IBM—PC

汇编语言程序设计

例题习题集

温冬婵 沈美明 编著

清华大学出版社

IBM PC 汇编语言程序设计

例 题 习 题 集

温冬婵　沈美明　编

清华大学出版社

内 容 简 介

本书为《IBM PC [0520]汇编语言程序设计》、《IBM PC 汇编语言程序设计》二本书的教学参考书，共收集、编写了约 300 道复习练习题。按照由指令到程序设计的系统分为八章，每章包括复习提要、例题分析及习题三个部分，并给出部分答案。书后还附有二份自测题及其答案。本书在练习的形式和内容上，突出了基础知识的复习巩固，也注意了程序设计能力的培养和提高，既适用于大、中学生学习汇编语言，也适用于教师和工程技术人员作参考。

（京） 新登字 158 号

IBM PC 汇编语言程序设计例题习题集

温冬婵 沈美明 编
责任编辑 贾仲良
☆
清华大学出版社出版
北京 清华园
国防工业出版社印刷厂印刷
新华书店总店科技发行所发行
☆
开本：787×1092 1/16 印张：7.75 字数：198 千字
1991 年 6 月第 1 版 1992 年 11 月第 3 次印刷
印数：27001 — 42000
ISBN 7-302-00756 X/TP • 254
定价：3.60 元

前　言

近年来，由于微型计算机的迅速发展和广泛应用，越来越多的技术人员在系统设计或技术开发中需要分析汇编语言程序或编制汇编语言程序。在国内外的中、高等院校中，汇编语言程序设计也是计算机专业学生必修的专业基础课程之一. 为了帮助在校学生和自学汇编语言程序设计的工程技术人员掌握本课程的主要内容和学习重点，我们应广大读者的要求，配合清华大学出版社出版的《IBM PC [0520]汇编语言程序设计》一书，编写了这本习题集，供学习时使用和参考。

根据初学者的特点，本书按照由浅入深，由指令到程序设计的系统编写。每章包括复习提要、例题分析和习题三个部分。书后还附有自测题和部分习题的参考答案。书中有很多例题和习题可编写成完整的程序，在 IBM PC XT/AT 或 PC 兼容机上运行。如果练习编写的程序经过上机调试，则更有助于学习和掌握汇编语言程序设计的基本方法和技巧。

由于我们的水平有限，进行 IBM PC 汇编语言程序设计教学和实践的时间也不长，可能会出现错误和不妥之处，恳请广大读者批评指正。

编　者

1989 年于清华大学

目　　录

第一章　数和字符的表示法

复　习　提　要

1. 计算机只能识别以二进制形式存在的机器语言。计算机内部数的存储及运算也都是采用二进制。

2. 一个二进制数的值由 1 所在位置的权来确定。如$(1101)_2=2^3+2^2+2^0=(13)_{10}$

3. 二进制的负数用补码来表示。对一个二进制数求补(按位求反,末位再加 1),即得到这个数相反数的补码表示。

4. 补码的加法和减法的规则是:

$$[X+Y]_{补}=[X]_{补}+[Y]_{补},\quad [X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

5. 16 进制是一种很重要的短格式计数法,它把二进制数每 4 位分成一组,分别用 0～9 和 A～F 来表示 0000～1111。反之,16 进制数的每一位用四位二进制表示,就是相应的二进制数。

6. 十进制转换为二进制数的方法主要有降幂法和除法。计算机十化二程序中采用下面的算法:

$$N_mN_{m-1}\cdots N_1N_0=(\cdots((N_m+0)\times 10+N_{m-1})\times 10+\cdots+N_1)\times 10+N_0$$

7. 标志位 OF＝1 表示带符号数的运算结果无效。CF＝1 表示无符号数的运算结果无效。

8. 计算机中的字符数据用 ASCII 码表示,一个字符在存储器中占用一个字节(8 位二进制码)。

9. BCD 码是一种用二进制编码的十进制数,又称二-十进制数或 8421 码,它用 4 位二进制数表示一个十进制数码。BCD 码有压缩和非压缩两种格式。压缩的 BCD 码用 4 位二进制数表示一个十进制数位,如 95_{10}表示为 1001,0101。非压缩的 BCD 码用 8 位二进制数表示一个十进制数位。如 95_{10}表示为 00001001 00000101。

例　题　分　析

例 1.1　完成下列各式补码数的运算,并根据结果设置标志位 SF、ZF、CF 和 OF,指出运算结果有效否。

(1) 0100,1001b＋1001,1101b

(2) 0100,0001b－1010,1011b

(3) 0A95Bh＋8CA2h

(4) 6531h－42DAh

解:

(1)　0100,1001　　SF ZF CF OF

＋1001,1101　　1　0　0　0　　运算结果有效

1110,0110

这两个二进制补码数相加，和为一个非零的负数(SF＝1，ZF＝0)；最高有效位无进位(CF＝0)；正负两数相加，结果不会溢出(OF＝0)。

(2)　0100,0001　　SF ZF CF OF

＋0101,0101　　1　0　1　1　　结果无效

1001,0110

两个补码数相减，先对减数求补，(1010,1011 求补后得 0101,0101)，再把减法转化为加法进行运算。运算结果为非零负数(SF＝1，ZF＝0)；最高位无进位(CF＝1)；结果符号与减数相同(均为负)，则 OF＝1，结果是错误的。

(3)　A95B　　SF ZF CF OF

＋8CA2　　0　0　1　1　　结果无效

1 35FD

这两个十进制数相加，所得结果是一个非零的正数(SF＝0，ZF＝0)；最高位有进位(CF＝1)；两负数相加，结果为正数，故 OF＝1，结果无效。

(4)　6531　　SF ZF CF OF

＋BD26　　0　0　0　0　　结果有效

1 2257

先对减数求补(42DA 求补后得 0BD26)，然后用加法进行运算。运算结果为非零正数(SF＝0，ZF＝0)；最高位有进位(CF＝0)；结果符号与减数不同(OF＝0)，结果正确。

例 1.2　把字符串“PART1：Memory”存放在 1100 开始的存储区中，请写出字符串的存储情况。

字　符：	P	A	R	T	1	：	M	e	m	o	r	y
ASCII 码：	50	41	52	54	31	3A	4D	65	6D	6F	72	79
地　址：	1100	1101	1102	1103	1104	1105	1106	1107	1108	1109	110A	110B

IBM PC 的存储器按字节编址，一个 ASCII 码字符占用一个字节。

例 1.3　写出十进制数 3590 的非压缩 BCD 码和压缩的 BCD 码，并分别把它们存入数据区 UNPAK 和 PAKED。

UNPAK	
＋0	0 0
＋1	0 9
＋2	0 5
＋3	0 3

PAKED	
＋0	9 0
＋1	3 5

3590 的非压缩 BCD 码为 0000 0011，0000 0101，0000 1001，0000 0000；压缩的 BCD 码为 0011 0101，1001，0000。它们以相反的顺序存储在数据区中。

习　题

1.1　用降幂法和除法将下列十进制数转换为二进制数和 16 进制数。

(1) 369　　(2) 10000　　(3) 4095　　(4) 32767

1.2 将下列二进制数转换为 16 进制数和十进制数。

(1) 101101 (2) 10000000 (3) 1111111111111111 (4) 11111111

1.3 将下列 16 进制数转换为二进制数和十进制数。

(1) FA (2) 5B (3) FFFE (4) 12D4

1.4 完成下列二进制数的加法：

(1) 00010101 (2) 00111110 (3) 11111111 (4) 00111001

+00001101 +00101001 +00000001 +01000111

1.5 完成下列 16 进制数的加法：

(1) 23A6 (2) 51FD (3) 7779 (4) EABE

+0076 +0036 +0887 +26C4

1.6 完成下列八位二进制数的减法：

(1) 00011111 (2) 00011001 (3) 10101010 (4) 00111000

−00000101 −00010000 −00010101 −11111111

1.7 完成下列十六进制数的减法：

(1) FFFF (2) 12AA

−AAAA −02AB

1.8 写出下列二进制数的补码表示：

(1) −00010011 (2) −00111100 (3) −00111001 (4) −01000000

1.9 写出下列补码数的相反数：

(1) 11001000 (2) 10111101 (3) 10000000 (4) 00000000

1.10 下列各数均为十进制数，请用 8 位二进制补码计算下列各题，并用 16 进制数表示其运算结果，同时说明进位值，并判断是否溢出。

(1) (−85)+76 (2) 85+(−76) (3) 85−76 (4) 85−(−76)

(5) (−85)−76 (6) (−85)−(−76)

1.11 把下列数和字符用 16 进制表示出来：

(1) 字母 Q (2) ASCII 码 7 (3) 二进制数 10111101

(4) 十进制数 100 (5) 空格(space) (6) ? 符

1.12 下列各数均为用 16 进制表示的 8 位二进制数，请说明当它们分别看作补码表示的数及字符的 ASCII 码时，它们所表示的十进制数及字符是什么？

(1) 4F (2) 2B (3) 73 (4) 59

1.13 请写出下列字符串的 ASCII 码。

For example,

This is a number 3692.

1.14 分别用 8 位二进制和二位 16 进制数写出下列十进制数的补码表示：

(1) 16 (2) 100 (3) 127 (4) 0

(5) −16 (6) −100 (7) −128 (8) −1

1.15 16 位的二进制补码数所能表示的十进制最大数和最小数分别是什么？

1.16　16 位二进制数所能表示的无符号数的范围是多大?

1.17　假设两个二进制数 A＝01101010,B＝10001100,试比较它们的大小。

(1) A、B 两数均为带符号的补码数。

(2) A、B 两数均为无符号数。

1.18　有一个 16 位的数值 0101,0000,0100,0011,

(1) 如果它是一个二进制数,和它等值的十进制数是多少?

(2) 如果它们是 ASCII 码字符,则是些什么字符?

(3) 如果是压缩的 BCD 码,它表示的数是什么?

1.19　求出以下各 16 进制数与 62A0H 的和,并根据结果设置标志位 SF、ZF、CF 和 OF:

(1) 1234　　(2) 4321　　(3) CFA0　　(4) 9D60

1.20　求出以下各 16 进制数减去 4AE0H 的差值,并根据结果设置标志位 SF、ZF、CF 和 OF:

(1) 1234　　(2) 5D90　　(3) 9090　　(4) EA04

1.21　从键盘输入一个十进制数 4096 存入缓冲区 BUFFER,再将该数所对应的非压缩 BCD 码和压缩的 BCD 码分别存入 UNPAK 和 PAKED 数据项,请写出 BUFFEK、UNPAK 和 PAKED 中各字节的内容。

1.22　变量 INCOME 用伪操作 DW 定义了一个 5 位的十进制数,请将此数以压缩和非压缩 BCD 码的格式,用伪操作 DB 定义在变量 INCOME1 和 INCOME2 中。

```
INCOME  DW  32767
```

第二章　IBM PC 计算机组织

复　习　提　要

1. PC XT/AT 的核心是 8088/80286 的微处理器，它能完成存取字或字节并对其进行处理等功能，也能对存储器进行管理和保护。

2. 两种类型的内部存储器是 ROM(只读存储器)和 RAM(随机存储器)。存储器按字节编址，存储器地址一般用 16 进制的无符号数表示。

3. 字数据在存储器中存放的顺序为高地址字节存放高 8 位，低地址字节存放低 8 位。

4. AX、BX、CX 和 DX 是通用寄存器，每个通用寄存器可作两个 8 位寄存器使用(如 AH 和 AL)。

5. 一个 20 位的物理地址可表示成段地址：偏移地址。计算存储器单元的物理地址，可将段地址乘以 10H，再加上偏移地址。

$$\text{物理地址} = (\text{段地址} \times 10H) + \text{偏移地址}$$

6. 一个程序可包括四个段：代码段包含可执行的指令；堆栈段包含一个后进先出的数据区，它可保存程序的返回地址，数据以及寄存器的内容；数据段是程序的数据区；附加段也是一个数据区，它通常和数据段定义在同一存储区。

7. 段寄存器 CS、SS、DS 和 ES 分别寄存代码段、堆栈段、数据段和附加段的段地址。

8. 指针寄存器 SP 和 BP，如无特殊说明，它们指示堆栈段内存储单元的地址。

9. 堆栈的最高地址叫做栈底，堆栈指示器 SP 总是指向栈顶。

10. 堆栈存取必须以字为单位。一个字存入堆栈时，SP 减 2，再把字数据存入 SP 所指示的字单元中。从堆栈中取出一个字时，将 SP 所指示的字单元中的数据取出，然后 SP 加 2。

11. 变址寄存器 SI 和 DI 一般指示数据段内单元的地址，有时也可作为数据寄存器用。

12. 16 位的标志寄存器包括 6 个状态标志(SF、ZF、PF、CF、AF、OF)和 3 个控制标志(DF、IF、TF)。CF、AF、SF、ZF 和 OF 反映了算术运算以及移位、循环、逻辑等操作的结果状态。

例　题　分　析

例 2.1　一个有 16 个字的数据区，它的起始地址为 70A0:DDF6，请写出这个数据区首末字单元的物理地址。

解：首地址为：

(70A0×10H)+0DDF6=7E7F6H

末地址为：

7E7F6H+(10H−1)×2=7E814H

物理地址是一个 20 位的数值，分别由 16 位的段地址和 16 位的偏移地址来表示。数据区的最后一个字的地址为：首地址+(字数−1)×2

例 2.2 假设堆栈段寄存器 SS 的内容为 2250，堆栈指示器 SP 的内容为 0140，如果在堆栈中存入 5 个数据，SS 和 SP 的内容各是什么？如果又从堆栈中取出 2 个数据，SS 和 SP 的内容又各是什么？

答：在堆栈中存入 5 个数据后，SP 的内容变为 0136(0140－2×5＝0136H)。如又取出 2 个数据，SP 的内容又成为 013A，(0136＋4＝013A)。SS 的内容不变。

往堆栈中存入一个数(只能以字为单位)时，(SP)←(SP)－2，然后把数存入 SP 指示的地址。

从堆栈中取数据时，先将 SP 所指示的地址的内容取出，然后(SP)←(SP)＋2

习　题

2.1　一台微型计算机的字长为 16 位，如果采用字节编址，那么它可以访问的最大存储空间是多少字节？试用 16 进制数表示该机的地址范围。

2.2　IBM PC 机的 I/O 端口号通常是由 DX 寄存器来提供的，但有时也可以在指令中用一个字节来表示端口号。试问可以直接由指令指定的 I/O 端口数是多少？

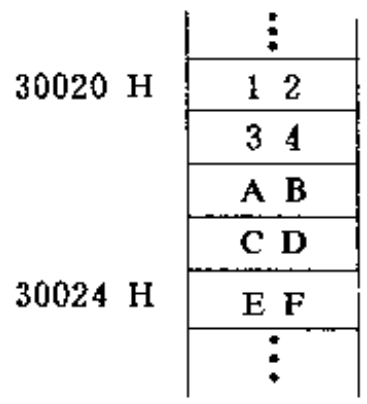
2.3　IBM PC 机有哪两种主要的存储器？它们所起的主要作用是什么？

2.4　有两个 16 位字 1EF5 和 2A3C 分别存放在 PC 机存储器的 000B0H 和 000B3H 单元中，请用图表示出它们在存储器里的存放情况。

2.5　将下列字符的 ASCII 码依次存入 00100 开始的字节单元中。(用图标出各单元的地址及其内容)。

IBM PC PERSONAL COMPUTER

2.6　在 PC 机的存储器中存放的信息如下图所示，试读出 30022，30024 字节单元的内容以及 30021、30022 字单元的内容。



2.7　写出下列存储器地址的段地址、偏移地址和物理地址。

(1) 2314：0035　　(2) 1FD0：000A

2.8　如果在一个程序段开始执行之前，(CS)＝0A7F0H，(IP)＝2B40H，试问该程序段的第一个字的物理地址是什么？

2.9　存储器中的每一段最多可含有 64K 个字节(1K＝1024)，假设用 DEBUG 命令显示出当前各寄存器的内容如下，请画出此时存储器分段的示意图，以及状态标志 OF、SF、ZF、CF 的值。

```
A> debug
-r
AX=0000  BX=0000  CX=0080  DX=0000  SP=0000
```

BP=0000　SI=0000　DI=0000　DS=10E4　ES=10F4

SS=21F0　CS=31FF　IP=0000　NV UP DI PL NZ NA PO NC

2.10　下列操作可使用哪些寄存器?

(1) 加法和减法;　(2) 循环计数;　(3) 乘法和除法;　(4) 保存段地址;

(5) 表示运算结果的特征;　(6) 将要执行的指令地址;　(7) 将要从堆栈中取出数据的地址。

2.11　IBM PC 有哪些寄存器可用来指示存储器的地址?

2.12　如果一个堆栈从地址 1250：0000 开始,它的最后一个字的偏移地址为 0100H,SP 的内容为 0052H,

问:(1) 栈顶地址是什么?

(2) 栈底地址是什么?

(3) 在 SS 中的段地址是什么?

(4) 存入数据 3445H 后,SP 的内容是多少?

2.13　请将左边的词汇和右边的说明联系起来,括号内填入所选的 A,B,C…。

(1) CPU　(　)　A. 保存当前栈顶地址的寄存器。

(2) 存储器　(　)　B. 指示下一条要执行的指令的地址。

(3) EU　(　)　C. 总线接口部件,实现执行部件所需要的所有总线操作。

(4) BIU　(　)　D. 分析并控制指令执行的部件。

(5) 堆栈　(　)　E. 存储程序、数据等信息的记忆装置,PC 机有 RAM 和 ROM 两种。

(6) IP　(　)

(7) SP　(　)　F. 以后进先出方式工作的存储器空间。

(8) 状态标志　(　)　G. 把汇编语言程序翻译成机器语言程序的系统程序。

(9) 控制标志　(　)　H. 唯一代表存储器空间中的每个字节单元的地址。

(10) 段寄存器　(　)　I. 能被计算机直接识别的语言。

(11) 物理地址　(　)　J. 用指令的助记符、符号地址、标号等符号书写程序的语言。

(12) 汇编语言　(　)

(13) 机器语言　(　)　K. 把若干个模块连接起来成为可执行文件的系统程序。

(14) 汇编程序　(　)　L. 保存各逻辑段的起始地址的寄存器。PC 机有四个寄存器 CS、DS、SS、ES。

(15) 连接程序　(　)

(16) 目标码　(　)　M. 控制操作的标志,PC 机有三位:DF、IF、TF。

(17) 指令　(　)　N. 记录指令操作结果的标志,共六位:OF、SF、ZF、AF、PF、CF

(18) 伪指令　(　)

O. 执行部件,由算逻单元(ALU)和寄存器组等组成。

P. 由汇编程序在汇编过程中执行的指令。

Q. 告诉 CPU 要执行的操作(一般还要指出操作数地址),在程序运行时执行。

R. 机器语言代码。

第三章　IBM PC 指令系统与寻址方式

复　习　提　要

1. 寻址方式小结

寻址方式	操作数地址(PA)	指令格式举例
立即寻址	操作数由指令给出	MOV DX,100H;(DX)←100
寄存器寻址	操作数在寄存器中	ADD AX,BX;(AX)←(AX)+(BX)

直接寻址	操作数的有效地址由指令直接给出	MOV AX,[100]
		MOV AX, VAR
		;(AX)←(100)或(VAR)

寄存器间接寻址	$PA=\begin{cases}(BX)\\(SI)\\(DI)\end{cases}\times 16+(DS)$ $PA=(SS)\times 16+(BP)$	MOV AX,[BX] ;(AX)←((DS)×16+(BX))
寄存器相对寻址	$PA=\begin{cases}(BX)\\(SI)\\(DI)\end{cases}+\text{位移量}+(DS)\times 16$ $PA=(SS)\times 16+(BP)+\text{位移量}$	MOV AL,MESS[SI] ;(AL)←$\begin{pmatrix}(DS)\times 16+(SI)\\+\text{OFFSET MESS}\end{pmatrix}$

基址变址寻址	$PA=(DS)\times 16+(BX)+\begin{cases}(SI)\\(DI)\end{cases}$
	或$-(SS)\times 16+(BP)+\begin{cases}(SI)\\(DI)\end{cases}$
		MOV AX,[BX][DI]
		MOV AX,[BX+DI]
		;(AX)←((DS)×16+(BX)+(DI))

相对基址变址寻址	$PA=(DS)\times 16+(BX)+\begin{cases}(SI)\\(DI)\end{cases}+\text{位移量}$ 或$=(SS)\times 16+(BP)+\begin{cases}(SI)\\(DI)\end{cases}+\text{位移量}$	MOV AX,BUFF[BX][DI] ;(AX)←$\begin{pmatrix}(DS)\times 16+(BX)+(DI)\\+\text{OFFSET BUFF}\end{pmatrix}$

2. 编写指令时，应注意的几个问题

(1) 注意区别立即寻址方式和直接寻址方式。

如：MOV AX,126;将数据 126 送入 AX 寄存器

MOV AX,[126];将数据段中的 126 单元的内容送 AX.

(2) 使用寄存器间接寻址时应注意和寄存器寻址方式的区别。

如：MOV AX,BX ;BX 中的内容传送到 AX

MOV AX,[BX];BX 所指示的地址中的内容送 AX

(3) 在双操作数指令中，源操作数和目的操作数的地址不能同时为存储器地址。

如：M1 和 M2 为两个存储器变量，

则 ADD M1,M2 是错误指令。

(4) 段跨越前缀可修改操作数所在的段。

如：MOV DL,MESS1[SI]；源操作数地址为：

；(DS)×16+(SI)+OFFSET MESS1

MOV DL，ES：MESS2[SI]；源操作数地址为：

；(ES)×16+(SI)+OFFSET MESS2

应注意：段跨越前缀不能使用 CS。

(5) 代码段寄存器 CS 不能用作指令的目的寄存器。

3. 正确使用指令系统，关键要清楚每条指令的功能以及它们规定或限制使用的寄存器。下面是初学者易混淆的几个问题：

(1) 指令对地址还是对地址中的内容进行操作，这要严格加以区分。

```
如：LEA BX,MESS              ；(BX)←MESS 的偏移地址
    MOV BX,OFFSET MESS       ；(BX)←MESS 的偏移地址
    MOV BX,MESS              ；(BX)←字变量 MESS 中的内容
```

(2) 使用指令时，要清楚指令隐含的操作寄存器。

如在乘法和除法指令中，只指出源操作数地址，但要清楚目的操作数必须存放在(AX)或(AL)中(乘法)，或(AX)、(DX：AX)中(除法)。又如串指令(MOVS、STOS、LODS、CMPS、SCAS)，它们的寻址方式也是隐含的，指令规定操作是在数据段中 SI 所指示的地址和附加段中 DI 所指示的地址之间进行串处理的；在存取串时，AL 是隐含的存取寄存器。十进制调整指令(DAA、DAS、AAA、AAS、AAM、AAD)也隐含地使用了 AL 寄存器。

类似这些在指令语句中不反映出隐含操作数的指令还有换码指令 XLAT、循环指令 LOOP、LOOPE、LOOPNE 等，它们都要求预先在规定的寄存器内设置好操作数地址或计数值。

(3) 对带符号数和无符号数的操作应正确选择相应的条件转移指令。

(4) 用移位指令来倍增或倍减一个值是很方便的，但要注意对带符号数和无符号数所使用的指令应是不同的。

如：(AX)=8520H，当(AX)为无符号数时，(AX)/2 可用指令 SHR AX,1，结果是(AX)=4290H。

当(AX)为带符号数时，(AX)/2 应用指令 SAR AX,1，结果为(AX)=0C290H

(5) 标号是程序中指令的符号地址，要注意和变量(数据符号)的区别.

如定义 VAR 是一个变量，LAB 是程序中的一个标号，则 JMP LAB 指令的转移地址为 LAB，而 JMP VAR 是一条非法指令。

例 题 分 析

例 3.1 程序在数据段中定义的数组如下：

```
NAMES   DB      'TOM..'
        DB      20
        DB      'ROSE.'
        DB      30
        DB      'KATE.'
```

```
                    DB      25
```

请指出下列指令是否正确？为什么？

(1) MOV BX,OFFSET NAMES
    MOV AL,[BX+5]

(2) MOV AX,NAMES

(3) MOV AX,WORD PTR NAMES+1

(4) MOV BX,6
    MOV SI,5
    MOV AX,NAMES [BX][SI]

(5) MOV BX,6 * 2
    MOV SI,5
    MOV AX, OFFSET NAMES [BX][SI]
    INC [AX]

(6) MOV BX, 6
    MOV SI, 5
    LEA DI,NAMES [BX][SI]
    MOV AL,[DI]

解：(1) 两条指令都是合法指令。第一条指令取得 NAMES 的 偏移地址，第二条 MOV 指令使用间接寻址方式，将地址为 (DS)×10H+(BX)+5 字节中的数据传送给 AL，结果 (AL)=20。

(2) 这条指令不正确，因为 NAMES 的属性为字节，而目的寄存器是 AX，所以类型不匹配。

(3) 为合法指令。指令中将已定义的字节变量用伪操作 PTR 改变为字类型，所以避免了类型不匹配的错误。操作结果(AX)=4D4FH (即 M 和 O 的 ASCII 码)。

(4) 前两条指令使用的是立即数方式，第三条指令的源操作数字段使用的是相对基址变址方式，但形成的数据段地址中的数据属性为字节，而源操作数据寄存器为 AX，故出现"类型不匹配"的错误。如 AX 改为 AL，这条指令就是合法指令。

(5) 前两条指令是正确的，后两条指令有错误。在汇编过程中，OFFSET 操作将得到变量的偏移值，但对相对基址变址寻址方式形成的值在汇编指令时还是未知的。同样 MOV BX, OFFSET NAMES[SI]也是错误指令。第四条指令中，AX 不能作为基址寄存器用。

(6) 均为合法指令。第三条指令中的 DI 取得一个字节地址：(BX)+(SI)+OFFSET NAMES 然后再按 DI 中的偏移地址，在数据段中将一字节内容传送给 AL 寄存器。操作结果 (AL)=30。

例 3.2 两个数据段 DSEG1 和 DSEG2 的定义如下：

```
;-------------------------------------
DSEG1   SEGMENT  'DATA'     ;define data segment 1
L1      EQU      100         ;Length of STR1
STR1    DB       L1 DUP(?)
ASTR2   DB       5 DUP(?)    ;For storing STR2
DSEG1   ENDS
```

```
;-----------------------------------------------
DSEG2    SEGMENT  'DATA'     ;define data segment 2
L2       EQU      5          ;Length of STR2
STR2     DB       L2 DUP(?)
COUNT    EQU      L1-L2+1
DSEG2    ENDS
;-----------------------------------------------
```

图 3.01 （例 3.2）

请编写一段程序，在串 STR1 中查找子串 STR2(子串 STR2 的长度小于 STR1)，如果 STR1 中包含有子串 STR2，则程序结束；如果 STR1 中不包含子串 STR2，则将 STR2 加在 STR1 之后。

```
;-----------------------------------------------
CSEG     SEGMENT
MAIN     PROC     FAR
         ASSUME CS : CSEG, DS : DSEG2, ES : DSEG1
START:
         PUSH     DS                ;for return to DOS
         SUB      AX,AX
         PUSH     AX
         MOV      AX,DSEG2          ;put dseg2 in DS
         MOV      DS,AX
         MOV      AX,DSEG1          ;put dseg2 in ES
         MOV      ES,AX

         CLD                        ;set direction flag to forward
         MOV      CX,COUNT          ;put count in CX
         MOV      BX,OFFSET STR1
NEXT:    MOV      DI,BX             ;dest addr in DI
         MOV      SI,OFFSET STR2    ;source string addr
         PUSH     CX
         MOV      CX,L2
         REP      CMPSB             ;compare STR2 in STR1
         POP      CX
         JE       MATCH             ;STR1 contain STR2
         INC      BX                ;no match,point next addr
         LOOP     NEXT              ;compare again
NO_MATCH:
         LEA      DI,ASTR2          ;STR1 not contain STR2
         LEA      SI,STR2
         MOV      CX,L2
         REP      MOVSB             ;move STR2 at after STR1
MATCH:   RET                        ;return to DOS
MAIN     ENDP
```

```
CSEG        ENDS
;-------------------------------------------------
END         START
```

图 3.02 （例 3.2）

如果 STR1 和 STR2 定义在同一个数据段，而且把 ES 和 DS 都初始化为同一数据段如下，则使用串指令将更方便

```
MOV   AX, DSEG
MOV   DS, AX
MOV   ES, AX
```

习　题

寻址方式

3.1　假定(DS)＝2000H，(ES)＝2100H，(SS)＝1500H，(SI)＝00A0H，(BX)＝0100H，(BP)＝0010H，数据变量 VAL 的偏移地址为 0050H，请指出下列指令的源操作数字段是什么寻址方式？它的物理地址是多少？

(1) MOV AX,0ABH　　(2) MOV AX,BX　　(3) MOV AX, [100H]

(4) MOV AX,VAL　　(5) MOV AX,[BX]　　(6) MOV AX, ES：[BX]

(7) MOV AX,[BP]　　(8) MOV AX,[SI]　　(9) MOV AX,[BX＋10]

(10) MOV AX,VAL[BX]　　(11) MOV AX,[BX][SI]　　(12) MOV AX,VAL[BX][SI]

3.2　假定(DS)＝212AH，(CS)＝0200H，(IP)＝2BC0H，(BX)＝1200H，位移量 D＝5119H，(224A0)＝0600H，(275B9)＝098AH，试确定 JMP 指令的转移地址。

(1) 段内直接寻址。

(2) 使用 BX 及寄存器寻址方式的段内间接寻址。

(3) 使用 BX 及寄存器相对寻址方式的段内间接寻址。

3.3　设有关寄存器及存储单元的内容如下：

(DS)＝2000H，(BX)＝0100H，(SI)＝0002H，(20100)＝12H，(20101)＝34H，(20102)＝56H，(20103)＝78H，(21200)＝2AH，(21201)＝4CH，(21202)＝0B7H，(21203)＝65H，试说明下列各条指令执行完后 AX 寄存器的内容。

(1) MOV AX,1200H　　(2) MOV AX,BX　　(3) MOV AX, [1200H]

(4) MOV AX,[BX]　　(5) MOV AX,1100[BX]　　(6) MOV AX, [BX][SI]

(7) MOV AX,1100[BX][SI]

3.4　在 ARRAY 数组中依次存储有 7 个字数据：23,36,2,100,32000,54,0，紧接着是 ZERO 字变量单元：

```
ARRAY DW 23,36,2,100,32000,54,0
ZERO  DW  ?
```

(1) 如果 BX 包含数组 ARRAY 的初始地址，请编写指令，将数据 0 传送给 ZERO 单元。

(2) 如果 BX 包含数据 0 在数组 ARRAY 中的位移量，请编写指令将数据 0 传送给 ZERO 单元。

3.5 下面有四条等值语句：

```
C1   EQU   1000
C2   EQU   1
C3   EQU   20000
C4   EQU   25000
```

下列指令哪些是不对的？请说明原因。

(1) ADD AL,C1－C2　　(2) MOV AX,C3＋C4　　(3) SUB BX,C4－C3

(4) SUB AH,C4－C3－C1　(5) ADD AL,C2

3.6 下列 ASCII 码字符串(包括空格符)依次存储在首地址为 CSTRING 的字节单元中：

```
CSTRING DB 'BASED ADDRESSING'
```

请编写指令将字符串中的第 1 个和第 7 个字符传送给 DX 寄存器。

3.7 编写指令将附加段中的一个字节变量 COUNT 送给 AL 寄存器。

3.8 编写指令将 BX 寄存器初始化为变量 MYDAT 的偏移地址。

3.9 下列程序段完成什么工作？

```
DATX1   DB   300 DUP (?)
DATX2   DB   100 DUP (?)
   ⋮

        MOV CX,100
        MOV BX,200
        MOV SI, 0
        MOV DI, 0
NEXT:   MOV AL, DATX1 [BX][SI]
        MOV DATX2 [DI],AL
        INC SI
        INC DI
        LOOP NEXT
```

