : CANDY 4 --- 28 dollars 0

PA4 : CANDY 5 --- 22 dollars 0

PA5 : Insert 1 dollar 0

PA6 : Insert 5 dollar 0

PA7 : Insert 10 dollar 0

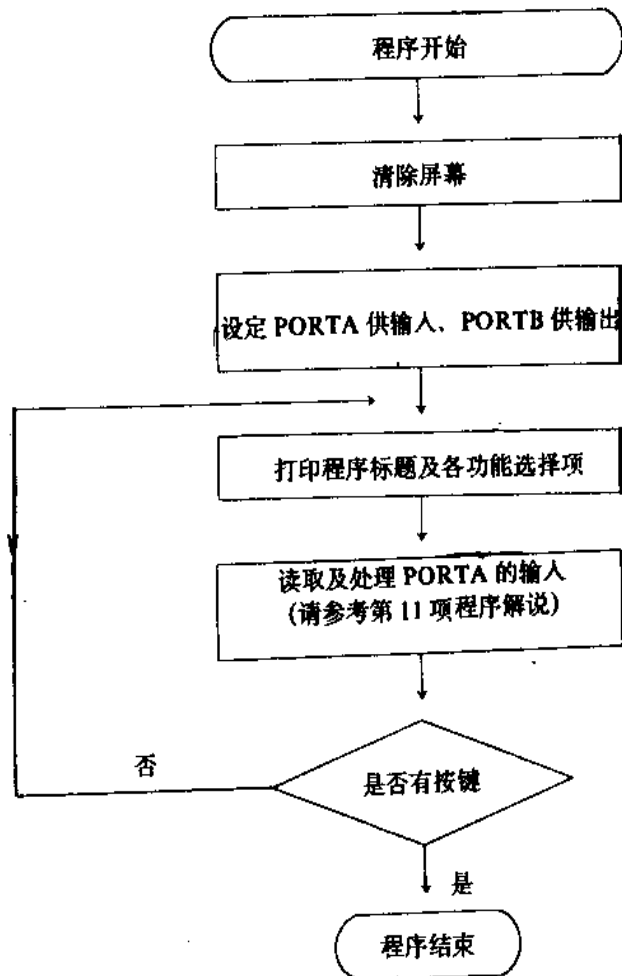
Total : 18

Money is not enough, please insert money again.

碰上此种情形，只要再继续投币就可以了。每当一次交易之后，屏幕自动清除，然后显示下列信息。

Press any key to exit.

此时只要随意按一个键便可以让程序自动结束。本程序的工作流程如下所示：



```

01  /* ----- */
02  /*      Program Name : seller.c      */
03  /*      Automatic Seller Simulation with Screen Function      */
04  /*      For 8255A, Mode 0, PORTA input and PORTB output      */
05  /* ----- */
06  #include <dos.h>
07  #include <conio.h>
08  #define SETPORT      0x3e3
  
```



```

09 #define  PORTA      0x3e0
10 #define  PORTB      0x3e1
11 int      money;
12 int      ret_money;
13 void main( )
14 {
15     void initialport( );
16     void getporta( );
17     void display( );
18     void ptitle( );
19     unsigned char bytewrite,byteread;
20     unsigned char light;
21     int i;
22
23     clrscr( );          /* clear the screen */
24
25     /* set up the PORTA input and PORTB output */
26     bytewrite = 0x90;
27     outportb(SETPORT,bytewrite);
28
29     /* running the light */
30     while ( !kbhit( ) )
31     {
32         initialport( );
33         ptitle( );
34         display( );
35         gotoxy(36,23);    /* initial the total money */
36         printf("Total :  0");
37         getporta( );
38         sleep(3);
39         clrscr( );
40         gotoxy(29,12);
41         printf("Press any key to exit.");
42         sleep(3);
43     }
44 }
45 /* ----- */
46 /*  Get the input data from PORTA */
47 /* ----- */
48 void getporta( )
49 {
50     void outok( );
51     void outno( );
52     void distotal( );

```

```

53     unsigned char byteread, bytetest;
54
55     money = 0;           /* initial the input money */
56
57     /* make sure there is input happen */
58     while ( 1 )
59     {
60         byteread = inportb(PORTA);
61         bytetest = 0xFF & byteread;
62         if ( bytetest == 0 )      /* no input */
63             continue;
64
65         bytetest = byteread & 0x80;
66         if ( bytetest == 0x80 )    /* PA7 = 1 ON */
67         {
68             money += 10;
69             distotal( );           /* display the total money */
70             gotoxy(56,20);
71             printf("\n2");
72             delay(500);
73             gotoxy(56,20);
74             printf("\n1");
75             sleep(1);
76             continue;
77         }
78
79         bytetest = byteread & 0x40;
80         if ( bytetest == 0x40 )    /* PA6 = 1 ON */
81         {
82             money += 5;
83             distotal( );           /* display the total money */
84             gotoxy(56,18);
85             printf("\n2");
86             delay(500);
87             gotoxy(56,18);
88             printf("\n1");
89             sleep(1);
90             continue;
91         }
92
93         bytetest = byteread & 0x20;
94         if ( bytetest == 0x20 )    /* PA5 = 1 ON */
95         {
96             money += 1;

```

```

97         distotal( );                /* display the total money */
98         gotoxy(56,16);
99         printf("\n2");
100        delay(500);
101        gotoxy(56,16);
102        printf("\n1");
103        sleep(1);
104        continue;
105    }
106
107    bytetest = bytetest & 0x10;
108    if ( bytetest == 0x10 )            /* PA4 = 1 ON */
109    {
110        gotoxy(56,12);
111        printf("\n2");
112        delay(500);
113        gotoxy(56,12);
114        printf("\n1");
115        if ( money < 22 )
116        {
117            outno( );                /* output money is not enough */
118            continue;
119        }
120        else
121        {
122            ret_money = money - 22;
123            outok( );                /* you got the candy */
124            break;
125        }
126    }
127
128    bytetest = bytetest & 0x08;
129    if ( bytetest == 0x08 )            /* PA3 = 1 ON */
130    {
131        gotoxy(56,10);
132        printf("\n2");
133        delay(500);
134        gotoxy(56,10);
135        printf("\n1");
136        if ( money < 20 )
137        {
138            outno( );                /* output money is not enough */
139            continue;
140        }

```

```

141     else
142     {
143         ret_money = money - 20;
144         outok( );          /* you got the candy */
145         break;
146     }
147 }
148
149 bytetest = bytread & 0x04;
150 if ( bytetest == 0x04 ) /* PA2 = 1 ON */
151 {
152     gotoxy(56,8);
153     printf("\2");
154     delay(500);
155     gotoxy(56,8);
156     printf("\1");
157     if ( money < 16 )
158     {
159         outno( );          /* output money is not enough */
160         continue;
161     }
162     else
163     {
164         ret_money = money - 16;
165         outok( );          /* you got the candy */
166         break;
167     }
168 }
169
170 bytetest = bytread & 0x02;
171 if ( bytetest == 0x02 ) /* PA1 = 1 ON */
172 {
173     gotoxy(56,6);
174     printf("\2");
175     gotoxy(56,6);
176     delay(500);
177     printf("\1");
178     if ( money < 12 )
179     {
180         outno( );          /* output money is not enough */
181         continue;
182     }
183     else
184     {

```

```

185         ret_money = money - 12;
186         outok( );          /* you got the candy      */
187         break;
188     }
189 }
190
191 bytetest = bytetest & 0x01;
192 if ( bytetest == 0x01 )    /* PA0 = 1 ON */
193 {
194     gotoxy(56,4);
195     printf("\n2");
196     delay(500);
197     gotoxy(56,4);
198     printf("\n1");
199     if ( money < 10 )
200     {
201         outno( );          /* output money is not enough */
202         continue;
203     }
204     else
205     {
206         ret_money = money - 10;
207         outok( );          /* you got the candy      */
208         break;
209     }
210 }
211 }
212 }
213 /* ----- */
214 /* Let PB0 = 1, and print you get the candy */
215 /* ----- */
216 void outok( )
217 {
218     unsigned char bytewrite;
219
220     gotoxy(33,23);
221     printf("You get the candy."); /* print you get the candy */
222
223     gotoxy(36,25);
224     printf("return %2d",ret_money); /* print the result money */
225
226     bytewrite = 0x01;
227     outportb(PORTB,bytewrite); /* let PB0 = 1 */
228 }

```

```

229 / * ----- */
230 / *   Output the money is not enough   */
231 / * ----- */
232 void outno( )
233 {
234     gotoxy(18,24);
235     printf("Money is not enough, please insert money again.");
236     sleep(1);
237     gotoxy(18,24);
238     printf("
239 }
240 / * ----- */
241 / *   Display the total money on the screen   */
242 / * ----- */
243 void distotal( )
244 {
245     gotoxy(45,23);
246     printf("%3d",money);
247 }
248 / * ----- */
249 / * ----- */
250 / * ----- */
251 / *   Display the Selection Board   */
252 / * ----- */
253 void display( ){
254     gotoxy(26,4);
255     printf("PA0 : CANDY 1 — 10 dollars \1");
256     gotoxy(26,6);
257     printf("PA1 : CANDY 2 — 12 dollars \1");
258     gotoxy(26,8);
259     printf("PA2 : CANDY 3 — 16 dollars \1");
260     gotoxy(26,10);
261     printf("PA3 : CANDY 4 — 20 dollars \1");
262     gotoxy(26,12);
263     printf("PA4 : CANDY 5 — 22 dollars \1");
264     gotoxy(26,16);
265     printf("PA5 : Insert 1 dollar \1");
266     gotoxy(26,18);
267     printf("PA6 : Insert 5 dollar \1");
268     gotoxy(26,20);
269     printf("PA7 : Insert 10 dollar \1");
270 }
271 / * ----- */
272 / *   initial the PORTA and PORTB as 0   */

```

```

273 / * ----- */
274 void initialport( )
275 {
276     unsigned char bytewrite;
277
278     bytewrite = 0x00;
279     outportb(PORTB,bytewrite);
280     outportb(PORTA,bytewrite);
281 }
282 / * ----- */
283 / * print the program title */
284 / * ----- */
285 void ptitle( )
286 {
287     gotoxy(29,1);
288     printf("Automatic Seller Machine");
289 }

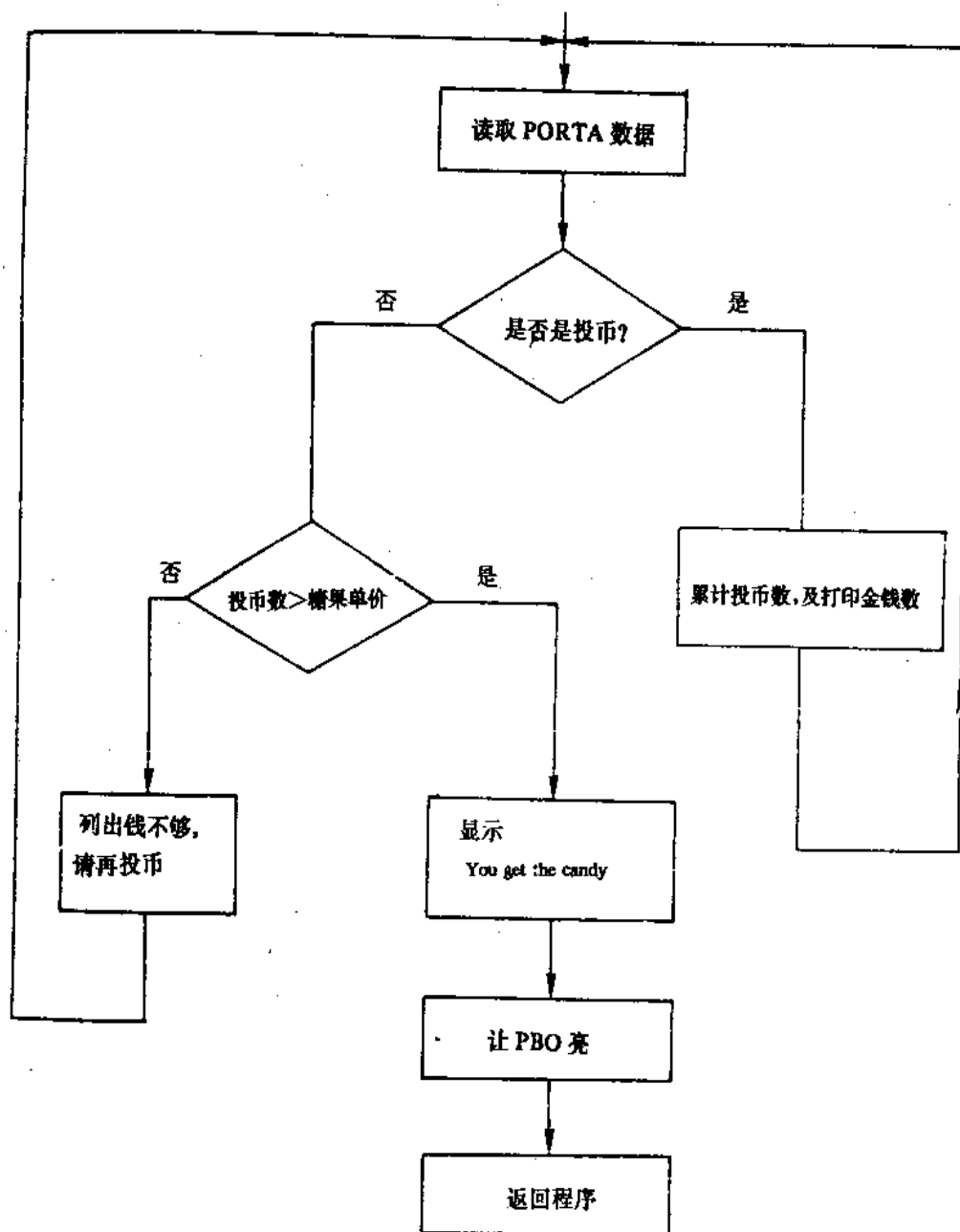
```

程序示例 seller.c 解释:

1. 第 8 行是设置 SETPORT 值为控制寄存器的口, 其值是 0x3c3.
2. 第 9 行是设置 PORTA 口值是 0x3c0.
3. 第 10 行是设置 PORTB 口值是 0x3c1.
4. 第 23 行是清除屏幕内容.
5. 第 26 至第 27 行是用于设置 PORTA 专供输入, PORTB 专供输出使用.
6. 第 32 行是调用 initialport( )函数, 令 PORTA 和 PORTB 值都为零.
7. 第 33 行是调用 ptitle( )函数显示程序标题.
8. 第 34 行是调用 display( )函数显示各种糖果单价及所投硬币字符串表.
9. 第 35 行是将光标移至 (36,23) 位置.
10. 第 36 行是显示字符串 "Total:0".
11. 第 37 行是调用 getporta( )函数, 本函数的工作流程如下页图.
12. 第 38 行是令程序暂停 3 秒.
13. 第 39 行是清除屏幕.
14. 第 40 行是将光标移至 (29,12) 位置.
15. 第 41 行是显示字符串 "Press any key to exit.".
16. 第 42 行是令程序暂停 3 秒.
17. 第 30 行至第 43 行是一个循环, 主要是检查是否有键盘输入. 如果有则离开循环.

函数 getporta( )解释:

1. 第 55 行是令变量 money 等于零.
2. 第 60 行是读取 PORTA 口数据.
3. 第 61 行至第 63 行是检查 PORTA 是否有输入, 如果没有则返回 60 行否则往下运行.



4. 第66行至第77行是检查是否投一枚10元 (PA7=1), 如果是则累计钱数及调用 distotal( )函数予以显示。
5. 第79行至第91行是检查是否投一枚5元 (PA6=1), 如果是则累计钱数及调用 distotal( )函数予以显示。
6. 第93行至第105行是检查是否投一枚一元 (PA5=1), 如果是则累计钱数, 及调用 distotal( )函数予以显示。
7. 第107行至第126行是查看是否选22元的糖果 (PA4=1), 如果是则检查累计钱数是否大于糖果单价, 如果是则调用 outok( )函数, 如果累计数小于糖果单价则调用 outno( )函数。



8. 第128行至第147行是检查是否选20元的糖果 (PA3=1), 如果是则检查累计钱数是否大于糖果单价, 如果是则调用 outok( )函数, 如果累计钱数小于糖果单价则调用 outno( )函数。
9. 第149行至第168行是检查是否选16元的糖果 (PA2=1), 如果是则检查累计钱数是否大于糖果单价, 如果是则调用 outok( )函数, 如果累计钱数小于糖果单价则调用 outno( )函数。
10. 第170行至第189行是检查是否选12元的糖果 (PA1=1), 如果是则检查累计钱数是否大于糖果单价, 如果是则调用 outok( )函数, 如果累计钱数小于糖果单价则调用 outno( )函数。
11. 第191行至第211行是检查是否选10元的糖果 (PA0=1), 如果是则检查累计钱数是否大于糖果单价, 如果是则调用 outok( )函数, 如果累计钱数小于糖果单价则调用 outno( )函数。

函数 outok 解释:

1. 第 220 行是将光标移至 (33,23) 位置。
2. 第 221 行是显示字符串“You get the candy”。
3. 第 223 行是将光标移至 (36,25) 位置。
4. 第 224 行是显示退钱币数。
5. 第 226 行至第 227 行是令 PORTB 的 PB0=1。

函数 outno( )解释:

1. 第 234 行是将光标移至 (18,24) 位置。
2. 第 235 行是显示字符串“Money is not enough, please insert money again.”
3. 第 236 行是令时间暂停 1 秒。
4. 第 237 行是将光标移至 (18,24) 位置。
5. 第 238 行是显示空格字符串, 相当于将 235 行所显示字符串删除。

函数 distotal( )解释:

1. 第 245 行是将光标移至 (45,23) 位置。
2. 第 246 行是显示累计投钱数。

函数 display( )解释:

1. 第 255 行是将光标移至 (26,4) 位置。
2. 第 256 行是显示字符串“PA0:CANDY1...10 dollars”。
3. 第 257 行是将光标移至 (26,6) 位置。
4. 第 258 行是显示字符串“PA1:CANDY2...12 dollars”。
5. 第 259 行是将光标移至 (26,8) 位置。
6. 第 260 行是显示字符串“PA2:CANDY3...16 dollars”。
7. 第 261 行是将光标移至 (26,10) 位置。
8. 第 262 行是显示字符串“PA3:CANDY4...20 dollars”。
9. 第 263 行是将光标移至 (26,12) 位置。
10. 第 264 行是显示字符串“PA4:CANDY5...22 dollars”。
11. 第 265 行是将光标移至 (26,16) 位置。

12. 第 266 行是显示字符串 "PA5:Insert 1 dollars".
13. 第 267 行是将光标移至 (26,18) 位置.
14. 第 268 行是显示字符串 "PA6:Insert 5 dollars".
15. 第 269 行是将光标移至 (26,20) 位置.
16. 第 270 行是显示字符串 "PA7:Insert 10 dollars".

函数 initialport( ) 解释:

1. 第 279 行是令 bytewrite 值为零.
2. 第 280 行是将零送至 PORTB 口.
3. 第 281 行是将零送至 PORTA 口.

函数 ptitle( ) 解释:

1. 第 288 行是将光标移至 (29,1) 位置.
2. 第 289 行是显示字符串 "Automatic seller Machine"

## 14.4 结 论

微型计算机处理的自动售货机可以解决非常复杂的问题, 如思考题所述.

## 14.5 思考题

1. 如果钱投太多, 应如何设计出找钱的方式?
  2. 如果某项东西已售完了, 应如何配合软硬件设计?
  3. 如果机械故障, 应如何防止用户投币? 是否可以让机器自动拨电话通知售货机拥有人修理机器.
  4. 如果小偷在偷钱时震动机器太猛, 是否可以拉警报? 或自动报警?
  5. 如何防范停电时硬币和糖果全掉出来?
- 请读者思考, 让明天的自动售货机更好.

## 第十五章 防盗器专题

### 本实验目的

1. 通过此实验可以了解:

- IBM PC I/O 接口的控制
- IBM PC 声音的产生
- 基本防盗器设计概念

### 本章内容

- 15.1 工作目标
- 15.2 硬件设计
- 15.3 软件设计
- 15.4 结论
- 15.5 思考题

市面上已经有不少的防盗产品。本章将以 IBM PC 模拟防盗器。它的基本工作方式在于一个开关,当开关被盗贼误踩时,警报器则发出警报声,且警报灯全部闪动。我们另定义了一个开关用以关闭警报器。

### 15.1 工作目标

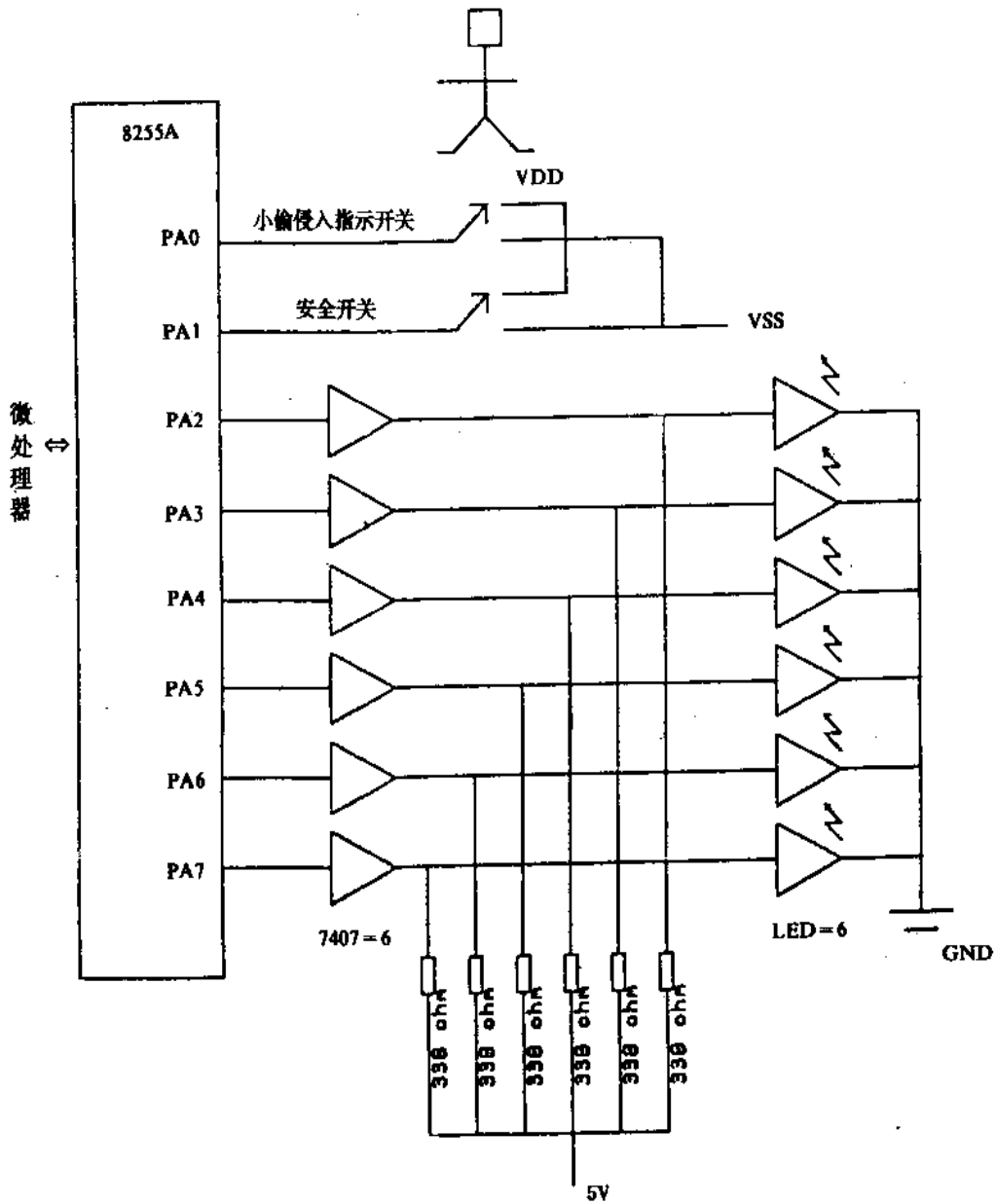
参考 15.2 定义 PA1 为安全开关,当  $PA1=0$  时,表示防盗器 OFF,也就是不工作,平时  $PA1=1$ 。

定义 PA0 为小偷指示开关,当  $PA0=0$  时表示小偷入侵。

定义 PA2~PA7 为小偷入侵显示,当小偷入侵时这 6 个 LED 全部闪动,并发出警报声。

在夜晚时将 PA1 设置为 1,当有小偷入侵时,小偷踩住 PA0 将 PA0 置为 0,此时 PA2~PA7 的 LED 灯号全部闪动,并由 IBM PC 发出警报声。

## 15.2 硬件设计



## 15.3 软件设计

### 程序示例 ALARM.ASM

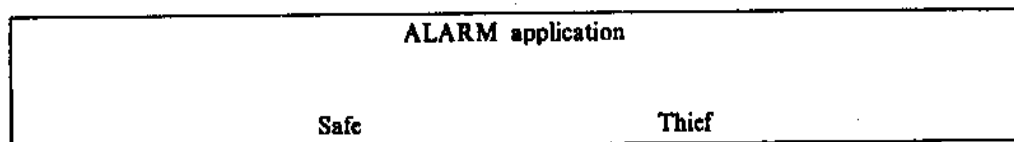
防盗器的设计。本程序在设计时遵照下列原则：

1. PA1 等于 0 表示防盗器是 OFF 状态。
2. PA1 等于 1 表示防盗器是 ON 状态。
3. 当 PA1 等于 1 时，同时 PA0 等于 1 表示没有小偷，若是 PA0 等于 0 时表示小偷入侵

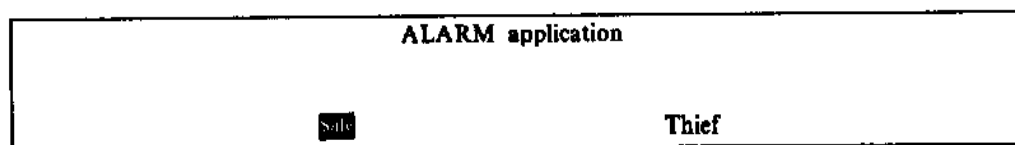
了，此时 PA2 至 PA7 的 LED 全部闪动。

此外，在设计本程序时，请先让 PA0 等于 1，以免当 PA1 是 ON 时，立即有小偷入侵的信号出现。

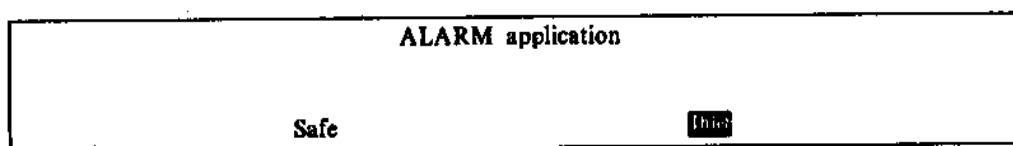
本程序运行时，首先屏幕将如下所示：



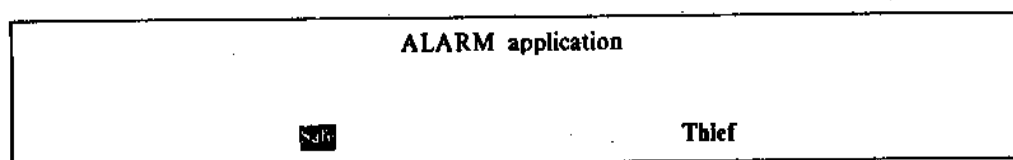
当 PA1 是 1，防盗器为 ON 状态时，屏幕将如下所示：



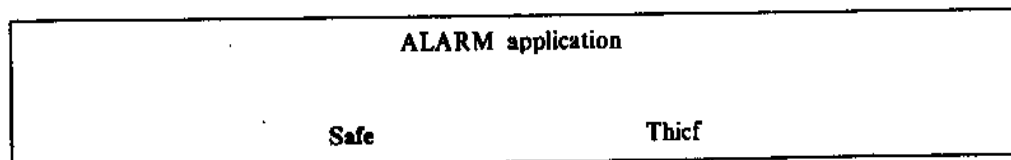
当防盗器为 ON 状态，若 PA0 是 0 时，PA2 至 PA7 会一直闪，且会发出警报声，同时屏幕将如下所示：



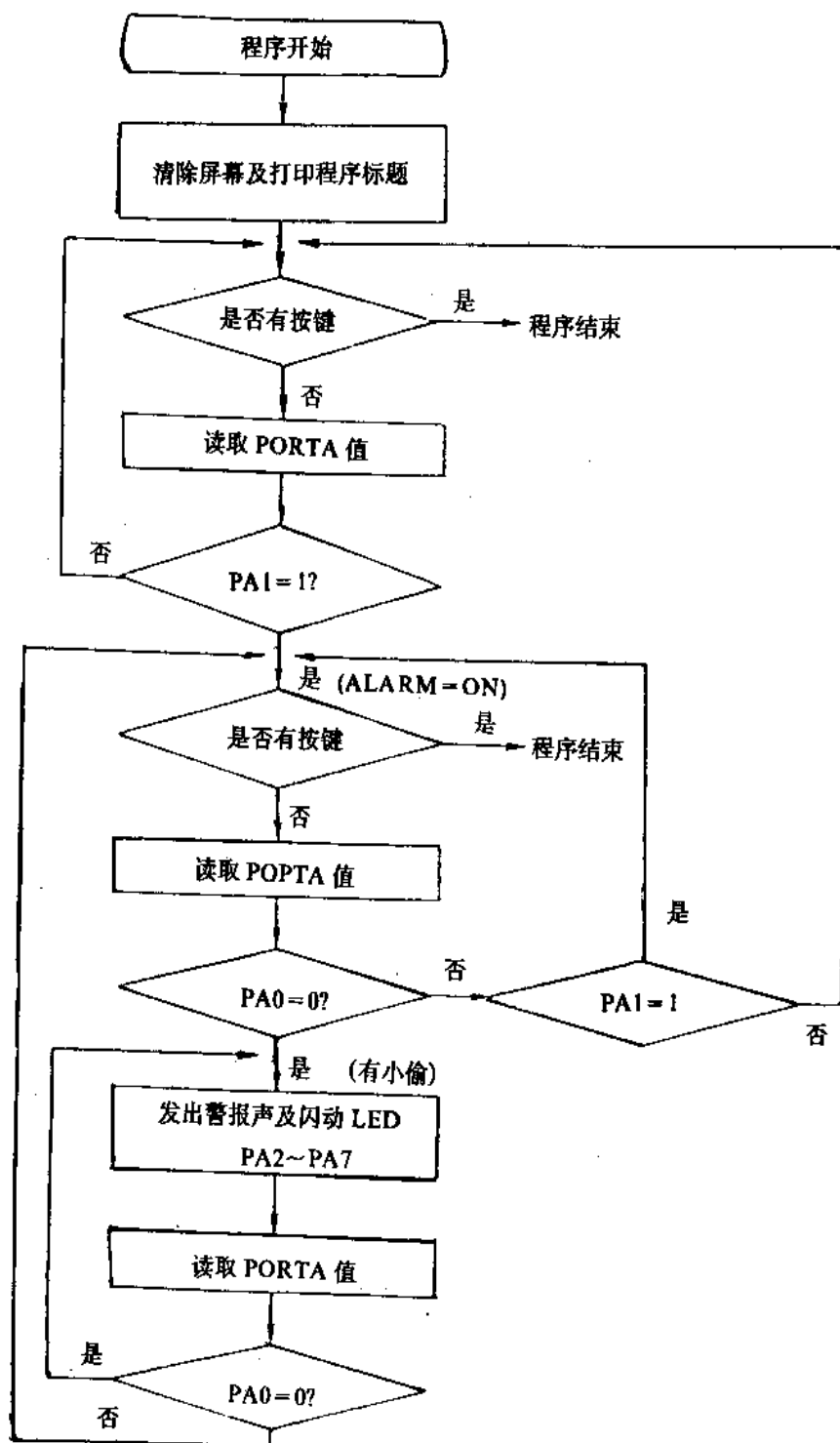
若此刻 PA0 为 1 时，PA2 至 PA7 会停止闪动，且声音也会暂停。此时屏幕将如下所示：



若此刻 PA1 是 0 时，防盗器将变成 OFF，屏幕将如下所示：



本程序工作流程如下所示：



```

01 /* ----- */
02 /*      Program Name : alarm.c      */
03 /*      Alarm application              */
04 /*      For 8255A, Mode 0 application.  */

```

```

05  / * ----- * /
06  #include <dos.h>
07  #include <conio.h>
08  #define SETPORT    0x3c3
09  #define PORTA      0x3c0
10
11  void main( )
12  {
13      void thief__is__coming( );
14      unsigned char byteread,bytewrite;
15
16      window(1,1,80,25);    / * set the screen as a window * /
17      clrscr( );           / * clear the screen * /
18      gotoxy(32,1);
19      cprintf("ALARM application");
20
21      while ( 1 )
22      {
23          / * print the safe and Thief condition *
24          textcolor(WHITE);
25          textbackground(BLACK);
26          gotoxy(18,4);
27          cprintf("Safe");
28          gotoxy(60,4);
29          cprintf("Thief");
30
31          / * set up the input mode * /
32          bytewrite = 0x90;
33          outportb(SETPORT,bytewrite);
34
35          / * check if there is input * /
36          if ( kbhit( ) != 0 )
37              break;
38
39          / * get PORTA PA1 bit message * /
40          byteread = inportb(PORTA);
41          byteread &= 0x02;    / * if PA1 == 0, mean alarm is OFF * /
42          if ( byteread == 0x00 )
43              continue;
44
45          / * set the alarm on screen condition * /
46          while ( 1 )
47          {
48              textcolor(BLACK);

```

```

49     textbackground(WHITE);
50     gotoxy(18,4);
51     cprintf("Safe");
52     textcolor(WHITE);
53     textbackground(BLACK);
54     gotoxy(60,4);
55     cprintf("Thief");
56
57  /* check if the Thief is coming */
58     bytread = inportb(PORTA);
59     bytread &= 0x01;          /* check bit 0 */
60     if ( bytread == 0x00 )
61         thief_is_coming( );
62
63     bytread = inportb(PORTA); /* check bit 1 */
64     bytread &= 0x02;
65     if ( bytread == 0x02 )
66         continue;
67     else
68         break;
69 }
70 }
71 }
72 /* ----- */
73 /* Thief is coming */
74 /* ----- */
75 void thief_is_coming( )
76 {
77     void alarmsound( );
78     unsigned char bytread, bytewrite;
79
80     while ( 1 )
81     {
82         textcolor(BLACK);
83         textbackground(WHITE);
84         gotoxy(60,4);
85         cprintf("Thief");
86         textcolor(WHITE);
87         textbackground(BLACK);
88         gotoxy(18,4);
89         cprintf("Safe");
90
91  /* set the control register as output mode */
92     bytewrite = 0x80;

```



```

93     outportb(SETPORT,bytewrite);
94
95  /* flashing the light and generate the alarm sound */
96     bytewrite = 0xfe;
97     outportb(PORTA,bytewrite); /* PA1 - PA7 on */
98     sleep(1);
99     bytewrite = 0x01;
100    outportb(PORTA,bytewrite); /* PA1 - PA7 off */
101    alarmsound( );
102
103  /* set the control register as input mode */
104     bytewrite = 0x88;
105     outportb(SETPORT,bytewrite);
106
107  /* get the input */
108     bytewrite = inportb(PORTA);
109     bytewrite &= 0x01;
110     if ( bytewrite == 0x00 )
111         continue;
112     else
113         break;
114 }
115 }
116 /* ----- */
117 /* Generate the alarm sound */
118 /* ----- */
119 void alarmsound( )
120 {
121     unsigned int freq;
122
123     for ( freq = 0; freq <= 1000; freq++ )
124     {
125         sound(freq);
126         delay(1);
127     }
128     nosound( )
129 }

```

程序示例 alarm.c 解释:

1. 第 8 行是用于设置 SETPORT 值为控制寄存器的口, 值是 0x3e3.
2. 第 9 行是设置 PORTA 口值是 0x3e0.
3. 第 16 行是设置窗口指令, 设置整个屏幕为一个窗口, 以便使 cprintf( ) 函数使用 (此函数可产生某些字符串以反白显示).

