第1章　　微机概述

1.1　CPU 是英文_____的缩写，中文译为_____，微型机采用_____芯片构成 CPU。

Central　Processing Unit　　中央处理单元　　超大规模集成电路

1.2　什么是通用微处理器、单片机（微控制器）、DSP 芯片、嵌入式系统？

- 通用微处理器：冯-诺伊曼结构中的运算器和控制器，有基本的指令处理和执行功能，通过总线可与内存和外设通信。
- 单片机（微控制器）：将运算器、控制器和基本内存制作在一块芯片上，外接少量电路即可完成强大功能的计算机系统。
- DSP 芯片：采用哈佛结构，即指令和数据分开存储，并有相应的总线，以便于处理大量的数据，常用于数字信号处理。
- 嵌入式系统：将冯-诺伊曼结构的五大部件：运算器 控制器 存储器 输入输出电路制作在一块芯片中，可方便使用的微机系统。

1.3　什么是摩尔定律？它能永久成立吗？

每隔十八个月，计算机的芯片集成度会提高一倍，功能提高一倍，而价格则下降为一半。

1.4　冯-诺伊曼计算机的基本设计思想是什么？

- 采用二进制形式表示数据和指令，指令由操作码和地址码组成
- 将程序和数据存放在存储器中，计算机在工作时从存储器中取出指令执行，自动完成
- 指令的执行是顺序进行的，即一般按照指令在存储器中存放的顺序执行，程序分支由转移指令实现
- 计算机由存储器、运算器、控制器、输入设备和输出设备五大基本部件组成，并具有各自相应的功能。

1.5　说明微型计算机系统的硬件组成及各部分作用。

- 微型计算机由运算器、控制器、存储器、输入设备和输出设备五大部分组成。
- 其中存储器又分内存储器、外存储器；通常把输入设备及输出设备称为外围设备；运算器和控制器合称为中央处理器——CPU(Central Processing Unit)。 存储器的主要功能是存放程序和数据，中央处理器的主要功能是执行存储器内的程序，输入设备的任务是把用户要求计算机处理的数据、字符、文字、图形和程序等各种形式的信息转换为计算机所能接受的编码形式存入到计算机内、并进行处理。输出设备的任务是把计算机的处理结果以用户需要的形式(如屏幕显示、文字打印、图形图表、语言音响等）输出。输入输出接口是外部设备与中央处理器之间的缓冲装置，负责电气性能的匹配和信息格式的转换。
- 也可以简单地说计算机由硬件和软件组成。

1.6　什么是总线？微机总线通常有哪 3 组信号？各组信号的作用是什么？

- 总线（Bus）是计算机各功能部件之间传送信息的公共通信干线，它是由导线组成的传输线束。按照计算机所传输的信息种类，计算机的总线可以划分为数据总线、

地址总线和控制总线，分别用来传输数据、地址和控制信号。

- 总线使用特点之一，某一时刻只能由一个总线主控设备来控制系统总线；特点之二，在连接总线的多个设备中，向总线发送信号只能有一个设备，而从总线上读取信号的设备可以不止一个。

1.7 简答如下概念：

（1） 计算机字长：

CPU 在一个时钟周期内能够处理的最多的二进制的位数，

（2） 取指—译码—执行周期

CPU 执行一条指令时，首先从内存读取指令，在读取下一条指令的同时，对当前指令进行译码，并送相应部件执行，其所花费的时间为取指—译—执行周期

（3） ROM-BIOS

BIOS, Basic Input/Output System 基本输入输出系统，通常固化在只读存储器 ROM 中，可完成计算机的初始化工作，提供基本的功能调用。

（4） 中断

CPU 与外设的一种数据传送方式，在外设申请被 CPU 响应后，CPU 会停下当前工作，执行相应的中断服务程序。

（5） ISA 总线

Industrial Standard Architecture 工业标准体系结构总线，又称 PC 总线，首先用于 PC/AT 微机，有 98 根线，数据 16 位，用于 CPU 与外设交换数据。现已淘汰。

1.8 下列十六进制数表示无符号数，请转换为十进制形式的数值：

（1）FFH （2）0H （3）5EH （4）EFH

1.9 将下列十进制数真值转换为压缩 BCD 码

（1）12 （2）24 （3）68 （4）99

1.10 将下列压缩 BCD 码转换为十进制数：

（1）10010001 （2）10001001 （3）00110110 （4）10010000

1.11 将下列十进制数用 8 位二进制补码表示：

（1）0 （2）127 （3）—127 （4）—57

1.12 数码 0—9、大写字母 A—Z、小写字母 a—z 对应的 ASCII 码分别是多少？ASCII 码 0DH 和 0AH 分别对应什么字符？

无符号数	十进制数	十进制真值	压缩 BCD 码	压缩 BCD 码	十进制真值	十进制数	8 位二进制补码
FFH	255	12	0001 0010	1001 0001	91	0	0000 0000
0H	0	24	0010 0100	1000 1001	89	127	0111 1111
5EH	94	68	0110 1000	0011 0110	36	-127	1000 0001
EFH	239	99	1001 1001	1001 0000	90	-57	1100 0111

第2章　　指令系统

2.1 微处理器内部有哪 3 个基本部分？8088 分为哪两大功能部件？其各自的主要功能是什么？这种结构与 8 位 CPU 相比为什么能提高其性能？

解：微处理器内部的 3 个基本部分为：算术逻辑单元 ALU、寄存器组和控制器；

8088 的两大功能部件 BIU 和 EU。各自主要功能为：

总线接口单元 BIU：管理 8088 与系统总线的接口负责 CPU 对接口和外设进行访问

执行单元 EU：负责指令译码、执行和数据运算；

与 8 位 CPU 相比的优点是：8088 不需要等待取指令。8088 可以将需要译码的指令预先取到指令队列，8 位 CPU 在指令译码前必须等待取指令操作的完成。而取指令是 CPU 最为频繁的操作，因此 8088 的结构和操作方式节省了大量等待时间，比 8 位 CPU 节省了时间，提高了性能。

2.2 说明 8088 的 8 个 8 位和 8 个 16 位通用寄存器各是什么？

解：8 个 8 位寄存器：AH、AL、BH、BL、CH、CL、DH、DL；

8 个 16 位寄存器：累加器 AX、基址寄存器 BX、计数器 CX、数据寄存器 DX、源地址寄存器 SI、目的地址寄存器 DI、基址指针 BP、堆栈指针 SP。

2.3 什么是标志？状态标志和控制标志有什么区别？画出标志寄存器 FLAGS，说明各个标志的位置和含义。

解：标志用于反映指令执行结果或者控制指令执行形式。

状态标志用于记录程序运行结果的状态信息；控制标志用于控制指令执行的形式。

8088/8086 有一个 16 位的标志寄存器，也称为程序状态字，分为 6 个状态标志和 3 个控制标志。其首字母可形成一句话：IS PC A DOT? （0 为 No，1 为 Yes，故答 Zero）。

也可依从低位到高位的顺序记忆为：**CPA**（注册会计师考试）考了零（**Z**ero）分，作为学生（**ST**udent），义不容辞，只好我来做了（**I DO**）。

Carry **进位** / **P**arity **偶** / **A**uxiliary **辅助（进位）** / **Z**ero **零**

Sign **符号** / **T**rap **陷阱** / **I**nterruption **中断** / **D**irection **方向** / **O**verflow **溢出**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				O	D	I	T	S	Z		A		P		C

2.4 举例说明 CF 和 OF 标志的差异。

解：进位（Carry）：表示无符号数运算结果是否超出范围，有则 CF=1，否则 CF=0

溢出（Overflow）：表示有符号数运算结果是否超出范围，有则 OF=1，且结果错误，否则 OF=0

以 8 位机器数为例。字节相加：3AH+7CH=B6H。若认为是无符号数，58+124=182，在 0~255 范围内，没有产生进位，CF=0，若认为是有符号数，58+124=182，已经超出 -128~+127 范围，产生溢出，所以 OF=1；另一方面，B6H 作为有符号数的补码表达真值是-74，显然运算结果也不正确。

此例中，其余标志位：ZF=0, SF=1, PF=0, AF=1

2.5 什么是 8088 中的逻辑地址和物理地址？逻辑地址如何转换成物理地址？1MB 最多能分成多少个逻辑段？请将如下逻辑地址用物理地址表示：

（1）FFFFH：0　（2）40H：17H　（3）2000H：4500H　（4）B821H：4567H

解：物理地址：8088 的 20 根地址线直接寻址的存储单元编号，范围是 00000~FFFFFH。

逻辑地址：由段基地址和段内偏移地址组成。

段地址左移 4 位加上段内偏移地址可得实际的物理地址。

由于 CPU 内寄存器及算术逻辑运算均以 16 位进行，所以 1MB 存储空间被分成许多逻辑段来管理。最多可分成 $2^{20} \div 2^4 = 2^{16} = 64K$ 个逻辑段。

（1）FFFF：0　→FFFF0H

（2）40H：17H　→00417H

（3）2000：4500H　→24500H

（4）B821H：4567H　→BC777H

2.6 8088 有哪 4 种逻辑段，各种逻辑段分别是什么用途？

代码段：存放程序的指令序列；

堆栈段：确定堆栈所在的主存储区；

数据段：存放当前运行程序的数据；

附加段：附加数据段，用于数据保存。另外串操作指令将其作为目的操作数的存放区。

2.7 什么是有效地址 EA？8088 的操作数如何在主存中，有哪些寻址方式可以存取它？

8088 的存储空间分段管理，程序设计时采用逻辑地址。由于段地址在默认的或指定的段寄存器中，所以只需要偏移地址，称为有效地址 EA.

操作数在主存中有以下几种寻址方式：直接寻址、寄存器间接寻址、寄存器相对寻址、基址变址寻址、相对基址变址寻址。

2.8 已知 DS=2000H，BX=0100H，SI=0002H，存储单元[20100H] — [20103H]依次存放 12H、34H、56H、78H，[21200H] — [21203H]依次存 2AH、4CH、B7H、65H，说明下列每条指令执行完后 AX 寄存器的内容以及源操作数的寻址方式？

（1）MOV　AX，　1200H

（2）MOV　AX，　BX

（3）MOV　AX，　[1200H]

（4）MOV　AX，　[BX]

（5）MOV　AX，　[BX+1100H]

（6）MOV　AX，　[BX+SI]

（7）MOV　AX，　[BX][SI+1100H]

(1) ax=1200h	立即数寻址
(2) ax=0100h	寄存器寻址
(3) ax=4C2A h	存储器直接寻址
(4) ax=3412h	存储器间接寻址
(5) ax=4C 2Ah	存储器相对寻址
(6) ax=7856h	存储器基址变址寻址
(7) ax=65B7h	相对基址变址寻址

2.9 说明下列各条指令的具体错误原因

（1）MOV　CX，　DL

（2）MOV　IP，　AX

（3）MOV　ES，　1234H

（4）MOV　ES，　DS

（5）MOV　AL，　300

（6）MOV　[SP]，　AX

（7）MOV　AX，　BX+DI

（8）MOV　20H，　AH

(1) 操作数字长不一致 (2) 指令指针 IP 不能被赋值 (3) 立即数不能送给扩展段 ES (4) 段寄存器间不能相互传送数值 (5) 300 超过 AL 的数据范围 (6) 堆栈指针 SP 不能用于寄存器间接寻址 (7) bx + di 语句格式不对，应加括号 (8) 20H 为立即数，不能作为目的操作数

解：错误原因分别为

2.10 已知数字 0—9 对应的格雷码依次为：18H、34H、05H、06H、09H、0AH、0CH、11H，12H、14H，它存在于又 table 为首地址（设为 200H）的连续区域中。请为如下程序段的每条指令加上注释，说明每条指令的功能和执行结果。

```
LEA     BX,     TABLE           ；获取 table 的首地址，BX=200H
mov     al,  8                  ；传送欲转换的数字，AL=8
xlat                            ；转换为格雷码，AL=12H
```

2.11 给出下列各条指令执行后的 AL 值，以及 CF、ZF、SF、OF 和 PF 的状态：

```
1   MOV     AL,     89H
2   ADD     AL,     AL
3   ADD     AL,     9DH
4   CMP     AL,     0BCH
5   SUB     AL,     AL
6   DEC     AL
7   INC     AL
```

（1）al=89H
（2）al=12H,cf=1,zf=0,sf=0,of=1,pf=1
（3）al=0AFH,cf=0,zf=0,sf=1,of=0,pf=1
（4）al=0AFH,cf=1,zf=0,sf=1,of=1,pf=1
（5）al=0,cf=0,zf=1,sf=0,of=0,pf=1
（6）al=0FFH,cf=0,zf=0,sf=1,of=0,pf=1
（7）al=0,cf=0,zf=1,sf=0,of=0,pf=1

2.12 请分别用一条汇编语言指令完成如下功能：

（1）把 BX 寄存器和 DX 寄存器的内容相加，结果存入 DX 寄存器。

（2）用寄存器 BX 和 SI 的基址变址寻址方式把存储器的一个字节与 AL 寄存器的内容相加，并把结果送到 AL 中

（3）用 BX 和位移量 0B2H 的寄存器相对寻址方式把存储器中的一个字和 CX 寄存器的内容相加，并把结果送回到存储器中。

（4）用位移量为 0520H 的直接寻址方式把存储器中的一个字与数 3412H 相加，并把结果送回到该存储单元中。

（5）把数 0A0H 与 AL 寄存器的内容相加，并把结果送回 AL 中。

解：(1) add dx, bx

(2) add al, [bx+si]

(3) add word ptr[bx+0B2H], cx

(4) add word ptr[0520H],3412H

(5) add al, 0A0H

2.13 设 X、Y、Z 和 V 均为 16 位带符号数，分别装在 X、Y、Z、V 存储单元中，阅读如下程序段，得出它的运算公式，并说明运算结果存于何处。

```
mov     ax,  X
imul    Y
mov     cx,  ax
mov     bx,  dx
mov     ax,  Z
cwd
add     cx,  ax
adc     bx,  dx
sub     cx,  540
sbb     bx,  0
mov     ax,  V
cwd
sub     ax,  cx
sbb     dx,  bx
idiv    V
```

运算公式：
$(V - Z - X * Y + 540) / X$。
商存储在 ax，余数存储在 dx

2.14 给出下列各条指令执行后的结果，以及状态标志 CF、OF、SF、ZF、PF 的状态

```
mov     ax,  1470h
and     ax,  ax
or      ax,  ax
xor     ax,  ax
not     ax
test    ax,  0f0f0h
```

解：如表所示

AX=1470h	CF	OF	SF	ZF	PF
AX=1470h	0	0	0	0	0
AX=1470h	0	0	0	0	0
AX=0000H	0	1	0	0	1
AX=FFFFH	0	1	0	0	1
AX=FFFFH	0	0	1	0	1

2.15 控制转移类指令中有哪三种寻址方式？

相对寻址方式

直接寻址方式

间接寻址方式。

2.16 假设 DS = 2000H，BX = 1256H，TABLE 的偏移地址是 20A1H，物理地址 232F7H 处存放 3280H，试问执行下列段内间接寻址的转移指令后，转移的有效地址是什么？

（1）JMP　　BX

（2）JMP　　TABLE[BX]

解：（1）1256H

（2）3280H

2.17 判断下列程序段跳转的条件：

```
（1）xor     ax,  1e1eh
     je      equal
（2）test    al,  1000 0001b
     jnz     there
（3）cmp     cx,  64h
     jb      there
```

（1）ax=1E1EH
（2）al 的 D0、D7 位不全为 0
（3）cx<64H

2.18 如下是下段软件延时程序，请问 NOP 指令执行了多少次？

```
        xor     cx,  cx
delay:  nop
        loop    delay
```

解：FFFFH ~ 0000H = 2^{16} = 65536 次。

2.19 有一个首地址为 array 的 20 个字的数组，说明下列程序段的功能。

```
        mov     cx,  20
        mov     ax,  0
        mov     si,  ax
sumlp:  add     ax,  array[si]
        add     si,  2
        loop    sumlp
        mov     total,   ax
```

解：20 个数组元素按**字**累加求和。结果存入 total 存储单元中。

2.20 按照下列要求，编写相应的程序段：

（1）由 string 指示的起始地址的主存单元中存放一个字符串（长度大于 6），把该字符串的第 1 个和第 6 个字符（字节量）传送给 DX 寄存器

（2）有两个 32 位数，按“小端方式”存放在两个缓冲 buffer1 和 buffer2 中，编写程序段完成 DX.AX ←buffer1←buffer2 功能