3.10 执行下列指令后，AX 寄存器中的内容是什么？

```
TABLE    DW   10,20,30,40,50
ENTRY    DW   3
   ⋮
MOV      BX, OFFSET TABLE
ADD      BX, ENTRY
MOV      AX,[BX]
```

3.11 指出下列指令的错误：

(1) MOV AH, BX　　(2) MOV [BX],[SI]

(3) MOV AX,[SI][DI]　　(4) MOV MYDAT [BX][SI],ES：AX

(5) MOV BYTE PTR [BX],1000　　(6) MOV BX, OFFSET MYDAT [SI]

(7) MOV CS, AX　　(8) MOV DS, BP

3.12 按下列要求编写指令，将 BLOCK 数组中的第六个字数据存放在 DX 寄存器中。

(1) 寄存器间接寻址。

(2) 寄存器相对寻址。

(3) 基址变址寻址。

3.13 根据以下要求写出相应的汇编语言指令。

(1) 把 BX 寄存器和 DX 寄存器的内容相加，结果存入 DX 寄存器中。

(2) 用寄存器 BX 和 SI 的基址变址寻址方式，把存储器中的一个字节与 AL 寄存器的内容相加，并保存在 AL 寄存器中。

(3) 用寄存器 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)相加，并把结果送回存储器单元中。

(4) 用位移量 0524 的直接寻址方式把存储器中的一个字与数 2A59 相加，并把结果送回该存储单元中。

(5) 把数 0B5H 与(AL)相加，结果送回 AL 中。

指令练习

3.14

```
DATA SEGMENT
    TABLE_ADDR DW 1234H
DATA ENDS
    ⋮
    MOV AX, TABLE_ADDR
    LEA AX, TABLE_ADDR
```

请写出上述两条指令执行完后，AX 寄存器中的内容。

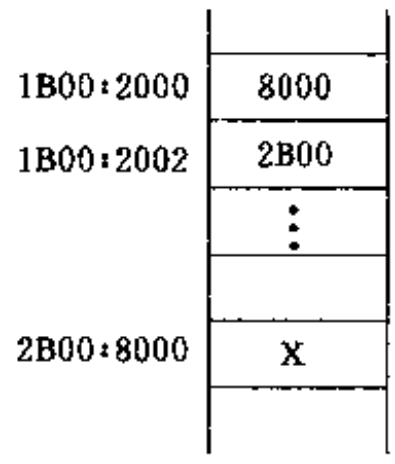
3.15 完成下列操作，选用什么指令？

(1) 把4629H 传送给 AX 寄存器。

(2) 从 AX 寄存器中减去036A。

(3) 把数组 MYDAT 的段地址和偏移地址保存在 DS 和 BX 中。

3.16 设(DS)＝1B00H，(ES)＝2B00H，有关存储器地址及其内容如下图所示。请用两条指令把字变量 X 装入 AX 寄存器。



3.17 写出完成下述功能的程序段：

(1) 传送25H 到 AL 寄存器

(2) 将 AL 的内容乘以2

(3) 传送15H 到 BL 寄存器。

(4) AL 的内容乘以 BL 的内容

问最后结果 (AX)＝？

3.18 写出完成下述功能的程序段：

(1) 从地址 DS:00中传送一个数据58H 到 AL 寄存器。

(2) 把 AL 的内容右移二位

(3) AL 的内容与字节单元 DS:01中的内容相乘

(4) 将乘积存入字单元 DS:02中

3.19 变量 DATAX 和变量 DATAY 的定义如下：

```
DATAX   DW   0148H
        DW   2316H
DATAY   DW   0237H
        DW   4052H
```

按下述要求写出指令序列：

(1) DATAX 和 DATAY 中的两个字数据相加，和存放在 DATAY 和 DATAY+2中。

(2) DATAX 和 DATAX 两个双字数据相加，和存放在 DATAY 开始的字单元中。

(3) 解释下列指令的作用：

```
STC
MOV BX, DATAX
ADC BX, DATAY
```

(4) DATAX 和 DATAY 两个字数据相乘（用 MUL）。

(5) DATAX 和 DATAY 两个双字数据相乘（用 MUL）。

(6) DATAX 除以23（用 DIV）。

(7) DATAX 双字除以字 DATAY（用 DIV）。

3.20 完成下面的双字乘法程序：

```
CODE    SEGMENT
        ASSUME CS:CODE, DS:CODE
        ORG 100H
BEGIN: JMP SHORT MAIN
;-----------------------------
MULTCND  DW 3206H       ;被乘数低位字
         DW 2521H       ;被乘数高位字
MULTPLR  DW 6400H       ;乘数
PRODUCT  DW 0           ;乘积
         DW 0
         DW 0
;-----------------------------
;Doubleword * word
MAIN PROC  ┐
      ⋮    ├双字乘法程序
MAIN ENDP  ┘
```

3.21 求双字长数 DX:AX 的相反数。

3.22 下面哪些指令是错误的?(假设 OP1,OP2是已经用 DB 定义的变量)。

(1) CMP 15, BX (2) CMP OP1, 25 (3) CMP OP1, OP2 (4) CMP AX, OP1

3.23 若(AL)=96H,(BL)=12H,指令 MUL BL 和 IMUL BL 分别执行后，它们的结果为何值? OF、CF 为何值?

3.24 写出执行以下计算的指令序列：

(1) Z←W+(Z−X)　　(2) Z←W−(X+6)−(R+9)

(3) Z←(W * X)/(R+6)　　(4) Z←((W−X)/5 * Y) * 2

3.25 已知在N到N+i的存储区中有一组ASCII码字符串(共i+1个),试编写一个汇编语言程序,将此字符串传送到NI到NI+i单元中,并使字府串的顺序与原来的顺序相反。

3.26 试编写一程序求出双字长数的绝对值。双字长数在A和A+2单元中,结果存放在B和B+2单元中。

3.27 假定(BX)=11100011B,变量VALUE的值为01111001B,确定下列各条指令单独执行后的结果。

(1) XOR BX, VALUE (2) AND BX, VALUE (3) OR BX, VALUE

(4) XOR BX, 1111,1111B (5) AND BX, 0 (6) TEST BX, 0000,0001B

3.28 编写指令序列:测试DL寄存器的低4位是否为0。

3.29 如果要检查BX寄存器中的第13位是否为1,应该用什么指令?

3.30 (1) 用一条逻辑指令清除AX寄存器。

(2) 用一条逻辑指令使DX寄存器的高3位为1,其余位不变。

(3) 写一条逻辑指令使BL寄存器的低4位为0,其它位不变。

(4) 用一条逻辑指令将AX中与BX中的对应位不相同的位均置为1。

3.31 假定(DX)=10111001b,(CL)=03,(CF)=1,试确定下列各条指令单独执行后,DX中的值。

(1) SHR DX,1 (2) SAR DX, CL (3) SHL DX, CL

(4) SHL DL,1 (5) ROR DX, CL (6) ROL DL,CL

(7) SAL DH,1 (8) RCL DX, CL (9) RCR DL, 1

3.32 利用移位、传送和加指令完成(AX)与10的乘法运算。

3.33 把DX:AX中的双字右移4位。

3.34 下列程序段执行后,BX寄存器中的内容是什么?

```
MOV CL, 3
MOV BX, 0B7H
ROL BX, 1
ROR BX, CL
```

3.35 用两条循环指令将DX:AX中的双字长数乘以2。

3.36 试分析下面的程序段完成什么功能?

```
MOV CL, 04
SHL DX, CL
MOV BL, AH
SHL AX, CL
SHR BL, CL
OR  DL, BL
```

3.37 假定一个48位数存放在DX:AX:BX中,请编写一段程序,把这个48位数乘以2。

3.38 试写出执行下列指令后,BX寄存器的内容。

```
MOV CL, 7
MOV BX, 8D16H
SHR BX, CL
```

3.39 如果把AX,BL和DH中的内容分别乘以8,连用下面的指令序列能完成此工作

吗？为什么？

```
MOV CL，3
SHL AX，CL
SHL BL，CL
SHL DH，CL
```

3.40 要求测试字节变量 STATUS，如果第1位、或第3位、或第5位为1，则转移到 ROUTINE_1；如果第1位和第3位同时为1，则转移到 ROUTINE_2；如果第1位和第3位同时为0，则转移到 ROUTINE_3，此外的其它情况都继续执行 ROUTINE_4，试画出流程图，并编制相应的程序段。

3.41 用其它指令完成和下列指令一样的功能：

(1) REP MOVSB (2) REP LODSB (3) REP STOSB (4) REP SCASB

3.42 假设数据项定义如下：

```
CONAME  DB 'SPACE EXPLORERS INC.'
PRLINE  DB 20 DUP (' ')
```

用串指令编写程序段分别完成以下功能：

(1) 从左到右把 CONAME 中字符串传送到 PRLINE。

(2) 从右到左把 CONAME 中字符串传送到 PRLINE。

(3) 把 CONAME 中的第3个和第4个字节装入 AX。

(4) 把 AX 中的内容存入 PRLINE+5开始的字节单元中。

(5) 比较 CONAME 和 PRLINE 字符串是否相同。

(6) 扫描 CONAME 字符串中有无空格符(space)，如有，把第一个空格符的地址传送给 BX 寄存器。

3.43 编写程序段将 STRING1中的20个字符移到 STRING2中，假设 DS 和 ES 都初始化为同一数据段。

3.44 编写程序段，将字符串 STRING 中的'&'字符用空格符代替。

字符串 STRING 为：'The date is FEB&03'

3.45 有一段程序如下：

```
MOV CX,10
LEA SI, FIRST
LEA DI, SECOND
REP MOVSB
```

(1) 这个程序段完成什么动作？

(2) REP 和 MOVSB 哪条先执行？

(3) MOVB 第一次执行时，要完成什么动作？

(4) REP 指令第一次执行时，要完成什么工作？

3.46 编写一段程序，比较两个5字节的字符串 OLDS 和 NEWS，如果 OLDS 字符串不同于 NEWS 字符串，则执行 NEW_LESS，否则顺序执行程序。

3.47 编写一段程序，用空格符将字符区 CHAR_FIFLD 中的字符全部清除。字符数存在 COUNT 单元中。

3.48 编写一程序段：查找 TELEPHONE 中的电话号码(10位)有无'-'符，如有则转向

FOUND_DASH，若无，则执行 NO_DASH。

3.49 假定程序中数据定义如下：

```
STUDENT_NAME   DB   30DUP (?)
STUDENT_ADDR   DB   9   DUP (?)
PRINT_LINE     DB   50 DUP (?)
```

分别编写完成下述功能的程序段：

(1) 用空格符清除 PRINT_LINE 字符域

(2) 在 STUDENT_ADDR 中查找第一个'-'

(3) 在 STUDENT_ADDR 中查找最后一个'-'

(4) 如果 STUDENT_NAME 域中全是空格符时，填入'*'

(5) 把 STUDENT_NAME 移到 PRINT_LINE 中的前30个字节中，把 STUDENT_ADDR 移到 PRINT_LINE 中的后9个字节中。

3.50 NEAR JMP、LOOP 以及条件转移指令可以跳转的最大字节数是多少?

3.51 求下面两条短转移指令的转移地址 A10和 A20分别是多少?(用16进制)

(1) 0110 EB F7 JMP A10　　(2) 0110 EB 09 JMP A20

3.52 假定 AX 和 BX 中的内容为带符号数，CX 和 DX 中的内容为无符号数，请用比较指令和条件转移指令实现以下判断。

(1) 若 DX 的值超过 CX 的值，则转去执行 EXCEED

(2) 若 BX 的值大于 AX 的值，则转去执行 EXCEED

(3) CX 中的值为零吗?若是则转去执行 ZERO

(4) BX 的值与 AX 的值比较，会产生溢出吗?若溢出转 OVERFLOW

(5) 若 BX 的值小于等于 AX 的值，则转 EQ_SMA

(6) 若 DX 的值低于等于 CX 的值，则转 EQ_SMA

3.53 AX 和 BX 中的两个带符号数相加，如果没有溢出转去执行 OK，请写出完成这个工作的指令。

3.54 假如在程序的括号中分别填入指令：

(1) LOOP L20　　(2) LOOPNE L20　　(3) LOOPE L20

试说明在三种情况下，当程序执行完后，AX、BX、CX、DX 四个寄存器的内容分别是什么?

```
TITLE     EXLOOP. COM
CODESG    SEGMENT
          ASSUME CS:CODESG, DS:CODESG, SS:CODESG
          ORG 100H
BEGIN:    MOV AX, 01
          MOV BX, 02
          MOV DX, 03
          MOV CX, 04
L20:      INC AX
          ADD BX, AX
          SHR DX, 1
          (          )
          RET
```

```
CODESG    ENDS
          END BEGIN
```

3.55　比较 AX,BX,CX 中带符号数的大小,将最大的数放在 AX 中。试编写此程序段。

3.56　请编写比较两个变量 NUM1和 NUM2的程序段,如果它们相等则转向 EQUAL,如 NUM1小于 NUM2,则转向 LESS。(假定两数均为带符号数)

3.57　已知存储器中有一个首地址为 ARRAY 的100个字的数组,现要求数组中的每个数加1(不考虑溢出),试编写完成此功能的程序段。

3.58　试编写一段程序把从 LIST 到 LIST＋100中的内容传送到 BLK 到 BLK＋100中去。

3.59　试分析下列程序段:

```
ADD AX, BX
JNO L1
JNC L2
SUB AX, BX
JNC L3
JNO L4
JMP SHORT L5
```

如果 AX 和 BX 的内容给定如下:

AX	BX	AX	BX
(1) 14C6	80DC	(2) B568	54B7
(3) 42C8	608D	(4) D023	9FD0
(5) 9FD0	D023		

问该程序分别在上面五种情况下执行后,程序转向哪里?

3.60　指令序列为:

```
CMP  AX, BX
J××  L1
```

请在能引起转移到 L1单元的条件转移指令下面划钩,AX,BX 的内容给定如下:

	AX	BX	JB	JNB	JBE	JNBE	JL	JNL	JLE	JNLE
(1)	1F52	1F52								
(2)	88C9	88C9								
(3)	FF82	007E								
(4)	58BA	020E								
(5)	FFC5	FF8B								
(6)	09A0	1E97								
(7)	8AEA	FC29								
(8)	D367	32A6								

3.61　试编制一个程序段完成(图3.03)流程图所规定的功能。

3.62　假设 X 和 X＋2单元的内容为双精度数 P,Y 和 Y＋2单元的内容为双精度数 Q (X 和 Y 为低位字),试说明下列程序段做什么工作?

```
MOV DX,X+2
MOV AX,X
ADD AX,X
ADC DX, X+2
CMP DX,Y+2
JL   L2
JG   L1
CMP AX,Y
JBE L2
L1:MOV AX,1
   INT 20H
L2:MOV AX, 2
   INT 20H
```

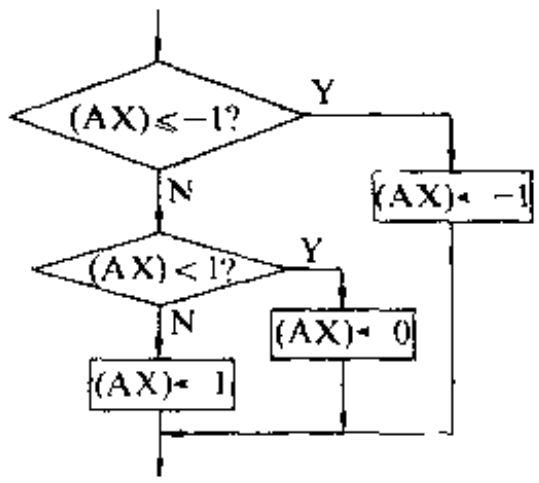


图 3.03 （3.61流程图）

3.63 编写程序将 ELEMS 中的100个字节数据的位置颠倒过来(即第一个字节和第100个字节的内容交换,第二个字节和第99个字节的内容交换…)。

3.64 变量 N1和 N2均为2字节的非压缩 BCD 数码,请写出计算 N1与 N2之差的指令序列。

3.65 有两个3位的 ASCII 码数串 ASC1和 ASC2定义如下：

```
ASC1  DB  '578'
ASC2  DB  '694'
ASC3  DB  '0000'
```

请编写计算 ASC3←ASC1＋ASC2的程序

3.66 请编写 ALPHA 中的4位压缩的 BCD 数码与 BETA 中4位压缩的 BCD 数码相加的程序。

3.67 编写4字节 ASCII 码数串'3785'与1字节 ASCII 数码'5'相乘的程序。

3.68 编写 ASCII 码数串'3785'与'5'相除的程序。

第四章　汇编语言程序格式

复　习　提　要

1. 伪操作也称为汇编程序命令，它是给汇编程序提供操作命令信息的，因此它和机器指令的区别在于机器指令是在程序运行期间执行的，而伪操作是汇编程序对源程序进行汇编时由汇编程序执行的。

2. 伪操作的用法类似于使用助记符的机器指令。和机器指令一样，每条伪操作（伪指令）的含义是唯一的。

3. 常用的伪操作：

(1) 数据定义及存储器分配伪操作：

DB、DW、DD、DQ、DT、DUP

属性伪操作：PTR、LABEL

(2) 符号定义伪操作：EQU、=

(3) 段定义和段结束伪操作：

SEGMENT、ENDS、ASSUME

定位类型 PARA、BYTE、WORD、PAGE

组合类型 PUBLIC、COMMON、AT、STACK、MEMORY

类别 'class_name'

(4) 过程定义和过程结束 PROC、ENDP

过程属性 NEAR、FAR

(5) 程序结束伪操作 END [start]

(6) 对准伪操作 EVEN、ORG

(7) 基数控制伪操作　.RADIX

二进制数标记　B　　十进制数标记　D

八进制数标记　O、Q　　16 进制数标记　H

汇编程序默认无标记数为十进制数，DEBUG 程序默认无标记数为 16 进制数。

4. 机器指令、伪指令和宏指令中的操作数项可用表达式表示。表达式由常数、寄存器、标号、变量及各种操作符组成。表达式在由汇编程序处理时，应能得出一个常数值填入机器代码。在汇编期间不能求得确定值的表达式是错误的。

5. 变量中的表达式的属性应和变量的属性相同。在指令中使用的表达式，其类型应和其它操作数项匹配。

6. 表达式中常用的操作符：

(1) 算术操作符 +、-、*、/、MOD

(2) 逻辑操作符 AND、OR、XOR、NOT

(3) 关系操作符 EQ、NE、LT、GT、LE、GE

(4) 数值回送操作符

TYPE、LENGTH、SIZE、OFFSET、SEG

(5) 属性操作符 PTR、SHORT、THIS、HIGH、LOW

7. 编写一个要转换成 EXE 文件的程序，一般要定义一个数据段和一个代码段，根据需要还可定义堆栈段和附加段。通常指令放在代码段，变量放在数据段。

下面是一个 EXE 文件的程序框架。

```
TITLE     PROGAM     NAME (EXE)
; < EQU STATEMENT GOES HERE >
DSEG      SEGMENT    PUBLIC 'DATA'
; < DATA GOES HERE >
DSEG      ENDS
;-----------------------------------------
ESEG      SEGMENT
; < DATA GOES HERE >
ESEG      ENDS
;-----------------------------------------
CSEG      SEGMENT    PUBLIC 'CODE'
          ASSUME CS:CSEG,DS:DSEG,ES:ESEG
MAIN      PROC       FAR
          PUSH       DS              ;set up for return
          XOR        AX,AX
          PUSH       AX
          MOV        AX,DSEG         ;initialize DS
          MOV        DS,AX
          MOV        AX,ESEG         ;initialize ES
          MOV        ES,AX
; <PROGRAM GOES HERE >
          RET                        ;return to DOS
MAIN      ENDP
;-----------------------------------------
SUBR      PROC       NEAR
; <SUBROUTINES GOES HERE >
          RET                        ;return to caller
SUBR      ENDP
;-----------------------------------------
CSEG      ENDS
          END        MAIN            ;end assembly
```

图 4.01 (提要7)

8. COM 文件只有一个段并限制在64KB 之内，COM 程序的堆栈由 DOS 自动产生，数据定义在代码段内。COM 程序不必初始化段寄存器，操作系统将所有段寄存器设置为 PSP(程序段前缀)的地址。COM 程序总是从偏移地址100H 开始执行。EXE 文件可由 EXE2BIN 程序转换成 COM 文件。

下面是 COM 程序的框架。

```
TITLE     PROGRAM  NAME  (COM)
;-------------------------------
CSEG      SEGMENT  PARA  'CODE'
          ASSUME   CS:CSEG, SS:CSEG, ES:CSEG
          ORG      100H
BEGIN:    JMP      MAIN
;-------------------------------
; < DATA GOES HERE >
;-------------------------------
MAIN      PROC     NEAR
; < INSTRUCTIONS GOES HERE >
MAIN      ENDP
;-------------------------------
CSEG      ENDS
          END      BEGIN
```

图 4.02 （提要8）

例 题 分 析

例4.1 用伪操作定义一个字符串变量 EMESS，其内容为“ERROR_TRY AGAIN!”。要求该字符串能在显示器上从新的一行显示。

解：定义字符串可用数据定义伪操作 DB，如字符串要在新的一行上显示，可在字符串之前加上回车换行符。另外在字符串之后还可加上字符串结束符‘$’

```
EMESS DB 0DH,0AH,'ERROR_TRY AGAIN!$'
```

例4.2 写出数据段中每个符号所对应的值。

```
DATAREA   SEGMENT
  MAX     EQU      0FFFH
  VALONE  EQU      MAX MOD 10H
  VALTWO  EQU      VALONE * 2
  BUFSIZ  EQU      ((VALTWO GT 10H)AND 10H)+10H
  BUFFER  DB       BUFSIZ DUP(?)
  BUFEND  EQU      BUFFER+BUFSIZ-1
DATAREA   ENDS
```

图 4.03 （例 4.2）

解： MAX＝0FFFH

VALONE＝000FH（VALONE 为 MAX 的值除以10H 的余数）

VALTWO＝001EH(000FH * 2＝001EH)

BUFSIZ＝0020H

（若 VALTWO 的值＞10H，则结果为真，表示为0FFFH，再和10H 相与，结果为10H，加上10H，最后取得的值为20H。若 VALTWO 的值≤10H，则结果为0，和10H 相与为0，再加上10H，因此最后结果为10H）。

BUFEND=001FH（数据区 BUFFER 最后一个字节的地址值）。

例4.3 用段伪操作定义一个数据段 DATA_SEG，要求段界起始于字边界，连接时，该段将与同名逻辑段连接成一个物理段，其类别为“DATA”。

解：
```
DATA_SEG SEGMENT    ;定义数据段 DATA_SEG
    WORD            ;段界为字
    PUBLIC          ;该数据段为组合类型
    'DATA'          ;类别为“DATA”
```
所以按要求该段的定义语句应写为：
```
DATA_SEG  SEGMENT  WORD  PUBLIC  'DATA'
```

习　题

伪操作

4.1 假设 VAR1和 VAR2为字变量，LAB 为程序中的一个标号，试找出下列指令的错误之处：

(1) ADD VAR1, VAR2　　(2) SUB AL,VAR1　　(3) JNZ VAR1

(4) JMP LAB[SI]　　(5) JMP NEAR LAB

4.2 画图说明下列语句分配的存储空间及初始化的数据值。

(1) BYTE_VAR DB 'BYTE',12,12H,2 DUP(0,?,3 DUP(1,2),?)

(2) WORD_VAR DW 4 DUP(0,1,2),?,-5,'BY','TE',256H

4.3 试列出几种方法使汇编程序把5150H 存入一个存储器字中(例如 DW 5051H)。

4.4 请设置一个数据段 DATASG，其中定义下述字符变量或数据变量：

(1) FLD1B 为字符串变量：'Personal Computer'

(2) FLD2B 为十进制数字节变量：32

(3) FLD3B 为16进制数字节变量：20

(4) FLD4B 为二进制数字节变量：01011001

(5) FLD5B 为 ASCII 码字符变量：32654

(6) FLD6B 为10个零的字节变量。

(7) FLD7B 为零件名(ASCII 码)及其数量(十进制数)的表格：
```
PART1  20
PART2  50
PART3  14
```
(8) FLD1W 为16进制数字变量：FFF0

(9) FLD2W 为二进制数字变量：01011001

(10) FLD3W 为零件表的地址变量

(11) FLD4W 为5个十进制数的字变量：15,16,17,18,19

(12) FLD5W 为5个零的字变量

(13) FLD6W 为本数据段中字数据变量和字节数据变量之间的地址差。

4.5 下面两个语句有何区别？
```
X1 EQU 1000H
```

X2 = 1000H

4.6 假设程序中的数据定义如下：

```
PARTNO DW?
PNAME  DB 16 DUP(?)
COUNT  DD?
PLENTH EQU $ -PARTNO
```

问 PLENTH 的值为多少?它表示什么意义?

4.7 有符号定义语句如下：

```
BUFF  DB 1,2,3,'123'
EBUFF DB 0
L EQU EBUFF-BUFF
```

问 L 的值为多少?

4.8 对于下面的符号定义,指出下列指令的错误。

```
A1 DB  ?
A2 DB 10
K1 EQU 1024
(1) MOV K1,AX          (2) MOV A1, AX
(3) MOV BX,A1
    MOV [BX],1000;将1000送入 A1单元
(4) CMP A1, A2         (5) K1 EQU 2048
```

4.9 对于下面的数据定义,三条 MOV 指令分别汇编成什么?(可用立即数方式表示)

```
TABLEA DW 10 DUP(?)
TABLEB DB 10 DUP(?)
TABLEC DB '1234'
    ⋮
MOV AX,LENGTH TABLEA
MOV BL,LENGTH TABLEB
MOV CL,LENGTH TABLEC
```

4.10 对于下面的数据定义,各条 MOV 指令单独执行后,有关寄存器的内容是什么?

```
FLDB    DB ?
TABLEA  DW 20 DUP(?)
TABLEB  DB 'ABCD'
(1) MOV AX,TYPE FLDB          (2) MOV AX,TYPE TABLEA
(3) MOV CX, LENGTH TABLEA     (4) MOV DX,SIZE TABLEA
(5) MOV CX,LENGTH TABLEB
```

表达式

4.11 假设数据段 DSEG 中的符号及数据定义如下,试写出此数据段汇编后各行语句的初始地址及其内容。

```
DSEG SEGMENT
  JOE=100
```

```
SAM=JOE+20
S_F     DB '/XYZ/',0DH,0AH
b_F     DB 101B,19,'a'
.RADIX 16
BLK     DB 11 DUP(' ')
EVEN
W_F1    DW '12',13D,11010B,333,SAM
.RADIX 10
W_F2    DW 15
LEN     EQU $-S_F
DSEG ENDS
```

4.12 假设程序中的数据定义如下：

```
LNAME      DB 30 DUP(?)
ADDRESS    DB 30 DUP(?)
CITY       DB 15 DUP(?)
CODE_LIST  DB 1,7,8,3,2
```

(1) 用一条 MOV 指令将 LNAME 的偏移地址放入 BX。

(2) 用一条指令将 CODE_LIST 的头两个字节内容放入 SI。

(3) 写一条伪指令使 CODE_LENTH 的值等于 CODE_LIST 域的实际长度。

4.13 指令 AND AX,OPR1 AND OPR2中,OPR1和 OPR2是两个已赋值的变量,问两个 AND 操作有什么区别?

4.14 给出的等值语句如下：

```
ALPHA  EQU 100
BETA   EQU 25
GAMMA  EQU 2
```

下列表达式的值为多少?

