

## 第一章：80X86计算机组织结构

声明：本文件只能用于交流学习，在此基础上允许共享资源，相关图片来自于汇编语言PPT

1. 中央处理机
  - a. 总线接口部件Bus Interface Unit
    - i. 段寄存器 CS, DS, ES, SS
    - ii. 指针寄存器 IP
    - iii. 地址加法器
    - iv. 指令队列
    - v. 总线接口控制电路
  - b. 执行部件 Execution Unit
    - i. 通用寄存器 AX, BX, CX, DX, SI, DI, SP, BP
    - ii. 标志寄存器
    - iii. 算术逻辑部件
    - iv. 执行部件控制电路
2. 80X86的课件寄存器组
  - a. 通用寄存器组
    - i. 数据寄存器（用来保存操作数或者运算结果等）
      - 1) Accumulator 累加器，用于算数、逻辑运算以及与外设传送信息等
      - 2) Base 基址，常用于存放存储器地址
      - 3) Count 计数器，一般用于循环或者串操作等指令中的隐含寄存器
      - 4) Data 数据，常用来存放双字数据的高十六位，也可存放外设端口地址
    - ii. 变址寄存器（主要用于存放某个存储单元的偏移地址）
      - 1) Source Index 源变址寄存器，一般与DS联用，字符串操作中与DS连用来确定源操作数的地址
      - 2) Destination Index 目的变址寄存器，一般与DX连用，字符串操作中用来确定目的操作数的地址
    - iii. 指针寄存器
      - 1) Stack Pointer 堆栈指针寄存器，用于指示段顶的偏移地址
      - 2) Base Pointer 基址指针寄存器，用于堆栈段的基地址
  - b. 专用寄存器
    - i. Instruction Pointer 指令指针寄存器，存放下一次将从主存取出指令的偏移地址
    - ii. Program Status Word (FLAGS) 程序状态寄存器（标志寄存器）
      - 1) Carry Flag 进位标志，最高有效位进位或借位为1，多字节运算以及无符号数比较大小和移位操作时使用
      - 2) Zero Flag 零标志，运算结果为0则置1
      - 3) Sign Flag 符号标志，运算结果为负置1
      - 4) Parity Flag 奇偶标志，操作数低8位中1的个数为偶数（包括0）置1
      - 5) Overflow Flag 溢出标志，运算结果溢出置1
      - 6) Direction Flag 方向标志（控制标志位），设置DF=0，串操作的寄存器SI, DI地址自动增加，设置DF=1时则自动减少。用CLD指令复位，STD指

令置位

- 7) Interrupt-enable Flag 中断允许标志（系统标志位），设置IF=1，允许中断，设置IF=0，禁止中断。用CLI复位，STI置位

iii. 段寄存器

- 1) Code Segment 代码段
- 2) Data Segment 数据段
- 3) Stack Segment 堆栈段
- 4) Extra Segment 附加段

访存类型	所用段及寄存器	缺省选择规则
指令	代码段：CS	用于取指令
堆栈	堆栈段：SS	进栈或出栈，用SP、BP作为基址寄存器的访存
局部数据	数据段：DS	除堆栈或串处理操作以外的所有数据访问
目的串	附加段：ES	串处理指令的目的串

3. 存储器

a. 总览

- i. Physical Address 物理地址= 段地址：Effective Address有效地址（Offset 偏移地址）
- ii. 段长最少为16bytes，最长为64KBytes

## 第二章：80X86寻址方式 addressing mode

1. 概念

a. 操作数

- i. 立即数操作数 2020H
- ii. 寄存器操作数 AX
- iii. 存储器操作数（内存操作数） [2020H]

b. 有效地址（EA）组成

- i. 位移量（displacement）
- ii. 基址（base）BX BP
- iii. 变址（index）SI DI

2. 与数据有关的寻址方式

- a. 立即寻址 MOV AX,3069H
- b. 寄存器寻址 MOV AX,BX（快）  
以上两种不用去内存取数
- c. 直接寻址 MOV AX,[2000H]
- d. 寄存器间接寻址 MOV AX,[BX]（可用寄存器，BX，BP，SI，DI）
- e. 寄存器相对寻址 MOV AX, COUNT[BX]（可用如上）  
使用BP时默认段为SS，其它为DS
- f. 基址变址寻址 MOV AX, [BP] [DI]（可用BX/BP+SI/DI）
- g. 相对基址变址寻址 MOV AX, MASK [BX] [SI]
- h. 段跨越不允许情况
  - i. 指令必须在代码段中
  - ii. PUSH与POP必须使用SS
  - iii. 串处理必须使用ES

? ★ 3. 与转移地址有关的寻址方式

- a. 段内直接寻址 JMP NEAR PTR NEXT（位移量）
  - i. 八位短转 SHORT（可正可负）
  - ii. 十六位近转 NEAR PTR（可正可负）只能用于无条件转移
- b. 段内间接寻址 JMP（word ptr 可省）TABLE [BX]
  - i. 得到的有效地址直接代替IP  
段内只改变IP，段间还要改变CS
- c. 段间直接寻址 JMP FAR PTR NEXT
  - i. 用地址NEXT的CS与IP值直接代替现有的CS与IP
- d. 段间间接寻址 JMP DWORD PTR [BX]
  - i. 将目的位置的双字作为转移地址（先IP后CS）

## 第三章：汇编语言伪指令

## 1. 段定义

STACK\_NAME SEGMENT 【定位类型】 【组合类型】 【使用类型】 【 '类别' 】

- a. 定位类型：确定逻辑段的边界在存储器的位置（需要将本程序与其它程序相连时使用）
  - i. PAGE：表示相应段必须从某一页的边界开始（256倍数）
  - ii. PARA：表示相应段必须从某一节的边界开始（16倍数）
  - iii. DWORD：表示相应段必须从某一节的边界开始（4倍数）
  - iv. WORD：表示相应段必须从某一节的边界开始（2倍数）
  - v. BYTE：任意  
默认为**PARA**
- b. 组合类型：表示该段与其它同名段之间的组合连接方法（同样应用于多程序）
  - i. PRIVATE(MEMORY)：私有段，不合并
  - ii. PUBLIC：同名段合并为一个段，连接顺序由命令指定
  - iii. COMMON：同名段共享同存储空间，产生覆盖，后面一段覆盖前面一段
  - iv. STACK：同名段组合为堆栈段，长度为原有段之和  
默认为**PRIVATE**
- c. 使用类型：适用于386之后机型，USE16（默认）表示十六位寻址，USE32
- d. 类名：必须用单引号括起来。连接时将同类名的各段在物理地址上连接在一起但不合并