（3）编写一个程序段，在 DX 高 4 位全为 0 时，使 AX = 0；否则，使 AX = --1。

（4）把 DX.AX 中的双字右移 4 位。

（5）有一个 100 个字节元素的数组，其首地址为 array，将每个元素减 1（不考虑溢出或借位）存于原处。

```
(1) lea   si,  string
    Mov   dh,  [si]
    Mov   dl,  [si+5]
(2) mov   ax，word ptr buffer1
    Sub   ax，word ptr buffer2
    Mov   dx，word ptr buffer1+2
    Sbb   dx，word ptr buffer2+2
(3)  test  dx,  0F000H
     jz    even
     mov    ax, 0ffffh
even: mov  ax,  0
(4)  mov   cx,  4
again: sar  dx,  1
     rcr   ax,  1
     loop  again
(5)  mov   cx,  100
     mov   si,  0
again: mov  al,  0FFH
     add   al,  array[si]
     mov   array[si],al
     inc   si
```

2.21 AAD 指令是用于除法指令之前，进行非压缩 BCD 码调整的。实际上，处理器的调整过程是：AL←AH*10 + AL,AH←0。如果指令系统没有 AAD 指令，请用一个子程序完成这个调整工作。

```
Htoasc  proc
        Mov     bl,  al
        Mov     al,  ah
        Mov     bh,  10
        Mul     bh
        And     ax,  00FFH
        Add     al,  bl
Htoend:  ret
Htoasc  endp
```

2.22 什么是系统功能调用？在汇编语言中，调用系统功能的一般步骤是什么？

计算机系统利用中断为用户提供硬件设备驱动程序。在 IBM-PC 系列微机中，基本输入输出系统 ROM-BIOS 和 DOS 都提供了丰富的中断服务程序，称为系统功能调用。

调用步骤

（1）AH 中设置系统功能调用号

（2）在指定寄存器中设置入口参数

（3）使用中断调用指令执行功能调用

（4）根据出口参数分析调用情况

2.23 DAA 指令的调整操作是：

（1）如果 AL 的低 4 位是 A ~ F，或者 AF 标志为 1，则 AL←AL + 6，且使 AF = 1；

（2）如果 AL 的高 4 位是 A ~ F，或者 CF 标志为 1，则 AL←AL + 60H，且使 CF = 1；

阅读如下子程序，说明它为什么能够实现 AL 低 4 位表示的一位 16 进制数转换成对应的 ASCII 码，并且将该程序加上在屏幕显示的功能，编写成通用的子程序。

```
Htoasc    proc
          And al,0FH
          Add al,90H
          Daa
          Adc al,40H
          Daa
          Mov ah,02H
          Mov dl,al
          Int 21H
          Ret
Htoasc    endp
```

2.24 乘法的非压缩 BCD 码调整指令 AAM 执行的操作是：AH←AL/10 的商，AL←AL/10 的余数。利用 AAM 可以实现将 AL 中的 100 以内的数据转换为 ASCII 码，程序如下：

```
xor     ah,     ah
aam
add     ax,     3030h
```

利用这段程序，编写一个显示 AL 中数值（0 ~ 99）的子程序。

```
numout  proc
        xor ah,   ah
        aam
        add ax,   3030h
        mov dl,   ah
        mov ah,   02h
        int 21h
        mov dl,   al
        mov ah,   02h
        int 31h
        ret
numout  endp
```

2.25 编写一个程序段，先提示输入数字：“Input Number: 0 ~ 9”，然后在下一行显示输入的数字，结束：如果不是键入了 0 ~ 9 数字，就提示“Error!”，继续等待输入数字。

```
msgkey  db”input number 0-9”,”$”
msgwrg  db ”error”,    ”$”
        mov     ah,  09h
        mov     dx,  offset msgkey
        int     21h
again:  mov     ah,  01h
        int     21h
        cmp     al,  30h
        jb  disp
        cmp     al,  39h
        ja  disp
        mov     dl,  al
        mov     ah,  02h
        int     21h
        jmp     done
disp:   mov     dx,  offset msgwrg
        mov     ah,  09h
        int     21h
        jmp     again
done:   mov     ah,  02h
        mov     dl,  al
        int     21h
```

第3章　　汇编语言

3.1 汇编语言有什么特点?

汇编语言是一种以处理器指令系统为基础的低级程序设计语言,它采用助记符表达指令操作码，采用标识符号表示指令操作数，可以直接、有效地控制计算机硬件，因而容易创建代码序列短小、运行快速的可执行程序。作为一种低级语言,汇编语言编程较繁琐，但在病毒防护、加密与解密等方面，汇编语言的作用是不容置疑和不可替代的。

3.2 编写汇编语言源程序时，一般的组成原则是什么？

- 完整的汇编语言源程序由段组成
- 一个汇编语言源程序可以包含若干个代码段、数据段、附加段或堆栈段，段与段之间的顺序可随意排列
- 需独立运行的程序必须包含一个代码段，并指示程序执行的起始点，一个程序只有一个起始点
- 所有的可执行性语句必须位于某一个代码段内,说明性语句可根据需要位于任一段内
- 通常，程序还需要一个堆栈段

3.3 .MODEL 伪指令是简化段定义源程序格式中必不可少的语句,它设计了哪 7 种存储模式，各用于创建什么性质的程序？

存储模式	特 点
TINY	COM 类型程序，只有一个小于 64KB 的逻辑段（MASM 6.x 支持）
SMALL	小应用程序，只有一个代码段和一个数据段（含堆栈段），每段不大于 64KB
COMPACT	代码少、数据多的程序，只有一个代码段，但有多个数据段
MEDIUM	代码多、数据少的程序，可有多个代码段，只有一个数据段
LARGE	大应用程序，可有多个代码段和多个数据段（静态数据小于 64KB）
HUGE	更大应用程序，可有多个代码段和多个数据段（对静态数据没有限制）
FLAT	32 位应用程序，运行在 32 位 80x86CPU 和 Windows 9x 或 NT 环境

3.4 如何规定一个程序执行的开始位置，主程序执行结束应该如何返回 DOS，源程序在何处停止汇编过程？

- 为了指明程序开始执行的位置，需要使用一个标号。CS、IP、SS、SP 可自动设置，但 DS 和 ES 需在程序最开始进行规定。
- 应用程序执行结束，应将控制权交还操作系统。有如下几种返回 DOS 的方法:
  1） 利用 DOS 功能调用的 4CH 子功能;
  2） 调用 20H 号中断
  3） MASM 6 以上版本，.EXIT 语句可自动完成返回 DOS 功能
- 源程序在程序的最后利用 END 伪指令停止汇编。

3.5 MASM 为什么规定十六进制常数不能用字母 A – F 开头？

MASM 规定十六进制常数以 0 – 9 开头，或以 0A、0B、0C、0D、0E、0F 开头。不能以字母 A – F 开头，以免与常量或变量相混淆。

3.6 给出采用一个源程序格式书写的例 3-1 源程序。

```
stack     segment   stack
```

```
        db      1024(0)
stack   ends
data    segment
        string  db   'Hello,Assembly！'，0dH，0aH，‘$’
data    ends
code    segment  'code'
        assume   cs:code,ds:data,ss:stack
```

3.7 DOS 支持哪两种执行程序结构，编写这两种程序时需要注意什么？

1) .EXE 程序

程序可以有多个代码段和多个数据段，程序长度可以超过 64KB

通常生成 EXE 结构的可执行程序

2) .COM 程序

只有一个逻辑段，程序长度不超过 64KB

需要满足一定条件才能生成 COM 结构的可执行程序（MASM 6.x 需采用 TINY 模式）

3.8 举例说明等价“EQU”伪指令和等号“=”伪指令的用途。

符号定义伪指令有“等价 EQU”和“等号＝”:

符号名 EQU 数值表达式

符号名 EQU <字符串>

符号名 ＝ 数值表达式

EQU 用于数值等价时不能重复定义符号名，但“＝”允许有重复赋值。例如:

X= 7 ；等效于: X equ 7

X= X+5 ；“X EQU X+5”是错误的

3.9 给出下列语句中，指令立即数（数值表达式）的值：

（1）mov al, 22h AND 45h OR 67h

（2）mov ax, 1234h/16 + 10h

（3）mov ax, 23h SHL ‘’4

（4）mov al, ‘a’ AND (NOT (‘a’ – ‘A’))

（5）mov ax, (76543 LT 32768) XOR 7654h

（1） al=67h
（2） ax=133h,dx=4h
（3） ax=0230h
（4） al=41h
（5） ax=7654h

3.10 画图说明下列语句分配的存储区间及初始化的数据值

（1）byte_var db ‘ABC’ , 10 , 10H, ‘EF’ , 3 dup(--1, ?, 3 dup(4))

’A’	’B’	’C’	10	10H	’E’	’F’	-1	?	4	4	4	-1	?	4	4	4	-1	?	4	4	4

（2）word_var dw 10h, --5, 3 dup(?)

10H	00H	FB	00	?	?	?	?	?	?

3.11 请设置一个数据段，按照如下要求定义变量：

(1) my1b 为字符串变量，表示字符串"Personal Computer".
(2) my2b 为用十进制数表示的字节变量，这个数的大小为 20
(3) my3b 为用十六进制数表示的字节变量，这个数的大小为 20
(4) my4b 为用二进制数表示的字节变量，这个数的大小为 20
(5) my5w 为 20 个未赋值的字变量
(6) my6c 为 100 的符号常量
(7) my7c 为字符串常量，代替字符串"Personal Computer".

```
.data
my1b    db   'Personal   Computer'
my2b    db   20
my3b    db   14h
my4b    db   00010100b
my5w    dw   20 dup(?)
my6c    =    100
my7c    =    <'Personal   Computer'>
```

3.12 希望控制变量或程序代码在段中的偏移地址，应该使用哪个伪指令？

- 利用定位伪指令控制，如 org, even, align

3.13 名字和标号有什么属性？

- 包括逻辑地址和类型两种属性。

3.14 设在某个程序中有如下片段，请写出每条传送指令执行后寄存器 AX 的内容：

```
                ；数据段
                org 100h
varw            dw   1234h, 5678h
varb            db 3, 4
vard            dd   12345678h
buff            db   10   dup(?)
mess            db   'hello'
                ；代码段
                mov ax,   offset     mess
                mov ax,   type buff + type mess + type vard
                mov ax,   sizeof varw + sizeof buff + sizeof mess
                mov ax,   lengthof varw + lengthof vard
```

3-14 解：
AX=114H
AX=1+1+4=6
AX=4+10+5=13H
AX=2+1=3

3.15 假设 myword 是一个字变量，mybyte1 和 mybyte2 是两个字节变量，指出下列语句中的具体错误原因。

（1）mov byte ptr [bx], 1000
（2）mov bx, offset myword[si]
（3）cmp mybyte1, mybyte2
（4）mov mybyte1, al + 1
（5）sub al, myword
（6）jnz myword

（1）1000 超过一个字节所能表达的最大整数
（2）SI 应为偶数
（3）两个内存单元不能直接运算
（4）应改为[al+1]
（5）类型不匹配。AL 为字节，myword 为字。
（6）条件转移指令后面应接标号，而不是变量

3.16 编写一个程序，把从键盘输入的一个小写字母用大写字母显示出来。

```
mov ah,1            ; 只允许输入小写字母
int 21h
sub al,20h          ; 转换为大写字母
mov dl,al
mov ah,2
int 21h             ; 显示
```

3.17 已知用于 LED 数码管的显示代码表为：

```
LEDtable    db   0c0h, 0f9h, 0a4h, 0b0h, 99h, 92h, 0f8h
2           db   80h, 90h, 88h, 83h, 0c6h, 0c1h, 86h, 8eh
```

它依次表示 0 ~ 9、A ~ F 这 16 个数码的显示代码。现编写一个程序实现将 lednum 中的一个数字（0 ~ 9、A ~ F）转换成对应的 LED 显示代码。

```
;程序如下，可参考第 2 章习题 2-10
mov bx, offset LEDtable
mov al, lednum
xlat
```

3.18 编写一个程序，把变量 bufX 和 bufY 中较大者存入 bufZ；若两者相等，则把其中之一存入 bufZ 中，假设变量存放的是 8 位有符号数。

```
        ; 程序如下
        mov     ax,  bufX
        cmp     ax,  bufY
        jae     done
        mov     ax,  bufY
done:   mov     bufZ, ax
```

3.19 设变量 bufX 为有符号 16 位数，请将它的符号状态保存在 signX，即：如果变量值大于等于 0，保存 0；如果 X 小于 0，保存—1。编写该程序。

```
        ; 程序如下
        .model small
        .stack
        .data
        bufX    dw -7
        signX   db ?
        .code
        .startup
        cmp     bufX,   0    ;test bufX,80h
        jl      next         ;jnz next
        mov     signX,  0
        jmp     done
next:   mov signX,-1
done:   .exit 0
        end
```

3.20 bufX，bufY 和 bufZ 是 3 个有符号 16 进制数，编写一个比较相等关系的程序；

（1）如果这 3 个数都不相等，则显示 0；

（2）如果这 3 个数中有两个数相等，则显示 1；

（3）如果这 3 个数都相等，则显示 2。

```
        ；程序如下
        mov     dl, ’2’
        mov     ax, bufX
        cmp     ax, bufY
        je      next1
        dec     dl
next1:  cmp     ax, bufZ
        je      next2
        dec     dl
next2:  mov     ah, 2
        int     21h
```

3.21 例 3-7 中，如果要实现所有为 1 的位都顺序执行相应的处理程序段（而不是例量中仅执行最低为 1 位的处理程序段），请写出修改后的代码段。

```
        ；程序如下
        ;代码段
        mov     al, number
        mov     bx, 0           ;BX←记录为 1 的位数
restart: cmp    al, 0           ;AL＝0 结束
        jz      done
again:  shr     al, 1           ;最低位右移进入 CF
        jc      next            ;为 1，转移
        inc     bx              ;不为 1，继续
        jmp     again
next: push      ax
        push    bx
        shl     bx, 1           ;位数乘以 2（偏移地址要用 2 个字节单元）
        jmp     addrs[bx]       ;间接转移：IP←[table＋BX]
        ;以下是各个处理程序段
fun0:   mov     dl, '0'
        jmp     disp
fun1:   mov     dl, '1'
        jmp     disp
fun2:   mov     dl, '2'
        jmp     disp
fun3:   mov     dl, '3'
        jmp     disp
fun4:   mov     dl, '4'
        jmp     disp
fun5:   mov     dl, '5'
```

```
        jmp     disp
fun6:   mov     dl,  '6'
        jmp     disp
fun7:   mov     dl,  '7'
        jmp     disp
        ;
disp:   mov     ah,  2          ;显示一个字符
        int     21h
        pop     bx
        pop     ax
        jmp     restart
done:   …
```

3.22 编制程序完成 12H、45H、F3H、6AH、20H、FEH、90H、C8H、57H 和 34H 共 10 个无符号字节数据之和，并将结果存入字节变量 SUM 中（不考虑进位）。

```
        ；程序如下
        .model  small
        .stack
        .data
        b_data  db  12h,45h,0f3h,6ah,20h,0feh,90h,0c8h,57h,34h  ；原始数据
        num equ  10                     ；数据个数
        sum db  ?                       ；预留结果单元
        .code
        .startup
        xor  si,  si            ；位移量清零
        xor  al,  al            ；取第一个数
        mov     cx,  num        ；累加次数
again:  add  al,  b_data[si]；累加
        inc  si                 ；指向下一个数
        loop    again           ；如未完，继续累加
        mov     sum,    al          ；完了，存结果
        .exit 0
        end
```

3.23 求主存 0040H：0 开始的一个 64KB 物理段中共有多少空格？

```
        ；程序如下
        .model small
        .code
start:  mov     ax,  0040h      ；送段地址
        mov     ds,  ax
        mov     si,  0          ；偏移地址
        mov     cx,  si         ；计数（循环次数）
        xor  ax,  ax            ；空格计数器清零
again:  cmp     byte ptr [si], 20h  ；与空格的 ASCII 码比较
        jne next                ；不是空格，转
```

```
        inc   ax                    ; 是空格，空格数加 1
next:   inc si                      ; 修改地址指针
        loop again                  ; cx＝cx－1，如 cx＝0 退出循环
        .exit 0
        end start
```