4. 第 17 行是清除屏幕内容。
5. 第 18 行是将光标移至 (32,1) 位置。
6. 第 19 行是显示字符串“ALARM application”。
7. 第 26 行至第 27 行是将光标移至 (18,4)，然后显示字符串“Safe”。
8. 第 28 行和第 29 行是将光标移至 (60,4)，然后显示字符串“Thief”。
9. 第 32 行和第 33 行是设置 PORTA 专供输入使用。
10. 第 36 行和第 37 行检查是否有键盘输入，如果有则离开循环，在本例中相当于程序运行结束。
11. 第 40 行至第 43 行检查 alarm 是否有打开，如果没有打开，则回到循环起始位置 (第 24 行)，如果打开则往下运行。
12. 第 48 行至第 51 行是设置以反白显示“Safe”。
13. 第 52 行至第 55 行是设置以正常显示“Thief”。
14. 第 58 行至第 61 行是检查是否有小偷入侵，如果有则调用 thief\_is\_coming( ) 函数。
15. 第 63 行至第 68 行是检查 alarm 是否打开，如果是则持续第 46 行至 69 行间的循环，否则退出此循环。
16. 第 46 行至第 69 行是一个循环，如果 alarm 是打开则此循环会一直持续。
17. 第 21 行至第 70 行是一个循环。只要没有键盘输入此循环将持续运行。

函数 thief\_is\_coming( ) 解释:

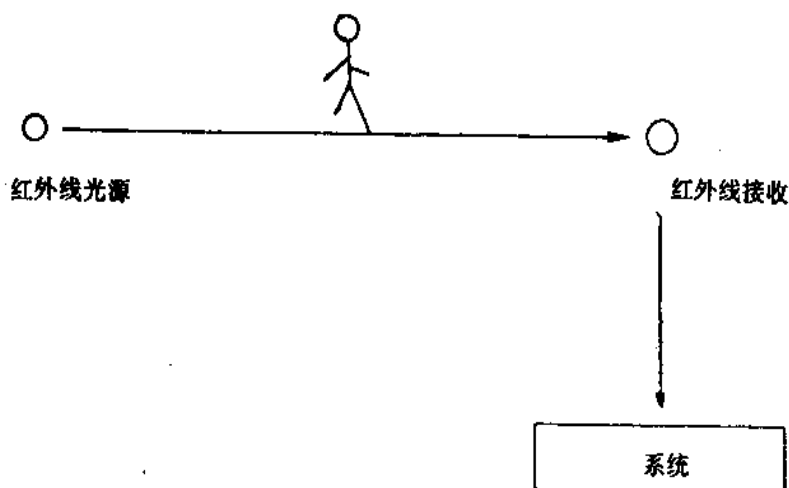
1. 第 82 行至第 85 行是设置在 (60,4) 位置，以反白显示“Thief”字符串。
2. 第 86 行和第 89 行是设置在 (18,4) 位置，以正常方式显示字符串“Safe”。
3. 第 92 行和第 93 行是设置 PORTA 口供输出使用。
4. 第 96 行和第 100 行是令 PA1 至 PA7 的灯不断闪动。
5. 第 101 行是调用 alarmsound( ) 函数发出警报器声音。
6. 第 104 行至第 105 行是设置 PORTA 口专供输入使用。
7. 第 108 至第 113 行是检查 alarm 是否小偷入侵开关移回，如果没有则第 80 行至第 114 行的循环继续。
8. 第 80 行至第 114 行是一个循环，只要小偷入侵开关是 ON，则此循环将继续。

函数 alarmsound( ) 解释

1. 第 123 行至第 127 行是一个循环，此循环将产生频率从 0 至 1000 之间的声音，其中每一声延迟 0.001 秒。
2. 第 128 行是关闭声音。

## 15.4 结 论

这是一个最简单的防盗装置，给读者一个基本概念。现今市面上的防盗系统多为红外线装置。如下所示:



当小偷入侵时，造成红外线接收器收不到红外线，电流呈现中断现象，此时可以启动系统，通报小偷入侵。

### 15.5 思考题

1. 如果小偷知道 PA1 开关在哪，应如何设计程序防范？
2. 如何判定防盗器是小猫误触或真是小偷？软件如何配合？
3. 汽车防盗器的最主要元件是可以感受震动的开关及电子元件，请问它是否由微型计算机控制？微型计算机在此处扮演何种角色。

问题提示：

1. 可设计一个读取系统时间的子程序，在晚上20:00—08:00期间让PA1开关失效。纵使小偷知道开关在哪，也影响不了系统。如何读取系统时间请参考附录C。
2. 可依踩下的时间长短而定，小猫动作较敏捷，暂定在0.5秒之内松开PA0的定为小猫，否则为小偷。

16.1 工作目标

## 第十六章 数字 IC 测试器专题

### 本实验目的

1. 通过此实验，读者可以替自己设计一个实用的数字 IC 测试器。
2. 通过此实验，读者可以了解

■数字 IC 的 IC 工作方式

■PC 接口 I/O 控制

### 本章内容

16.1 工作目标

16.2 硬件设计

16.3 软件设计

16.4 结论

16.5 思考题

### 16.1 工作目标

在电子电路实验中，我们常用一些逻辑 IC，本章将详细说明逻辑 IC 的测试方式，我们选用的 IC 为 7400，7408，7432，7468，选用这些 IC 的目的是因为它们都为 14 脚，而且输入/输出的功能相同，易于让初学者了解。

### 16.2 硬件设计

PORTA 定义为输入，PORTB 定义为输出。微处理器根据这些 PORTB 输出的信号辨认 IC 的功能。

表 16.1 7400,7408,7432,7486 的测试规格

测试输入 TEST PATTERN PORTA							
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1

不同 IC 测试输出结果							
7400 PORT B				7408 PORT B			
PB3	PB2	PB1	PB0	PB3	PB2	PB1	PB0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1

不同 IC 测试输出结果							
7432 PORT B				7486 PORT B			
PB3	PB2	PB1	PB0	PB3	PB2	PB1	PB0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0

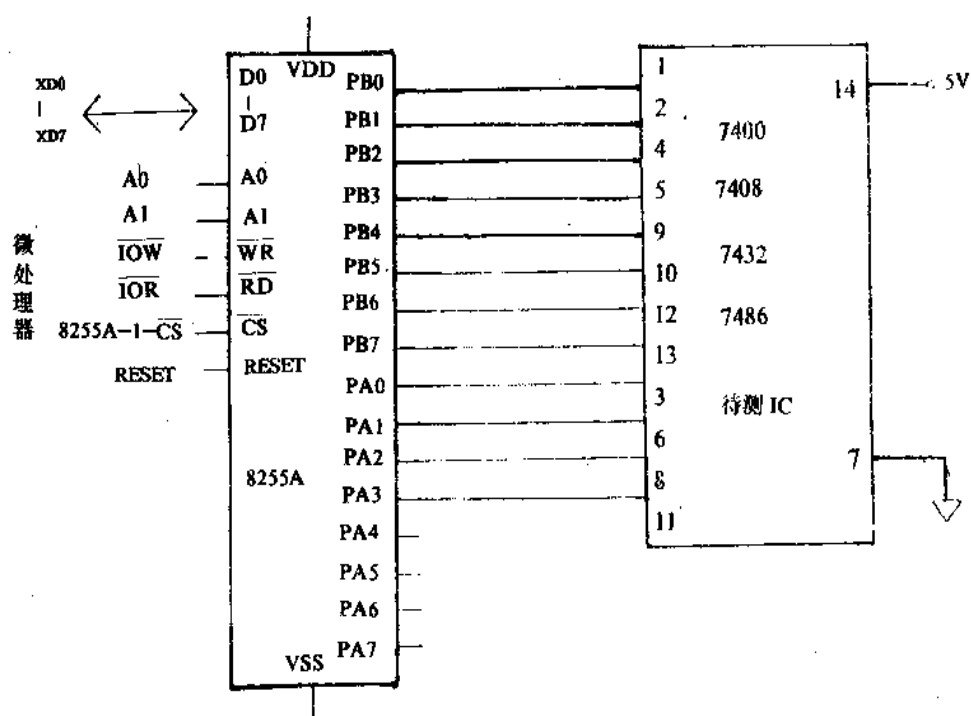
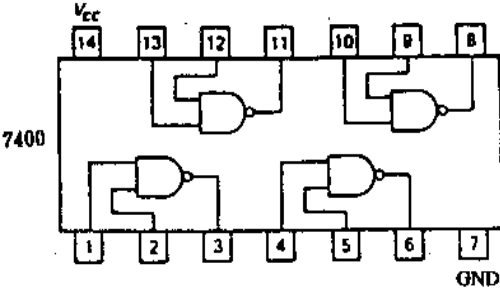


图 16.1 本章实验所用硬件

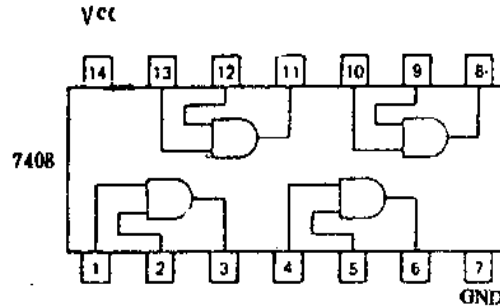
表 16.1 为 IC 测试器的输出 (由 PORTB) 及输入 (由 PORTA) 表。微型处理机经由 8255A 输入数据给待测 IC, 这些待测 IC 将结果传给 8255A PORTB, 微型处理机据以检查 IC 的功能。

图 16.2 为 7400,7408,7432,7486 的引脚图与真值表。

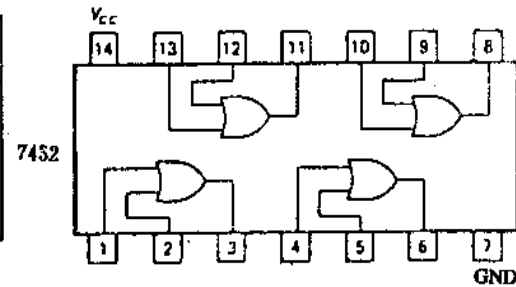
A (输入)	B (输入)	Y (输出)
0	0	1
0	1	1
1	0	1
1	1	0



A (输入)	B (输入)	Y (输出)
0	0	0
0	1	0
1	0	0
1	1	1



A (输入)	B (输入)	Y (输出)
0	0	0
0	1	1
1	0	1
1	1	1



A (输入)	B (输入)	Y (输出)
0	0	0
0	1	1
1	0	1
1	1	0

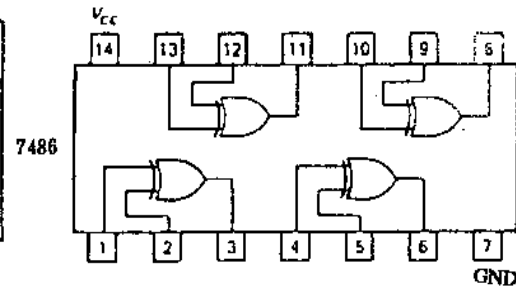


图 16.2 7400,7408,7432,7486 的引脚图与真值表

## 16.3 软件设计

程序示例 ictest.c

数字 IC 测试程序，本程序主要是用于测试所插入的 IC 是否为 7400,7408,7432 或 7486，如果是则显示出来，否则显示“This is not a good IC.”。

本程序在运行时屏幕将如下所示：

```
Digital IC Testing

Insert the IC right now and press any key
```

当你将 IC 插入后，如果 IC 是 7400，则屏幕将如下所示：

```
Digital IC Testing

Insert the IC right now and press any key
This is IC 7400

Press any to exit the program
```

当你将 IC 插入后，如果 IC 是 7408 则屏幕将如下所示：

```
Digital IC Testing

Insert the IC right now and press any key
This is IC 7408.

press any to exit the program.
```

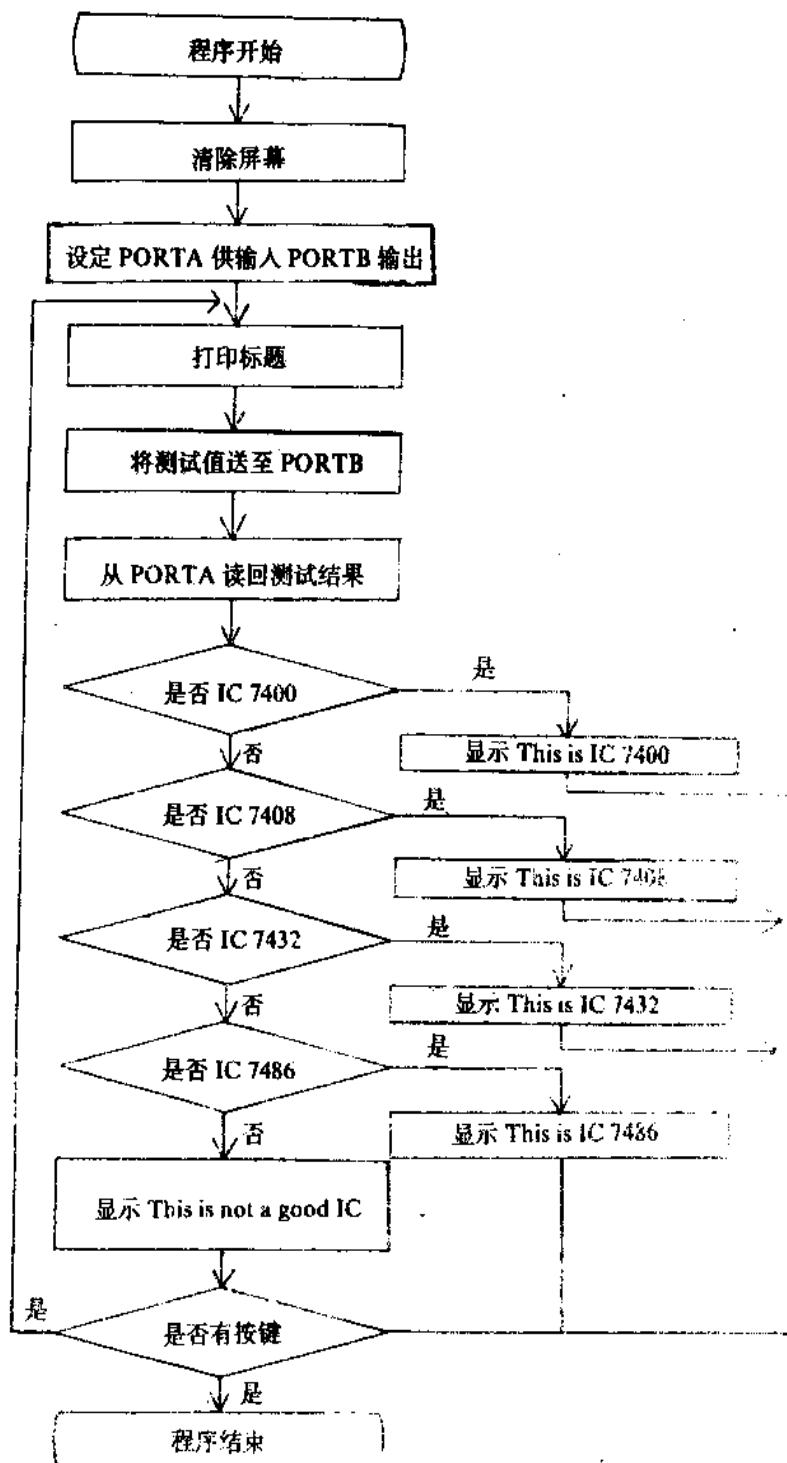
如果是其它 IC 或是有损坏，则屏幕将如下所示：

```
Digital IC Testing

Insert the IC right now and press any key
This is not a good IC

Press any to exit the program
```

当你见到“Press any exit the program”字符串时，若按任何一个键则程序结束，否则屏幕又将要求你插入一个 IC 做测试。





```

01  /* ----- */
02  /*      Program Name : ictest.c      */
03  /*      Digital IC testing application.      */
04  /*      For 8255A, Mode 0, PORTA input and PORTB output      */
05  /* ----- */
06  #include <dos.h>
07  #include <conio.h>
08  #define SETPORT    0x3e3
09  #define PORTA      0x3e0
10  #define PORTB      0x3e1
11  #define IC7400      0x07
12  #define IC7408      0x08
13  #define IC7432      0x0e
14  #define IC7486      0x06
15  #define INDATA      0xe4;    /* for testing IC input data */
16  int result;
17  void main( )
18  {
19      void get__porta( );
20      void output__portb( );
21      void pttitle( );
22      unsigned char bytewrite, byteread;
23
24      clrscr( );          /* clear the screen */
25      /* set up the output port */
26      bytewrite = 0x90;
27      outportb(SETPORT,bytewrite);
28
29      while ( !kbhit( ) )
30      {
31          pttitle( );      /* print the program title */
32          getch( );
33          output__portb( ); /* send the testing data to PORTB */
34          get__porta( );    /* get the result from PORTA */
35          gotoxy(33,6);
36          switch ( result )
37          {
38              case IC7400 : printf("This is IC 7400.");
39                          break;
40              case IC7408 : printf("This is IC 7408.");
41                          break;
42              case IC7432 : printf("This is IC 7432.");
43                          break;
44              case IC7486 : printf("This is IC 7486.");

```

```

45             break;
46     default   : printf("This is not a good IC.");
47     }
48     gotoxy(26,10);
49     printf("Press any to exit the program.");
50     sleep(5);
51     clrscr( );
52 }
53 }
54 / * ----- */
55 / *  Get the result from PORTA                */
56 / * ----- */
57 void get__porta( )
58 {
59     unsigned char byteread;
60
61     byteread = inportb(PORTA);
62     result = 0x0f & byteread;
63 }
64 / * ----- */
65 / *  Send the testing data to PORTB            */
66 / * ----- */
67 void output__portb( )
68 {
69     unsigned char bytewrite;
70
71     bytewrite = INDATA;
72     outportb(PORTB,bytewrite);
73 }
74 / * ----- */
75 / *  Print the program title                    */
76 / * ----- */
77 void ptitle( )
78 {
79     gotoxy(32,1);
80     printf("Digital IC Testing");
81     gotoxy(20,4);
82     printf("Insert the IC right now and press any key");
83 }

```

程序示例 ictest.c 解释:

1. 第 8 行是用于设置 SETTPORT 为控制寄存器的口, 值是 0x3e3.
2. 第 9 行是设置 PORTA 口, 值是 0x3e0.
3. 第 10 行是设置 PORTB 口, 值是 0x3e1.

4. 第 11 行是定义 IC 7400 值为 0x07.
  5. 第 12 行是定义 IC 7408 值为 0x08.
  6. 第 13 行是定义 IC 7432 值为 0x0e.
  7. 第 14 行是定义 IC 7486 值为 0x06.
  8. 第 15 行是定义 INDATA 值为 0xc4.
  9. 第 24 行是清除屏幕内容.
  10. 第 26 行至第 27 行是用于设置 PORTB 专供输出, PORTA 专供输入使用.
  11. 第 31 行是调用 ptitle( )函数显示程序标题.
  12. 第 32 行是调用 getch( )函数, 主要是在上一行显示完程序标题后令程序暂停, 直至按下下一个键再继续运行.
  13. 第 33 行是调用 output\_portb( )函数, 将测试值 INDATA 送至 PORTB.
  14. 第 34 行是调用 get\_porta( )函数, 将测试结果值读回.
  15. 第 35 行是将光标移至 (36,6) 位置.
  16. 第 36 行至第 47 行是按第 34 行所读回的测试结果值, 比较及显示测试结果.
  17. 第 48 行是将光标移至 (26,10) 位置.
  18. 第 49 行是显示字符串 "Press any key to exit the program".
  19. 第 50 行是令程序暂停 5 秒.
  20. 第 51 行是清除屏幕.
  21. 第 29 行至第 51 行是一个循环, 如果键盘有输入则离开此循环.
- 函数 get\_porta( )解释:
1. 第 61 行是从 PORTA 读回值.
  2. 第 62 行是处理所读回值.
- 函数 putput\_portb( )解释:
1. 第 71 行是设置要输出至 PORTB 的值.
  2. 第 72 行是将测试值送至 PORTB.
- 函数 ptitle( )解释:
1. 第 79 行是将光标移至 (32,1) 位置.
  2. 第 80 行是显示字符串 "Digital IC Testing".
  3. 第 81 行是将光标移至 (20,4) 位置.
  4. 第 82 行是显示字符串 "Insert the IC right now and press any key".

## 16.4 结 论

市面上的 IC 测试器即是利用这种工作原理将所有的测试数据 (Pattern) 存入存储器中, 再读回待测 IC 的数据互相对比, 如果相同则表示 IC 是好的, 如果不同则是坏的.

这个实验中挑选 7400, 7408, 7432, 7486 的目的在于, 这四个的引脚相同, 可以省去用继电器跳引脚的麻烦. 一般市面上的 IC 测试器则具有这个功能.

## 16.5 思考题

1. 仔细研究本章实验，并加以改进。

[注]: 本章的实验数据并不完全。

2. 设计一个可以测7410, 7411, 7427的实验电路，并将结果在屏幕上显示出来，图16.3为这些IC的内部结构。

3. 找一个自己最常用的IC，设计一个IC Checker (IC测试器) 为自己服务。

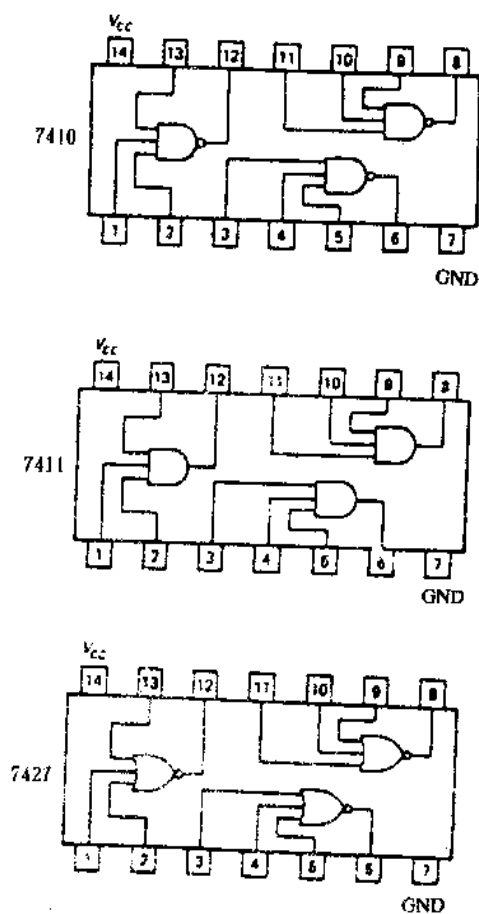


图 16.3 7410, 7411, 7427 的引脚图

## 第十七章 电梯模拟专题

### 本实验目的

1. 通过 8255A 电梯的模拟以了解:

■ IBM PC I/O 的接口控制应用

■ 随机数选择

■ 电梯的基本工作概念

### 本章内容

17.1 工作目标

17.2 硬件设计

17.3 软件设计

17.4 结论

17.5 思考题

### 17.1 工作目标

1. 用户进入电梯后, 按下 SW4 使  $PB0=1$ , 由随机数选择现在所在的楼层。
2. 用户按下所要前往的楼层 (由 SW1~SW3 选择)。
3. 按键 SW4 设置  $PB0=0$ , 代表电梯关门开始服务。

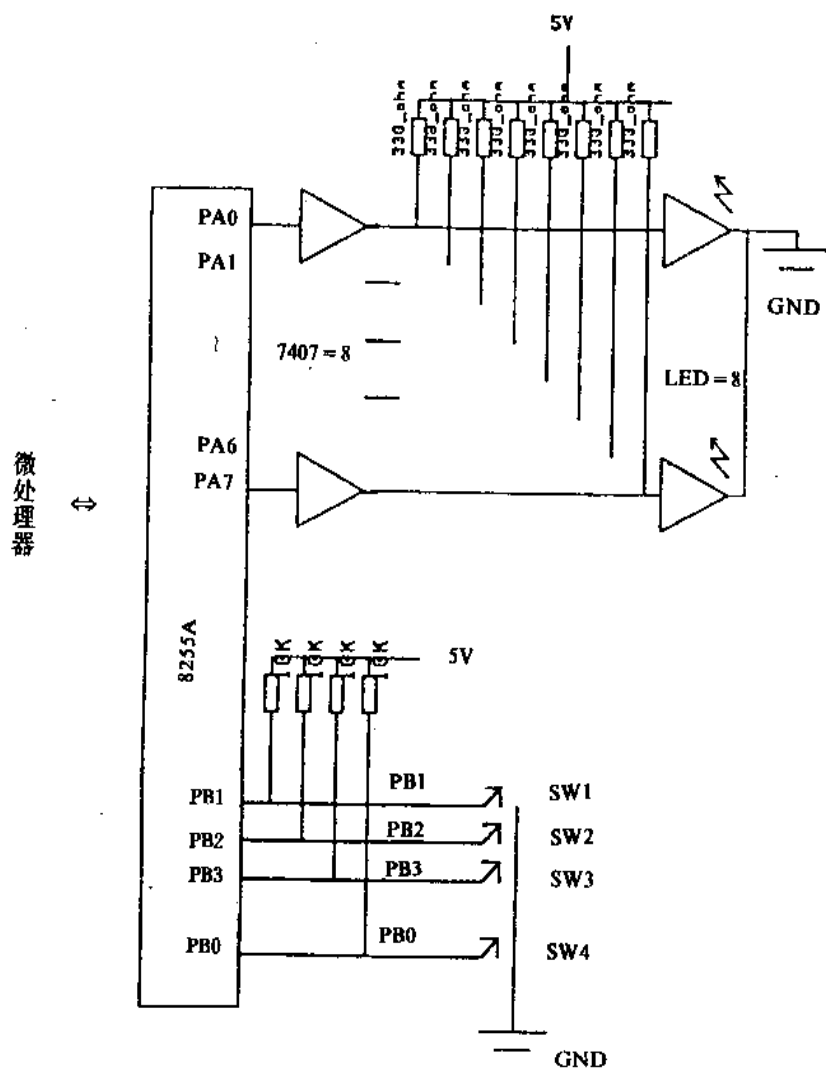
### 17.2 硬件设计

硬件设计见下页图所示。

由  $PB1, PB2, PB3$  选择所要前往的楼层如表 17.1。

表 17.1 楼层选择

楼层	1	2	3	4	5	6	7	8
PB1	1	0	1	0	1	0	1	0
PB2	0	1	1	0	0	1	1	0
PB3	0	0	0	1	1	1	0	0



### 17.3 软件设计

#### 示例程序 cleva.c

电梯的程序设计。本程序在运行时首先会检查 PB0 是否为 1，如果为 1，表示开始启动电梯，此时可以利用 PB3，PB2 和 PB1 决定电梯最后停留的楼层编号。例如，如果此三个位值如下所示：

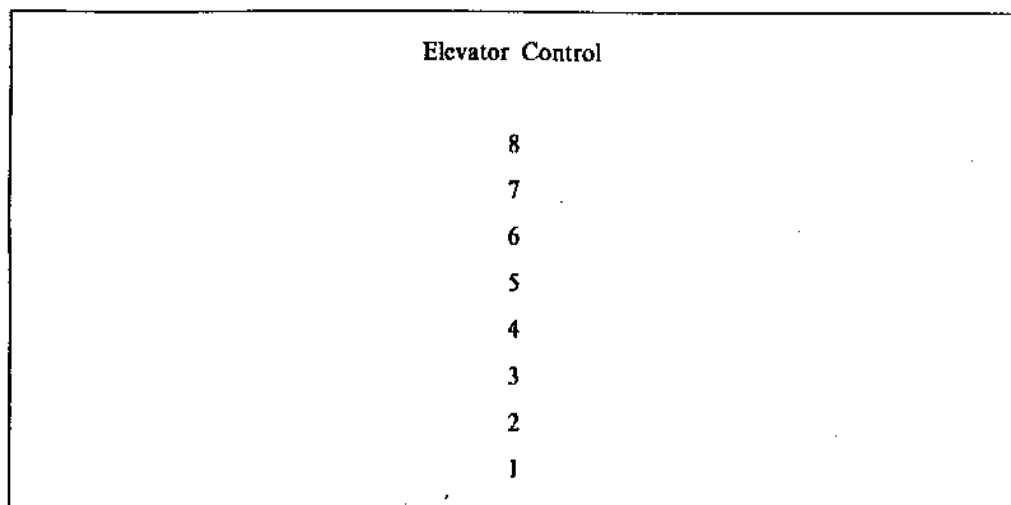
PB3	PB2	PB1
1	1	0

表示最后电梯的楼层编号是 6，但是如果此三个位是 000，则代表最后电梯的楼层编号是 8。

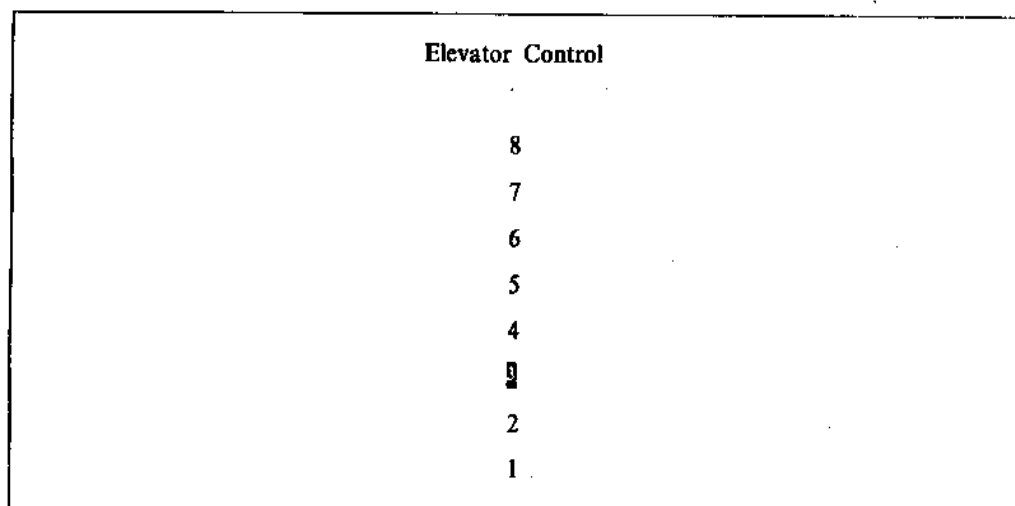
有了电梯最初楼层编号及最后楼层编号之后，下一步是再读取 PB0 的值，如果 PB0 的值是 0 表示可开始移动电梯。移动完电梯之后，如果随意按一个键则程序结束，否则重

新运行电梯。

本程序在运行之初屏幕内容将如下所示:



当 PB0 等于 1 时, 程序会产生最初楼层编号, 假设是 3 楼, 则屏幕内容如下所示:



此时 PORTA 的 LED 内的 PA2 灯也将发亮, 如下所示:

○ ○ ○ ○ ○ ● ○ ○ ○

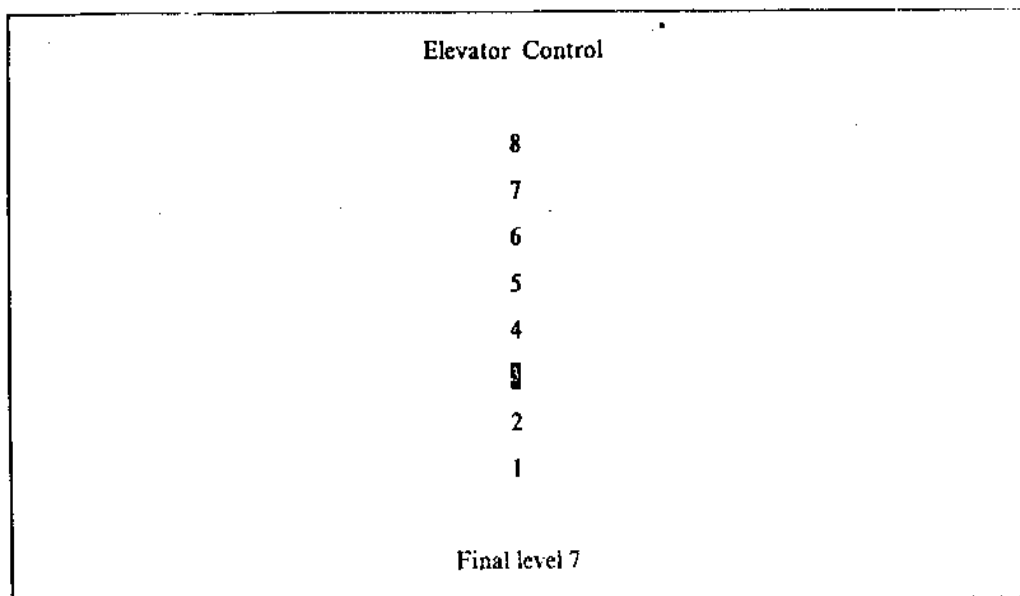


PA2 发亮

现在你可以利用 PB3, PB2 和 PB1 决定电梯移动的最后楼层数。如果 PB3, PB2 和 PB1 值如下所示:

0 1 1

则屏幕内容应如下所示:



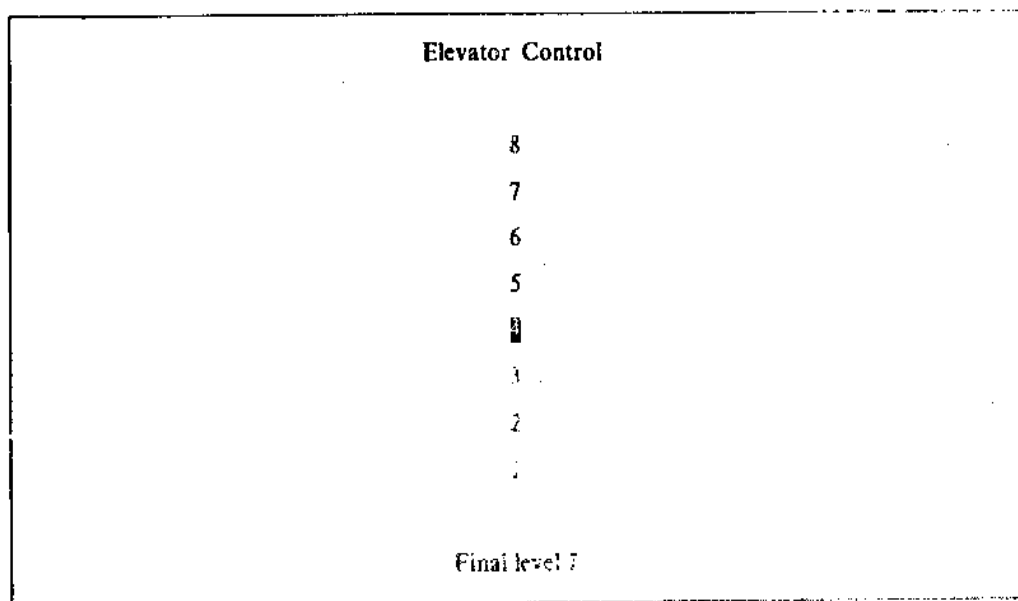
然后程序以每隔一秒方式移动升降机从目前楼号移至最后楼号，例如本示例，首先 PORTA 的 LED 将如下所示：

○ ○ ○ ○ ● ○ ○ ○



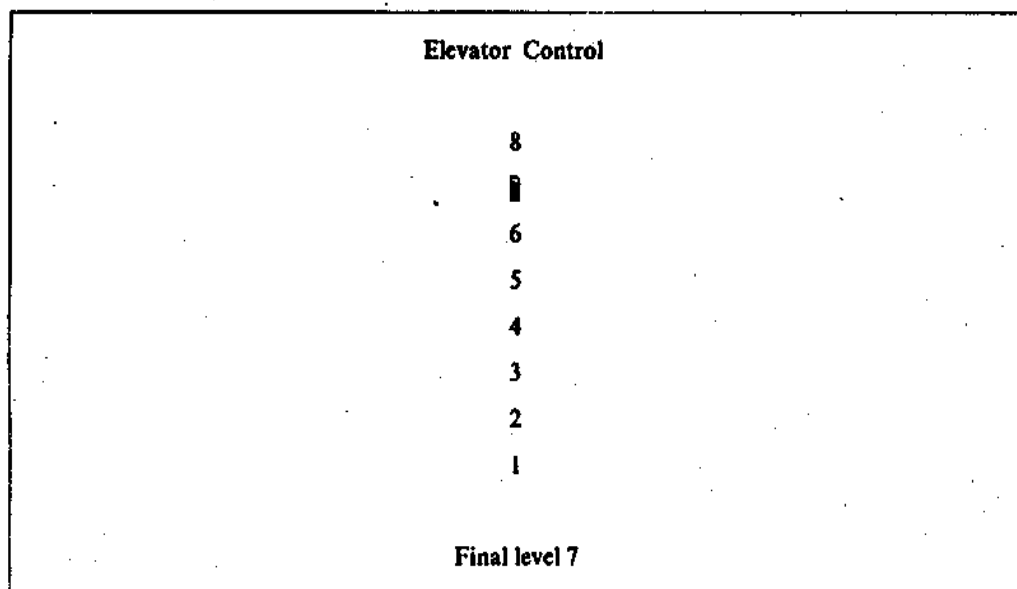
PA3 灯亮

而屏幕将如下所示：



升降机将一直移动直至屏幕如下所示：





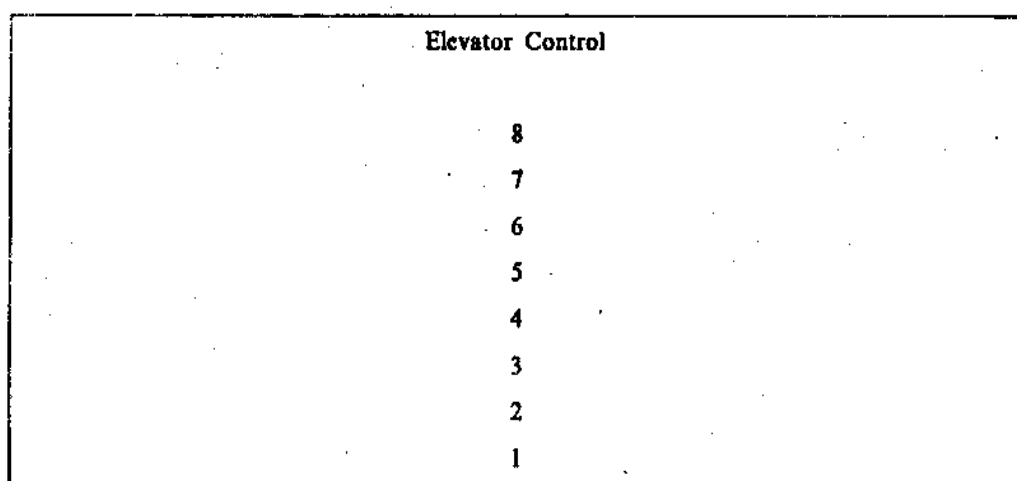
而 PORTA 的 LED 如下所示:

○●○○○○○○○

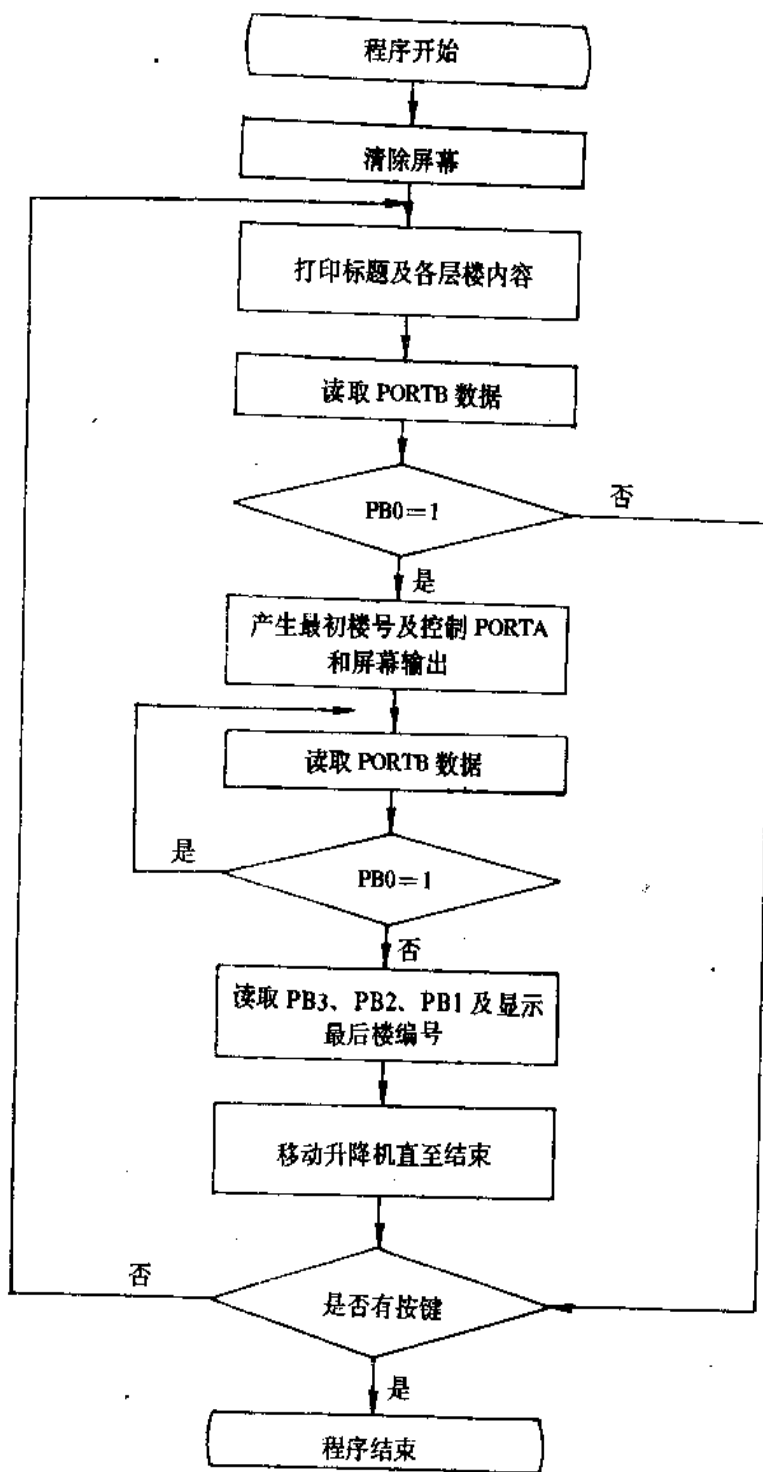


PA6 灯亮

此时升降梯才算完成一个回合，如果此时键盘有按键则程序结束，否则又是一个新回合的开始，屏幕将如下所示:



本程序的流程如下所示:



01 /\* ----- \*/  
 02 /\* Program Name : eleva.c \*/  
 03 /\* Elevator control with Screen Function. \*/

```

04  /* For 8255A, Mode 0, PORTA output and PORTB input      */
05  /* ----- */
06  #include <dos.h>
07  #include <stdlib.h>
08  #include <math.h>
09  #include <conio.h>
10  #define SETPORT    0x3e3
11  #define PORTA      0x3e0
12  #define PORTB      0x3e1
13  int    inieleva, finaleleva;
14  void main( )
15  {
16      void ptitle( );
17      void plevel( );
18      void randomelevator( );
19      void trans( );
20      void getlevel( );
21      void move__elevator( );
22      void printattr( );
23      unsigned char bytewrite, bytread, bytetest;
24
25      window(1,1,80,25);    /* set up the screen as a window */
26
27      clrscr( );
28      /* set up the PORTA output and PORTB input */
29      bytewrite = 0x82;
30      outportb(SETPORT,bytewrite);
31
32      /* do the elevator control */
33      while ( !kbhit( ) )
34      {
35          ptitle( );        /* print the program title */
36          plevel( );        /* print the 8 level elevator */
37
38          /* read PORTB control message */
39          bytread = inportb(PORTB);
40          bytetest = 0x01 & bytread;    /* check PB0 = 1? */
41          if ( bytetest == 0x01 )
42          {
43              randomelevator( );
44              trans( );
45              printattr(inieleva);    /* print the elevator level */
46              getlevel( );            /* get the final level */
47              move__elevator( );      /* move the elevator */

```

```

48     sound(500);
49     sleep(1);
50     nosound( );
51     sleep(2);
52     clrscr( );
53 }
54 }
55 }
56 /* ----- */
57 /* Move the elevator */
58 /* ----- */
59 void move_elevator( )
60 {
61     void plevel( );
62     void printattr( );
63     int i, bytewrite;
64
65     if ( finaleleva > inieleva )
66         for ( i = inieleva + 1; i <= finaleleva; i++ )
67         {
68             plevel( );
69             printattr(i);
70             bytewrite = (int) pow(2,(float)(i - 1));
71             outportb(PORTA,bytewrite);
72             sleep(2);
73         }
74     else
75         for ( i = inieleva - 1; i >= finaleleva; i-- )
76         {
77             plevel( );
78             printattr(i);
79             bytewrite = (int) pow(2,(float)(i - 1));
80             outportb(PORTA,bytewrite);
81             sleep(2);
82         }
83 }
84 /* ----- */
85 /* read PORTB to get the final elevator level */
86 /* ----- */
87 void getlevel( )
88 {
89     unsigned char bytetest;
90
91     sleep(2);

```

```

92  /* check PB0, if PB0 = 0, then read final level */
93      while ( 1 )
94      {
95          bytread = inportb(PORTB);
96          bytetest = bytread & 0x01;
97          if ( bytetest == 0x00 )
98              break;
99      }
100
101  /* check PB1, PB2, PB3 */
102      bytread = inportb(PORTB);
103      finaleleva = bytread & 0x0e;
104      finaleleva = finaleleva >> 1;
105      if ( finaleleva == 0 )          /* 000 mean level 8 */
106          finaleleva = 8;
107
108  /* print the final level on the screen */
109      gotoxy(33,22);
110      cprintf("Final level %2d",finaleleva);
111  }
112  /* ----- */
113  /* Print the elevator level with reverse attribute */
114  /* ----- */
115  void printattr(int index)
116  {
117      textcolor(BLACK);
118      textbackground(WHITE);
119      switch ( index )
120      {
121          case 1 : gotoxy(40,18);
122                  cprintf("1");
123                  break;
124          case 2 : gotoxy(40,16);
125                  cprintf("2");
126                  break;
127          case 3 : gotoxy(40,14);
128                  cprintf("3");
129                  break;
130          case 4 : gotoxy(40,12);
131                  cprintf("4");
132                  break;
133          case 5 : gotoxy(40,10);
134                  cprintf("5");
135                  break;

```

```

136     case 6 : gotoxy(40,8);
137         cprintf("6");
138         break;
139     case 7 : gotoxy(40,6);
140         cprintf("7");
141         break;
142     case 8 : gotoxy(40,4);
143         cprintf("8");
144         break;
145 }
146 textcolor(WHITE);
147 textbackground(BLACK);
148 }
149 / * ----- */
150 / * Transfer the level data to PORTA */
151 / * ----- */
152 void trans( )
153 {
154     unsigned char bytewrite;
155
156     bytewrite = (int) pow(2,(float)(inieleva - 1));
157     outportb(PORTA,bytewrite);
158 }
159 / * ----- */
160 / * Using the Random number to generate the initial */
161 / * elevator location. */
162 / * Because random(8) will generate 0 — 7, so let */
163 / * it added by 1, then will generate 1 — 8 */
164 / * ----- */
165 void randomelevator( )
166 {
167     inieleva = random(8) + 1;
168 }
169 / * ----- */
170 / * Print 8 level elevators */
171 / * ----- */
172 void plevel( )
173 {
174     gotoxy(40,4);
175     cprintf("8");
176     gotoxy(40,6);
177     cprintf("7");
178     gotoxy(40,8);
179     cprintf("6");

```

```

180     gotoxy(40,10);
181     cprintf("5");
182     gotoxy(40,12);
183     cprintf("4");
184     gotoxy(40,14);
185     cprintf("3");
186     gotoxy(40,16);
187     cprintf("2");
188     gotoxy(40,18);
189     cprintf("1");
190 }
191 / * ----- */
192 / *   Print the program title                       */
193 / * ----- */
194 void ptitle( )
195 {
196     gotoxy(32,1);
197     cprintf("Elevator Control");
198 }

```

示例程序 eleva.c 解释:

1. 第 10 行是用于设置 SETPORT 为控制寄存器的口值是 0x3e3.
2. 第 11 行是设置 PORTA 口值是 0x3e0.
3. 第 12 行是设置 PORTB 口值是 0x3e1.
4. 第 25 行是设置屏幕为一个 Window, 这样我们便可以使用 cprintf( ) 函数, 达到以反白显示某些字符串的方式.
5. 第 27 行是清除屏幕内容.
6. 第 29 行至第 30 行是用于设置 PORTA 专供输出, PORTB 专供输入使用.
7. 第 35 行是调用 ptitle( ) 函数显示屏幕标题.
8. 第 36 行是调用 plevel( ) 函数显示升降梯的楼层编号.
9. 第 39 行是读取 PORTB 值.
10. 第 40 行是分析 PB0 值.
11. 第 41 行是检查 PB0 是否是 1, 如果是 1 代表已经启动升降梯, 则运行第 43 行, 否则跳回第 33 行.
12. 第 43 行是调用 radomelevator( ) 函数产生最初升降梯位置.
13. 第 44 行是调用 trans( ) 函数将最初升降梯位置转换成适当数据送至 PORTA.
14. 第 45 行是调用 printattr( ) 函数将升降梯所在楼层以反白显示其楼层编号.
15. 第 46 行是调用 getlevel( ) 函数, 读取 PORTB 值以便了解升降梯移动的目的楼层编号.
16. 第 47 行是调用 move\_elevator( ) 函数, 实际移动升降梯.
17. 第 48 行是产生频率为 500 的声音.