(1) ALPHA * 100+BETA　　(2) ALPHA MOD GAMMA+BETA

(3) (ALPHA+2) * BETA-2　　(4) (BETA/3) MOD 5

(5) (ALPHA+3) * (BETA MOD GAMMA)　　(6) ALPHA GE GAMMA

(7) BETA AND 7　　(8) GAMMA OR 3

程序框架

4.15 下面的文件名哪些是无效的?请说明原因。

(1) NEW ITEM　　(2) CUSTOMER_NAM

(3) 2ND_LINE　　(4) LINE2. ASM

4.16 阅读下列程序段并分析它将实现什么功能?

```
MYDATA    SEGMENT
  GRAY    DB    18H,34H,05H,06H,09H
          DB    0AH,0CH,11H,12H,14H
MYDATA    ENDS
MYCODE    SEGMENT
          ASSUME    CS:MYCODE,DS:MYDATA
```

```
GO:       MOV     AX,MYDATA
          MOV     DS,AX
          MOV     BX,OFFSET GRAY
          MOV     CX,2
CYCLE:    IN      AL,1
          XLAT    GRAY
          OUT     1,AL
          LOOP    CYCLE
          RET
MYCODE    ENDS
          END     GO
```

图 4.04 (4.16)

4.17 指出下列伪指令表达方式的错误，并改正之。

(1) STACK_SEG SEGMENT 'STACK'

(2) DATA-SEG SEG

(3) SEGMENT 'CODE'

(4) MYDATA SEGMENT 'DATA'

⋮

ENDS

(5) MAIN-PROC PROC FAR

⋮

END MAIN-PROC

MAIN-PROC ENDP

4.18 按下面的要求写出程序的框架。

(1) 数据段的位置从0E000H 开始；数据段中定义一个有100字节的数组，其类型属性既是字又是字节。

(2) 堆栈段从小段开始，段组名为 STACK。

(3) 代码段中指定段寄存器；主程序指定从1000H 开始；给有关段寄存器赋值。

(4) 程序结束。

4.19 编写一个完整的程序，要求把数据段 DSEG 中的双精度数 AUGEND 和附加段 ESEG 中的双精度数 ADDEND 相加，结果存放在 DSEG 中的 SUM 中，代码段为 CSEG。

4.20 按下列要求编写一个完整的程序：

(1) 数据段 DATASEG 中定义字变量 NUM，其值为5。字数组变量 DATALIST 中的头5个字单元中存放－1，0，2，5，4，其余5个字单元备用。

(2) 代码段中的程序将 DATALIST 中头5个数中的最大值和最小值，5个数的和以及乘积分别存入 DATALIST 的后5个字单元中。

4.21 定义一个数据区，它包含有23H，24H，25H 和26H 四个字符数据，把这个数据区复制20次，并显示出复制结果。

第五章　程序设计方法

复　习　提　要

1. 程序设计的一般步骤

(1) 分析所要解决的问题，确定适当的算法。

(2) 设计整个程序的逻辑结构，画出程序框图。

(3) 编写程序，正确运用 IBM PC 提供的指令、伪操作以及 DOS、BIOS 功能调用。同时写出简洁明了的说明和注释。

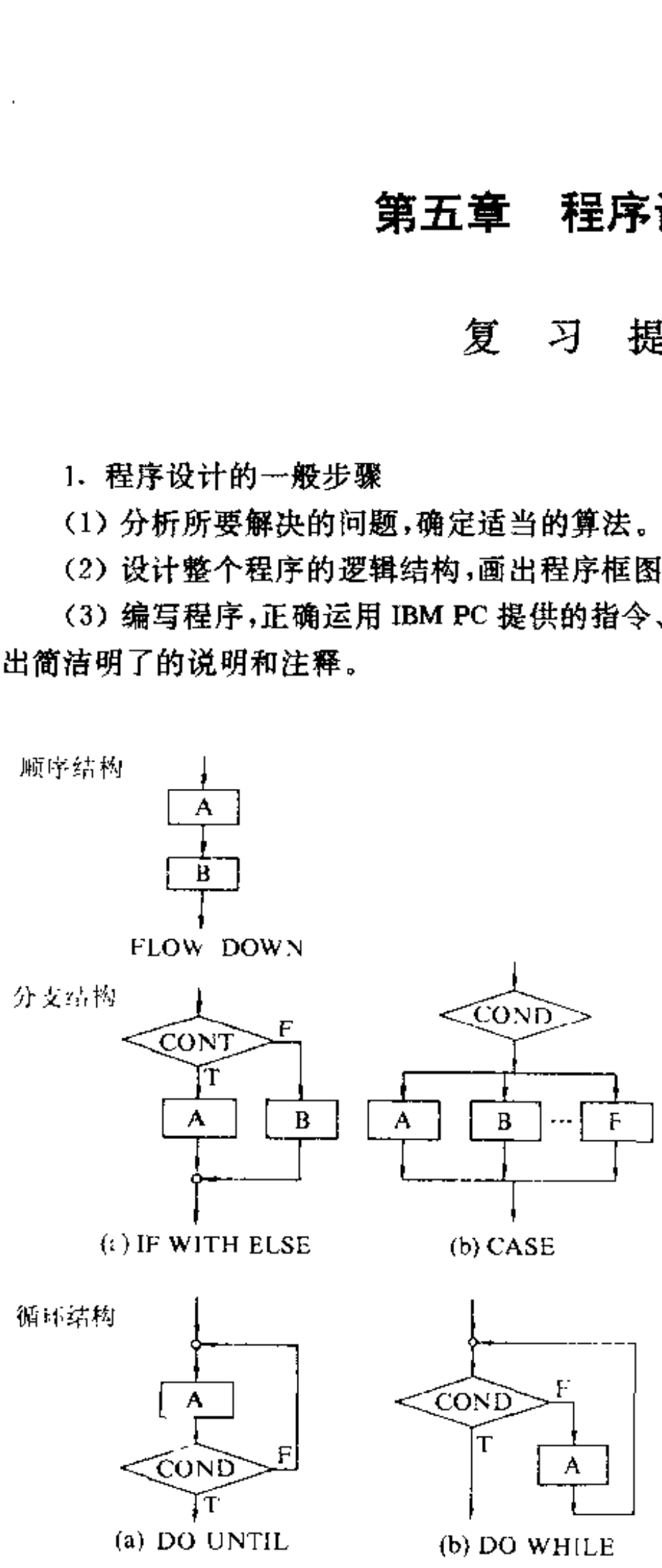


图 5.01　(提要 3)

(4) 上机调试程序。

2. 一个高质量程序应具有以下特点：

(1) 程序有较好的逻辑结构，便于进行二次开发。

(2) 源程序有较好的可读性，使非专业人员能读懂会用，甚至能加以修改。

(3) 程序应有很好的可靠性和可维护性，也就是说要保证程序能正确地工作，并且易于做进一步的改进和完善。

(4) 程序运行效率高而且有可重入性，这就要求尽量使用效率高的指令，尽量减少程序的额外开销，同时程序的运行不能破坏程序的原始数据和指令。

对初学者来说，首先要求程序中的错误尽可能地少，经上机调试后，程序能正确地工作。

3. 程序设计的几种结构

例　题　分　析

例 5.1　变量 X、Y 为二位数字的 ASCII 码串，请编写程序计算并显示出下式的值：

$$Z \leftarrow X + (Y - 15)$$

解：求一个代数式(或表达式)的值，一般的做法是根据运算符的优先级顺序进行计算，如上式先计算(Y－15)，再与 X 相加，将其结果存放在 Z 变量中，最后可用 DOS 功能调用和将结果显示出来。

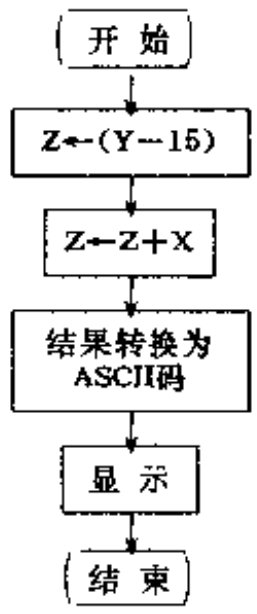


图 5.02 (例5.1)

程序框图如图 5.02：

```
TITLE    ASCDAT (COM)     add & sub ASCII numbers
CODESG   SEGMENT
         ASSUME CS:CODESG, DS:CODESG
         ORG      100H
BEGIN:   JMP      SHORT MAIN
;-------------------------------------------
  X      DB       '60'              ;ASCII number
  Y      DB       '24'
  Z      DB       '000','$'
;-------------------------------------------
MAIN     PROC     NEAR
         MOV      DX,CODESG         ;initialize DS
         MOV      DS,DX
         MOV      AH,0              ;clear AH
         MOV      AL,Y+1            ;load Y's low_order
         SUB      AL,05             ;sub low_order digit
         AAS                        ;adjust for sub
         MOV      Z+2,AL            ;put result in Z's low_order
         MOV      AL,Y              ;load Y's high_order
         SBB      AL,01             ;sub digit with carry
         AAS                        ;adjust for sub
         XCHG     AL,Z+2            ;exchange lower_order in AL
         ADD      AL,X+1            ;add lower_order in AL
         AAA                        ;adjust for add
         XCHG     AL,Z+2            ;exchange high_order in AL
         ADC      AL,X              ;add high_order digit
         AAA                        ;adjust for add
         MOV      Z+1,AL            ;store high—order result
         MOV      Z,AH              ;store carry
         OR       Z,30H             ;Z string should be
         OR       Z+1,30H           ; ASCII string
         OR       Z+2,30H
         MOV      DX,OFFSET Z       ;offset of string Z
         MOV      AH,9              ;display string function
         INT      21H               ;call DOS
         MOV      AX, 4C00h         ;return
```

```
        INT     21H
MAIN ENDP
;-----------------------------------------------------
CODESG  ENDS
        END     BEGIN
```

图 5.03 （例 5.1）

例 5.2 请编写一程序，从附加段中一个未排序的字数组 UNORDLST 中，找出最大数和最小数分别存放在 AX 和 BX 寄存器中。

解：要找出数组中的最大数和最小数，可以取数组中的某一个数作为比较的基数，然后用数组中的其它数和它进行比较，如果发现一个数小于这个基数，则它就是一个新的最小数，然后再与其他数一一比较，以此得出最终的最小数。同样如果程序发现一个数大于这个基数，那么它就是一个新的最大数。依法泡制得出最大数。数组中的每个数逐一进行这种比较操作，所以可采用循环程序结构，控制循环的条件就是数组中数的个数，如果数组中有 M 个数，则循环次数为 M－1。

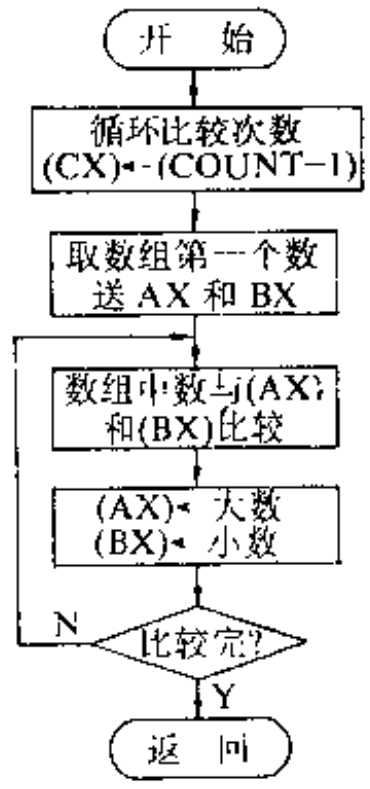


图 5.04 （例 5.2）

这个程序的框图如右：

```
TITLE MINMAX ——
;Finds the max and min words in an unordered
;   list in the extra segment
ESEG    SEGMENT
   UNORDLST     DW     50 DUP(?)
   COUNT        EQU    ($-UNORDLST)/2
ESEG        ENDS
;-----------------------------------------------------
CSEG        SEGMENT  PARA 'code'
            ASSUME    CS:CSEG,ES:ESEG
MINMAX      PROC  FAR
            PUSH      DS                    ;save DS for return
            SUB       AX,AX
            PUSH      AX
            MOV       AX,ESEG               ;init ES
            MOV       ES,AX

            MOV       CX,COUNT              ;get ready for
            DEC       CX                    ;   count-1 compares
            LEA       DI,UNORDLST           ;address of the list
            MOV       BX,ES:[DI]            ;declare it both min
            MOV       AX,BX                 ;   and max.
CHKMIN:
            ADD       DI,2                  ;point to next element
            CMP       ES:[DI],BX            ;compare element to min
```

```
                JAE     CHKMAX          ;new minimum found ?
                MOV     BX,ES:[DI]      ;yes,put it in BX
                JMP     SHORT NEXTEL
CHKMAX:
                CMP     ES:[DI],AX      ;compare element to max
                JBE     NEXTEL          ;new max found ?
                MOV     AX,ES:[DI]      ;yes,put it in AX
NEXTEL:
                LOOP    CHKMIN          ;check entire list ?
                RET                     ;exit
MINMAX          ENDP
;-------------------------------------------
CSEG            ENDS
                END     MINMAX
```

图 5.05 (例 5.2)

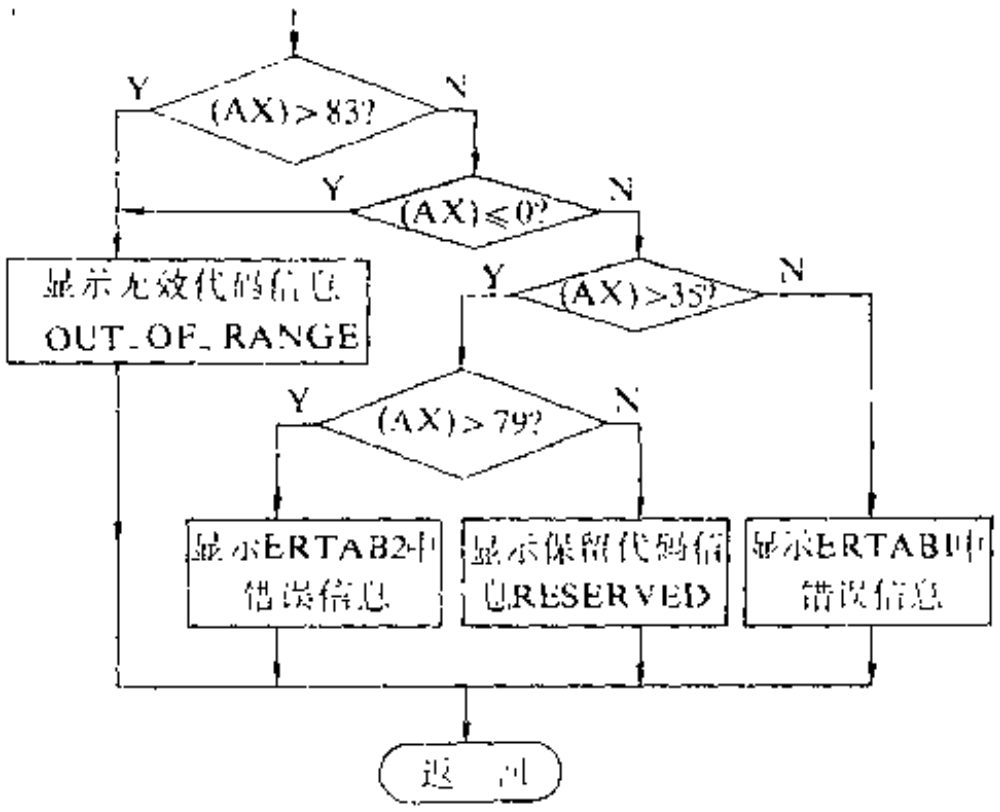


图 5.06 (例 5.3)

例 5.3 在调用 DOS 的文件管理功能时，如果出现错误(比如使用了非法功能号等)，DOS 处理程序则把进位标志(CF)置为 1，并且在 AX 中装入错误码，然后调用 SHOW-ERR 程序，将错误信息显示出来。

错误码 1～35，80～83 各表示一种 DOS 错误，36～79 作为保留的代码，其它代码(小于 1 或大于 83)没有使用，是无效的。

请编写 SHOW-ERR 程序，根据 AX 中的错误码分别显示出错误信息或保留代码信息，或无效代码信息。

分析：显然这个问题要求程序根据 AX 中的代码分类，错误码 1～35 为一类，80～82 为一类，保留代码 36～79 为一类，无效代码(<1 或>82)为一类，所以此程序的结构属于分支程序的 CASE 结构，也就是说把 AX 中的值作为判定条件，以控制程序在多个分支中选择执行其中一个分支。具体的判定方法如前面的程序框图 5.06 所示。

```
TITLE SHOW_ERR---Display DOS function call error messages
;Display a message based on an error code in AX
;All registers are preserved
        PUBLIC  SHOW_ERR
DSEG    SEGMENT PARA  PUBLIC  'DATA'
CR      EQU     13
LF      EQU     10
```

```
EOM     EQU         '$'
OUT_OF_RANGE  DB    'Error code is not in valid range (1～83)'
              DB    CR,LF,EOM
RESERVED      DB    'Error code is reserved(36～79)',CR,LF,EOM
ER1     DB    'Invalid function number',CR,LF,EOM
ER2     DB    'File not found',CR,LF,EOM
ER3     DB    'Path not found',CR,LF,EOM
ER4     DB    'Too many open files',CR,LF,EOM
ER5     DB    'Access denied',CR,LF,EOM
ER6     DB    'Invalid handle',CR,LF,EOM
ER7     DB    'Memory control blocks destroyed',CR,LF,EOM
ER8     DB    'Insufficient memory',CR,LF,EOM
ER9     DB    'Invalid memory block address',CR,LF,EOM
ER10    DB    'Invalid environment',CR,LF,EOM
ER11    DB    'Invalid format',CR,LF,EOM
ER12    DB    'Invalid access code',CR,LF,EOM
ER13    DB    'Invalid data',CR,LF,EOM
ER14    DB    'No such message',CR,LF,EOM
ER15    DB    'Invalid drive was specified',CR,LF,EOM
ER16    DB    'Attempted to remove the current directory'CR,LF,EOM
ER17    DB    'Not same device',CR,LF,EOM
ER18    DB    'No more files',CR,LF,EOM
ER19    DB    'Disk is write-protected',CR,LF,EOM
ER20    DB    'Unknown unit',CR,LF,EOM
ER21    DB    'Drive not ready',CR,LF,EOM
ER22    DB    'Unknown command',CR,LF,EOM
ER23    DB    'Data error (CRC)',CR,LF,EOM
ER24    DB    'Bad request structure length',CR,LF,EOM
ER25    DB    'Seek error',CR,LF,EOM
ER26    DB    'Unknown media type',CR,LF,EOM
ER27    DB    'Sector not found',CR,LF,EOM
ER28    DB    'Printer out of paper',CR,LF,EOM
ER29    DB    'Write fault',CR,LF,EOM
ER30    DB    'Read fault',CR,LF,EOM
ER31    DB    'General failure',CR,LF,EOM
ER32    DB    'Sharing violation',CR,LF,EOM
ER33    DB    'Lock violation',CR,LF,EOM
ER34    DB    'Invalid disk change',CR,LF,EOM
ER35    DB    'FCB unavailable',CR,LF,EOM
ER80    DB    'File exists',CR,LF,EOM
ER81    DB    'Reserved',CR,LF,EOM
ER82    DB    'Cannot make',CR,LF,EOM
ER83    DB    'Fail on INT 24',CR,LF,EOM
```

```
ERTAB1    DW    ER1,ER2,ER3,ER4,ER5,ER6,ER7,ER8,ER9
          DW    ER10,ER11,ER12,ER13,ER14,ER15,ER16
          DW    ER17,ER18,ER19,ER20,ER21,ER22,ER23
          DW    ER24,ER25,ER26,ER27,ER28,ER29,ER30
          DW    ER31,ER32,ER34,ER35
ERTAB2    DW    ER80,ER81,ER82,ER83

DSEG      ENDS
CSEG      SEGMENT  PARA  PUBLIC  'CODE'
          ASSUME  CS:CSEG,DS:DSEG
SHOW_ERR  PROC  FAR
          MOV     SI,DSEG          ;initialize DS
          MOV     DS,SI
          PUSH    AX               ;save input error number
          PUSH    BX               ;save other working regs
          PUSH    DX
          CMP     AX,83            ;check for error code in range
          JG      O_O_R
          CMP     AX,0
          JG      IN_RANGE
O_O_R:    LEA     DX, OUT_OF_RANGE
          JMP     SHORT DISP_MSG
;Error code is valid ,determine with table to use
IN_RANGE:
          CMP     AX,35            ;error code 1 —— 35 ?
          JG      TRY79
          LEA     BX,ERTAB1        ;yes, point to ERTAB1
          DEC     AX
          JMP     FORM_ADDR
TRY79:
          CMP     AX,79            ;error code 36 —— 79 ?
          JG      LAST_4
          LEA     DX, RESERVED     ;yes, display message
          JMP     DISP_MSG
LAST_4:
          LEA     BX,ERTAB2        ;error code 80 —— 83
          AND     AX,3
FORM_ADDR:
          SHL     AX,1             ;point to correct offset
          ADD     BX,AX
          MOV     DX,[BX]          ;put message addr into DX
DISP_MSG:
          MOV     AH,9             ;display messagge string
```

```
            INT     21H
            POP     DI
            POP     BX
            POP     AX
            RET                         ;return to calling program
SHOW_ERR    ENDP
CSEG        ENDS
            END
```

图 5.07 （例 5.3）

习　　题

顺序程序设计

5.1 试编写一个汇编语言程序，要求对键盘输入的小写字母用大写字母显示出来。

5.2 把三个连续存放的正整数，按递增次序重新存放在原来的三个存储单元中。

5.3 编写程序：从键盘接收一个小写字母，然后找出它的前导字符和后续字符，并按顺序输出这三个字符。

5.4 编写程序，计算 S←(a+b)－2＊(a AND b)

5.5 设有两个带符号整数变量 A 和 B，求 A 和 B 之差并判断结果是否溢出。

5.6 定义一个双字变量 VOLUME，从键盘输入长方体的长、宽和高，计算它的体积，并保存在 VOLUME 中(注意判断溢出)。

5.7 将 AX 寄存器中的16位数分成四组，每组四位，然后把这四组数分别放在 AL、BL、CL、和 DL 中。

5.8 试编制一程序，把变量 X 和 Y 中较大者存入 BIG，若 X＝Y，则把其中之一存入 BIG。

分支程序及跳跃表程序设计

5.9 试编一程序，比较两个字符串 STRING1和 STRING2所含字符是否完全相同，若相同则显示 'MATCH'，若不同则显示 'NO MATCH'。

5.10 编写一程序，从键盘上输入一个字符，如果该字符为 Y，则表示将两个整数变量 DATAX 和 DATAY 中的数据交换，否则输出"NO SWAP!"

5.11 试编写一个汇编语言程序，要求从键盘接收三个16进制数，并根据对三个数的比较显示出如下信息：

(1) 如果三个数都不相等则显示0；

(2) 如果三个数中有二个数相等则显示1；

(3) 如果三个数都相等，则显示2。

5.12 设在变量单元 A、B 和 C 中存放有三个数，若三个数都不为0，则求出三个数之和存入 D 中；若有一个为0，则将其它两个单元也清零，请编写此程序。

5.13 已知两个整数变量 A 和 B，试编写完成下述操作的程序：

(1) 若两个数中有一个是奇数，则将奇数存入 A 中，偶数存入 B 中。

(2) 若两个数均为奇数，则两数分别加1，并存回原变量。

(3) 若两个数均为偶数，则两变量不变。

5.14 从键盘输入一系列字符，然后按小写字母、数字字符和其它字符分类计数，最后分别显示出这三类字符的计数结果。

5.15 设变量 X 为带符号整数，试按下面的要求编制程序：

(1) 如果 X 的绝对值大于5，变量 FX 赋值为零。

(2) 如果 X 的绝对值不大于5，变量 FX 的值为1－X。

5.16 假设已编制好五个唱歌程序，它们的段地址和偏移地址存放在数据区的跳跃表 singlist 中。编制一程序，根据从键盘上输入的歌曲编号01～05，分别执行这五个唱歌程序。

5.17 编制一个能循环显示十条新闻标题的控制程序，每条新闻的地址转换表 NEWS 放在数据区中。

循环程序

5.18 编写程序，将一个包含有20个数据的数组 M 分成两个数组：正数数组 P 和负数数组 N，并分别把这两个数组中数据的个数显示出来。

5.19 试编写一个程序，要求能从键盘接收一个个位数，然后响铃 N 次(响铃的 ASCII 码为07)。

5.20 试编制一个汇编语言程序，求出首地址为 DATA 的100_{10}字数组中的最小偶数，并把它存放于 AX 中。

5.21 试编写一个汇编语言程序，要求从键盘上接收一个四位的16进制数，并在终端上显示出与它等值的二进制数。

5.22 将 AX 中存放的16位二进制数 K 看作是8个二进制的“四分之一字节”，试编写一个程序，要求数一下值为3(即11_2)的四分之一字节数，并将该数在终端上显示出来。

5.23 编写一个汇编语言程序，统计变量 X 中的值有多少位为1，并记入 ONE 变量中。

5.24 试编一程序，求级数$1^2+2^2+3^2+\cdots\cdots$的前 n 项和刚大于1000的项数 n。

5.25 设有一段英文，其字符变量名为 ENG，试编写一个程序，查对单词 SUN 在该文中出现的次数，并显示出次数：“SUN：××××”。

5.26 一个数组 DATAX，其中的数据排列规律是：头三项是0，0，1，第四项是前三项之和。试编一程序，将项值小于等于2000以前的各项填入数组 DATAX。

5.27 从键盘输入一系列字符，以字符‘$’为结束符，然后对其中的非数字字符计数，并显示出计数结果。

5.28 计算100个正整数的和，如果不超过机器的数的范围(65535)，则计算并输出其平均值，如超过则显示“overflow”。

5.29 100_{10}字数组的首地址为 MEM，试编制一个汇编语言程序，要求删除数组中所有为零的项，并将后续项向前压缩，最后将数组中的剩余部分补上零。

5.30 数据区保存有10个学生的姓名及其成绩，要求编写一程序将每个学生的成绩转换成六个等级。(A：90～100；B：80～89；C：70～79；D：66～69；E：60～65；F：60以下)

5.31 在 STRING 到 STRING＋99单元中存放着一个字符串，试编制一个程序，测试该字符串中是否存在数字，如有数字则把 DL 的第5位置1，否则将该位置0。

5.32 试编写一个程序，将首地址为 DATFIL 的存储区中的100个数据用十六进制在终端上显示出来，要求显示的格式为：

1　　数据1

2　　数据2

5.33 在首地址为 TABLE 的数组中按递增次序存放着100_{16}个16位补码数，试编写一个汇编语言程序把出现次数最多的数及其出现次数分别存放于 AX 和 CX 中。

5.34 数据段中已定义一个有 n 个字的数组 M，试编写一程序求出 M 数组中绝对值最大的数，放在数据段的 M+2n 单元中，并将该数的偏移地址存放在 M+2(n+1)单元中。

5.35 在 DATA 字数组中存放有100H 个16位补码数，试编写一程序求出它们的平均值放在 AX 寄存器中，并求出数组中有多少个数小于此平均值，将结果放在 BX 寄存器中。

5.36 试编制一个程序把 AX 中的16进制数转换为 ASCII 码，并将对应的 ASCII 码依次存放到 MEM 数组中的四个字节中。例如，当(AX)=2A49H 时，程序执行完后，MEM 中的4个字节内容成为39H，34H，41H，32H。

5.37 把0～100_{10}之间的30个数，存入以 GRAD 为首地址的30字数组中，GRADE+i 表示学号为 i+1的学生成绩。另一个数组 RANK 为30个学生的名次表，其中 RANK+i 的内容是学号为 i+1的学生的名次。试编写一程序，根据 GRAD 中的学生成绩，将排列的名次填入 RANK 数组中(提示：一个学生的名次等于成绩高于这个学生的人数加1)。

5.38 设有两个数组 A 和 B，其数据均为20个，两数组中的数据都按自小而大的顺序存放，现在要将这两个数组合并成一个数组 C，使 C 数组的数据也按自小而大的顺序存放。

5.39 已知数组 A 包含15个互不相等的整数，数组 B 包含20个互不相等的整数，试编一程序，将既在 A 数组中出现又在 B 数组中出现的整数存放于数组 C 中。

5.40 假定一个象棋模型，其行和列的编号是从0～7，试编写一段汇编语言程序，它可以从键盘上接收 ij 形式(0≤i≤7，0≤j≤7)的输入，并从 i 行和 j 列交点处开始，打印出“象”所走的所有可能位置。例如，当输入是42时，其输出应是：

```
    0  1  2  3  4  5  6  7
0                     *
1                  *
2  *           *
3     *     *
4        *
5     *     *
6  *           *
7                  *
```

第六章　子程序设计和模块连接

复　习　提　要

1. 子程序又称为过程，由过程伪操作 PROC 定义，ENDP 结束，属性可以是 NEAR 或 FAR。和调用程序在同一代码段中的子程序使用 NEAR 属性，和调用程序不在同一代码段中的子程序使用 FAR 属性。

2. 子程序的调用和返回使用 CALL 指令和 RET 指令。

3. 在标准子程序中，它所使用的工作寄存器一般要存入堆栈保存，在返回调用程序之前，再恢复它们的内容。

4. 子程序在多层嵌套中，要格外注意程序中的堆栈操作，要确保子程序在每次返回时，SP 指向的是正确的返回地址，而不是数据或其它信息。

5. 具有递归定义的函数，可设计成简短而又效率高的递归子程序。递归子程序也是子程序嵌套的一种形式，它嵌套调用的子程序就是它自身。同样在编写递归子程序时，要注意堆栈状态的变化。

6. 结构伪操作 STRUC 定义一种可包括不同类型数据的结构模式。其格式为：

```
结构名     STRUC
  字段名 1     db   ?
  字段名 2     dw   ?
    …
结构名   ENDS
```

结构预置语句可把结构中数据存入存储器，此语句的格式如下：

```
  变量名   结构名〈     〉
```

或变量名　结构名〈预赋值说明〉

在汇编语言程序中用如下格式访问结构数据变量：

```
      变量名·字段名[变址寄存器]
如：MOV   AL,TELIST·NAME   [SI]
                 |          |    |
              变量名      字段名  位移量
```

7. 结构设计中的变量传送

下面以两个数据变量 X 与 Y 相乘为例来说明几种主要的变量传送方式。

(1) 子程序和调用程序在同一程序模块中，则子程序可直接访问模块中的变量。

(2) 一个汇编模块要引用另一模块定义的变量，则定义变量的模块用 PUBLIC 属性来说明此变量为公共变量；引用变量的模块用 EXTRN 来说明该变量为外部变量。

```
DATASG      SEGMENT
  X     DW      100
  Y     DW      10
DATASG      ENDS
;-------------------
CODESG      SEGMENT
MAIN    PROC    FAR
          .
          .
          .
  call    subp
          .
          .
          .
MAIN    ENDP
;-------------------
SUBP    PROC    NEAR
mov       ax,x
mov       bx,y
mul       bx
ret
SUBP   ENDP
CODESG   ENDS
          END MAIN
```

图 6.01 （提要 7(1)）

```
EXTRN     SUBP:FAR
PUBLIC    X,Y
DATASG      SEGMENT
  X   DW      100
  Y   DW      10
DATASG      ENDS
;-------------------
CODESG      SEGMENT
MAIN        PROC    FAR
      .
      .
      .
  call far ptr subp
      .
      .
      .
MAIN   ENDP
;
CODESG   ENDS
         END   MAIN
```

```
EXTRN   X:WORD,Y:WORD
PUBLIC      SUBP
;-------------------
CODESG      SEGMENT
SUBP        PROC    FAR
  mov       ax,x
  mov       bx,y
  mul       bx
  ret
SUBP        ENDP
CODESG    ENDS
          END
```

图 6.02 （提要7(2)）

(3) 通过堆栈传送参数

```
EXTRN     SUBP:FAR
DATASG        SEGMENT
 X   DW        100
 Y   DW         10
DATASG        ENDS
;-----------------
CODESG        SEGMENT
MAIN   PROC   FAR
        .
        .
        .
push      x
push      y
call     far   ptr   subp
        .
        .
        .
MAIN    ENDP
CODESG      ENDS
              END   MAIN
```

```
PUBLIC   SUBP
CSEG     SEGMENT
SUBP     PROC      FAR
  push     bp
  mov      bp,sp
  mov      ax,[bp+8]
  mov      bx,[bp+6]
  mul      bx
  pop      bp
  ret      4
SUBP     ENDP
CSEG     ENDS
         END
```

图 6.03 (提要 7(3))

(4) 各模块分别定义一个同名的数据段,并用组合类型 COMMON 合并为一个复盖段,这样变量即成为本模块的局部变量,可直接引用。

```
  EXTRN   SUBP:FAR
;-----------------
  DATASG       SEGMENT COMMON
   X   DW        100
   Y   DW        10
  DATASG       ENDS
;-----------------
  CODESG       SEGMENT
  MAIN      PROC      FAR
         .
         .
         .
call     far   ptr   subp
         .
         .
         .
MAIN          ENDP
CODESG        ENDS
              END   MAIN
```

```
  PUBLIC   SUBP
;-----------------
  DATASG   SEGMENT   COMMON
    X      DW   100
    Y      DW   10
  DATASG   ENDS
;-----------------
  CSEG       SEGMENT
  SUBP       PROC   FAR
   mov       ax,x
   mov       bx,y
   mul       bx
   ret
  SUBP       ENDP
  CSEG       ENDS
             END
```

图 6.04 (提要 7(4))

8. 程序模块的连接

段定义的语句中如果指定了定位类型,或组合类型、或类别,连接程序将把各段按指定

地址边界和指定方式装入存储器。

定位类型：指定段起始地址的边界。

PAGE	页边界××00H
PARA	段边界×××0H
WORD	偶地址边界
BYTE	任意地址边界

组合类型：指定与同名段连接的方式

PUBLIC	同名段连接成一个物理段
COMMON	同名段重叠在一起形成一个段
STACK	各堆栈段紧接着组成一个堆栈段
MEMORY	该段装入模块的最高地址
AT	指定段地址(16位)，但不能指定代码段