2. ASSUME伪指令：段定义之后还需要明确段与段寄存器之间的关系，但是并不代表真正的装入了段寄存器中，在代码段的开始需要先进行DS\ES\SS段的基址的装填。

## 3. MODEL伪指令

.MODEL 存储模式 【, 语言类型】 【, 操作系统类型】 【, 堆栈选项】

- a. 存储模式：TINY, SMALL, COMPACT, MEDIUM, LARGE, HUGE, FLAT
- b. 语言类型：C, BASIC, PASCAL等
- c. 操作系统：OS\_DOS(默认), OS\_OS2
- d. 堆栈选项：NEARSTACK, FARSTACK

## 4. .SEGMENTNAME 简化段定义伪指令

**.MODEL** SMALL

.DATA

      ;此处输入数据段代码

.STACK

      ;此处输入堆栈段代码

.CODE

START:

    MOV AX, @DATA

    MOV DS, AX

      ;此处输入代码段代码

    MOV AH, 4CH

    INT 21H

    END START

## 5. 数据定义以及存储器分配的伪操作

DB DW DD DF DQ DT

## 6. 复制操作符 repeat\_count DUP optr\_num1,2,3（可嵌套）

## 7. PTR伪操作：Type PTR 变量或常量

MOV AX, WORD PTR OPER1+1

## 8. LABEL伪操作：name LABEL type

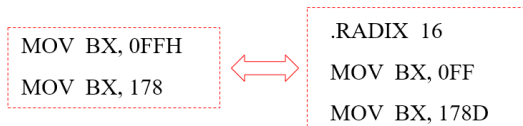
BYTE\_ARRAY LABEL BYTE

## 9. 表达式赋值伪操作 EQU：给表达式赋予名字，之后均可用名字替代该表达式时

表达式名 EQU 表达式

## 10. 地址计数器 \$：保存当前正在汇编的指令的地址

## 11. 基数控制伪操作 .RADIX



## 12. 表达式操作符

- 算数操作符+、-、\*/Mod：可用于数字表达式或地址表达式（地址表达式需要有明确的物理意义）
- 关系操作符EQ,NE,LT,LE,GT,GE：真为0FFFFH,假为0000H
- 逻辑与移位操作符AND,OR,XOR,NOT：
- 数值回送操作符TYPE,LENGTH,SIZE,OFFSET,SEG：
  - TYPE：返回变量以字节数表示的类型  
DB 1 DW 2 DD 4 DQ 8 DT 10 NEAR -1 FAR -2 CONST 0
  - LENGTH：对于DUP返回Repeat\_count，否则为1
  - SIZE：返回分配给该变量的字节数，等于TYPE\*LENGTH
  - OFFSET：回送变量或者标号的偏移地址
  - SEG：回送变量或者标号的段地址
- 属性标识符
  - SHORT：修饰词，表示跳转指令中转向地址在-127~127字节范围之内
  - HIGH,LOW：对数或者表达式字节分离，HIGH取高位，LOW取低位
  - HIGHWORD,LOWWORD：字分离表达式，其它同上

## 第四章：汇编语言指令操作

### 1. 概述

- 操作数类型中若指令中无类型依据，则必须要对存储器操作数加以说明
- 寄存器操作数、立即数操作数、存储器操作数指令速度依次降低

### 2. 数据传送指令

- 通用数据传送指令MOV,PUSH,POP,XCHG
  - 操作数类型相同
  - 目的操作数和源操作数不能同时为段寄存器



- 立即数不能直接传送给段寄存器
  - 目的操作数不能为CS，操作数不能为IP
  - 不影响标志位
  - POPA中赋值给SP的内容被丢弃
  - XCHG中两个操作数必须有一个为寄存器且不能用段寄存器
- 累加器专用传送指令IN,OUT,XLAT（只限用AX,AL传送信息）
    - 长格式（地址的内容）：
 

```
IN AL,PORT (AL<-(PORT)) (PORT为立即数)
IN AX,PORT (AX<-(PORT+1,PORT))
```
    - 短格式（存储器存放的地址的内容）：
 

```
IN AL,DX(AL<-((DX)))
IN AX,DX(AX<-((DX)+1,(DX)))
```
    - OUT相同，只是赋值方向相反
    - 所有的IO端口与CPU之间的通信都由IN与OUT指令完成
    - 外部设备最多有 $2^{16}$ 个端口，端口号为0000~0FFFFH
    - 前256个端口（00~0FFH）可以直接在指令中指定，这叫做长格式
    - 大于256号端口只能使用短格式，此时需要先将端口号存放到DX
    - 注意端口号或者是DX的内容都是地址，而传送的内容是端口中的信息使用短格式的时候，DX内容就是端口号

本身

ix. 指令不影响标志位

x. 换码指令XLAT(或XLAT PTR)，执行操作 $(AL) \leftarrow ((BX) + (AL))$

意义：(BX) 表示字节表格首地址，(AL) 表示位移量

MOV 1 MOV BX, OFFSET TABLE; (BX)=0040H

MOV 1 MOV AL, 3

XLAT XLAT TABLE

指令执 指令执行后 (AL)=33H

(DS)=F000H		
TABLE		
(BX) →	30 H	F0040
	31 H	F0041
(AL) = 3	32 H	F0042
	33 H	F0043

c. 地址传送指令LEA,LDS,LES

i. LEA是指令，而OFFSET是伪指令

ii. LDS REG, SRC 相继两字先后存在REG与DS (LES则是REG与ES)

d. 标志寄存器传送指令LAHF,SAHF,PUSHF,POPF

i. LAHF: load AH with Flags(low)

ii. SAHF: save AH into Flags(low)

iii. PUSHF: push Flags(all)

iv. POPF: pop Flags(all)

e. 类型转换指令CBW,CWD

i. CBW: AL拓展到AH

ii. CWD: AX拓展到 (AX,DX)

iii. 拓展是带符号的

### 3. 算术指令

a. 加法指令ADD,ADC,INC

i. ADC带进位的加法 (+CF)

双精度运算ADD AX,X; ADC DX,X+2;

ii. INC不影响标志位

iii. 标志位的影响

1) SF=1 结果为负

2) ZF=1 结果为0

3) CF=1 和的最高有效位向高位进位 (表示无符号数相加的溢出)

4) OF=1 两个操作数符号相同但是结果与之相反 (表示带符号数相加的溢出)

b. 减法指令SUB,SBB,DEC,NEG,CMP

i. SBB带借位减法指令 (-CF)

双精度运算SUB AX,X; SBB DX,X+2;

ii. DEC不影响标志位

iii. NEG求补 (补码转换) NEG OPR: (OPR)  $\leftarrow$  -0FFFFH-(OPR)+1

iv. SUB标志位的影响

1) CF=1 被减数的最高有效位向高位借位 (表示无符号数减法的溢出)