18.第 49 行是令程序暂停 1 秒。

19.第 50 行是关闭声音。

20.第 51 行是令程序暂停 2 秒。

21.第 52 行是清除屏幕。

22.第 33 行至第 54 行的循环是检查有没有键盘输入，如果有则程序结束。

函数 `move_elevator()` 解释：

1. 第65行是比较最后楼层编号是否大于最初楼层编号，如果是则电梯往上移，此时要运行第 66 行至第 73 行。否则电梯往下移，此时要运行第 75 行至第 82 行。

2. 第66行至第73行，分别以反白显示电梯所移位置（往上），同时每移一层楼暂停2秒。

3. 第75行至第82行，分别以反白显示电梯所移位置（往下），同时每移一层楼暂停2秒。

函数 `getlevel()` 解释

1. 第 91 行是令程序暂停 2 秒。

2. 第 93 行至第 99 行是检查 `PB0` 的值，直至 `PB0=0` 才退出此循环。

3. 第102行至第106行是读取 `PORTB` 内 `PB1`、`PB2` 和 `PB3` 的值，以确定电梯移动最终位置。

4. 第 109 行是将光标移至 (32,22) 位置。

5. 第 110 行是显示数据 `Final level` 及最后楼层编号。

函数 `printattr()` 解释：

1. 第117行至第145行主要是用于以反白显示 `index` 变量（也就是电梯的楼层编号）。

2. 第146行至147行是设置未来在屏幕显示数据时以正常方式显示。

函数 `trans()` 解释：

1. 第156行是调用 `pow()` 函数（可参考附录C）将楼层编号转换成要送至 `PORTA` 的值。

2. 第 157 行是将值送至 `PORTA`。

函数 `randomelevator()` 解释：

1. 第167行是产生最初电梯位置，由于 `random(8)` 将产生 0~7 之间的值，而我们要产生 1~8 楼，所以将它的值加 1。

函数 `plevel()` 解释：

1. 第 174 行会将光标移至 (40,4) 位置。

2. 第 175 显示字符串“8”。

3. 第 176 行会将光标移至 (40,6) 位置。

4. 第 177 行是显示字符串“7”。

5. 第 178 行会将光标移至 (40,8) 位置。

6. 第 179 行是显示字符串“6”。

7. 第 180 行会将光标移至 (40,10) 位置。

8. 第 181 行是显示字符串“5”。

9. 第 182 行会将光标移至 (40,12) 位置。



- 10.第 183 行是显示字符串“4”。
- 11.第 184 行会将光标移至 (40,14) 位置。
- 12.第 185 行是显示字符串“3”。
- 13.第 186 行会将光标移至 (40,16) 位置。
- 14.第 187 行是显示字符串“2”。
- 15.第 188 行会将光标移至 (40,18) 位置。
- 16.第 189 行是显示字符串“1”。

函数 ptitle( )解释:

1. 第 196 行是将光标移至 (32,1) 位置。
2. 第 197 行是显示字符串“Elevator Control”。

## 17.4 结 论

通过 PC 控制的电梯还可以具备非常多的功能,譬如:停电、呼叫,请读者思考下列问题。

## 17.5 思考题

若发生火灾时最危险的逃生方式就是搭乘电梯,因为如果搭乘期间电力系统中断,升降梯会被卡住,造成人员伤亡。请读者更改程序,加入另一个控制元件当作火警信号,当发生火灾时用以阻止升降梯继续操作,以防不测。

## 第十八章 加油机专题

### 本实验目的

1. 通过 8255A 的应用了解系统处理屏幕以及简易的运算。
2. 通过此实验专题可以了解
  - 接口 I/O 控制
  - 加油机的基本工作概念
  - 更进一步的 C 语言软件设计技巧

### 本章内容

- 18.1 工作目标
- 18.2 硬件设计
- 18.3 软件设计
- 18.4 结论
- 18.5 思考题

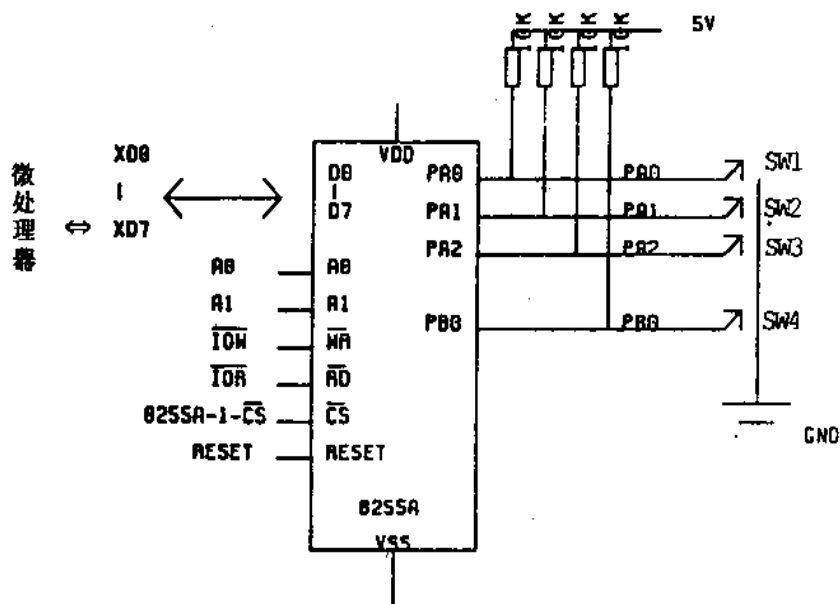
### 18.1 工作目标

本实验专题是以微型处理机模拟加油机的加油操作。工作方式如下:

1. 油品种类的选择, 如 SW1~SW3 选择。
2. 加油枪由SW4选择, 此时屏幕将出现加油种类、加油费用及加油公升数。

### 18.2 硬件设计

本实验硬件线路设计如下图所示:



SW1: A 类油品  
 SW2: B 类油品  
 SW3: C 类油品  
 SW4: 加油枪

### 18.3 软件设计

#### 程序示例 oil.c

加油机程序设计。本程序在运行时首先屏幕将如下所示:

Petroleum application			
	Petroleum A	Petroleum B	Petroleum C
Unit Price	20	18	16
Total Money: 000			
Total Liter: 000			

此程序尝试去读取读者所要加油的种类。其规则如下所示:

PA0=1, 表示加 Petroleum A

PA1=1, 表示加 Petroleum B

PA2=1, 表示加 Petroleum C

其中 Petroleum A 油单价是 20 元, Petroleum B 油单价是 18 元, Petroleum C 油单

价是 16 元。当你选择某一项之后，该项目会以反白方式显示，例如若是你让 PA1=1，则屏幕将如下图所示：

Petroleum application			
	Peltroleum A	Petroleum B	Petroleum C
Unit Price	20	18	16
Total Money: 000			
Total Liter: 000			

然后程序会尝试去读取 PB0 的值，如果此值是 1 表示开始加油，此时屏幕会列出加油的数量及金钱数。直至 PB0 等于 0，表示加油过程结束。下图是加油结束后屏幕的显示内容。

Petroleum application			
	Peltroleum A	Petroleum B	Petroleum C
Unit Price	20	18	16
Total Money: 216			
Total Liter: 12			

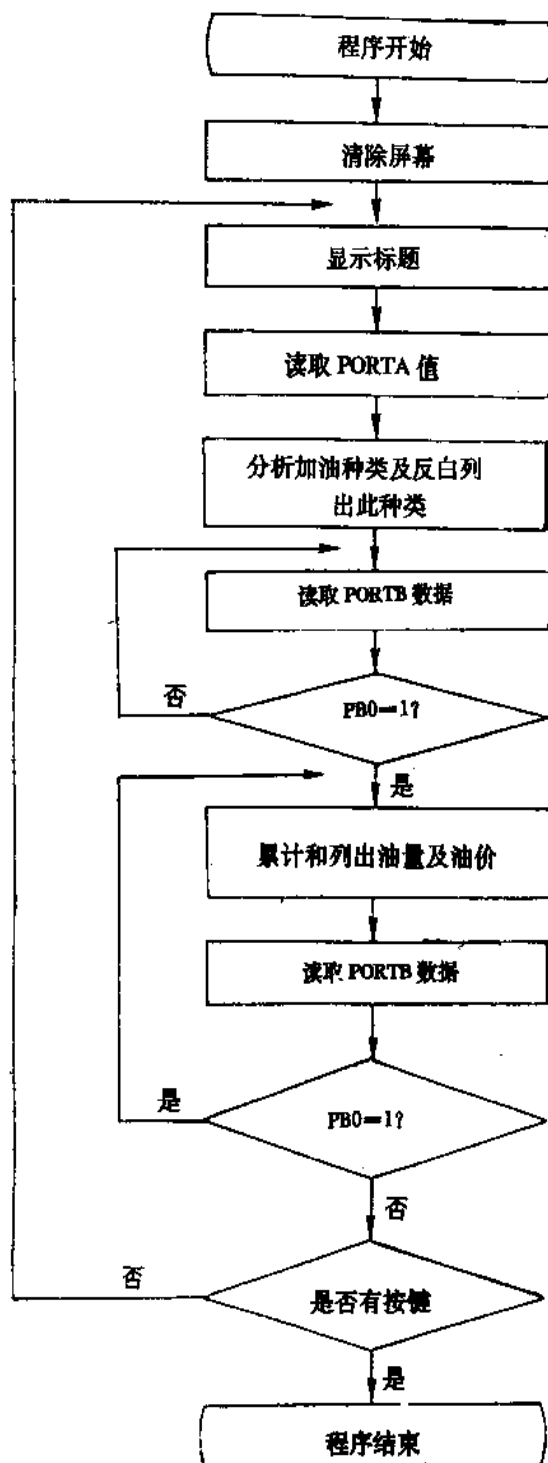
过了 5 秒，屏幕将如下所示：

Press any key to exit the program.
------------------------------------

此时你如果随意按一个键，程序将立刻结束且返回 DOS。否则程序又将返回加油状态，如下图所示：

Petroleum application			
	Peltroleum A	Petroleum B	Petroleum C
Unit Price	20	18	16
Total Money: 000			
Total Liter: 000			

本程序的工作流程如下所示:



```

01 / * ----- */
02 / *      Program Name : oil.c      */
03 / *      Petroleum application.    */
04 / *      For 8255A, Mode 0 application. */
05 / * ----- */
06 #include <dos.h>
07 #include <conio.h>
08 #define  SETPORT    0x3e3
09 #define  PORTA      0x3e0
10 #define  PORTB      0x3e1
11 int      uprice;      / * define UNIT price * /
12 int      tprice;      / * define total price * /
13 int      tliter;      / * define total liter * /
14 void main( )
15 {
16     void addpetroleum( );
17     void getpetroleum( );
18     void printtitle( );
19     unsigned int bytewrite;
20
21     window(1,1,80,25); / * set screen as a window * /
22     clrscr( );        / * clear the screen * /
23
24     / * set up the PORTA and PORTB as input mode * /
25     bytewrite = 0x92;
26     outportb(SETPORT,bytewrite);
27
28     / * do the job * /
29     while ( !kbhit( ) )
30     {
31         printtitle( );    / * print the program title * /
32         getpetroleum( );  / * get the Petroleum Type * /
33         addpetroleum( );  / * start adding the Petroluem * /
34         sleep(5);         / * delay 5 seconds * /
35         clrscr( );        / * clear the screen * /
36         gotoxy(23,10);
37         cprintf("Press any key to exit the program.");
38         sleep(2);         / * delay 2 seconds * /
39         clrscr( );        / * clear the screen * /
40     }
41 }
42 / * ----- */
43 / *      Add the Petroleum and calculate the Total Price      */
44 / * ----- */

```

```

45 void addpetroleum( )
46 {
47     unsigned char byteread;
48
49     tprice = 0;
50     tliter = 0;
51
52     /* check if PB0 = 1, mean adding the Petroleum */
53     byteread = inportb(PORTB);
54     while ( ( ( byteread = inportb(PORTB) ) & 0x01 ) == 0 )
55         ;
56
57     /* add the petroleum */
58     while ( 1 )
59     {
60         sleep(1); /* delay 1 second as add 1 liter */
61         tliter++;
62         tprice = uprice * tliter;
63         gotoxy(46,8);
64         cprintf("%4d",tprice);
65         gotoxy(46,10);
66         cprintf("%4d",tliter);
67         if ( ( ( byteread = inportb(PORTB) ) & 0x01 ) == 0 )
68             break;
69     }
70 }
71 /* ----- */
72 /* Get the Petroleum Type */
73 /* ----- */
74 void getpetroleum( )
75 {
76     unsigned char byteread;
77
78     /* get the Petroleum signal */
79     while ( ( ( byteread = inportb(PORTA) ) & 0x07 ) == 0 )
80         ;
81
82     /* check if PA0 = 1, mean add Petroleum A */
83     if ( ( byteread & 0x01 ) == 0x01 )
84     {
85         gotoxy(23,4);
86         textcolor(BLACK);
87         textbackground(WHITE);
88         cprintf("Petroleum A");

```

```

89     textcolor(WHITE);
90     textbackground(BLACK);
91     uprice = 20;
92     return;
93 }
94
95 /* check if PA1 = 1, mean add Petroleum B */
96 if ( ( byteread & 0x02 ) == 0x02 )
97 {
98     gotoxy(38,4);
99     textcolor(BLACK);
100    textbackground(WHITE);
101    cprintf("Petroleum B");
102    textcolor(WHITE);
103    textbackground(BLACK);
104    uprice = 18;
105    return;
106 }
107
108 /* check if PA2 = 1, mean add Petroleum C */
109 if ( ( byteread & 0x04 ) == 0x04 )
110 {
111     gotoxy(53,4);
112     textcolor(BLACK);
113     textbackground(WHITE);
114     cprintf("Petroleum C");
115     textcolor(WHITE);
116     textbackground(BLACK);
117     uprice = 16;
118     return;
119 }
120 }
121 /* ----- */
122 /* Print the program title */
123 /* ----- */
124 void printtitle( )
125 {
126     gotoxy(32,1);
127     cprintf("Petroleum application");
128     gotoxy(23,4);
129     cprintf("Petroleum A");
130     gotoxy(38,4);
131     cprintf("Petroleum B");
132     gotoxy(53,4);

```



```

133     cprintf("Petroleum C");
134     gotoxy(10,5);
135     cprintf("Unit Price      20          18          16");
136     gotoxy(32,8);
137     cprintf("Total Money:  000");
138     gotoxy(32,10);
139     cprintf("Total Liter:  000");
140 }

```

程序示例 oil.c 解释:

1. 第 8 行是用于设置 SETPORT 为控制寄存器的口, 值是 0x3e3.
2. 第 9 行是设置 PORTA 口, 值是 0x3c0.
3. 第 10 行是设置 PORTB 口, 值是 0x3e1.
4. 第 21 行是设置屏幕为一个窗口, 这样我们便可以使用 cprintf( ) 函数达到以反白显示某些字符串的目的.
5. 第 22 行是清除屏幕内容.
6. 第 25 行至第 26 行是用于设置 PORTA 和 PORTB 专供输入使用.
7. 第 31 行是调用 printtitle( ) 函数显示程序标题.
8. 第 32 行是调用 getpetroleum( ) 函数读取要加油的种类.
9. 第 33 行是调用 addpetroleum( ) 函数处理加油过程.
10. 第 34 行是令程序暂停 5 秒.
11. 第 35 行是用于清除屏幕内容.
12. 第 36 行是将光标移至 (23,10) 位置.
13. 第 37 行是显示字符串 "Press any key to exit the program.".
14. 第 38 行是令程序暂停 5 秒.
15. 第 39 行是用于清除屏幕内容.
16. 第 29 行至第 40 行是一个循环, 这样循环会检查是否有键盘输入, 如果有键盘输入则退出此循环.

函数 addpetroleum( ) 解释:

1. 第 49 行是令油总价变量 tprice 为零.
2. 第 50 行是令油总量变量 tliter 为零.
3. 第 53 行是读取 PORTB 的值.
4. 第 54 行至第 55 行是一个循环, 此循环将持续, 直至所读取的 PORTB 内的 PB0=1 (表示开始加油了) 才退出.
5. 第 60 行是令程序暂停 1 秒.
6. 第 61 行是计算油总量.
7. 第 62 行是计算油总价.
8. 第 63 行是将光标移至 (46,8) 位置.
9. 第 64 行是列出油总价.
10. 第 65 行是将光标移至 (46,10) 位置.

11.第 66 行是列出油总量。

12.第 67 行和 68 行检查是否停止加油( $PB0=0$ )，如果是则退出循环。

13.第58行和第69行是一个循环，只有在停止加油( $PB0=0$ )的情况下才可退出此循环。

函数 `getpetroleum()` 解释：

1. 第79行至第80行是读取PORTA值，如果PA0，PA1和PA2值都为0，则表示没有指明所加的油类，此循环将继续。

2. 第83行至第93行检查是否加油A，如果是则将字符串“Petroleum A”以反白显示，同时令单位油价是 20。

3. 第96行至第106行检查是否加油B，如果是则将字符串“Petroleum B”以反白显示，同时令单位油价是 18。

4. 第109行至第119行检查是否加油C，如果是则将字符串“Petroleum C”以反白显示，同时令单位油价是 16。

函数 `printtitle()` 解释：

1. 第 126 行是将光标移至 (32,1) 位置。

2. 第 127 行是显示字符串“Petrolrum application”。

3. 第 128 行是将光标移至 (23,4) 位置。

4. 第 129 行是显示字符串“Petroleum A”。

5. 第 130 行是将光标移至 (38,4) 位置。

6. 第 131 行是显示字符串“Petroleum B”。

7. 第 132 行是将光标移至 (53,4) 位置。

8. 第 133 行是显示字符串“Petroleum C”。

9. 第 134 行是将光标移至 (10,5) 位置。

10.第 135 行是显示字符串“Unit Price 20 18 16”。

11.第 136 行是将光标移至 (32,8) 位置。

12.第 137 行是显示字符串“Total Money: 000”。

13.第 138 行是将光标移至 (32,10) 位置。

14.第 139 行是显示字符串“Total Liter: 000”。

## 18.4 结 论

8255A 是一个非常好用的 I/O 处理器。配合系统的运行，很多功能均可通过它完成。

## 18.5 思考题

1. 请修改此程序，让三种油不可能同时被选择。
2. 请修改软件，使拥有贵宾卡的人享有九折优惠。

## 第十九章 键盘模拟专题

### 本实验目的

1. 通过 8255A 的扫描，模拟键盘的工作方式。
2. 通过此实验专题可以了解

■接口 I/O 控制

■键盘的基本知识

### 本章内容

- 19.1 工作目标
- 19.2 硬件设计
- 19.3 软件设计
- 19.4 结论
- 19.5 思考题

### 19.1 工作目标

模拟键盘就是在自作的键盘上按下一个字键之后，在屏幕上显示出这个字型。

### 19.2 硬件设计

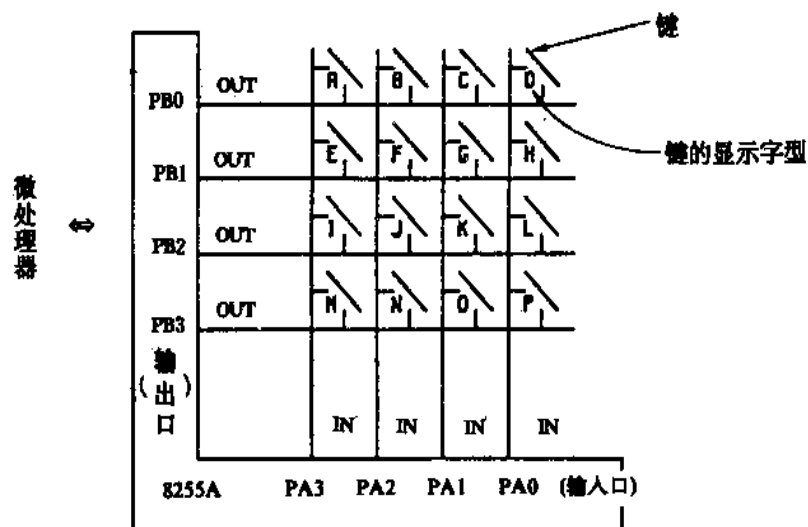
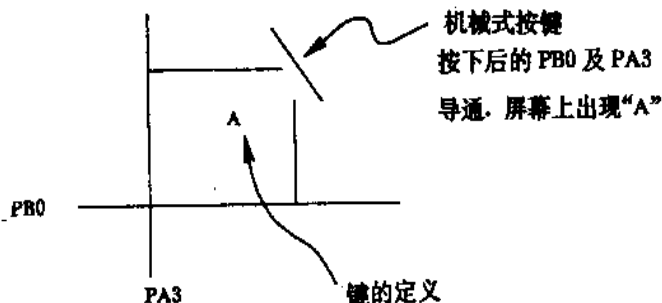


图 19.1 16 键 (4×4) 的键盘结构

PB0~PB3 送出扫描信号，PA0~PA3 负责接收。譬如 PB2 送出“1”且由 PA2 收到，则表示键“J”被按下。此时计算机送出“J”的字形到屏幕上。

下图为键的单元，当键按下之后 PA3 可以收到 PB0 送出的扫描信号。

键的单元



### 19.3 软件设计

#### 程序示例 key.c

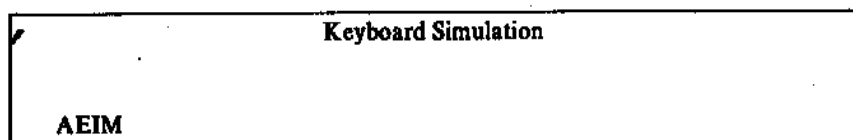
键盘的模拟，本程序在运行时会不断输出 01,02,04,08 至 PORTB，然后，从 PORTA 读取 PA0,PA1,PA2 和 PA3 数据，基本上我们将 PORTB 的数据定为扫描码的高字节，PORTA 的数据定为扫描码的低字节。然后以下列二进制数据判别按键数据。

```
0000000100001000 : A 键扫描码
0000000100000100 : B 键扫描码
0000000100000010 : C 键扫描码
0000000100000001 : D 键扫描码
0000001000001000 : E 键扫描码
0000001000000100 : F 键扫描码
0000001000000010 : G 键扫描码
0000001000000001 : H 键扫描码
0000010000001000 : I 键扫描码
0000010000000100 : J 键扫描码
0000010000000010 : K 键扫描码
0000010000000001 : L 键扫描码
0000100000001000 : M 键扫描码
0000100000000100 : N 键扫描码
0000100000000010 : O 键扫描码
0000100000000001 : P 键扫描码
```

本程序在运行时，首先屏幕将只列出程序标题，如下所示：

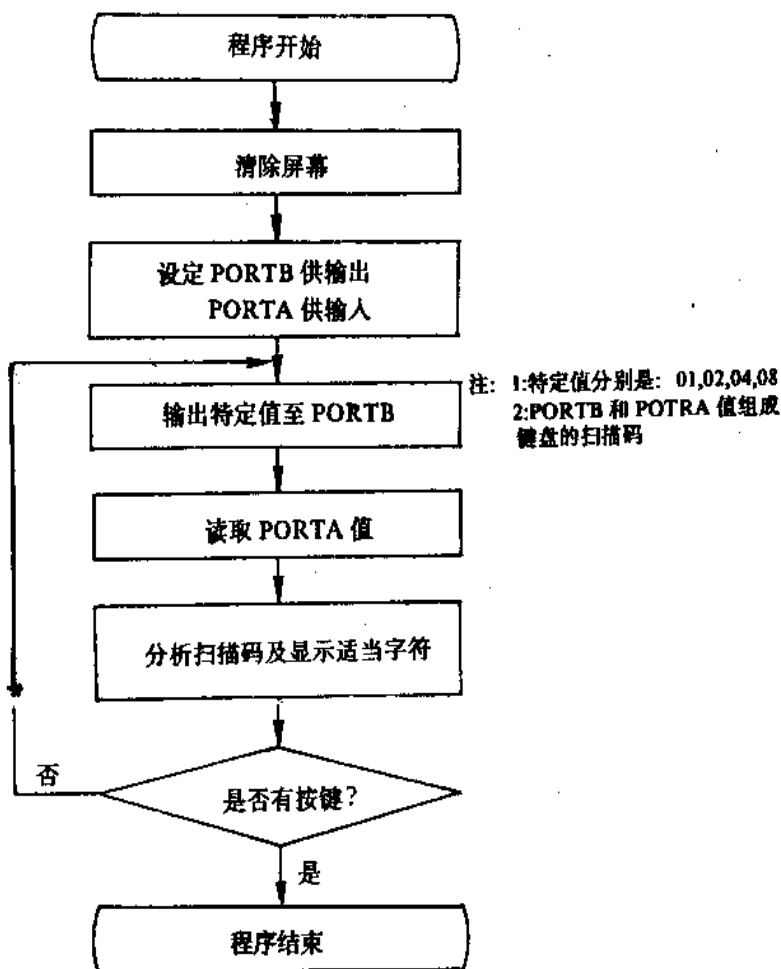
Keyboard Simulation

当你按下任何键时，屏幕将列出所按的键，如下图所示：



同时本程序的 delay(500)用来设置延迟 0.5 秒，这是本程序所定的键盘反应速率，读者可以自行更改此键盘反应速率。

本程序的流程如下所示：



```

01  /* ----- */
02  /*      Program Name : key.c      */
03  /*      Keyboard simulation.      */
04  /*      For 8255A, Mode 0, PORTA input and PORTB output      */
05  /* ----- */
06  #include <dos.h>
07  #include <conio.h>
  
```

```

08 #define SETPORT      0x3e3
09 #define PORTA        0x3e0
10 #define PORTB        0x3e1
11 #define ACODE        0x0108
12 #define BCODE        0x0104
13 #define CCODE        0x0102
14 #define DCODE        0x0101
15 #define ECODE        0x0208
16 #define FCODE        0x0204
17 #define GCODE        0x0202
18 #define HCODE        0x0201
19 #define ICODE        0x0408
20 #define JCODE        0x0404
21 #define KCODE        0x0402
22 #define LCODE        0x0401
23 #define MCODE        0x0808
24 #define NCODE        0x0804
25 #define OCODE        0x0802
26 #define PCODE        0x0801
27 unsigned char CODEH, CODEL;
28 void main( )
29 {
30     void process( );
31     unsigned char bytewrite, byteread;
32
33     clrscr( );          /* clear the screen */
34
35     /* set up the output port */
36     bytewrite = 0x92;
37     outportb(SETPORT,bytewrite);
38
39     /* print the program title */
40     gotoxy(32,1);
41     printf("Keyboard Simulation");
42
43     /* set the initial cursor position */
44     gotoxy(1,5);
45
46     /* main loop */
47     while ( !kbhit( ) )
48     {
49         /* case 1 */
50         bytewrite = 0x01;
51         CODEH = bytewrite;

```

```

52     outportb(PORTB,bytewrite);
53     byteread = inportb(PORTA);
54     CODEL = byteread;
55     process( );
56
57 /* case 2 */
58     bytewrite = 0x02;
59     CODEH = bytewrite;
60     outportb(PORTB,bytewrite);
61     byteread = inportb(PORTA);
62     CODEL = byteread;
63     process( );
64
65 /* case 3 */
66     bytewrite = 0x04;
67     CODEH = bytewrite;
68     outportb(PORTB,bytewrite);
69     byteread = inportb(PORTA);
70     CODEL = byteread;
71     process( );
72
73 /* case 4 */
74     bytewrite = 0x08;
75     CODEH = bytewrite;
76     outportb(PORTB,bytewrite);
77     byteread = inportb(PORTA);
78     CODEL = byteread;
79     process( );
80 }
81 }
82 void process( )
83 {
84     int testing;
85
86     testing = CODEL + CODEH * 0x100;
87     switch ( testing )
88     {
89         case ACODE : printf("A");
90                     delay(500);
91                     break;
92         case BCODE : printf("B");
93                     delay(500);
94                     break;
95         case CCODE : printf("C");

```

```

96             delay(500);
97             break;
98     case DCODE : printf("D");
99             delay(500);
100            break;
101    case ECODE : printf("E");
102            delay(500);
103            break;
104    case FCODE : printf("F");
105            delay(500);
106            break;
107    case GCODE : printf("G");
108            delay(500);
109            break;
110    case HCODE : printf("H");
111            delay(500);
112            break;
113    case ICODE : printf("I");
114            delay(500);
115            break;
116    case JCODE : printf("J");
117            delay(500);
118            break;
119    case KCODE : printf("K");
120            delay(500);
121            break;
122    case LCODE : printf("L");
123            delay(500);
124            break;
125    case MCODE : printf("M");
126            delay(500);
127            break;
128    case NCODE : printf("N");
129            delay(500);
130            break;
131    case OCODE : printf("O");
132            delay(500);
133            break;
134    case PCODE : printf("P");
135            delay(500);
136            break;
137 }
138 }

```



程序示例 key.c 解释:

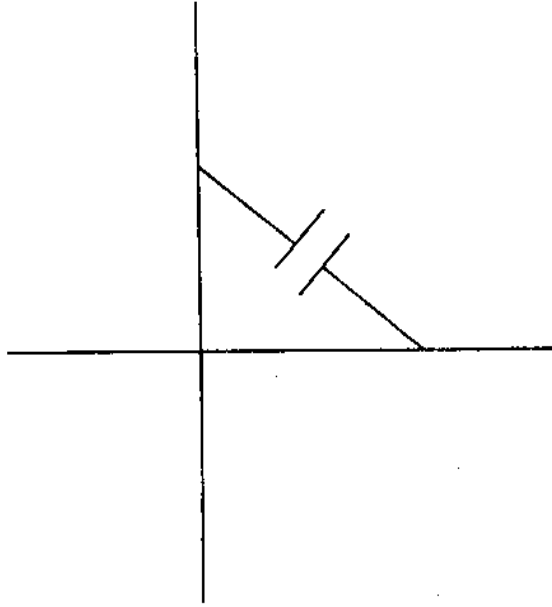
1. 第 8 行是用于设置 SETPORT 为控制寄存器的口值是 0x3c3.
  2. 第 9 行是设置 PORTA 口值是 0x3c0.
  3. 第 10 行是设置 PORTB 口值是 0x3c1.
  4. 第 11 行是设置 ACDDE (A 扫描字符) 值是 0x0108.
  5. 第 12 行是设置 BCDDE (B 扫描字符) 值是 0x0104.
  6. 第 13 行是设置 CCDDE (C 扫描字符) 值是 0x0102.
  7. 第 14 行是设置 DCDDE (D 扫描字符) 值是 0x0101.
  8. 第 15 行是设置 ECDDE (E 扫描字符) 值是 0x0208.
  9. 第 16 行是设置 FCDDE (F 扫描字符) 值是 0x0204.
  10. 第 17 行是设置 GCDDE (G 扫描字符) 值是 0x0202.
  11. 第 18 行是设置 HCDDE (H 扫描字符) 值是 0x0201.
  12. 第 19 行是设置 ICDDE (I 扫描字符) 值是 0x0408.
  13. 第 21 行是设置 JCDDE (J 扫描字符) 值是 0x0404.
  14. 第 21 行是设置 KCDDE (K 扫描字符) 值是 0x0402.
  15. 第 22 行是设置 LCDDE (L 扫描字符) 值是 0x0401.
  16. 第 23 行是设置 MCDDE (M 扫描字符) 值是 0x0808.
  17. 第 24 行是设置 NCDDE (N 扫描字符) 值是 0x0804.
  18. 第 25 行是设置 OCDDE (O 扫描字符) 值是 0x0802.
  19. 第 26 行是设置 PCDDE (P 扫描字符) 值是 0x0801.
  20. 第 33 行是清除屏幕内容.
  21. 第 36 行和第 37 行是设置 PORTA 供输入, PORTB 供输出.
  22. 第 40 行是将光标移至 (32,1) 位置.
  23. 第 41 行是显示字符串 "Keyboard Simulation".
  24. 第 44 行是将光标移至 (1,5) 位置.
  25. 第 50 行至第 55 行是情况 1. 也就是将测试的高字节设置成 0x01 送至 PORTB, 然后从 PORTA 读取值 (低字节值), 最后调用 process( ) 函数处理.
  26. 第 58 行至第 63 行是情况 2, 也就是将测试的高字节设置成 0x02.
  27. 第 66 行至第 71 行是情况 3, 也就是将测试的高字节设置成 0x04.
  28. 第 74 行至第 79 行是情况 4, 也就是将测试的高字节设置成 0x08.
- 函数 process( ) 解释
1. 第 86 行是将低字节值 (CODEL) 和高字节值 (CODEH) 组成 testing 变量.
  2. 第 87 行至第 137 行是按 testing 值测试应是哪一个字符然后予以显示.

## 19.4 结 论

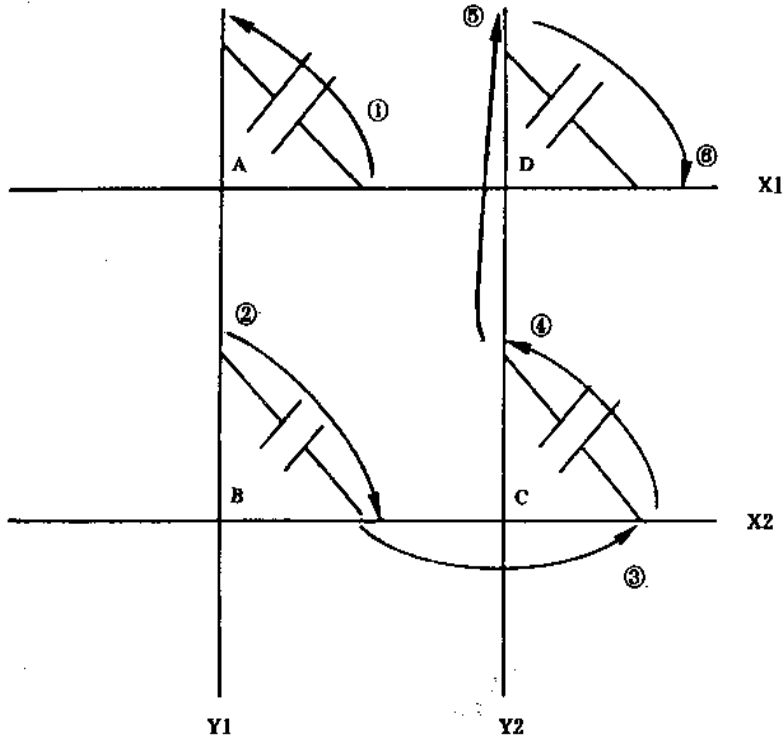
在 APPLE II 时代, 6502 微型处理机除了本身的运行之外, 仍要负责键盘的处理. 常用的键盘扫描用 IC 也就是 8255A.