类别'class'，同类别的段装配在相邻的位置

定位类型、组合类型、类别在模块连接中的作用请参见例 6.3、6.4。

例 题 分 析

例6.1 请用子程序结构编写如下程序：从键盘输入一个二位十进制的月份数(01～12)，然后显示出相应的英文缩写名。

分析：这是一个比较简单的问题，我们可以按题目要求的几项功能，分别编写成几个子程序。

INPUT	从键盘接收一个二位数，并把它转换为二进制数。
LOCATE	把输入数与英文缩写对应起来，这可在一个字符表中去查找。
DISPLAY	将找到的缩写字母在屏幕上显示出来。显示可用 DOS 提供的显示功能(INT 21H 的09功能)。

```
TITLE   MONTH   (EXE)
;Convert the numeric month to alphabetic abbreviation
DATASG      SEGMENT   PARA   'DATA'
 THREE      DB 3
 MONIN      DB 3,4 DUP(?)
 ALFMON     DB '???','$'
 MONTAB     DB 'JAN','FEB','MAR','APR','MAY','JUN'
            DB 'JUL','AUG','SEP','OCT','NOV','DEC'
DATASG      ENDS
;
CODESG      SEGMENT   PARA   'CODE'
            ASSUME    CS:CODESG, DS:DATASG, ES:DATASG
;-------------------------------------------
MAIN        PROC      FAR
            PUSH      DS              ;set up for return
            SUB       AX,AX
```

```
                PUSH    AX
                MOV     AX,DATASG       ;point to data seg.
                MOV     DS,AX
                MOV     ES,AX
                CALL    INPUT           ;input and convert
                CALL    LOCATE          ;locate month
                CALL    DISPLAY         ;display alpha month
                RET
MAIN            ENDP
;-----------------------------------------------
;Input and convert ASCII to binary;
INPUT           PROC    NEAR
                PUSH    DX
                MOV     AH,0AH          ;input from keyboard
                LEA     DX,MONIN        ;address of buffer
                INT     21H
                MOV     AH,MONIN+2      ;fetch numeric month
                MOV     AL,MONIN+3
                XOR     AX,3030H        ;clear ASCII 3's
                CMP     AH,00           ;month 01-09 ?
                JZ      RETURN          ;yes,bypass
                SUB     AH,AH           ;no,clear AH
                ADD     AL,10           ;correct for binary
RETURN;         POP     DX
                RET
INPUT           ENDP
;-----------------------------------------------
; Locate month in table
LOCATE          PROC    NEAR
                PUSH    SI              ;save the registers
                PUSH    DI
                PUSH    CX
                LEA     SI,MONTAB
                DEC     AL              ;correct for table
                MUL     THREE           ;3 chars for each month
                ADD     SI,AX

                MOV     CX,03           ;init'ze 3 chars move
                CLD
                LEA     DI,ALFMON
                REP     MOVSB           ;mov 3 chars
                POP     CX              ;restore registers
                POP     DI
                POP     SI
```

```
            RET
LOCATE      ENDP
;-------------------------------------------
;Display alpha month
DISPLAY     PROC    NEAR
            PUSH    DX
            LEA     DX,ALFMON       ;address of output buffer
            MOV     AH,09           ;display function
            INT     21H             ;DOS call
            POP     DX
            RET
DISPLAY     ENDP
;-------------------------------------------
CODESG      ENDS
            END     MAIN
```

图 6.05 (例 6.1)

例6.2 编写计算 ackermann 函数的程序。

对于 m≥0和 n≥0的函数 ack(m,n)由下式定义：

$$\begin{cases} ack(0,n)=n+1 \\ ack(m,0)=ack(m-1,1) \\ ack(m,n)=ack(m-1,ack(m,n-1)) \end{cases}$$

m 和 n 的值从键盘输入；如果 m 或 n 小于零则显示"Error in input data!"；如果 m 和 n 都大于零则转递归子程序 ack；在计算 ack(m,n)函数值时，当 n 等于零则递归以降低 m 的值；当 m 等于零，则递归结束，求得函数值 n+1。

分析：很显然，递归函数 ack(m,n)编写成递归子程序比较简单清晰。当 m>0,n>0时，根据函数定义，问题将转化为求 m 和 n−1的 ack 函数，如果 n−1仍大于0，则递归调用，使 n 再减1，一直到 n=0时，函数关系就变为 ack(m,0)=ack(m−1,1)。如果我们把含有参数 m,n 的数据结构用伪操作 STRUC 定义为一个结构的话，则在对 ack 子程序的递归调用过程中，在堆栈形成了 n 帧含有 m 和 n−1的结构数据，直到 n 变为基数0后，再逐次递减 m，一直到 m=0，这时堆栈又形成了 m 帧含有 m−1和1的结构数据。根据 ack(0,n)=n+1，即可求得中间结果 result=n+1。在程序嵌套退出的过程中，对每帧结构中的 m,n 仍按 ack 函数的三个递归公式来处理。

基于上面的算法分析，我们先设计出主程序和子程序的框图如图6.06：

```
TITLE     ACK.EXE     ackermann function
; ack (0,n) = n+1
; ack (m,0) = ack (m-1,1)
; ack (m,n) = ack (m-1,ack(m,n-1))
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
datarea   segment                ;define data segment
  mm          dw        ?
  nn          dw        ?
  result      dw        ?
```

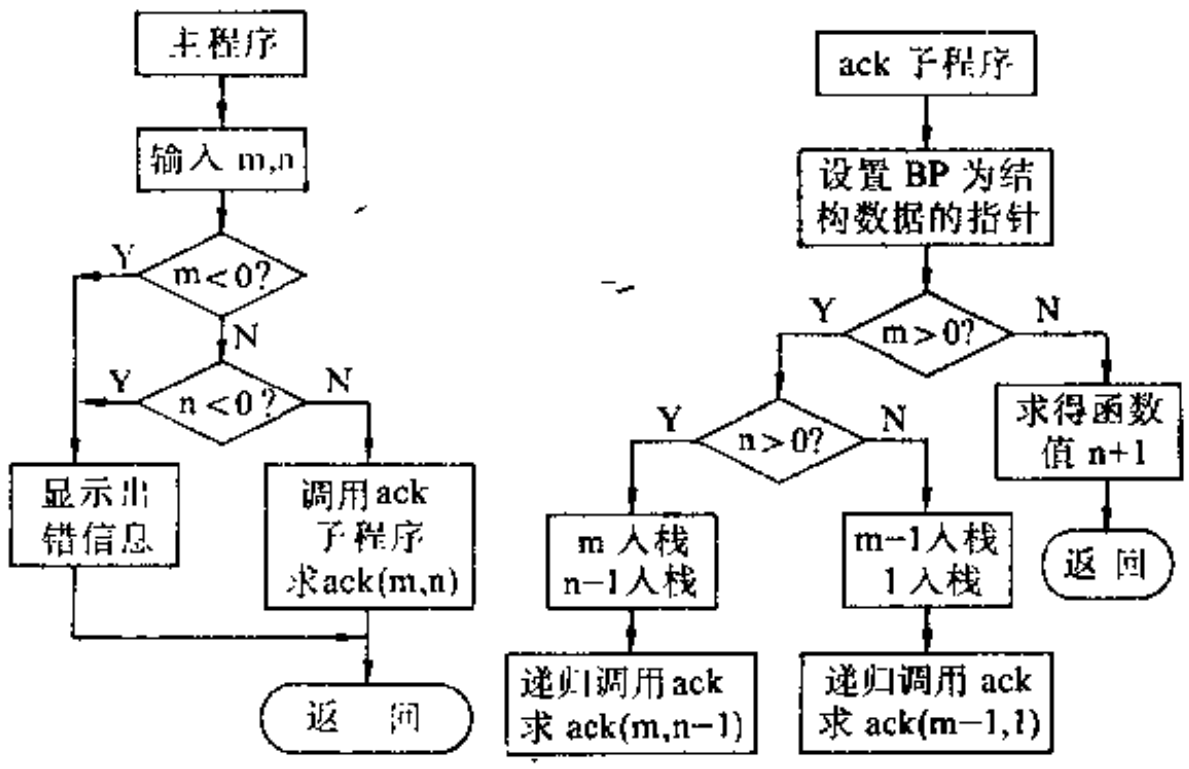


图 6.06 (例 6.2)

```
  temp        dw      ?
  mess        db      'Error in input data!',0ah,0dh,'$'
datarea ends
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
prognam  segment                ;define code segment
main     proc    far
         assume  cs:prognam,ds:datarea
;
frame   struc
  save_bp       dw          ?
  save_cs_ip    dw          2 dup(?)
  n             dw          ?
  m             dw          ?
  result_addr   dw          ?
frame   ends
;
start:                          ;starting execution address
;set  up  stack  for  return
         push    ds             ;save old data segment
         sub     ax,ax          ;put zero in AX
         push    ax             ;save it on stack
;set DS register to current data segment
         mov     ax,datarea     ;datarea segment addr
         mov     ds,ax          ; into DS register
         call    input          ;input m
         mov     ax,temp
         mov     mm,ax
         call    crlf           ;CR/LF
;
         call    input          ;input n
         mov     bx,temp
         mov     nn,bx
         call    crlf           ;CR/LF
```

```
;
        cmp     ax,0            ;m < 0 ?
        jl      error           ;yes,display error mess
        cmp     bx,0            ;n < 0 ?
        jl      error           ;yes,display error mess
        lea     si,result
        push    si
        mov     ax,mm

        push    ax
        mov     bx,nn
        push    bx
        call    far ptr ack     ;computing ack function
        jmp     exit
error:  lea     dx,mess         ;display error message
        mov     ah,09h
        int     21h
exit:
        ret                     ;return to DOS
main    endp
;-------------------------------------------
input   proc    near            ;define subprocedure
        push    ax
        push    bx
        push    cx
        push    dx
        mov     bx,0
newchar1:
        mov     ah,1            ;reguest kbd input
        int     21h
        sub     al,30h          ;ASCII to digit
        jl      exit1           ; < 0,exit
        cmp     al,9d
        jg      exit1           ; > 9,exit
        cbw
        xchg    ax,bx           ;convert to binary
        mov     cx,10
        mul     cx
        xchg    ax,bx
        add     bx,ax
        jmp     newchar1
exit1:  mov     temp,bx
        pop     dx              ;restore registers
        pop     cx
        pop     bx
        pop     ax
        ret
input   endp                    ;end subprocedure
;-------------------------------------------
crlf    proc    near
        mov     dl,0ah          ;display CR/LF
        mov     ah,02h
        int     21h
```

```
;
        mov     dl,0dh
        mov     ah,02h
        int     21h
        ret
crlf    endp
;-----------------------------------------
ack     proc    far

        push    bp
        mov     bp,sp           ;BP is pointer of struct
        mov     si,[bp]. result_addr
        mov     ax,[bp]. m
        mov     bx,[bp]. n
        cmp     ax,0            ;m>0?
        jg      testn           ;if so, jump to test n
        inc     bx              ;m=0. put n+1
        mov     [si],bx         ; to result
        jmp     exit2
;
testn:  cmp     bx,0            ;n>0?
        jg      mn              ;if so, jump to do ack(m,n)
        dec     ax              ;n=0
        mov     bx,1
        push    si
        push    ax
        push    bx
        call    ack             ;ack( m-1,1 )
A:      jmp     exit2
;
mn:     push    si
        push    ax
        dec     bx
        push    bx
        call    ack             ;ack( m,n-1 )
;
B:      push    si
        mov     ax,[bp]. m      ;push m-1
        dec     ax
        push    ax
        mov     dx,[si]         ;and ack(m-1,result)
        push    dx              ;      to stack
        call    ack             ;ack(m-1,ack(m,n-1))
;
exit2:  pop     bp
        ret     6
ack     endp
;-----------------------------------------
prognam ends                    ;end of code segment
;* * * * * * * * * * * * * * * * * * * * * * * * * * * *
        end     start           ;end assembly
```

图 6.07 （例 6.2）

从上面的递归程序中可以看出，堆栈使用得很频繁，所以特别要注意堆栈的变化，但因为程序采用了结构，所以每次递归调用形成的堆栈状态比较规律，且省去了变量地址的计算，所以能较安全地进行堆栈操作。

下面以 ack(1,2)为例，画出在递归调用 ack 子程序和达到基数后嵌套退出时的堆栈状态。堆栈变化依箭头①②…⑫的顺序入栈或出栈。最后结果(result)＝4。

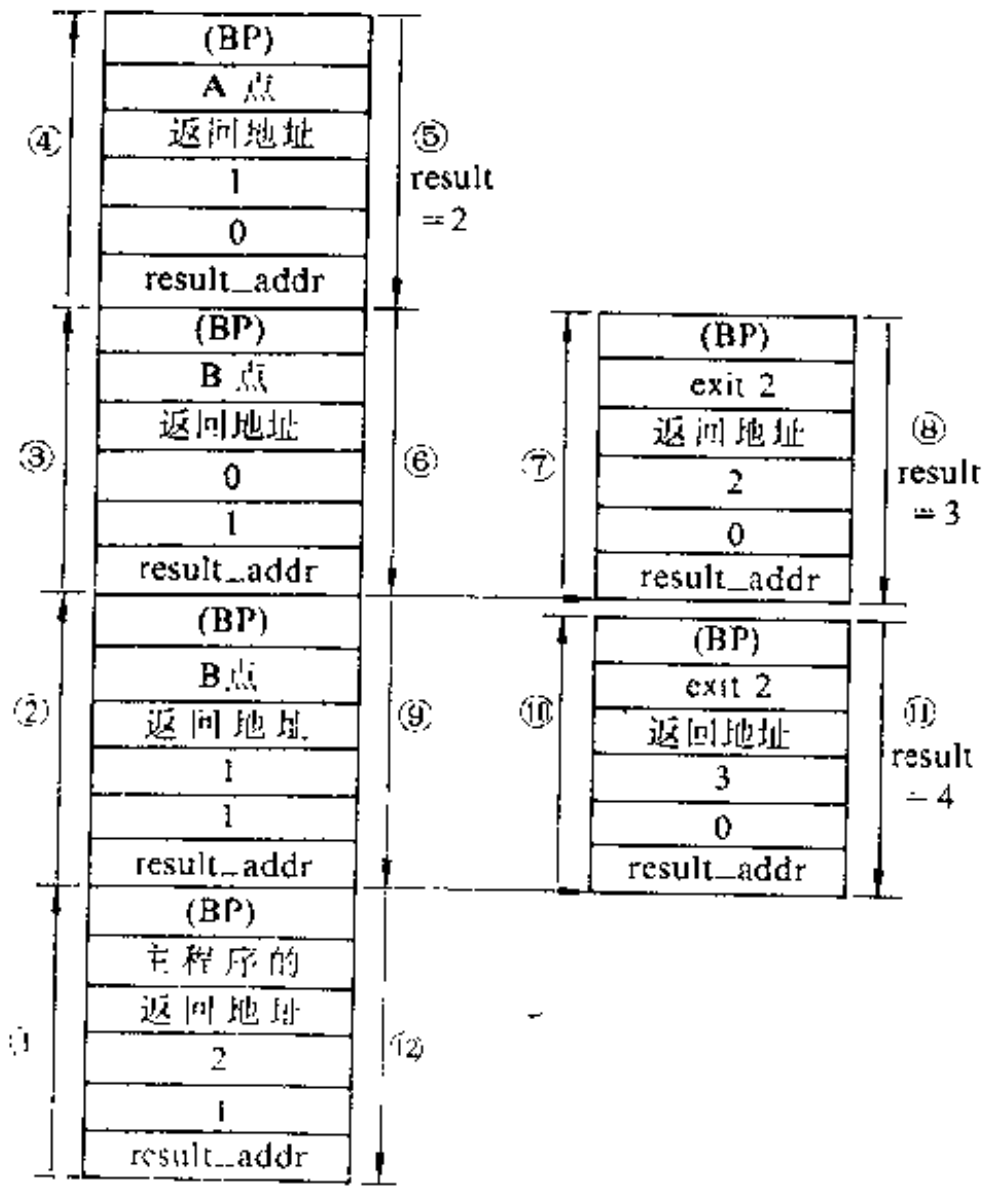


图 6.08　ack(1,2)的堆栈状态

例6.3　下面是两个汇编模块 CALLER 和 SUBP 的源程序清单。这两个程序分别汇编后，用命令 LINK CALLER＋SUBP 连接起来运行。请画出连接后各段在存储器中的位置。

```
                        TITLE CALLER (EXE) call subprogram to SUBP
                              EXTRN SUBP:FAR
                        ;--------------------------
0000                    STACKSG   SEGMENT PARA STACK 'STACK'
0000    64 [                      DW  100  DUP(?)
                ????
                        ]

00C8                    STACKSG   ENDS
                        ;--------------------------
0000                    DATASG    SEGMENT  PARA  'DATA'
0000    000A             DATAX    DW  10
```

```
0002    0005                DATAY   DW  5
0004    ????????            DATAZ   DD  ?
0008                        DATASG  ENDS
                            ;-------------------------------
0000                        CODESG  SEGMENT PARA  'CODE'
0000                        MAIN    PROC    FAR
                                    ASSUME  CS:CODESG,DS:DATASG
                                    ASSUME  SS:STACKSG

0000    1E                          PUSH    DS
0001    2B  C0                      SUB     AX,AX
0003    50                          PUSH    AX
0004    B8  ---- R                  MOV     AX,DATASG
0007    8E  D8                      MOV     DS,AX

0009    A1  0000  R                 MOV     AX,DATAX
000C    8B  1E  0002  R             MOV     BX,DATAY
0010    9A  0000  ----- E           CALL    SUBP
0015    A3  0004  R                 MOV     WORD  PTR  DATAZ,AX
0018    89  16  0006  R             MOV     WORD  PTR  DATAZ+2,DX
001C    CB                          RET
001D                        MAIN    ENDP
001D                        CODESG  ENDS
                                    END     MAIN
```

```
                            TITLE SUBP (EXE) called program
0000                        STACKSG SEGMENT PARA  STACK  'STACK'
0000    10  [                       DW  16  DUP (?)
                ????
                    ]

0020                        STACKSG ENDS
                            ;-------------------------------
0000                        CODESG  SEGMENT PARA  'CODE'
                                    PUBLIC  SUBP
0000                        SUBP    PROC    FAR
                                    ASSUME  CS:CODESG

0000    F7  E3                      MUL     BX
0002    CB                          RET

0003                        SUBP    ENDP
0003                        CODESG  ENDS
                                    END
```

图 6.09 （例 6.3）

分析：在主程序模块和子程序模块中都定义了一个名为 STACKSG 的堆栈段，而且使用了定位类型 PARA，即要求该段从小段边界开始装配，还使用了组合类型 STACK 和类别 'stack'，这就使这两个模块的堆栈段组合成一个有(100+16)×2个字节的堆栈段。

主模块中的数据段也定位在小段边界(PARA)。主模块和子模块的代码段都定义为 CODESG 且定位于小段边界(PARA)，类别 'CODE' 使得这两个代码段装入存储器的位置是紧相邻的。因此各段在存储器中的位置如图6.10：

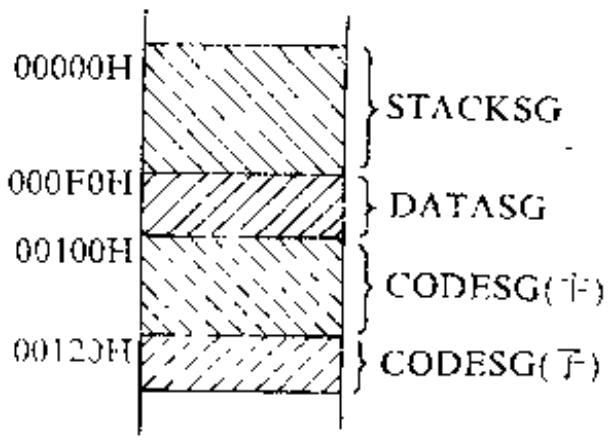


图 6.10 (例6.3)

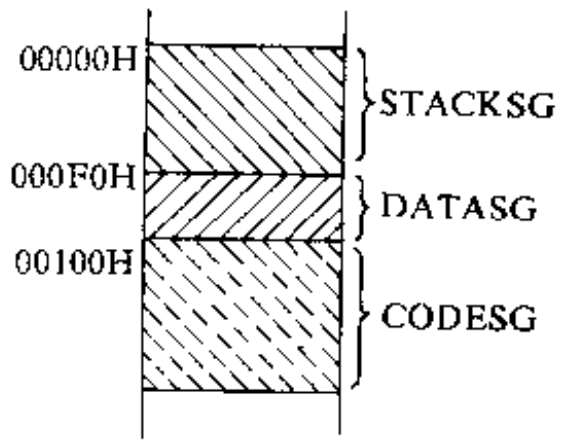


图 6.11 (例 6.4)

例6.4 如果在上题(例6.3)的 CALLER 程序模块和 SUBP 程序模块中，定义代码段时分别都加上组合类型 PUBLIC，即：

```
CODESG  SEGMENT  PARA  PUBLIC  'code'
```

模块连接后，各段的位置又如何？

分析：组合类型 PUBLIC 可以使同名段连接成一个物理段，因此连接后在装配表中只列出一个名为 CODESG 的代码段。

习　题

堆栈

6.1 下列 PUSH 指令和 POP 指令哪些是合法的？哪些是非法的？试分别用✓和×在(　)中表示出来。

(1) PUSH CX (　)
(2) PUSH BL (　)
(3) PUSH DS (　)
(4) PUSH CS (　)
(5) PUSH ES (　)
(6) PUSH SS (　)
(7) PUSH SI (　)
(8) PUSH BETA (　)
(9) PUSH BETA [BX] (　)
(10) PUSH BETA+1 (　)
(11) POP AL (　)
(12) POP CX (　)
(13) POP DX (　)
(14) POP DS (　)
(15) POP CS (　)
(16) POP SS (　)
(17) POP ES (　)
(18) POP DAT (　)
(19) POP DAT [CX] (　)
(20) POP DAT+2 (　)

6.2 下面的程序段有错吗？若有，请指出错误。

```
CRAY  PROC
```

```
        PUSH   AX
        ADD    AX, BX
        RET
ENDP    CRAY
```

6.3 写出适当的段定义和指令，将100字的数组 ARY 定义在数据段，将有100字的堆栈初始化为 DS:100，堆栈指示器指向栈底。

6.4 已知堆栈段寄存器 SS 的内容是0F0A0H，堆栈指示器 SP 的内容是00B0H，先执行两条 PUSH 指令把8057H 和0F79BH 入栈，然后执行一条 POP 指令，试画出示意图，说明堆栈及 SP 内容的变化过程。

6.5 分析下列程序，画出堆栈最满时各单元的地址及内容。

```
S_SEG     SEGMENT    AT      1000H
   DW     200  DUP (?)
   TOS    LABEL      WORD
S_SEG     ENDS
;
D_SEG     SEGMENT
   DW     32 DUP(?)
D_SEG     ENDS
;
C_SEG     SEGMENT
ASSUME    CS:C_SEG, SS:S_SEG
          MOV    AX,S_SEG
          MOV    SS,AX
          MOV    SP,OFFSET TOS
          PUSH   DS
          MOV    AX,0
          PUSH   AX
          .
          .
          .
          PUSH   AX
          PUSHF
          .
          .
          .
          POPF
          POP    AX
          RET
C_SEG     ENDS
          END
```

图 6.12 （习 6.5）

6.6 分析下列程序的功能，写出堆栈最满时各单元的地址及内容。

```
STACK     SEGMENT  STACK  'STACK'  AT  1000H
   DW     128  DUP (?)
  TOS     LABEL      WORD
STACK       ENDS
;
DSEG      SEGMENT
  DW      32 DUP (?)
DSEG      ENDS
;
CODE      SEGMENT
MAIN      PROC       FAR
ASSUME    CS:CODE,   SS:STACK
START:    MOV        AX,STACK
          MOV        SS,AX
          MOV        SP,OFFSET TOS
          PUSH       DS
          MOV        AX,0
          PUSH       AX
          MOV        AX,4321H
          CALL       HTOA
A:        RET
MAIN      ENDP
;--------------------
HTOA      PROC       NEAR
          CMP        AX,15
          JLE        B1
          PUSH       AX
          PUSH       BP
          MOV        BP,SP
          MOV        BX,[BP+2]
          AND        BX,0FH
          MOV        [BP+2],BX
          POP        BP
          MOV        CL,4
          SHR        AX,CL
          CALL       HTOA
     B:   POP        AX
     B1:  ADD        AL,30H
          JL         PRINTIT
          ADD        AL,7H
     PRINTIT:
          MOV        DL,AL
          MOV        AH,2
```

```
        INT     21H
        RET
HTOA    ENDP
CODE    ENDS
        END     START
```

图 6.13 （习题 6.6）

6.7 考虑以下的调用序列：

(1) MAIN 调用 NEAR 属性的 SUBA 子程序，返回的偏移地址为0400H。

(2) SUBA 调用 NEAR 属性的 SUBB 子程序，返回的偏移地址为0A00H。

(3) SUBB 调用 FAR 属性的 SUBC 子程序，返回的段地址为02000H，偏移地址为0B00H。

(4) 从 SUBC 返回 SUBB。

(5) SUBB 调用 NEAR 的 SUBD 子程序，返回的偏移地址为0C00H。

(6) 从 SUBD 返回 SUBB。

(7) 从 SUBB 返回 SUBA。

(8) 从 SUBA 返回 MAIN。

(9) MAIN 调用 SUBC，返回的段地址是2000H，偏移地址是0600H。

(10) 从 SUBC 返回 MAIN。

请画出每次调用或返回时的堆栈状态。

6.8 下面是一个简化的程序清单，请在题后的图中填入此程序在执行过程中堆栈的变化。

```
                        ;-------------------------
0000                    STACKSG  SEGMENT  PARA  STACK  'STACK'
0000    20  [              DW  32  DUP (?)
                ????
                    ]

0040                    STACKSG  ENDS

0000                    CODESG   SEGMENT  PARA  'CODE'
0000                    BEGIN    PROC     FAR
                        ASSUME   CS:CODESG,SS:STACKSG
0000    1E                       PUSH     DS
0001    2B  C0                   SUB      AX,AX
0003    50                       PUSH     AX
0004    E8  0008  R              CALL     B10
                        ;        ...
0007    CB                       RET
0008                    BEGIN    ENDP
                        ;-------------------------
0008                    B10      PROC
0008    E8  000C  R              CALL     C10
                        ;        ...
```

```
000B    C3                      RET
000C                    B10     ENDP
                        ;-------------------------------
000C                    C10     PROC
                        ;       ...
000C    C3                      RET
000D                    C10     ENDP
                        ;-------------------------------
000D                    CODESG  ENDS
                                END     BEGIN
```

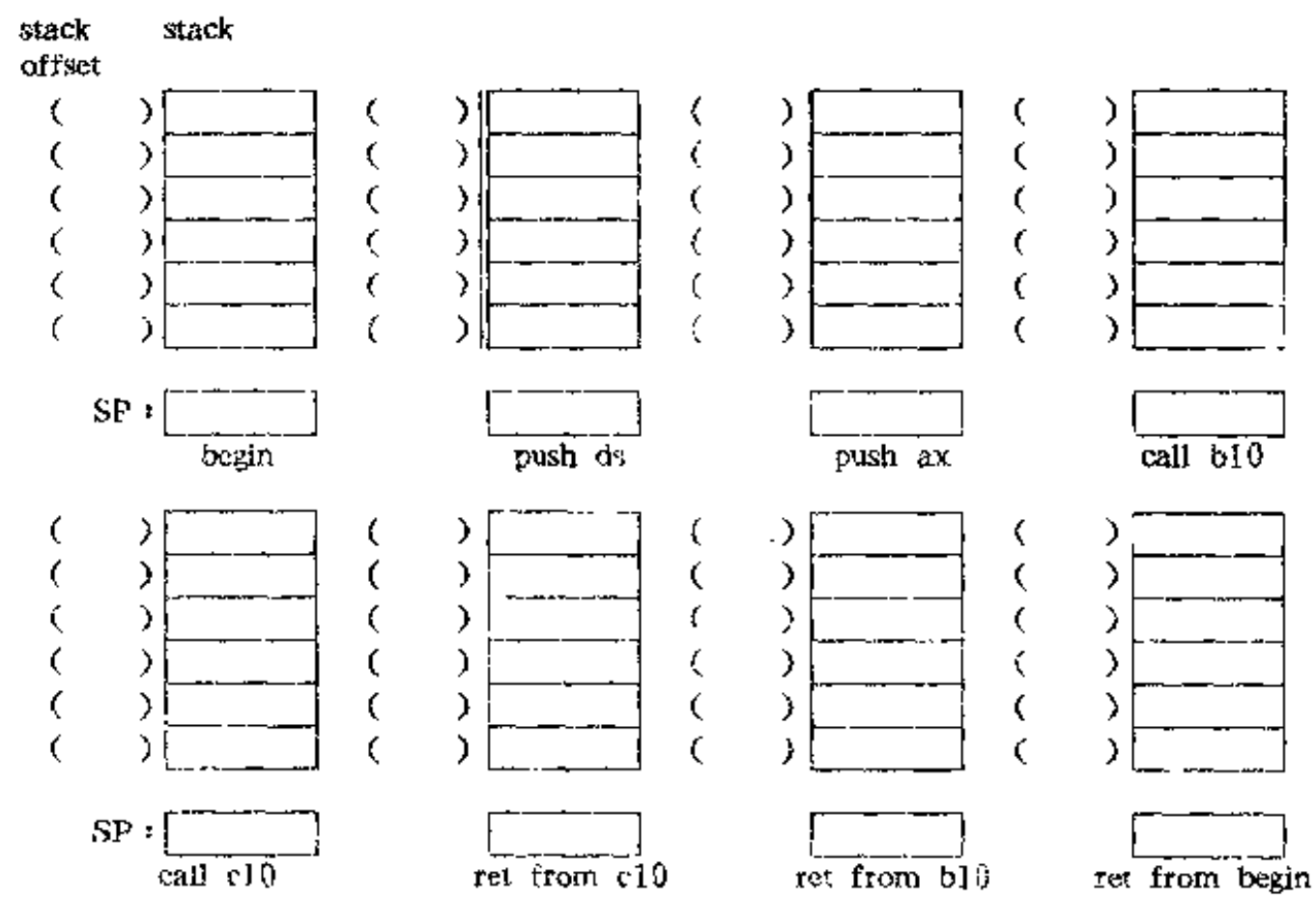


图 6.14 (习题 6.8)

子程序

6.9 编写一段程序，如果字节变量 TESTONE 和 TESTTWO 相等，则调用 ALLSAME 子程序，否则调用 NOTSAME 子程序。

6.10 写一段子程序 SKIPLINE，完成输出空行的功能，空出的行数在 AX 寄存器中。

6.11 用子程序结构编写一个完整的程序：

主程序 MAINPRO 允许用户在键盘上输入零件数量和价格；

子程序 SUBCONV 把 ASCII 码转换为二进制；

子程序 SUBCALC 计算出零件的单价；

子程序 SUBDISP 把二进制表示的单价转换为十进制数并显示出结果。

6.12 试编写一个子程序，使它能接收从键盘输入的十进制数(如溢出则指示出错)，并将它转换为二进制数存放在 DATA 单元中。

算法提示：一个任意的十进制数

$N = X_nX_{n-1}\cdots X_2X_1X_0$ 可以写成：

$N = ((\cdots((0 \times 10) + X_n) \times 10 + X_{n-1}) \times 10 + \cdots) \times 10 + X_1) \times 10 + X_0)$

6.13 试编写带符号数的双精度加法及乘法子程序，在调用子程序以前操作数已由主程

序送入堆栈，运算结果也由主程序保存在堆栈中。

6.14 编写一个有主程序和子程序结构的程序模块，子程序的参数是一个N字节数组的首地址TABLE，数N及字符CHAR，要求在N字节数组TABLE中查找字符CHAR，并记录该字符出现的次数。主程序则要求从键盘接收一串字符以建立数组TABLE，并逐个显示出从键盘输入的每个字符CHAR以及它在TABLE数组中出现的次数（为简化起见，假设出现次数≤15，可以用16进制的形式把它显示出来）。

6.15 设有10个学生的成绩分别为76，69，84，90，73，88，99，63，100，80，试编制一个子程序统计60～69分，70～79分，80～89分，90～99分及100分的人数，分别存放到S6，S7，S8，S9，及S10单元中。

子程序嵌套

6.16 编写一个子程序嵌套结构的程序模块，分别从键盘上输入姓名及8个字符的电话号码，并以一定的格式显示出来。

主程序TELIST：

- 显示提示符'Input name：'
- 调用子程序INPUTNAM输入姓名
- 显示提示符'Input a telephone number：'
- 调用子程序INPHONE输入电话号码
- 调用子程序PRTLINE，显示姓名及电话号码。