3.24 编写计算 100 个正整数之和的程序。如果和不超过 16 位字的范围（65535），则保存其和到 wordsum，如超过则显示‘Overflow!’。

```
        ；程序如下
        ；数据段
count   equ  100
parray  dw      count dup(?)        ；假设有 100 个数据
wordsum dw      0
msg     db      ‘overflow’,’$’
        ；代码段
        mov     cx,  count
        mov     ax,  0
        mov     bx,  offset parray
again:  add     ax,  [bx]
        jnc     next
        mov     dx,  offset msg
        mov     ah,  9
        int     21h                 ；显示溢出信息
        jmp     done                ；然后，跳出循环体
next:   add     bx,  2
        loop    again
        mov     wordsum,ax
done:   …
```

3.25 编制程序完成将一个 16 位无符号二进制数转换成为 8421BCD 码表示的 5 位十进制。转转换算法可以是：用二进制数除以 10000，商为“万位”，再用余数除以 1000，得到“千位”；依次用余数除以 100、10 和 1，得到“百位”、“十位”和“个位”。

```
        ；程序如下
        ；数据段
        .model small
        .stack 256
        .data
        array dw  ?                 ；源字数据
        dbcd    db  5   dup(?)      ；五位 bcd 结果，高对高低对低
        .code
        .startup
        mov     dx,  array          ；取源数据（余数）
        mov     bx,  10000          ；除数
        mov     cx,  10             ；除数系数
```

```
        mov     si,   4             ; 目的数据高位位移量
again:  mov     ax,   dx            ; dx.ax 中存放被除数
        mov     dx,   0
        div     bx                  ; 除于 bx，商 ax，余数 dx
        mov     dbcd[si], al        ; 商＜10，存结果
        push    dx                  ; 暂存余数
        mov     ax,   bx            ; 除数除于 10
        mov     dx,   0
        div     cx                  ; dx.ax 除于 cx，商 ax、余数 0 存在 dx
        mov     bx,   ax            ; bx 是除数
        pop     dx
        dec     si                  ; 目的数据位移量减 1
        jnz     again
        mov     dbcd,     dl        ; 存个位数（ ＜10 ）
        .exit   0
        end
```

3.26 过程定义的一般格式是怎样的？子程序开始为什么常有 PUSH 指令，返回前为什么常有 POP 指令？下面完成 16 位无符号数累加的子程序有什么不妥吗？若有，请改正：

```
crazy   PROC
        push    ax
        xor     ax,   ax
        xor     dx,   dx
again:  add     ax,   [bx]
        adc     dx,   0
        inc     bx
        inc     bx
        loop    again
        ret
        ENDP    crazy
```

```
crazy   proc
        push    bx
        push    cx
        xor     ax,   ax
        xor     dx,   dx
again:  add     ax,   [bx]
        adc     dx,   0
        inc     bx
        inc     bx
        loop    again
        ret
        pop     cx
        pop     bx
crazy   ENDP
```

- ✧ 汇编语言中，子程序要用一对过程伪指令 PROC 和 ENDP 声明，格式如下：
  过程名　PROC [NEAR|FAR]
  ……　　；过程体
  过程名　ENDP
- ✧ 子程序中必须保护用到的寄存器内容，以便子程序返回时进行相应的恢复。
- ✧ 改错：**本子程序入口参数为 BX，出口参数为 DX-AX**

3.27 编写一个源程序，在键盘上按一个键，将从 AL 返回的 ASCII 码值显示出来，如果按下 ESC 键则程序退出。

```
again:  mov     ah,   1
        int     21h
        cmp     al,   1bh       ; ESC 的 ASCII 码是 1bh
```

```
        je      done
        mov     dl, al
        mov     ah, 2
        int     21h             ; 是大写字母则转换为小写字母
        jmp     again
done:   …
```

3.28 请按如下说明编写子程序：

```
；子程序功能：把用 ASCII 码表示的两位十进制转换为对应二进制
；入口参数：DH = 十位数的 ASCII 码，DL = 个位数的 ASCII 码
；出口参数：AL = 对应的二进制数
asctob  proc
        push    cx
        and     dh, 0fh         ; 先转换十位数
        shl     dh, 1           ; 十位数乘以 10（采用移位指令）
        mov     ch, dh
        shl     dh, 1
        shl     dh, 1
        add     dh, ch
        and     dl, 0fh         ; 转换个位数
        add     dh, dl          ; 十位数加个位数
        mov     al, dh          ; 设置出口参数
        pop     cx
        ret
asctob  endp
```

3.29 调用 HTOASC 子程序，编写显示一个字节的 16 进制数、后跟“H”的子程序。

```
DIPASC  proc            ; 入口参数：AL＝要显示的一个 16 进制数
        push    cx
        push    dx
        push    ax
        mov     cl, 4   ; 转换高位
        shr     al, cl
        call    HTOASC
        mov     dl, al  ; 显示
        mov     ah, 2
        int     21h
        pop     ax      ; 转换低位
        call    HTOASC
        mov     dl, al  ; 显示
        mov     ah, 2
        int     21h
        mov     dl, ‘H’ ; 显示一个字母“H”
```

```
            mov     ah,  2
            int     21h
            pop     dx
            pop     cx
            ret
DIPASC      endp
HTOASC      proc            ：将 AL 低 4 位表达的一位 16 进制数转换为 ASCII 码
            and     al,  0fh
            cmp     al,  9
            jbe     htoasc1
            add     al,  37h    ：是 0AH～0FH，加 37H 转换为 ASCII 码
            ret                 ：子程序返回
htoasc1:    add     al,  30h    ：是 0～9，加 30H 转换为 ASCII 码
            ret                 ;子程序返回
HTOASC      endp
```

3.30 写一个子程序，根据入口参数 AL = 0、1、2，依次实现对大写字母转换成小写、小写转换成大写或大小写字母互换。欲转换的字符串在 STRING 中，用 0 表示结束。

```
lucase      proc
            push    bx
            mov     bx,  offset string
            cmp     al,  0
            je      case0
            cmp     al,  1
            jz      case1
            cmp     al,  2
            jz      case2
            jmp     done
case0:      cmp     byte ptr [bx],  0
            je      done
            cmp     byte ptr [bx],  ’A’
            jb      next0
            cmp     byte ptr [bx],  ’Z’
            ja      next0
            add     byte ptr [bx],  20h
next0:      inc     bx
            jmp     case0
case1:      cmp     byte ptr [bx],  0
            je      done
            cmp     byte ptr [bx],  ’a’
            jb      next1
            cmp     byte ptr [bx],  ’z’
            ja      next1
```

```
        sub     byte ptr [bx],  20h
next1:  inc     bx
        jmp     case1
case2:  cmp     byte ptr [bx],  0
        je      done
        cmp     byte ptr [bx],  ’A’
        jb      next2
        cmp     byte ptr [bx],  ’Z’
        ja      next20
        add     byte ptr [bx],  20h
        jmp     next2
next20: cmp     byte ptr [bx],  ’a’
        jb      next2
        cmp     byte ptr [bx],  ’z’
        ja      next2
        sub     byte ptr [bx],  20h
next2:  inc     bx
        jmp     case2
done:   pop     bx
        ret
lucase  endp
```

3.31 子程序的参数传递有哪些方法？请简单比较。

（1）用寄存器传递参数：最简单和常用的参数传递方法是通过寄存器，只要把参数存于约定的寄存器中就可以了。

由于通用寄存器个数有限，这种方法对少量数据可以直接传递数值，而对大量数据只能传递地址

采用寄存器传递参数，注意带有出口参数的寄存器不能保护和恢复，带有入口参数的寄存器可以保护、也可以不保护，但最好能够保持一致

（2）用共享变量传递参数

子程序和主程序使用同一个变量名存取数据就是利用共享变量（全局变量）进行参数传递

如果变量定义和使用不在同一个源程序中，需要利用 PUBLIC、EXTREN 声明

如果主程序还要利用原来的变量值，则需要保护和恢复

利用共享变量传递参数，子程序的通用性较差，但特别适合在多个程序段间、尤其在不同的程序模块间传递数据

（3）用堆栈传递参数

参数传递还可以通过堆栈这个临时存储区。主程序将入口参数压入堆栈，子程序从堆栈中取出参数；子程序将出口参数压入堆栈，主程序弹出堆栈取得它们

采用堆栈传递参数是程式化的，它是编译程序处理参数传递、以及汇编语言与高级语言混合编程时的常规方法

3.32 采用堆栈传递参数的一般方法是什么？为什么应该特别注意堆栈平衡问题。

- ✧ 方法：主程序将入口参数压入堆栈，子程序从堆栈中取出参数；子程序将出口参数压入堆栈，主程序弹出堆栈取得它们
- ✧ 注意：压栈与弹栈必须要一一对应。

3.33 编写一个求 32 位数据绝对值的子程序，通过寄存器传递入口参数。

方法 1：

```
neg32   proc ; 入口参数：DX.AX＝32 位有符号数
        neg     ax          ; 实现 0－DX.AX 功能
        neg     dx
        sbb     dx, 0       ; 这条指令也可以用 dec dx 代替
        ret
neg32   endp                ; 出口参数：DX.AX＝32 位有符号数的补码
```

方法 2：

```
neg32   proc                ; 入口参数：DX.AX＝32 位有符号数
        not     ax          ; 实现 DX.AX 求反加 1
        not     dx
        add ax, 1
        adc     dx, 0
        ret
neg32   endp                ; 出口参数：DX.AX＝32 位有符号数的补码
```

3.34 编写一个计算字节校验和的子程序。所谓“校验和”是指不记进位的累加，常用于检查信息的正确性。主程序提供入口参数，有数据个数和数据缓冲区的首地址。子程序回送求和结果这个出口参数。传递参数方法自定。

```
        ;数据段
array db 12h,25h,0f0h,0a3h,3,68h,71h,0cah,0ffh,90h;数组
count   equ  $-array            ;数组元素个数
result  db ?                ;校验和
    ;代码段
    mov     bx, offset array        ;BX←数组的偏移地址
    mov     cx, count               ;CX←数组的元素个数
    call    checksum                ;调用求和过程
    mov     result,     al      ;处理出口参数
    mov     ax, 4c00h
    int     21h
    ;计算字节校验和的通用过程
    ;入口参数：DS:BX＝数组的段地址:偏移地址，CX＝元素个数
    ;出口参数：AL＝校验和
    ;说明：除 AX/BX/CX 外，不影响其他寄存器
Checksum    proc
            xor     al,  al     ;累加器清 0
sum:        add     al,  [bx]   ;求和
            inc     bx          ;指向下一个字节
            loop    sum
```

```
        ret
checksum endp
        end
```

3.35 编制 3 个子程序，把一个 16 位二进制数用 4 位 16 进制数形式在屏幕上显示出来，分别运用如下 3 种参数传递方法，并配合 3 个主程序验证它。

（1）采用 AX 寄存器传递这个 16 位二进制数。

（2）采用 TEMP 变量传递这个 16 位二进制数

（3）采用堆栈方法传递这个 16 位二进制数。

(1)

```
        .model small
        .stack
        .data
wdata   dw 34abh
        .code
        .startup
        mov ax,wdata
        call dispa
        .exit 0
    ;
dispa   proc
        push  cx
        push    dx
        mov     cl,   4
        mov     dl,   ah
        shr     dl,   cl
        call    dldisp
        mov     dl,   ah
        and     dl,   0fh
        call    dldisp
        mov     dl,   al
        shr     dl,   cl
        call    dldisp
        mov     dl,   al
        and     dl,   0fh
        call    dldisp
        pop     dx
        pop     cx
        ret
dispa   endp
    ;
dldisp  proc
        push    ax
        or      dl,   30h
```

```
        cmp     dl,  39h
        jbe     dldisp1
        add     dl,  7
dldisp1: mov    ah,  2
        int     21h
        pop     ax
        ret
dldisp  endp
        end
```

(2)

```
        .model small
        .stack
        .data
wdata   dw 34abh
wordtemp    dw ?
        .code
        .startup
        mov     ax,  wdata
        mov     wordtemp,    ax
        call dispa
        .exit 0
        ;
dispa   proc
        push    cx
        push    dx
        mov     cl,  4
        mov     dl,  byte ptr wordtemp+1
        shr     dl,  cl
        call    dldisp
        mov     dl,  byte ptr wordtemp+1
        and     dl,  0fh
        call    dldisp
        mov     dl,  byte ptr wordtemp
        shr     dl,  cl
        call    dldisp
        mov     dl,  byte ptr wordtemp
        and     dl,  0fh
        call    dldisp
        pop     dx
        pop     cx
        ret
dispa   endp
    ;
dldisp  proc
```

```
        push    ax
        or      dl,  30h
        cmp     dl,  39h
        jbe     dldisp1
        add     dl,  7
dldisp1: mov    ah,  2
        int     21h
        pop     ax
        ret
dldisp  endp
        end
```

(3)

```
        .model small
        .stack
        .data
wdata   dw  34abh
        .code
        .startup
        push    wdata
        call    dispa
        pop     ax   ;
        add     sp,  2
        .exit 0
    ;
dispa   proc
        push    bp
        mov     bp,  sp
        push    ax
        push    cx
        push    dx
        mov     ax,  [bp+4]
        mov     cl,  4
        mov     dl,  ah
        shr     dl,  cl
        call    dldisp
        mov     dl,  ah
        and     dl,  0fh
        call    dldisp
        mov     dl,  al
        shr     dl,  cl
        call    dldisp
        mov     dl,  al
        and     dl,  0fh
        call    dldisp
```

```
        pop     dx
        pop     cx
        pop     ax
        pop     bp
        ret
dispa   endp
    ;
dldisp  proc
        push    ax
        or      dl,  30h
        cmp     dl,  39h
        jbe     dldisp1
        add     dl,  7
dldisp1: mov    ah,  2
        int     21h
        pop     ax
        ret
dldisp  endp
end
```

3.36 什么情况需要使用 PUBLIC 和 EXTERN 伪指令？请将例题 3-20 的子程序全部用寄存器传递参数，写成子程序模块。

✧ 如果利用共享变量传递函数，且变量定义和使用不在同一个源程序中，需要利用 PUBLIC、EXTERN 声明。

3.37 宏是如何定义、调用和展开的？

✧ 宏定义由一对宏汇编伪指令 MACRO 和 ENDM 来完成，格式如下：

```
宏名    MACRO [形参表]
        ……      ；宏定义体
        ENDM
```

宏定义之后就可以使用它，即宏调用：

```
        宏名 [实参表]
```

✧ 宏调用的格式同一般指令一样：在使用宏指令的位置写下宏名，后跟实体参数；如果有多个参数，应按形参顺序填入实参，也用逗号分隔

✧ 宏展开：在汇编时，宏指令被汇编程序用对应的代码序列替代，这就是宏展开。宏展开的具体过程是：当汇编程序扫描源程序遇到已有定义的宏调用时，即用相应的宏定义体完全替代源程序的宏指令，同时用位置匹配的实参对形参进行取代

3.38 宏参数有什么特点？宏定义的形参如何与宏调用的实参相结合？

✧ 宏调用的参数通过形参、实参结合实现传递，简捷直观、灵活多变。宏汇编的一大特色是它的参数。宏定义时既可以无参数，也可以有一个或多个参数；宏调用时实参的形式也非常灵活，可以是常数、变量、存储单元、指令（操作码）或它们的一部分，也可以是表达式；只要宏展开后符合汇编语言的语法规则即可。

3.39 说明宏汇编和子程序的本质区别，程序设计中如何选择？

- ✧ 宏：仅是源程序级的简化：宏调用在汇编时进行程序语句的展开，不需要返回；不减小目标程序，执行速度没有改变
  通过形参、实参结合实现参数传递，简捷直观、灵活多变
- ✧ 子程序：还是目标程序级的简化：子程序调用在执行时由 CALL 指令转向、RET 指令返回；形成的目标代码较短，执行速度减慢
  需要利用寄存器、存储单元或堆栈等传递参数
- ✧ 选择：宏与子程序具有各自的特点，程序员应该根据具体问题选择使用那种方法。通常，当程序段较短或要求较快执行时，应选用宏；当程序段较长或为减小目标代码时，要选用子程序

3.40 编写一个宏指令“move doprn, soprnd”，它实现任意寻址方式的字量源操作数送到目的操作数，包括存储单元到存储单元的传送功能。

```
move    macro doprnd, soprnd
mov     ax,   soprnd
mov     doprnd,   ax
endm
```