## 19.5 思考题

1. 计算机键盘有机械式及电容式两种，机械式的如本章中所述，电容式的键如下图所示。

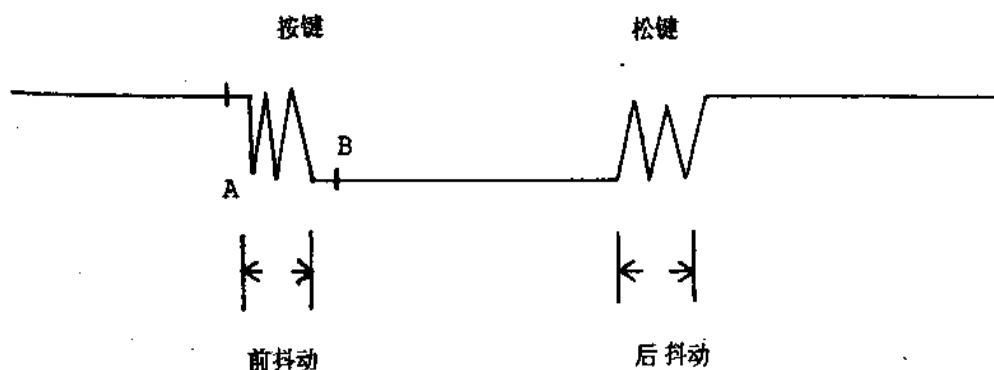


这种电容式的键好处为在 A,B,C 键同时按下时 D 键不会像机械式键盘一样自动反应。



如上图，X1 为扫描信号，它通过①-⑤路径传至⑥的信号被 Y1 接收时已太微弱，所以并不认为 D 键被按下。但在机械式的键盘中，因为它们完全是直接接触，所以当 A，B，C 键同时被按下时，通过①-⑥的信号几乎相同，所以 D 键自动反应。这也就是电容式 IC 所标榜的特色。请问：机械式的键盘如何防范这个问题？

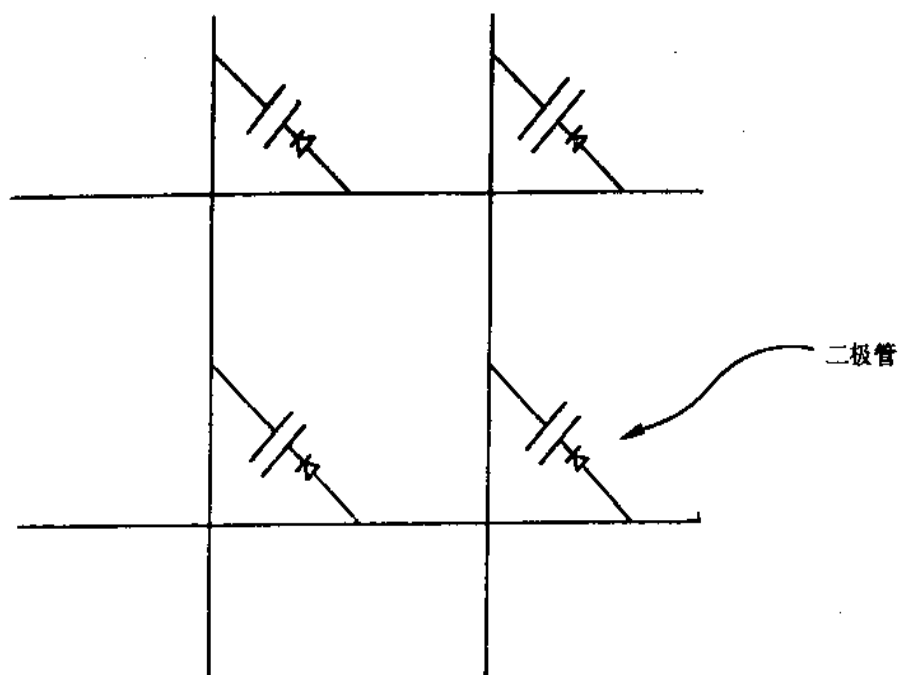
2. 键盘的触摸会有抖动现象如下：



这种抖动现象会让计算机分不清是否有按键或是按了很多次。如何以软件解决此问题？

提示：

1. 利用二极管隔开每个键，如图。



2. 软件可以在每次读取A点后隔一段时间，如5ms后再检查B点，如果是低电平则表示键被按下，如果为高电平则表示未按。

## 第二十章 声音控制（一）专题

### 本实验目的

1. 编程8255A，使其输出一些声音控制波，用以振动外接的喇叭，通过此实验，读者可以了解

■声音产生的原理

■接口 I/O 与声音产生的关系

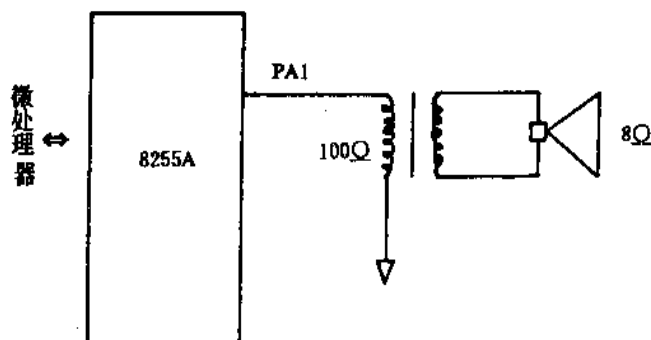
### 本章内容

- 20.1 工作目标
- 20.2 硬件设计
- 20.3 软件设计
- 20.4 结论
- 20.5 思考题

### 20.1 工作目标

由 IBM PC 送出信号给 8255A，用以振动外接喇叭，产生 2 秒钟长的机关枪子弹声。在下一章中将介绍另一种声音产生方式。

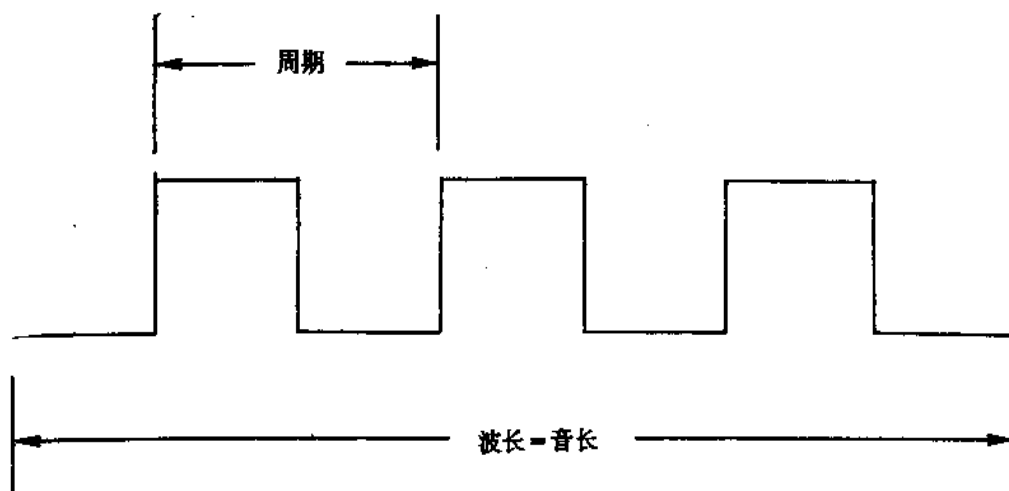
### 20.2 硬件设计



一个  $8\Omega$  的喇叭并不容易由 8255A 的口直接驱动，所以必须加线圈。这个线圈可以由简单的变压器线圈取代。笔者所用的线圈即是这样。

**声音的产生原理：**

声音的产生可借助快速的切换喇叭（ON 和 OFF）而完成。一个完整的方波为周期，它的倒数即为频率。下图为声音产生波。



80286 微型处理机的基本脉冲为 6MHZ，它的基本时钟周期为  $1/6\text{MHZ} = 167\text{ns}$ ， $1\text{ns} = 10^{-9}$  秒。80286 每一个指令所需的脉冲数可以参考 80286 的指令集。

## 20.3 软件设计

### 程序示例 sound.c

设计一个能产生 600Hz 音频，且持续二秒钟的声音。

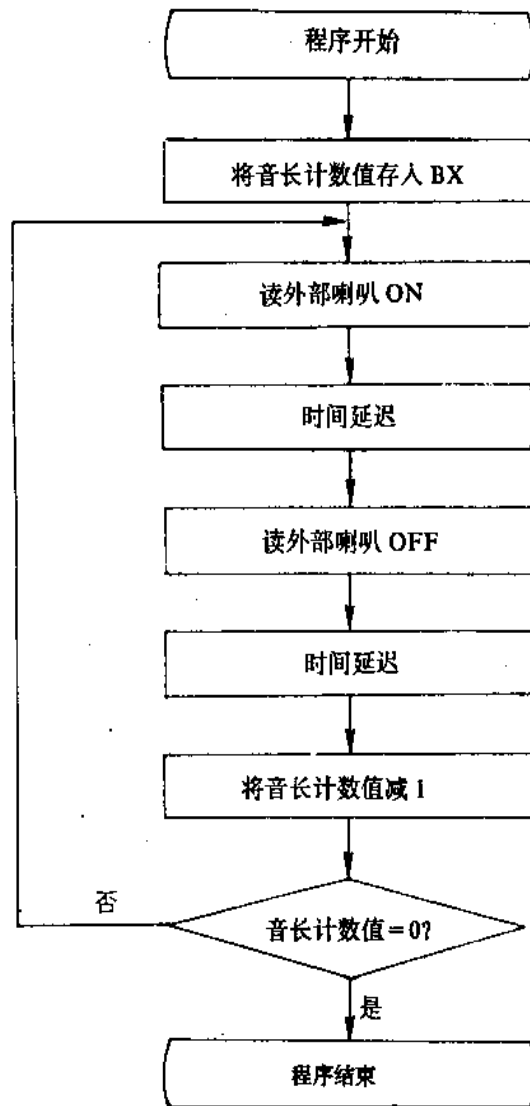
$$CX = \frac{1}{22 \times 167 \times 10^{-9} \times 600} = 454$$

$$\text{每一方波时间} = \frac{1}{600\text{Hz}} = 1.66\text{毫秒}$$

$$\text{周波数 } BX = \frac{2\text{秒}}{1.66\text{毫秒}} = 1200$$

本程序将用 DURA 变量地址存放 BX，用 FREQ 地址存放 CX。

本程序的工作流程如下所示：



```

01  /* ----- */
02  /*      Program Name : sound.c      */
03  /*      2 second and 500 Hz sound effects generation      */
04  /*      For 8255A,   Mode 0 application      */
05  /* ----- */
06  #include <dos.h>
07  #define SETPORT  0x3e3
08  #define PORTA    0x3e0
09  void main( )
10  {
11      unsigned char bytewrite;
12      int i;
13

```

```

14  /* set up the output port */
15      bytewrite = 0x80;
16      outportb(SETPORT,bytewrite);
17
18      for ( i = 0; i < 200; i++ )
19      {
20          bytewrite = 2;
21          outportb(PORTA,bytewrite); /* turn on speaker, let PA1 = 1 */
22          delay(2);                  /* delay 0.002 sec */
23          bytewrite = 0;
24          outportb(PORTA,bytewrite); /* turn off speaker, let PA1 = 0 */
25          delay(2);
26      }
27 }

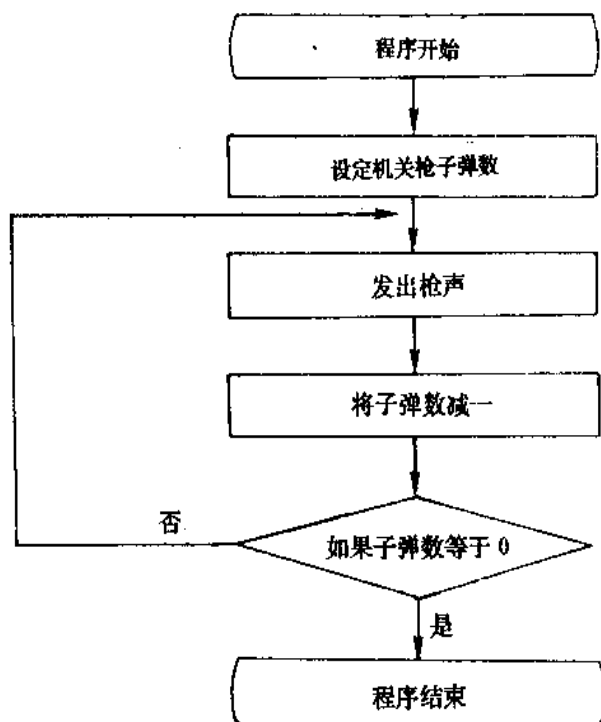
```

程序范例 sound.c 解释:

1. 第 7 行是用于设置 SETPORT 为控制寄存器的口, 其值为 0x3e3.
2. 第 8 行是用于设置 PORTA 口, 值是 0x3e0.
3. 第 12 行是用于设置 PORTA 口, 专供输出使用.
4. 第 20 行是设置 bytewrite 变量值为 2.
5. 第 21 行是将 bytewrite 值输出至 PORTA, 也就是令 PA1 等于 1.
6. 第 22 行是令时间延迟 0.002 秒.
7. 第 23 行至 28 行是设置 bytewrite 变量值为 0.
8. 第 24 行是将 bytewrite 值输出至 PORTA, 也就是令 PA1 等于 0.
9. 第 25 行是令时间延迟 0.002 秒.
10. 第 18 行至第 26 行是一个可运行 200 次的循环.

程序示例 gun.c

产生机关枪声音, 本程序每运行一次便可产生一个含 20 响的机关枪声音.  
本程序工作流程如下所示:



```

01 /* ----- */
02 /*      Program Name : gun.c      */
03 /*      Machine gun sound generation      */
04 /*      For 8255A,   Mode 0 application      */
05 /* ----- */
06 #include <dos.h>
07 #define  SETPORT  0x3e3
08 #define  PORTA    0x3c0
09 void main( )
10 {
11     unsigned char bytewrite;
12     int num = 20;
13     int i;
14
15     /* set up the output port */
16     bytewrite = 0x80;
17     outportb(SETPORT,bytewrite);
18
19     while ( num > 0 )
20     {
21         for ( i = 0; i < 50; i++ )
22         {
23             bytewrite = 2;

```



```

24      outportb(PORTA,bytewrite); /* turn on speaker, let PA1 = 1 * /
25      delay(2);                  /* delay 0.002 sec * /
26      bytewrite = 0;
27      outportb(PORTA,bytewrite); /* turn off speaker, let PA1 = 0 * /
28      delay(2);
29  }
30      num--;
31      delay(30);
32  }
33 }

```

程序示例 gun.c 解释:

1. 第 7 行是用于设置 SETPORT 为控制寄存器的口, 值为 0x3e3.
2. 第 8 行是用于设置 PORTA 口, 值是 0x3e0.
3. 第 16 行和第 17 行是设置 PORTA 口, 专供输出使用.
4. 第 23 行是设置 bytewrite 变量值为 2.
5. 第 24 行是将 bytewrite 值送至 PORTA, 相当于令 PA1=1.
6. 第 25 行是令时间延迟 0.002 秒.
7. 第 26 行是将 bytewrite 设为 0.
8. 第 27 行是将 bytewrite 值送至 PORTA, 相当于令 PA1=0.
9. 第 28 行是令时间延迟 0.002 秒.
10. 第 21 行至第 29 行是一个循环将运行 50 次.
11. 第 30 行是将 num (机关枪响声次数) 减 1.
12. 第 32 行是令时间延迟 0.032 秒.
13. 第 19 行至第 32 行是一个循环, 将运行 num 次.

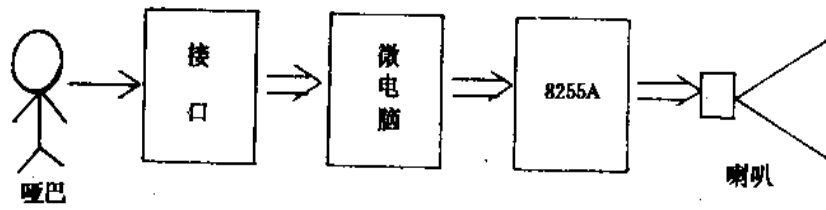
## 20.4 结 论

声音的控制并不难, 只要把握住发音的原理, 配合着程序设计即可设计出优美的计算机音乐.

在硬件设计中, 我们只用一个位即发出了声音, 如果我们装配更多的喇叭, 即可组成计算机音乐交响团.

## 20.5 思考题

1. 利用外接的喇叭定出 Do, Re, Mi, Fa, So, La, Si, Do 等基本音.
2. 利用 IBM PC 本身的喇叭和外接的喇叭唱出二重唱“望春风”.
3. 如何利用微型计算机帮助哑巴说话?



请您说明一个接口，接受哑巴喉咙振动波将信息传给微型计算机以发出声音。21 世纪的哑巴歌星可能因为获得您的协助而得到金钟奖最佳男女歌手奖。

## 第二十一章 声音控制 (二) 专题

### 本章学习目的

1. 编程8253 / 8254, 输出一些声音的控制波, 用以驱动外接喇叭。
2. 通过此实验, 读者可以了解

■声音产生的原理

■接口 I/O 与声音产生的关系

■如何在控制声音的同时, 让微型处理机处理其它工作。

### 本章内容

- 21.1 工作目标
- 21.2 硬件设计
- 21.3 软件设计
- 21.4 结论
- 21.5 思考题

当我们在玩小蜜蜂电动玩具时, 可以发现屏幕上的子弹伴随着声音移动, 时而咻咻扫过, 时而爆炸, 这种同时控制声音与屏幕的程序如何设计?

在前一章中, 我们提及声音的产生, 它主要是借助 8255A 操作, 微型处理机在编程 8255A 的期间, 不能做其它的事, 只能全力控制波的 ON, OFF, 借由 ON, OFF 的转变快慢决定声音的频率, 由 ON, OFF 的数目决定声音的长短。

在本章中将讨论另一种发音方式: 利用计时器 8253 / 8254, 这样微型处理机可同时控制声音及其它外设。

利用计时器发出声音:

在 IBM PC 内部有 3 个计时器:

计时器 2: DMA(Direct Memory Access)数据传送时使用。

计时器 1: 当系统时间使用。

计时器 0: 连接喇叭, 控制 IBM PC 的声音部分。

本章将不利用主机板的 8253 / 8254, 而使用外接 I/O 卡上的 8254 产生声音。

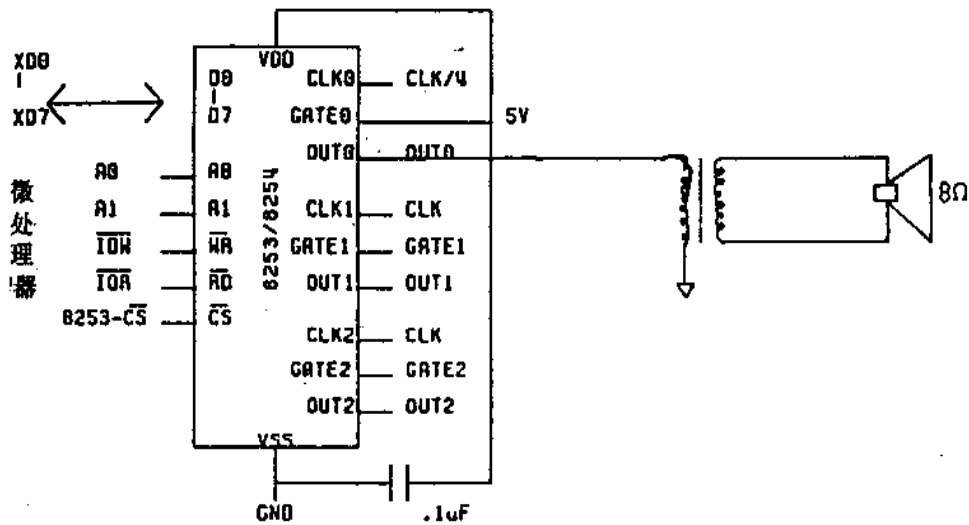
### 21.1 工作目标

编程控制外界的喇叭, 用以产生两只老虎的计算机音乐制作。

二只老虎音乐, 乐谱如下:

4/4 | 1 2 3 1 | 1 2 3 1 | 3 4 5 - | 3 4 5 - |  
| 56 54 3 1 | 56 54 3 1 | 1 5 1 - | 1 5 1 - |

## 21.2 硬件设计



## 21.3 软件设计

我们使用第 0 个计时器，当这个专题的发音来源。

利用计时器产生声音有三个步骤：

### 1. 设置计时器0的初始状况

```
SETPORT = 0x3eb;
bytewrite = 0x36;
outportb(SETPORT, bytewrite);
```

参考第七章，我们可以知道上述指令的目的。

- 8253 / 8254 的控制寄存器地址为 0x3eb。
- bytewrite = 0x36 表示我们编程了 8253 / 8254 的第 0 个计时器，数据长度为 16 位，在第 2 种操作模式下工作。

### 2. 载入一个 16 位的数值至计时器 0，编程我们要产生的音调频率。

```
PORT = 0x3e8;
bytewrite = 0xYY; /* YY 代表低字节值 */
outportb(PORT, bytewrite);
bytewrite = 0xXX; /* XX 代表高字节值 */
outportb(PORT, bytewrite);
```

- 第 0 个计数器的地址为 03E8H。
- 所载入值为 XXYYH，先送低字节，再送高字节。AX 值和频率的关系为

$$AX = 1331 \times \frac{1000}{\text{频率值}}$$

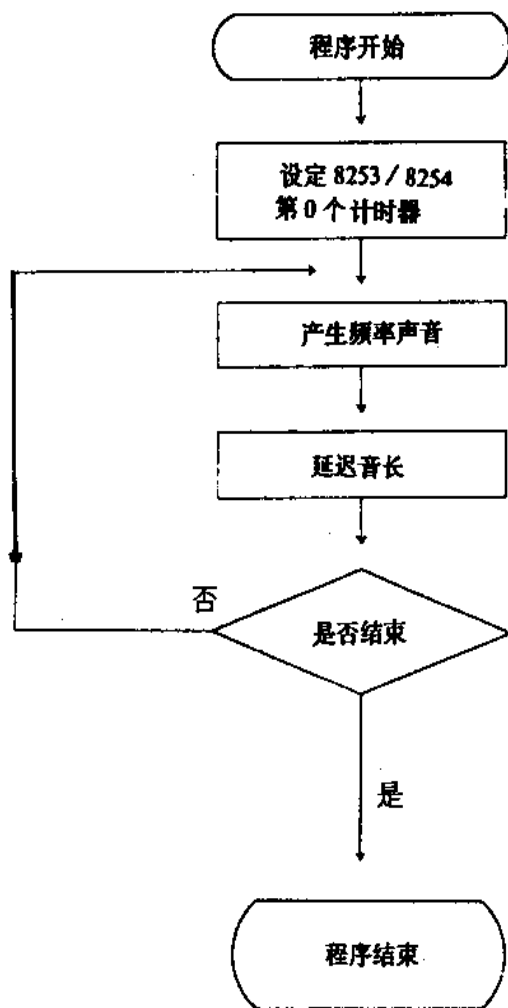
1331 为 80286 系统所须的常量。

对于不同的机型、不同速度的机种，此值（1331）必须改变。

3. 此时喇叭即有声音产生，而且此声会不断地产生，直到重新编程程序为止。在此期间微型处理机可以做屏幕控制，这种方式即是电动玩具产生声音的方式。

程序示例 mouse.c

设计两只老虎音乐，本程序流程如下所示：



```

01 /* ----- */
02 /*      Program Name : mouse.c      */
03 /*      Generate Mouse music.        */
04 /*      For 8253 / 8254 application   */
05 /* ----- */
06 #include <dos.h>

```

```

07 #define  SETPORT    0x3eb
08 #define  PORT      0x3e8
09 /* the final freq. I will generate no sound */
10 unsigned int freq[ ] = {
11     523, 587, 659, 523, 523, 587, 659, 523,
12     659, 698, 784, 1, 659, 698, 784, 1,
13     784, 880, 784, 698, 659, 523,
14     784, 880, 784, 698, 659, 523,
15     523, 392, 523, 1, 523, 392, 523, 1, 1 };
16 unsigned int duration[ ] = {
17     500, 500, 500, 500, 500, 500, 500, 500,
18     500, 500, 500, 500, 500, 500, 500, 500,
19     250, 250, 250, 250, 500, 500,
20     250, 250, 250, 250, 500, 500,
21     500, 500, 500, 500, 500, 500, 500, 500, 1 };
22
23 void main( )
24 {
25     unsigned char bytewrite;
26     int val;
27     int i;
28
29     /* set up the output port */
30     bytewrite = 0x36;
31     outportb(SETPORT,bytewrite);
32
33     /* generate the sound */
34     for ( i = 0; i <= 36; i++ )
35     {
36         val = freq[i] % 0xff;
37         outportb(PORT,val);          /* output lower byte */
38         val = freq[i] / 0xff;
39         outportb(PORT,val);          /* output higher byte */
40         delay(duration[i]);
41     }
42 }

```

程序示例 mouse.c 解释:

1. 第 7 行是用于设置 SETPORT 为控制寄存器的口, 值为 0x3eb.
2. 第 8 行是用于设置 PORT 口, 值是 0x3e8.
3. 第 10 行至第 15 行是设置声音频率.
4. 第 16 行至第 21 行是设置各声音的音长.
5. 第 30 行至第 31 行是设置 PORT 供输出.

6. 第 36 行至第 37 行是输出低字节值。

7. 第 38 行至第 39 行是输出高字节值。

8. 第 40 行是令时间延迟。

注：由于 8253 / 8254 一经驱动将无法中止，因此让计时器发出频率 1 的声音，相当于不产生声音。

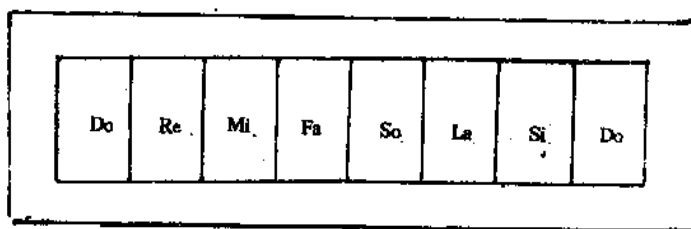
## 21.4 结 论

通过以上的讨论我们可以发现，控制 I/O 的喇叭并不困难，电子游戏程序的声音也不难设计。利用 8253 / 8254 所设计的声波可以连续不断。系统只须给一次初始值即可发出声音，此时系统可以做其他的事。

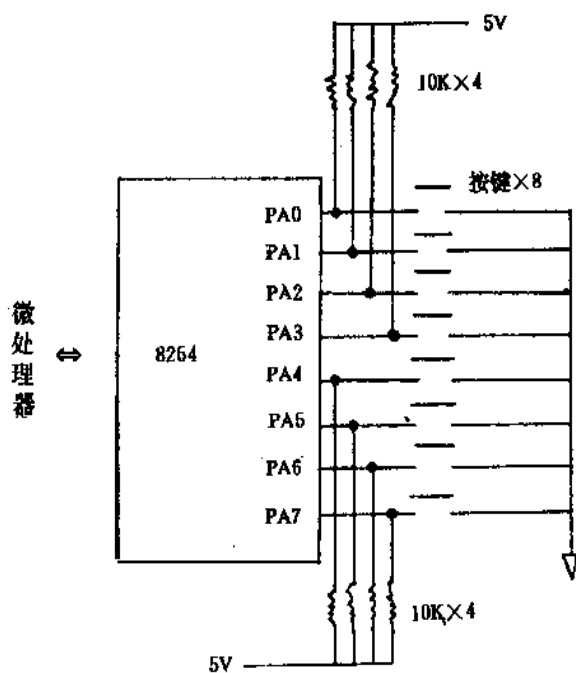
## 21.5 思考题

1. 本实验中只使用了计时器 0，请读者再使用另一个计时器，设计出一首情歌。
2. 使用外接计时器设计钢琴，必须有键盘显示在屏幕上，同时必须在按键后有正确的声音，如下图所示：

计算机屏幕显示如下：



硬件如下：







## 第三篇

# I/O 控制的思考专题研究

- 第二十二章 *ADC* 模拟 / 数字转换器
- 第二十三章 *DAC* 数字 / 模拟转换器
- 第二十四章 数字激光音响与数字电视
- 第二十五章 电话保密与数字通讯
- 第二十六章 数据通讯的接口技术
- 第二十七章 步进电机应用
- 第二十八章 *EPROM* 编程器
- 第二十九章 赌博性电动玩具与警察专题
- 第三十章 微型计算机的其它应用

## 第二十二章 ADC 模拟 / 数字转换器

### 本章学习目的

1. 了解 ADC 的工作原理
2. 了解 ADC 与微型处理机的接口
3. 了解 ADC 的应用

### 本章内容

- 22.1 ADC 结构
- 22.2 ADC-0804 简介
- 22.3 ADC 与微型处理机的接口
- 22.4 ADC 与微型处理机的应用实例
- 22.5 思考题

我们生活的大自然中充满着信息，如声音、温度、压力、重量、电波、速度、电压种种信息。这些信息大部分都是模拟信号 (Analog)。很早以前，我们就已经利用科学技术将这些信号以科学方式传播或表达。随着科学的发展，我们发觉，我们的表达方式总是有所缺陷，有不完美的地方。科学家们于是将这些信号转变为“0”与“1”的表达方式。为什么要转成数字方式呢？因为“0”与“1”是最容易存储的信号，它最容易保管和计算。ADC 是一种方便的模拟数字转换器，有了它我们在嘈杂的咖啡厅中录的音经过特殊处理，可以将嘈杂的人声、酒杯声消除，于是我们得到纯净的音乐声或对话声。

### 22.1 ADC 结构

我们以图 22.1 三位的平行比较器解释 ADC 结构。

图 22.1 为一简单的 ADC 结构。模拟电压输入与  $7V/8$ 、 $6V/8$ 、 $5V/8$ 、 $4V/8$ 、 $3V/8$ 、 $2V/8$ 、 $1V/8$  比较，结果分别为  $W7$ 、 $W6$ 、 $W5$ 、 $W4$ 、 $W3$ 、 $W2$ 、 $W1$ 。编码器用编码器将  $W7 \sim W1$  改为  $Y2$ 、 $Y1$ 、 $Y0$  输出。

本书偏重于微型处理机的应用，对于 ADC 的详细结构请参阅其它电子书籍。

### 22.2 ADC-0804 简介

在讨论 ADC 与微型计算机的接口之前，我们必须先对市面上所售的 ADC 有所了解。以国际半导体公司所生产的 ADC-0804 为例，它是一个 8 位的 I/O 转换器，它的特性如下：

1. 它拥有 8 位逐次渐近型的转换器。

4. 最大的误差  $\pm 1\text{LSB}$ .

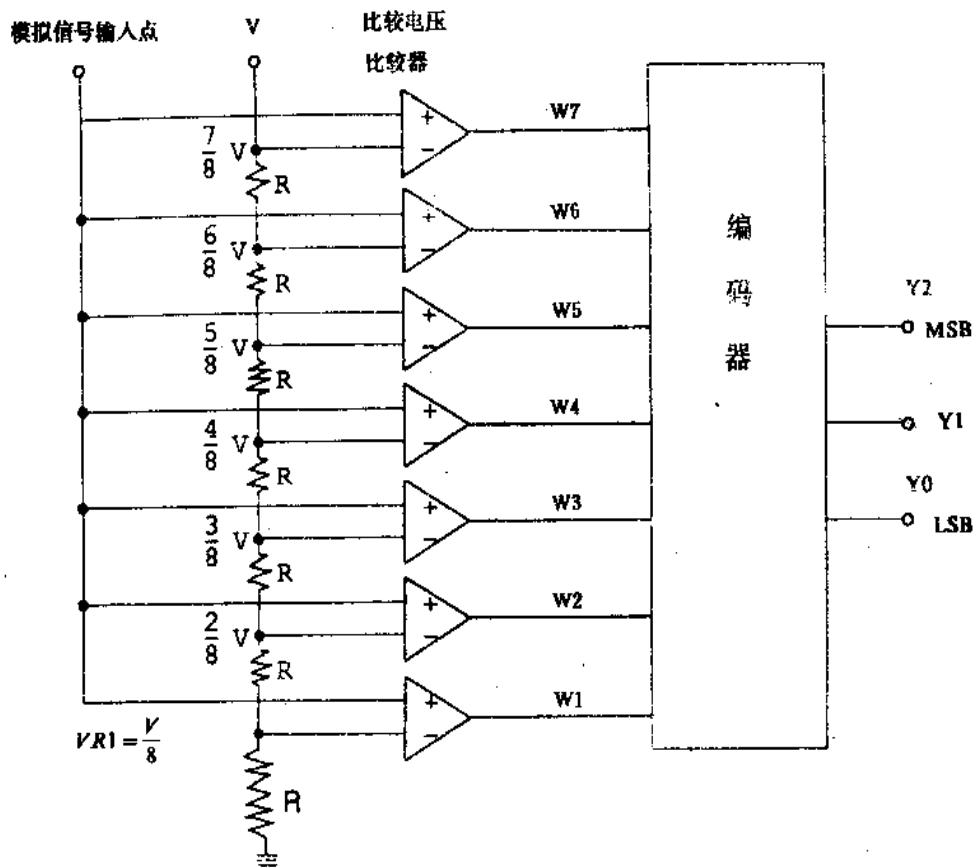


图 22.1 3 位的平行比较器

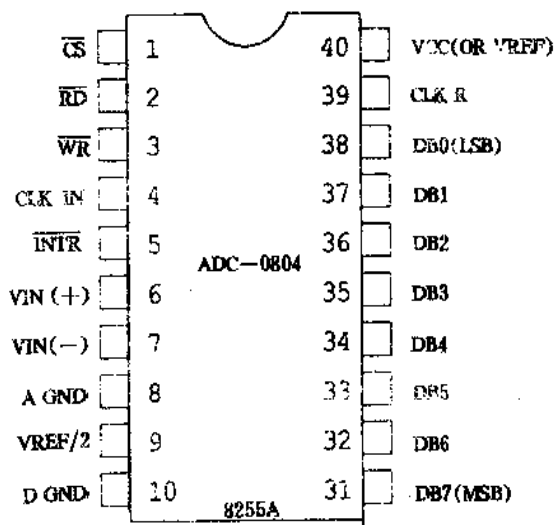


图 22.2 ADC-0804 引脚图

## 引脚介绍

$\overline{\text{CS}}$  :

Chip Enable, 芯片启动使能

$\overline{\text{RD}}$  :

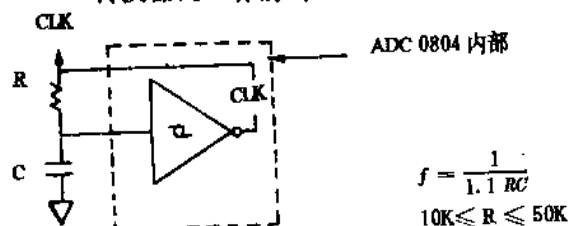
读取, 外界利用此信号读取ADC运行后的结果, 在 $\overline{\text{RD}}$ 由“0”转为“1”的瞬间 $\overline{\text{INTR}}$ 将从0变为1, 表示运行结果已被系统读走了。

$\overline{\text{WR}}$  :

写入, 外界利用此一信号复位这个IC, 并将 $\overline{\text{INTR}}$ 定为1, 当 $\overline{\text{WR}}$ 由0变为1的瞬间将触发转换器, 使转换器进入另一个转换周期。

$\text{CLK IN}$  :

转换器的工作脉冲。



这个输入引脚为史密特输入, 若与R,C相接则可自行产生一个振荡波, 它的频率值不可高于640KHz。

$\text{CLKR}$  :

$\text{CLK IN}$ 的反相输出。

$\overline{\text{INTR}}$  :

中断请求输出, 用以通知外界转换周期结束, 与微型处理机相连接时特别有用。

$\text{VCC}$  :

电源供应, 当 $\text{VREF}/2$ 引脚空接时, 此引脚可作为基准电压 $\text{VREF}$ 。

$\text{VREF}/2$  :

基准电压的输入端, 当使用此脚时, 必须将基准电压的二分之一电压接上此脚。

$\text{V+}, \text{V-}$  :

模拟电压的输入端, 模拟输入电压不可高于 $\text{V+}$ , 不可低于 $\text{V-}$ 。

$\text{A GND}$  :

模拟信号接地电源。

$\text{D GND}$  :

数字信号接地电源。

IC设计小知识:

为了避免数字转换时, 数字信号影响模拟信号的准确度, 通常这两种信号的接地电源均隔开。

$\text{DB0-DB7}$  :

三态锁存数据输出。

ADC的操作时序:

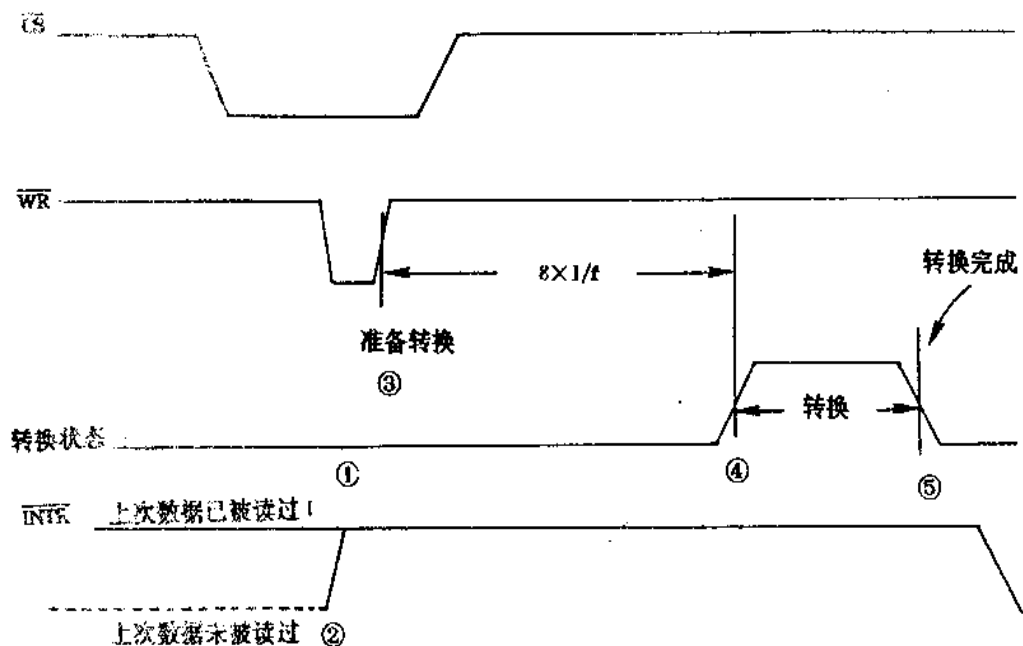
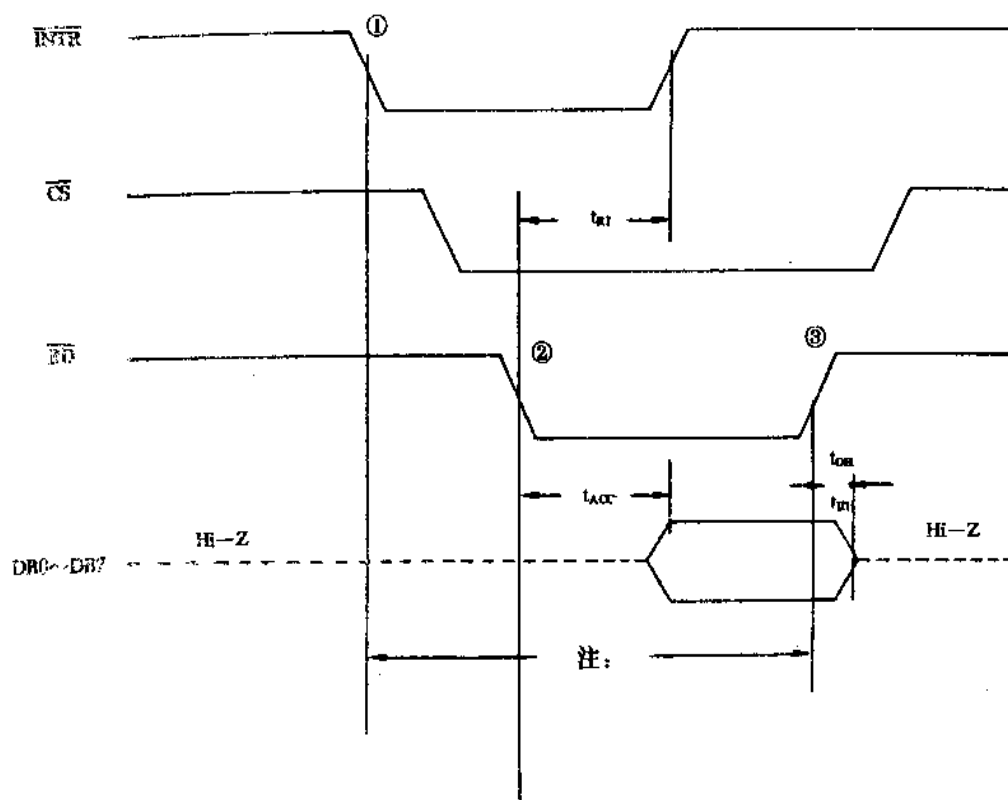


图 22.3 CS、WR的时序图



注: 在确定 $\overline{INTR}$ 有效之后, 此时间必须维持 $8/f$

$t_{RH}, t_{OH} : 125\text{nS} \sim 200\text{nS}$

$t_{R1} : 300\text{nS} \sim 450\text{nS}$

$t_{Acc} : 135\text{nS} \sim 200\text{nS}$

图 22.4 数据输出时序图

图 22.3 中:

- ①若上次的数据已被读过, 此时 $\overline{\text{INTR}}$ 为高电平。
- ②若上次的数据未被读过, 此时 $\overline{\text{INTR}}$ 为低电平。
- ③在 $\overline{\text{WR}}$ 上界边缘, ADC开始准备转换,  $8 \times 1 / f$ 的时间让输入模拟信号达到稳定。
- ④开始转换。
- ⑤转换完成。

ADC 的数据输出:

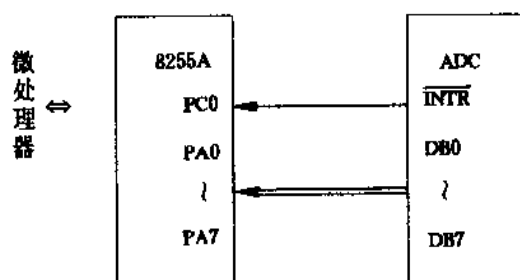
图 22.4 为 ADC-0804 数据输出的时序图

- ① $\overline{\text{INTR}}$ 通知外设转换完成, 外设可以取走数字输出了。
- ②系统准备取走数据 (数据在 DB0~DB7 上)。
- ③取走了。

## 22.3 ADC 与微型处理机的接口

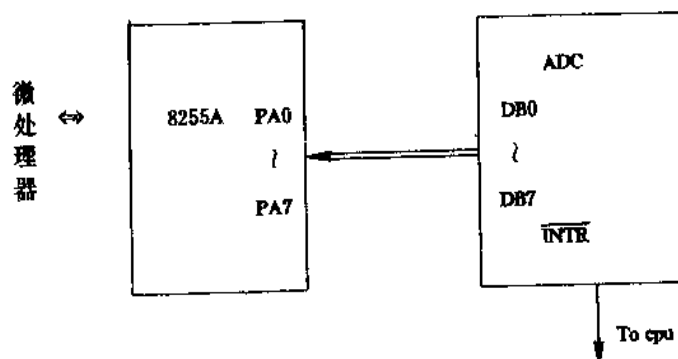
ADC 与微型处理机的接口有三种方式:

### 1. 软件数据的传输型:



当 ADC 转换完成之后,  $\overline{\text{INTR}}$ 将变为低电平, 8255A 随时读取 PC0 的值, 当 PC0=0 时, 系统送出 $\overline{\text{RD}}$ 信号, 将 DB0~DB7 的数据读回系统。在这种方式下, 因为系统随时要询问, 所以浪费系统时间。

### 2. 硬件交互式的传输型:

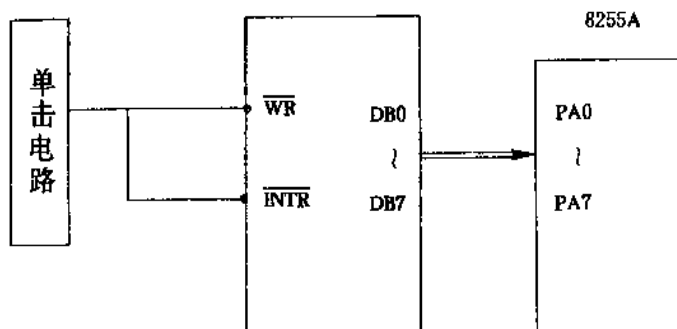


当 ADC 转换完成之后,  $\overline{\text{INTR}}$ 变为低电平, 用中断系统, 将 DB0~DB7 和数据读回。

### 3. 连续转换型:

前述两种方式，通过了微型处理机智能读取转换完的数据。事实上我们也可以利用一个外接的电路，使它不须微型处理机而达到连续操作的目的。方法如下：

- $\overline{\text{INTR}}$  直接启动  $\overline{\text{WR}}$
- 每次转换完成， $\overline{\text{INTR}}$  降为低电平，并将结果存入寄存器。
- 此时转换器被复位， $\overline{\text{INTR}}$  又变为高电平。
- $\overline{\text{WR}}$  也上升为高电平，新的转换周期又重新开始。

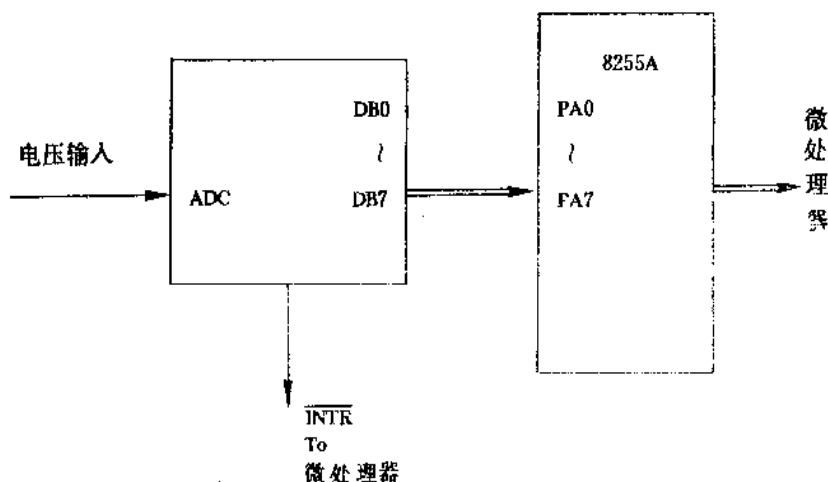


单击电路用以启动第一次转换周期。

## 22.4 ADC 与微型处理机的应用实例

### 22.4.1 电压的测量

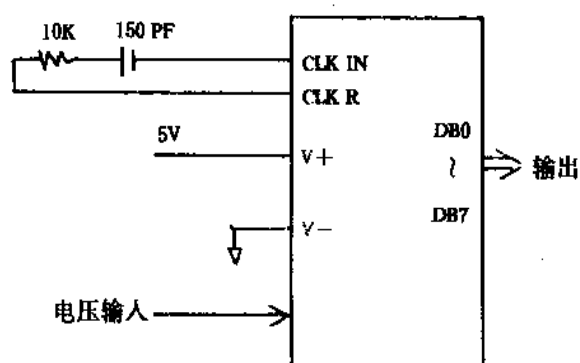
最简单的 ADC 与微型处理机应用实例即是电压的测量。



在测量之前，我们必须先计算一参考值，例如将已知的电压输入，取得 PA0~PA7 的值，下一次的电压输入后，即可由微型计算机聪明地算出它的正确电压值了。

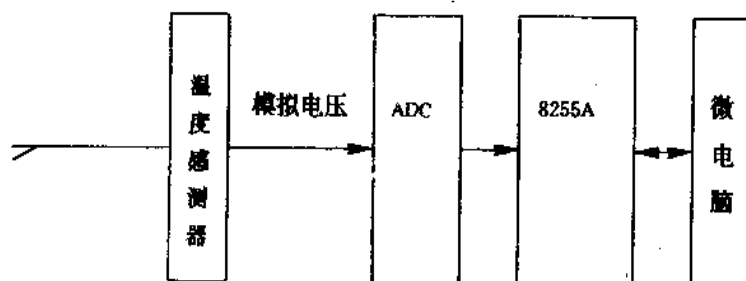
若我们所测的电压值在 5V 以内，我们设计了一个线图如下，如何用此原理设计电压表？

请您设计这个测量的系统，包含硬件编程、软件编程以及程序。Good Luck! Viel Spass!