子程序INPUTNAM：

- 调用键盘输入子程序GETCHAR，将输入的姓名存放在INBUF缓存区。
- 把INBUF中的姓名移入输出行OUTNAME

子程序INPHONE：

- 调用键盘输入子程序GETCHAR，将输入的8位电话号码存放在INBUF缓存区中
- 把INBUF中的号码移入输出行OUTPHONE

子程序PRTLINE：

显示姓名及电话号码，格式为：

NAME	TEL
×××	××××

6.17 编写子程序嵌套结构的程序，把整数分别用二进制和八进制形式成对显示出来。

主程序BAND0：将整数字变量VAL1和VAL2存入堆栈，并调用子程序PAIRS。

子程序PAIRS：从堆栈中取出VAL2；调用二进制显示程序OUTBIN；输出8个空格；调用八进制显示程序OUTOCT；调用输出回车换行子程序。

从堆栈中取出VAL1，再重复上述过程显示出它的二进制数和八进制数。

递归子程序

6.18 编写一递归子程序，计算指数函数X^n的值，X和n在主程序中从键盘输入。

6.19 编写一段递归子程序计算$n!$ $(n\geqslant 0)$

$$n!=n(n-1)(n-2)\cdots 1$$

其递归定义如下：

$$\begin{cases} 0!=1 \\ n!=n(n-1)!\ (n>0) \end{cases}$$

6.20 假定一个正数 N≥1存放在变量 NUM 中，试编写一段递归子程序计算 FIB(N)，并将结果存入变量 RESULT 中。

Fibnacci 函数的定义如下：

$$\begin{cases} FIB\ (1)=1 \\ FIB\ (2)=1 \\ FIB\ (n)=FIB\ (n-2)+FIB\ (n-1)\ \ (n>2) \end{cases}$$

6.21 设 T 是一棵以 N 为根的二叉树，它的树高 H(T)可递归定义如下：

基数：如果 T 只有一个结点，则 H (T)＝0

归纳步骤：如果 T 由一个结点 N 及其连接的二叉树 LT 和 RT 组成，则

$$H\ (T) = 1 + \max(H\ (LT), H\ (RT))$$

编写一段递归子程序，显示出所给定的二叉树的高度(例如对于下图所示的树，显示的树高应是5)。

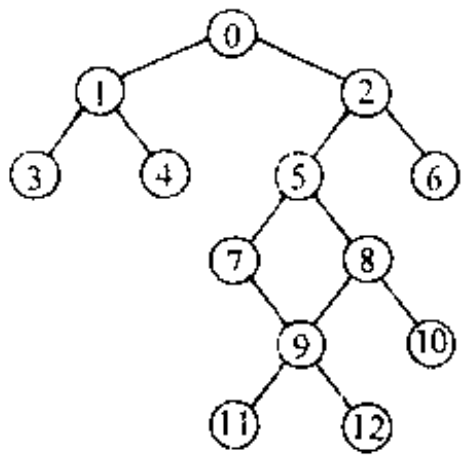


图 6.15 (习 6.21)

结构伪操作

6.22 将学生的班级、姓名、成绩定义为一个结构，再用五条结构预置语句产生五个学生的成绩登记表，然后编写程序将成绩＜60分的学生情况显示出来。

6.23 用 STRUC 伪操作定义的参数表 NAMELIST 如下：

```
NAMELIST    STRUC
MAXLEN      DB   100
ACTLEN      DB   ?
NAMEIN      DB   100   DUP(' ')
NAMELIST    ENDS
```

(1) 请用结构预置语句分配此结构的存储区。

(2) 编写一段指令从键盘输入字符(用 DOS 调用 INT 21H)存入结构中，然后将输入的字符数送入 DISPFILE 单元中。

程序模块

6.24 下面是两个独立的程序：

```
;-------------------------------
            PAGE    60,132
TITLE       CALLMUL
;
STACKSG     SEGMENT  PARA  STACK  'STACK'
            DW    64   DUP (?)
STACKSG     ENDS
;
DATASG      SEGMENT  PARA  'DATA'
QTY         DW       0140H
PRICE       DW       2500H
DATASG      ENDS
```

```
;
CODESG    SEGMENT  PARA  'CODE'
BEGIN     PROC     FAR
ASSUME    CS:CODESG,DS:DATASG,SS:STACKSG
          PUSH     DS
          SUB      AX,AX
          PUSH     AX
          MOV      AX,DATASG
          MOV      DS,AX
          CALL     SUBMUL
          RET
BEGIN     ENDP
CODESG    ENDS
          END      BEGIN
;-------------------------------
          PAGE     60,132
TITLE     SUBMUL
;
CODESG    SEGMENT  PARA  'CODE'
SUBMUL    PROC     FAR
          ASSUME   CS:CODESG
          MOV      AX,PRICE
          MOV      BX,QTY
          MUL      BX
          RET
SUBMUL    ENDP
CODESG    ENDS
          END      SUBMUL
;-------------------------------
```

图 6.16 （习题 6.24）

现在要求把这两个程序连接起来运行，要做哪些修改?请将修改后的程序清单打印出来。

6.25 有两个源模块如下：

```
;       Source    Module  1
;-----------------------
S_SEG   SEGMENT STACK
        DW      128  DUP (?)
TOP     LABEL   WORD
S_SEG   ENDS
;
D_SEG   SEGMENT COMMON
  VAR   DB      50  DUP (?)
```

```
D_SEG    ENDS
;
E_SEG    SEGMENT AT 1000H
  AREA   DW      70  DUP (?)
E_SEG    ENDS
;
C_SEG    SEGMENT PUBLIC
         .          }
         .          } 500H bytes
         .          }
C_SEG    ENDS
         END
;        Source     Model  2
;-----------------------
S_SEG    SEGMENT STACK
         DW      64  DUP (?)
S_SEG    ENDS
;
D_SEG    SEGMENT COMMON
  VECT   DW  30
D_SEG    ENDS
;
C_SEG    SEGMENT PUBLIC
         .          }
         .          } 1000H bytes
         .          }
C_SEG    ENDS
         END
```

图 6.17 （习题 6.25）

假设连接程序以 S-SEG，D-SEG，C-SEG 的次序把这两个模块连接起来，堆栈的栈顶地址为20000，试画图说明各段在存储器中的位置。

6.26 对于下面的程序，请说明其中哪些段地址和偏移地址是由汇编程序确定的，哪些是由连接程序确定的，为什么？

```
EXTRN    COST:WORD, ROUTINE:FAR
PUBLIC   BEGIN
;-------------------------------
DATA_1   SEGMENT
  TOTAL  DW   50  DUP (?)
  NUM    DW       ?
DATA_1   ENDS
```

```
;-----------------------------
DATA_2    SEGMENT
  PART    DW   100   DUP (?)
DATA_2    ENDS
;-----------------------------
CODE      SEGMENT
            .
            .
            .
          MOV          AX,DATA_1
          MOV          DS,AX
            .
            .
            .
          MOV          AX,SEG COST
          MOV          ES,AX
          MOV          AX,TOTAL
          ADD          AX,ES:COST
            .
            .
            .
          MOV          AX,DATA_2
          MOV          ES,AX
          INC          ES:PART[SI]
          MOV          CX,NUM
          CMP          TOTAL[DI],CX
          JE           NEXT
          JMP          FAR  PTR  ROUTINE
NEXT:       .
            .
            .
CODE      ENDS
```

图 6.18 （习题 6.26）

6.27 假定一个名为 MAINPRO 的程序要调用子程序 SUBPRO，试问：

(1) MAINPRO 中的什么语句告诉汇编程序 SUBPRO 是在外部定义的？

(2) SUBPRO 怎么知道 MAINPRO 要调用它？

6.28 假定程序 MAINPRO 和 SUBPRO 不在同一源模块中，MAINPRO 中定义字节变量 QTY 和字变量 VALUE 和 PRICE，SUBPRO 程序要把 VALUE 除以 QTY，并把商存在 PRICE 中，试问：

(1) MAINPRO 怎么告诉汇编程序外部子程序要调用这三个变量？

(2) SUBPRO 怎么告诉汇编程序这三个变量是在另一个汇编语言程序中定义的？

6.29 假设：

(1) 在模块1中定义了双字变量 VAR1，首地址为 VAR2的字节数组以及 NEAR 标号

LAB1，它们将由模块2和模块3所使用。

(2) 在模块2中定义了字变量 VAR3和 FAR 标号 LAB2，而模块1中要用到 VAR3，模块3中要用到 LAB2。

(3) 在模块3中定义了 FAR 标号 LAB3，而模块2中要用到它。

试对每个源模块给出必要的 EXTRN 和 PUBLIC 说明。

6.30 试编写一个执行以下计算的子程序 COMPUTE：

$$R \leftarrow X+Y-3$$

其中 X，Y 及 R 均为字变量。假设 COMPUTE 与其调用程序都在同一代码段中，数据段 D-SEG 中包含 X 和 Y 变量，数据段 E-SEG 中包含 R 变量，请写出主程序调用 COMPUTE 子程序的部分。

如果主程序和 COMPUTE 在同一程序模块中，但不在同一代码段中，程序应如何修改？如果主程序和 COMPUTE 不在同一程序模块中，程序又应如何修改？

第七章　高级汇编语言技术

复　习　提　要

1. 使用宏汇编和使用子程序一样，都能减少程序员编写程序的工作量，因而也减少了程序出错的可能性。子程序不管被调用多少次，它在程序中只须编写一次。虽然一条宏指令也只编写一次，但是每次宏调用都要在程序中展开并保留宏定义体中的每一行，所以宏调用不节省存储空间。因此宏汇编适合于代码较短，传送参数较多的子功能段使用，子程序适合代码较长，调用比较频繁的子功能段使用。

2. 定义一条宏指令相当于由用户给汇编程序提供了一个新的操作码。

宏定义的格式：

```
宏指令名　MACRO [哑元表]
          ⋮      }宏定义体
          ENDM
```

3. 哑元表中可以有多个哑元，也可以只有一个哑元，也可以无哑元。

4. 宏定义体包括实现子功能的指令和伪操作，如果宏定义体中有一个和多个标号，则必须用 LOCAL 伪操作列出所有的标号。

5. 用伪操作 MACRO 定义的宏指令可以在源程序的任何地方调用，但必须先定义后调用。

宏调用的格式：

```
宏指令名　[实元表]
```

实元表中的实元和哑元表中的哑元在位置上一一对应。若实元个数大于哑元个数，则多余的实元无效；若实元个数小于哑元个数，则多余的哑元作“空”(NUL)处理。

6. 源程序汇编时，汇编程序把每个宏调用展开，也就是把宏定义体复制到调用宏指令的位置，同时用实元取代哑元，由 LOCAL 定义的标号也由唯一的标号来替代。

7. 宏定义中可以调用先前定义的宏指令，也就是说可以实现宏嵌套，同样，宏定义中也允许递归调用。

8. IBM MASM 提供四组宏汇编伪操作：通用伪操作，重复伪操作、条件伪操作和列表伪操作。其功能总结于下表。

伪操作	格　式	功　能
(1) 通用伪操作		
MACRO	宏指令名MACRO[哑元表] ⋮ ENDM	定义相当于一段汇编语句序列的宏指令，必须由 ENDM 伪操作结束。

LOCAL	LOCAL 标号 1 [,标号 2,…]	在宏展开时,LOCAL 定义的标号由唯一的标号来替代。
PURGE	PURGE 宏指令 1 [,宏指令 2,…]	取消宏指令所定义的功能

(2) 重复伪操作

IRP	IRP 哑元,〈自变量表〉 ⋮ }重复块 ENDM	汇编程序重复展开重复块的代码,每次重复用自变量表中的一项取代哑元,重复次数由自变量个数来确定。
IRPC	IRPC 哑元,字符串 ⋮ }重复块 ENDM	重复次数由字符串中的字符数来确定,每次重复用字符串中的一个字符取代重复块的哑元。
REPT	REPT 表达式 ⋮ }重复块 ENDM	表达式的值确定重复块被展开的次数。

(3)条件伪操作

EXITM	EXITM	根据条件伪操作的结果,结束宏展开。
IF1	IF1 ⋮ ENDIF	汇编程序在第一遍扫视时,汇编其中的语句。通常用于将一个宏库调入源程序(INCLUDE)
IF	IF 表达式 ⋮ ENDIF	表达式的值不为 0,则汇编其中的语句。
IFE	IFE 表达式 ⋮ ENDIF	表达式的值为 0,则汇编其中的语句。
IFDEF	IFDEF 符号 ⋮ ENDIF	如果符号已在程序中定义(包括EXTRN 定义的外部符号),则汇编其中的语句。
IFNDEF	IFNDEF 符号 ⋮ ENDIF	如符号未定义,则汇编其中的语句。
IFB	IFB〈自变量〉 ⋮ ENDIF	如自变量为空,则汇编其中的语句。
IFNB	IFNB〈自变量〉 ⋮ ENDIF	如自变量不空,则汇编其中的语句。

IFIDN	IFIDN〈字符串 1〉,〈字符串 2〉 ⋮ ENDIF	如〈字符串 1〉与〈字符串 2〉相同,则汇编其中的语句。

(4)列表伪操作

.LALL	.LALL	在 LST 清单中列出宏调用的全部语句(包括注释)
.SALL	.SALL	在 LST 清单中不列出展开后的语句。
.XALL	.XALL	只列出宏定义体中产生目标代码的语句。亦为列表伪操作的缺省方式。

9. IMB MASM 提供四个宏汇编操作符

操作符	格式	
&	符号 1& 符号 2	宏展开时合并前后两个符号而形成一个符号
;;	;;注释	宏展开时,;;后的注释不予展开
!	! 字符	用于自变量中,使汇编程序把!后的字符当做一个数值,而不是一个符号。
%	%符号	汇编程序将%后的符号转换为数字,并在展开期间用数字替代哑元。

例 题 分 析

例 7.1 宏指令 MOVE 完成字节传送的功能。

宏定义:

```
MOVE    MACRO   TO ,FROM,COUNT
        PUSH    SI
        PUSH    DI
        PUSH    CX
        LEA     SI,FROM
        LEA     DI,TO
        MOV     CX,COUNT
        REP     MOVSB
        POP     CX
        POP     DI
        POP     SI
        ENDM
```

宏调用:

```
        MOVE    MESS2,MESS1,16
```

宏展开：

```
    +           PUSH        SI
    +           PUSH        DI
    +           PUSH        CX
    +           LEA         SI,MESS1
    +           LEA         DI,MESS2
    +           MOV         CX,16
    +           REP         MOVSB
    +           POP         CX
    +           POP         DI
    +           POP         SI
```

图 7.01 （例 7.1）

例 7.2 宏指令 MOVIF 根据所给条件 B 或 W，分别产生指令 REP MOVSB 或 REP MOVSW，如缺省给定条件，则产生 REP MOVSB。

宏定义：

```
    MOVIF   MACRO   TAG
            IFIDN   <&TAG>,<B>
            REP     MOVSB
            EXITM
            ENDIF
            IFIDN   <&TAG>,<W>
            REP     MOVSW
            ELSE
    ;;No B or W tag --- default to B
            REP     MOVSB
            ENDIF
            ENDM
```

宏调用：

```
            .LALL
            MOVIF   B
            MOVIF   W
            MOVIF
```

宏展开：

```
            .LALL
            MOVIF   B
    +       IFIDN   <B>,<B>
    +       REP     MOVSB
    +       EXITM
            MOVIF   W
    +       ENDIF
    +       IFIDN   <W>,<W>
    +       REP     MOVSW
    +       ENDIF
            MOVIF
```

```
+           ENDIF
+           ELSE
+
+           REP         MOVSB
+           ENDIF
```

图 7.02 （例 7.2）

例 7.3 宏指令 SHIFT 根据变元 R 的大小产生不同的屏蔽值。如 R≥64，则屏蔽值为 VALUE；如 32≤R<64，VALUE 右移一位；如 16≤R<32，VALUE 右移二位；如 8≤R<16，VALUE 右移三位；R<8，则 VALUE 右移四位。

宏定义：

```
SHIFT     MACRO     R,VALUE
          LOCAL     STRIP
          MOV       DH,VALUE      ;;set up mask value
          IF        R  LT  64
          SHR       DH,1          ;;if R<64, shift 1 bit
          IF        R  LT  32
          SHR       DH,1          ;;if R<32, shift 2 bits
          IF        R  LT  16
          SHR       DH,1          ;;if R<16, shift 3 bits
          IF        R  LT  8
          SHR       DH,1          ;;if R<8, shift 4 bits
STRIP:
          ENDIF
          ENDIF
          ENDIF
          ENDIF
          ENDM
```

宏调用：

```
          SHIFT     60,3FH
          SHIFT     5,3FH
```

宏展开：

```
          SHIFT     60,3FH
+         MOV       DH,3FH
+         SHR       DH,1
          SHIFT     5,3FH
+         MOV       DH,3FH
+         SHR       DH,1
+         SHR       DH,1
+         SHR       DH,1
+         SHR       DH,1
+  ?? 0001:
```

图 7.03 （例 7.3）

例 7.4 宏指令 RAND 产生一个 0 到 LIMIT 之间的随机数。

宏定义：

```
RAND    MACRO    LIMIT
;;Generate a pseudo random integer between
;; 0 and "limit"inclusive and return it in AL
IRP  REG,<CX,DX,AX>
     PUSH    REG          ;;Save affected registers
     ENDM
     MOV     AH,0         ;;Read the timer
     INT     1AH
     MOV     AX,DX        ;; Move low count into AX
     MOV     CL,LIMIT     ;; Move limit into CL
;;Strip enough high bits off the dividend (AX)
;; to ensure against divide overflow
     SHIFT   LIMIT,3FH    ;;set up AND mask in DH
     AND     AH, DH       ;;Strip off the bits
     DIV     CL           ;;divide result in AX
     MOV     AL,AH
     POP     CX
     MOV     AH,CH        ;;Restore AH
     POP     DX           ;; Restore other registers
     POP     CX
     ENDM
```

宏调用：

```
     RAND    32           ;generate 0-32 random
```

宏展开：

```
+    PUSH    CX
+    PUSH    DX
+    PUSH    AX
+    MOV     AH,0
+    INT     1AH
+    MOV     AX,DX
+    MOV     CL,32
+    MOV     DH,3FH
+    SHR     DH,1
+    AND     AH,DH
+    DIV     CL
+    MOV     AL,AH
+    POP     CX
+    MOV     AH,CH
+    POP     DX
+    POP     CX
```

图 7.04 （例 7.4）

习　题

宏定义和宏调用

7.1　定义宏指令 MOVE，使它能将 N 个字符从一个字符区传送到另一个字符区。

7.2　定义宏指令 SWAPBYTE，使它能完成交换两个字节单元内容的功能，然后展开宏调用：

```
SWAPBYTE  HIGH,LOW
```

7.3　编写一条宏指令 CLRB：完成用空格符将一字符区中的字符清除的工作。字符区首地址及其长度为变元。

7.4　某工厂计算周工资的方法是每小时的工资率 RATE 乘以工作时间 HOUR，另外每工作满 10 小时加 3 元奖金。请将周工资的计算编写成一条宏指令 WAGES，并展开宏调用：

```
WAGES  R1, 42
```

7.5　编写宏指令 CLS，完成清除全屏幕的工作（提示：调用 BIOS INT 10H 的清屏功能）。

7.6　定义宏指令 LOCATE，使其在屏幕上按指定的行(row)和列(col)设置光标位置。

7.7　为编写 EXE 程序，在程序开始要初始化段寄存器，请将此通用功能编写成一条宏指令 STARTER。

7.8　编写宏指令 DISPLAY：能显示已存放在数据区的信息 MESSAGE。

7.9　试定义宏指令，要求把存储区中的一个用‘$’字符结束的字符串，传送到另一个存储区中。

7.10 给出宏指令定义如下：

```
DIF         MACRO   X, Y
            MOV     AX,X
            SUB     AX,Y
            ENDM
ABSDIF      MACRO   V1,V2,V3
            LOCAL   CONT
            PUSH    AX
            DIF     V1,V2
            CMP     AX,0
            JGE     CONT
            NEG     AX
CONT:       MOV     V3,AX
            POP     AX
            ENDM
```

试展开以下调用，并判定调用是否有效。

(1) ABSDIF P1,P2,DISTANCE

(2) ABSDIF [BX], [SI],X [DI],CX

(3) ABSDIF [BX] [SI],X [BX] [SI],240H

(4) ABSDIF AX,AX,AX

7.11　宏指令 BIN_SUB 完成多个字节数据连减的功能：RESULT←(a-b-c-d-…)，减数的个数存放在 COUNT 单元中，最后结果存入 RESULT 单元。请编写此宏指令。

7.12　请用宏指令定义一个可显示字符串：

```
GOOD   DB 'Good students:class X   NAME'
```

其中 X 和 NAME 在宏调用时给出。

7.13　宏指令 MOV_B_W 完成串(字串或字节串)的传送功能，请写出此宏指令的定义。

宏嵌套

7.14　下面的宏指令 CNT 及 INC 完成相继字存储：

```
CNT   MACRO   A,B
      A&B     DW ?
      ENDM

INC   MACRO   A,B
      CNT     A,% B
      B=B+1
      ENDM
```

请展开宏调用：

```
C=0
INC   DATA,C
INC   DATA,C
INC   DATA,C
```

这里宏指令 INC 和机器指令 INC 有什么不同？

7.15　定义宏指令并展开宏调用：

宏指令 JOE 把一串信息"MESSAGE NO.K"存入数据存储区 XK 中。

宏调用：

```
I=0
      JOE     TEXT,I
      JOE     TEXT,I
      JOE     TEXT,I
```

7.16　对于大多数 DOS INT 21H，其功能调用都需要在 AH 寄存器中存放不同的功能码，请将这种功能调用定义成中断宏指令 DOS21，再定义宏指令 DISP 完成显示字符的功能，其中可使用已定义的宏指令 DOS21，然后展开宏调用：DISP '*'。

7.17　定义清除全屏幕宏指令 CLS；

定义设置光标位置的宏指令 LOCATE；

定义显示字符串宏指令 DISPLAY；

然后利用上面三条宏指令定义 CENTER，使其完成清屏，并从 12 行 30 列开始显示信息 PROMPT。

7.18　宏指令 STORE 定义如下：

```
STORE    MACRO    X, N
         MOV      X+I,I
I=I+1
         IF       I-N
         STORE    X,N
         ENDIF
         ENDM
```

试展开下列调用：

```
I=0
      STORE TAB,7
```

7.19　试编写非递归的宏指令，使它完成的工作与上题(7.18)的 STORE 相同。

重复汇编和条件汇编

7.20　试编写一段程序完成以下功能：如给定名为 X 的字符串长度大于 5 时，下列指令将汇编 10 次：ADD AX，AX

7.21　编写一段语句，使汇编程序建立一个字符显示区 PARTS，其中包含 10 个字符串“PNO. n”其中 n 为 1～10 的数字字符。

7.22　定义宏指令 FINSUN 比较两个数 X 和 Y，若 X＞Y，执行 SUM←X＋2＊Y，否则执行 SUM ←2＊X＋Y。

7.23　编写一段程序完成以下功能：

如变元 X＝‘VT55’，则汇编 MOV TERMINAL，0，否则汇编 MOV TERMINAL，1。

7.24　对于 DOS INT 21H，所有的功能调用都需要在 AH 寄存器中存放功能码，而只有一部份功能调用需要在 DX 中放一个值。请定义 DOS 的中断宏指令 DOS21，只有在程序中定义了缓存区时，才汇编为：

```
MOV   AH, DOSFUNC
MOV   DX, OFFSET BUFF
INT   21H
```

否则不汇编 MOV DX，OFFSET BUFF 指令。

另外请展开宏调用：

```
DOS21   01
DOS21   0AH, IPFIELD
```

7.25　编写一段程序，使汇编程序根据 SIGN 中的内容分别产生不同的指令。如果(SIGN)＝0，则用字节变量 DIVD 中的无符号数除以字节变量 SCALE；如果(SIGN)＝1，则用字节变量 DIVD 中的带符号数除以字节变量 SCALE。商都存放在字节变量 RESULT 中。

宏指令库

7.26　建立一个你自己的宏库 MYLIB. LIB，其中包括宏指令 CLS（清屏），LOCATE(光标定位)，DISPLAY（显示信息），MOVE(字符传送)和 STARTER（段寄存器初始化），并编一个小程序使用你的宏库。

第八章　I/O 程序设计

复　习　提　要

1. 程序控制 I/O 方式

这种方式使用 I/O 指令(IN 和 OUT)直接在端口级上进行信息的输入/输出。CPU 与各设备之间以串行方式工作。CPU 要通过测试 I/O 接口的状态来控制传送,若 I/O 设备没有准备好,CPU 就循环测试,直到设备准备好,CPU 就执行一次传送。程序控制 I/O 的流程如右图:

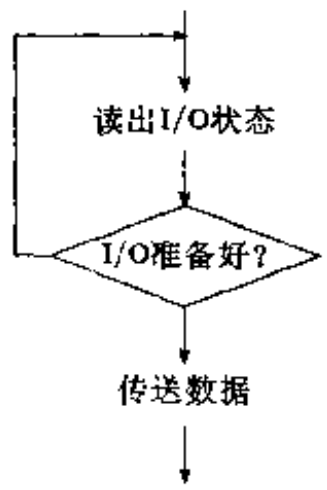


图 9 01　(提要1)

2. 中断传送方式

这种 I/O 方式实质上是一种特殊情况下的程序转移方式。所谓特殊情况一般是指:

(1) 计算机出现异常事件,如电源掉电,内存或 I/O 总线奇偶错等。出现这样的事件,CPU 应立即中断现行程序的运行,转去执行处理故障的子程序。

(2) 程序中预先安排的中断指令(INT)或其它内部原因(如除法错等),使现行程序暂时中断,转去执行相应的处理子程序。

(3) 外部设备一切准备就绪时,向 CPU 发出中断现行程序的请求,以处理外设的输入输出。

以上三类情况是引起中断产生的原因,称为中断源。第(1)类情况一般安排为非屏蔽中断。第(2)类情况称为内中断,第(3)类情况为外中断,这是一些可屏蔽的中断类型。

3. 中断程序的设计方法

对于要求以中断方式工作的外设,它们的中断类型已由硬件连线确定,另外在主程序中还要做如下的准备工作:

(1) 保存原中断向量(INT 21H 的 35H 功能),设置新的中断向量(INT 21H 的 25H 功能)。

(2) 设置设备的中断屏蔽位。

(3) 设置 CPU 的中断允许位(开中断)。

(4) 在主程序结束之前,恢复原中断向量。

主程序完成了上述一系列准备工作后,中断即以完全随机的方式产生程序转移。当 CPU 响应了中断请求,中断系统将完成以下工作:

(1) CPU 接收外设的中断类型号。

(2) 当前的 PSW、CS、IP 的内容保存入栈。

(3) 清除 IF、TF。

(4) 根据中断类型号取出的中断向量(中断处理子程序入口的段地址和偏移地址)送 CS 和 IP。

(5) 转中断处理子程序。

中断处理子程序的编写:

① 保存有关寄存器内容。

② 开中断(STI)。

③ 处理输入输出或其它功能。

④ 关中断(CLI)。

⑤ 发送中断结束命令(EOI)给中断命令寄存器。

⑥ 恢复寄存器内容。

⑦ 返回被中断的程序(IRET)。

4. BIOS 和 DOS 中断

驻留 ROM 的 BIOS 提供了主要 I/O 设备的中断例行程序以及接口控制等功能模块,因此可直接用指令设置参数,然后用中断指令 INT 调用 BIOS 中的例行程序。使用 DOS 功能调用其操作更为简易,对硬件的依赖性更少一些。

常用的 BIOS 中断调用:

INT	10H 显示器	INT 16H	键盘
INT	12H 内存检验	INT 17H	打印机
INT	13H 磁盘	INT 1AH	时钟
INT	14H 串行通讯	INT 40H	软盘

常用的 DOS 中断调用:

INT 20H 程序结束

INT 21H 功能调用

键盘 I/O(AH=1、6、7、8、A、B、C)

显示器 I/O(AH=2、6、9)

打印机 I/O(AH=5)

串行通讯 I/O(AH=3、4)

5. 发声系统的程序设计

计算机产生声音有两种方法,一种方法是通过 I/O 指令向设备寄存器(端口地址为 61H)的第 1 位交替送 0 和 1,使与第 1 位相连的扬声器脉冲门产生连续的脉冲电流,驱动扬声器发出声音。

第二种方法是利用 8254(系统定时器)中的 2 号定时器向扬声器发送不同频率的脉冲,使之产生音调高低不同的声音,这种产生声音的方法可使计算机演奏出各种乐曲。

6. 利用 FCB(文件控制块)存取磁盘文件的程序设计方法

(1) 在程序的数据段设置 FCB 和 DTA(数据传输区)。

(2) 利用 DOS 功能调用打开文件(读)或建立文件(写),同时将文件名、扩展名、记录大小等信息保存在 FCB 中。

(3) 用 DOS 功能设置 DTA 地址。

(4) 用相应方式的 DOS 功能,以记录或字节块为单位读写文件。

(5) 在存取文件之后,特别是在写入文件之后,用 DOS 功能关闭文件。

7. 文件代号式存取磁盘文件的程序设计方法

(1) 用一个完整的路径名(ASCIZ 串)打开文件(读)或建立文件(写)。

(2) 保存 DOS 功能返回的文件代号。

(3) 利用功能调用 3FH 或 40H 读写文件。

(4) 利用功能调用 3EH 关闭文件。

(5) 如操作成功,CF=0;如操作不成功 CF=1,AX 中为错误代码。

例 题 分 析

例 8.1 下列程序能在系统时钟的控制下定时地使计算机终端响铃,同时也能把程序运行的时间记录下来。计算机的系统时钟约每秒发出 18.2 次中断请求,每次时钟中断要嵌套执行一次 INT 1CH 指令,因此用户可使用中断类型 1CH 编写一些有周期性处理功能的中断程序。

请分析下列程序并回答:

(1) 程序控制的响铃时间间隔是多少?