2) OF=1 两个操作数符号相反，而结果的符号与减数相同 (表示带符号数减法的溢出)

v. NEG标志位的影响

1) CF=0 操作数为0

2) OF=1 操作数为-128 (字节运算) 或-32768 (字运算)

c. 乘法指令MUL,IMUL

i. IMUL为带符号的乘法

ii. AL,AX为隐含的乘数寄存器

iii. AX,(DX,AX)为隐含的乘积寄存器

iv. SRC不能为立即数

v. 只影响CF,OF标志位

MUL指令: CF,OF =  $\begin{cases} 00 & \text{乘积的高一半为零} \\ 11 & \text{否则} \end{cases}$

IMUL指令: CF,OF =  $\begin{cases} 00 & \text{乘积的高一半是低一半的符号扩展} \\ 11 & \text{否则} \end{cases}$

d. 除法指令DIV,IDIV

- IDIV带符号数的除法指令（余数的符号和被除数的符号相同）
- AX,(DX,AX)为隐含的被除数寄存器
- AL,AX为隐含的商寄存器
- SRC不能为立即数
- 对所有的标志位均无定义

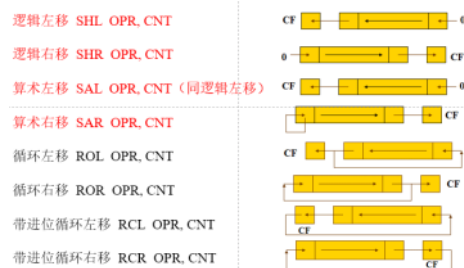
4. 逻辑指令

a. 逻辑运算指令 AND,OR,NOT,XOR,TEST

- NOT的OPR不能为立即数，且NOT不影响标志位
- CF=0 OF=0 SF,ZF,PF根据运算结果设置 AF无定义

b. 移位指令和循环移位指令SHL,SHR,SAL,SAR,ROL,ROR,RCL,RCR

- SHL逻辑左移，最高位移至进位标志，最低位0补齐
- SHR逻辑右移，最高位0补齐，最低位移至进位标志
- SAL算术左移，最高位移至进位标志，最低位0补齐
- SAR算术右移，最高位不变，最低位移至进位标志  
(算数移位指令用作带符号数的移位，逻辑移位用作无符号数)
- ROL循环左移，最高位移至最低位，同时赋值给进位标志
- ROR循环右移，最低位移至最高位，同时赋值给进位标志
- RCL带进位循环左移，最高位移至进位标志，原进位标志移至最低位
- RCR带进位循环右移，最低位移至进位标志，原进位标志移至最高位



ix. 注意

- OPR可用除了立即数之外的任何寻址方式
- CNT=1时可用立即数，大于1时只能为CL
- OF=1 表示CNT=1时最高有效位的值发生变化
- OF=0表示CNT=1时最高有效位的值不发生变化
- SF,ZF,PF根据移位结果设置，AF无定义
- 但是循环移位指令不影响SF,ZF,PF,AF

5. 串处理指令

a. 设置方向标志指令CLD,STD

b. 串处理指令MOVS,STOS,LODS,CMPS,SCAS

- 串传送: MOVS
- 存入串: STOS

```
lea di, mess2
mov al, 20H
mov cx, 10
cld
rep stosb
```

iii. 取出串: LODS

iv. 串比较: CMPS

```

lea si, mess1
lea di, mess2
mov cx, 8
cld
repe cmpsb

```

- v. 串扫描: SCAS
- c. 串重复前缀 REP, REPE/REPZ, REPNE/REPNZ
  - i. 重复: REP
    - 1) 操作: 重复操作直到 CX 为 0
  - ii. 相等/为零重复: REPE/REPZ
  - iii. 不相等/不为零重复: REPNE/REPNZ
- 6. 处理机控制与杂项操作指令
  - a. 标志处理指令
  - b. 其它

## 第五章: 循环与分支程序设计

概要: 首先确定数据结构和算法(关键), 画出流程图(减少出错), 上机, 调试

### 1. 循环转移指令

- a. 无条件转移 JMP
  - i. 段内短转(位移量) JMP SHORT OPR
  - ii. 段内近转(位移量) JMP NEAR PTR OPR
  - iii. 段内间接转移(直接量) JMP WORD PTR OPR
  - iv. 段间直接远转移(调至目的 OPR) JMP FAR PTR OPR
  - v. 段间间接转移(通过 OPR 内容跳转) JMP DWORD PTR OPR
- b. JZ/JE(ZF=1) 结果为 0 转移
- c. JNZ/JNE
- d. JS(SF=1) 结果为负转移
- e. JNS
- f. JO(OF=1) 结果溢出则转移
- g. JNO
- h. JP/JPE(PF=1) 1 的个数为偶数
- i. JNP/JPO
 

注意 jklm 用于无符号数的比较, 分别代表  $< \geq > \leq$
- j. JB/JNAE/JC(CF=1) 低于则转移(below)
- k. JNB/JAE/JNC
- l. JA/JNBE(CF  $\vee$  ZF = 1) 高于则转移(above)
- m. JNA/JBE
 

注意 nopq 用于带符号数的比较, 分别代表  $< \geq \leq >$
- n. JL/JNGE(SF  $\vee$  OF = 1)
- o. JNL/JGE
- p. JLE/JNG((SF  $\vee$  OF)  $\vee$  ZF = 1)
- q. JNLE/JG
- r. JCXZ((CX)=0) 测试 CX 的值为 0 则转移
- s. LOOP(CX  $\neq$  0)
- t. LOOPE/LOOPZ(CX  $\neq$  0  $\wedge$  ZF = 1)
- u. LOOPNE/LOOPNZ

### 2. 循环结构程序设计

循环初始化(初始): 设立地址指针、循环次数、数据保护、初始状态

循环体

循环修改部分

循环控制部分（重点）

a. 例题1.

【例5.1】试编制一个程序把 BX 寄存器内的二进制数用十六进制数的形式在屏幕上显示出来。

b. 例题2

【例5.7】有一个首地址为A的N字数组，编写程序使该数组中的数按照从大到小的次序整序。  
(冒泡算法，多重循环)

c. 例题3

【例5.5】有数组 x(x1,x2,...,x10) 和 y(y1,y2,...,y10)，编程计算 z(z1,z2,...,z10)

z1 = x1 + y1  
z2 = x2 + y2  
z3 = x3 - y3  
z4 = x4 - y4  
z5 = x5 - y5  
z6 = x6 + y6  
z7 = x7 - y7  
z8 = x8 - y8  
z9 = x9 + y9  
z10 = x10 + y10

逻辑尺：0 0 1 1 1 0 1 1 0 0

1 减法

0 加法

3. 分支程序结构设计

a. IF THEN结构转化

CMP AX,BX

JE EndOfIf

<THEN 程序段>

EndOfIf:

b. IF-THEN-ELSE结构

CMP AX,BX

JE ElseCode

<THEN 程序段>

JMP EndOfIf

ElseCode:

<ELSE 程序段>

EndOfIf:

c. 例题1

例：已知在内存中有一个字节单元NUM，存有带符号数据，要求计算出它的绝对值后，放入RESULT单元中。

d. 例题2

【例5.2】在Y中存放着16位数，试编制一个程序把Y中1的个数存入COUNT单元中。

e. 例题3

【例5.6】从键盘输入一行字符，要求输入的第一个字符必须是空格，如果不是，则退出，如果是，则开始接受输入的字符并顺序存入首地址为BUFFER的缓冲区，直到接收到第二个空格为止。

f. 例题4

【例5.9】在数据段中，有一个按从小到大顺序排列的无符号数数组，其首地址存放在DI寄存器中，数组中的第一个字单元存放着数组长度。在AX中有一个无符号数，要求在数组中查找(AX)：

- 如找到，则使CF=0，并在SI中给出该元素在数组中的偏移地址
- 如未找到，则使CF=1，并使SI中存放最后一次比较的数组元素的偏移地址



**折半查找：**在一个长度为n的有序数组r中，查找元素k的折半查找算法可描述如下：

- (1) 初始化被查找数组的首尾下标， $low \leftarrow 1$ ， $high \leftarrow n$ ；
- (2) 若 $low > high$ ，则查找失败，置CF=1，退出程序。否则，计算中点 $mid \leftarrow [(low+high)/2]$ ；
- (3) k与中点元素r[mid]比较。若 $k=r[mid]$ ，则查找成功，程序结束；若 $k < r[mid]$ ，则转(4)；若 $k > r[mid]$ ，则转(5)；
- (4) 低半部分查找， $high \leftarrow mid-1$ ，返回(2)，继续查找；
- (5) 高半部分查找， $low \leftarrow mid+1$ ，返回(2)，继续查找。

#### g. 例题5

**跳跃表法：**设有若干段分支程序，将每段分支程序的入口地址（也称跳跃地址）组成一个连续存放在内存中的表，称为跳跃表。

**例：**设某程序有8路分支，试根据给定的N值（1~8），将程序的执行转移到其中的一路分支。

<pre> DATAS SEGMENT     num    db 78h     adtab  dw ad0,ad1,ad2,ad3,ad4,ad5,ad6,ad7 DATAS ENDS  CODES SEGMENT     ASSUME CS:CODES,DS:DATAS START:     MOV AX,DATAS     MOV DS,AX      mov al,num     mov dl,'?'     cmp al,0     jz  disp ;为0则跳转显示?     mov bx,0 ;循环计数初值 again: shr al,1 ;测试最低位是否为1     jc  next ;为1则跳转     inc bx     jmp again next:  shl bx,1 ;bx*2，计算偏移量     jmp adtab[bx] ;跳转到指定分支         </pre>	<pre> ad0:  mov dl,'0'       jmp disp ad1:  mov dl,'1'       jmp disp ad2:  mov dl,'2'       jmp disp ad3:  mov dl,'3'       jmp disp ad4:  mov dl,'4'       jmp disp ad5:  mov dl,'5'       jmp disp ad6:  mov dl,'6'       jmp disp ad7:  mov dl,'7'       jmp disp disp: mov ah,2       int 21h       mov ah,4ch       int 21h CODES ENDS END START         </pre>
--	---

**注意：**每个分支程序的最后要有一条转移语句，以便跳过其他的分支。

#### h. 例题6

**转移表法：**把转移到各分支程序段的转移指令依次存放在一起，形成转移表。各转移指令在表中的位置——离表首地址的偏移量作为转移条件，偏移量加上表首地址作为转移地址，转到表的相应位置，执行相应的无条件转移指令。

<pre> CODE SEGMENT     ASSUME CS:CODE START:LEA BX,TAB     MOV AH,1     INT 21H     SUB AL,30H     MOV AH,0     ADD AX,AX     ADD BX,AX     JMP BX      TAB:JMP SHORT MODE0 ; 转移表         JMP SHORT MODE1         JMP SHORT MODE2         JMP SHORT MODE3         JMP SHORT MODE4         </pre>	<pre> MODE0:MOV DL,30H         JMP EXIT MODE1:MOV DL,31H         JMP EXIT MODE2:MOV DL,32H         JMP EXIT MODE3:MOV DL,33H         JMP EXIT MODE4:MOV DL,34H EXIT:MOV AH,2       INT 21H       MOV AH,4CH       INT 21H CODE ENDS END START         </pre>
--	--

## 第六章：子程序结构

### 1. 子程序的引出与定义

#### a. 定义

Name PROC [NEAR|FAR]

...

RET

Name ENDP

#### b. NEAR 段内近调用：只能被相同代码段的其他程序调用

#### c. FAR 段间远调用：还可以被不同代码段的其他程序调用

#### d. 主程序通常定义为FAR，被看作DOS调用的一个子程序

### 2. 子程序的调用与返回

#### a. 调用机制

CALL NEAR PTR SUBP 先将IP入栈再转子程序

CALL FAR PTR SUBP 先将(CS,IP)入栈再转子程序

b. 返回机制 RET[VAL](VAL表示SP的偏移量)

段内返回 IP出栈 再有SP<-SP+VAL

段间返回 IP,CS先出栈, 再有SP<-SP+VAL

c. 保护机制

i. 堆栈保护

ii. 内存单元保护

d. 说明

i. NEAR比FAR调用更快

ii. CALL的本质相当于PUSH+JMP, RET的本质相当于POP+JMP

### 3. 子程序设计方法

a. 子程序说明文件

i. 子程序名

ii. 子程序功能

iii. 入口条件 (入口参数及其意义和存放位置)

iv. 出口条件

v. 受影响的寄存器

b. 模块内参数传递常用方法

i. 寄存器法

例题:

入口: CX存放被开方数

出口: CX存放开方后的数值

例: 编制程序计算

$$Y = \sqrt{2x} + \sqrt{3y} + \sqrt{150}$$

;子程序清单如下:

```
SQROOT PROC
    PUSH AX
    PUSH BX
    PUSH DX
    XOR BX, BX
    AND CX, CX
    JZ SQRT3
SQRT1:
    MOV AX, BX
    MUL BX
    CMP CX, AX
    JB SQRT2
    INC BX
    JMP SQRT1
```

SQRT2:

DEC BX

SQRT3:

MOV CX, BX

POP DX

POP BX

POP AX

RET

SQROOT ENDP

ii. 变量传参

例题:

入口: ARGX存放被开方数

出口: ROOT存放开方后的数值

;子程序清单如下:

```
SQROOT PROC
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    XOR BX, BX
    MOV CX, ARGX
    AND CX, CX
    JZ SQRT3
SQRT1:
    MOV AX, BX
    MUL BX
```

CMP CX, AX

JB SQRT2

INC BX

JMP SQRT1

SQRT2:

DEC BX

SQRT3:

MOV ROOT, BX

POP DX

POP CX

POP BX

POP AX

RET

SQROOT ENDP

iii. 堆栈法 (注意要明确堆栈的存储情况, 容易出错)

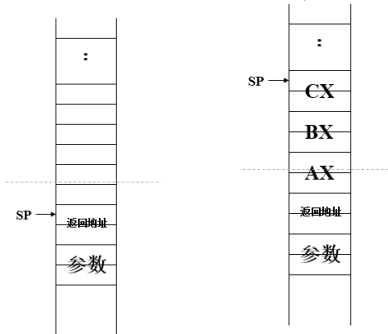
例题:

```
PUSH CX
CALL SQROOT
POP CX
```

;堆栈传递入口参数

;堆栈返回运算结果

;子程序清单如下:			
SQROOT	PROC		JB SQR2
	PUSH	AX	INC BX
	PUSH	BX	JMP SQR1
	PUSH	CX	
	ADD	SP, 8	SQR2:
	POP	CX	DEC BX
	XOR	BX, BX	
	AND	CX, CX	SQR3:
	JZ	SQR3	PUSH BX
SQR1:			SUB SP, 8
	MOV	AX, BX	POP CX
	MUL	BX	POP BX
	CMP	CX, AX	POP AX
			RET
			SQROOT ENDP



刚进入时堆栈的情况

执行完保护现场后堆栈的情况

技巧：用寻址代替POP，不改变SP的值

;子程序清单如下:			
SQROOT	PROC		JB SQR2
	PUSH	AX	INC BX
	PUSH	BX	JMP SQR1
	PUSH	CX	
	PUSH	BP	SQR2:
	MOV	BP, SP	DEC BX
	MOV	CX, [BP+10]	
	XOR	BX, BX	SQR3:
	AND	CX, CX	MOV [BP+10], BX
	JZ	SQR3	POP BP
SQR1:	MOV	AX, BX	POP CX
	MUL	BX	POP BX
	CMP	CX, AX	POP AX
			RET
			SQROOT ENDP

#### iv. 参数地址指针法

例题：

例：设内存有三组无符号字数据，三组数据的首地址分别为LIST1，LIST2和LIST3，数据个数分别存放在CNT1，CNT2和CNT3单元。编制程序计算三组数据中最小数之和并存入SUM开始的单元。

子程序说明文件如下：

- (1) 子程序名：FMIN
- (2) 子程序功能：求一组无符号字数据的最小值
- (3) 入口条件：数据首地址在SI中，元素个数在CX中
- (4) 出口条件：最小值在AX中
- (5) 受影响的寄存器：AX及标志寄存器。

;子程序清单如下:

```
FMIN PROC
    PUSH SI
    PUSH CX
    MOV AX, [SI]
    DEC CX
    JZ RETN ; 数组长度为1, 则结束
FMIN2: ADD SI, 2
    CMP AX, [SI]
    JB FMIN1
    MOV AX, [SI]
FMIN1: LOOP FMIN2
RETN: POP CX
    POP SI
    RET
FMIN ENDP
```

#### 4. 子程序嵌套

在子程序的内部再调用其它子程序

##### a. 例题1

有如下子程序说明文件:

- (1) 子程序名: **HTOA**;
- (2) 子程序功能: 将一位十六进制数转换为ASCII码;
- (3) 入口条件: 要转换的数据在AL中的低四位;
- (4) 出口条件: 十六进制数的ASCII码在AL中;
- (5) 受影响的寄存器: **AL**和标志寄存器。

子程序清单如下:

```
HTOA PROC
    AND AL, 0FH
    CMP AL, 10
    JC HTOA1
    ADD AL, 7
HTOA1: ADD AL, 30H
    RET
HTOA ENDP
```

0~9: 30H~39H, 差值为30H  
A~F: 41H~46H, 差值为37H

#### 5. 递归子程序

##### a. 递归性质

- i. 结束条件
- ii. 递归条件

##### b. 注意递归子程序必须采用寄存器或堆栈进行参数的传递

##### ★c. 例题1: 阶乘

#### ★6. 程序的连接

##### a. 模块化标识符

- i. 局部标识符: 在本模块定义在本模块使用
- ii. 外部标识符: 在一个模块中定义在另一个模块中引用该标识符

##### b. 模块相互调用原则

###### i. 声明共用的变量和过程

- 1) PUBLIC name[name2,name3,...]定义标识符的模块使用
- 2) EXTRN name: category[,name2: category2,...] 调用标识符的模块使用
- 3) 标识符name: 变量名、标号、过程名等
- 4) 类型category: byte/word/dword (变量) 或 near/far (过程)
- 5) 在一个源程序之中可以出现多个该语句
- 6) 各模块声明应相互配对

###### ii. 实现正确的段组合

###### iii. 处理好参数的传递问题

##### c. 模块连接方法

###### i. INCLUDE filename 该方法在汇编 (masm) 时将模块进行连接

- 1) 本质上只是一个文件分开书写, 不需要extrn和public
- 2) 所有的文件必须一同汇编

- 3) 标识符必须统一规定不能重复
- 4) 分开文件会增加汇编时间
- ii. 分开汇编一同连接 LINK main.obj proc1.obj proc2.obj...
  - 1) 需要声明EXTRN和PUBLIC
  - 2) 其它程序相当于子程序库
- d. 例题

子程序补充例题:

- ◆ 利用子程序方法将键盘输入的一组带符号十进制字数据存储在缓存中。
- ◆ 子程序从键盘输入一个有符号十进制数;子程序还包含将ASCII码转换为二进制数的过程。
- ◆ 输入时,负数用“-”引导,正数直接输入或用“+”引导。
- ◆ 子程序用寄存器传递出口参数,主程序调用该子程序输入10个数据。

## 第七章:高级汇编语言技术

### 1. 宏汇编

MACRO 【形参表】

...  
ENDM

- a. 实参和形参的个数可以不等,实参多余的将会被忽略,形参多余的置为NULL
- b. 宏定义必须在宏调用之前,特别是宏调用中含有另一个宏时
- c. 形参可以是操作码
- d. 与宏有关的操作符

- i. 连接操作符& (连接前后符号构成新符号)

```
Leap      MACRO COND, LAB
          J&COND   LAB
          ENDM
```

- ii. 表达式操作符% (强制求值)

的使用

```
DISP MACRO X
String DB 'ANSWER:', '&X', '$'
ENDM
```

宏调用: DISP %(2\*11-8) 产生的宏扩展为:  
1 String DB 'ANSWER:', '14', '\$'

不使用符号“%”的宏调用: DISP 2\*11-8  
产生的宏扩展却是:  
1 String DB 'ANSWER:', '2\*11-8', '\$'

- iii. 转义操作符! (将跟随字符作为普通字符)