3.41 定义一个宏 logical，用它代表 4 条逻辑运算指令：and / or / xor / test，注意需要利用 3 个形式参数，并给一宏调用以及宏展开的例子。

```
logical   macro lcode,   dopd,sopd
lcode     dopd,     sopd
endm
```

- ➢ 例如，如果使用“and ax,[bx]”指令，可以利用该宏定义，写出宏指令如下：

```
logical and, ax,[bx]
```

3.42 写一个宏，判断 AL 寄存器中的一个 ASCII 是否为大写字母，如果是大写字母就转换为小写字母，否则不转换。

```
utol  macro
      local     next
      cmp       al,   ‘A’          ; 小于“A”不转换
      jb        next
      cmp       al,   ‘Z’          ; 大于“A”不转换
      ja        next
      add       al,   20h          ; 是大写字母则转换为小写字母
      next:
endm
```

3.43 定义一个宏“movestr strN, dstr, sstr”，将 strN 个字符从一个字符区 sstr 传送到另一个字符区 dstr。

```
;（假设它们都在数据段）
movestr   macro strn, dstr, sstr
      mov       cx,   ds
      mov       es,   cx
      mov       cx,   strn
```

```
        mov     di,   offset dstr
        mov     si,   offset sstr
        cld
        rep     movsb           ; 重复传送 ES:[DI]←DS:[SI]
    endm
```

3.44 作为总结提高，读者可以综合前 3 章汇编语言的知识，编写一个通用的输入输出子程序库 IO.LIB，实现字符、字符串、二进制、十进制、十六进制的键盘输入和显示器输出，以及 8 个通用寄存器、状态标志等的显示功能。

为便于调用，可以配合一个包含文件 IN.INC。 这样，只要在主程序文件开始增加一个文件包含语句"INCLUDE IO.INC"，IO.INC 和 IO.LIB 保存在当前目录下，就可又使用该子程序库中的子程序了，详见附录 F 的说明。

第4章　　引脚与总线

4.1 微机总线的信号线包括 数据线 、 地址线 、 控制线 以及电源和地线。微机系统可以将总线划分为三层（类），它们是 数据总线 、 地址总线 、 控制总线 。

4.2 占用总线进行数据传输，一般需要经过总线请求和仲裁、 寻址 、 数据传送 和结束 4 个阶段。

4.3 什么是同步时序、半同步时序和异步时序？

- ✧ 同步时序：总线操作的各个过程由共用的总线时钟信号控制，具有固定的时序，主控模块和受控模块之间没有应答联络信号。优点是：简单快速，适合速度相当的模块之间传输数据。缺点是快速模块必须迁就等待慢速设备。
- ✧ 半同步时序：有共同的时钟信号，但增加了一条等待信号线，以便于慢速设备向快速模块发出申请，不作无效等待。
- ✧ 异步时序：也称应答方式。总线操作需要握手联络（handshake）（或应答）信号控制，时钟信号可有可无。数据传输开始时有启动信号（又称请求 request、选通 strobe），传输结束需要有一个确认（acknowledge）信号。

4.4 ISA 总线的时钟频率是 8MHz，每 2 个时钟可又传送一个 16 位数据，计算其总线带宽。

总线带宽(bandwidth)指单位时间传输的数据量，也称为总线传输速率或吞吐率(bus throughput)，常以每秒兆字节(MB/s)、每秒兆位(Mb/s)或每秒位(b/s, bps)为单位。原表示频带宽度，计算机领域的总线带宽表示数据传输能力。有以下公式：

总线带宽 = 传输的数据量 ÷ 需要的时间

wd = 16 ÷ (2 x (1 ÷ (8 x 10^6))) = 64 x 10^6 bps = 64 Mb/s = 8 MB/s

4.5 何为引脚信号的三态能力？当具有三态能力的引脚输出高阻时究竟意味着什么？在最小组态下，8088 的哪些引脚具有三态能力？

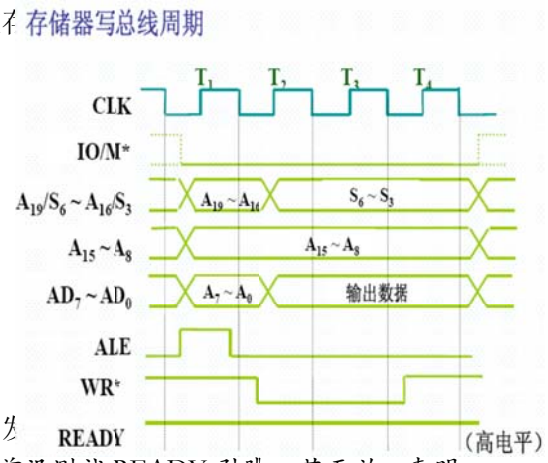
- ✧ 引脚三态能力主要针对引脚输出信号状态：高电平、低电平和高阻。
- ✧ 输出高阻意味着芯片放弃对引脚的控制。这样它所连接的设备就可以接管该引脚及连接导线的控制权。
- ✧ 8088 的三态引脚：AD7 – AD0、A19 – A8、ALE、IO/M*、WR*、RD*、DT/R*、DEN*。共 26 个引脚。

4.6 以下 8088 的输入信号：RESET、HOLD（最小组态）、NMI 和 INTR，其含义各是什么？当它们有效时，8088 CPU 将出现何种反应？

- ✧ RESET：复位请求，高电平有效时，CPU 回到初始状态
- ✧ HOLD：总线请求，高电平有效时，其他总线主控设备向 CPU 申请占用总线
- ✧ NMI：不可屏蔽中断请求。外界向 CPU 申请不可屏蔽中断
- ✧ INTR：可屏蔽中断请求。高电平有效时，中断请求设备向 CPU 申请可屏蔽中断。

4.7 执行一条指令所需要的时间被称为 指令 周期，而总线周期指的是 完成一次总线操作所需的时间，8088 基本的总线周期由 4 个 T 组成。如果 8088 的 CLK 引脚接 5MHz 的时钟信号，那么每个 T 状态的持续时间为 200ns 。

4.8 请解释在最小组态下 8088 以下引脚信号的含义：CLK、A19/S6 ~ A16/S3、A15 ~ A8，AD7 ~ AD0、ALE、IO/M*、RD*、WR*。并写出它们在存储器写总线周期



CLK —— 时钟输入

A19/S6 ~ A16/S3 —— 地址/状态分时复用引脚

A15 ~ A8 —— 8 位地址引脚

AD7 ~ AD0 —— 地址/数据分时复用引脚

ALE —— 地址锁存允许

IO/M* —— 访问 IO 或者存储器

RD* —— 读控制

WR* —— 写控制

4.9 在 8088 的工作过程中，什么情况下会产生 TW？为

解：总线操作周期中，8088 在第三个时钟周期的前沿测试 READY 引脚，若无效，表明被访问的设备与 CPU 操作不同步，CPU 插入等待周期。

4.10 以 8088 的“读总线”周期为例，说明从 T1 – T4 各个 T 状态时的总线操作。

（1）T1 周期，CPU 进行读操作。

（2）T2 - T4 期间，CPU 对数据总线输出高阻态，选通存储器或 I/O 接口，向 CPU 传送数据。

（3）T4 的下降沿，CPU 对数据总线采样。

4.11 在 8088 系统（最小组态）中，读取指令“ADD [2000H]，AX”（指令长度为 4B）和执行该指令各需要几个总线周期？它们各是什么样的总线周期？

8088 每个总线周期只能读取一个字节数据，

ADD [2000H]，AX 的指令码为 01060020，共四个字节，存储在代码段 CS 中，

读取该指令需要四个“存储器读”总线周期，

执行该加法指令时，CPU 先访问数据段 DS 中的存储单元[2000H]和[2001H]，读取其中存放的一个字（两个字节）数据，需要两个“存储器读”总线周期，

由于读指令的同时 CPU 执行指令，故加法运算时间可忽略不计。

与 AX 相加后的结果还需送到存储器，此结果为两个字节，还需要两个“存储器写”总线周期。

4.12 对比 Intel 8088 最小组态的引脚和 IBM PC 总线，说明它们主要的异同点。

8088 最小组态引脚（40 脚）		PC 总线引脚（62 脚）	
地址/数据	AD7 ~ AD0, A19 ~ A8		相同
读写控制	ALE, IO/M*, WR*, RD*, READY		
中断请求	INTR, INTA*, NMI		
总线仲裁	HOLD, HLDA		
初始化	RESET		
时钟	CLK		
电源和地线	Vcc, GND(2 组)		
其他	MN/MX*, TEST*, DEN*, DT/R*, SS0*		
		DMA 控制	

4.13 请解释 IBM PC 总线中 D7 – D0、A19 – A0、ALE、IOR*、IOW*、IOCHRDY 信号线的含义，并画出执行外设读取指令“IN AL，DX”时引起的总线周期时序图。

D7 – D0

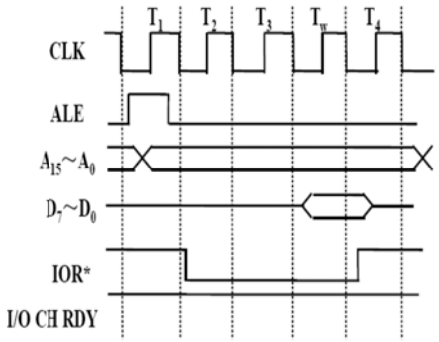
A19 – A0

ALE

IOR*

IOW*

IOCHRDY

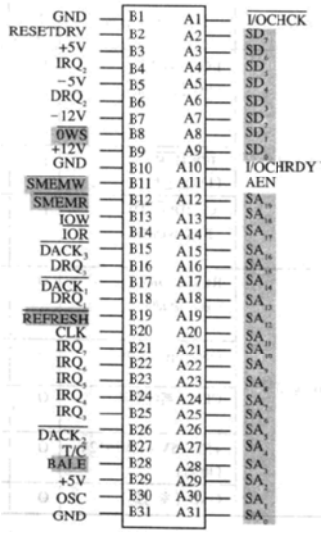


4.14 对比 Intel 8088 最大组态的引脚和 IBM PC 总线，说明它们主要的异同点。

8088 最大组态引脚（40 脚）		PC 总线引脚（62 脚）	
地址/数据	AD7 ~ AD0, A15 ~ A8, A16/S0 ~ A19/S3		相同
读写控制	ALE, IO/M*, WR*, RD*, READY		
中断请求	INTR, INTA*, NMI		
总线仲裁	HOLD, HLDA		
初始化	RESET		
时钟	CLK		
电源和地线	Vcc, GND(2 组)		
其他	MN/MX*, TEST*, DEN*,DT/R*,SS0*		基本相同
		DMA 控制	

4.15 对比 IBM PC 总线，ISA 总线主要增加了什么信号线？

ISA（Industry Standard Architecture），意思为工业标准结构。是 IBM PC/AT 机的系统总线，也称 AT 总线，亦为 IEEE P996 标准。有前 62 引脚和 36 引脚。前 62 脚基本与 PC 总线相同。后 36 脚扩展了 8 位数据线、7 位地址线以及控制引脚等，共有 A 面和 B 面，C 面和 D 面。



A 面和 B 面（与 PC 总线同）

C 面和 D 面

第5章　　内存

5.1 可读可写的存储器为什么被称为 RAM？什么是 SRAM 和 DRAM？说明各自的特点。

RAM, Random Access Memory, 随机存取存储器, 与顺序存取相对应. RAM 可从任意位置开始读写，存取位置可随机确定，只要给出存取位置就可以读写内容。RAM 亦可解释为与 ROM 相对应，ROM，Read Only Memory，不可写入，只能读出其中数据。
SRAM：Static RAM，静态随机存取存储器，以触发器为基本存储单元，无需刷新操作，优点是速度快，但价格较高。
DRAM：Dynamic RAM，动态 RAM，以单个 MOS 管为基本存储单元，以极间电容是否充有电荷表示两种逻辑状态，由于电容会很快放电，需要不断地进行刷新操作，即读取原内容、放大再写入。DRAM 优点是集成度高、价格低、功耗小，但速度较慢。

5.2 只读的半导体存储器 ROM 能写入吗？从编程角度说明 MROM、OTP-ROM、UV-EPROM、EEPROM 和 Flash Memory 的不同。

掩膜 ROM 出厂时存储的内容已制作好，不可再写入，只能读出。
PROM 出厂后可编程写入一次，以后只能读出。亦称 OTP-ROM
EPROM 可用紫外线擦除，并可重复编程写入。亦称 UV-EPROM
EEPROM，电可擦除，可写入。
Flash Memory，闪速存储器，加电可按块(block)擦除再写入，寿命比 EEPROM 短

5.3 类似于微处理器总线，存储器芯片也分成数据、地址和控制 3 类引脚。以存储结构为 32K x 8 的 SRAM 62256 为例，该芯片应有 8 个数据引脚、 15 个地址引脚，3 个不同类型的控制引脚分别是 CS 、 RD 和 WR 。

5.4 都是描述半导体存储器的存取速度，存取时间和存取周期有什么不同？制作一张表罗列本章介绍的 10 个存储器芯片的读取时间和读取周期。

存取时间：从读写命令发出，到数据传输操作完成所经历的时间。
存取周期：两次存储器访问所允许的最小时间间隔。存取周期一般大于存取时间。

芯片	读取时间 TRC	读取周期 TAA
SRAM 6116	120ns	120ns
DRAM 4164	150ns	270ns
EPROM 2764	120ns	
EEPROM 2816	150ns	<10ms
EEPROM28C040	200ns	<10ms
Flash 29c512	<150us/512~=~300ns	<10ms/512

5.5 什么是动态 RAM 的刷新？为什么动态 RAM 需要经常刷新？存储系统如何进行动态 RAM 的刷新？

动态随机存取存储器 (DRAM)的存储单元是由 MOS 管的栅极电容 C 和门控管组成的。数据以电荷的形式存储在栅极电容上，电容上的电压高表示存储数据 1；电容没有

储存电荷，电压为 0，表明存储数据 0 。因存在漏电，使电容存储的信息不能长久保持，为防止信息丢失，就必须定时地给电容补充电荷，这种操作称为“ 刷新 ”。 由于要不断地刷新，所以称为动态存储。

方法：采用“仅行地址有效”方法刷新；

刷新周期：15μs

刷新次数：128

5.6 可编程 ROM 芯片的备用工作方式有什么特点？芯片识别代码有什么作用？

备用(standby)是 ROM 芯片常支持的一种无数据输出状态，此时芯片功耗可以降低，如 27C64 从 20mA 下降到 100nA。

许多电子器件在生产过程中制作了一些识别代码，用户可以通过程序读出，以便确认器件并充分利用器件本身的特点。所谓识别(Identity)工作方式，就是在特定条件下读取这些识别代码的操作状态。如 27C64 在 Vpp 接高电平，CE*和 OE*为低，PGM*为高，A9 接+13V，地址 A0 = 0 时可读取生产厂商 Microchip 的代码 29H，A0 = 1 时可读取器件代码 02H。

5.7 EEPROM 的擦写与闪存的擦写有什么不同？以 AT28C040 或 AT29C512 为例，说明两种判断擦写是否完成的常用方法，并估算两者完成整个芯片编程的最快时间。

EEPROM 的擦写一般以字节为单位进行，闪存的擦写必须以数据块(block，如 512 字节)进行，所以闪存与硬盘一样是半随机存储器。

EEPROM 判断擦写完成与否的方法之一是查询方式，即：写入进行中，数据引脚 IO7 读取的数据位与写入的相反，则写入完成。AT28C040 还可以采用翻转位(Toggle bit)方法查询是否完成写入操作，具体为：写入进行中，连续对数据 IO6 读取，其结果在 0 和 1 之间不断翻转；写入完成，该引脚停止翻转，写操作完成。

AT29C512 支持上述两种判断擦写是否完成的方法。

查询方式：

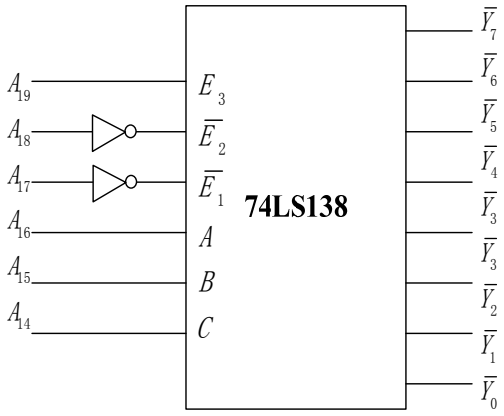
翻转方式：

5.8 SRAM 芯片的片选引脚有什么用途？假设在 8088 微处理器系统中，地址信号 A19 – A15 输出 01011 时译码电路产生一个有效的片选信号，则该片选信号将占多少主存容量？其地址范围是什么？

SRAM 片选引脚通常与 CPU 的部分地址线相连，用以选定该芯片进行读写。

A19 – A15 输出 01011 产生片选，地址范围为 0101 1000 0000 0000 0000 – 0101 1111 1111 1111 1111，即 68000 H – 6FFFF H，容量为 32K。