(2) 程序在执行第一次 CALL INCTEST 指令后的堆栈状态是什么?

```
;-----------------------------------------------
dseg    segment
        count   dw   0
        sec     dw   0
        min     dw   0
        hours   dw   0
dseg    ends
;-----------------------------------------------
cseg    segment
main    proc    far
        assume  cs:cseg, ds:dseg, es:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax

        mov     al,1ch              ;save old interrupt
        mov     ah,35h              ;  vector in stack
        int     21h
        push    es
        push    bx

        push    ds
        mov     dx,offset clint     ;set new interrupt
        mov     ax,seg clint        ;  vector
        mov     ds,ax
        mov     al,1ch
        mov     ah,25h
        int     21h
        pop     ds
```

```
        in      al,21h
        and     al,0feh
        out     21h,al          ;set interrupt mask bit
        sti

        mov     cx,10000        ;main part of program
mainp:  mov     ax,1000
again:  dec     ax
        jne     again
        loop    mainp

        pop     dx
        pop     ds
        mov     al,1ch          ;restore old
        mov     ah,25h          ;  interrupt
        int     21h             ;  vector
        ret
main    endp
;-------------------------------------------

clint   proc    near
        push    ds
        push    bx
        mov     bx,seg count
        mov     ds,bx
        lea     bx,count
        inc     word ptr [bx]
        cmp     word ptr [bx],18
        jne     exit
        call    inctest
adj:    cmp     hours,12
        jle     timeok
        sub     hours,12
timeok: mov     ax,sec
        and     ax,0007
        cmp     ax,5
        jne     exit
        mov     dl,07h          ;the bell ring
        mov     ah,2
        int     21h
exit:   mov     al,20h
        out     20h,al
        pop     bx
        pop     ds
```

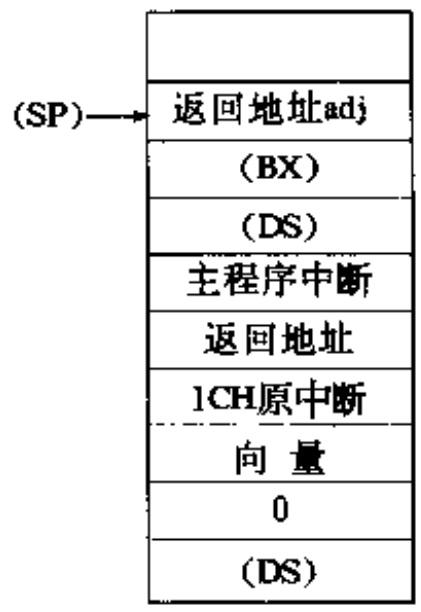
```
            iret                                ;interrupt return
clint       endp
;-------------------------------------------
inctest     proc near
            mov         word ptr [bx],0
            add         bx,2
            inc         word ptr [bx]
            cmp         word ptr [bx],60
            jne         return
            call        inctest                 ;adjust time
return:     ret
inctest     endp
;-------------------------------------------
cseg        ends
            end         start
```

图 8.02 （例 8.1）

这个程序由主程序 MAIN、中断处理程序 CLINT 和子程序 INCTEST 组成。在主程序中用 DOS 功能调用 35H，把中断类型 1CH 的原中断向量保存入栈，又将中断处理程序程序 CLINT 的段地址和偏移地址置入中断向量表，然后再把系统时钟的中断屏蔽位置为 0，用 STI 指令允许 CPU 开中断，这样在主程序的运行期间，系统时钟就以 1 秒 18.2 次的频率中断主程序。进入系统时钟的中断子程序后，当执行到 INT 1CH 指令时，计算机即从中断向量表中取得 CLINT 的地址，CPU 就转入 CLINT 中断子程序。CLINT 完成 COUNT 的计数，并且调用子程序 INCTEST，调整时、分、秒的计时值。响铃是根据秒值(SEC)的低位是否为 5 来控制的，即控制秒值为 5、15、25、35、…时响铃。分析整个程序，题目的两个问题应是：

（1）程序控制的响铃时间间隔是 10 秒。

（2）程序第一次执行 CALL INCTEST 指令后，堆栈状态如右图



例 8.2 从键盘上输入来宾的姓名 Mr. ×××或 Mrs. ×××，当按动任意一个键时，屏幕上显示出：

“Welcome Mr. ×××”。

```
;-------------------------------------------
dseg        segment
            maxlen      db      16
            actlen      db      ?
            names       db      16 dup(' ')
            mess1       db      'WELCOME '
            mess2       db      16 dup(' ')
            crlf        db      0dh,0ah,'$'
dseg        ends
;-------------------------------------------
```

```
cseg        segment
main        proc        far
            assume      cs:cseg, ds:dseg, es:dseg
            push        ds              ;set up stack for return
            sub         ax,ax
            push        ax
            mov         ax,dseg         ;set DS,ES to dseg
            mov         ds,ax
            mov         es,ax
            cld
begin:      mov         cx,16           ;initialize output buffer
            mov         al,20h
            lea         di,mess2
            rep         stosb
            lea         dx,maxlen
            mov         ah,0ah          ;input name
            int         21h

            mov         ah,7            ;input any key
            int         21h             ;  not echo
            cmp         al,3            ;ctrl_c,exit
            je          exit
display:
            mov         ch,0
            mov         cl,actlen       ;move name to
            lea         si,names        ; output buffer
            lea         di,mess2
            rep         movsb

            mov         ah,09h          ;display message
            lea         dx,mess1
            int         21h
exit:       ret
main        endp
cseg        ends
;-------------------------------------------
            end         main
```

图 8.03 （例 8.2）

利用 BIOS 和 DOS 功能调用，给编写程序带来很大的方便，但使用功能调用时要注意参数的设置以及其它一些特殊要求。如 INT 21H 的 0AH 功能是带缓冲区的键盘输入，在调用此功能时，要设置一个正确的数据缓冲区，缓冲区的第一个字节是允许键入的最大字符数，第二个字节将由 DOS 填入实际键入的字符数，从第三个字节开始才是由键盘输入的字符，而且字符的个数不能超过规定的最大字符数。又比如 INT 21H 的 09H 功能是显示字符串的

功能，此功能除要求在 DS：DX 中设置缓冲区首地址外，还要求字符串必须以‘$’结束。

有些 I/O 功能，既可用 DOS 中断，也可用 BIOS 中断，此时应尽可能使用 DOS 中断，因为 DOS 比 BIOS 更简便，更少依赖硬件，其程序的兼容性也好。

例 8.3 编写一个顺序写磁盘文件的程序。该文件包括姓名(<16 个字符)、年龄(1 个字)和电话号码(<10 个字符)，这些字符和数据在屏幕上出现提示符之后，由用户从键盘输入。

```
TITLE   SWRIT. ASM   ——   Write squential records
dseg        segment
            fcb         equ         5ch
            org         6ah
recsz       dw          ?
            org         7ch
recno       db          ?
            org         7eh
maxlen      db          32
actlen      db          ?
            org         80h
dta         db          32 dup(' ')
prompt      db          'please input name_age_tel.'
crlf        db          0dh,0ah,'$'
errmsg      db          'error !'
dseg        ends
;-----------------------------------------------
cseg        segment
main        proc        far
            assume      cs:cseg, ds:dseg
start:      push        ds
            sub         ax,ax
            push        ax
            mov         dx,fcb              ;create file func.
            mov         ah,16h
            int         21h
            cmp         al,0
            jnz         error               ;create file error
            mov         recsz,32            ;record size
            mov         recno,0             ;record number
            lea         dx,dta              ;set address of DTA
            mov         ah,1ah
            int         21h
disp:       push        ds
            mov         ax,dseg             ;set DS to dseg
            mov         ds,ax
            mov         ah,09h
```

```
            lea     dx,prompt
            int     21h
            pop     ds
input:      lea     dx,maxlen       ;input record
            mov     ah,0ah
            int     21h
            cmp     actlen,1
            jle     exit            ;no chars input
            mov     bl,actlen
            mov     bh,0
            mov     [dta+bx+1],0ah  ;insert LF
write:      mov     ah,15h          ;squent write func.
            mov     dx,fcb
            int     21h
            cmp     al,0
            jnz     error
            cld
            lea     di,dta          ;clear DTA with
            mov     al,20h          ;  space
            mov     cx,32
            rep     stosb
            push    ds
            mov     ax,dseg
            mov     ds,ax
            lea     dx,crlf
            mov     ah,09h          ;display CR/LF
            int     21h
            pop     ds
            jmp     input           ;get another record
exit:       mov     dta,1ah         ;set EOF mark
            mov     ah,15h          ;write EOF
            mov     dx,fcb
            int     21h
            cmp     al,0
            jnz     error
            mov     ah,10h          ;close file func.
            mov     dx,fcb
            int     21h
            ret
error:      push    ds
            mov     ax,dseg
            mov     ds,ax
            lea     dx,errmsg       ;display error
```

```
                mov     ah,09h          ; message
                int     21h
                pop     ds
                ret
main            endp
cseg            ends
;-------------------------------------------
                end     start
```

图 8.04 （例 8.3）

这个程序利用顺序写的 DOS 功能调用 15H，将数据传输区(DTA)中的记录写入磁盘。编写程序时，需要注意的是数据段寄存器 DS 的切换，因为 FCB 和 DTA 都在程序段前缀(PSP)中，FCB 开始于 5CH，DTA 开始于 80H。当程序被装入存储器时，操作系统使 DS 指向 PSP，因此读写文件时，程序访问 PSP 中的 FCB 和 DTA 的内容就直接使用 DS 的初始值，但要访问数据段中的内容，如显示提示信息和出错信息等就必须把 DS 切换为数据段的地址。

运行此程序时，同时写上电话记录的文件名(如 tellist. txt)，这样文件名就填入了 FCB。

C> swrit tellist. txt

文件写入磁盘后，可用 TYPE 命令检查刚输入的记录： C> type tellist. txt

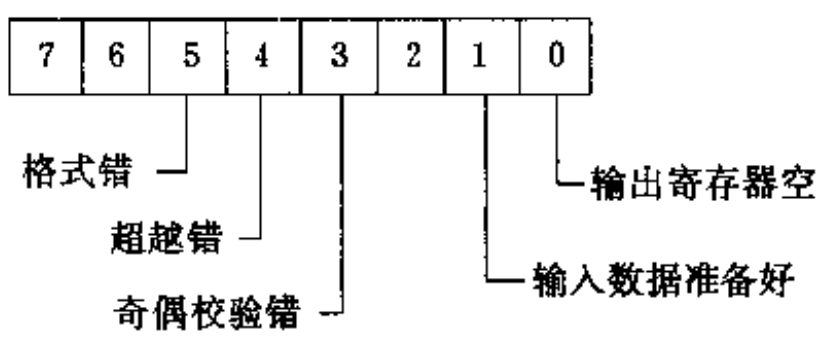
习　　题

程序控制输入/输出

8.1　写出指令，将一个字节输出到端口 25H。

8.2　写出指令，将一个字从端口 1000H 输入。

8.3　假设串行通讯口的输入数据寄存器的端口地址为 50H，状态寄存器的端口地址为 51H，它的各位为 1 时的含义如下：



请编写程序，输入一串字符并存入缓冲区 BUFF，同时检验输入的正确性，如有任何错误则转出错处理程序 ERR_ROUT。

8.4　试编写一段程序，它轮流测试两个设备的状态寄存器，只要一个状态寄存器的第 0 位为 1，则与其相应的设备就输入一个字符；如果其中任一状态寄存器的第 3 位为 1，则整个输入过程就结束。两个状态寄存器的端口地址分别是 0024 和 0036，与其相应的数据输入寄存器的端口为 0026 和 0038，输入字符分别存入首地址为 BUFF1 和 BUFF2 的存储区中。

8.5　从终端输入字符保存在一个 64 字节的数组 BUFFER 中，当输入一个回车符或字符数多于 62 时，输入结束。如果输入的前 63 个字符没有发现回车符，则从终端输出信息“BUFFER OVERFLOW”，否则自动在回车符后填入一个换行符。输入字节的第 7 位为偶校验

位，如果发生偶校验错，则转向出错处理程序 ERROR，如无校验错，则将字节的校验位清 0 后送 BUFFER。

假设终端接口的数据输入寄存器的端口地址是 0052，数据输出寄存器是 0053，状态寄存器是0054，其中第 1 位为 1 表示输入寄存器数据准备好，第 0 位为 1 表示输出寄存器是空闲的。

8.6　假设外部设备中有一台硬币兑换器，其状态寄存器的端口地址为 0006，数据输入寄存器的端口地址为 0005，数据输出寄存器的端口地址为 0007。试用查询方式编制一程序，该程序作空闲循环等待纸币输入，当状态寄存器的第 2 位为 1 时表示有纸币输入，此时可从数据输入寄存器输入的代码中测出纸币品种，一角纸币的代码为 01，二角纸币为 02，五角纸币则为 03，然后程序在状态寄存器的第 3 位变成 1 后，把应兑换的 5 分硬币数(用 16 进制表示)从数据输出寄存器输出。

中断处理

8.7　类型 14H 的中断向量在存储器的哪些单元里？

8.8　给定(SP)＝0100，(SS)＝0300，(PSW)＝0240，以及存储单元的内容(00020)＝0040，(00022)＝0100 ，在段地址为 0900 及偏移地址为 00A0 的单元中有一条中断指令 INT 8，试问执行 INT 8 指令后，SP、SS、IP、PSW 的内容是什么？栈顶的三个字是什么？

8.9　假设中断类型 9 的中断处理程序的起始地址为 INT_ROUT，试写出主程序为建立这一中断向量而编制的程序段。

8.10　编写指令序列，使类型 1CH 的中断向量指向中断处理程序 SHOW_CLOCK。

8.11　假设某类型的中断向量为 F000：FEA5，你如何找到此类型的中断处理程序所包含的指令？

8.12　如设备 D1、D2、D3、D4、D5 是按优先级次序排列的，设备 D1 的优先级最高，而中断请求的次序如下所示，试给出各设备的中断处理程序的运行次序。假设所有的中断处理程序开始后就有 STI 指令，并在中断返回之前发出结束命令。

(1) 设备 3 和 4 同时发出中断请求。

(2) 在设备 3 的中断处理程序完成之前，设备 2 发出中断请求。

(3) 在设备 4 的中断处理程序完成之前，设备 5 发出中断请求。

(4) 以上所有中断处理程序完成并返回主程序后，设备 1、3、5 同时发出中断请求。

8.13　在 8.12 题中，假设所有的中断处理程序中都没有 STI 指令，而它们的 IRET 指令都可以由于 PSW 出栈而使 IF 置 1，则各设备的中断处理程序的运行次序应是怎样的？

8.14　存储器中有一个首地址为 BUFFER 的 64 字节的缓冲区，其中存放着 ASCII 字符串，试编制实现以下功能的程序：在主程序运行期间(可用空闲循环来模拟)，每 10 秒钟响铃一次，(响铃的 ASCII 码为 07H)，当按动键盘上的一个键时，主程序和响铃被挂起。打印机以循环方式打印出 BUFFER 缓冲区中的相继字节(即打完最后一个字节后再从头重新开始打印)；当再按键盘上的一个键时，打印停止并恢复主程序及响铃，这一过程可以重复任意次。

8.15　试编制一程序，要求测出任意程序的运行时间，并把结果打印出来。

键盘和屏幕处理

8.16　INT 21H 的键盘输入功能 1 和功能 8 有什么区别？

8.17　编写一个程序，接收从键盘输入的 10 个十进制数字，输入回车符则停止输入，然

后将这些数字加密后(用 XLAT 指令)存入内存缓冲区 BUFFER。加密表为:

输入数字:0、1、2、3、4、5、6、7、8、9、

密码数字:7、5、9、1、3、6、8、0、2、4

8.18 对应屏幕上第 40 列最下边一个象素的内存单元地址是什么?

8.19 写出把光标置在第 12 行、第 8 列的指令。

8.20 编写指令把 12 行 0 列到 22 行 79 列的屏面清除。

8.21 编写指令使其完成下列要求:

(1) 读当前光标位置。

(2) 把光标移到屏底一行的开始。

(3) 在屏幕的左上角以正常属性显示一个字母 M。

8.22 编写程序段:按下 Home 键(扫描码为 47H),则将光标置在 0 行 0 列,否则光标位置不动。

8.23 编写一段程序,显示如下格式的信息:

Try again, you have n starfighters left。

其中 n 为 CX 寄存器中的 1～6 之间的二进制数。

8.24 从键盘上输入一行字符,如果这行字符比前一次输入的一行字符长度长,则保存该行字符,然后继续输入另一行字符;如果它比前一次输入的行短,则不保存这行字符。按下'$'键,则输入结束,最后将最长的一行字符显示出来。

8.25 编写程序,它在屏幕上显示出信息"What is the date(mm/dd/yy)?"并响铃(响铃符 007),然后从键盘上按要求格式输入数据并保存在 DATAFLD 存储区中。

8.26 在文本方式下,什么属性值产生以下各种字符?

(1) 黑底白字　　(2) 黑底白字,下划线　　(3) 白底黑字

(4) 兰底红字　　(5) 兰底红字、闪烁

8.27 写出以下指令序列:

(1) 设置 80 列黑白方式。　　(2) 把光标设置在第 5 行的开始。

(3) 上卷 10 行。　　(4) 显示 10 个闪烁的 * 号。

8.28 编写指令,以文本方式在品红底显示 5 个浅绿色的笑脸符。

8.29 从键盘上输入一串字符,当输入字符是回车符时,输入结束,并把光标置在 24 行 0 列。

8.30 编写指令:设置图形方式并选择背色为绿色。

8.31 编写指令:以图形方式从 12 行 13 列读一个点。

8.32 游戏程序常常用随机数来控制某图形在屏幕上移动,请编写一程序,用随机数来控制笑脸符(ASCII 码 02)显示的位置。笑脸符每次显示的列号总是递增 1,而行的位置可能是前行的上一行、下一行或同一行,这根据随机数是 0、1 或 2 来决定,当行号为 0、24 或列号为 79 时显示结束。笑脸符在每个位置上显示 1/4 秒。

(提示:随机数的产生可利用 INT 1AH 来读取时钟的低位数作为随机数的基数)

8.33 编写程序使一只鸟的图形飞过屏幕。飞鸟的动作可由小写字母 V(ASCII 码 76H)变为破折号(ASCII 码 0C4H)来模仿,这两个字符先后交替在两列上显示。鸟的开始位置是 0 列 20 行,每个字符显示 1/10 秒,然后消失。

打印和音响输出

8.34　请使用 DOS 中断按下列要示编写程序：

(1) 设置换页方式。

(2) 打印你的姓名。

(3) 执行换行并打印你的地址。

(4) 执行换行并打印你的城市。

8.35　用 BIOS INT 17H 完成上题的要求，包括测试打印机状态。

8.36　请在数据区中提供以下打印命令：回车，换页方式，压缩方式、一个文件名，退出压缩方式。

8.37　用户从键盘输入一文件并在屏幕上回显出来。每输入一行(80 个字符)，用户检查一遍，如果用户认为无须修改，则键入回车键，此时这行字符就存入 BUFFER 缓冲区保存，同时打印机把这行字符打印出来并换行。

8.38　按下面的乐谱编写一个乐曲程序。

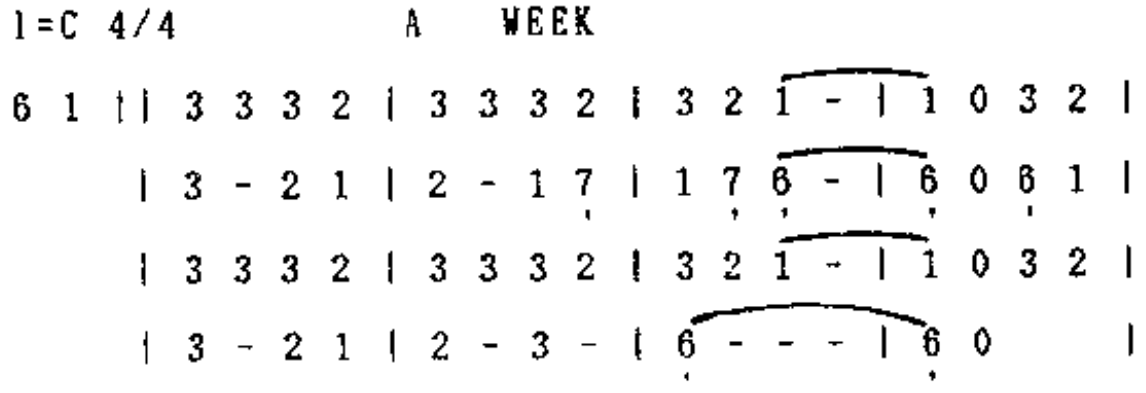


图 8.05　(习题 8.38)

8.39　编写用键盘选择计算机演奏歌曲的程序。程序首先在屏幕上显示出歌曲名单如下：

A　MUSIC1

B　MUSIC2

C　MUSIC3

当从键盘上输入歌曲的序号 A、B 或 C 时，计算机则演奏所选择的歌曲，当键盘上按下 0 键时，演奏结束。

8.40　请编写程序，使计算机在当前时间的 10 秒钟后执行乐曲程序 MY_SONG。

磁盘文件存取

8.41　请用 DOS 功能调用下面的操作：

(1) 建文件　(2) 设置 DTA　(3) 顺序写　(4) 打开文件　(5) 顺序读

8.42　编写建立并写入磁盘文件的程序，这个磁盘文件包括零件号(5 个字符)，零件名称(12 个字符)和单价(1 个字)。程序允许用户从键盘输入这些数据。

8.43　编写一个程序读出并显示 8.42 题建立的文件内容。

8.44　用下面的随机记录号确定当前块和当前记录。

(1) 45,　(2) 73,　(3) 750,　(4) 260

8.45　将随机记录号 2652(十进制)按规定填入 FCB。

8.46　写出以下操作的功能调用：

(1) 随机写　(2) 随机读　(3) 随机分块写　(4) 随机分块读

8.47　写出确定文件记录数的指令，假定打开文件操作已经执行，FCB 中的文件长度域为 FCBFLSZ，记录长度域为 FCBRCSZ。

8.48　利用 8.42 题建立起来的文件，写一个随机分块读程序，按下面的格式在屏幕上显示每个记录：

Part #	Price	Description
023	00315	Assemblers
024	00430	Linkages
027	00525	Compilers
049	00920	Compressors
114	11250	Extractors
117	00630	Hanlers
122	00520	Lifters
124	21335	Processors
127	00960	Labelers
232	05635	Bailers
999	00000	

图 8.06　(习题 8.48)

8.49　“文件未发现”和“非法文件代号”的错误返回码是什么？

8.50　定义一个名为 PATH1 的 ASCIZ 串：磁盘驱动器为 C，文件名为 CUST. LST。

8.51　按下列要求为 8.50 题的文件编写程序段。

(1) 为文件代号定义一个数据项 CUSTHAN。

(2) 建立这个文件。

(3) 从 CUSTOUT 缓冲区(128 字节)中写一个记录到磁盘。

(4) 关闭此文件，测试错误。

8.52　对于 8.51 题建立的文件，编写程序：

(1) 打开该文件。

(2) 把记录读入缓冲区 CUSTIN 并测试错误。

8.53　在什么情况下，应当关闭只读文件？

8.54　修改 8.52 题的程序，使用户能从键盘上输入要读出的文件名，并且按下回车键后即停止读出文件。

8.55　编写指令：用 BIOS INT 13H 来读出一个扇区的内容，存储器缓冲区为 INDSK，驱动器为 A、0 头、6 磁道、3 扇区。

8.56　编写指令：用 BIOS INT 13H 来写三个扇区，存储区地址为 OUTDSK，驱动器 B，0 头，8 磁道，1 扇区。

《汇编语言程序设计》自测题(一)

一、现有(AX)=2000H,(BX)=1200H,(SI)=0002H,(DI)=0003H,(DS)=3000H,
(SS)=3000H,(SP)=0000H,(31200H)=50H,(31201H)=02H,(31202H)=0F7H,
(31203H)=90H

请写出下列各条指令独立执行完后有关寄存器及存储单元的内容,若该指令影响条件码则请给出条件码 SF,ZF,OF,CF 的值。

1. ADD AX, 1200H
2. SUB AX, BX
3. MOV [BX], AX
4. PUSH AX
5. DEC BYTE PTR [1200H]
6. NEG WORD PTR [1200H]
7. SAR BYTE PTR 1200[SI],1
8. ROL BYTE PTR [BX+SI+1],1
9. MUL WORD PTR [BX][SI]
10. DIV BYTE PTR 1200[DI]

二、判断下列指令是否正确:

1. POP CS ( )
2. PUSH WORD PTR 20[BX+SI-2] ( )
3. LEA BX, 4[BX] ( )
4. JMP BYTE PTR[BX] ( )
5. SAR AX,5 ( )
6. MOV BYTE PTR[BX],1000 ( )
7. CMP [DI],[SI] ( )
8. ADD BX,OFFSET A ( )
9. IN AL,DX ( )
10. MUL 25 ( )

三、试分析下列程序段执行完后,A 单元的内容是什么?

```
data    segment
        A    dw    0
        B    dw    0
        C    dw    230,20,54
data    ends
code    segment
          ⋮
        MOV   BX,OFFSET  C
```

```
        MOV     AX,[BX]
        MOV     B,  AX
        MOV     AX,2[BX]
        ADD     AX,B
        MOV     A,  AX
          ⋮
code    ends
```

四、假设 X 和 X+2 单元的内容为双精度数 P,Y 和 Y+2 单元的内容为双精度数 Q (X,Y 为低位字),下列程序段使 2P>Q 时 (AX)=1,2P<=Q 时 (AX)=2. 请把程序段填写完整。

```
        MOV     DX, X+2
        MOV     AX, X
        ADD     AX, X
        ADC     DX, X+2
        CMP     DX, Y+2
        (  )    L2
        (  )    L1
        CMP     AX, Y
        (  )    L2
L1:     MOV     AX, 1
        JMP     EXIT
L2:     MOV     AX,2
EXIT:   INT     20H
```

五、分析下列备有定时器中断处理程序 UPDAT 的程序。

问:如果第一次定时器中断发生在 a. (t1)点 b. (t2)点 c. (t3)点 d. (t4)点,主程序运行完后 AX 的内容分别是什么?

```
dseg    segment
  save_ip8   dw?
  save_cs8   dw?
dseg    ends
;-------------------
cseg    segment
  assume  cs:cseg,ds:dseg
main    proc    far
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
```

```
        mov     al,08h                  ;保存原中断向量
        mov     ah,35h
        int     21h
        mov     save_cs8,es
        mov     save_ip8,bx

        push    ds
        mov     ax,seg updat            ;设置新中断向量
        mov     ds,ax
        mov     dx,offset updat
        mov     al,08h
        mov     ah,25h
        int     21h
        pop     ds

        in      al,21h                  ;允许定时器中断
        and     al,0feh
        out     21h,al
        sti                             ;开中断
(t1)→   mov     ax,5
        or      ax,ax
(t2)→
        js      l1
(t3)→
        jns     l2
(t4)→
        inc     ax
        jmp     l2
l1:     dec     ax
l2:     cli                             ;关中断
        push    ds
        push    ax
        mov     dx,save_ip8             ;恢复原中断向量
        mov     ax,save_cs8
        mov     ds,ax
        mov     al,08h
        mov     ah,25h
        int     21h
        pop     ax
        pop     ds
```

```
        sti
        ret                                     ;返回 DOS
main    endp
;-------------------------------------------------
updat   proc    far                             ;中断处理程序
        push    ax
        push    bp
        mov     bp,sp
        mov     dx,8[bp]
        or      dx,0080h
        mov     8[bp],dx
        mov     al,20h
        out     20h,al
        pop     bp
        pop     ax
        iret                                    ;中断返回
updat   endp
;-------------------------------------------------
cseg    ends
        end     start
```

六、定义宏指令 PRINTBK：利用 DOS 调用完成打印机连续打印一串字符的功能，如果字符串中出现列表符 TAB（ASCII 码为 09H），则打印 8 个空格符（ASCII 码为 20H）来代替它，字符串首地址及长度为变元。

七、现有

```
data    segment
ARRAY   DW      64      DUP ( ? )
        DW      5       DUP ( ? )
data    ends
```

试编制一程序段，要求在 ARRAY+8 到 ARRAY +10（两者都包括在内）单元中插入 5 个 0。（提示：先把从 ARRAY +8 单元开始的数据后移 5 个单元，然后再在规定位置插入 0 值）

《汇编语言程序设计》自测题 （二）

一、假设 (CS)＝3000H，(DS)＝4000H，(ES)＝2000H，(SS)＝5000H，(AX)＝2060H，
(BX)＝3000H，(CX)＝0005H，(DX)＝0，　(SI)＝2060H，(DI)＝3000H，
(43000H)＝0A006H，(23000H)＝0B116H，(33000H)＝0F802H，
(25060H)＝00B0H，(SP)＝0FFFEH，(CF)＝1，(DF)＝1

请写出下列各条指令独立执行完后，有关寄存器及存储单元的内容，若影响条件码请给出条件码 SF，ZF，OF，CF 的值。

1. SBB　　AX，　BX
2. CMP　　AX，　WORD　PTR[SI＋0FA0H]
3. MUL　　BYTE PTR[BX]
4. AAM
5. DIV　　BH
6. SAR　　AX，　CL
7. XOR　　AX，　OFFE7H
8. REP　　STOSB
9. JMP　　WORD　PTR[BX]
10. XCHG　AX，　ES:[BX＋SI]

二、填空：

1. 一个有 16 个字的数据区，它的起始地址为 70A0:DDF6，那么该数据区的最后一个字单元的物理地址为(　　H)。
2. 假设 (SS)＝2250H，(SP)＝0140H，如果在堆栈中存入 5 个数据，则栈顶的物理地址为(　　H)。如果又从堆栈中取出 3 个数据，则栈顶的物理地址为(　　H)。
3. 在 sub ax，bx 指令执行后，CF＝1，说明：
   (a) 最高有效位_____。
   (b) 对_____数，操作结果溢出。
4. 某程序的数据段定义如下：

```
DATASG      SEGMENT
  PARTLIST    DB    'PART#'
              DB     60
              DB    'PRICE'
              DW     0125
              DB    'DESCRIPTION'
              DB    'RADIO'
  LISTLENG    EQU   (          );数组长度，用表达式表示。
DATASG      ENDS
```

三、某程序设置数据区如下：

```
        ORG     100H
NAMES   DB      'TOM..',20
        DB      'ROSE.',25
        DB      'KATE.',22
```

1. 列出该数据区的 LST 清单(即各字节单元的存储情况)。
2. 下列各组指令,若为合法指令,请写出执行后的结果;若为非法指令,请指出其错误.