如:

```
DISP MACRO X
String DB 'ANSWER:', '&X', '$'
ENDM
```

宏调用: DISP !%(2\*11-8)

产生的宏扩展为:  
1 String DB 'ANSWER:', '%(2\*11-8)', '\$'

- iv. LOCAL伪指令 解决标号的可能重复问题

### 2. 重复汇编

REPT 整数表达式

重复体

ENDM

```

X=0
REPT 10
    X=X+1
    DB X
ENDM

```

汇编后：

```

1    DB    1
1    DB    2
1    DB    3
-----
1    DB    10

```

a. IRP伪操作

IRP 形参, <实参表>

重复体

ENDM

```

IRP  REG, <AX, BX, CX, DX>
    PUSH REG
ENDM

```

b. IRPC伪操作

IRPC 形参, 字符串

重复体

ENDM

```

IRPC X, 0123456789
    DB X
ENDM

```

其结果等价于：

```

DB 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

```

### 3. 库的使用

a. 建立宏库

## 第八章：输入输出设计

三种数据传送方式速度：直接IO<中断<DMA

### 1. IO设备的数据传送方式

a. 接口信号

i. 数据信息

ii. 状态信息

iii. 控制信息

b. 数据传送方式

i. Direct Memory Access 基本概念

1) CPU向DMA控制器下达指令，具体交给DMA

2) DMA一般包括四个寄存器：控制寄存器、状态寄存器、地址寄存器、字节计数器

3) 在实现DMA传输的时候，又DMA控制器直接掌管总线。DMA传输之前，CPU要把总线控制权交给DMA控制器，结束传输后再返还

ii. 程序直接控制IO（查询方式）

iii. 中断传送

## 2. 程序直接控制IO方式

- a. IO端口存放在一个独立的地址空间中，有两种寻址方式
- b. 直接寻址 00H~0FFH 操作数表示端口号
- c. 间接寻址 0000H~0FFFFH (DX)表示端口号
- d. 所有IO端口与CPU的通信都由IN与OUT指令来完成（可见第四章2.b)

### 输出指令 OUT (CPU → I/O)

长格式: OUT PORT, AL (字节)

OUT PORT, AX (字)

执行操作: (PORT) ← (AL) (字节)

(PORT+1, PORT) ← (AX) (字)

短格式: OUT DX, AL (字节)

OUT DX, AX (字)

执行操作: (DX) ← (AL) (字节)

((DX)+1, (DX)) ← (AX) (字)

### 输入指令IN: (只限使用AX或AL)

长格式: IN AL, PORT (字节)

IN AX, PORT (字)

执行操作: (AL) ← (PORT) (字节)

(AX) ← (PORT+1, PORT) (字)

短格式: IN AL, DX (字节)

IN AX, DX (字)

执行操作: (AL) ← ((DX)) (字节)

(AX) ← ((DX)+1, (DX)) (字)

## e. 例题

### 程序直接控制I/O方式

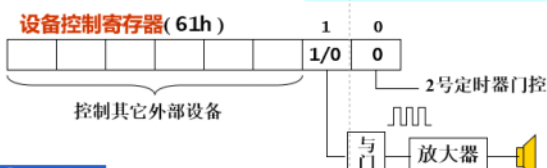
#### 例: Sound 程序

```
CODES SEGMENT
START:
    mov dx, 0ffffh ;响铃持续多长
    in al, 61h
    and al, 11111100b

sound:
    xor al, 2
    out 61h, al
    mov cx, 140h

wait1:
    loop wait1 ;延时
    dec dx
    jne sound

    MOV AH, 4CH
    INT 21H
CODES ENDS
END START
```



## 3. 中断传送方式

### a. 中断分类

- i. 8086可以管理256个中断，包括IO，异常事故或其它内部原因。内中断优先级比外中断高（除单步中断）
- ii. 软件中断（内中断）：由程序安排的中断指令产生的中断或者是错误结果
  - 1) INT
  - 2) CPU errors
  - 3) Debug INT
  - 4) 分类：中断0-除法错中断，中断1-单步中断，中断3-断点中断，中断4-溢出中断，用户自定义（指定中断）
  - 5) 特点
    - a) 中断向量码来源：指令，预定
    - b) 除单步中断外其它内部中断无法禁止
- iii. 硬件中断（外中断）：由外设控制器或协处理器引起的中断
  - 1) 可屏蔽中断（INTR）：IF可控制

- 2) 非屏蔽中断 (NMI)：为电源错误、内存或者是 IO 总线的奇偶校验等异常事件保留的中断，不受 IF 控制，整个体系只能有一个 NMI

#### iv. INT 中断调用指令

INT n

- 1) 说明：n 是中类型号，必须为 00H~0FFH 之间的立即数

- 2) 具体操作

```
PUSH FALGS
IF<-0,TF<-0
PUSH CS
PUSH IP
IP<-(4n+1, 4n)
CS<-(4n+3,4n+2)
```

#### v. IRET 终端返回指令

IRET

- 1) 说明：该指令是任何中断服务程序最后要执行的指令

- 2) 操作：

```
POP IP
POP CS
POP FLAGS
```

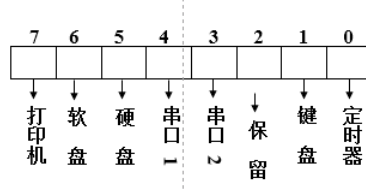
#### vi. 注意：中断中有两个控制条件起决定作用

同时满足才能中断

- 1) 外设的中断请求是否被屏蔽：由 8259A 的中断屏蔽寄存器 (IMR) 控制  
2) CPU 是否允许相应的中断：由 8259A 的标志寄存器中的中断允许位 (IF) 控制

##### 8259A

中断屏蔽寄存器 21H



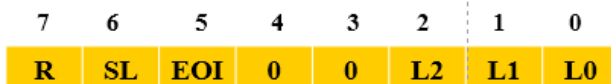
```
IN AL, 21H
AND AL, 0FDH
OUT 21H, AL
```

{ CLI IF=0  
STI IF=1

关中断  
开中断

#### vii. 中断命令寄存器

##### 8259A 中断命令寄存器 20H



```
MOV AL, 20H
OUT 20H, AL
```

- 1) 中断处理结束前，应将 8259A 的中断命令寄存器发送中断结束命令 (EOF)  
2) EOI=1 时，当前处理的中断请求被清除  
3) L2~L0 指定 IR0-IR7 中具有最低优先级的中断请求，第 6、7 位控制 IR0-IR7 中断优先级的顺序

#### viii. 中断步骤

- 1) 响应中断 (可屏蔽中断独有)
- 2) 将 FLAGS 压栈
- 3) 清除中断允许标志位 IF 和陷阱标志位 TF
- 4) 将 CS, IP 压栈
- 5) 找到程序入口，调用程序
- 6) 执行用户中断服务程序
- 7) IP, CS 栈弹出
- 8) FLAGS 弹出
- 9) 中断返回