### 22.4.2 温度的测量

此部分与电压的测量类似，只是它多了一个温度传感器 (Sensor)。



温度传感器可以将温度转变为电压，以 AD590 为例，它的工作感应温度为 $-55^{\circ}\text{C} \sim 150^{\circ}\text{C}$ ，它可将温度转换为  $4\text{V} \sim 30\text{V}$ 。

请您设计整个测量系统。

### 22.4.3 其它应用

ADC 是一个当今非常重要的电子元件，所有与数字有关的东西都必须先经过它的处理，譬如：数字电视、数字通讯、数字电压计、数字温度计等等。它的价格与精度有极大的关系，高精度由它的位数而定。一般 8 位的 ADC 并不足够用在高品质的仪器中，16 位的 ADC 需求愈来愈大，但是高精度的 ADC IC 制作不易，因为它的基准电压及电阻在 IC 制作过程中不能很精密地被控制。

## 22.5 思考题

1. 完成 22.4.1 的编程。
2. 完成 22.4.2 的编程。



## 第二十三章 DAC 数字 / 模拟转换器

### 本章学习目的

1. 了解 DAC
2. 了解 DAC 与微型处理机的接口
3. 了解 DAC 的实际应用

### 本章内容

- 23.1 DAC 的基本结构
- 23.2 DAC-08 简介
- 23.3 DAC 与微型处理机的接口
- 23.4 思考题

在前一章中提及了 ADC 的原理和简单的应用。当 ADC 将大自然的模拟信号转换为数字信号之后，计算机即可很方便地处理和运算了。运算完毕的信号仍是数字的，不容易为人类所接受。此时，我们仍然必须将它转变为模拟信号，如声音、电压、温度、重量等。本章的讨论重点即在于数字 / 模拟转换器 (DAC)。

### 23.1 DAC 的基本结构

符号说明:

$S_{N-1} \sim S_0$	为数字控制开关。
$V_R$	为比较电压。
Bit $N-1 \sim$ Bit 0	为 DAC 的输入数据位。
$V_o$	为 DAC 的输出。

当 Bit  $N-1 \sim$  Bit 0 为 1 时，其相对应的  $S_{N-1} \sim S_0$  数字控制开关导通，否则为开路。加法器主要的目的即是将数字输入对应为“1”的位转换为电压。

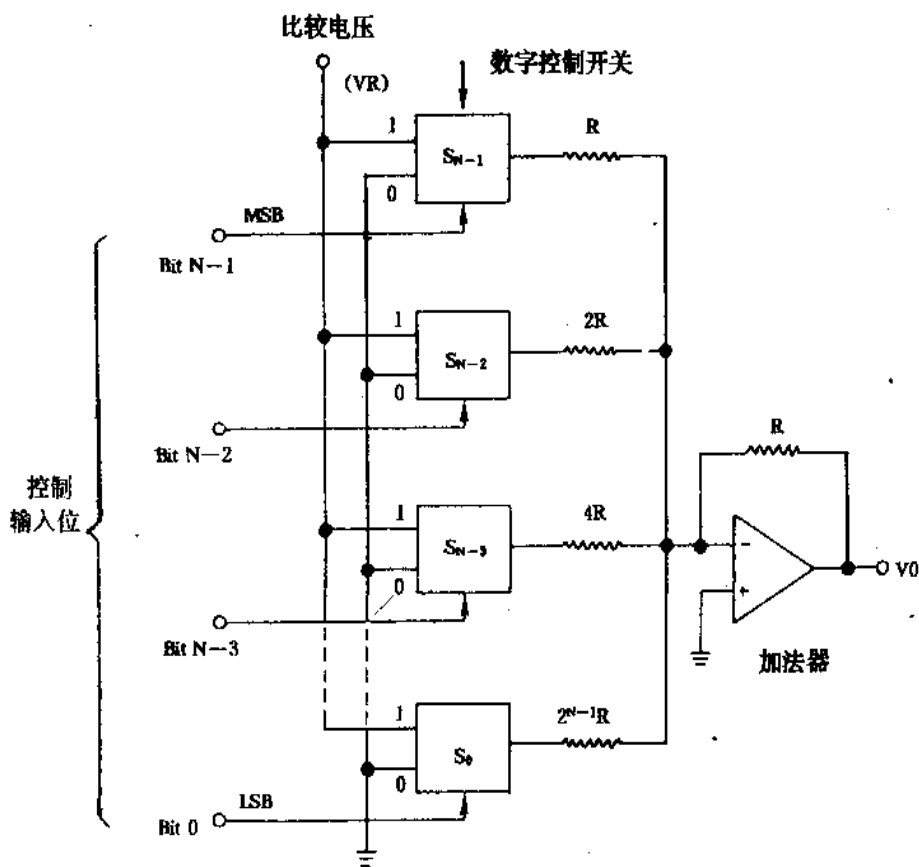


图 23.1 加权电阻式 DAC

它的输出点  $V_0$  可由下述公式得到

$$V_0 = \left( 2^{N-1} a_{N-1} + 2^{N-2} a_{N-2} + \cdots + 2^2 a_2 + 2^1 a_1 + a_0 \right) V$$

$$= \left( a_{N-1} + \frac{1}{2} a_{N-2} + \frac{1}{4} a_{N-3} + \cdots + \frac{1}{2^{N-2}} a_1 + \frac{1}{2^{N-1}} a_0 \right) 2^{N-1} V$$

对一个 4 位的输入 (假设 4 个位均为 1) 则

$$V_0 = (8a_3 + 4a_2 + 2a_1 + a_0) V$$

$$= (8 + 4 + 2 + 1) \frac{V_R R'}{8R}$$

若  $R = 20K\Omega$ ,  $R' = 10K\Omega$ ,  $V_R = -5V$

则输出电压为

$$V_0 = 15 \cdot \frac{+5 \cdot 10K\Omega}{8 \cdot 20K\Omega} = \frac{75}{16} V$$

对于  $a_0 \sim a_3$  的所有组合产生的电压整理如下:

$a_3$	$a_2$	$a_1$	$a_0$	V0
0	0	0	0	0
0	0	0	1	$1 \times 5V / 16$
0	0	1	0	$2 \times 5V / 16$
0	0	1	1	$3 \times 5V / 16$
0	1	0	0	$4 \times 5V / 16$
0	1	0	1	$5 \times 5V / 16$
0	1	1	0	$6 \times 5V / 16$
0	1	1	1	$7 \times 5V / 16$
1	0	0	0	$8 \times 5V / 16$
1	0	0	1	$9 \times 5V / 16$
1	0	1	0	$10 \times 5V / 16$
1	0	1	1	$11 \times 5V / 16$
1	1	0	0	$12 \times 5V / 16$
1	1	0	1	$13 \times 5V / 16$
1	1	1	0	$14 \times 5V / 16$
1	1	1	1	$15 \times 5V / 16$

研究以上有关数据我们可以发现连续的数字输入有阶梯电压差, 而每一个电压差即为  $5V / 16$ , 即为由 0001 所得到的转换电压。

使用上述的加权电阻做成的 DAC 转换器有 2 个缺点:

1. 使用了七个电阻, 以现今的高科技 IC 制成, 仍不太容易做出这些非常精密的电阻。因为一般的 IC 制作中, 不容易控制每一个参数。
2. 而且 LSB 所使用的电阻为 MSB 的  $2^7$  倍 (以 8 位而言), 更增加了 IC 制作的难度。

一般而言, 部分的 DAC 使用 R-2R 阶梯网络式 DAC。这种 DAC 超出本书的范围, 本书主要讨论 DAC 和微型处理机的接口, 所以不加详述。

在本节中提及到了  $a_3, a_2, a_1, a_0$  等四个信号, 如果这些信号由微型处理机提供, DAC 就得了它的输出电压。

如果 DAC 的位数愈多, 它的分辨率将更高, 以表 23.1 为例:

表 23.1 DAC 的位数与分辨率

位数	分辨率
4	14.296%
6	6.667%
8	1.587%
10	0.392%
12	0.0244%
14	0.0061%
16	0.00153%

位数增加一倍时，分辨率增加将近 10 倍。

## 23.2 DAC-08 简介

我们以 DAC-08 当成 DAC 的说明，它是一个 8 位的 DAC，有很多厂商生产。

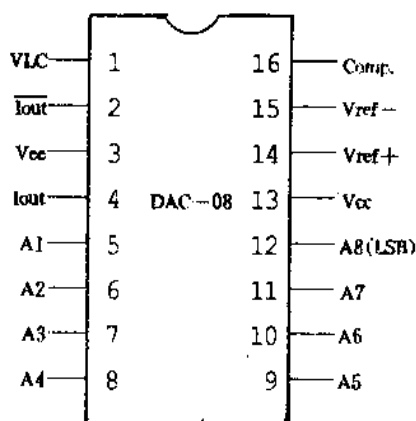


图 23.2 DAC-08 引脚图

信号说明:

VLC : 调整Threshold Voltage的基准电压输入。

$$V_{th} = VLC + 1.4V$$

$\overline{iout}, iout$  : DAC的输出。

$iout + \overline{iout} = \text{常量}$ ，当 $iout$ 增加时， $\overline{iout}$ 必定减少。

一般而言 $iout + \overline{iout} = \text{满刻度电流}$ 。

若 $R_f = 10K\Omega$ ，满刻度电流为 $1mA$ ，转换电压 $= 1mA \times 10K\Omega = 10V$ 。

$V_{cc}$  : DAC-08的负电源输入，它的范围为 $-4.5V > V_{cc} > -18V$

A1~A8 : 数字数据输入位。

**COMP. :** 内部运算放大器的频率补偿引脚, 高频率时才使用。

Figure 1 shows the DAC-08 circuit diagram. It includes an 8255A PPI and a DAC-08. The 8255A is connected to a microprocessor (微处理器) via data bus (XD0-XD7), address bus (A0-A1), and control signals (IOW, IOR, CS, RESET). The DAC-08 is connected to the 8255A's PA0-PA7. The DAC-08 output (pin 4) is connected to an op-amp configured as a voltage follower, with a feedback resistor  $R_f = 5K$ . The op-amp output is  $V_A$ .

图 23.3 DAC 与微型处理机的接口

例如：8255A 送出逐渐增加的数据（最小为 0，最大为 255），Iout 即缓慢增加，只要将这电流改变为电压，即可以看到锯齿状的电压。如果稍加以调整即可出现正弦波、三角波等。微型计算机负责的部分只有传送数字数据给 DAC，DAC 根据它转换为模拟输出。当然要传送什么样的数据就由微型处理机决定了。

1. 编程三角波, 最高点为 5V.



2. 数字电视中的 DAC 有何功能? 您认为它是多少位的 DAC?
3. 猜猜看一个 8 位的 DAC 价格多少? 16 位的呢? 为什么有这么大的差别?
4. 如何产生方波?

**提示:** 8255A 送出数据如 0, 0, 0, 0, 5, 5, 5, 5, 0, 0, 0, 0

- ### 5.如何产生正弦波?

## 第二十四章 数字激光音响与数字电视

### 本章学习目的

1. 经过一连串的实验专题后, 本章将以浅谈的方式说明微型计算机在数字音响和数字电视中所扮演的角色。
2. 通过本章的学习, 将使您对数字音响和数字电视有了初步的概念, 进而希望您为未来的数字电视, 数字激光音响提出改进的意见。

### 本章内容

#### 24.1 数字音响

#### 24.2 数字电视

#### 24.3 数字音响和数字电视中有一部 IBM PC 吗

### 24.1 数字音响

#### 24.1.1 传统唱片

传统式的唱片是将声音的波连续录制在唱片的 V 型沟槽上, 再借由唱针顺着沟槽的磨擦, 将振动波转变为强弱电流, 再加以放大, 送入喇叭而让声音重现。由于唱片不断的播放, 以及磁头的磨损, 渐渐的声音已不再清晰, 这也就是唱片的缺点。

#### 24.1.2 CD 唱片

CD 唱片是将连续变化的声音分割成许多间隔, 再将这些切割的波形, 按其振幅大小转变成 0 或 1 的二进制数字信号。将这种信号数字化, 再予以存录的方法有许多优点。

1. 在存录到播放的阶段, 即使混合杂音或失真, 只要能辨别脉波的有无, 即可使原信号复原。
2. CD 唱片是靠激光读取, 基本上不接触, 所以只有 0 与 1 的数据可以顺利读出, 不必担心有任何失真。

##### 24.1.2.1 PCM

PCM 为脉冲码调制 (Pulse Code Modulation) 的简称, 亦即将声波转变成脉冲的电信号。PCM 属于数字录音。它是将原有信号数字化, 并按一定的方式传送和复制, 由于“0”与“1”是单纯的形状, 因此不容易变形, 当有杂音成份混入时, 也不会出现。

##### 24.1.2.2 PCM 录音过程

1. 低通滤波器 (Lowpass Filtering)

将声音利用麦克风而转换成电波信号, 再经过低通滤波器除去高频部分。

2. 采样 (Sampling)

每隔一小段时间 (愈小愈好) 读取模拟信号值, 这个过程称为采样。采样的频率愈

高，准确度也愈高。采样频率至少必须为所要读取模拟信号的 2 倍，对于音乐而言，它的频率低于 20KHz,所以 EIAJ (国际标准) 所定的采样频率为 44.056KHz.

### 3.量化 (Quantization)

将采样所得的各数值给予一数值。图 24.1 为一音波的量化情形。纵座标为振幅，横座标为采样点。

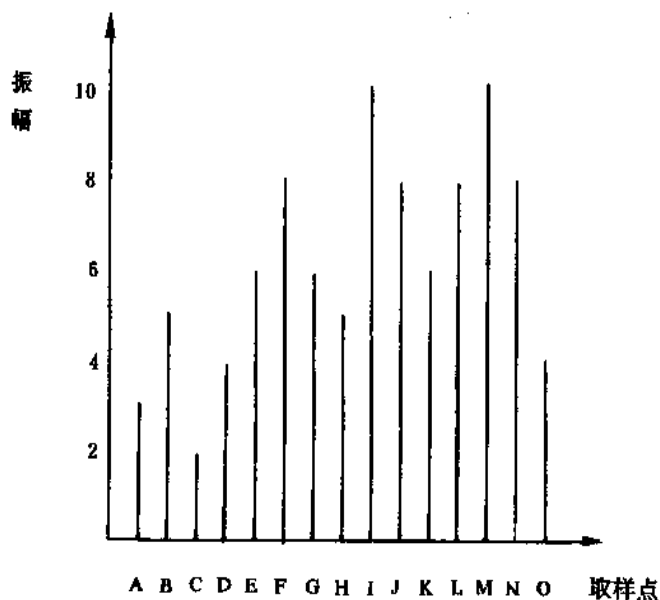


图 24.1 音波的量化

表 24.1 音波的编号

采样点 A 的二进位码为	0011
采样点 B 的二进位码为	0101
采样点 C 的二进位码为	0010
采样点 D 的二进位码为	0100
采样点 E 的二进位码为	0110
采样点 F 的二进位码为	1000
采样点 G 的二进位码为	0110
采样点 H 的二进位码为	0101
采样点 I 的二进位码为	1010
采样点 J 的二进位码为	1000
采样点 K 的二进位码为	0110
采样点 L 的二进位码为	1000
采样点 M 的二进位码为	1010
采样点 N 的二进位码为	1000
采样点 O 的二进位码为	0100

#### 4. 编号

在量化之后，每一个振幅给予一个数值，此值以二进制表示，如表 24.1。

PCM 的录音过程与传统的录音过程最大的不同，即是 PCM 录音过程中有数字化的程序，再将此 0 与 1 的信号存在 CD 唱盘上。图 24.2 为模拟波形和数字波形的比较。

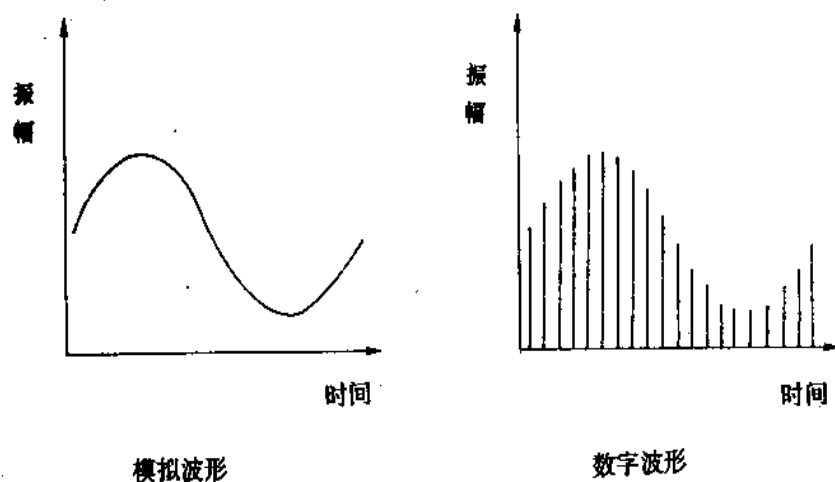


图 24.2 模拟波形和数字波形的比较

#### 24.1.3 数字音响与微型计算机的关系

量化之后的信号变成了数字的“0”、“1”信号，这些信号可存于 CD 唱盘上或者经过 D/A 变成模拟的信号由喇叭输出。微型计算机主要的功能在于唱机的各项控制，譬如：自动选曲、自动入片、播放的自动程序以及唱机的工作状态、各种显示的指示灯或数字显示。

### 24.2 数字电视

数字电视和数字音响的数字化原理基本相同，最主要的差别，在于数字电视最终目标是将处理完的信号分成 RGB 三色送给屏幕的电子枪。而数字音响只涉及声音的控制。

#### 24.2.1 画面处理

传统的电视由于不容易将模拟的信号处理，所以一些母子画面、暂停画面、将特殊镜头的放大等功能不容易达到。而在数字电视上，这些功能均非常容易通过数据的处理而达到。

类似于数字音响的 PCM 录音过程，影像的模拟信号经过量化之后，会产生非常多的数字信号，平常这些信号直接输出到屏幕。当有屏幕功能要求时，微型处理机会将这些数字信号加以计算，再按用户定义的方式输出。一般先进的屏幕功能如下：

1. 母子画面、看某台、另可设置一小窗看另一台。
2. 屏幕静止，可将所喜欢的镜头静止。
3. 主题放大，可将所喜欢的主题放大。



### 24.2.2 数字电视与微型处理机的关系

前一节中曾讲到了数字电视的一些特殊效果。这些效果都必须借助于非常快速的微型处理机加以处理。对于屏幕所使用的微型处理机，速度非常的重要。除此之外，现在一般传统电视已拥有的功能如“On Screen Display”可将现在的音量大小、颜色对比、选台、存储器中的台号选择以及各个频道的相关数据显示在屏幕上。这种功能也是靠微型处理机处理。

### 24.3 数字音响和数字电视中有一部 IBM PC 吗

我们常听到，那部电视、音响、冷气机、微波炉、汽车等是微型计算机控制。这些电器里面真有一部 IBM PC 吗？对此问题，我们必须先对 IBM PC 及微型计算机作一比较：

IBM PC 包含下列几个部分：

1. 中央处理单元 (CPU) 如 80386, 80486
2. 外部 RAM (不和 CPU 在同一个 IC 上)
3. 外部 ROM (不和 CPU 在同一个 IC 上)
4. 驱动器
5. 屏幕
6. 打印机
7. 计算机键盘

以上所列统称 PC 系统。如果我们的音响是微计算机处理，我们须要将计算机键盘放入音响吗？我们须要将计算机屏幕放入音响吗？答案当然是否定的。我们只须将中央处理机放入音响即可。一般而言中央处理机 (CPU) 并不能单独运行，必须借助于 RAM 和 ROM，其中 RAM 用于数据计算后的寄存处，ROM 则存放系统程序。如果您的汽车是微型计算机控制，当您启动之后，您汽车里的微型计算机 (CPU, RAM, ROM) 中的 ROM 会指示 CPU，叫它检查刹车系统、引擎、油门.....，如何检查呢？CPU 只是一个大脑，没有手、没有脚的大脑并不能做什么事，它只能发号命令，真正在动作的则是接口，这些接口如 8255A 接受微型处理机的命令行事。

所以微型计算机必须包含下列单元：

1. CPU
2. RAM
3. ROM
4. 接口

Intel 8048 是一个使用非常广泛的单片机，它包含了 CPU, RAM, ROM，它是一个 IC，由于拥有了计算机系统的结构，所以可以称为是麻雀虽小却五脏俱全。

读者在下次看到“微型计算机控制时”可以知道，它里面可能有一个单片机 IC。

## 第二十五章 电话保密与数字通讯

### 本章学习目的

对电话保密的原理加以认识。

### 本章内容

- 25.1 模拟通讯
- 25.2 数字通讯
- 25.3 信号的数字化与传送
- 25.4 电话保密
- 25.5 思考题

或许您是一位重要人物，或许您一直在怀疑电话被窃听，或许您主持一项国际研究计划，所以要提防工业间谍。您要如何增强电话的保密性呢？我们在本章将探讨如何用微型计算机达到电话保密的方法。如何将我们正常的声音转变为叽叽声，而如何让听电话的友人听懂我们的叽叽声？

### 25.1 模拟通讯

现今我们的电话系统均靠模拟信号传播，偶而电话声清晰，大部分则含有杂音。



图 25.1 理想通讯波

图 25.1 为我们所期望的通讯波，往往由于电话线品质或者电话机问题，我们常听到如图 25.2 的波形，电话中夹杂一些杂音。

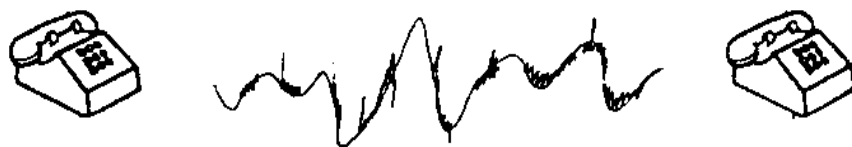


图 25.2 实际通讯波

## 25.2 数字通讯

数字通讯即靠数字信号传递信息如图 25.3.

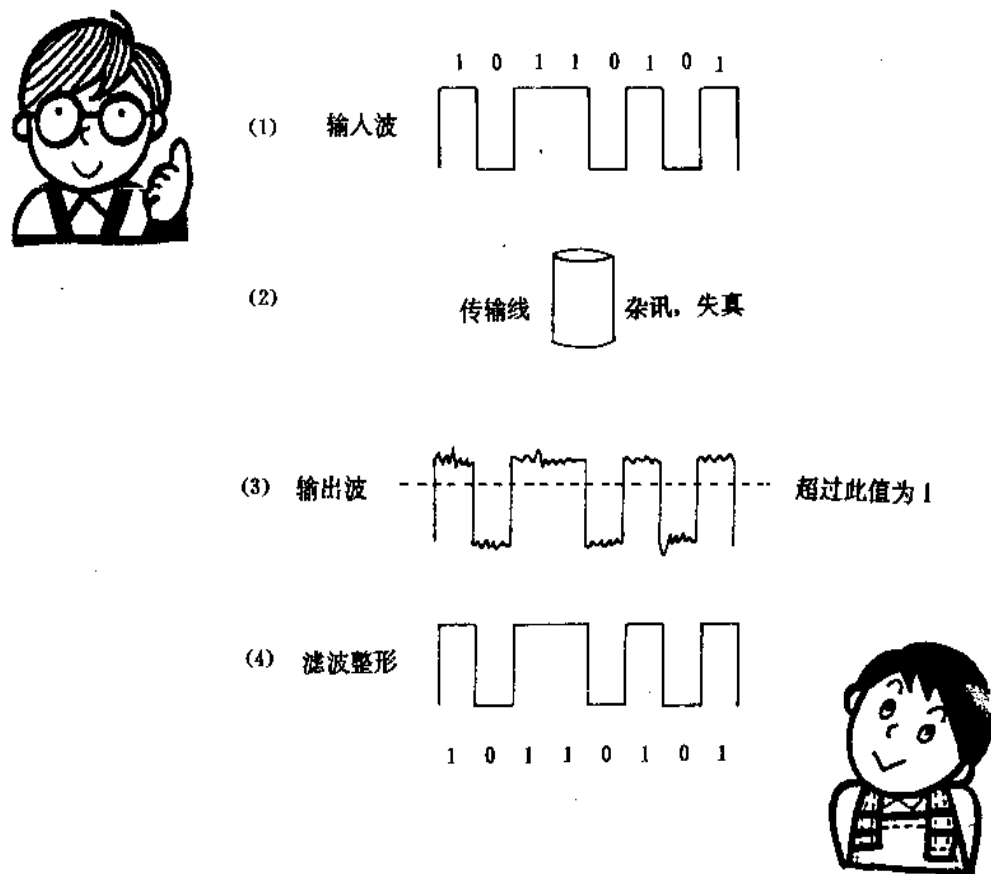


图 25.3 数字通讯

发出信号者送出一个完整的逻辑信号如 (1), 经过电话线造成了失真如 (2) 以及夹杂了杂讯等, 到了对方之后波形变的杂乱如 (3), 此时经过了脉冲整形, 又变回了漂亮的波如 (4)。

数字通讯的通讯方式, 即是将数字化的信息经过电话线传递, 到了对方之后再经过脉冲整形即可恢复原有的波形。此种传输方式可提供准确、高品质、低错误的通讯。

## 25.3 信号的数字化与传送

在传递信号之前, 必须将原始信号改变为数字, 再予以传送, 它包含 4 个步骤如下:

### 1. 采样 (Sampling)

大自然的信号几乎都是随着时间而连续变化的模拟信号。这种信号不容易作运算处

理，所以必须将它转换成数字通讯所能处理的信号。为了将此连续性的信号转换成数字信号，我们必须在一定时间周期内抽样，我们称此动作为采样。常用的采样定理如下：频率低于  $f_0$  的模拟信号以  $2f_0$  的采样信号予以采样并加以处理，则可由该采样值再复原为原有信号。此时它的采样间隔为“倪奎斯特间隔” (Nyquist Interval)

图 25.4 为著名的采样定理

- (1) 为原有信号
- (2) 经过采样点，取出采样信号如 (3)
- (3) 取样信号
- (4) 经过滤波、复原，原始信号重现

## 2. 量化 (Quantization)

随着时间而改变的模拟信号经过采样之后，必须用数字化代表它。这种过程即称为量化。最简易的作法即是利用 ADC 模拟数字转换器。

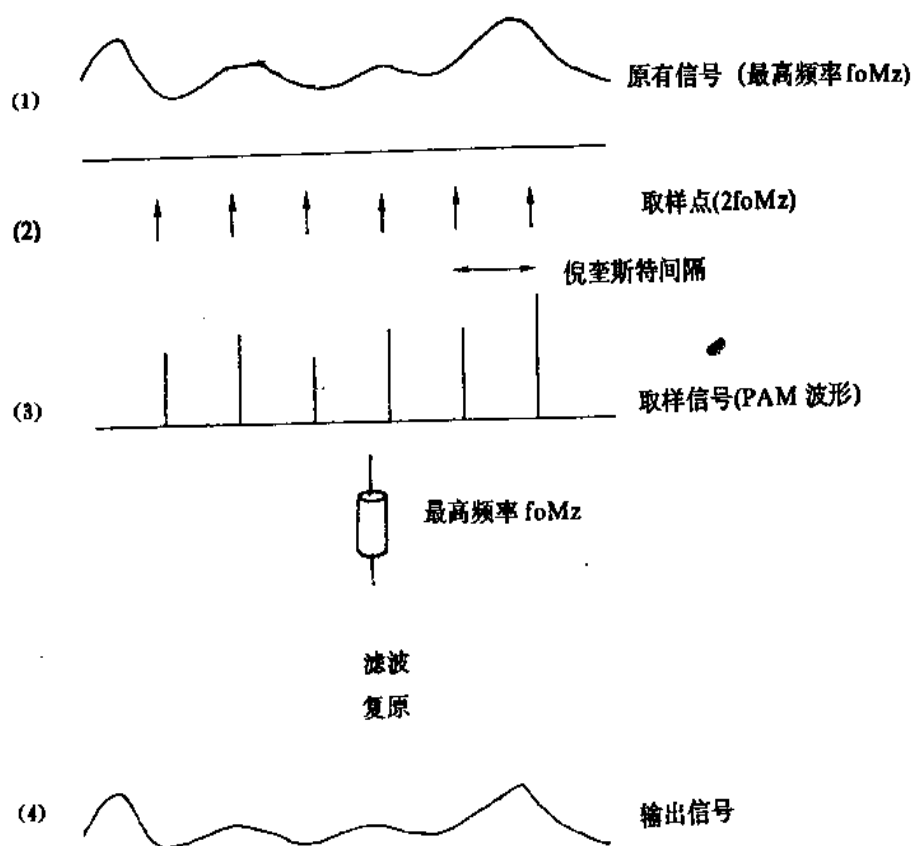


图 25.4 采样定理

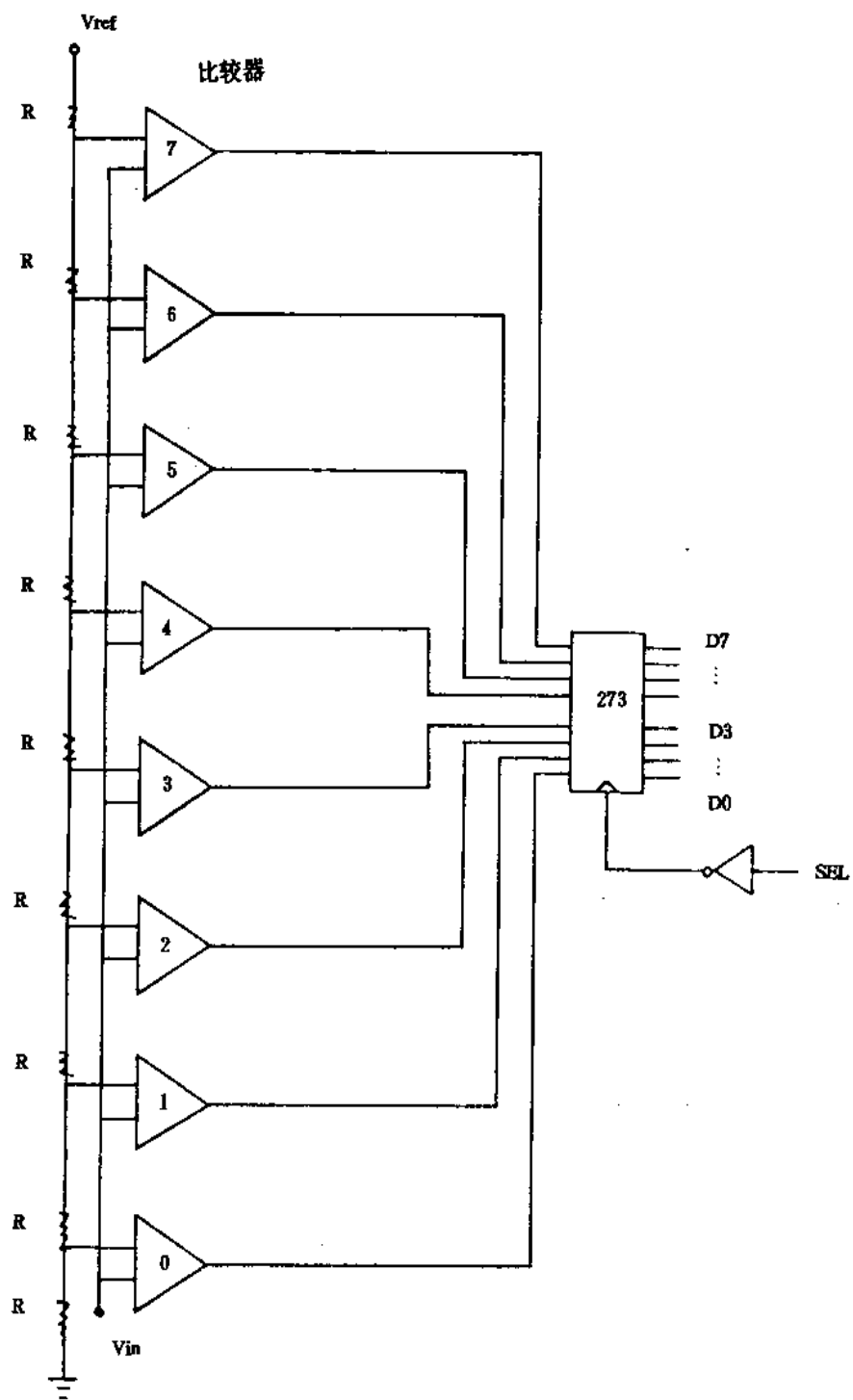


图 25.5 ADC 比较器

### 3. 数字传送

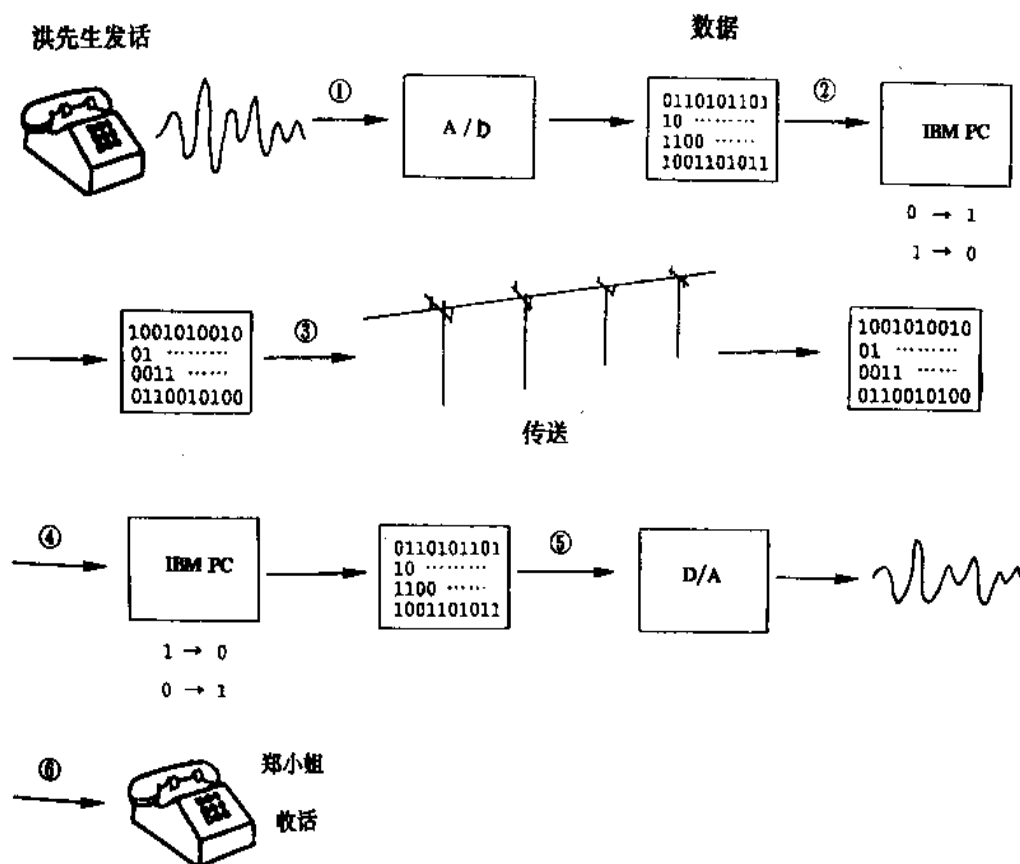
在采样、量化之后，我们得到了一些数字数据。参考图 25.3。我们可以通过数字通讯，将这些数字数据传递至远方。

#### 4. 再生

当收到这些数字数据后，利用 DAC(Digital Analog Converter)数字模拟转换器将这些数字信息转变成模拟信号。

### 25.4 电话保密

电话保密方式以下图流程解释它。



0. 发话者洪先生送出声音，此声音为模拟信号。

1. 此模拟信号经过 A/D 转换成数字。

2. 经过 IBM PC 处理，将“0”转为“1”，再将“1”转为“0”。

3. 再将处理过的数据送出。收话者收到的即为此处理过的数据。

4. 收话者将此数据送入 IBM PC 将“0”转为“1”，再将“1”转为“0”。此时原始的数据即复原。

5. 将此数据送入 D/A 转换器，可以得到原来的模拟信号。

6. 送入电话机，收话者郑小姐可以很清晰地收到洪先生的谈话。

讨论：

1. 在 2-4 过程中，若数据为外人所截取然后送入电话机，由于原数据已被处理过

了，所以听不出所以然来。

2. 在此流程中，IBM PC 所扮演的角色只是很简单的将 0 变为 1，1 变为 0。此时须要 IBM PC 吗？IBM PC 此时确实是奢侈了些，但是如果我们增加保密程度如将 4 位的数据按不规则转换，收话者再将数据变回如下：

0011-0101	0101-0011
1010-1001	1001-1010
---16 种不规则	---
...	...
发话者	收话者

此时 IBM PC 所扮演的角色也跟随着重要起来。

3. 除了 IBM PC 外，是否有其它电子装置可以达到这种功能？事实上，单一的复合电路（Combination Circuit）即可处理，或者单片机如 Z80，8048，6502 均可以妥善地处理这一类问题。但是如果将数据存储在驱动器内，或作更快、更复杂的数据处理，计算机的应用是必然的。

## 25.5 思考题

请把此原理当成专题研究，并在硬件上验证它。

### 小知识

语音 IC：当我们了解了数字信号也可以存储声音之后，我们不禁要问：“是不是所有的录音带产品均会被语音 IC 取代？譬如：电话答录机，它每次使用时，都必须机械式的倒带，使用久了，电机皮带松了，录音带品质也变了，针对这种不便，为什么不用一个 IC 取代。

于是语音 IC 就开发出来了，目前简单的语音 IC 可以存录一些声音，但是由于它须要很大的存储器来存储这些声音转变成 0，1 信息，几百万位的存储器只能存几秒钟的声音，因此电话答录机一直无法由 IC 取代。

数字录音机：为利用语音 IC 和数字录音带结合的产品，它仍使用录音带，因为录音带（事实上它是计算机磁带）可以存储大量的信息，飞利浦公司已经正式宣布开发成功，它即将优美的旋律以低廉的价格带我们的视听享受进入另一个空间。

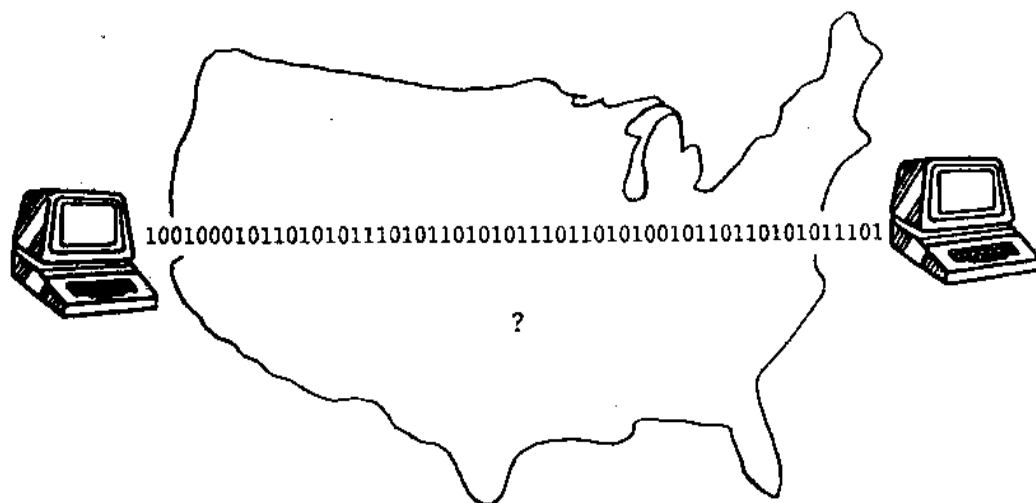
## 第二十六章 数据通讯的接口技术

### 本章学习目的

1. 了解 8255A 的通讯技巧
2. 了解调制解调器的基本工作方式
3. 了解 RS-232C

### 本章内容

- 26.1 计算机通讯方式
- 26.2 调制解调器(Modem)
- 26.3 创建通讯规则



当您的计算机知识丰富之后，您也许会想知道两部计算机如何“交谈”的。如何在台湾立即可以向美国的大学图书馆查询数据？这个问题可以分为两个方面解释。

1. 计算机硬件含外设装置及调制解调器。
2. 电信网络。

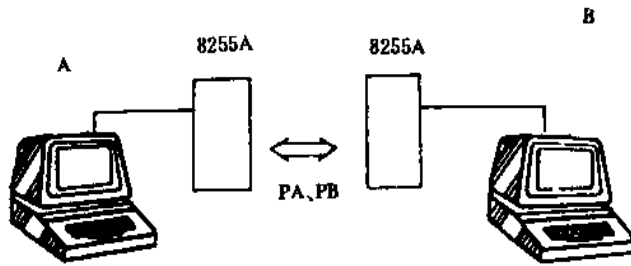
其中计算机硬件是我们要在这一章中讨论的。

### 26.1 计算机通讯方式

#### 26.1.1 8255A 模式

利用 2 片 I/O 适配器上的 8255A 将计算机连在一起。A 计算机的 PA, PB 分别接上 B 计算机的 PA 及 PB。





#### 工作方式:

1. A 通过 PB 送出一个特殊码给 B, 通知要送数据。
2. B 收到后, 回复一个响应信号于 PB, 表示知道了。
3. A 从 PB 收到回复信号。
4. A 利用 PA 将第一个数据送出。
5. B 收到后, 回复一个响应信号。
6. A 收到从 B 送出的响应信号。
7. A 送出第二个数据。

...