5.9 请给图 5-24 中 138 译码的所有译码输出引脚对应的地址范围。



A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13} – A_0	Y	地址范围
1	0	0	0	0	0	全0 – 全1	0	80000 – 83FFF
			0	0	1		1	84000 – 87FFF
			0	1	0		2	88000 – 8BFFF
			0	1	1		3	8C000 – 8FFFF
			1	0	0		4	90000 – 93FFF
			1	0	1		5	94000 – 97FFF
			1	1	0		6	98000 – 9BFFF
			1	1	1		7	9C000 – 9FFFF

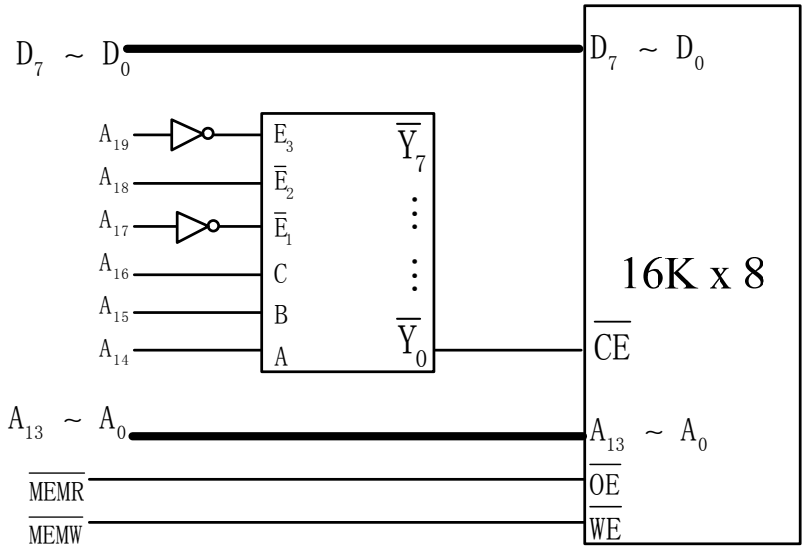
5.10 什么是系统地址信号的全译码和部分译码，各有什么特点？哪种译码方式会产生地址重复？如果连接一个存储器芯片时有 2 个高位系统地址信号没有参加译码，则该芯片的每个存储单元占几个存储器地址？

全译码：所有地址线均参加地址译码，不会产生地址重复
部分译码：部分地址线参加地址译码，会产生地址重复。
若 CPU 的 2 个高位地址线未参加译码，则每个地址单元占 4 个存储器地址。

5.11 什么是存储芯片连接中的“位扩展”和“字扩展”？采用 DRAM 21256（256K x 1）构成 512KB 的 RAM 存储模块，需要多少个芯片，怎样进行位扩展和字扩展？

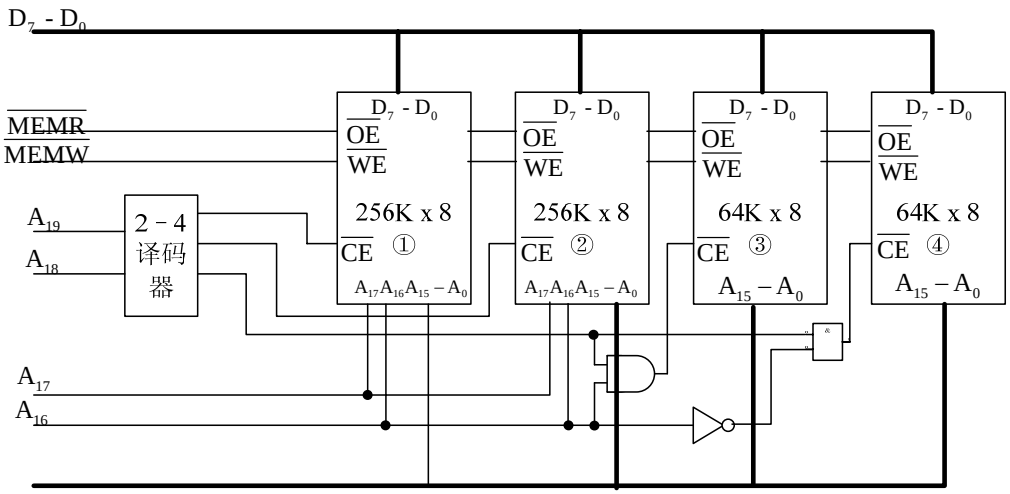
位扩展：数据线扩展，将存储器的数据线依次与 CPU 数据线相连即可
字扩展：存储单元扩展。可对片选端采用译码电路进行。
16 片 256k x 1 的 DRAM 21256 可构成 512KB 的存储体。

5.12 使用一个 16K x 8 结构的 SRAM，采用全译码方式，在 8088 系统中设计首地址是 20000H 的存储器，画出该芯片与系统总线的连接示意图。

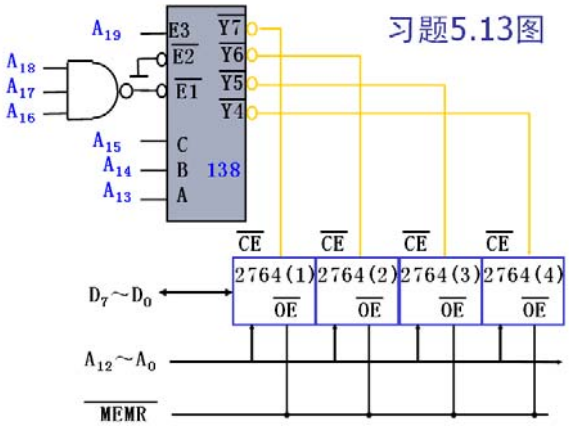


5.13 给出图 5-28 中 4 个存储器芯片各自占用的地址范围。如果采用部分译码，要指出重复的地址范围。

芯片	地址范围
①	00000 – 3FFFF
②	40000 – 7FFFF
③	C0000 – CFFFF/E0000 - EFFFF
④	D0000 – DFFFF/F0000 - FFFFF



5.14 使用 3-8 译码器 74LS138 和多片 8K x 8 结构的 SRAM，采用全译码方式，在 8088 系统中设计存储模块，占用从 0 开始的最低 32KB 地址空间，画出连接示意图。



5.15 开机后，微机系统常需要检测主存储器是否正常。例如，可以先向所有存储单元写入数据 55H（或 00H）然后读出，看是否还是 55H（或 00H）。利用两二进制各位互反的“花样”数据的反复写入，读出和比较就能够识别有故障的存储单元。利用获得的有故障存储单元所在的物理地址，如果能够分析出该存储单元所在的存储器芯片，就可以实现芯片级的维修。试利用汇编语言编写一个检测程序，检测逻辑地址从 9000H：0000H 到 9000H：FFFFH 的存储空间是否有读写错误，如果发现错误请显示其逻辑地址。

```
          ；代码段
          mov     ax,  9000h
          mov     ds,  ax
          mov     ah,  55h        ;先用 55H
again:    mov     bx,  0
          mov     al,  ah
again1:   mov     [bx], al        ;写入
          dec  bx
          jnz  again1
again2:   mov     al,  [bx]       ;读出
          cmp     al,  ah         ;检测
          jz   next2
          dispcrlf
          push    ax
          mov     ax,  ds
          call  disphw            ;显示段地址
          mov     al,  ':'
          call  dispc
          mov     ax,  bx
          call  disphw            ;显示偏移地址
          pop  ax
next2:    dec  bx
          jnz  again2
          pop  ax
          cmp     ah,  0aah       ;后用 0AAH
          jz   done
          mov     ah,  0aah
          jmp  again
done:     ……
```

第6章　　I/O 接口

6.1　典型的 IO 接口电路通常有哪 3 类可编程寄存器?各自的作用是什么?

- ✧ 数据寄存器：输入时保存从外设发往 CPU 的数据，输出时保护从 CPU 发往外设的数据。
- ✧ 状态寄存器：保存状态数据。CPU 可以获取当前接口电路或外设的状态。
- ✧ 控制寄存器：保存控制数据。CPU 可以向其写入命令，选择接口电路工作方式，控制外设。

6.2　IO 端口与存储器地址常有__独立编址__和__混合编址____两种编排方式，8088/8086 处理器支持后者，设计有专门的 IO 指令。其中指令 IN 是将数据从__端口__传输到__CPU__，执行该指令时，8088/8086 处理器引脚产生__两个__总线周期。指令“OUT DX，AL”的目的操作数是__寄存器间接__寻址方式，源操作数是__寄存器直接__寻址方式。

6.3　参考例 6-1 读取 CMOS RAM 数据，编写一个显示当前时分秒时间的程序。

```
        ; 数据段
date    db '2000-01-01','$'
        ; 代码段
        mov     bx,   offset date+2
        mov     cl,   4
        mov     al,   9             ;准备从 9 号单元获取年代数据
        out     70h, al             ;选择 CMOS RAM 的 9 号单元
        in      al,   71h           ;获取 9 号单元的内容
        mov     ah,   al            ; 转存 AH
        shr     ah,   cl            ; 处理年代高位
        add     ah,   30h           ; 转换为 ASCII 码
        mov     [bx], ah            ; 存入数据区
        add     bx,   1             ; 指向下位
        and     al,   0fh           ; 处理年代低位
        add     al,   30h           ; 转换为 ASCII 码
        mov     [bx], al            ; 存入数据区
        add     bx,   2             ; 指向下位
        mov     al,   8             ; 从 8 号单元获取月份数据
        out     70h, al
        in      al,   71h
        …
        mov     al,   7             ;从 7 号单元获取日期数据
        …
        mov     dx,   offset date
        mov     ah,   9
        int     21h                 ; 显示日期
```

6.4 基于图 6-7 接口电路，编程使发光二极管循环发光。具体要求是：单独按下开关 K0，发光二极管以 L0、L1、L2、……、L7 顺序依次点亮，每个维持 200ms，并不断重复，直到有其他按键操作；单独按下开关 K1，发光二极管以 L7、L6、、L5、……、L0 顺序依次点亮，每个也维持 200ms，并不断重复，直到有其他按键操作；其他开关组合均不发光，单独按下开关 K7，则退出控制程序。延时 200ms 可以直接调用子程序 DELAY 实现。

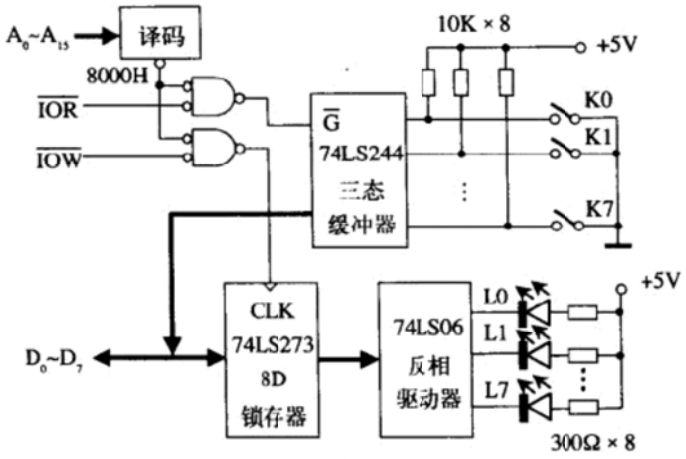


图6-6 无条件传送接口举例

6.5 有一个类似于图 6-9 的查询输入接口电路，但其数据端口为 8F40H，状态端口为 8F42H。从状态端口最低位可以获知输入设备是否准备好一个字节的数据：D0 = 1 表示准备好，D0 = 0 说明没准备好。不考虑查询超时，编程从输入设备读取 100 个字节保存到 INBUF 缓冲区。

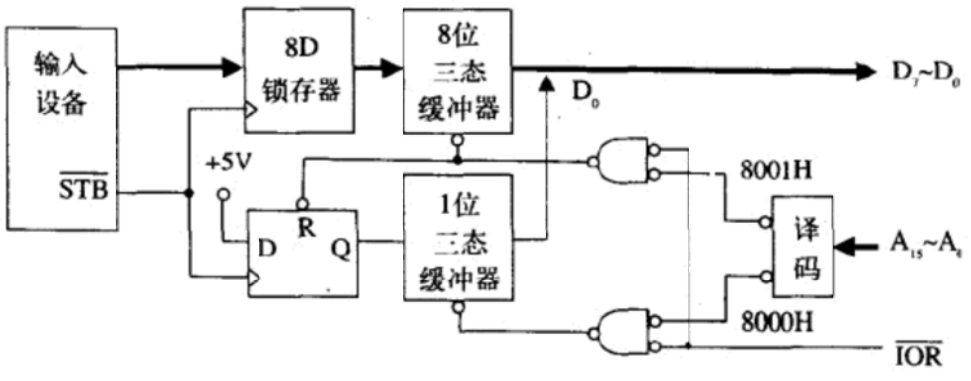


图6-8 查询输入接口

```
Code
Start:  mov     ax,   SEG INBUF
        mov     ds,   ax
        mov     bx,   offset INBUF
        mov     dx,   8F42H
        mov     cx,   100
Next:   in      al,   dx
        test    al,   1             ;
        jz      next
        sub     dx,   2             ; 8F42 – 2 = 8F40
```

```
        in      al,   dx
        mov     [bx], al
        inc     bx
        add     dx,   2
Loop    next
```

6.6 有一个类似于图 6-10 的查询输出接口电路，但其数据输出和状态端口均为 8000H，并从状态端口的 D6 位获知输出设备是否能够接收一个字节的数据：D6 = 1 表示可以接收，D6 = 0 说明不能接收。不考虑查询超时，编程将存放于缓冲 OUTBUF 处的字符串（以 0 为结束标志）传送给输出设备。

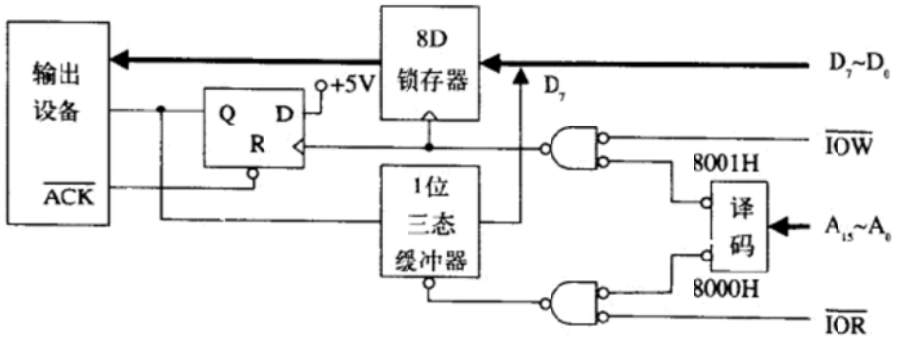


图6-9 查询输出接口

```
        mov     bx,   offset ADDR
again:  mov     ah,   [bx]
        cmp     ah,   '0'
        jz      done
status: in      al,   8000h          ;查询一次
        test    al,   0100 0000B
        jnz     status
        mov     al,   ah
        out     8000h, al            ;输出一个字节
        inc     bx
        jmp     again                ;循环
done:   .......
```

6.7 结合中断传送的工作过程，简述有关概念：中断请求、中断响应、中断关闭、断点保护、中断源识别、现场保护、现场恢复、中断开放、中断返回、中断优先权和中断嵌套。

- ✧ 中断请求　外设以硬件信号的形式、向处理器发送有效信号，应保持有效到被响应。中断传送过程由外设的中断请求启动，获得处理器认可，才真正进入中断传送过程
- ✧ 中断响应　处理器需要满足一定条件，才能响应中断请求：
  （1）指令执行结束后才能响应外设的中断请求
  （2）处理器处于开放中断的状态
  （3）中断请求的同时，没有更高级别的其他请求、
- ✧ 中断关闭　不允许可屏蔽中断被响应，中断被屏蔽了、被禁止了，关中断
- ✧ 断点保护　断点是指被中断执行的指令位置。断点保护，保护断点指令所在的