#### ix. 中断步骤



## 中断优先级

中 断	优先级
除法错、INT <u>nn</u> 、INTO	最高
NMI	↓
INTR	↓
单步中断	最低

可屏蔽中断（INTR）的优先级又分为八级，优先级由高到低依次是：

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

8259A

IR0	← 08 系统定时器
IR1	← 09 键盘
IR2	← 0A CRT
IR3	← 0B 保留(通讯)
IR4	← 0C 串行通讯
IR5	← 0D 保留(ALT)
IR6	← 0E 软盘
IR7	← 0F 保留(打印机)

### b. 中断向量表

中断服务程序的入口地址

- 8086/8088在内存前1K字节（0000H~03FFH）建立了中断向量表
- 表中存有类型0~255号中断
- 每个类型的中断程序都含有4个字节的空间，前两字节存有程序入口的偏移地址，后两字节存有程序入口的段地址

INT N

- $IP \leftarrow (4N+1, 4N)$
- $CS \leftarrow (4N+3, 4N+2)$
- 例：

若中断号为20，则中断向量为：

IP=2010H

CS=4030H

00080H	10H
00081H	20H
00082H	30H
00083H	40H

### iv. 中断向量的分配

#### 1) 分配表

表 8.2 中断向量分配

地址	中断	地址	中断
0—7F	0—1F BIOS 中断向量	1C0—1DF	70—77 I/O 中断向量
80—FF	20—3F DOS 中断向量	1E0—1FF	78—7F 保留
100—17F	40—5F 扩充 BIOS 中断向量	200—3C3	80—FD BASIC 中断向量
180—19F	60—67 用户中断向量	3C4—3FF	F1—FF 保留
1A0—1BF	68—6F 保留		

如：DOS功能调用INT 21H

BIOS功能调用INT 10H（显示）、INT 14H（串口通信）等

#### 2) 用户自主扩充中断

- 如果中断只供自己使用，或者代替原有中断程序的时候，要注意保存原中断向量（按 保存-设置-恢复 的顺序操作）
- 可调用DOS功能调用（21H）来存取中断向量
  - 设置中断向量

AH=25H, AL=中断类型号, DS:DX=中断向量

INT 21H

ii) 读取中断向量

AH=35H, AL=中断类型号

INT 21H

返回: ES:BX=中断向量

iii) 完整代码

```
.....
MOV AL, N
MOV AH, 35H
INT 21H
PUSH ES
PUSH BX
PUSH DS
MOV DX, OFFSET INTHAND
MOV AX, SEG INTHAND
MOV DS, AX
MOV AL, N
MOV AH, 25H
INT 21H
POP DS
POP DX
POP DS
MOV AL, N
MOV AH, 25H
INT 21H
RET
.....
INTHAND: ..... ;中断处理程序
.....
IRET
```

例: 为中断类型 N 设置中断向量

;保存原来的中断向量

;偏移地址=>DX

;段地址=>DS

;设置新的中断向量

;恢复原来的中断向量

c. 中断程序设计方法

i. 中断处理程序的结构

与子程序(即过程)类似

ii. 中断程序的编写步骤

1) 主程序

- 设置中断向量
- 设置CPU的中断允许位IF
- 设置设备中断屏蔽位

2) 中断子程序

- 保存寄存器内容
- 如允许中断嵌套则开中断(STI)
- 中断处理功能
- 关中断
- 送EOI给中断命令寄存器
- 恢复寄存器内容
- IRET

例:

保护现场
STI
中断服务程序
CLI
恢复现场
中断返回

INTPRG PROC FAR	
STI	; 若允许中断嵌套
PUSH DS	
PUSH DX	
PUSH AX	
PUSH BX	
.....	; 中断处理
CLI	; 关中断
MOV AL, 20H	; 发中断结束命令EOI
OUT 20H, AL	
POP BX	; 恢复现场
POP AX	
POP DX	
POP DS	
IRET	; 中断返回
INTPRG ENDP	

## 第九章：BIOS及DOS调用

### 1. BIOS及DOS简介

#### a. BIOS Basic Input Output System 基本功能

- i. 系统自检和初始化
- ii. 系统服务
- iii. 硬件中断处理

#### b. DOS Disk Operating System 基本构成

- i. IBMBIO.COM 输入输出处理程序
- ii. IBMDOS.COM 文件管理
- iii. COMMAND.COM 命令处理

#### c. 层次关系



#### d. 调用关系

- i. 大部分情况既可以选择DOS中断也可以选择BIOS中断执行相同的任务
- ii. 少数情况必须使用BIOS中断
- iii. 一般来说DOS中断操作更简便，而且对硬件的依赖更加少

#### e. 用户编程原则

- i. 尽可能使用DOS系统功能调用，提高程序的可移植性
- ii. 在DOS不能实现的情况之下再考虑BIOS
- iii. 两者都无法实现的时候使用IN/OUT指令直接控制硬件

#### f. 调用基本方法

##### INT n

- i. DOS中断:  $n = 20H \sim 3FH$
- ii. BIOS:  $n = 5 \sim 1FH$
- iii. 自由中断  $n = 40H \sim FFH$
- iv. 完整的中断调用

- 1) 将参数装入指定寄存器
- 2) 如需功能调用号，装入AH
- 3) 如需子功能调用号，装入AL
- 4) 中断调用
- 5) 接受返回参数

#### g. DOS功能调用与BIOS的区别

- i. DOS的调用层次更高
- ii. 调用BIOS更复杂
- iii. 调用BIOS更快，功能更强
- iv. DOS调用只适用于DOS环境，BIOS不受操作系统的约束
- v. 某些功能只有BIOS具有

## 2. 键盘IO (可用DOS或BIOS进行键盘通信)

### a. 字符码和扫描码

#### i. 键盘组成

- 1) 字符数字 (直接传送ASCII码)
- 2) 功能扩展: Home, F1-10, PgDown等 (产生动作)
- 3) 组合控制键: Alt, Shift等 (改变其它键产生的字符码)