反复不断, 直至数据送完。

#### 讨论:

这种纯粹以软件作为交谈的系统有下列问题:

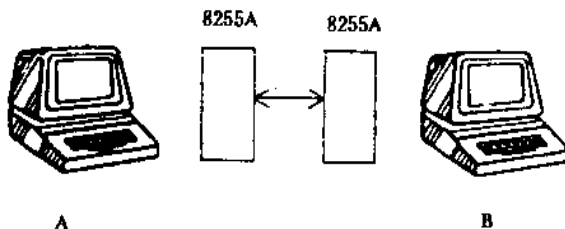
1. A和B两部计算机, 当一部在工作时, 另一部则在作循环等待, 浪费了系统时间。
2. A和B必须先清楚响应信号和数据的表示方式, 以免发生误会。

例如: 可定义响应信号为 10101010。数据信号的第一位为 1, 譬如 1×××××××  
×, 这样数据最大值只可以为  $2^7 = 128$ 。

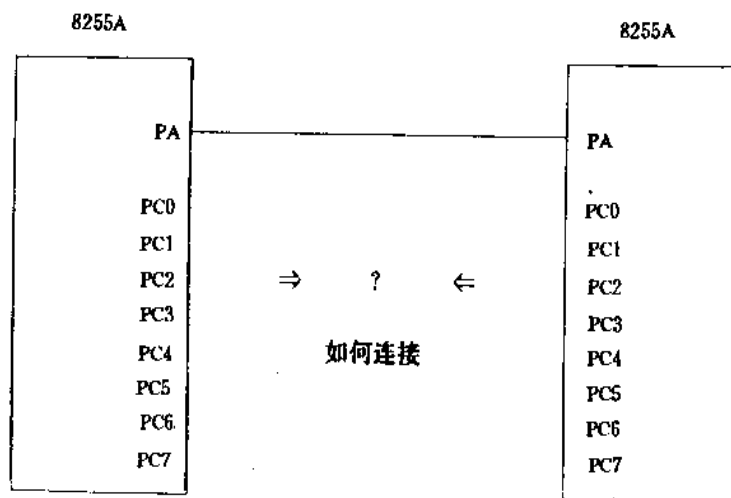
3. 当系统送出 10101010 时, 如何让另外一部计算机判别这个到底是数据还是响应信号?

请设计一套软件, 并运行它。您将发现一些其它问题。

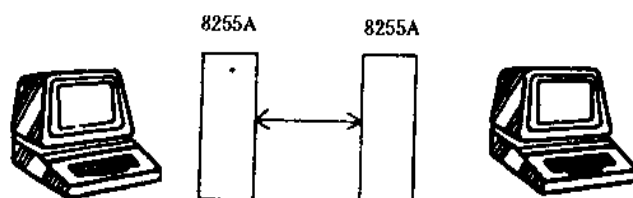
#### 26.1.2 8255A 模式 1



利用 8255A 第一种模式将两部计算机接在一起。参考第三章。连接的方式有很多种, 每一种连接方式都有不同的软件处理。请读者研究, 并说明所遇见的困难以及处理过程。



### 26.1.3 8255A 模式 2



利用 8255A 第二种工作模式。请读者将此当成专题处理，编程。

## 26.2 调制解调器 (Modem)

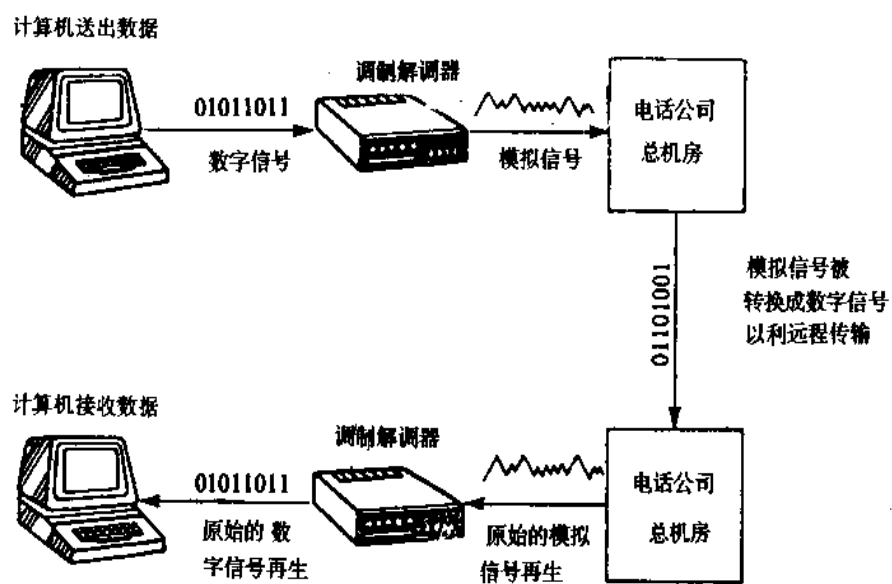


图 26.1 调制解调器通讯

一种被广泛使用的通讯产品——调制解调器。它可以通过电话线将两部计算机连接在一起。计算机利用 RS-232C 接头与调制解调器相接，调制解调器主要的目的是将数字信号变成模拟信号，再送至电话公司。接收时则相反，将模拟信号转为数字信号，由计算机接收。

市面上已经有不少的调制解调器，请选择一部装在您的计算机上，与其他人共同“试”一次。注意：大多数的调制解调器都有许多变通的选择，它在使用之前都须经过“跳线”（Jumper），以免因为这种错误而造成调制解调器不工作。

#### RS-232C:

目前用的数据通讯接头其设计是为了符合美国电子工业协会（EIA）的 RS-232C 标准。如图 26.2。

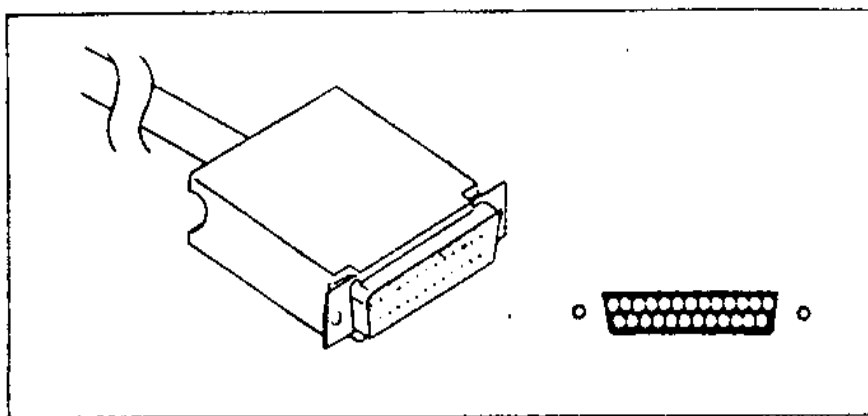


图 26.2 RS-232C

RS-232C 共有 25 支脚，其中 22 支脚有既定的用途，是否使用则由外设或计算机制造商决定。RS-232C 有三条重要的线：

- a. 发送数据线
- b. 接收数据线
- c. 信号接地线

RS-232C 有两种稍微不同的型式，其一是给数据通讯设备（DCE），另一给数据终端设备（DTE）。通常我们可能会在第二脚及第三脚碰到一些问题，因为 DTE 在第二脚发送数据，而 DCE 则是在第三脚接受数据。

### 26.3 创建通讯规则

前一节中我们讨论了如何利用 8255A 作为通讯的 IC。在作这件事之前，我们必须定义一些信号，以告诉对方哪一个信号代表数据，哪一个信号代表确认等等。在国际通讯中，这一类的定义很早就规定好了，以免发生“讲不通”的情形，譬如有 Bisync 及 SDLC。

## 第二十七章 步进电机的应用

### 本章学习目的

- 了解步进电机
- 了解步进电机与微处理器的接口
- 了解步进电机的应用, 传真机、X-Y 绘图机、打印机、软盘驱动器

### 本章内容

- 27.1 步进电机简介
- 27.2 步进电机与微处理器的接口
- 27.3 步进电机的应用实例

### 27.1 步进电机简介

对于步进电机, 它的基本原理和电机类似。图 27.1 为步进电机的基本线路及线圈。

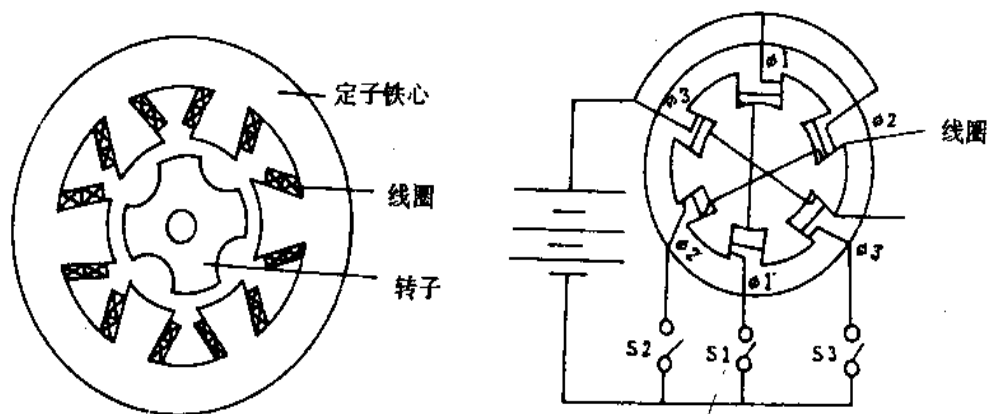


图 27.1 VR 型步进电机的转子、定子铁心、线圈的关系

步进电机的线圈称为相。一般步进电机由 3~4 (或更多) 的线圈组成, 如图 27.1 为三相式的步进电机。当步进电机要启动时, 即须由这些线圈输入一串脉冲完成, 这些相即由  $S_1$ ,  $S_2$ ,  $S_3$  三个开关控制。

在未输入电流之前,  $\phi_1$ ,  $\phi_2$ ,  $\phi_3$  为 N 极,  $\phi_1'$ ,  $\phi_2'$ ,  $\phi_3'$  为 S 极。当  $\phi_1 \sim \phi_1'$  通过电流之后, 图 27.2 即为转子的安定位置。

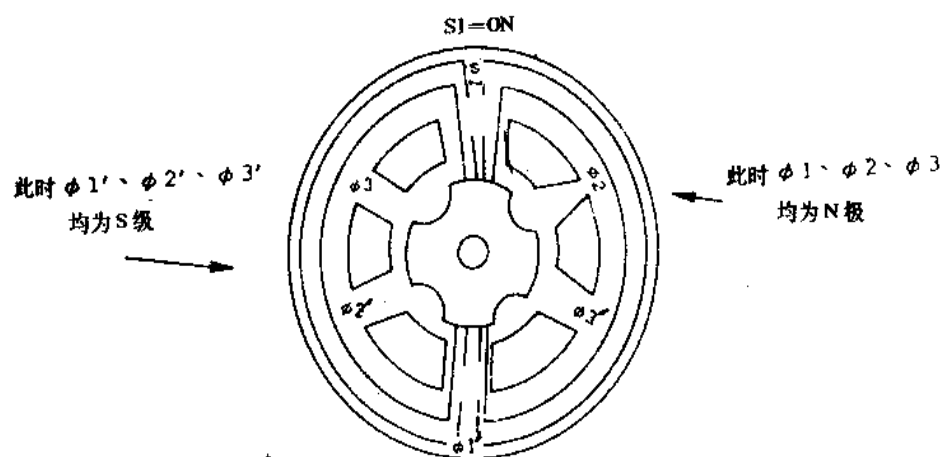


图 27.2  $\phi_1 \sim \phi_1'$  相激磁时，转子的安定位置

如果此时  $\phi_2 \sim \phi_2'$  流入电流，则由于磁力线的偏转而产生了旋转力，如图 27.3。

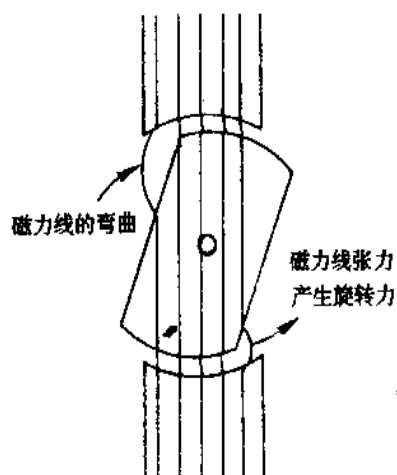
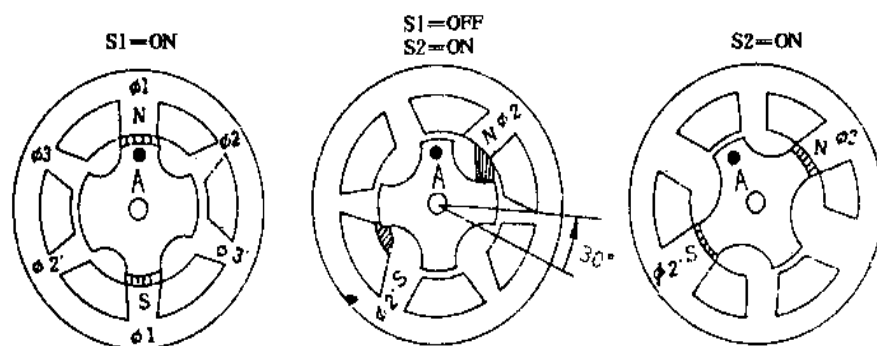


图 27.3 磁力线的弯曲产生旋转力·(转矩)

当我们要使步进电机转动时，我们必须操作  $S1 \sim S3$  三个开关。如图 27.4。



(1)  $\phi_1 \sim \phi_1'$  激磁的安定位置 (2)  $\phi_2 \sim \phi_2'$  激磁， $\phi_1 \sim \phi_1'$  切断 (3) 转子的 CCW 方向 1 步级 ( $30^\circ$ )

图 27.4 由  $\phi_1 \sim \phi_1'$  向  $\phi_2 \sim \phi_2'$  转动时会如何的步进

图 27.4 说明如下:

(1)  $\Phi 1 \sim \Phi 1'$  流通电流, (又称  $\Phi 1 \sim \Phi 1'$  激磁) 达到安定位置。

(2)  $\Phi 2 \sim \Phi 2'$  流通电流,  $\Phi 1 \sim \Phi 1'$  的电流则关掉, 此时磁力线张力产生旋转力将向反时针方向转  $30^\circ$ , 如图(3)。

(3) 达到它的安定状态, 转子转了  $30^\circ$ 。

利用以上的原理, 按 S1, S2, S3, S1 ON 的顺序即可让步进电机转  $90^\circ$ , 按图 27.5 所示:

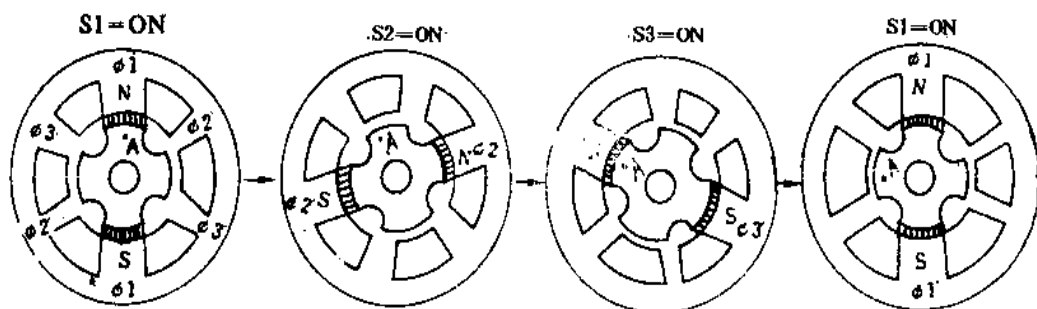


图 27.5 按激磁法的切换, 转子连续步进的情形, 每 3 步级旋转 1 个齿距

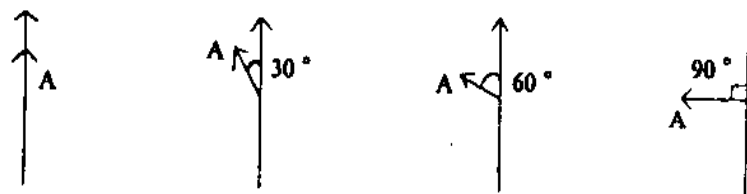


图 27.6 为图 27.5 转子连续转动的分析情形

## 27.2 步进电机与微处理器的接口

图 27.5、图 27.6 简述了转子连续转动的角度与三个开关 S1, S2, S3 的关系。如果我们要让步进电机反转一圈, 其步骤如表 27.1。

表 27.1 步进电机旋转一周的步骤

	S1	S2	S3
step1	1	0	0
step2	0	1	0
step3	0	0	1
step4	1	0	0
step5	0	1	0
step6	0	0	1
step7	1	0	0
step8	0	1	0

(续表)

	S1	S2	S3
step9	0	0	1
step10	1	0	0
step11	0	1	0
step12	0	0	1
step13	1	0	0

如果将以上步骤反向运行，则步进电机将正转一圈。

**驱动电路:**

图 27.7 为分配电路与激磁电路接线的方式,用以驱动步进电机。

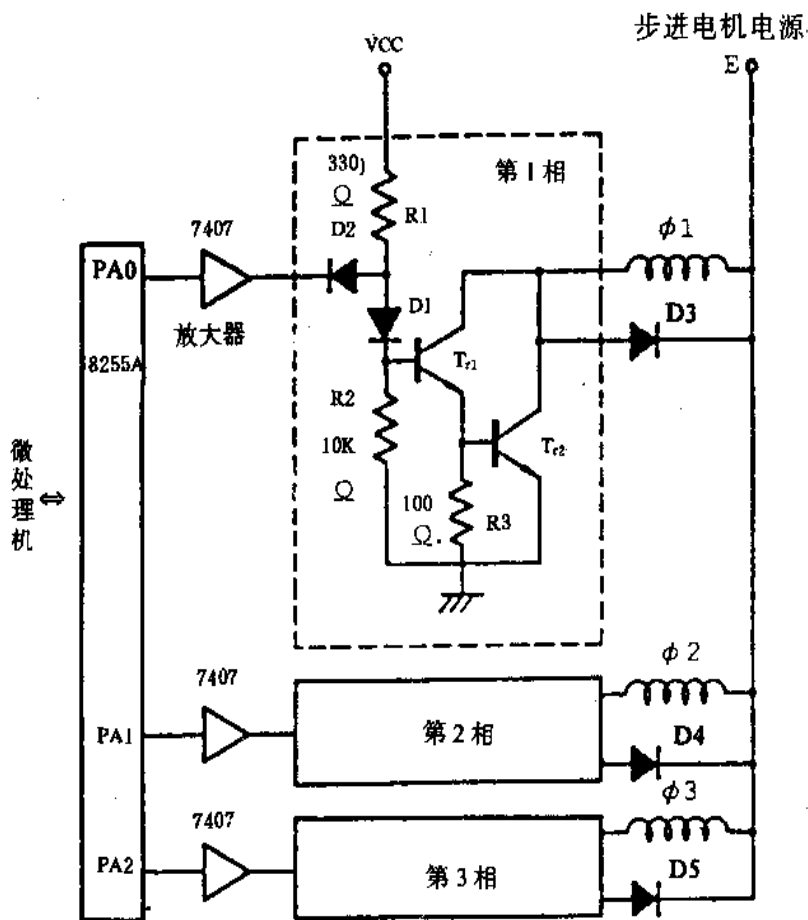


图 27.7 分配电路与激磁电路连线的一例

分配电路的信号送到激磁电路必须经过一个电流放大器如 7407。微处理器通过

8255A 的 PA0~PA2 将数据送出, 经过达林顿 (Darlington) 连接的晶体管驱动步进电机。

请读者设计一个软件以配合这个硬件。

工作目标:

步进电机必须正转 5 圈, 另反转 5 圈。转动时可以目测。

### 27.3 步进电机的应用实例

随着步进电机的发明, 以及电子零件的成熟发展, 它在计算机外设系统中已占了一个非常重要的角色。如下所述。

串行打印机

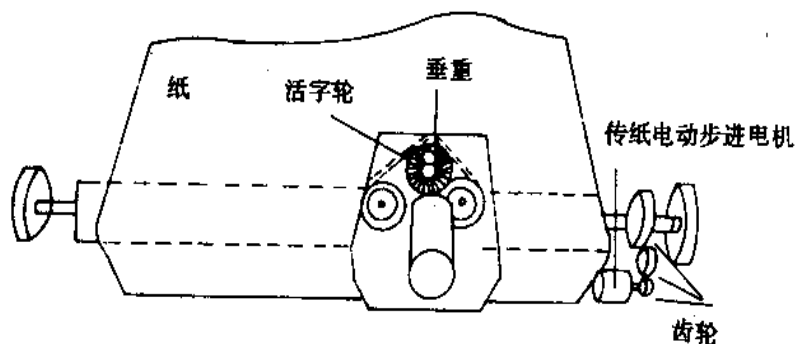


图 27.8 活字冲击型打印机的电动机驱动结构

串行打字机的步进电机将纸一格一格的往上滚, 这样它才可以一行行的打印字。

传真机:

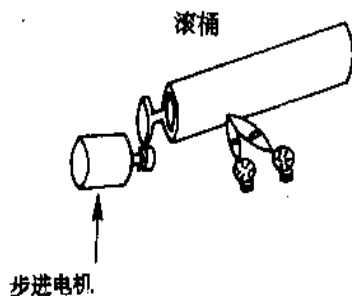


图 27.9 传真机原理

传真机的步进电机也是利用步进电机将纸一张一张的往上滚, 以利数据输出或接收。

X-Y 绘图机:

曲线或图表的打印驱动常使用步进电机, 用以控制笔尖在 X 轴及 Y 轴的转动。



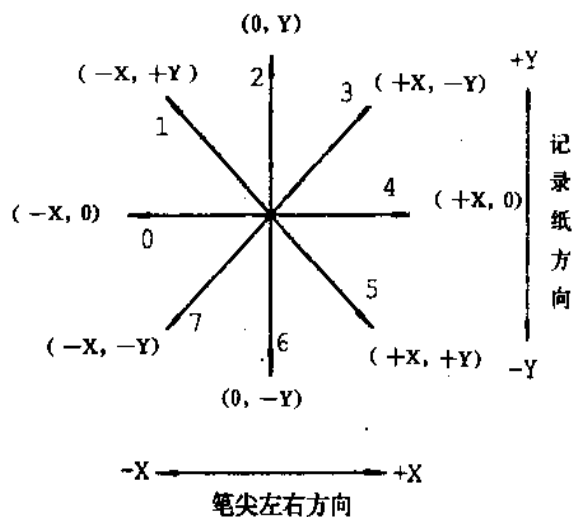


图 27.10 曲线描画器的笔尖运动方向

### 软盘驱动器

图 27.11 为软盘驱动器的构造与装置。

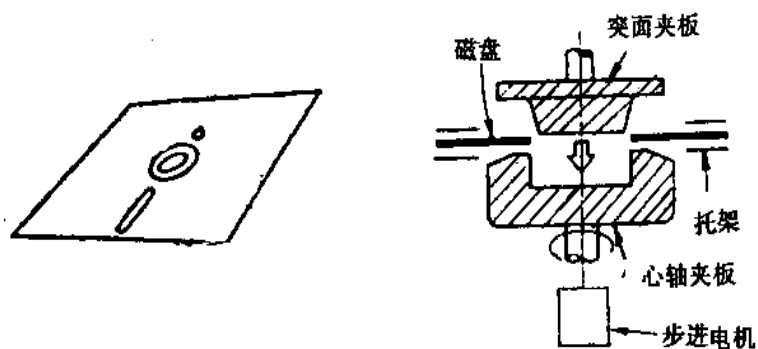


图 27.11 软盘驱动器的构造与装置结构

步进电机一步一步的旋转，寻找位置，用以存储数据或读回磁盘上的信息。

## 第二十八章 EPROM 编程器

### 本章学习目的

1. 了解 EPROM
2. 通过 8255A 的学习，以了解并制作一个 EPROM 编程器。

### 本章内容

- 28.1 EPROM 简介
- 28.2 操作模式
- 28.3 编程框图
- 28.4 EPROM 编程器的设计
- 28.5 EPROM 编程器的硬件设计
- 28.6 软件编程

### 28.1 EPROM 简介

EPROM 是可以清除的 ROM (Erasable Programmable Read Only Memory)。它的主要特色即是可以保存内部数据达十年之久。在任何时间均可以用紫外线清除所有内容，再重新编程。

一般的 ROM 存储器可以分为下列数种：

1. Mask ROM：在 IC 做好之后，数据已经编程于内部，所以数据不可修改。客户必须在 IC 做好之前提供这些数据给 IC 设计商或制造公司。这些数据存在一片掩模 (Photomask) 上，所以称为 Mask ROM。由于这些数据不能修改，客户必须保证数据的正确性。
2. EPROM：即本章所述及的，客户可以随时修改内容，而且不需 IC 设计公司或制造公司协助，即可在几秒钟之内完成。缺点为这类 IC 必须谨防暴露于紫外线下，以免数据流失，且单价高。
3. EEPROM (Electrical Erasable Programmable Read Only Memory) 与 EPROM 类似，数据可由电性改变。且可存十年以上。数据存储单元可以编程一万次以上，缺点为存储单元的面积较大，价格更贵。

EPROM 的存储单元为场效应晶体管，它包含了一个浮门极 (Floating Gate)。编程方式是在特定的 VPP 引脚上加上高电脉冲，将高能量的电子注入浮门极，这样可将所想要的数据写入存储器单元，也就是编程之意，若要清除则可以紫外线照射在透明的窗口上，因为紫外线的波长短、能量高，使 IC 内的电子动能增加，产生光电子流，让电子由浮门极返回 IC 基座 (Substrate) 达到清除数据的目的。由于所有的记忆元件均在窗口之下，所以紫外线照射时会对所有存储元件产生作用，因此在紫外线照射时，所有的数据均

会流失。平常窗口上均贴有一片锡泊纸，用以防止紫外线侵害。

VPP: 编程所需电压在数年前它的要求为 21V, 新的制造技术只要求 VPP 为 12V~13V, 如国际半导体的 NMC27C16B, 它的存储单元为 2K 字节。

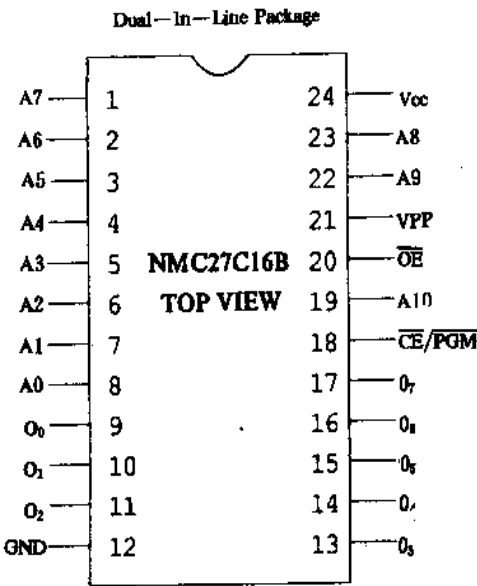


图 28.1 NMC27C16B 引脚图

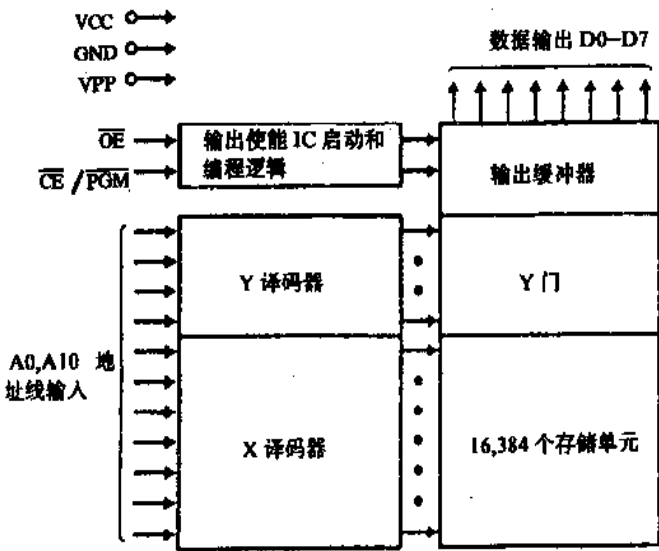


图 28.2 NMC27C16B 方块图

本章所讨论的 EPROM 为国际半导体的 NMC27C16B, 它的特色如下:

1. 快速存取时间, 350ns.
2. 低功率.
3. 完全和 CMOS 微处理机兼容.
4. 外部电源为 5V.



图 28.3 为 EPROM 读的时序图, 说明如下:

- ①地址线完成准备。
- ② $\overline{CE}$ (Chip Enable)变为低电平, 使能 EPROM。
- ③ $\overline{OE}$ (Output Enable)变为低电平, 使能 EPROM 输出。
- ④输出数据出现在输出总线。此输出为三态输出, 平常为了避免和系统不兼容, 所以被定义为 Hi-Z, 也就是高阻抗的意思。
- ⑤地址线,  $\overline{CE}$ ,  $\overline{OE}$ 输出此时在数据出现一段时间之后, 又重归无效状态。

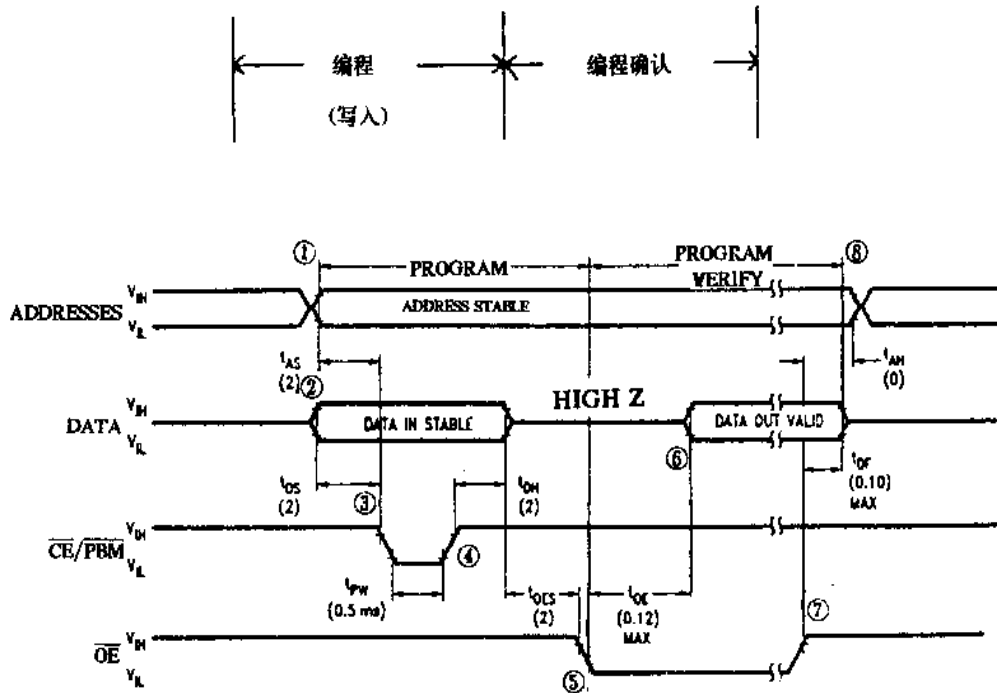


图 28.4 编程时序图

图 28.4 为编程时序图, 包含两部分:

1. 编程 (写入)
2. 编程确认

时序图的说明如下:

- ①地址线有效, 指示要被编程的位。
- ②编程数据必须稳定的在数据位上出现。
- ③此时 $\overline{CE}$ (chip enable)及 $\overline{PGM}$ (program)有效, 达0.5ms之久, 这样才可能确认高能量的电子注入浮门极可以达 10 年之久。
- ④0.5ms后 $\overline{CE}$ 及 $\overline{PGM}$ 恢复高电平。此时编程完成, 以下为编程确认。
- ⑤此时 $\overline{OE}$ (output enable)有效, 用以确认刚才的编程正确。
- ⑥ $\overline{OE}$ 有效的  $t_{OE}$  时间后, 数据从 EPROM 读出, 用以对比刚才所写入的数据。
- ⑦ $\overline{OE}$ 恢复高电平。
- ⑧编程确认结束, 地址线转向另一地址继续编程或结束。

### 28.3 编程框图

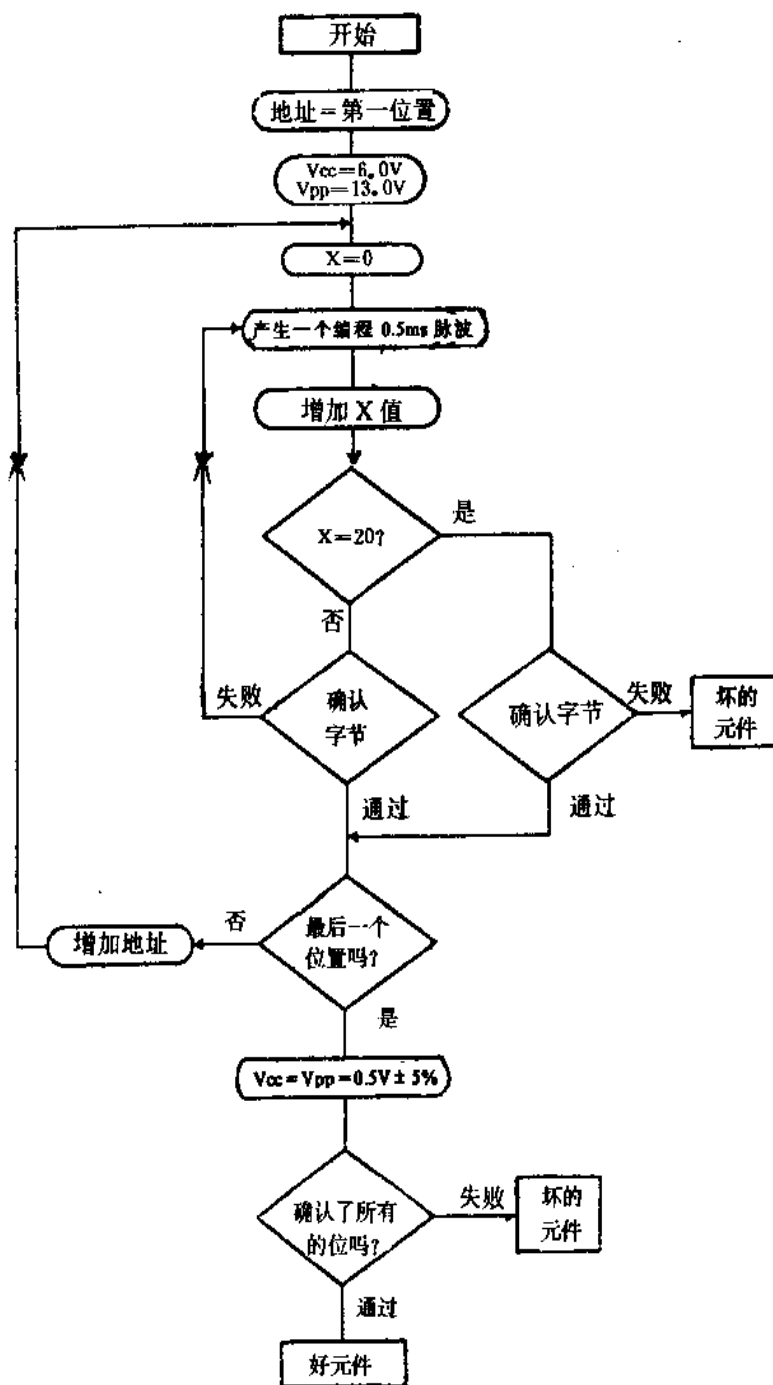


图 28.5 NMC27C16B 的编程框图

## 28.4 EPROM 编程器的设计

EPROM 编程器最主要可分为四个部分:

1. 提供地址线给 EPROM, 用以选择 EPROM 的字节位置.
2. 提供数据位 (8 位) 总线, 用以传送编程器与 EPROM 的数据.
3. 提供编程所须要的控制信号:
  - a.  $\overline{CE}$ (Chip Enable)
  - b.  $\overline{OE}$ (Output Enable)
  - c.  $\overline{PGM}$ (program)
  - d. VPP (高电压) 的控制信号

### 4. VPP 高电压

以上四项中的 1~3 项可以由 8255A 来完成. 一般市售的 EPROM 编程器会另加上一些小型开关用以选择不同型号的 EPROM. 第 4 项 VPP 必须由外界提供, 控制线路如图 28.6.