存储器地址，断点保护一般由处理器自动完成，有的处理器还可能自动保护程序状态

- ✧ 中断源识别 解答1：中断向量（图6-12）解答2：中断查询（图6-13）
- ✧ 现场保护 现场：对处理器执行程序有影响的工作环境，进入中断后需要保护现场中断返回前需要恢复现场，
- ✧ 现场恢复 断点地址（或加上程序状态）由处理器硬件自动保护，其他需要由中断服务程序进行保护和恢复，（通常是处理器的通用寄存器），具体的编程方法可以类似子程序编程
- ✧ 中断开放 允许可屏蔽中断被响应，中断允许、开中断
- ✧ 中断返回 处理器返回断点继续执行原来的程序，中断服务程序最后的一条中断返回指令实现
- ✧ 中断优先权 中断优先权排队，解答1：软件查询：逐个判断解答2：硬件电路：编码电路、链式排队电路
- ✧ 中断嵌套 中断处理过程中，又有中断提出请求，原则1：优先权低于或等于，不予理会；原则2：优先权高于，中断嵌套

6.8 基于图6-13中断查询接口电路，按照图6-14优先权排队流程，编写中断查询程序。假设中断i的请求状态由数据Di位反映（为1表示有请求），对应中断服务子程序INTPi。

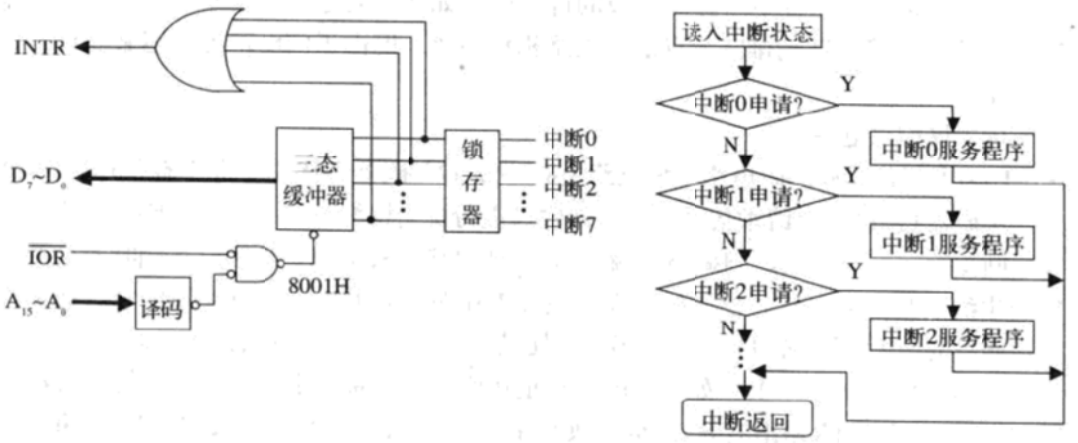


图6-14 中断查询接口与流程

```
Interrupt：    mov dx,8000H
               In al,    dx
               Cmp al,   1
               Jnz   next1
               Call proc1
               Jmp done
Next1:         cmp al,   2
               Jnz next2
               Call proc2
               Jmp done
Next2:         cmp al,   3
               Jnz done
               Call proc3
Done:          ……
```

6.9 简述 DMA 传送的工作过程，什么是 DMA 读和 DMA 写？

⑴ DMA 预处理：CPU 对 DMA 控制器进行初始化设置
⑵ DMA 请求和应答：外设、DMAC 和 CPU 三者通过应答信号建立联系，CPU 将总线交给 DMAC 控制
⑶ DMA 数据传送
DMA 读存储器：存储器 → 外设
DMA 写存储器：存储器 ← 外设
⑷ 自动增减地址和计数，判断传送完成否

6.10 查询、中断和 DMA 传送是微机中常用的外设数据交换方式，说明各自的特点。
无条件传送

慢速外设需与 CPU 保持同步
查询传送
简单实用，效率较低
中断传送
外设主动，可与 CPU 并行工作，但每次传送需要大量额外时间开销
DMA 传送
DMAC 控制，外设直接和存储器进行数据传送，适合大量、快速数据传送

第7章　　中断控制器

7.1　8088CPU 具有哪些中断类型？各种中断如何产生，如何得到中断向量号？

（一）内部中断。由８０８８内部执行程序出现异常引起。

（１）除法错中断

（２）指令中断

（３）溢出中断（

（４）单步中断

（二）外部中断。由８０８８外部提出中断请求引起。

（１）不可屏蔽中断。外部通过ＮＭＩ向ＣＰＵ提出中断请求，ＣＰＵ执行完当前指令就予以响应。

（２）可屏蔽中断。外部通过ＩＮＴＲ信号向ＣＰＵ发出请求。当ＩＦ＝１时，ＣＰＵ在当前指令结束予以响应。

可屏蔽中断向量号由外部提供，处理器产生中断响应周期的同时读取一个字节的中断向量号数据，其他类型的中断向量号包含在指令中或者已经预置。

7.2　8088 中断向量表的作用是什么？

中断向量表是一种表数据结构。是中断向量号与对应中断服务程序之间的连接表。

7.3　说明如下程序段的功能：

```
cli
mov     ax,   0
mov     es,   ax
mov     di,   90h*4
mov     ax,   offset intproc  ;
;intproc 是一个过程名
Cld
Stosw
mov     ax,   seg  intproc
stosw
sti
```

；字符串传送
；每次传送一个字，2 字节
；目的地址为 0000：90H*4
；DI 每次+2

7.4　8259A 中 IRR、IMR 和 ISR 三个寄存器的作用是什么？

ＩＲＲ：中断请求寄存器。存储外界中断请求信号。

ＩＳＲ：中断服务寄存器。存储正在被服务的中断状态。

ＩＭＲ：中断屏蔽寄存器。保存对中断信号的屏蔽状态。

7.5　PC/XT 机的 ROM-BIOS 对 8259A 的初始化程序如下：

```
mov     al,   13h       ；13H＝0001 0011，设定工作方式：单片，边沿触发，要写入 ICW4
out     20h, al         ；写入 ICW1，主（单）片地址是 20H
mov     al,   08h       ；08H ＝ 0000 1000，设定主（单）片 IR0 的中断向量号为
```

08H

out 21h, al ；写入 ICW2，

mov al, 09h ；09H＝0000 1001，设定为 16 位 80x86CPU，非自动中断结束，该片 8259A 是从片，数据线采用缓冲方式，工作于普通全嵌套方式

out 21h, al ；写入 ICW4，

请说明其设定的工作方式。

7.6 某时刻 8259A 的 IRR 内容是 08H，说明_____。某时刻 8259A 的 ISR 内容是 08H，说明_____。在两片 8259A 级联的中断电路， 主片的第 5 级 IR5 作为从片的中断请求输入，则初始化主、从片时，ICW3 的控制字分别是_____和_____。

ＩＲ３有中断请求 -------- ＩＲ３正在服务 -------- ２０Ｈ，０５Ｈ

7.7 8259A 仅占用两个 IO 地址，它是如何区别 4 条 ICW 命令和 3 条 OCW 命令的？在址引脚 A0 = 1 时，读出的是什么内容？

表8-1 8259A的命令字/状态字读写条件

A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	主8259A地址	从8259A地址	功 能
0	1	0	0	20H	A0H	写入ICW1, OCW2和 OCW3①
1	1	0	0	21H	A1H	写入ICW2, ICW3, ICW4和OCW1②
0	0	1	0	20H	A0H	读出IRR，ISR和查询字③
1	0	1	0	21H	A1H	读出IMR
×	1	1	0			数据总线高阻状态
×	×	×	1			数据总线高阻状态

注：① 由命令字中的D4D3两个标志位决定。

② 写入ICW1后，由片内的顺序逻辑确定后续ICW；否则写入OCW1。

③ 由OCW3的内容选择（详见8.3.4节）。

CPU 可读出 IRR、ISR、IMR 和查询字

A0 为低，由 OCW3 中 RR 和 RIS 位设定读取 IRR 或 ISR，由 OCW3 中 P 位设定读取查询字

而 A0 引脚为高电平时读取的都是 IMR

查询字反映 8259A 是否有中断请求

(1) 利用读写信号区别写入的控制寄存器和读出的状态寄存器

(2) 利用地址信号区别不同 I/O 地址的寄存器

(3) 由控制字中的标志位说明是哪个寄存器

(4) 由芯片内顺序控制逻辑按一定顺序识别不同的寄存器

(5) 由前面的控制字决定后续操作的寄存器

实际上，这就是微机区别接口电路（芯片）中不同的寄存器（或控制字和状态字）的主要方法。后续章节的接口电路中同样采用了这些方法，另外也还有一些其他方法。

例如，应该怎样读写ICW2呢？ICW2是一个命令字，需要使用OUT指令，PC机中I/O地址是A_0＝1时的21H或A1H，并且跟在ICW1写入之后。再如，下段程序读出了什么内容？

```
mov al,0ah          ;0AH=00001010B
out 20h,al
nop                 ;延时等待操作完成
in al,20h
```

从端口20H（A_0＝0）输出的命令字有ICW1、OCW2和OCW3，从命令字0AH的D_4D_3＝01判断这是OCW3。再从表8-5看出$D_2D_1D_0$＝010时，下一个读指令（地址A_0＝0时）应该是读取中断请求寄存器IRR的内容。

7.8 某一 8086CPU 系统中，采用一片 8259A 进行中断管理。设定 8259A 工作在普通全嵌套方式，发送 EOI 命令结束中断，采用边沿触发方式请求中断，IR0 对应的中断向量号为 90H。另外，8259A 在系统中的 IO 地址是 FFDCH（A0 = 0）和 FFDEH（A0 = 1）。请编写 8259A 的初始化程序段。

```
        mov     al,   13h
        mov     dx,   offdch
        out     dx,   al
        jmp     intr1
intr1:  mov     al,   90h
        mov     dx,   0ffdeh
        out     dx,   al
        jmp     intr2
intr2:  mov     al,   1
        mov     dx,   0ffdeh
        out     dx,   al
```

7.9 PC 系列机中设定的 8259A 采用何种优先权方式和中断结束方式？它们的主要特点是什么？

- 利用上升沿做为中断请求 IRQ 的有效信号
- IRQ0 ~ IRQ7 的中断向量号依次为 08H ~ 0FH，IRQ8 ~ IRQ15 依次为 70H ~ 77H
- 采用普通全嵌套优先权方式，中断优先权从高到低顺序为 IRQ0 ~ IRQ2、IRQ8 ~ IRQ15、IRQ3 ~ IRQ7，且不能改变
- 采用普通中断结束 EOI 方式，需要在中断服务程序最后发送普通 EOI 命令
- 一般采用普通屏蔽方式，通过写入 IMR 允许中断，但注意不要破坏原屏蔽状态

7.10 8259A 的中断请求有哪两种触发方式，它们分别对请求信号有什么要求？PC 系列机中采用哪种方式？

✧ 边沿触发方式

8259A 将中断请求输入端出现的上升沿作为中断请求信号

✧ 电平触发方式

中断请求端出现的高电平是有效的中断请求信号

7.11 下段程序读出的是 8259A 的哪个寄存器？

```
mov     al,   0bh
out     20h, al
nop
in      al,   20h
```

- 读取中断服务寄存器 ISR 的内容
- 因为执行输入指令(A0 = 0)之前，执行的输出指令，写入了 OCW3 (D4D3 = 01)．其中 P RR RIS (D2D1D0 = 011)指明随后读出 ISR
- 不是查询字

7.12 PC 系列机执行了下面两条指令后，会产生什么控制状态？

```
mov     al,   0bch      ; 0bc = 1011 1100
out     21h, al         ;
```

7.13 下面是 XT 机 ROM-BIOS 中的 08 号中断服务程序，请说明各个指令的作用。

```
Int08h      proc    far                 ; 远过程
            sti                         ; 开中断，允许中断嵌套
            push    ds                  ; 现场保护
            push    ax
            push    dx
            ……          ;
                ;是时钟计时             ; 日时钟计时
            ……          ;
                ;控制软驱马达           ; 控制软驱马达
            int     1ch                 ; 调用指令中断 1CH
            mov     al,  20h            ; 发送 EOI 命令
            out     20h,     al
            pop     ax                  ; 现场恢复
            pop     dx
            pop     ds
            iret                        ; 中断返回
int08h      endp
```

7.14 中断服务程序的入口处为什么通常要使用开中断指令？

开中断，以便可以实现中断嵌套

7.15 编写一个程序，将例题 INT 80H 内部中断服务程序驻留内存。然后在调试程序或其他程序中执行 INT 80H，看能否实现其显示功能。

```
；代码段
        jmp  start
；80H 内部中断服务程序：显示字符串（以 0 结尾）；DS：DX = 缓冲区首地址
new80h  proc        ；过程定义
        sti         ；开中断
        push    ax  ；保护寄存器
        push    bx
        push    si
        mov     si,  offset intmsg
new1:   mov     al,  cs:[si]    ;获取欲显示字符
        cmp     al,  0    ；为 0 结束
        jz   new2
        mov     bx,  0    ;采用 BIOS 调用显示一个字符
        mov     ah,  0eh
        int  10h
        inc  si   ；显示下一个字符
        jmp  new1
new2:   pop  si         ;恢复寄存器
        pop  bx
```

```
        pop  ax
        iret  ;中断返回
intmsg  db   'A instruction interrupt !' , 0dh,0ah,0;   字符串以 0 结尾
new80h  endp      ；中断服务程序结束

；主程序
start:  mov ax,  cs
        mov ds,  ax   ;设置 04H 中断向量
        cli
        mov ax,  2580h
        int   21h
        sti
        mov ax,  offset    tsrmsg   ;显示安装信息
        call  dispmsg
        mov dx,  offset    start；计算驻留内存程序的长度
        add  dx,  15
        shr  dx,  4    ；调整为以 16 个字节为单位
        mov ax,  3100h    ；程序驻留，返回 DOS
        int   21h
tsrmsg  db   'INT 80h Program Installed !', 0dh, 0ah, 0
```

7.16 PC 系列机的 ICH 号中断每隔 55ms 被调用一次，它是内部中断还是外部中断？

该中断由 8253 定时器产生，位于 CPU 之外， 是外部中断。

第8章　　定时计数器

8.1 微机中实现定时控制的主要方法是什么？

- ✧ 软件延时
- ✧ 不可编程的硬件定时
- ✧ 可编程的硬件定时

8.2 8253 每个计数通道与外设接口有哪些信号线，每个信号的用途是什么？

- ✧ CLK 时钟输入信号——在计数过程中，此引脚上每输入一个时钟信号（下降沿），计数器的计数值减 1
- ✧ GATE 门控输入信号——控制计数器工作，可分成电平控制和上升沿控制两种类型
- ✧ OUT 计数器输出信号——当一次计数过程结束（计数值减为 0），OUT 引脚上将产生一个输出信号

8.3 8253 每个通道有 6 种工作方式可供选择。若设定某通道为方式 0 后，其输出引脚为 低 电平；当 写入计数初值（并进入减 1 计数器） 后通道开始计数，CLK 信号端每来一个脉冲 减 1 计数器 就减 1，当 计数器减为 0 ，则输出引脚输出 高 电平，表示计数结束。8253 的 CLK0 接 1.5MHz 的时钟，欲使 OUT0 产生频率为 300KHz 的方波信号，则 8253 的计数值应为 5（＝1.5MHz÷300KHz），应选用的工作方式是 方式 3 。

8.4 试按如下要求分别编写 8253 的初始化程序，已知 8253 的计数器 0 – 2 和控制字 IO 的地址依次为 204H – 207H。

（1） 使计数器 1 工作在方式 0，仅用 8 位二进制计数，计数初值为 128

（2） 使计数器 0 工作在方式 1，按 BCD 码计数，计数值为 3000

（3） 使计数器 2 工作在方式 2，计数值为 02F0H。

```
（1）   mov   al,  50h
        mov   dx,  207h
        out   dx,  al
        mov   al,  128      ；80h
        mov   dx,  205h
        out   dx,  al

(2)     mov   al,  33h
        mov   dx,  207h
        out   dx,  al
        mov   ax,  3000h    ；不是 3000
        mov   dx,  204h
        out   dx,  al
        mov   al,  ah
        out   dx,  al

(3)     mov   al,  0b4h
        mov   dx,  207h
```

```
out     dx,  al
mov     al,  02f0h
mov     dx,  206h
out     dx,  al
mov     al,  ah
out     dx,  al
```