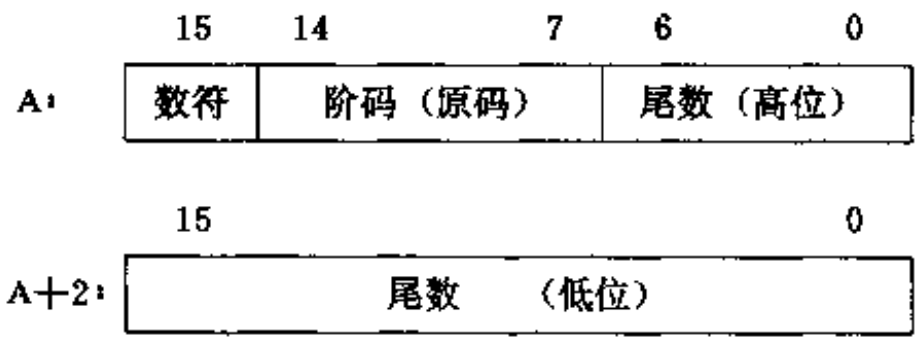
```
(a)     MOV     BX, OFFSET  NAMES
        MOV     AX, [BX+5]
(b)     MOV     BX, OFFSET  NAMES+11
        CMP     [BX], BYTE  PTR  NAMES+5
(c)     MOV     BX, 6 * 2
        MOV     SI, 5
        MOV     DI, OFFSET [BX][SI]
        INC     [DI]
(d)     MOV     SI, 5
        LEA     DI, NAMES 6 [SI]
        MOV     AL, [DI]
```

四、1. 定义宏指令 BIGER :把字变量 X 和 Y 中较大者存入 BIG ,若 X 和 Y 相等时,则把其中之一存入 BIG。

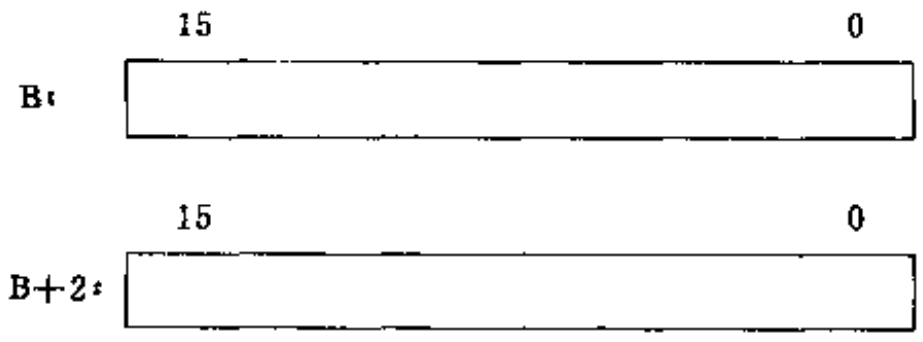
2. 宏指令 DISP 完成:根据不同的功能码(2,6,9)分别产生显示单字符或字符串的程序段。

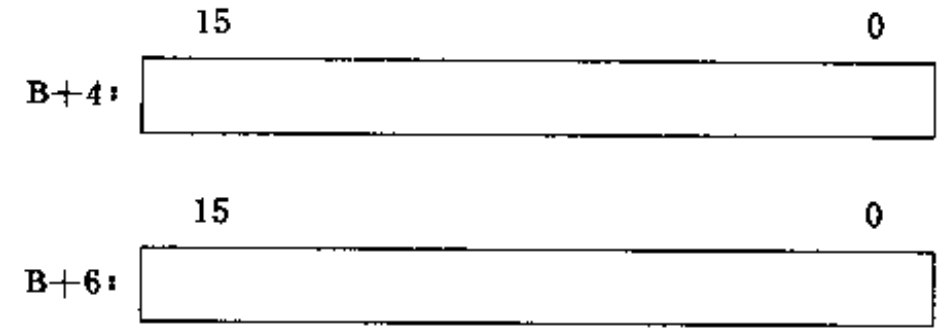
并调用:
```
DISP  2, 0DH
DISP  2, 0AH
DISP  9, STRING
```

五、假如某计算机的浮点数格式为:



计算机在作浮点运算时,通常要把浮点数的数符,阶码及尾数分离出来分别进行处理。下面的程序就是把一个浮点数分离开来,并分别存入 B 数组。请阅读该程序,并填出分离后 B 数组各单元的存储格式。






```
dseg    segment
   A    DD   ?                    ;浮点数
   B    DW   4 dup ( ? )
dseg    ends
cseg    segment
   main  proc  far
   assume  cs: cseg,ds: dseg
start:    push    ds
          sub     as,0
          push    ax
          mov     ax,dseg
          mov     ds,ax
          mov     bx,0            ;分离数符
          mov     ax,wokd ptr A
          shl     ax,1
          rcr     bx,1
          mov     B,bx
          test    ax,8000h        ;分离阶码
          jz      brk             ;并形成补码
          xor     ax,8000h
          neg     ah
brk:      and     ax,0ff00h
          mov     B+2,ax
          mov     ax,word ptr A   ;分离尾数高位
          and     ax,007fh
          or      ax,80h
          mov     B+4,ax
          mov     ax, word ptr A+2    ; 尾数低位
          mov     B+6, ax
          ret
main      endp
cseg      ends
          end     start
```

六、已建立的某数据文件每个记录都是长度为 22 个字节的字符数据。现要求根据键入的记

录序号 00～99，将相应的记录读取并显示出来。该项工作要求由两个独立的程序模块完成：主模块包括定义了 FCB 和 DTA 数据段以及主程序（main）。子模块中应包括接收数据文件名的子程序（get_name）以及打开文件、设置记录号并读取记录的子程序（openf，recno，readf）。

请阅读主模 块程序，并根据主模块的调用和要求编写子模块。

```
;----------<主 模 块>----------
extrn   get_name:far, openf:far
extrn   recno:far, readf:far
dseg    segment     common
  fcbdriv           db          0
  fcbname           db          8 dup (20h)
  fcbext            db          3 dup (20h)
  fcbblk            dw          0
  fcbrecsz          dw          0
                    db          16 dup (?)
  fcbcurec          db          0
  fcbrarec          dw          0
                    dw          0
  recfld            db          22 dup (20h)
                    db          0dh,0ah, '$'
  maxlen            db          12
  actlen            db          ?
  kbuffer           db          12 dup(20h)
  ermsg   db   0dh, 0ah, 'error! $'
  mess0   db   0dh, 0ah, 'data file name: $'
  mess1   db   0dh, 0ah, 'record number: $'
  crlf    db   0dh, 0ah, '$'
dseg        ends
;-------------------------------
cseg        segment   'code'
assume      cs : cseg, ds : dseg, es : dseg
print       macro     mess
            mov       dx, offset mess
            mov       ah,9
            int       21h
            endm
kbinp       macro     buffer
            mov       dx, offset buffer
            mov       ah,0ah
            int       21h
```

```
        endm
;-----------------------------------------------
main    proc    far
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
        print   mess0
        kbinp   maxlen          ;键入文件名.扩展名
        call    far ptr get name    ;文件名送 FCB
        call    far ptr openf       ;打开文件,设置 DTA
next:   print   mess1
        kbinp   maxlen          ;键入记录号
        cmp     actlen,0
        je      exit
        call    far ptr recno       ;随机记录号送 FCB
        call    far ptr readf       ;读出指定的记录
        print   crlf
        print   recfld          ;显示记录
        jmp     next
exit:   ret
main    endp
;-----------------------------------------------
cseg    ends
        end     start
```

七、编写一完整的中断程序,要求在键盘上每按键 10 次,即响铃并显示信息:"Pressed 10 times!"。

参 考 答 案

1.1

(1) 1,0111,0001b;171h

(3) 1111,1111,1111b;0FFFh

1.2

(1) 2Dh; 45d

(3) 0FFFFh; 65535d

1.3

(1) 1111,1010b;250d

(3) 1111,1111,1111,1110b;65535d

1.4

(1) 0010,0010 (3) 1,0000,0000

1.5

(1) 241C (3) 8000

1.6

(1) 00011010 (3) 1001,0101

1.7

(1) 5555 (2) 0FFF

1.8

(1) 11101101　(3) 11000111

1.9

(1) 00111000　(3) 10000000

1.10

(1) 0F7h；CF=0；OF=0

(3) 9h；　CF=0；OF=0

(5) 5Fh；　CF=0；OF=1

1.11

(1) 51h　(3) 0BDH　(5) 20h

1.12

(1) 79；O　(3) 115；S

1.13

46 6F 72 20 65 78 61 6D 70 6C 65 2C 0A 0D 54 68 69 73 20 69 73 20 61 20 6E 75 6D 62 65 72 20 33 36 39 32

1.14

(1) 0001，0000b；10h

(3) 0111，1111b；7Fh

(5) 1111，0000b；0F0h

(7) 1000，0000b；80h

1.15

32767～－32768

1.16

0～65535

1.17

(1) A>B　(2) B>A

1.18

(1) 20547　(3) 5043

1.19

(1) 74D4h；0，0，0，0

(3) 3240h；0，0，1，0

1.20

(1) 0C754h；1，0，1，0

(3) 45B0h；0，0，0，1

1.21

BUFFER　DB 34，30，39，36

UNPAK　DB 06，09，00，04

1.22

INCOME1 DB 07，06，07，02，03

2.1

65536 字节(64K)；0～FFFF

2.3

ROM（只读 存储器)永久保存计算机的启动程序和处理 I/O 的例行程序。RAM（随机存储器)暂时保存计算机执行时驻留的程序的数据。

2.5

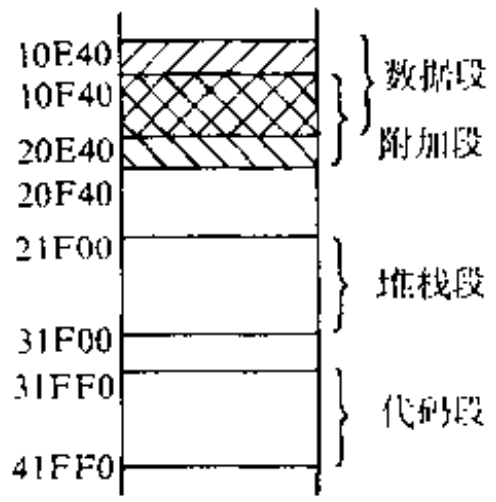
0100 49 42 4D 2D 50 43 20 50 0108 45 52 53 4F 4E 41 4C 20 0110 43 4F 4D 50 55 54 45 52

2.7

(1) 2314；0035；23175

(2) 1FD0；000A；1FD0A

2.9



OF、SF、ZF、CF 均为 0

2.11

DS、ES、SS、CS、BX、SI、DI、IP、BP、SP

2.13

1. (D)　2. (E)　3. (O)　4. (C)　5. (F)　6. (B)　7. (A)　8. (N)　9. (M)　10. (L)　11. (H)　12. (J)　13. (I)　14. (G)　15. (K)　16. (R)　17. (Q)　18. (P)

3.1

(1) 立即方式；操作数在本条指令中

(3) 直接寻址方式；20100H

(5) 寄存器间接方式；20100H

(7) 寄存器间接方式；15010H

(9) 相对寻址方式；20110H

(11) 基址变址方式；201A0H

3.3

(1) 1200H　(3) 4C2AH

(5) 4C2AH　(7) 65B7H

3.5

(1)、(2)、(4)不对

3.7

```
mov bx, offset count
mov al, es:[bx]
```

3.9

将 DAX1 中的后 100 个数据传送到 DATX2 中。

3.11

(1) 寄存器类型不匹配。

(3) SI、DI 不能一起用。

(5) 1000 超出一个字节的范围。

(7) CS 不能用作目的寄存器。

3.13

(1) add dx, bx

(3) add [bx+0b2H],cx

(5) add al, 0b5H

3.15

```
(1) mov ax, 4629h
(3) mov bx, seg mydat
    mov ds, bx
    mov bx, offset mydat
```

3.17

```
mov al,25h
shl al, 1
mov bl, 15h
mul bl
```

3.19

```
(1) mov     ax, datax
    add     datay, ax
    mov     ax, datax+2
    add     data+2, ax
```

(3) DATAX 和 DATAY 中的字数据之和加 1.

```
(5) aa dw 0
    bb dw 0
    cc dw 0
    dd dw 0
     ⋮
    mov     ax, datax
    mul     datay
    mov     aa , ax
    mov     bb, dx
    mov     ax, datax
    mul     datay+2
    add     bb, ax
    adc     cc, dx
    mov     ax, data+2
    mul     datay
    add     bb, ax
    adc     cc, dx
    mov     ax, datax+2
    mul     datay+2
    add     cc, ax
    adc     dd, dx
(7) mov     dx, datax+2
    mov     ax, datax
    div     datay
```

3.21

```
neg dx
neg ax
sbb dx,0
```

3.23

0a8ch; (0F)=(CF)=1

f88ch; (0F)=(CF)=1

3.25

```
lea si,n
lea di,ni
add di, i
mov cx, i+1
move:   cld
        lodsb
        std
        stosb
        loop move
```

3.27

(1) 10011010 (3) 11111011

(5) 0

3.29

```
test bx, 2000H
jnz yes
```

3.31

(1) 005ch (3) 05c8h

(5) 2017h (7) 00b9h

(9) 00dch

3.33

```
mov     cl,4
shr     ax, cl
mov     bl, dl
shr     dx, cl
shl     bl, cl
or      ah, bl
```

3.35

```
sal ax, 1
rcl dx,1
```

3.37

```
sal bx,1
rcl ax,1
rcl dx,1
```

3.39

能,因为移位指令不改变 cl 的值。

3.41

```
(1) next:  mov di, [si]
           mov [di],dl
           inc si (或 dec)
           inc di (或 dec)
           loop next
(3) next:  mov [di],ax
           inc di (或 dec)
           loop next
```

3.43

```
cld
mov     cx, 20
lea     si, string1
lea     di, string2
rep     movsb
```

3.45

(1) 把 first 数组中的 10 个字符传送到 second 数组。

(2) movsb 先执行

(3) 将(first)→(second),

(si)+1→(si),(di)+1→(di)

(4) (cx)−1→(cx),然后重复执行 rep movsb。

3.47

```
cld
mov cx, count
mov al, ' '
lea di, char_field
rep stosb
```

3.49

```
(1) cld
    mov cx,50
    mov al, ' '
    lea di, print_line
    rep stosb
(3) mov cx, 9
    lea di, student_addr+8
    mov al, '_'
    std
    repne scasb
    cld
    jne no_dash
    ...
(5) mov cx,30
    lea si, student_name
    lea di, print_line
    rep movsb
    std
    mov cx,9
    lea si, student_addr+8
    lea di, print_line+49
    rep movsb
    cld
```

3.51

(1)109h (2) 11bh

3.53

```
  add ax, bx
  jno ok
```

3.55

```
        cmp     ax,bx
        jge     next
        xchg    ax,bx
next:   cmp     ax, cx
        jge     done
        xchg    ax,cx
done:…
```

3.57

```
        mov     cx,100
        lea     bx, array
incr:   inc     [bx]
        add     bx, 2
        loop    incr
```

3.59

(1)L1 (2) L1 (3) L2

(4) L5 (5) L5

3.61

```
          cmp     ax,−1
          jle     none
          cmp     ax, 1
          jl      zero
          mov     ax,1
          jmp     out
  none:   mov     ax,−1
          jmp     out
  zero:   mov     ax,0
  out:    ret
```

3.63

```
        mov     cx, 50
```

```
        lea     si, elems
        mov     [di],al
        add     di,99
reverse: mov    al,[si]
        xchg    al, [di]
        mov     [si],al
        inc     si
        inc     di
        loop    reverse
```

3.65

```
        asc1    db '578'
        asc2    db '694'
        asc3    db '0000'
        ...
        clc
        lea     si, asc1+2
        lea     di, asc2+2
        lea     bx, asc3+2
        mov     cx,3
a20:    mov     ah,0
        mov     al, [si]
        adc     al,[di]
        aaa
        mov     [bx],al
        dec     si
        dec     di
        dec     bx
        loop    a20
        mov     [bx], ah
        ret
```

3.67

```
        multend db '3785'
        multplr db '5'
        product db 5 dup (0)
        ...
        mov     cx, 4
        lea     si, multend+3
        lea     di, product+4
        and     multplr, 0fh
A30:    mov     al, [si]
        and     al, 0fh
        mul     multplr
        aam
        add     al, [di]
        aaa
        dec     di
        mov     [di], ah
        dec     si
        loop    a30
        ret
```

4.1

(1) 应有一个操作数为寄存器。

(2) 转向地址应为标号,不能是变量。

(5) 少 ptr

4.3

```
dw   5150h
db   50h, 51h
db   'PQ'
```

4.5

x1 不能多次赋值,

x2 可以多次赋值。

4.7

L=6

4.9

```
mov  ax,10
mov  bl, 10
mov  cl,1
```

4.11

```
0000
=0064
=0078
0000 2F 78 79 7A 2F 0D 0A
0007 05 13 61
0010
000A      11 [20]
001B 90
001C 3132 000D 001A 0333 0078
000A
0026 000F
=0028
0028
```

4.13

opr1 and opr2 是一个表达式,and 为逻辑操作符,它在汇编时求得一个值。and 指令在程序运行时执行。

4.15

(1) 不能有空格。

(3) 第一个字符不能为数字。

4.17

(1) stack_seg segment stack 'stack'

(3) segname segment 'code'

(5) main_proc endp

end main_proc

4.19

```
d_seg    segment
augw     label word
augend   dd 99251
sum      dw 2 dup (?)
d_seg    ends
e_seg    segment
addw     label word
addend   dd -15962
e_seg    ends
c_seg    segment
assume   ds: d_seg, es:e_seg
main     proc    far
start:   push    ds
         mov     ax,0
         push    ax
         mov     ax, d_seg
         mov     ds,ax
         mov     ax, e_seg
         mov     es,ax
addt:    mov     ax, augw
         mov     bx, augw+2
         add     ax, es:addw
         adc     bx, es:addw+2
         mov     sum, ax
         mov     sum+2,bx
         ret
main     endp
c_seg    ends
         end     start
```

4.21

```
dseg     segment
  pattern   db 23h, 24h, 25h,26h
  display db 80 dup (' '), '$'
cseg     segment
         ⋮
         cld
         lea     si, pattern
         lea     di, display
         mov     cx, 20
         rep     movsw
         mov     ah, 09h
         lea     dx, display
         int     21h
         ret
cseg     segment
         end
```

5.1

```
begin:   mov     ah,1
         int     21h
         cmp     al, 'a'
         jb      stop
         cmp     al, 'z'
         ja      stop
         sub     al,20h
         mov     dl,al
         mov     ah,2
         int     21h
         jmp     begin
stop:    ret
```

5.3

```
begin:   mov     ah,1
         int     21h
         cmp     al, 'a'
         jb      stop
         cmp     al, 'z'
         ja      stop
         dec     al
         mov     dl,al
         mov     cx,3
display:
         mov     ah,2
         int     21h
         inc     dl
         loop    display
stop:    ret
```

5.5

```
         mov     ax,A
         sub     ax,B
         jo      error
         ...
error:
```

5.7

```
dseg    segment
store   db 4 dup (?)
dseg    ends
        ...
begin:  mov     cl,4
        mov     ch,4
        lea     bx,store
a10:    mov     dx,ax
        and     dx,0fh
        mov     byte  ptr[bx],dl
        inc     bx
        shr     ax,cl
        dec     ch
        jnz     a10
b10:    mov     dl,store
        mov     cl,store+1
        mov     bl,store+2
        mov     al,store+3
        ret
```

5.9

```
dseg    segment
string1 db 'I am a student.'
string2 db 'I am a student!'
yes     db 'MATCH',0dh,0ah,'$'
no      db 'NO  MATCH',0dh,0ah,'$'
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,      ds:dseg,es:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
begin:  lea     si,string1
        lea     di,string2
        mov     cx,string2-string1
        repe    cmpsb
        jnc     dispno
        mov     ah,09
        lea     dx,yes
        int     21h
        ret
dispno: mov     ah,09
        lea     dx,no
        int     21h
        ret
main    endp
cseg    ends
        end     start
```

5.11

```
dseg    segment
array   dw 3 dup (?)
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     cx,3
        lea     si,array
begin:  push    cx
        mov     cl,4
        mov     di,4
        mov     dl,' '
        mov     ah,02
        int     21h
input:  mov     ah,01
        int     21h
        and     al,0fh
        shl     dx,cl
        or      dl,al
        dec     di
        jnc     input
        mov     [si],dx
        add     si,2
        pop     cx
        loop    begin
compa:  lea     si,array
        mov     dx,0
        mov     ax,[si]
        mov     bx,[si+2]
        cmp     ax,bx
```

```
jne     next1
        inc     dx
next1:  cmp     [si+4],ax
        jne     next2
        inc     dx
next2:  cmp     [si+4],bx
        jne     num
        inc     dx
num:    cmp     dx,3
        jl      disp
        dec     dx
disp:   mov     ah,2
        add     dl,30h
        int     21h
        ret
main    endp
cseg    ends
        end     start
```

5.13

```
dseg    segment
a       dw?
b       dw?
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
begin:  mov     ax,a
        mov     bx,b
        xor     ax,bx
        test    ax,0001
        jz      class
        test    bx,0001
        jz      exit
        xchg    bx,a
        mov     b,bx
        jmp     exit
class:  test    bx,0001
        jz      exit
        inc     b
        inc     a
exit:   ret
main    endp
cseg    ends
        end     start
```

5.15

```
dseg    segment
x       dw -4
fx      dw ?
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
begin:  cmp     x,5
        jg      a0
        cmp     x,-5
        jl      a0
        mov     bx,1
        sub     bx,x
        mov     fx,bx
        ret
a0:     mov     fx,0
        ret
main    endp
cseg    ends
        end     start
```

5.17

```
dseg    segment
new1    db 'newsA',0dh,0ah,'$'
new2    db 'newsB',0dh,0ah,'$'
new3    db 'newsC',0dh,0ah,'$'
  ⋮
new10   db 'newsJ',0dh,0ah,'$'
news    dw new1,new2,new3,...,new10
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg
start:  push    ds
```

```
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
begin:  mov     cx,10
        lea     bx,news
disp:   mov     dx,[bx]
        add     bx,2
        mov     ah,09
        int     21h
        loop    disp
        jmp     begin
main    endp
cseg    ends
        end     start
```

5.19

```
begin:  mov     ah,1
        int     21h
        and     al,0fh
        cbw
        mov     cx,ax
        jcxz    exit
bell:   mov     dl,07
        mov     ah,02
        int     21h
        loop    bell
exit:   ret
```

5.21

```
        mov     bx,0
        mov     ch,4
        mov     cl,4
input:  shl     bx,cl
        mov     ah,1
        int     21h
        cmp     al,39h
        ja      af
        and     al,0fh
        jmp     binary
af:     and     al,0fh
        add     al,9
binary: or      bl,al
        dec     ch
        jne     input
dispn:  mov     cx,16
disp:   mov     dl,0
        rol     bx,1
        rcl     dl,1
        or      dl,30h
        mov     ah,02
        int     21h
        loop    disp
        ret
```

5.23

```
dseg    segment
x       dw      32767
one     dw      0
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg, ds:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     bx,0
        mov     ch,4
        mov     cl,4
begin:  mov     cx,16
        mov     bx,0
        mov     dx,0001h
comp:   mov     ax,x
        and     ax,dx
        jz      next
        inc     bx
next:   shl     dx,1
        loop    comp
        mov     one,bx
        ret
main    endp
cseg    ends
        end     start
```

5.25

```
dseg    segment
eng     db   'Here is sun,sun,...'
disp    db   'Sun : '
dat     db   '0000' , '$'
keyword db  'sun'
```

```
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg, ds:dseg, es:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
begin:  mov     ax,0
        mov     dx,disp-eng-2
        lea     bx,eng
comp:   mov     di,bx
        lea     si,keyword
        mov     cx,3
        repe    cmpsb
        jnz     no_match
        inc     ax
no_match:
        inc     bx
        dec     dx
        jnz     comp
done:   mov     ch,4
        mov     cl,4
        lea     bx,dat
done1:  rol     ax,cl
        mov     dx,ax
        and     dx,0fh
        add     dl,30h
        cmp     dl,39h
        jle     store
        add     dl,07h
store:  mov     [bx],dl
        inc     bx
        dec     ch
        jnz     done1
display:
        lea     dx,disp
        mov     ah,09h
        int     21h
        ret
main    endp
cseg    ends
        end     start
```

5.27

```
dseg    segment
buff    db  50 dup(' ')
count   dw  0
dseg    ends
        ...
begin:  lea     bx,buff
        mov     count,0
input:  mov     ah,01
        int     21h
        mov     [bx],al
        inc     bx
        cmp     al,'$'
        jnz     input
        lea     bx,buff
next:   mov     cl,[bx]
        inc     bx
        cmp     cl,'$'
        jz      disp
        cmp     cl,30h
        jb      cont
        cmp     cl,39h
        jbe     next
cont:   inc     count
        jmp     next
disp:   ...
```

5.29

```
dseg    segment
mem     dw  100  dup (?)
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg, ds:dseg, es:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
begin:  mov     si,(100-1)*2
        mov     bx,-2
        mov     cx,100
comp:   add     bx,2
```

```
        cmp     mem[bx],0
        jz      cons
        loop    comp
        jmp     finish
cons:   mov     di,bx
cons1:  cmp     di,si
        jae     nomov
        mov     ax,mem [di+2]
        mov     mem [di],ax
        add     di,2
        jmp     cons1
nomov:  mov     word ptr [si],0
        loop    comp
finish: ret
main    endp
cseg    ends
        end     start
```

5.31

```
dseg    segment
string  db  100  dup (?)
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg,es:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
begin:  mov     cx,100
        mov     si,0
repeat: mov     al,string [si]
        cmp     al,30h
        jb      goon
        cmp     al,39h
        ja      goon
        or      dl,20h
        jmp     exit
goon:   inc     si
        loop    repeat
        and     dl,0dfh
exit:   ret
main    endp
cseg    ends
        end     start
```

5.33

```
dseg    segment
table   dw  100h  dup (?)
data    dw  ?
count   dw  0
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg,es:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
begin:  mov     bx,100h
        mov     di,0
next:   mov     dx,0
        mov     si,0
        mov     ax,table [di]
        mov     cx,100h
comp:   cmp     table [si],ax
        jne     addr
        inc     dx
addr:   add     si,2
        loop    comp
        cmp     dx,count
        jle     done
        mov     count,dx
        mov     data,ax
done:   add     di,2
        dec     bx
        jnz     next
        mov     cx,count
        mov     ax,data
        ret
main    endp
cseg    ends
        end     start
```

5.35

```
dseg    segment
data    dw  100h  dup  (?)
```

```
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg,es:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
begin:  mov     cx,100h-1
        mov     dx,0
        mov     si,0
        mov     ax,data [si]
        cwd
next:   add     si,2
        add     ax,data [si]
        adc     dx,0
        jo      error
        loop    next
        mov     cx,100h
        idiv    cx
        mov     bx,0
        mov     si,0
comp:   cmp     ax,data [si]
        jle     no
        inc     bx
no:     add     si,2
        loop    comp
        ret
error:  mov     dl,'!'
        mov     ah,02
        int     21h
        ret
main    endp
cseg    ends
        end     start
```

5.37

```
dseg    segment
grade   dw  30  dup  (?)
rank    dw  30  dup  (0)
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg,es:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
begin:  mov     di,0
        mov     cx,30
loop1:  push    cx
        mov     cx,30
        mov     si,0
        mov     ax,grade [di]
        mov     dx,0
loop2:  cmp     grade [si],ax
        jbe     go_on
        inc     dx
go_on:  add     si,2
        loop    loop2
        pop     cx
        inc     dx
        mov     rank [di],dx
        add     di,2
        loop    loop1
        ret
main    endp
cseg    ends
        end     start
```

5.39

```
dseg    segment
a       dw  15  dup  (?)
b       dw  20  dup  (?)
c       dw  15  dup  ('.')
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg,es:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
begin:  mov     si,0
```

```
            mov     bx,0
            mov     cx,15
loop1:      push    cx
            mov     di,0
            mov     cx,20
            mov     ax,a [si]
loop2:      cmp     b [di],ax
            jne     no
            mov     c [bx],ax
            add     bx,2
no:         add     di,2
            loop    loop2
            add     si,2
            pop     cx
            loop    loop1
            ret
main        endp
cseg        ends
            end     start
```

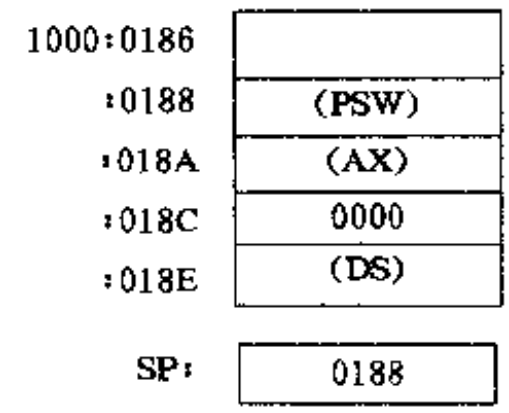
6.1

(1)、(11)、(15)、(19)是非法指令，其余指令合法。

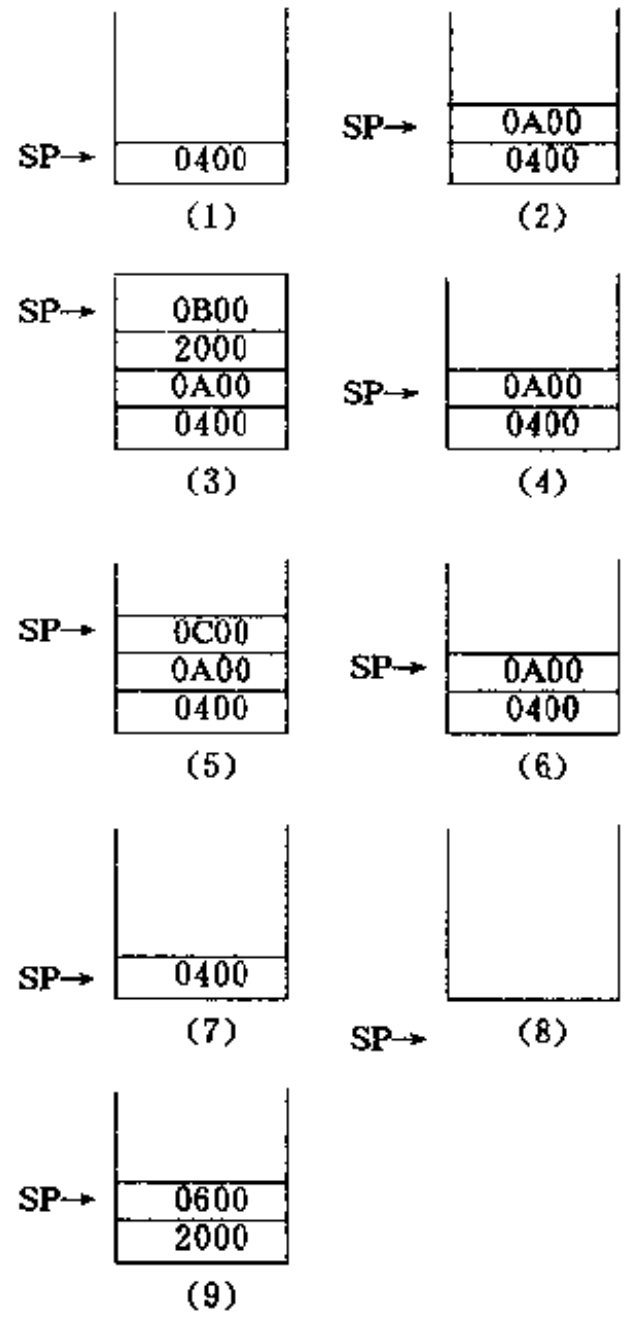
6.3

```
dseg        segment
            org     100h
            dw  100  dup  (?)
tos         label   word
ary         dw  100  dup  (?)
dseg        ends
cseg        segment
main        proc    far
  assume    cs:cseg, ds:dseg
  assume    ss:dseg, es:dseg
start:
            mov     ax,dseg
            add     ax,100h
            mov     ss,ax
            mov     sp,offset tos
            push    ds
            sub     ax,ax
            push    ax
            mov     ax,dseg
            mov     ds,ax
            mov     es,ax
            ...
            ret
main        endp
cseg        ends
            end     start
```

6.5



6.7



6.9

```
dseg        segment
testone     db?
testtwo     db?
dseg        ends
cseg        segment
main        proc    far
assume      cs:cseg,ds:dseg,es:dseg
```

```
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     es,ax
begin:  mov     al,testone
        cmp     al,testtwo
        je      callsame
        call    notsame
        jmp     continx
callsame:
        call    allsame
continx: ret
main    endp
allsame proc    near
        mov     dl,'#'
        mov     ah,2
        int     21h
        ret
allsame endp
notsame proc    near
        mov     dl,'!'
        mov     ah,2
        int     21h
        ret
notsame endp
cseg    ends
        end     start
```

6.11

```
dseg    segment
prompt1 db  'The quantity of parts: $'
prompt2 db  'The value of parts: $'
crif    db  0dh,0ah,'$'
quantity db 8,?,8 dup (' ')
value   db  8,?,8 dup (' ')
qty     dw  ?
price   dw  ?
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
begin:  lea     dx,prompt1
        mov     ah,09h
        int     21h
        lea     dx,quantity
        mov     ah,0ah
        int     21h
        lea     dx,crlf
        mov     ah,09h
        int     21h
        lea     dx,prompt2
        mov     ah,09
        int     21h
        lea     dx,value
        mov     ah,0ah
        int     21h
        lea     dx,crif
        mov     ah,09
        int     21h
        call    subconv
        call    subcalc
        call    subdisp
        ret
main    endp
subconv proc    near
        push    ax
        push    bx
        push    cx
        push    si
        push    di
        lea     si,quantity
        mov     al,[si+1]
        cbw
        mov     cx,ax
        call    decbin
        mov     qty,bx
        lea     si,value
        mov     al,[si+1]
        cbw
        mov     cx,ax
        call    decbin
        mov     price,bx
```

```
            pop     di
            pop     si
            pop     cx
            pop     bx
            pop     ax
            ret
subconv     endp
decbin      proc    near
            mov     bx,0
digit:      mov     al, [si+2]
            inc     si
            sub     al,30h
            cbw
            xchg    ax,bx
            mov     di,10
            mul     di
            xchg    ax,bx
            add     bx, ax
            loop    digit
            ret
decbin      endp
subcalc     proc    near
            push    ax
            push    dx
            mov     ax,price
            cwd
            div     qty
            mov     price,ax
            pop     dx
            pop     ax
            ret
subcalc     endp
subdisp     proc    near
            push    ax
            push    bx
            mov     ax, price
            mov     bx, 10000
            call    bindec
            mov     bx,1000
            call    bindec
            mov     bx,100
            call    bindec
            mov     bx,10
            call    bindec
            mov     dl,al
            or      di,30h
            mov     ah,02
            int     21h
            pop     bx
            pop     ax
            ret
subdisp     endp
bindec      proc    near
            mov     dx,0
a10:        cmp     ax,bx
            jl      a20
            inc     dl
            sub     ax,bx
            jmp     a10
a20:        push    ax
            or      dl,30h
            mov     ah,02
            int     21h
            pop     ax
            ret
bindec      endp
cseg        ends
            end     start
```