这个编程使用了一个 8255A 及几个晶体管, 8255A 提供地址线、数据线及控制线. 高电压控制部分是一个非常重要的部分. 当  $PC5=0$  时, 它提供了 13V 的高电压给 NM2716 的 VPP 输入点, 也就是提供高能量电子, 使它进入 EPROM 浮门极.

请读者按下列工作目标自行设计一个 EPROM 编程器.

工作目标:

这个编程器必须具备下列功能:

1. 检查 EPROM 的所有内容是否空白.
2. 读取 EPROM 的内容.
3. 比较两片同型的 EPROM 内容是否相同.
4. 将数据写入 EPROM.
5. 将数据从一个 IC 复制到另一个空白的 EPROM 上.

## 28.5 EPROM 编程器的硬件设计

见图 28.6.

## 28.6 软件编程

本章将不详细写出程序, 留给读者练习的机会.

软件编程的注意事项如下:

1. 正确编程 8255A. 定出 8255A 的 PORTA, PORTB 及 PORTC 的功能.
2. 参照时序图及所使用的 IBM PC 的时钟, 正确编程出每一个时间间隔(Timing)要求.

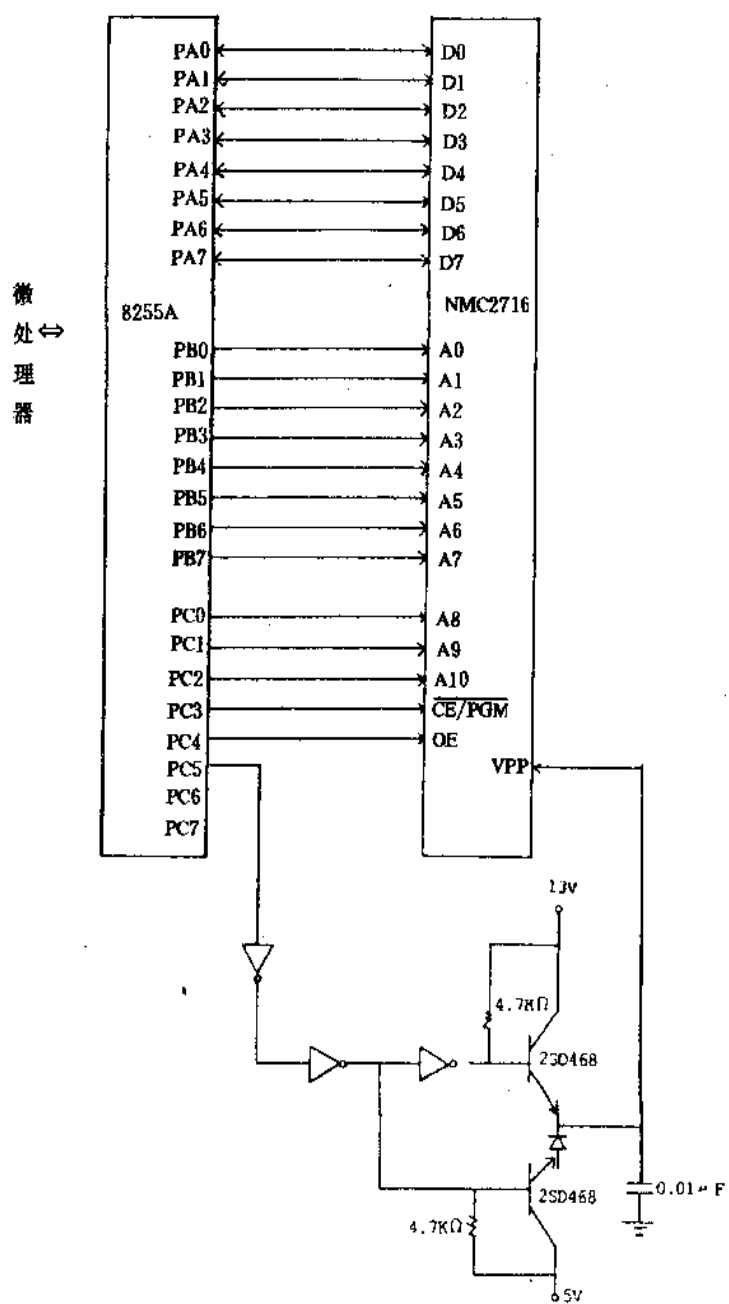


图 28.6 NM27C16 的编程器硬件



## 第二十九章 赌博性电动玩具与警察专题

### 本章学习目的

1. 8259A 的中断应用
2. 赌博的实质

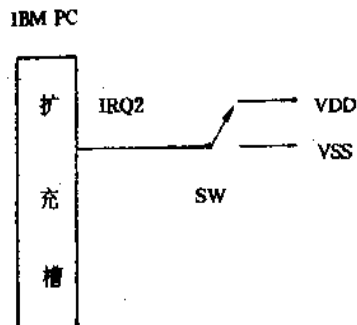
### 本章内容

- 29.1 工作目标
- 29.2 硬件设计
- 29.3 软件设计

#### 29.1 工作目标

平常 IBM PC 运行吃角子老虎程序，当 8259A 产生中断时，计算机会运行两只老虎的音乐，再次产生中断时，计算机又恢复先前的吃角子老虎程序。

#### 29.2 硬件设计



IBM PC 扩充槽平常在 VSS，当 SW 转换  $IRQ2 = VDD$  时，对 CPU 产生中断，平时 IBM PC 运行吃角子老虎程序，此时 CPU 运行另一程序。

#### 29.3 软件设计

利用磁盘上的吃角子老虎(game6.c)做出此程序。

十余年前，电动玩具企业如雨后春笋般地在各地成立，随着警察机关的取缔，这些赌博性电动玩具业者转入地下。利用计算机公司作为掩护，实际上则为赌场。

在一间店面里摆设着十余部计算机，每部计算机的硬件都有一个赌博式电动玩具程序，当客户光临时老板会安排赌法及逃避警察的方法，在一个赌客兴致浓厚时，如果警察

来了怎么办？通常这种计算机公司的门口，都有一个柜台，柜台下有一个开关，控制着每部计算机的中断，只要将开关关上，计算机立即被中断，此时计算机运行一些通常的软件程序，一群赌客个个变为计算机程序的高手。警察临检完毕，老板和赌客的纠纷怎么处理？老板在惊魂之后，再次打开开关产生一次中断，赌博性电动玩具又可继续运行，没有赌资的困扰。

随着学习计算机的人数增加，与计算机十赌十输的事实将赌客教乖了，这种计算机公司也就一家家的关门了。

## 第三十章 微型计算机的其它应用

### 本章学习目的

本章将以浅谈的方式让您了解我们生活的世界里还有哪些东西是以微型计算机控制的, 以及还有哪些东西可以让微型计算机替我们服务。

### 本章内容

- 30.1 为什么要用微型计算机
- 30.2 单片机与 IBM PC
- 30.3 机器人(I)(利用 IBM PC)
- 30.4 机器人(II)
- 30.5 火车站台自动控制
- 30.6 霹雳车与计算机
- 30.7 IC 卡

### 30.1 为什么要用微型计算机

也许读完本书, 您又有了新的质疑, “为什么要用微型计算机控制一部电梯”? “为什么要用微计算机做出一部自动售货机”? 如果您学过电子学, 您一定会说: “其实这些实验只须用几个 TTL 的逻辑电路就可以做出来了。”您是对的! 一般的电子应用可以使用两种方式完成:

1. 利用电子零件, 一个一个的拼凑出来。这种方式对于简单的电路非常有用, 若遇到复杂的电路, 您可能要搬逻辑分析仪、电表、示波器等实验仪器和满头大汗的思考, 才将目标完成。以经济效益而言, 若您使用了 10 个以上的 TTL 门, 您可能花在接线上不少时间, 另外成本可能在数百元至数千元之间才能把一个实验完成。
2. 利用软件, 也就是利用一个单片机, 您此时所需要的就是书写软件了。软件写完, 工程也完成了一大半。利用单片机所做的电子装置, 当然免不了还要加上一些缓冲区(Buffer)。由于微型计算机是顺序(Sequential)运行, 所以时序(Timing)对它而言并不是太复杂, 它的困难在于软件的编写。一个 80C51 单片机的批发价格在 1~2 美元之间, 一个 Eprom 的价格也在 3~4 美元之间。对于每一种特殊的单片机, 每一家公司都会提供一些辅助设计工具, 如 ICE 模拟开发工具等。

## 30.2 单片机与 IBM PC

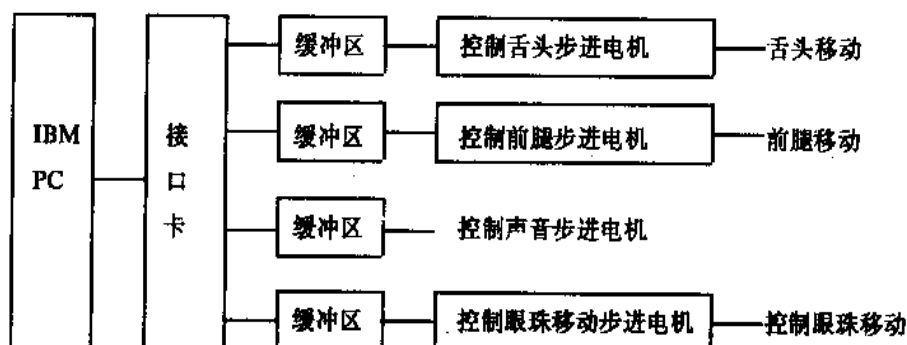
本书的学习重点在于 IBM PC 的接口应用。这些应用和单片机的工业应用（如 8048、8051、Z80）息息相关。本书所提及的实验几乎都可以在其它单片机上完成。唯一不同之处在于 IBM PC 可利用其它外设（如打印机、驱动器、屏幕）可以做更复杂、更先进的处理及运算。在本书序中曾提及：美国航天飞机内部的 PC 可将地球任何一角落的地理信息及时显示，这就非单片机可以取代的。

本书的专题结果都在计算机屏幕上及时显示出来，对于一个单片机而言，这类屏幕的处理却是非常困难。

## 30.3 机器人(I)(利用 IBM PC)

在瑞士 Lenzburg 的一个介绍中古时期的博物馆中，一只只原始时期的恐龙，对着观众发出怒吼，时而眼珠露出红光、绿光，时而一只前腿往前踢，巨长的舌头不时地吐出，十分逼真，让人觉得置身在古老的时代里，实际上它却是一个由 IBM PC 控制的“机器人”。

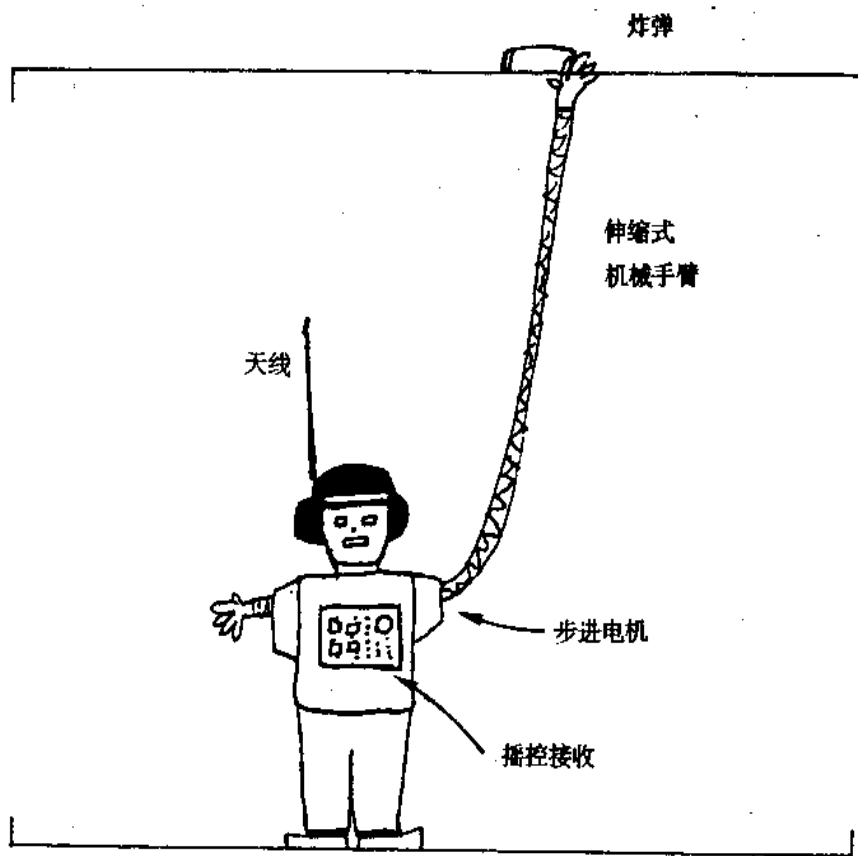
它的接口如下：



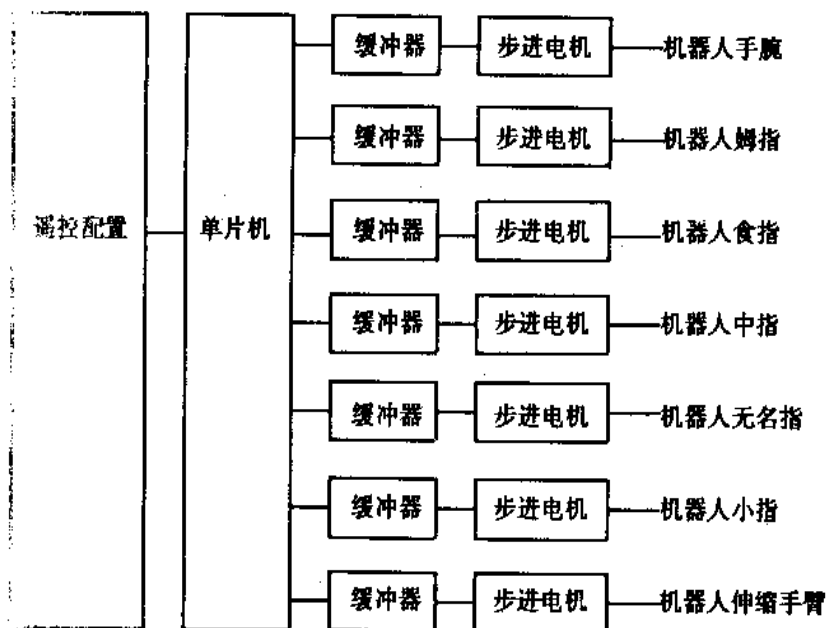
如果以上的恐龙不以 IBM PC 控制，而以单片机处理，则观众所见的将是千篇一律的周期性的表情。如果使用 PC 驱动器存储复杂的程序，这类的机械人可借助 PC 的协助做出各种更复杂的动作表演。

## 30.4 机器人(II)

1992 年 5 月份台湾麦当劳炸弹事件，让我们深感遗憾。如果当初去触碰炸弹的是机器人，而不是一位爆破员，我们损失的可能只是几件铁片和弹簧。这种爆破机器人应如何设计呢？



这种机器人的应用结构非常简单，它只是一个单片机和几个步进电机控制五双手指以及一个可伸缩性手臂即可。它的电子部分如下：



如果您稍加注意，可以发现这个线路并不需要适配器。为什么呢？因为单片机已经提

供几个口供驱动外界装置使用了。

为什么我们不使用 IBM PC 呢？当然可以，不过您希望您的机器人背着 5 公斤的计算机屏幕和 5 公斤重的 PC 以及打印机吗？

当今世界上最先进的机器人在美国，但是使用机器人最多的国家却是日本，日本的汽车工业须要大量的机器人做装配工作。

**思考题：**

焊接使用的机械手臂是由 IBM PC 还是由单片机控制？

让我们将最危险、最伤身的工作交给机器人吧！21 世纪的机器人将在您的构想中一一呈现，它须有以下功能：

1. 视觉 Video Cammera
2. 听觉 语言辨认系统
3. 发音 语言合成
4. 行动 步进电机或遥控装置
5. 思考 快速、周密的 CPU 计算

或许数十年后您的机器人会向您苦苦哀求，希望和女机器人约会、跳舞、抽烟时，你可别惊讶。不过机器人的“情感”理论基本到目前为止尚未发展成熟。

### 30.5 火车站台自动控制

也许您曾听说过，在美国一定要以车代步，为什么呢？道理很简单，您可否听过有人从纽约搭火车到旧金山？或从佛罗里达到芝加哥？答案是否定的，他们不是开车就是搭飞机到达目的地。反观欧洲大陆，却是以火车连接起来的陆地，铁路网密集且分散各地，任何一个城市都有火车经过，所以到欧洲自助旅行的人，只要凭一张地图，便可畅通无阻。夜晚 10 点以后，想搭夜快车的人常把火车站挤得水泄不通。在欧洲任何一个大城市的火车站都有 20~30 个站台。苏黎士火车站每天早上有二十几万外地人口进入这个城市工作，下班时每线火车要把这些人口送回家，而火车很少误时误事。因此您可以知道这个火车站的站台控制是何等重要。如果您驻足过苏黎士火车站，您将发现每一个站台的控制室中都有一部 PC，随时监督着每个站台火车往来的情形。如果有一天您负责编程一个 30 个站台的火车站管制，您应如何设计这一套系统？这个系统的基本要求如下：

1. 如何准确地引导火车进站、离站。
2. 配合灯号，时时防止火车互撞事件。
3. 如何应付突发事件，如果有一班国际火车严重误点，如何让这班火车在不影响其它列车的情况之下进站、离站。
4. 将火车站所有的火车进出站的电子显示装置也纳入管理。

**思考题：**

您是否曾经注意过 1992 年 5 月间，中正机场的通讯故障事件？凭您的想象力编程一座飞机场的塔台控制。

## 30.6 霹雳车与计算机

我们想像中的霹雳车的功能是：

1. 语言辨认。
2. 语言模拟，如警车声。
3. 具有切入其它计算机系统能力，扰乱其它计算机，如非法取得自动提款机的钞票。
4. 信息自动查询，随时可以提供当地地形、市街信息。
5. 自动驾驶，具有视觉系统（如果您拜访过好莱坞电影城，您将发现霹雳车里藏有另外一个人负责驾驶）。
6. 人工智能，有思考能力。
7. 影视通讯。

以上特色除了第3项不太可能在未来实现以及第6项不容易成熟之外，其余的部分应在不久之后可以完全由汽车上的微计算机控制。美国已经投入不少的研究人员致力于开发21世纪的计算机车。试想如果有一天，每一个十字路口的红绿灯全部改为交通控制中心的交通状况发报系统，每一部汽车装上了接收器，您可以坐在汽车驾驶座位上舒服的休息，您车上的计算机将配合着交通指挥中心所传给您的信息前进。

## 30.7 IC卡

IC卡是二十世纪的新产物，它极有可能取代身份证，成为人手一张的卡片。它如何能取代身份证呢？因为它具有大量的存储器，用以存储个人的所有数据。身份证只是其中的一小部分。

### 30.7.1 个人数据/病历数据

我的外祖父曾多次突然倒在街头，好心的路人帮忙送至医院，医生只能凭当时的状况判断他的病情。如果当时须要开刀急救，院方如何通知家属呢？几年之后，这种问题可以利用IC卡解决。此时IC卡可将个人数据（如姓名、地址、电话、出生年月日、父母及兄弟姐妹姓名、血型、病历数据...）全部存入IC卡内，倘若有人倒卧街头，可凭他身上的一张卡片，就可以知道他的所有数据。

### 30.7.2 指纹辨认

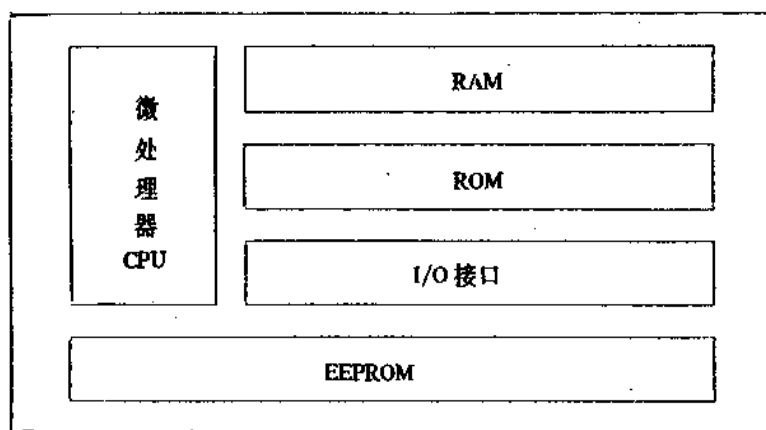
如果您手上有一张银行卡，您的口令可能设置为自己的出生年月日、或学号、或是不规则的数字等等。笔者曾好几次因口令记错，卡片被佛朗非的一家银行没收了。有了IC卡，它可以将您的指纹输入卡内，将来您领钱时，提款机将辨识您的指纹取代口令。

### 30.7.3 银行卡/签帐卡/IC卡

IC卡的另一个功能就是财务管理了。您上街购物不需要带任何现金，立即刷卡，便

可消费。由于它的内部有巨大的存储器。可以随时存储您银行的存款余额。而商店可以决定将您的钱立即存入商店内或者像 Visa 卡一样一个月之后才交易这笔款项。IC 卡的发明将可替一些商号提供了一项更正确的服务，那就是审查用户的信用度。

#### 30.7.4 IC 卡上 IC 的结构



以下分别介绍它的模块：

**微处理器CPU：**利用CPU可做一些数据处理及运算。

**RAM：** Random Access Memory提供一些地址给CPU运算时使用。

**ROM：** Read Only Memory可以存储系统程序。

**I/O接口：** 负责与外界沟通。

**EEPROM：** Electricity Erasable Programmable Read Only Memory存储个人的文件数据及个人财务基本数据。

IC 卡的中心就在于这片 IC 具有心脏(CPU)、存储器以及接口。其中存储器可以分为 3 种，RAM 只是提供暂时的存储位置供微处理器使用。当 IC 卡不使用时，数据就没了。ROM 则提供了系统程序如 IBM PC 的操作系统，它的数据是永远存在，不随电源影响。EEPROM 数据则是可以改变的，它存储个人的基本数据，当 IC 卡不接上电源时，它的数据仍然存在。

**思考题：**

既然 EEPROM 具有这么多的好处，它会不会取代 ROM 存取系统程序？

**提示：** EEPROM 的存储器单位面积太大。

#### 30.7.5 IC 卡的保密性

IC 卡的数据可不可以自己改变，持有者可不可以将卡片上的银行记录更改？答案当然是否定的。但又如何防范意志不坚的银行专员更改 IC 卡的金钱记录？这方面的保密可能就要请专家共同参与了。

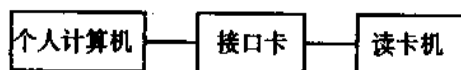
医务人员可以阅读卡片上的病历数据，可以增加新的数据，但却不能更改旧有的数据。不法的医务人员可否趁职务之便顺便查看病人的银行财务状况？如果您是 IC 卡系统设计工程师，您应如何防范这类事件发生？可行的办法即是每个单位都有自己的权责。医院的计算机只可以查看 IC 卡上病历、银行的计算机只能取得银行相关的数据、警政单位



的计算机只能获取个人数据，而您可能只能看数据而不能更改任何数据。

#### 30.7.6 IC 卡与个人计算机

您应如何读取 IC 卡上的数据呢？最便捷的方法就是和自己的个人计算机相连接，接法如下：



这一个接口卡，市面上还没有，请您自行设计。

## 附录 A ASCII 字符和扩展字符表

十进制 X <sub>10</sub>	十六进制 X <sub>16</sub>	八进制 X <sub>8</sub>	ASCII	IBM 绘图符	按键
0	00	00	NUL	(null)	<Ctrl-@>
1	01	01	SOH	☉	<Ctrl-A>
2	02	02	STX	●	<Ctrl-B>
3	03	03	ETX	▼	<Ctrl-C>
4	04	04	EOT	◆	<Ctrl-D>
5	05	05	ENQ	♣	<Ctrl-E>
6	06	06	ACK	♠	<Ctrl-F>
7	07	07	BEL	●	<Ctrl-G>
8	08	10	BS	■	<Ctrl-H>
9	09	11	HT	○	<Ctrl-I>
10	0A	12	LF	■	<Ctrl-J>
11	0B	13	VT	♂	<Ctrl-K>
12	0C	14	FF	♀	<Ctrl-L>
13	0D	15	CR	♪	<Ctrl-M>
14	0E	16	SO	♫	<Ctrl-N>
15	0F	17	SI	♫	<Ctrl-O>
16	10	20	DLE	▶	<Ctrl-P>
17	11	21	DC1	◀	<Ctrl-Q>
18	12	22	DC2	↑	<Ctrl-R>
19	13	23	DC3		<Ctrl-S>
20	14	24	DC4	¶	<Ctrl-T>
21	15	25	NAK	♠	<Ctrl-U>
22	16	26	SYN	..	<Ctrl-V>
23	17	27	ETB	‡	<Ctrl-W>
24	18	30	CAN	‡	<Ctrl-X>
25	19	31	EM	‡	<Ctrl-Y>
26	1A	32	SUB	→	<Ctrl-Z>
27	1B	33	ESC	←	<Esc>
28	1C	34	FS	⌞	<Ctrl-[>
29	1D	35	GS	↔	<Ctrl-^>
30	1E	36	RS	▲	<Ctrl-=>
31	1F	37	US	▼	<Ctrl->>
32	20	40	SP	(Space)	<SPACE BAR>

十进制 X <sub>10</sub>	十六进制 X <sub>16</sub>	八进制 X <sub>8</sub>	ASCII	IBM 编图符	按键
33	21	41	!	!	!
34	22	42	"	"	"
35	23	43	#	#	#
36	24	44	\$	\$	\$
37	25	45	%	%	%
38	26	46	&	&	&
39	27	47	'	'	'
40	28	50	(	(	(
41	29	51	)	)	)
42	2A	52	*	*	*
43	2B	53	+	+	+
44	2C	54	,	,	,
45	2D	55	-	-	-
46	2E	56	.	.	.
47	2F	57	/	/	/
48	30	60	0	0	0
49	31	61	1	1	1
50	32	62	2	2	2
51	33	63	3	3	3
52	34	64	4	4	4
53	35	65	5	5	5
54	36	66	6	6	6
55	37	67	7	7	7
56	38	70	8	8	8
57	39	71	9	9	9
58	3A	72	:	:	:
59	3B	73	;	;	;
60	3C	74	<	<	<
61	3D	75	=	=	=
62	3E	76	>	>	>
63	3F	77	?	?	?
64	40	100	@	@	@
65	41	101	A	A	A

十进制 X <sub>10</sub>	十六进制 X <sub>16</sub>	八进制 X <sub>8</sub>	ASCII	IBM 绘图符	按键
66	42	102	B	B	B
67	43	103	C	C	C
68	44	104	D	D	D
69	45	105	E	E	E
70	46	106	F	F	F
71	47	107	G	G	G
72	48	110	H	H	H
73	49	111	I	I	I
74	4A	112	J	J	J
75	4B	113	K	K	K
76	4C	114	L	L	L
77	4D	115	M	M	M
78	4E	116	N	N	N
79	4F	117	O	O	O
80	50	120	P	P	P
81	51	121	Q	Q	Q
82	52	122	R	R	R
83	53	123	S	S	S
84	54	124	T	T	T
85	55	125	U	U	U
86	56	126	V	V	V
87	57	127	W	W	W
88	58	130	X	X	X
89	59	131	Y	Y	Y
90	5A	132	Z	Z	Z
91	5B	133	[	[	[
92	5C	134	\	\	\
93	5D	135	]	]	]
94	5E	136	^	^	^
95	5F	137	_	_	_
96	60	140	.	.	.
97	61	141	a	a	a
98	62	142	b	b	b

十进制 X <sub>10</sub>	十六进制 X <sub>16</sub>	八进制 X <sub>8</sub>	ASCII	IBM 绘图符	按键
99	63	143	c	c	c
100	64	144	d	d	d
101	65	145	e	e	e
102	66	146	f	f	f
103	67	147	g	g	g
104	68	150	h	h	h
105	69	151	i	i	i
106	6A	152	j	j	j
107	6B	153	k	k	k
108	6C	154	l	l	l
109	6D	155	m	m	m
110	6E	156	n	n	n
111	6F	157	o	o	o
112	70	160	p	p	p
113	71	161	q	q	q
114	72	162	r	r	r
115	73	163	s	s	s
116	74	164	t	t	t
117	75	165	u	u	u
118	76	166	v	v	v
119	77	167	w	w	w
120	78	170	x	x	x
121	79	171	y	y	y
122	7A	172	z	z	z
123	7B	173	{	{	{
124	7C	174			
125	7D	175	}	}	}
126	7E	176	-	-	-
127	7F	177	DEL	DEL	<Del>

二进制 $X_2$	八进制 $X_8$	十进制 $X_{10}$	十六进制 $X_{16}$	Ext. ASCII
1000 0000	200	128	80	Ç
1000 0001	201	129	81	ü
1000 0010	202	130	82	é
1000 0011	203	131	83	à
1000 0100	204	132	84	ä
1000 0101	205	133	85	å
1000 0110	206	134	86	ä
1000 0111	207	135	87	ç
1000 1000	210	136	88	è
1000 1001	211	137	89	é
1000 1010	212	138	8A	è
1000 1011	213	139	8B	ï
1000 1100	214	140	8C	î
1000 1101	215	141	8D	ì
1000 1110	216	142	8E	À
1000 1111	217	143	8F	Á
1001 0000	220	144	90	Ê
1001 0001	221	145	91	æ
1001 0010	222	146	92	Æ
1001 0011	223	147	93	ô
1001 0100	224	148	94	ö
1001 0101	225	149	95	ó
1001 0110	226	150	96	û
1001 0111	227	151	97	ü
1001 1000	230	152	98	ÿ
1001 1001	231	153	99	Ö
1001 1010	232	154	9A	Ü
1001 1011	233	155	9B	é
1001 1100	234	156	9C	£
1001 1101	235	157	9D	¥
1001 1110	236	158	9E	Þ
1001 1111	237	159	9F	ƒ
1010 0000	240	160	A0	à

二进制 $X_2$	八进制 $X_8$	十进制 $X_{10}$	十六进制 $X_{16}$	Ext. ASCII
1100 0010	302	194	C2	␣
1100 0011	303	195	C3	␣
1100 0100	304	196	C4	␣
1100 0101	305	197	C5	␣
1100 0110	306	198	C6	␣
1100 0111	307	199	C7	␣
1100 1000	310	200	C8	␣
1100 1001	311	201	C9	␣
1100 1010	312	202	CA	␣
1100 1011	313	203	CB	␣
1100 1100	314	204	CC	␣
1100 1101	315	205	CD	␣
1100 1110	316	206	CE	␣
1100 1111	317	207	CF	␣
1101 0000	320	208	D0	␣
1101 0001	321	209	D1	␣
1101 0010	322	210	D2	␣
1101 0011	323	211	D3	␣
1101 0100	324	212	D4	␣
1101 0101	325	213	D5	␣
1101 0110	326	214	D6	␣
1101 0111	327	215	D7	␣
1101 1000	330	216	D8	␣
1101 1001	331	217	D9	␣
1101 1010	332	218	DA	␣
1101 1011	333	219	DB	␣
1101 1100	334	220	DC	␣
1101 1101	335	221	DD	␣
1101 1110	336	222	DE	␣
1101 1111	337	223	DF	␣
1110 0000	340	224	E0	␣
1110 0001	341	225	E1	␣
1110 0010	342	226	E2	␣

二进制 X <sub>2</sub>	八进制 X <sub>8</sub>	十进制 X <sub>10</sub>	十六进制 X <sub>16</sub>	Ext. ASCII
1110 0011	343	227	E3	ƒ
1110 0100	344	228	E4	Σ
1110 0101	345	229	E5	σ
1110 0110	346	230	E6	μ
1110 0111	347	231	E7	τ
1110 1000	350	232	E8	ϕ
1110 1001	351	233	E9	θ
1110 1010	352	234	EA	π
1110 1011	353	235	EB	δ
1110 1100	354	236	EC	∞
1110 1101	355	237	ED	ϕ
1110 1110	356	238	EE	€
1110 1111	357	239	EF	∩
1111 0000	360	240	F0	≡
1111 0001	361	241	F1	±
1111 0010	362	242	F2	≥
1111 0011	363	243	F3	≤
1111 0100	364	244	F4	{
1111 0101	365	245	F5	}
1111 0110	366	246	F6	÷
1111 0111	367	247	F7	≡
1111 1000	370	248	F8	•
1111 1001	371	249	F9	•
1111 1010	372	250	FA	•
1111 1011	373	251	FB	√
1111 1100	374	252	FC	η
1111 1101	375	253	FD	ι
1111 1110	376	254	FE	■
1111 1111	377	255	FF	(blank 'F')



# 扩展 ASCII 码

按键	产生的扩展	按键	产生的扩展
	ASCII 码		ASCII 码
F1	0,59	Ctrl-F1	0,94
F2	0,60	Ctrl-F2	0,95
F3	0,61	Ctrl-F3	0,96
F4	0,62	Ctrl-F4	0,97
F5	0,63	Ctrl-F5	0,98
F6	0,64	Ctrl-F6	0,99
F7	0,65	Ctrl-F7	0,100
F8	0,66	Ctrl-F8	0,101
F9	0,67	Ctrl-F9	0,102
F10	0,68	Ctrl-F10	0,103
Shift-F1	0,84	Alt-F1	0,104
Shift-F2	0,85	Alt-F2	0,105
Shift-F3	0,86	Alt-F3	0,106
Shift-F4	0,87	Alt-F4	0,107
Shift-F5	0,88	Alt-F5	0,108
Shift-F6	0,89	Alt-F6	0,109
Shift-F7	0,90	Alt-F7	0,110
Shift-F8	0,91	Alt-F8	0,111
Shift-F9	0,92	Alt-F9	0,112
Shift-F10	0,93	Alt-F10	0,113

## 附录 B Turbo C 用于 I/O 的函数介绍

Turbo C 提供了 4 个函数可供我们运行 I/O 控制，由于这些函数是定义在 dos.h 内，所以使用前请在程序前方加上以下指令：

```
#include <dos.h>
```

### B.1 inportb( )

本函数主要是供你读取所指定 I/O 口的字节数据 (8 位)，本函数的使用格式如下：

```
byteread=inportb(portnum);
```

运行完后 portnum 所指 I/O 口地址的字节值，便会被读入 byteread 变量内。

### B.2 inport( )

本函数主要是供你读取所指定 I/O 口的字节数据 (16 位)，本函数的使用格式如下：

```
wordread=inport(portnum);
```

运行完后 portnum 所指的 I/O 口地址的字值，便会被读入 wordread 变量内。

### B.3 outportb( )

本函数主要是将字节数据，输出至所指定的 I/O 口内，本函数的使用格式如下：

```
outportb(portnum, bytewrite);
```

运行完后 bytewrite 8 位数据会被送至 portnum 所指的 I/O 口地址内。

## B.4 output( )

本函数主要是将字组数据输出至所指定的 I/O 口内, 本函数的使用格式如下:

```
output(portnum,wordwrite);
```

运行完后 wordwrite 16 位数据会被送至 portnum 所指的口地址内。

## B.5 简单 I/O 控制声音的应用

PC 上绝大多数的 I/O 操作都是由系统主机板上的 8255 可编程外设接口 (programmable Peripheral Interface, PPI) 芯片控制。PPI8255 内部有 3 个 8 位的寄存器, 分别对应到 I/O 口 (Port) 的 60H, 61H 和 62H。其中有 60H 主要是用来处理键盘输入, 62H 则是专门处理其它输入, 例如, PC XT 的卡带输入, 由于以上二个口和喇叭发音没有关系, 在此不多叙述。

真正和喇叭发音有关的是 61H Port, 这是 PPI 8255 的输出口, 如图 B-1 所示。在此图中有两个信号来源可使 PC 的喇叭动作, 这两个来源分别由 PPI 8255 输出寄存器的最低两个位所控制。若第 0 位为 1, 喇叭由 8253 计时器来控制。若第 0 位为 0, 喇叭是属直接控制, 此时若第 1 位为 1 时, 喇叭是 ON, 第 1 位为 0 时, 喇叭是 OFF, 这样借着喇叭不停地 ON 和 OFF, 便发出声音了。本章将介绍直接喇叭控制的方法, 至于以计时器控制喇叭的方法, 可参考有关汇编语言应用书籍。

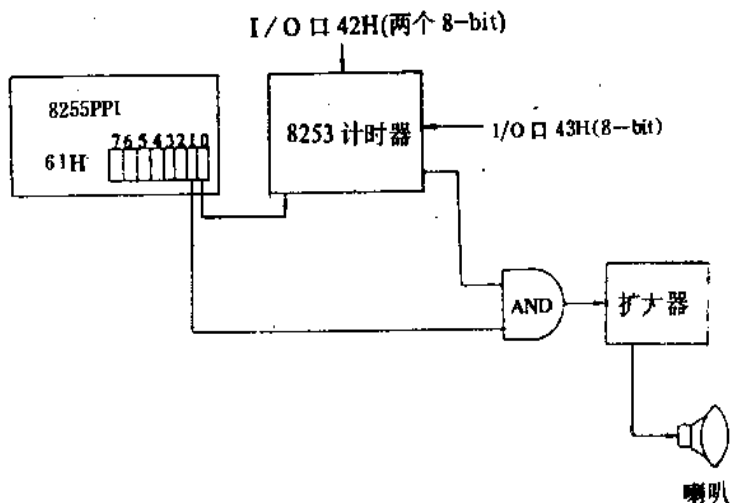


图 B.1

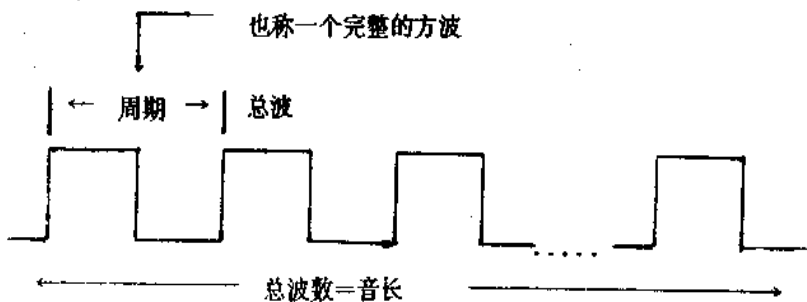
刚刚已经说过 61H 口的第 0 位为 0 时, 就可直接控制喇叭, 所以在编写程序中要用到下列指令:

```
portnum=0x61;          /* 设置portnum为口61h */
byteread=inportb(portnum); /* 读取61H口值 */
byteread &= 0xfc;       /* 设置第0位为0 */
outportb(portnum, byteread); /* 完成直接控制喇叭动作 */
```

另外当第 1 位等于 1 时，喇叭是 ON，第 1 位等于 0 时，喇叭是 OFF，所以编写 ON、OFF 时，你须分别利用下列的程序：

```
byteread = 0x02;        /* 设置喇叭ON */
outportb(portnum, byteread);
byteread &= 0xfd;       /* 设置喇叭OFF */
outportb(portnum, byteread);
```

在上述 ON 和 OFF 不断动作时，将可产生声音方波，如下图所示：



方波的半周期大约等于一个延迟循环的运行时间。一个完整的方波时间称为周期(period)，它的时间相当于运行延迟循环时间的两倍。声音的频率就是这周期的倒数，因此只要改变延迟循环的运行时间就能改变声音频率。

假设我们想产生 500Hz 的声音 2 秒钟，则所应产生的周期波数如下：

$$\text{每一方波时间} = \frac{1}{500\text{Hz}} = 2\text{毫秒}$$

$$\text{周期波数} = \frac{2\text{秒}}{2\text{毫秒}} = 1000$$

由于每一个方波时间是 2 毫秒，所以半个波长时间是 1 毫秒。

程序示例 b\_1.c

按照上述概念设计一个程序来产生频率 500Hz 持续 2 秒的声音。

```

/* ----- Program Description ----- */
/* program name : b_1.c */
/* sound generation */
/* ----- */
#include <dos.h>

void main(void)
{
    int portnum = 0x61;
    int loopnum = 1000;
    int timedelay = 1;
    unsigned char byteread;
    int i;

    /* initialize the speaker */
    byteread = inportb(portnum);
    byteread &= 0xfe;
    outportb(portnum, byteread);
    for ( i = 0; i < loopnum; i++ )
    {
        /* speaker on */
        byteread |= 0x02;
        outportb(portnum, byteread);
        delay(timedelay);

        /* speaker off */
        byteread &= 0xfd;
        outportb(portnum, byteread);
        delay(timedelay);
    }
}