8.5 设 8253 计数器 0 – 2 和控制字的 IO 地址依次为 F8H – FBH，说明如下程序段的作用。

```
mov     al,     33h
out     0fbh,   al
mov     al,     80h
out     0f8h,   al
mov     al,     50h
out     0f8h,   al
```

; 33h = 0011 0011b
; 写入计数器 0 地址：0fbh

; 写入低字节计数初值

; 写入高字节计数初值

作用：计数器 0 的计数初值为 5080h

8.6 PC 机中如何应用 8253 每个通道的？

计数器 0——每隔 55ms 产生一个 IRQ0 中断请求

计数器 1——每隔 15μs 产生一个 DRAM 刷新请求

计数器 2——控制扬声器音调

8.7 例 8-2 中 CLK0 端实际输入多少个下降沿后产生中断？按照要求，还可以采用 8253 的什么工作方式完成同样的功能？如果利用外部信号启动计数，则 GATE0 应怎样使用，应选用什么工作方式？写出初始化程序。

101 个下降沿，还可以采用方式 4

GATE0 接外部启动计数器的控制信号，可以选用方式 1 或方式 5

```
mov     dx,  203h
mov     al,  12h        ;方式 5 为 1ah
out     dx,  al
mov     dx,  200h
mov     al,  64h
out     dx,  al
```

8.8 某系统中 8253 芯片的计数器 0 – 2 和控制字端口地址分别是 FFF0H – FFF3H。定义计数器 0 工作在方式 2，CLK0 = 5MHz，要求输出 OUT0 = 1KHz 频率波。定义通道 1 工作在方式 4，用 OUT0 作计数脉冲，计数值为 1000，计数器计到 0，向 CPU 发中断请求信号，接于 PC 系列机 IRQ4。编写 8253 两个计数器通道的初始化及中断服务程序入口地址、中断屏蔽位设置程序，并画出两个计数器通道的连接图。

计数器 0 的计数值：5M/1K＝5000＝1388H

方式控制字：00100101＝25H、2DH、35H、3DH
（十进制计数）

00100100＝24H、2CH、34H、3CH
（二进制计数）

计数器 1 的计数值：1000

方式控制字：01101001＝69H、79H

（十进制计数）

01101000＝68H、78H

（二进制计数）

```
    mov     dx,  0fff3h
    mov     al,  25h        ;通道 0，只写高字节，方式 2，十进制
    out     dx,  al
    mov     dx,  0fff0h
    mov     al,  50h        ；计数初值 5000
    out     dx,  al
    mov     dx,  0fff3h
    mov     al,  69h        ；通道 1，方式 4
    out     dx,  al
    mov     dx,  0fff1h
    mov     al,  10h        ；计数初值 1000
    out     dx,  al
```

8.9 利用扬声器控制原理，编写一个简易乐器程序：

- 当按下 1 – 8 数字键时，分别发出连续的中音 1 – 7 和高音 i（对应频率依次为 524Hz、588Hz、660Hz、698Hz、784Hz、880Hz、988Hz 和 1048Hz）
- 当按下其他键时暂停发音。
- 当按下 ESC 键（ASCII 码为 1BH），程序返回操作系统。

```
stack   segment  stack
            dw   1024 dup(?)
stack   ends
data    segment
    freq    dw   8,2277.1,2029.2,1807.9,1709.4,1521.9,1355.9,1207.7,1138.5
data    ends
code    segment  'code'
    assume  cs:code,  ds:data,ss:stack
start:      mov      ax,  data
            mov      ds,  ax
again:      mov      ah,  01h
            in   21h
            cmp      al,  1BH
            jz   next
            cmp      al,  31H
            jb   next1
            cmp      al,  38H
            ja   next1
            and      al,  0fh
            mov      ah,  00h
            mov      si,  ax
            mov      bx,  offset     freq
```

```
                mov     ax,  [BX+SI]
                call    speaker
                call    speakon
                jmp  again
next1:          call    speakoff
                jmp  again
next:           mov     ax,  4c00h
                int  21h
speaker  proc
                push ax
                mov     al,  0b6h
                out     43h, al
                pop  ax
                out     42h, al
                mov     al,  ah
                out     42h, al
                ret
speaker  endp
;
speakon  proc
                push    ax
                in   al, 61h
                or   al, 03h
                out     61h,al
                pop  ax
                ret
speakon  endp
;
speakoff proc
                push    ax
                in      al,  61h
                and     al,  0fch
                out     61h, al
                pop  ax
                ret
speakoff endp
code     ends
         end  starts
```

8.10 计数器的定时长度和精度受脉冲输入信号频率和计数值影响。对于频率为 F 的脉冲输入，计数器输出的最小定时时间为 $1/f$ ；此时计数初值应为 1 。16 位计数器输出的最大定时时间是 $2^{16}/f$ 。当需要加大定时时间时，或者利用硬件方法进行多个计数器的级联；或者利用软件辅助方法，使计数单元扩大计数值。

第9章　　DMA

9.1　8237A 在什么情况下处于空闲周期和有效周期?

9.2　什么是 8237A 的单字节传送方式和数据块传送方式，两者的根本区别是什么？数据块传送方式和请求传送方式对 DREQ 信号有效有什么要求？

9.3　DMA 传送分成哪 3 种类型？3 种类型下的 8237A 的存储器和 IO 控制线如何有效？

9.4　8237A 有几种对其 DMA 通道屏蔽位操作的方法？

9.5　PC 机为什么设置 DMA 传送的页面寄存器？

9.6　设置 PC 机 8237A 通道 2 传送 1KB 数据，请给出其字节数寄存器编程。

9.7　PC 机进行软盘 DMA 传输前，若通道 2 的初始化过程 DMA-SETUP 返回标志 CF = 1，说明了什么？

9.8　PC 机 8237A 通道 2 传送的内存起始地址为 C8020H，请给出其地址寄存器编程。

9.9　XT 机执行了下面两条指令，会产生什么作用？

```
mov     al,     47h
out     0bh,    al
```

9.10 如下是利用 PC 机 DMA 通道 1 进行网络通信的传输程序。其中 ES：BX 中设置主存缓冲区首地址，DI 中设置传送的字节数，SI 中为模式字。请阅读此程序，为每条指令加上注释，并说明每个控制字的含义。若主机通过它发送数据，SI 应为何值？若主机通过它接收数据，SI 应为何值？

```
        mov     dx,     0ch
        mov     al,     0
        out     dx,     al
        mov     dx,     09h
        out     dx,     al
        or      ax,     01
        mov     dx,     es
        mov     cl,     04
        rol     ax,     cl
        mov     ch,     al
        and     al,     0f0h
        add     ax,     bx
        jnc     net1
        inc     ch
net1:   mov     dx,     02
        out     dx,     al
        mov     al,     ah
        out     dx,     al
        mov     al,     ch
        and     al,     0fh
        mov     dx,     83h
        out     dx,     al
        dec     ax
        mov     dx,     03
        out     dx,     al
        mov     al,     ah
        out     dx,     al
        mov     dx,     0ah
        mov     al,     1
        out     dx,     al
        mov     dx,     8
        mov     al,     60h
        out     dx,     al
        mov     dx,     08h
net2:   in      al,     dx
        and     al,     02h
        jz      net2
```

第10章　　并行接口

10.1 8255A 的 24 条外设数据线有什么特点？

24 条 IO 脚分成 3 个 8 位的端口：端口 A、端口 B、端口 C。都可编程设定为输入或输出，共有三种工作方式。端口 A 和端口 B 可作为 IO 数据端口。端口 C 作为控制或状态端口,C 口高四位和低四位分别与 A 端口和 B 端口配合使用,工作在方式 1 或方式 2。端口 C 的 8 个引脚可直接按位置位或复位。

10.2 8255A 两组都定义为方式 1 输入，则方式控制字是什么？此时方式控制字 D3 和 D0 两位确定什么功能？

方式控制字为 10110110B。方式控制字另外两位确定 C 口的高 4 位和低 4 位中的空余位的 I/O 状态。

10.3 总结 8255A 端口 C 的使用特点。

端口 C 的各位在方式 0 时分高 4 位和低 4 位分别设定 I/O 状态；在方式 1 和方式 2 中部分引脚用于做控制或状态线,C 口的高 4 位和低 4 位中的空余位的 I/O 状态仍分别可编程设定。对端口 C 的各位可分别位控输出

10.4 设定 8255A 的端口 A 为方式 1 输入，端口 B 为方式 1 输出，则读取端口 C 的数据的各位是什么含义？

C 口各引脚	A 口为方式 1 输入， 且 B 口为方式 1 输出时	
	C 口作用	
PC7	IO 引脚	
PC6	IO 引脚	
PC5	IBFA 引脚	
PC4	INTEA 引脚(STBA*)	
PC3	INTRA	
PC2		INTEB（ACKB*）
PC1		IBFB*
PC0		INTRB

10.5 对 8255A 的控制寄存器写入 B0H，则其端口 C 的 PC5 引脚是什么作用的信号线？

因为控制字为 10110000B，则说明端口 A 工作在方式 1 输入，PC3、PC6、PC7 被征用，而 PC5 仍为基本 I/O 线，这里为输出线。

10.6　10-2-2 节用 8255A 端口 A 方式 0 与打印机接口，如果采用端口 B，其他不变，请说明应该如何修改接口电路和程序。

修改电路：将端口 B 的 PB0 - PB7 接打印机的数据位 DATA0 - DATA7 即可
修改程序：将输出数据端口改为 FFFAH 即可

10.7 10-2-3 节用 8255A 端口 A 方式 0 与打印机接口，如果采用端口 B，其他不变，请说明应该如何修改接口电路和程序。

修改电路：PA0 - PA7 改为 PB -PB7；PC6 改用 PC2，PC7 改用 PC1，PC2 改用 PC0

修改程序：

```
        mov dx,  0fffeh
        mov al,  84h
        out  dx,  al
        mov al,  04h
             ；使 INTEB（PC2）为 0，禁止中断
        out  dx,  al
        ;
        mov cx,  counter   ;打印字节数送 CX
        mov bx,  offset  buffer ；取字符串首地址
        call  prints     ;调用打印子程序
        ;
prints  proc
        push ax   ;保护寄存器
        push dx
print1: mov al,  [bx] ；取一个数据
        mov dx,  0fffah
        out  dx,  al   ；从端口 B 输出
        mov dx,  0fffch
print2: in   al,  dx   ；检测（PC1）是否为 1
        jz   print2
        inc  bx
        loop print1
        pop  dx
        pop  ax
        ret
prints  endp
```

10.8 设一工业控制系统，有四个控制点，分别由四个对应的输入端控制，现用 8255A 的端口 C 实现对该系统的控制，如图题所示。开关 K0 – K3 打开则对应发光二极管 L0 – L3 亮，表示该控制点运行正常；开关闭合则对应发光二极管不亮，说明该控制点出现故障。编写 8255A 的初始化程序和这段控制程序。

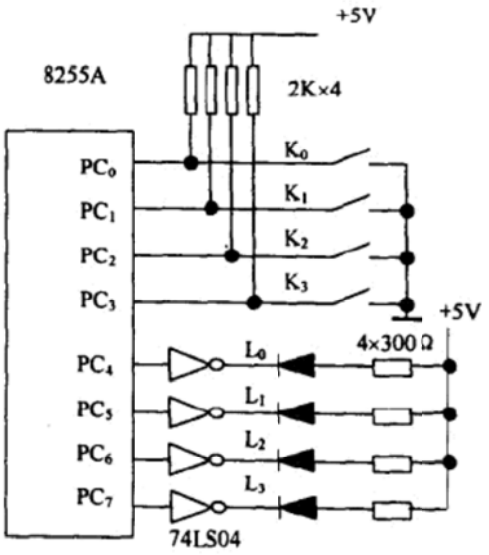


图11-22 习题11.8图

-303h

10.9 设定 8255A 的端口 B 为方式 1 连接某一输入设备，其中断请求信号引入 PC 机的 IRQ。欲使 CPU 响应该外设的中断请求，初始化时应开放 3 级中断，请编程说明。

```
data        segment
aport       equ         300h                ;假设地址为 300h-303h
bport       equ         301h
cport       equ         302h
conport     equ         303h
data        ends
code        segment
            assume      cs:code,ds:data
start:      mov         ax,data
            mov         ds,ax
            mov         dx,conport          ;8255 初始化
            mov         al,10000110b
            out         dx,al
            mov         al,00000101b        ;允许 8255a 的 b 口中断
            out         dx,al
            mov         ah,35h              ;获取原中断向量
```

```
        mov     al,   0bh
        int     21h
        push    es                      ;保存原中断向量入栈
        pop     bx
        cli                             ;关中断
        push    ds
        mov     dx,offset new0bh        ;设置新中断向量
        mov     ax,seg new0bh
        mov     ds,ax
        mov     ah,25h
        mov     al,0bh
        int     21h
        pop     ds
        in      al,   21h               ;读 imr
        push    ax                      ;保存原 imr 内容
        and     al,0f7h                 ;设置新 imr 内容
        out     21h,al
        sti                             ;开中断
        …                               ;主程序部分
        cli                             ;关中断
        pop     ax                      ;恢复原 imr
        out     21h, al
        pop     dx                      ;恢复原中断向量
        pop     ds
        mov     ah,   25h
        mov     al,0bh
        int     21h
        sti                             ;开中断
        mov     ax,       4c00h         ;返回 dos
        int     21h

new0bh  proc                            ;中断服务程序
        …
new0bh  endp
code    ends
            end       start
```

10.10 什么是机械按键的抖动，给出软、硬件解决抖动问题的方法。

- 当按下或释放一个键时,往往会出现按键在闭合位置和断开位置之间跳几下才稳定到闭合状态的现象就是机械按键的抖动

  解决方法:
- 硬件：消抖电路
- 软件：程序延时，以避开抖动的时间

10.11　什么是键盘识别中的重键，怎样解决这个问题。

- 重键是指两个或多外键同时闭合
- 简单情况：不予识别，认为是错误的按键
- 通常情况：只承认先识别出来的键
- 连锁法：直到所有键都释放后才读入下一个键
- 巡回法：等被识别的键释放以后，就可以对其他闭合键作识别，而不必等待全部键释放
- 正常的组合键：都识别出来

10.12　在 10-3-1 节的键盘接口电路中，假设 8255A 的数据端口 A、B、C 和控制端口地址为 218H – 22BH，写出完整的采用反转法识别按键的键盘扫描程序。

```
        ……
        ；设置行线接输出端口，列线接输入端口
key2:   mov al,00
        mov dx,rowport
        out dx,al ；设置行线全为低
        mov dx,colport
        in al,dx  ；读取列值
        cmp al,0ffh
        jz key2   ；无闭合键，循环等待
        push ax   ；有闭合键，保存列值
        push ax
        ；设置行线接输入端口，列线接输出端
        mov dx,colport
        pop ax
        out dx,al ；输出列值
        mov dx,rowport
        in al,dx  ；读取行值
        pop bx    ；组合行列值
        mov ah,bl     ；此时，al＝行值，ah＝列值
    ；键盘的行列值表
    table   dw  0fefeh  ；键 0 的行列值（键值）
        dw  0fdfeh  ；键 1 的行列值
        dw  0fbfeh  ；键 2 的行列值
        ……      ；其他键的行列值
        ；键盘的键代码表
    char db ……  ；键 0 的代码值
        db ……  ；键 1 的代码值
    ……      ；其他键的代码值
    mov si,offset table
        mov di,offset char
        mov cx,64     ；CX＝键的个数
    key3:   cmp ax,[si]   ；与键值比较
        jz key4   ；相同，说明查到
```

```
        inc si      ；不相同，继续比较
        inc si
        inc di
        loop key3
        jmp key1
        ；全部比较完，仍无相同，说明是重键
key4:   mov al,[di]     ；获取键代码送 AL
        ……
        ；判断按键是否释放，没有则等待
        call delay
        ；按键释放，延时消除抖动
        ……      ；后续处理
```

10.13　对照 10-3-2 节的键盘缓冲区，说明“先进先出、循环队列”的工作过程。

键盘缓冲区 BUFFER 共 10 个字节,按“先进先出、循环队列”原则建立循环队列，可存放 10 个按键字符，bufptr1 和 bufptr2 分别指向队列头和队列尾的指针单元。操作如图所示：

bufptr2分别指向队列头和队列尾的指针单元。先进先出循环队列的操作如图11-16所示。

（1）队列空：队列中无字符，队列头指针等于队列尾指针。

（2）进队列：数据进入由队列尾指针指示的单元，同时尾指针增量，指向下一个单元。

（3）出队列：数据从队列头指针指示的单元取出，同时头指针增量，指向下一个单元。

（4）队列满：当数据不断进入队列，使尾指针指向队列末端时（9号单元），尾指针循环重新绕回队列始端（0号单元）。如果继续到尾指针与头指针再次相等，则表明队列已满，不能再存入数据。