6.13

```
stack       segment
            dw  64  dup  (?)
  tos       label   word
stack       ends
dseg        segment
data        label   word
  data1     dd      ?
  data2     dd      ?
result      label   word
  result1   dd  0
  result2   dd  0
            dd  0
dseg        ends
cseg        segment
main        proc    far
assume      cs:cseg,ds:dseg,ss:stack
start:      mov     ax,stack
            mov     ss,ax
            mov     sp,offset tos
```

```
        push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        push    data+2
        push    data
        push    data+6
        push    data+4
        call    madd
        call    mmul
        push    result
        push    result+2
        push    result+4
        push    result+6
        push    result+8
        push    result+10
        add     sp,14h
        ret
main    endp
madd    proc    near
        push    bp
        mov     bp,sp
        push    ax
        push    bx
        mov     ax, [bp+4]
        mov     bx, [bp+8]
        add     ax,bx
        mov     result,ax
        mov     ax, [bp+6]
        mov     bx, [bp+0ah]
        adc     ax,bx
        mov     result+2,ax
        pop     bx
        pop     ax
        pop     bp
        ret
madd    endp
;
mmul    proc    near
        push    bp
        mov     bp,sp
        push    ax
        push    dx
        mov     ax, [bp+4]
        mov     bx, [bp+8]
        mul     bx
        mov     result+4,ax
        mov     result+6,dx
        mov     ax, [bp+4]
        mov     bx, [bp+0ah]
        imul    bx
        add     result+6,ax
        adc     result+8,dx
        mov     ax, [bp+6]
        mov     bx, [bp+8]
        imul    bx
        add     result+6,ax
        adc     result+8,dx
        mov     ax, [bp+6]
        mov     bx, [bp+0ah]
        imul    bx
        add     result+8,ax
        adc     result+10,dx
        pop     dx
        pop     bx
        pop     ax
        pop     bp
        ret
mmul    endp
cseg    ends
        end     start
```

6.15

```
stack   segment
        dw  64  dup  (?)
stack   ends
dseg    segment
record  dw  76,69,84,90,73
        dw  88,99,63,100,80
s6      dw  0
s7      dw  0
s8      dw  0
s9      dw  0
s10     dw  0
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg
```

```
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     cx,10
        call    count
        ...
        ret
main    endp
count   proc    near
        mov     si,0
next:   mov     ax,record [si]
        mov     bx,10
        div     bl
        mov     bl,al
        cbw
        sub     bx,6
        sal     bx,1
        inc     s6 [bx]
        add     si,2
        loop    next
        ret
count   endp
cseg    ends
        end     start
```

6.17

```
dseg    segment
val1    dw  ?
val2    dw  ?
dseg    ends
cseg    segment
bando   proc    far
assume  cs:cseg, ds:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        push    val1
        push    val2
        call    pairs
        call    pairs
        ret
bando   endp
pairs   proc    near
        push    bp
        mov     bp,sp
        push    bx
        mov     bx, [bp+4]
        call    outbin
        mov     cx,8
space:  mov     dl,' '
        mov     ah,2
        int     21h
        loop    space
        call    outoct
        call    crlf
        pop     bx
        pop     bp
        ret     2
pairs   endp
outbin  proc    near
        push    bx
        mov     cx,16
onebit: rol     bx,1
        mov     dx,bx
        and     dx,1
        or      dx,30h
        mov     ah,2
        int     21h
        loop    onebit
        pop     bx
        ret
outbin  endp
outoct  proc    near
        mov     cx,5
        rol     bx,1
        mov     dx,bx
        and     dx,01
        or      dx,30h
        mov     ah,2
        int     21h
next:   push    cx
        mov     cl,3
        rol     bx,cl
        mov     dx,bx
        and     dx,7
```

```
        or      dx,30h
        mov     ah,2
        int     21h
        pop     cx
        loop    next
        ret
outoct  endp
crlf    proc    near
        mov     ah,2
        mov     dl,0dh
        int     21h
        mov     ah,2
        mov     dl,0ah
        int     21h
        ret
crlf    endp
cseg    ends
        end     start
```

6.19

```
dseg    segment
n_v     dw  ?
result  dw  ?
dseg    ends
sseg    segment
        dw  128 dup (0)
tos     label   word
sseg    ends
cseg1   segment
main    proc    far
assume  cs:cseg1,ds:dseg
assume  ss:sseg
start:  mov     ax,sseg
        mov     ss,ax
        mov     sp,offset tos
        push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     bx,offset result
        push    bx
        mov     bx,n_v
        push    bx
        call    far ptr fact
        ret
main    endp
cseg1   ends
cseg    segment
  frame struc
        save_bp  dw ?
        save_cs_ip  dw  2 dup  (?)
        n       dw  ?
        result_addr  dw  ?
frame   ends
assume  cs:cseg
fact    proc    far
        push    bp
        mov     bp,sp
        push    bx
        push    ax
        mov     bx,[bp].result_addr
        mov     ax,[bp].n
        cmp     ax,0
        je      done
        push    bx
        dec     ax
        push    ax
        call    far ptr fact
        mov     bx,[bp].result_addr
        mov     ax,[bx]
        mul     [bp].n
        jmp     short return
done:   mov     ax,1
return: mov     [bx],ax
        pop     ax
        pop     bx
        pop     bp
        ret     4
fact    endp
cseg    ends
        end     start
```

6.21

```
;--------binary tree---------
dseg    segment
;                   RS    LS
;                 ----------
tree    db      2,    1     ; N = 0
        db      4,    3     ;     1
```

```
                db      6,      5       ;       2
                db      0,      0       ;       3
                db      0,      0       ;       4
                db      8,      7       ;       5
                db      0,      0       ;       6
                db      0,      0       ;       7
                db      10,     9       ;       8
                db      12,     11      ;       9
                db      0,      0       ;       10
                db      0,      0       ;       11
                db      0,      0       ;       12
dseg            ends
;* * * * * * * * * * * * * * * * * * * * * * *
;*                  inorder  trversal                *
;* print  no.  of  node  and  highth  of  tree*
;* * * * * * * * * * * * * * * * * * * * * * *
cseg            segment
main            proc    far
assume          cs:cseg,ds:dseg
start:          push    ds                      ;for return
                sub     ax,ax
                push    ax
                mov     ax,dseg
                mov     ds,ax

                mov     si,1                    ;left index
                mov     dx,0                    ;max highth
                mov     cx,0                    ;init highth
                call    inorder
                mov     bx,0                    ;node 0
                call    output
                mov     si,0                    ;right index
                mov     cx,0                    ;init highth
                call    inorder
                call    crlf                    ; CR/LF
                mov     bx,dx
                call    output                  ;print highth
                ret
main            endp
;-----------------------------------
inorder         proc    near            ;       inorder
traversal
                shl     bx,1
                mov     bl,tree [bx+si]
                mov     bh,0
                cmp     bx,0
                je      retn
                push    bx
                inc     cx                      ;inc highth
                push    cx
                cmp     dx,cx
                jge     nextl
                mov     dx,cx                   ;save max H
nextl:          mov     si,1                    ;left index
                call    inorder
                pop     cx
                pop     bx
                call    output
                mov     si,0                    ;right index
                call    inorder
retn:           ret
inorder         endp
;-----------------------------------
output          proc    near                    ;print node
                push    bx
                push    cx
                push    dx
                mov     ax,bx
                mov     cl,10d
                div     cl
                mov     dl,al
                add     dl,30h
                call    disp
                mov     dl,ah
                add     dl,30h
                call    disp
                mov     dl,20h
                call    disp
                pop     dx
                pop     cx
                pop     bx
                ret
output          endp
;
disp            proc    near
                push    ax
                mov     ah,2
                int     21h
```

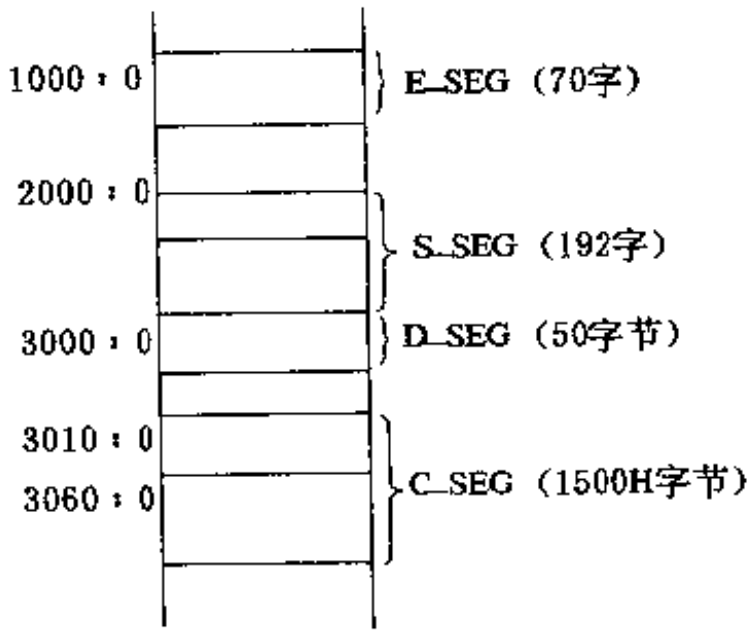
```
        pop     ax
        ret
disp    endp
;-------------------------------
crlf    proc    near
        push    dx
        mov     dl,0dh
        call    disp
        mov     dl,0ah
        call    disp
        pop     dx
        ret
crlf    endp
;-------------------------------
cseg    ends
        end     start
```

6.23

(1) students namelist< >

```
(2) mov     ah,0ah
    lea     dx,students
    int     21h
    mov     al,students.actlen
    mov     dispfile,al
```

6.25



6.27

(1) EXTRN SUBPRO

(2) PUBLIC SUBPRO

7.1

```
move    macro   to,from,n
        lea     si,from
        lea     di,to
        mov     cx,n
        rep     movsb
        endm
```

7.3

```
clrb    macro   n,cfil
        mov     cx,n
        mov     al,' '
        lea     di,cfil
        rep     stosb
        endm
```

7.5

```
cls     macro
        mov     ah,6
        mov     al,0
        mov     ch,0
        mov     cl,0
        mov     dh,24
        mov     dl,79
        mov     bh,7
        int     10h
        endm
```

7.7

```
starter macro   csname,dsname,ssname
        assume  cs:csname,ds:dsname
        assume  ss:ssname,es:dsname
        push    ds
        sub     ax,ax
        push    ax
        mov     ax,dsname
        mov     ds,ax
        mov     es,ax
        endm
```

7.9

```
send    macro   schars,dchars
        local   next,exit
        push    ax
        push    si
        mov     si,0
next:   mov     ax,schars[si]
        mov     dchars[si],al
        cmp     al,24h
        jz      exit
        inc     si
        jmp     next
exit:
```

```
        endm
```

7.11

```
bin_sub   macro   result,a,b,count
          local   next_sub
          push    cx
          push    bx
          push    ax
          mov     cx,count
          mov     ax,a
          lea     bx,b
          clc
next_sub: sbb     ax, [bx]
          inc     bx
          loop    next
          mov     result,ax
          pop     ax
          pop     bx
          pop     cx
          endm
```

7.13

```
mov_b_w   macro   from, to,count,tag
          lea     si,from
          lea     di,to
          mov     cx,count
          rep     movs &tag
          endm
```

7.15

```
joe  macro a,j
     mary a,% j
j=j+1
     endm
mary macro x,k
x&k db  'message no. &k'
     endm
宏展开     i=0
           joe       text,i
+       text0 db  'message no. 0'
           joe       text,i
+       text1 db  'message no. 1'
           joe       text,i
+       text2 db  'message no. 2'
```

7.17

```
center macro prompt
  cls
  locate 12,30
  display prompt
  endm
```

7.19

```
store   macro k
        mov   tab+k,k
        endm
```

宏调用：i=0

```
        rept 7
        store % i
        i=i+1
        endm
```

7.21

```
parts   db  'pno. 1'
irp n,<2,3,4,5,6,7,8,9,10>
        db  'pno. &n'
endm
```

7.23

```
     move   macro x
ifidn <x>,<vt55>
     mov      terminal,0
else
     mov      terminal.1
endif
     endm
```

7.25

```
     mov   al,divd
ife  sign
     mov   ah,0
     div   scale
else
     cbw
     idiv  scale
endif
     mov   result,al
```

8.1

```
out       25h al,
```

8.3

```
          mov       di,0
          mov       cx,80
begin:    in        al,51h
          test      al,02h
          jz        begin
          in        al,50h
```

```
        mov     buff [di],al
        inc     di
        in      al,51h
        test    al,00111000b
        jnz     error
        loop    begin
        jmp     exit
error:  call near ptr  err_rout
exit:   ...
```

8.5

```
message db      'buffer overflow'
        db      0dh,0ah
buffer  db      64 dup (?)
;       ...
        mov     di,offset buffer
        mov     cx,63
next:   in      al,54h
        test    al,02
        jz      next
        in      al,52h
        or      al,0
        jpe     no_error
        jmp near ptr error
no_error:
        and     al,7fh
        mov     [di],al
        inc     di
        cmp     al,0dh
        loopne  next
        jne     overflow
        mov byte ptr [di],0ah
        ret
overflow:
        mov     si,offset message
        mov     cx,17
out:    in      al,54h
        test    al,01
        jz      out
        mov     al, [si]
        inc     si
        out     53h,al
        loop    out
;       ...
        ret
```

8.7

在 0:50h,0:51h,0:52h,0:53h 四个字节中

8.9

```
...
push    ds
mov     ax,seg  int_rout
mov     ds,ax
mov     dx,offset  int_rout
mov     al,09
mov     ah,25h
int     21h
pop     ds
...
```

8.11

用 DEBUG 的反汇编命令(-u),起始地址为 F000:FEA5

8.13

D3→D2→D4→D5→D1→D3→D5

8.15

利用定时器中断,编写类型为 1ch 的计时程序(参考例 8.1)

8.17

```
scode   db  7,5,9,1,3,6,8,1,2,4
buffer  db    10  dup  (?)
;       ...
        mov     si,0
        mov     cx,10
        lea     bx,scode
input:  mov     ah,01
        int     21h
        and     al,0fh
        xlat
        mov     buffer [si],al
        inc     si
        loop    input
        ret
```

8.19

```
        mov     dh,0bh
        mov     dl,07h
        mov     bh,0
        mov     ah,2
        int     10h
```

8.21

(1) mov ah,3

```
        mov   bh,0
        int   10h
(2)     mov   dh,18h
        mov   dl,0
        mov   bh,0
        int   10h
(3)     mov   ah,2
        mov   bh,0
        mov   dx,00
        int   10h
        mov   ah,9
        mov   al, 'M'
        mov   bh,0
        mov   bl,7
        mov   cx,1
        int   10h
```

8.23

```
message   db   'Try again,you have'
cont      db   '\n'
          db   'starfighters left. $'
;         ...
          add  cl,30h
          mov  cont,cl
          lea  dx,message
          mov  ah,9
          int  21h
```

8.25

```
message   db   'What is the date'
          db   '(mm/dd/yy)?',07,'$'
datafld   db   10,0,10  dup(' ')
;         ...
          mov  ah,09
          lea  dx,message
          int  21h
          mov  ah,0ah
          lea  dx,datafld
          int  21h
```

8.27

```
(1)     mov   ah,00
        mov   al,02
        int   10h
(2)     mov   dh,4
        mov   dl,0
        mov   ah,02
        int   10h
(3)     mov   ah,6
        mov   al,10
        mov   bh,07
        mov   cx,00
        mov   dx,184fh
        int   10h
(4)     mov   ah,09
        mov   al,'*'
        mov   bh,0
        mov   bl,87h
        mov   cx,10
        int   10h
```

8.29

```
input:  mov   ah,01
        int   21h
        cmp   al,0dh
        jne   input
        mov   ah,02
        mov   dh,24
        mov   dl,0
        int   10h
```

8.31

```
        mov   ah,00
        mov   al,04
        int   10h
        mov   ah,odh
        mov   cx,13
        mov   dx,12
        int   10h
```

8.33

```
dseg    segment
bird    db   76h,07
        db   0c4h,07
dseg    ends
cseg    segment
main    proc      far
assume  cs:cseg,ds:dseg
start:  push      ds
        sub       ax,ax
        push      ax
        mov       ax,dseg
        mov       ds,ax
        mov       ah,0fh
```

```
        int     10h
        mov     dh,20
        mov     dl,0
begin:  mov     si,2
        mov     cx,1
        lea     di,bird
disp:   cmp     dl,80
        jae     exit
        mov     ah,2
        int     10h
        mov     ah,9
        mov     al,[di]
        mov     bl,[di+1]
        int     10h
        call    diy
        mov     ah,9
        mov     al,' '
        mov     bl,07
        int     10h
        inc     dl
        add     di,2
        dec     si
        jnz     disp
        jmp     begin
exit:   ret
main    endp
diy     proc    near
        push    cx
        push    dx
        mov     dx,50
di1:    mov     cx,2801
di2:    loop    di2
        dec     dx
        jnz     di1
        pop     dx
        pop     cx
        ret
diy     endp
cseg    ends
        end     start
```

8.35

```
(1) mov     ah,0
    mov     al,0ch
    mov     dx,0
    int     17h
(2) address db oah, 'Address NO.'
    length1 equ $-address
    ...
    lea     si,address
    mov     cx,length1
b30: mov    ah,0
    mov     al,[si]
    mov     dx,0
    int     17h
    inc     si
    loop    b30
```

8.37

```
inarea      db  80
actlen      db  ?
buffer      db  80  dup  (?)
;           ...
input:      mov     ah,0ah
            mov     dx,offset inarea
            int     21h
            cmp     actlen,0
            jz      exit
            mov     bx,0
            mov     ch,0
            mov     cl,actlen
print:      mov     ah,5
            mov     dl,buffer[bx]
            int     21h
            inc     bx
            loop    print
            mov     ah,5
            mov     dl,0ah
            int     21h
            mov     dl,0dh
            int     21h
            jmp     input
exit:       ret
```

8.39

```
muslist     db      'A music1',0dh,0ah
            db      'B music2',0dh,0ah
            db      'C music3',0dh,0ah,'$'
;           ...
            mov     ah,09
            mov     dx,offset muslist
```

```
        int     21h
input:  mov     ah,01h
        int     21h
        cmp     al, '0'
        jz      exit
        cmp     al, 'A'
        jnz     b0
        call    music1
        jmp     input
b0:     cmp     al, 'B'
        jnz     c0
        call    music2
        jmp     input
c0:     cmp     al, 'C'
        jnz     input
        call    music3
        jmp     input
exit:   ret
```

8.41

```
(1) mov   ah,16h
    mov   dx,seg fcbaddr
    mov   ds,dx
    mov   dx,offset fcbaddr
    int   21h
(3) mov   ah,15h
    mov   dx,seg fcbaddr
    mov   ds,dx
    mov   dx,offset fcbaddr
    int   21h
(5) mov   ah,14h
    mov   dx,seg fcbaddr
    mov   ds,dx
    mov   dx,offset fcbaddr
    int   21h
```

8.43

```
dseg    segment
fcb     equ   5ch
org     6ah
recsz   dw    19
org     7ch
recno   db    0
org     80h
dta     db    21   dup   (?)
crlf    db    0dh,0ah,'$'
dseg    ends
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     dx,fcb
        mov     ah,0fh
        int     21h
        cmp     al,0
        jnz     error
        mov     recsz,21
        mov     recno,0
        lea     dx,dta
        mov     ah,1ah
        int     21h
readrec: mov    ah,14h
        mov     dx,fcb
        int     21h
        cmp     dta,1ah
        je      exit
        cmp     al,0
        jnz     error
        mov     cx,17
        mov     bx,0
partno: mov     dl,dta [bx]
        mov     ah,2
        int     21h
        inc     bx
        loop    partno
        mov si, word ptr dta [bx]
        call    bindec
        push    ds
        mov     dx,dseg
        mov     ds,dx
        mov dx, offset crlf
        mov     ah,09
        int     21h
        pop     ds
        jmp     readrec
error:  mov     dl, '?'
        mov     ah,2
        int     21h
```

```
exit:   ret
bindec  proc    near
        push    cx
        mov     cx,10000
        call    dec_div
        mov     cx,1000
        call    dec_div
        mov     cx,100
        call    dec_div
        mov     cx,10
        call    dec_div
        mov     cx,1
        call    dec_div
        pop     cx
        ret
bindec  endp
dec_div proc    near
        mov     ax,si
        mov     dx,0
        div     cx
        mov     si,dx
        mov     dl,al
        add     dl,30h
        mov     ah,02
        int     21h
        ret
dec_div endp
main    endp
cseg    ends
        end     start
```

8.45

ds:7dh 5c 0a 00 00

8.47

```
lea     bx,fcbflsz
mov     ax,word ptr [bx]
add     bx,2
mov     dx,word ptr [bx]
div     fcbrcsz
```

8.49

02 and 06

8.51

```
(1) custhan  dw  ?
(2) mov  ah,3ch
    mov  cx,0
    lea  dx,path1
    int  21h
    jc   error
    mov  custhan,ax
    ...
(3) mov  ah,40h
    mov  bx,custhan
    mov  cx,128
    lea  dx,custout
    int    21h
    jc   error
    ...
(4) mov  ah,3eh
    mov  bx,custhan
    int  21h
    test ax,06
    jnz  error
    ...
```

8.53

在一个程序打开许多文件时

8.55

```
mov     ah,02
mov     al,01
lea     bx,indsk
mov     ch,06
mov     cl,03
mov     dh,00
mov     dl,00
int     13h
```

自 测 题 (一)

一、1. (AX)=3200,(SF,ZF,OF,CF)=(0,0,0,0)

2. (AX)=0E00H,(SF,ZF,OF,CF)=(0,0,0,0)

3. (31200H)=2000H,不影响条件码

4. (3FFFEH)=2000H,(SP)=0FFFEH,
不影响条件码

5. (31200H)=4FH,(SF,ZF,OF,CF)=(0,0,
0,/)

6. (31200H)=0FDB0H,
(SF,ZF,OF,CF)=1,0,0,1)

7. (31202H)=0FBH,(SF,ZF,OF,CF)=(1,0,
0,1)

8. (31203H)=21H,(SF,ZF,OF,CF)=(/,/,1,
1)

9. (DX)=121EH,(AX)=0E000H,

(OF,CF)=(1,1)

10. (AL)=38H,(AH)=80H,条件码无定义

二、1.(X) 2.(V) 3.(V) 4.(X) 5.(X) 6.(X)
7.(X) 8.(V) 9.(V) 10.(X)

三、(AX)=250

四、(1) JL (2) JG (3) JBE

五、a. 5 b. 4 c. 6 d. 5

六、
```
PRINTBK   MACRO   MESS,COUNT
    local   next,loop1
            mov cx,count
            mov bx,0
    next:   mov si,1
            mov dl,mess [bx]
            cmp dl,09h
               jne        loop1
               mov        dl,20h
               mov        si,8
    loop1:     mov        ah,5
               int        21h
               dec        si
               jne        loop1
               inc        bx
               loop       next
               ENDM
```

七、
```
code    segment
    assume  cs:code, ds:data, es:data
    start:  push    ds
            xor     ax,ax
            push    ax
            mov     ax,data
            mov     ds,ax
            mov     es,ax
            mov     si,offset array+7Eh
            mov     di,offset array+88h
            std
            mov     cx,60
            rep     movsw
            mov     bx,offset array+8h
            mov     cx,5
    next:   mov     word ptr [bx],0
            add     bx,2
            loop    next
            ret
code    ends
            end     start
```

自 测 题 (二)

一、1. (AX)=0F05FH,(SF,ZF,OF,CF)=(1,0,0,1)

2. (SF,ZF,OF,CF)=(1,0,1,1)

3. (AX)=0240H,(OF,CF)=(1,1)

4. (AX)=0906H,(SF,ZF)=(0,0)

5. (AX)=20ACH,条件码无定义

6. (AX)=0103H,(SF,ZF,OF,CF)=(0,0,/,0)

7. (AX)=0DF87H,(SF,ZF,OF,CF)=(1,0,0,0)

8. (22FFCH)——(23000H)=60H,(DI)=2FFBH,(CX)=0

9. (IP)=0A006H,不影响条件码

10. (AX)=00B0H,(25060H)=2060H,不影响条件码

二、1. 7E814H　2. 22636H, 2263CH

3. (a) 向高位无进位(或有借位) (b) 无符号

4. $ —PARTLIST

三、1. 0100

0100 54 4F 4D 2E 2E 14

0106 52 4F 53 45 2E 19

010C 4B 41 54 45 2E 16

2. (a) 第二条指令类型不匹配

(b) 第二条指令的源,目的都为存储器寻址

(c) OFFSET 操作符得不到确定值

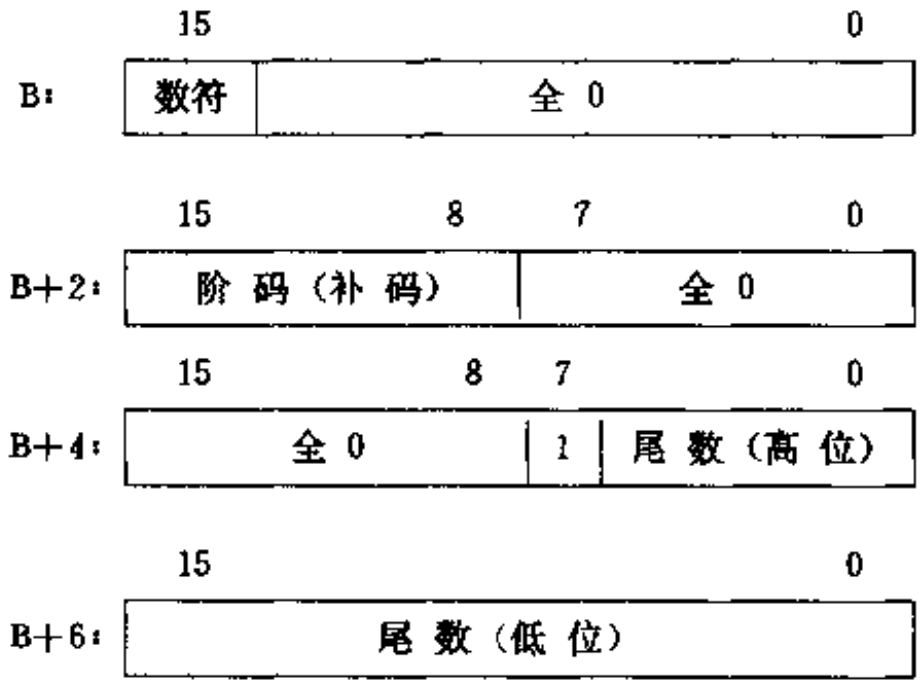
(d) (AL)=25

四、1.
```
biger   macro   x,y
        local   store
                mov     ax,x
                cmp     ax,y
                jge     store
                mov     ax,y
store:          mov     big,ax
                endm
```

2.
```
disp    macro   func,chadr
        mov     ah,func
        if      func-9
                mov     dl,chadr
        else
                lea     dx,chadr
        endif
        int     21h
        endm

+       disp    2, 0dh
+       mov     ah,2
+       mov     dl,0dh
+       int     21h
+       disp    2, 0ah
+       mov     ah,2
+       mov     dl,0ah
+       int     21h
+       disp    9, string
+       mov     ah,9
+       lea     dx,string
+       int     21h
```

五、



六、 <子 模 块>
```
public      get_name,openf,recno,readf
dseg        segment common
  fcbdriv       db      0
  fcbname       db      8  dup  (20h)
  fcbext        db      3  dup  (20h)
  fcbblk        dw      0
  fcbrecsz      dw      0
                db      16  dup  (?)
  fcbcurec      db      0
  fcbrarec      dw      0
                dw      0
  recfld        db      22  dup  (20h)
                db      0  dh,0ah,'$'
  maxlen        db      12
  actlen        db      ?
  kbuffer       db      12  dup  (20h)
  ermsg         db      0dh,0ah,'error!  $'
dseg        ends
;-------------------------------
cseg1       segment 'code'
assume      cs:cseg1,ds:dseg,es:dseg
get_name    proc    far
            cld
            mov     ch,0
            mov     cl,actlen
```

```
        sub     cx,4
        mov     si,offset kbuffer
        mov     di,offset fcbname
        rep     movsb
        std
        mov     si,offset kbuffer
        mov     cl,actlen
        mov     ch,0
        add     si,cx
        dec     si
        mov     di,offset fcbext
        add     di,2
        mov     cx,3
        rep     movsb
        ret
get_name endp
;-----------------------
openf   proc    far
        mov     ah,0fh
        lea     dx,fcbdriv
        int     21h
        cmp     al,0
        jnz     error
        mov     fcbrecsz,22
        mov     ah,1ah
        lea     dx,recfld
        int     21h
        ret
error:  lea     dx,ermsg
        mov     ah,0ah
        int     21h
        ret
openf   endp
;-----------------------
recno   proc    far
        cmp     actlen,1
        ja      dd2
        sub     ah,ah
        mov     al,kbuffer
        jmp     asc_bin
dd2:    ;temp
        mov     ah,kbuffer
        mov     al,kbuffer+1
asc_bin:
        and     ax,0f0fh
        aad
        mov     fcbrarec,ax
        ret
recno   endp
;-----------------------
readf   proc    far
        mov     an,21h
        lea     dx,fcbdriv
        int     21h
        cmp     al,0
        jnz     erm
        cmp     recfld. 1ah
        jnz     retn
erm:    lea     dx,ermsg
        mov     ah,0ah
        int     21h
retn:   ret
readf   endp
;-----------------------
cseg1   ends
        end
```

七、

```
;-----------------------
dseg    segment
count   dw  0
message db  'entrned 10 times',0dh,0ah,'$'
dseg    ends
;-----------------------
cseg    segment
main    proc    far
assume  cs:cseg,ds:dseg
start:  push    ds
        sub     ax,ax
        push    ax
        mov     ax,dseg
        mov     ds,ax
        mov     al,09h
        mov     ah,35h
        int     21h
        push    es
        push    bx
        push    ds
        mov     dx,offset kbint
```

39101

```
                mov     ax,seg kbint
                mov     ds,ax
                mov     al,09h
                mov     ah,25h
                int     21h
                pop     ds
                in      al,21h
                and     al,11111101b
                out     21h,al
                mov     count,10
                sti
                mov     di,2000
delay:          mov     si,3000
delay1:         dec     si
                jnz     delay1
                dec     di
                jnz     delay
                pop     dx
                pop     ds
                mov     al,09h
                mov     ah,25h
                int     21h
                ret
main            endp
;-------------------------------
kbint           proc    far
                push    ax
                in      al,60h
                push    ax
                in      al,61h
                mov     ah,al
                or      al,80h
                out     61h,al
                xchg    ah,al
                out     61h,al
                pop     ax
                test    al,80h
                jnz     return
                dec     count
                jnz     return
                call    bel_dsp
return:         cli
                mov     al,20h
                out     20h,al
                pop     ax
                iret
kbint           endp
;-------------------------------
bel_dsp         proc    near
                push    ax
                push    dx
                mov     di,07
                mov     ah,02
                int     21h
                mov     dx,offset message
                mov     ah,09h
                int     21h
                mov     count,10
                pop     dx
                pop     ax
                ret
bel_dsp         endp
;-------------------------------
cseg            ends
                end     start
```

08251

[General Information]
书名=IBM PC汇编语言程序设计例题习题集
作者=
页数=118
SS号=10280282
出版日期=