```

## 附录 C 本书所使用的 Turbo C 的函数表

### C.1 clrscr( )

clrscr( )函数主要功能是将屏幕内容清成空白，它的使用格式如下：

```
clrscr( );
```

由于本功能函数是被定义在 conio.h 内，所以在程序前面要加上下列指令：

```
#include <conio.h>
```

#### 程序示例 c\_1.c

简单清除屏幕的程序设计，本程序会在列出 5 个笑脸后，将屏幕内容清除。

```
/* ===== Program Description ===== */
/* program name : c_1.c */
/* clear the screen */
/* ===== */
#include <conio.h>
```

```
void main(void)
{
    int i;

    for ( i = 0; i < 5; i++ )
    {
        printf("\1\n");
        sleep(1);
    }
    clrscr( );
}
```

### C.2 cprintf( )

cstdio( )函数主要是与 window( )函数(C-20) 配合使用，当它与 window( )函数配

合使用时，它的功能和 printf( )函数 (C-11)相同，不过若是各位在不同前景和背景颜色下使用时，本函数可达到以反白显示字符串的效果，而 printf( )函数在 window( )函数下使用时，无法达到反白显示的效果。

由于本功能函数是被定义在 conio.h 内，所以在程序前面要加上下列指令。

```
#include <conio.h>
```

### 程序示例 c 2.c

本程序在运行时的 window 1 字符串会闪烁，其它文字前景是蓝色，背景是黄色。

```
/* ===== Program Description ===== */
/*   program name : c_2.c                               */
/*   textcolor and textbackground application.           */
/* ===== */
#include <conio.h>

void main( )
{
    int i;

    clrscr( );

    window(20,3,60,10);
    gotoxy(17,2);
    textcolor(BLINK+WHITE);
    cprintf("Window 1");
    gotoxy(1,5);
    textcolor(BLUE);
    textbackground(YELLOW);
    for ( i = 0; i < 100; i++ )
        cprintf("\1");
}
```

## C.3 delay( )

delay( )函数主要是让 CPU 暂停工作，它的使用格式如下：

```
delay(unsigned milliseconds);
```

在上述使用格式中,milliseconds 代表让 CPU 暂停工作时间,其单位是千分之一秒。

示例: 下列指令将可令 CPU 暂停 0.3 秒。

```
delay(300);
```

由于本功能函数是被定义在 dos.h 内,所以在使用前请在程序前方加上下列指令:

```
#include <dos.h>
```

#### 程序示例 c\_3.c

以 delay( )函数每隔 0.3 秒显示一次笑脸。

```
/* ===== Program Description ===== */
/*   program name : c_3.c                      */
/*   Print smiling face 5 times.                */
/* ===== */
#include <dos.h>
```

```
void main(void)
```

```
{
```

```
    int i = 5;
```

```
    for ( i = 0; i < 5; i++ )
```

```
    {
```

```
        delay(300);
```

```
        printf("\1\n");
```

```
    }
```

```
}
```

### C.4 getch( )

本函数主要功能是由于读取键盘所输入的字符,不过键盘所输入的字符将不在屏幕上显示。本函数的使用规则如下:

```
ch=getch( );
```

由于本功能函数是被定义在 stdio.h 内,所以在使用前请在程序前方加上下列指令:

```
#include <stdio.h>
```



### 程序示例 c\_4.c

getch( )函数的基本应用。

```
/* ===== Program Description ===== */
/*   program name : c_4.c                      */
/*   getch( ) application.                      */
/* ===== */
#include <stdio.h>

void main( )
{
    char ch1, ch2;

    printf("\1: Please enter 2 characters \n==>");
    ch1 = getch( );
    ch2 = getch( );
    printf("\n");
    printf("\2: The first character is \n==>");
    putchar(ch1);
    printf("\n");
    printf("\2: The second character is \n==>");
    putchar(ch2);
}
```

## C.5 getvect( )

本功能可以取得指定中断号码的中断处理程序的地址。它的使用格式如下：

```
void interrupt ( * int__handler)( )
int__handler = getvect(number);
```

经上述函数后 number (中断号码) 的处理程序地址将被放在 int\_\_handler 内。由于本功能函数是被定义在 dos.h 内，所以在使用前在程序前面加上下列指令：

```
#include <dos.h>
```

### 程序示例 c 5.c

请将中断 0×18 存在 int\_\_handler 内，且将它显示出来，在本程序示例中 %Fp 会将地址以 rrrr:tttt 方式显示，其中 rrrr 是段地址；tttt 是偏移地址。

```

/* ===== Program Description ===== */
/* program name : c_5.c */
/* getvect( ) application. */
/* ----- */
#include <dos.h>

void main( )
{
    void interrupt ( * int_handler )( );

    int_handler = getvect(0x18);
    printf("\n1: The address of 0x18 is %Fp",int_handler);
}

```

## C.6 gotoxy( )

IBM PC 个人计算机系统文字模式下，其屏幕坐标关系图形如下所示：

	1	X 轴	80
1	1,1	.....	80,1
Y	.		.
轴	.		.
	.		.
	.		.
25	1,25	.....	80,25

注：本图坐标以(X,Y)为基准

gotoxy( )函数可将光标移至所指定的位置，它的使用格式如下：

```
gotoxy(x,y)
```

由于本函数是被定义在 conio.h 内，所以在使用前请在程序前面加上下列指令：

```
#include <conio.h>
```

### 程序示例 c\_\_6.c

基本 gotoxy( )函数的应用。

```
/* ----- Program Description ----- */
/*   program name : c__6.c               */
/*   gotoxy( ) application.               */
/* ----- */
#include <conio.h>

void main( )
{
    int i;

    clrscr( );
    for ( i = 1; i < 10; i++ )
    {
        gotoxy(i,i+2);
        printf("\1\2\1\2\1");
    }
}
```

## C.7 kbhit( )

kbhit( )函数主要是用于检查我们是否按下任何一个键，如果有则返回非零数值，如果没有则返回零，它的使用格式如下：

```
int kbhit( );
```

由于本函数是定义在 conio.h 内，所以在使用前请在程序前方加上下列指令。

```
#include <conio.h>
```

### 程序示例 c\_\_7.c

本程序在运行时将产生噪音，只要随便按一个键就可令噪音终止。

```
/* ----- Program Description ----- */
/*   program name : c__7.c               */
/*   Noise generation.                   */
/* ----- */
```

```

#include <dos.h>
#include <stdlib.h>
#include <conio.h>
void main(void)
{
    unsigned frequency;

    while ( !kbhit() )
    {
        frequency = random(65535);
        sound(frequency);
    }
    nosound( );
}

```

## C.8 localtime( )

localtime( )函数的主要功能是将以长整数存放的时间和日期数据转换成 tm 结构的时间和日期数据形式。本函数的使用格式如下：

```
struct tm * localtime(time_t * time);
```

由于 localtime( )函数是定义在 time.h 中，所以在用这些函数前，请在程序前面加上下列指令。

```
#include <time.h>
```

同时在 time.h 内，包含两种数据结构与时间的存储有关。

1. tm: 这是一个结构(struct)的数据形式，主要是用来存放时间和日期数据，如下图所示：

```

struct tm
{
    int tm_sec;           /* seconds, 0-59 */
    int tm_min;           /* minutes, 0-59 */
    int tm_hour;          /* hours, 0-23 */
    int tm_mday;          /* day of the month, 1-31 */
    int tm_mon;           /* months since Jan., 0-11 */
    int tm_year;          /* years from 1900 */
    int tm_wday;          /* days since Sunday, 0-6 */
    int tm_yday;          /* days since Jan., 1-365 */
    int tm_isdst;         /* daylight savings time indicator */
}

```

上图中若 `tm_isdst` 是正数，表示是夏时制(Daylight savings time)，如果是 0 则代表日常时间，如果是负数代表无日光节约时间信息。

2. `time_t`: 这是一个长整数，它也是用来存放时间和日期数据的。

程序示例 `c_8.c`

显示现在时间 (时: 分)

```

/* ===== Program Description ===== */
/*   program name : c_8.c                      */
/*   Print DOS system hour and minutes.          */
/* ===== */

#include <time.h>
#include <stdio.h>

void main(void)
{
    time_t lt;
    struct tm * ptr;

    lt = time(NULL);
    ptr = localtime(&lt);
    printf("\1: current time is %d:%d \n", ptr->tm_hour, ptr->tm_min);
}

```

## C.9 nosound( )

`sound( )`函数，发出声音之后，即使你离开 Turbo C 整个窗口环境，喇叭所发出的声音仍将继续，本小节所介绍的 `nosound( )`函数主要功能是关闭喇叭声音。此

nosound( )函数的使用格式如下:

```
nosound(void);
```

由于本功能函数是定义在 dos.h 内, 所以在使用前, 请在程序前面加上下列指令.

```
#include <dos.h>
```

#### 程序示例 c\_9.c

请输入声音频率及声音长度, 本程序将产生此声音.

```
/* ===== Program Description ===== */
/*   program name : c_9.c                      */
/*   Sound generation.                          */
/* ===== */
#include <dos.h>

void main(void)
{
    int    frequency, last;

    printf("\n! Please input sound frequency. \n");
    scanf("%d",&frequency);
    printf("\n! Please input second. \n");
    scanf("%d",&last);
    sound(frequency);
    sleep(last);
    nosound( );
}
```

#### C.10 pow( )

这个函数可让我们求某数的某次方值, 它的使用方式如下:

```
pow(double x, double y);
```

上述函数调用时, 会返回下列公式的运算结果.

$$x^y$$

注意:  $x$  和  $y$  都是双精度数, 由于主函数是定义在 `math.h` 内, 所以在使用前请在程序前面加上下列指令:

```
#include <math.h>
```

#### 程序示例 c\_\_10.c

`pow` 函数的基本应用.

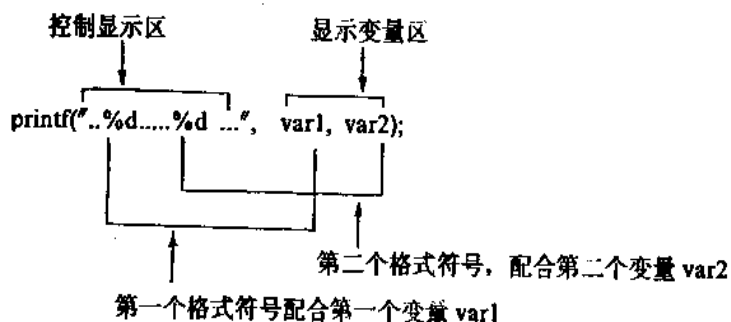
```
/* ===== Program Description ===== */
/*   program name : c__10.c                               */
/*   pow(x,y)                                             */
/* ===== */
#include <math.h>
void main( )
{
    double x = 3.2;
    double y = 1.8;

    printf("\n2: The pow(x,y) is --> %f \n",pow(x,y));
}
```

### C.11 printf( )

#### C.11.1 %d 十进制整数的显示

`printf` 除了可以直接显示字符串 (只要在控制显示区内直接放字符串就可以了) 之外, 我们还可以格式化的方式控制输出的结果, 本节我们教你如何利用 `%d` 控制十进制整数的显示, 基本上它的显示结构如下所示:



在使用上述显示数据结构时, 必须注意下列几点:

1. 第一个格式符号配合第一个要显示的变量, 其它依此类推.

2. 在控制显示区内的格式符号之间，可以有许多空格，或是没有任何空格。
3. 在显示变量区内，各变量之间一定要有逗号隔开。
4. 控制显示区需用双撇号" "包夹起来。
5. 控制显示区和显示变量区之间需用逗号隔开。

另外，在使用显示变量时，还必须要知道如何修饰输出的位置。这个修饰字通常是由阿拉伯数字构成，一般我们把它放在%和d之间。修饰字和整数格式输出间的规则如下所示：

1. %d：在此类的输出格式下，C语言输出的格数和变量的长度相同。

示例：假设变量值是 356，则输出时 C 语言预留 3 格空间给它，如下所示：

3	5	6
---	---	---

假设变量值是 18，则输出时 C 语言会预留 2 格空间给它，如下所示：

1	8
---	---

2. %nd：n是整数值，代表C语言输出时预留的输出格式。使用此种方式输出时，会遇到两个情况：第一，预留格数比输出值所需要的空间还大，此时 C 语言会将输出结果向右对齐。另一种情况是预留格数比输出值所需空间还小，此时 C 语言会忽略格数，而自动配予实际所需的格数。

示例：假设变量值是 356，控制显示的格式符号是%2d，则显示结果如下所示：

3	5	6
---	---	---

示例：假设变量值是 356，控制显示的格式符号是%5d，则显示结果如下所示：

		3	5	6
--	--	---	---	---

3. %-nd：这个输出格式和前一个类似，唯一不同的是，若预留格数比输出值所需的  
空间还大时，此时 C 语言会将输出结果向左对齐。

示例：假设变量值是 356，控制显示的格式符号是%-5d，则显示结果如下所示：

3	5	6		
---	---	---	--	--

### C.11.2 %f 小数的显示

小数变量显示使用规则如下：

1. %f：在此类的输出格式下，C语言会预留10格空间供输出使用，假设格数空间大



于变量值所需的空问，则剩余空问则供变量的小数点使用。

示例：假设变量值是 123.46，控制格式符号是 %f 则输出结果如下所示：

1	2	3	.	4	6	0	0	0	0
---	---	---	---	---	---	---	---	---	---

值得注意的是，一般系统小数只能存储 6 或 7 个数字的精确度（又称有效位数）而我们所要的输出数字是 10 格，所以真正输出时也许小数部分的价值会略为不同于实际值。

2. %m.nf：在这种格式输出下，m 代表小数的输出宽度，n 代表小数部分所需宽度。和整数输出格式一样，如果所要求的空问不够，系统会自己分配足够的空问输出使用。若是所分配的空问太多，则系统输出结果会向右靠齐。

示例：假设变量值是 123.56，控制格式符号是 %8.2f，则输出结果如下所示：

		1	2	3	.	5	6
--	--	---	---	---	---	---	---

3. %-m.nf：这个输出格式和上一规则类似，唯一的不同是，若预留格数比输出值所需的空问大时，C 语言会将输出结果向左对齐。

示例：假设变量值是 123.56，控制格式符号是 %-8.2f，则输出结果如下所示：

1	2	3	.	5	6		
---	---	---	---	---	---	--	--

### C.11.3 %c 字符的显示

字符显示的规则如下所示：

1. %c：在此格式下 C 语言会预留一格空问供输出使用。

示例：假设变量值是 'a'，控制格式符号是 %c，则输出结果如下所示：

a
---

2. %nc：在此格下 C 语言会预留 n 格空问供输出使用，但输出结果将会向右靠齐。

示例：假设变量值是 'a'，控制格式符号是 %3c，则输出结果如下所示：

		a
--	--	---

3. %-nc：在此格下 C 语言会预留 n 格空问供输出使用，但输出结果将会向左靠齐。

示例：假设变量值是 'a'，控制格式符号是 %-3c，则输出结果如下所示：

a		
---	--	--

#### C.11.4 其它格式化数据显示原则

除了以上常用的格式化输出变量值的应用外, print( )尚提供下列格式化显示方式:

1. %s: 主要用于显示字符串。
2. %e: 以 e 记号 (也就是科学符号) 表示法输出小数。
3. %u: 不带符号的 10 进制整数输出。
4. %o: 8 进制整数输出。
5. %x: 16 进制整数输出。

以上五种输出格式, 也和整数或小数输出格式一样有类似的输出原则:

1. 在 % 和符号格式值之间若没有任何修饰字, 则 C 语言会按照实际需要输出。  
    %s——对字符串而言, 会按照字符串长度输出。  
    %e——预留 12 格供输出使用。  
    %u,%o, %x——按实际需要格数输出。
2. 若 % 和符号格式值之间有修饰词指定输出长度, 则有两种情况: 第一, 若指定长度大于输出要求, 则显示时会向右靠齐。若是指定长度小于输出要求, 则 C 语言会自动配给足够空间供它使用。
3. 当 % 和修饰词之间“-”有符号时, 若指定输出长度大于输出要求长度, 则显示时会向左靠齐。

程序示例 c\_11.c

printf( )函数的基本应用。

```
/* ----- Program Description ----- */
/*   program name : c_11.c                      */
/*   Formatted output.                          */
/* ----- */
```

```
void main( )
```

```
{
```

```
    int i = 10;
```

```
    float j = 123.56;
```

```
    printf("floating point output\n");
```

```
    printf("/ %8.2f / \n",j);
```

```
    printf("integer output\n");
```

```
    printf("/ %5d / \n");
```

```
    printf("format octal value output\n");
```

```
    printf("/ %o / \n",i);
```

```
    printf("/ %-8o / \n",i);
```

```
    printf("format hexidecimal value output\n");
```

```

printf("/ %x / \n",i);
printf("/ %8x / \n",i);
printf("format unsigned value output\n");
printf("/ %u / \n",i);
printf("/ %8u / \n",i);
printf("format scientific symbol value output\n");
printf("/ %e / \n",i);
printf("/ %8.3e / \n",i);
}

```

## C.12 random( )

在 Turbo C 内, 有一个很常用的随机数函数是 random( ), 此函数可产生 0 到某一特定值间的随机数。此函数的使用格式如下:

```
random(int num);
```

上述函数运行完后, 可产生 0~(num-1) 间的随机数。由于本函数被定义在 stdlib.h 内, 所以在使用前请在程序前面加上下列指令:

```
#include <stdlib.h>
```

### 程序示例 c\_12.c

行出 10 个 1 至 100 间的随机数。

```

/* ----- Program Description ----- */
/*   program name : c_12.c               */
/*   random( ) function application.       */
/* ----- */
#include <stdlib.h>

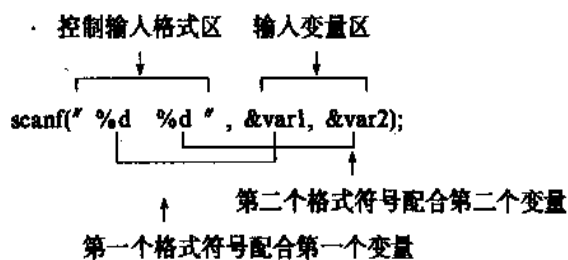
void main(void)
{
    int num = 100;
    int i;

    for ( i = 0; i < 10; i++ )
        printf("\1: %d\n",random(num));
}

```

## C.13 scanf( )

scanf( )函数和 printf( )相类似, 不过它主要是用来做数据输入。和 printf( )一样, 我们也可以将它的参数区分成两部分, 一是控制输入格式区, 另一是输入变量区, 如下所示:



在使用上述函数读取数据时, 必须注意下列几点:

1. 第一个格式符号配合第一个要输入的变量, 其它依此类推。
2. 控制输入格式区需用双撇号包夹起来。
3. 控制输入格式区和变量区之间用逗号分开。
4. 输入变量前面要加&符号, 这是一个地址符号, 数据读入时C语言会将所读入的值放在这个地址内。截至目前为止, 读者只要知道在变量前面加上&符号就可以了, &符号代表变量的地址, 至于有关&符号的细节, 我们将在第5章为各位做详细说明。
5. scanf( )函数所能读取数据的种类和 printf( )所能显示数据种类相同。
6. 读取字符串变量时, 我们不必在字符串变量前面加上&符号。

下面是控制输入格式符号和输入数据形式的对照图。

输入格式符号	输入数据形式
%d	整数
%f	实数
%c	字符
%s	字符串
%e	科学符号
%u	不带符号 10 进制整数
%o	8 进制整数
%x	16 进制整数

注意：在输入整数或小数时，我们可以用空格区别所输入的数据。但是在输入字符时，字符不可有空格。

程序示例 c\_13.c

基本 scanf( )函数的应用。

```
/* ===== Program Description ===== */
/*   program name : c_13.c                               */
/*   scanf and printf application.                         */
/* ===== */
#include <stdio.h>
void main( )
{
    int i,j,k,sum;
    char ch1,ch2;
    float x1,x2,ave;
    printf("\1: Please enter 2 characters \n==>");
    scanf("%c%c",&ch1,&ch2);
    printf("\2: The reverse of these 2 characters are \n==>");
    printf("%c%c\n",ch2,ch1);
    printf("\1: Please enter 3 integer numbers \n==>");
    scanf("%d %d %d",&i,&j,&k);
    sum = i + j + k;
    printf("\2: The sum of your input is ==> %d\n",sum);
    printf("\1: Please enter 2 floating numbers\n==>");
    scanf("%f %f",&x1,&x2);
    ave = ( x1 + x2 ) / 2.0;
    printf("The average of your input is ==> %6.2f\n",ave);
}
```

## C.14 setvect( )

本功能可用于指定的中断号码重新设置中断处理例程的地址。它的使用格式如下：

```
void interrupt new_handler(void);
setvect(int __number, new_handler);
```

在上述格式中，int \_\_number 是中断号码，而 new\_handler 则是新的中断处理例程地址的指针。由于本函数是定义在 dos.h 内，所以在使用前请在程序前面加上下列指令：

```
#include <dos.h>
```

有关这方面的实例，各位可参考第 8 章程序示例。

## C.15 sleep( )

sleep( )函数主要功能是让 CPU 暂时停止工作，它的使用格式如下：

```
sleep(unsigned seconds);
```

在上述使用格式中，seconds 代表让 CPU 暂停工作的时间，其单位是秒。

示例：假设我想让 CPU 暂停 5 秒钟，则指令应如下：

```
sleep(5);
```

由于此函数是包含在 dos.h 内，所以程序前方加上下列指令：

```
#include <dos.h>
```

### 程序示例 c\_15.c

```
/* ===== Program Description ===== */
/* program name : c_15.c */
/* Print smiling face 5 times. */
/* ===== */
```

```
#include <dos.h>
```

```
void main(void)
```

```
{
```

```
    int i = 5;
```

```
    for ( i = 0; i < 5; i++ )
```

```
    {
```

```
        sleep(1);
```

```
        printf("\n\n");
```

```
    }
```

```
}
```

## C.16 sound( )

sound( )函数主要功能是让计算机的喇叭发出声音，它的使用格式如下：

```
void sound(unsigned int frequency);
```

在上述使用格式中，frequency 代表声音的频率值，由于本函数是被定义在 dos.h 内，所以请在程序前面加上下列指令：

```
#include <dos.h>
```

有关本函数应用的程序示例可参考 c\_9.c

## C.17 textbackground( )

textbackground( )函数则是用于设置窗口内所显示数据的背景颜色，它的使用格式如下：

```
textbackgrounded (整数值);
```

整数值的定义如下图所示：

整数值	英文符号名称	中文说明
0	BLACK	黑
1	BLUE	蓝
2	GREEN	绿
3	CYAN	青
4	RED	红
5	MAGENTA	紫
6	BROWN	棕
7	LIGHTGRAY	浅灰

由于本函数是被定义在 conio.h 内，所以在使用前请在程序前面加上下列指令：

```
#include <conio.h>
```

有关本函数使用的程序示例可参考 c\_2.c.

## C.18 textcolor( )

textcolor( )函数主要是供你设置窗口内所显示数据的前景颜色，它的使用格式如下：

textcolor (整数值);

整数值的定义如下图所示：

整数值	英文符号名称	中文说明
0	BLACK	黑
1	BLUE	蓝
2	GREEN	绿
3	CYAN	青
4	RED	红
5	MAGENTA	紫
6	BROWN	棕
7	LIGHTGRAY	浅灰
8	DARKGRAY	深灰
9	LIGHTBLUE	浅蓝
10	LIGHTGREEN	浅绿
11	LIGHTCYAN	浅青
12	LIGHTRED	浅红
13	LIGHTMAGENTA	浅紫
14	YELLOW	黄
15	WHITE	白
128	BLINK	闪烁

在上图中，最后一项是闪烁功能，假设你想令某个前景字符串闪烁，只要将原先 textcolor( )函数内的值加上 128 (或 BLINK) 就可以了，如下所示：

```
textcolor(YELLOW+BLINK);
```



由于本功能函数是被定义在 conio.h 内, 所以在使用前请在程序前面加上下列指令:

```
#include <conio.h>
```

有关本函数使用的程序示例可参考 c\_\_2.c.

### C.19 time( )

time( )函数主要是以长整数的方式, 存放系统日期和时间数据, 它的使用格式如下:

```
time_t time(time_t * time);
```

上述函数运行完后, 系统的时间和日期数据会被存至长整数内, 若是我们想打印长整数内的时间和日期数据, 我们必须利用其它转换函数, 由于本函数是被定义在 dos.h 内, 所以在使用前, 请在程序前面加上下列指令:

```
#include <dos.h>
```

有关本函数的程序示例可参考 c\_\_2.c.

### C.20 window( )

截至目前为止, 各位都是把整个屏幕当做一个窗口, 然后利用此屏幕做数据处理, 其实各位在设计程序的同时, 也可以在屏幕任何位置创建各个大小不等的窗口的。窗口的创建函数如下所示:

```
window(left, top, right, bottom);
```

在上述函数中, 各参数的意义如下:

left : 窗口左边x轴坐标

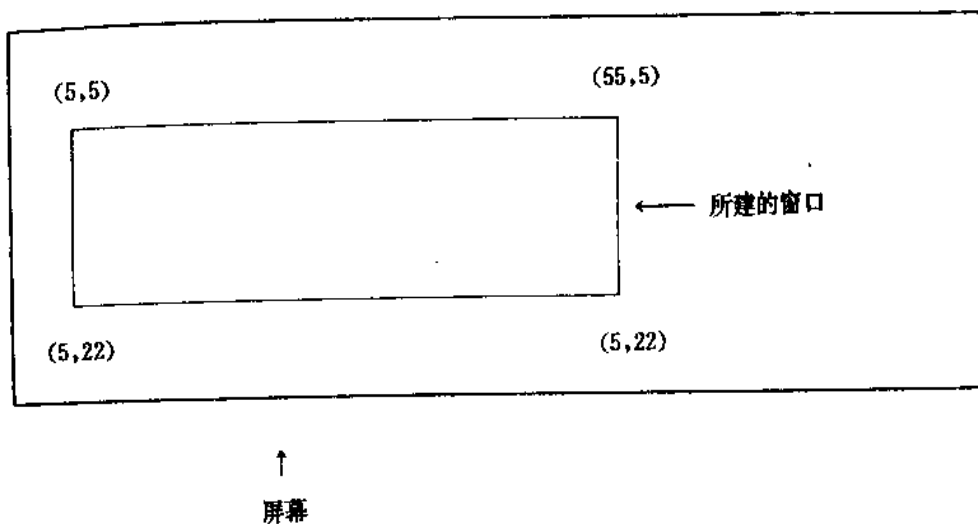
top : 窗口上端y轴坐标

right : 窗口右边x轴坐标

bottom : 窗口下端y轴坐标

假设有一道指令如下所示:

```
window(5,5,55,22);
```



在你创建窗口之前所有的坐标处理，都以屏幕坐标为参考坐标，但当创建好窗口后，所有的坐标处理则以所建的窗口为基准。

由于本函数是被定义在 conio.h 内，所以在使用前，请在程序前面加上下列指令：

```
#include <conio.h>
```

有关本函数使用的程序示例可参考 C\_\_2.c。

## 附录 D IC 的基本常识

### IC 是什么

当打开计算机，我们可以看见一排排的黑色胶体，它们就是 IC(Integrated Circuit)。这些 IC 内部含有数百、数千或数万个晶体管。由于 IC 工业不断的进步，这些 IC 愈做愈小（为了提高优良品率及增加每片芯片的集成度）。现今的技术已经到了 sub-Micron（亚微米），也就是说晶体管门极的宽度已经小于  $10^{-6}$  公尺。

### IC 设计方式

IC 的设计可以分为两个方面来讨论

1. 全定制 (Fully customer) 设计
2. 半定 (Semi customer) 设计

一般全定制 IC 是指 IC 的要求规格由客户或 IC 设计公司指定，设计公司根据规格，设计出一个个的晶体管，然后组合成一个全新的 IC，这一个设计需时较长，从定规格到交货常常超出两年。

半定制则指利用标准单元 (Standard Cell) 去从事快速的设计，因为标准单元已经成功的在其它设计中验证过了。所以此类的设计较省事。这一种设计可以由 IC 的使用商自行负责，国内一些厂家则提供 Stand Cell Library 供客户使用。另一种方式则是厂商提供门阵列 (Gate Array)，也就是说 IC 的制造已经完成了 90%。

所有的晶体管已经成功的装造在芯片 (Wafer) 上了，客户只须提供这些晶体管的连线即可做完最后一部分的完成工作。这种设计非常快速。一般只须数周即可完成 IC 的制造。客户参与了这类的设计，所以称为半定制式设计。

### IC 的保密性

当您辛辛苦苦的花费十万或百万的费用利用了一些标准 IC 设计出全世界第一部手提影视电话时，您一定非常恐惧一些海盗商人花几个小时的时间就可以将您的心血拷贝完成，怎么办？也许您接下来的工作就是跑法院，有时候法官们基于专业知识的不足，让您气极，有时候生米煮成熟饭，您又欲哭无泪，怎么办？最好的方式即是刚开始时，您必须舍弃现成的标准 IC，和 IC 设计公司合作，共同开发您的专用 IC，然后与 IC 设计公司签约，保证只售你们一家。您就获得了法律上的保障。

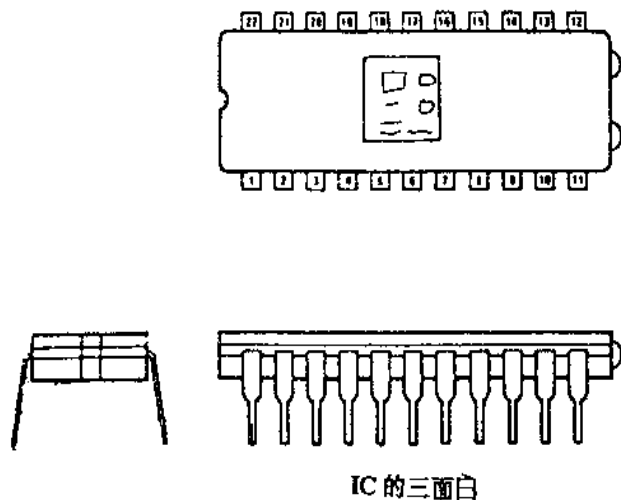
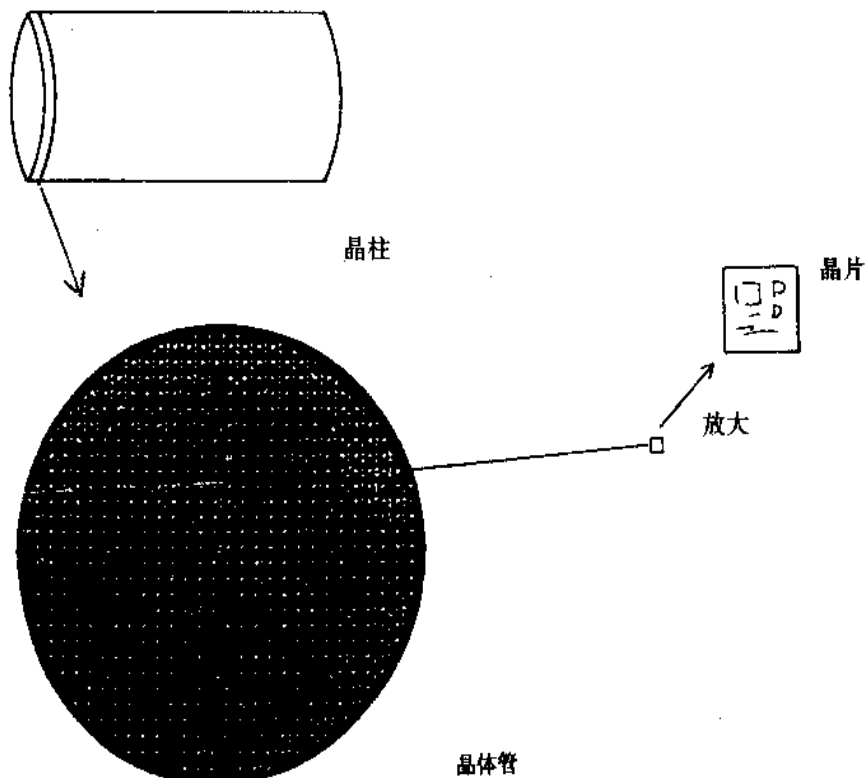
### 何谓 Reverse And Duplicate Engineer? R & D Engineer

再复杂的 IC 也有一定的轨迹可以追寻，十余年前台湾的 IC 刚起步，有些厂家将市面上的 IC 成品打开，利用化学药品打开塑料部分再加以照像，将一张张的照片组合后，加以反跟踪 (Reverse) 再加以复制 (Duplicate)，成为一个新的 IC。这一些工程师常戏称自己为此形式的 R & D Engineer。其实 R & D 的原义是 Research and Development。

何谓芯片(wafer)?何谓晶体管? 何谓 IC?

在 IC 制造过程中, 通常的起始材料为矿晶, 矿晶通常做成圆柱状, 再加以切片而成为芯片。晶体管就是这片芯片上研制而成的, 一般而言有 4 寸和 6 寸的。每片芯片能容纳的晶体管数按它的面积大小而有所不同。

何谓标准 IC? 何谓客户型的 IC?



IC 的三面图

简单而言，在市面上所见的 IC 都是标准型的 IC，譬如 TTL 门 74 系列或 54 系列，这些 IC 均可在 Data Book 中找到相关数据。而定制型的 IC 则是 IC 设计公司所设计的，这种特殊功能 IC 定制所强调的即是它的保密性。

## IC 开发过程

### 1. 市场调查

对一个新 IC 的开发，往往需要很长的时间作市场调查。对于复杂的 IC，此时还需要做系统模拟。公司还需估算出 IC 的获利率，以决定是否要继续开发。

### 2. 定出规格

市场部的市场工程师必须与将来的可能客户合定规格，一般 IC 的规格包含功能说明及电性特性。此时 IC 设计工程师也会参与讨论。

### 3. 设计线路

IC 线路的设计随着计算机的进步，已经不如以前复杂，现今的软件已有能力将逻辑真值表直接利用计算机转换成线路。未来的 IC 设计工程师可能只需书写“电路产生语言”，计算机即可将电路图设计出来。线路设计最困难的部分在于线路的稳定性，由于 IC 制作过程(Process)参数不易控制，往往 IC 将不能正常工作。所以线路的设计最困难的部分就是要设计成过程独立的特征(Process independent)，也就是让电路不受制作过程参数(Process Parameter)影响。

### 4. 线路模拟

在线路完成之后，通常都须要利用计算机系统模拟。这种模拟可在 Apollo 上或者 VAX 系统上进行，小的线路可以在 PC 上做，这部分的工作目的在于验证线路的正确性。由于制作过程(Process)参数并不甚准确，以及 IC 元件会随温度的变化而产生不同的特性，所以这一类的模拟并不百分之百可靠。线路的模拟有两方面：

① 数字电路：利用逻辑模拟软件。

② 模拟电路：利用 Spice。

### 5. 布局

布局对 IC 而言非常重要，如果布局不好不但会影响面积大小，甚至于因为有些电路因线路太长造成时间延迟，以致于 IC 不工作。对于精密、重要的计划，有时候必须做 Post Layout Simulation (布局后模拟)。

### 6. 设计规则(DRC)、电性规则(ERC)、布局对线路(LVS)检查。

在布局完之后，通常需要做 DRC,ERC,LVS 检查。

DRC：用以检查布局是否合乎制造过程要求，此检查目的用以提高良品率。

ERC：用以检查是否有些信号是否异常。在布局中往往因为疏忽会将一些信号接错，ERC 有助于将此类问题找出。

LVS：用以验证布局是否和线路完全吻合。

### 7. 完成工作

在布局之后，IC 设计公司针对工厂要求将封装时所需要的切割道，以及封装时打线用的辩识方位 Pattern 设计入内。

### 8. 送 Tape

将 GDS II 形式的 Tape 送给工厂做掩模，开始制造。

#### 9. 封装、测试

芯片做完之后还需要封装、测试，最后送入客户手中。

市场部人员，则必须对产品推广，并将应用方向和方式告诉客户。

# 希 望 汉 字 系 统

UCDOS Ver 3.0

- ▲支持直接写屏，中西文兼容，英文制表符识别效果最佳，如实再现原版软件的神奇风采
- ▲充分利用扩展内存，可实现零字节占用，不受DOS版本限制
- ▲配备WPS文字处理系统，可在网络环境下正常运行，并可同时使用26种矢量字库进行打印
- ▲真正实现网络共享，工作站数目不受限制，彻底解决共享打印和远程通讯
- ▲矢量汉字打印速度奇快，无与伦比
- ▲可在各种图形模式下显示任意大小的文字，并具有屏幕作图、背景音乐和图像保存恢复功能
- ▲提供多种汉字输入方法(含五笔)，记忆词组方便灵活，可在多种输入法中随意使用

单用户版市场零售价：¥880.00元    网络版市场零售价¥2000.00元

地 址：北京市海淀路82号8721信箱软件部	联系人：张军 夏克 陈江
电 话：01-2579873, 8422024, 8422025	传 真：01-2561057
开 户：工商银行海淀分理处	帐 号：661924-61
户 名：北京希望电脑公司	

## 欢迎加入希望用户协会

当今的计算机技术发展迅猛,应用领域繁多,如何准确、有效、地为您提供服务,已成为我们迫切关心的问题。为此,我们决定创立希望用户协会,其宗旨在于加强与用户的联络,了解用户的各种信息与需求,为用户提供更完善更周到的服务。

本协会的成员将定期获得各种软件、资料与培训等讯息,优先得到有关技术服务。请您认真填写后面的会员登记表,其中的信息将用电脑进行管理。

作为美国Borland软件公司在中国的总代理,北京希望电脑公司经Borland公司授权,对其产品dBASE IV 2.0 进行系统级汉化。另外,对Borland C++ & AF 3.1、Turbo C++ for DOS 3.0、Turbo C++ for Windows、Turbo C++ Visual Edition for Windows四个产品进行了本地化。预计94年3月推出以上产品。

Borland 软件系列清单:

1. Borland C++ & AF 3.1
2. Borland C++ 3.1
3. Turbo C++ for DOS 3.0
4. Turbo C++ for Windows
5. Paradox for DOS 4.0
6. Paradox for Windows
7. Pdox Engine & Database Framework 3.0
8. Turbo Pascal for DOS 7.0
9. Turbo Pascal for Windows 1.5
10. dBASE IV 2.0
11. dBASE IV Compiler
12. Object Vision 2.1

### 软件与培训

北京希望电脑公司即将与Borland公司密切合作,经销Borland多种软件产品的同时,提供配套的函授与培训服务,以满足用户的不同要求。欢迎用户索取软件与培训资料。

### 联系方法

有关资料事宜,请与朱红小姐联系;有关软件事宜,请与周东先生联系;有关函授与培训事宜,请与宋明华先生联系。联系方法如下:

通信地址: 北京 8721 信箱      邮政编码: 100080

联系电话: (01)2562329 (01)2541992 (01)2579874

传真电话: (01)2561057



## 北京希望电脑公司推出松岗电脑系列图书

序号	书 名	售 价
1.	SCO UNIX 入门	11.00 元
2.	经典 C 语言习题解答	17.00 元
3.	PC TOOLS 8.X 实用指南	28.00 元
4.	Borland C++ 软件集成技术与范例(含盘)	59.00 元
5.	中文 Microsoft Excel (V4.0) 入门	29.00 元
6.	Visual Basic for Windows 程序设计	39.00 元
7.	Borland C++/Turbo C++ 编程实例剖析(含盘)	49.00 元
8.	Windows 程序设计绘图篇(含盘)	88.00 元
9.	Turbo C 学习指南(含盘)	39.00 元
10.	数据结构——使用 C 语言	29.00 元
11.	Windows 3.1 使用手册	35.00 元
12.	Visual Basic 问题精粹集(含盘)	39.00 元
13.	计算机系统原理及试题集	39.00 元
14.	面向对象的游戏、图形与动画(含盘)	49.00 元
15.	如何用 Borland C++ 设计 Windows 应用程序(含盘)	49.00 元
16.	看图例学 Microsoft Windows 3.1	39.00 元
17.	中文 Windows 入门	21.00 元
18.	Windows 绘图软件 CorelDRAW 入门与使用指南	12.00 元
19.	AutoCAD(R11)问题集	33.00 元
20.	易学 C 语言基础	25.00 元
21.	AutoCAD 12.0 使用手册	89.00 元
22.	AutoCAD 12.0 使用技巧(含盘)	59.00 元

欢迎广大新老朋友选购

通信地址:北京 8271 信箱      邮政编码:100080

联系电话:(01)2562329,(01)2541992

传真:(01)2561057