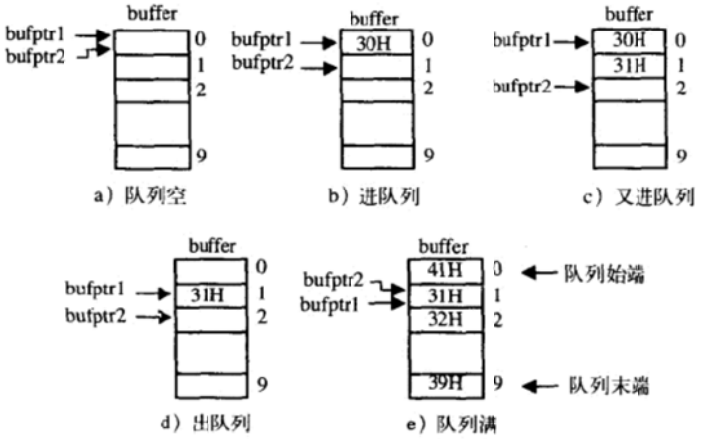


图11-16　先进先出循环队列的操作

10.14　编写一个程序，每当在键盘上按下一键时，就显示其接通和断开扫描码。

```
        ;数据段
done    byte 0
        ; 代码段，  主程序
        mov     ax, 3509h
        int     21h
        push    es
```

```
        push bx
        cli
        push ds
        mov dx,  seg new09h
        mov ds,  dx
        mov dx,  offset  new09h
        mov ax,  2509h
        int  21h
        pop  ds
        in   al,  21h
        push ax
        and  al,  0fdh
        out  21h, al
        sti
start1: cmp done, 1
        jne  start1
        cli
        pop  ax
        out  21h, al
        pop  dx
        pop  ds
        mov ax,  2509h
        int  21h
        sti
        ; 代码段，子程序
new09h  proc
        sti
        push ax
        push bx
        in   al,  60h
        push ax
        in   al,  80h
        out  61h, al
        and  al,  7fh
        out  61h, al
        pop  ax
        cmp al, 1
        je   next3
        push ax
        shr  al,  4
        cmp al,  0ah
        jb   next1
        add  al,  7
next1:  add  al,  30h
```

```
        mov bx, 0
        mov ah,  0eh
        int  10h
        pop  ax
        and  al,  0fh
        cmp al,  0ah
        jb    next2
        add  al,  7
next2:  add  al,  30h
        mov ah,  0eh
        int  10h
        mov ax,  0e20h     ;输出 2 个空格，分隔
        int  10h
        mov ax,  0e20h
        int  10h
        jmp  next4
next3:  push ds
        mov ax,  @data
        mov ds,  ax
        mov done,1
        pop  ds
next4:  mov al,  20h
        out  20h, al
        pop  bx
        pop  ax
        iret
new09h: endp
```

10.15　补充 10-4 节 LEDtb 指示的 0 – F 显示代码。

```
LEDtb   db  3fh, 06h, 5bh, 4fh, 66h, 6dh, 7dh, 07h, 7fh, 6fh, 77h, 7ch, 39h, 5eh, 79h, 71h
```

10.16　如图题为用一片 8255A 控制 8 个 8 段共阴极 LED 数码管的电路。现要求按下某个开关，其代表的数字（K1 为 1，K2 为 2，……，K8 为 8）在数码管从左到右循环显示（已有一个延时子程序 delay 可以调用），直到按下另一个开关。假定 8255A 的数据端口 A、B、C 及控制端口的地址依次为 FFF8H – FFFBH。编写完成上述功能的程序，应包括 8255A 初始化、控制程序和数码管的显示代码表。

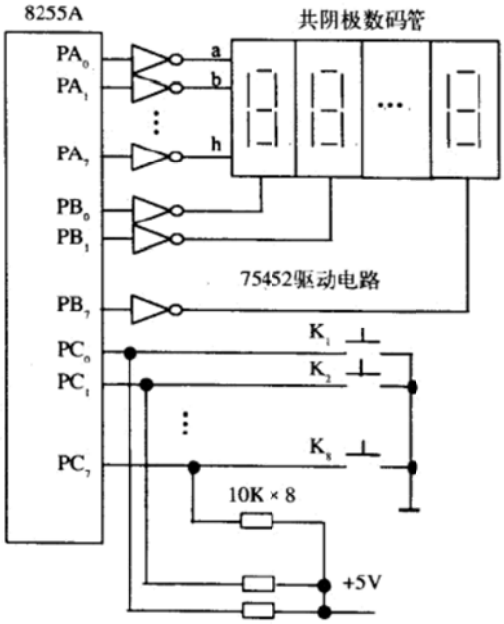


图11-23 习题11.16图

```
data    segment
aport   equ  0fff8h
bport   equ  0fff9h
cport   equ  0fffah
cwport  equ  0fffbh
segbuf  db       11h
bitbuf  db       01h
segtab  db       0c0h,0f9h,0a4h,0b0h,99h,92h,82h,0f8h
        db       80h,90h,88h,83h,0c6h,0a1h,86h,8eh
        db       0ffh,0bfh
data    ends

code    segment
        assume   cs:  code,     ds:  data
start:
        mov      ax,  data
        mov      ds,  ax
        mov      dx,  cwport
        mov      al,  10001001b      ;设置a口 b口为方式0输出,c口输入
        out  dx, al

key1:
        call disp                    ;调显示当前一位子程序
        mov      dx,cport            ;读键盘口
        in   al,dx
        cmp      al,0ffh             ;判有无键按下
        jz       key1                ;无键按下,返回继续查
```

```
            mov     cx,8                    ;有键按下,查找键值
            mov     ah,1
key2:     stc
shr   al,1
jnc   key_num                               ;查到键值,跳出去显示
inc   ah
loop      key2
jmp       key1

key_num:
            cmp      segbuf,ah              ;判与前键相同否
            jz       key1                   ;与前键相同,保持
            mov      segbuf,ah                  ;与前键不同,换新键值
            mov      bitbuf,01h                 ;重新指向左 led 管
            jmp      key1

p_end:    mov      ax,4c00h
            int   21h

delay_10ms    proc near
                  push     ax
                  push     cx
                  push     dx
                  mov      cx,  0
                  mov      dx,  5000
                  mov      ah,  86h
                  int      15h
                  pop      dx
                  pop      cx
                  pop      ax
                  ret
delay_10ms    endp

disp              proc
                  push     ax
                  push     bx
                  push     cx
                  push     dx
                  mov      bx,  offset segtab
                  mov      al,  segbuf
                  xlat
                  mov      dx,  aport
                  out      dx,  al
                  mov      al,  bitbuf
```

```
                mov     dx, bport
                out     dx, al
                rol     al, 1
                mov     bitbuf, al
                call    delay_10ms
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                ret
disp            endp

code    ends
    end start
```

10.17 Centronics 接口的前 11 个信号线的功能是什么？它们是怎样配合输出数据的？

- **一般采用 Centronics 标准接口或其简化接口**
- **Centronics 接口是的一个并行接口协议**
- **这个协议规定了 36 脚簧式插头座和信号含义**
- **其中前 11 条线是关键信号：**
  - **8 条数据线**
  - **3 条联络线（选通、响应和打印机忙）**
- **还有一些特殊控制线、状态线**
- **PC 系列机的并行打印机接口是一个 25 针插口**
  - **DATA0 ~ DATA7：8 位并行数据信号线，打印数据通过它们送至打印机**
  - **STROBE*：选通，用于使打印机接收数据的选通信号。负脉冲的宽度在接收端应大于 0.5μs**
  - **BUSY：忙，表示打印机不能接收数据**
  - **ACK*：响应，打印机接收一个数据字节后就回送一个响应的负脉冲信号（脉宽约为 5μs）**
  - **8 位数据的可靠输出通过选通 STROBE*、响应 ACK*和忙 BUSY 三个联络信号控制**

10.18 参照打印机 IO 功能程序，编写一个利用查询方式打印一个字符的子程序。假设不考虑超时错误，打印字符从 AL 传送至子程序，打印机基地址在 DX 中。

```
                mov cx,(sizeof okmsg)-1     ;获取打印字符数
                mov si,offset okmsg
                                ;获取打印字符首地址
prnbegin:       mov dx,378h   ;打印机基地址
                mov bx,100    ;超时参数，可视情况设置
                mov al,[si]   ;打印字符
                call printchar  ;调用字符打印子程序
                jc prnerr  ;返回 CF=1，打印出错
                inc si      ;返回 CF=0，打印正常
```

```
                loop prnbegin ;继续
printchar       proc
                push cx
                out dx,al  ;向数据端口输出打印字符
                inc dx     ;基地址加 1 成为状态端口地址
      print0:   sub cx,cx
      print1:   in al,dx    ;查询状态端口
                test al,80h      ;最高位 D7 反映打印机状态
                jnz print2 ;D7=1，打印机可以接收打印数据
                loop print1      ;D7=0，打印机不能接收打印数据
                dec bl     ;超时参数减 1
                jnz print0 ;循环检测
                mov al,20h     ;发送 EOI 命令
                out 20h,al
                pop bx
                pop ax
                iret  ;中断返回
scancode        endp
```

第11章　　串行接口

11.1 串行异步通信发送 8 位二进制数 01010101；采用起止式通信协议，使用奇校验和 2 个停止位。画出发送该字符的波形图。若用 1200bps，则每秒最多能发送多少个数据？

11.2 微机与调制解调器通过 232D 总线连接时常使用哪 9 个信号线，各自的功能是什么？利用 232D 进行两个微机直接通信时，可采用什么连接方式，画图说明。

11.3 8250 在识别起始位时采用什么方法防止误识别的？UART 芯片的接收方采用双缓冲或多缓冲结构，是为了防止发生什么错误？

11.4 8250 芯片能管理哪 10 个中断，并说明各个中断分别在何时产生？

11.5 欲使通信字符为 8 个数据位、偶校验、2 个停止位，则应向 8250_____寄存器写入控制字_____，其在 PC 系列机上的 IO 地址（COM2）是_____。XT 机通信适配器电路上设计 J9 – J12 跨接器的作用是_____。

11.6 PC 系列机执行以下 3 条指令后，将设定什么状态？

11.7 8250 的 IIR 是只读的，且高 5 位总是 0。试分析 XT 机系统 ROM – BIOS 中如下程序段的作用。如不发生条件转移，则 RS232 – BASE 字单元将存放什么内容？

11.8 设定某次串行异步通信的数据位为 8 位、无校验、1 个停止位，传输速率为 4800bps，采用中断工作方式。按此要求写出 PC 系列机中对第 2 个串行通信口的初始化程序。

11.9 8250 的除数寄存器、8253 的计数器，8237A 的通道寄存器都是 16 位的，但这 3 个芯片的数据线都是 8 位的。它们分别采用什么方法通过 8 位数据线操作 16 位寄存器？

第12章　　模拟接口

12.1 说明在模拟输入输出系统中，传感器、放大器、滤波器、多路开关、采样保持器的作用。DAC 和 ADC 芯片是什么功能的器件？

12.2 如果将 DAC0832 接成直通工作方式，画冬说明其数字接口引脚如何连接。

12.3 对应 12-2-4 节的图 12-9a 电路，编写输出一个 12 位数字量的程序段。假定这 12 位数据在 BX 的低 12 位中。

12.4 假定某 8 位 ADC 输入电压范围是 –5V – +5V，求出如下输出电压 Vin 的数字量编码（偏移码）：（1）1.5V,　　(2)2V,　　(3)3.75V,　　(4) –2.5V　　(5) –4.75V

12.5 ADC 的转换结束信号起什么作用，如何使用该信号，以便读取转换结果？

12.6 某控制接口电路如图 12-16 所示。需要控制时，8255A 的 PC7 输出一个正脉冲信号，START 启动 AD 转换；ADC 转换结束在提供一个低脉冲结束信号 EOC 的同时送出数字量。CPU 采集该数据，进行处理，产生控制信号。现已存在一个处理子程序 ADPRCS，其入口参数是在 AL 寄存器存入待处理的数字量，出口参数为 AL 寄存器给出处理后的数字量。假定 8255A 端口 A、B、C 及控制端口的地址依次为 FFF8H – FFFBH，要求 8255A 的端口 A 为方式 1 输入，端口 B 为方式 0 输出。编写采用查询方式读取数据，实现上述功能的程序段。

12.7 假设系统扩展一片 8255A 供用户使用，请设计一个用 8255A 与 ADC0809 接口的电路连接图，并给出启动转换，读取结果的程序段。为简化设计，可只使用 ADC0809 的一个模拟输入端，例如 IN0。

第13章　　32 位微机

13.1 IA –32 是指_____，与其兼容的 64 位指令集结构，被称为_____。它们的指令系统有通用整数指令外，还包括_____和多媒体指令。

13.2 IA – 32 处理器如何将 16 位通用寄存器扩展为 32 位，同时又保持兼容？

13.3 什么是 IA – 32 处理器的实地址方式、保护方式和虚拟 8086 方式？它们分别使用哪种存储模型？

13.4 在以 BP、EBP、ESP 作为基址寄存器访问存储器操作数时，其默认的段寄存器是_____；但是，通常 ESP 作为_____，不应该将其用于其他目的。

13.5 解释下列指令如何计算存储器操作数的单元地址：

13.6 简答下列问题

13.7 假设 ARRAY 是定义了若干 16 位整数的一个变量，阅读如下程序，为每条指令加上注释，并说明该过程的功能。

```
sum16       proc
            mov     ebx,    offset  array
            mov     ecx,    3
            mov     ax,     [ebx + 2*ecx]
            mov     ecx,    5
            add     ax,     [ebx + 28ecx]
            mov     ecx,    7
            add     ax,     [ebx + 2*ecx]
            ret
sum16       endp
```

13.8 已知 BF600000H 是一个单精度规格化浮点格式数据，它表达的实际数据是什么？

13.9 实际数据真值 28.75 如果用单精度规格化浮点数据格式表达，其编码是什么？

13.10　　利用 32 位扩展指令编写运行在 DOS 环境的源程序，应该注意哪些方面的问题？

13.11　　什么是紧缩数据类型？IA – 32 处理器支持哪些紧缩数据格式？为什么称多媒体指令为 SIMD 指令？

13.12　编写一个十进制显示子程序，用在例 13-1 中将排序后的数据显示出来。

13.13　Pentium 的 3 个最基本的读控制引脚是 M/IO*、_____和_____。USB 总线理论上最多能够连接_____个 USB 设备，USB2.0 支持低速_____、全速_____和高 480Mb/s 三种速率。

13.14　32 位 PC 机为什么采用多级总线结构，而不是单总线结构？

13.15　简述 USB 总线的主要特征。

13.16　在层次结构的存储系统中，高速缓存、主存和辅存的作用各是什么？虚拟存储器指是的什么？

13.17　什么是存储访问的局部性原理？它又分成哪两种局部性？

访问的局部性原理：在一个较短的时间间隔内，由程序产生的地址往往集中在存储器逻辑地址空间的很小范围内。指令地址的分布本来就是连续的，再加上循环程序段和子程序要重复执行多次。因此，对这些地址的访问就自然地具有时间上集中分布的倾向。数据分布的这种集中倾向不如指令明显，但对数组的存储和访问以及工作单元的选择都可以使存储器地址相对集中。这种对局部范围的存储器地址频繁访问，而对此范围以外的地址则访问甚少的现象，就称为程序访问的局部性。

cache 的作用：提高对存储器的访问速度。

虚拟存储：其目标是扩大程序员眼中的主存容量。

13.18　说明在 32 位保护方式下，通过段页式存储管理寻址一个操作数的过程。

13.19　简单说明如下名词（概念）的含义：

(1) L1 Cache 和 L2 Cache
(2) RISC 和 CISC
(3) 指令预取
(4) 指令流水线
(5) 超标量技术
(6) 动态执行技术
(7) 指令级并行
(8) 线程级并行
(9) 超线程技术
(10) 多核技术

13.20　选择微机某个方面，例如微处理器芯片、主板组成、总线结构、控制芯片组等，追踪其技术发展，搜集其最新资料，写一篇新技术发展的论文。