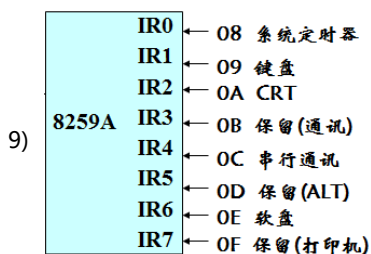
#### ii. 中断原理

- 1) 键盘与主机通过电源线、地线、复位线、键盘数据线、键盘时钟线相连
- 2) PC键盘触点电路用16\*8矩阵进行排列，用单片机Intel8048控制扫描
- 3) 按键识别通过行列扫描法确定闭合键位置，该位置值成为按键的扫描码，该扫描码将会被送回主机
- 4) 在键盘“按下”或者“放开”一个键的时候，如果键盘中断被允许（21H端口第一位为0），就会产生类型9的中断
- 5) 中断程序可以从输入端口读取一个字节，低7位为按键的扫描码。最高位为0表示按下，为1表示放开。按下时取得的字节成为通码，放开时取得的字节成为断码。如ESC键按下取得的通码为01H（00000001B），放开ESC键时会产生一个断码81H（10000001B）。
- 6) BIOS键盘处理程序将取得的扫描码转换为相应的字符码，大部分为标准ASCII码，没有ASCII码的键（如Alt和功能键），字符码为0.还有一些指定的操作键产生一个指定的操作（PrtSc）
- 7) 转换成的字符码及扫描码将存储在BIOS数据区的键盘缓冲区KB\_BUFFER中。键盘缓冲区是一个先进先出的队列，最多只能保存15个键盘信息。

```

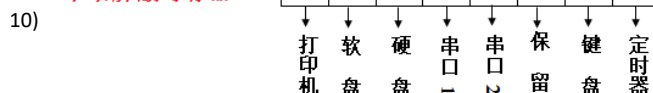
8) Buffer_Head DW ? ;键盘缓冲区头指针
   Buffer_Tail DW ? ;键盘缓冲区尾指针
   KB_Buffer DW 16 DUP(?) ;键盘缓冲区的缺省长度为16个字
   KB_Buffer_End Label Word

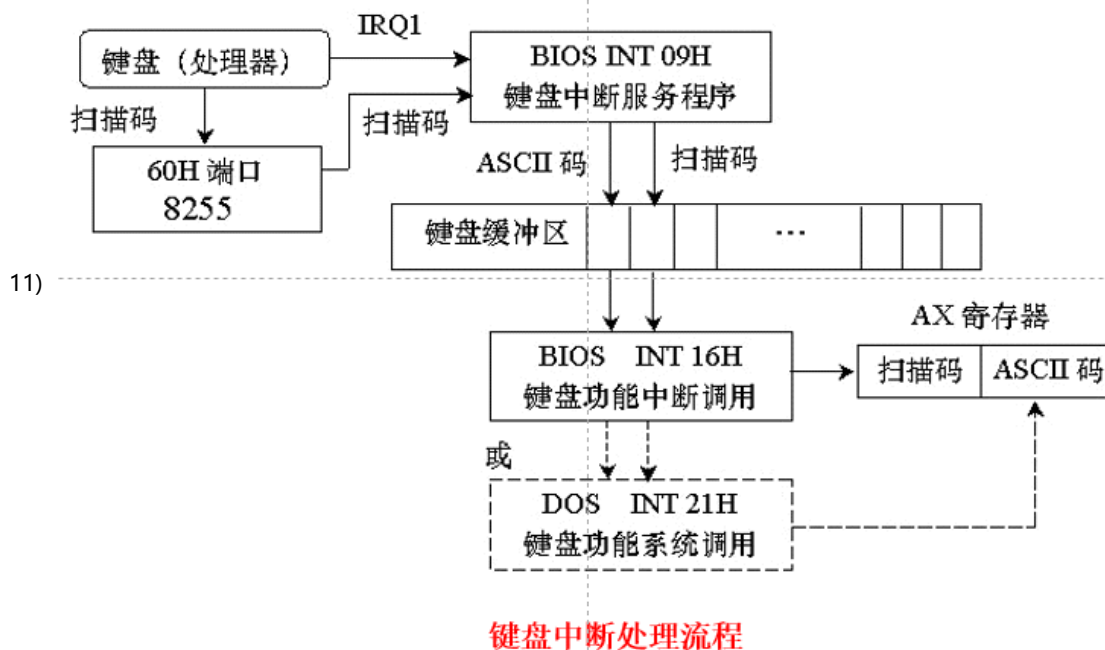
```



### 8259A

#### 中断屏蔽寄存器21H





## 12) 键盘中断09H与软件中断16H的比较

- 09H向键盘缓冲区写入，按下的时候触发
- 16H是应用程序调用的时候起作用

## b. BIOS键盘中断

### INT 16H

#### i. 中断的三种功能

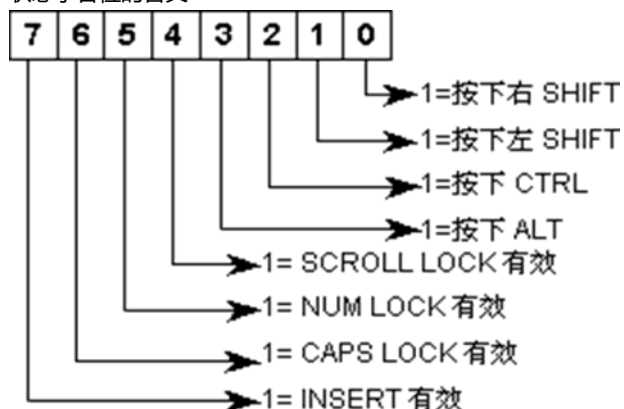
- AH=0: 从键盘读字符至AX，其中AL为字符码，AX为扫描码
- AH=1: 读键盘缓冲区字符到AX，并且置ZF标志位
  - ZF=1表示缓存区为空
  - ZF=0时，AL为字符码，AX为扫描码
- AH=2: 读取键盘状态字节至AL

#### ii. 键盘状态字

##### 1) 定义

- 控制键 Ctrl, Alt, Shift等
- 双态键 Num Lock, Caps Lock等
- 特殊请求键 Print Screen, Scroll Lock

##### 2) 状态字各位的含义



#### iii. 硬件中断过程

下面，我们通过下面几个键：

A、B、C、D、E、Shift\_A、A

的输入过程，简要地看一下 int 9 中断例程对键盘输入的处理方法。

(1) 初始状态下，没有键盘输入，键盘缓冲区空，此时没有任何元素。

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

(2) 按下 A 键，引发键盘中断；CPU 执行 int 9 中断例程，从 60h 端口读出 A 键的通码；然后检测状态字节，看看是否有 Shift、Ctrl 等切换键按下；发现没有切换键按下，则将 A 键的扫描码 1eh 和对应的 ASCII 码，即字母 “a” 的 ASCII 码 61h，写入键盘缓冲区。缓冲区的字单元中，高位字节存储扫描码，低位字节存储 ASCII 码。此时缓冲区中的内容如下。

1E61																			
------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

(3) 按下 B 键，引发键盘中断；CPU 执行 int 9 中断例程，从 60h 端口读出 B 键的通码；然后检测状态字节，看看是否有切换键按下；发现没有切换键按下，将 B 键的扫描码 30h 和对应的 ASCII 码，即字母 “b” 的 ASCII 码 62h，写入键盘缓冲区。此时缓冲区中的内容如下。

1E61	3062																		
------	------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

(4) 按下 C、D、E 键后，缓冲区中的内容如下。

1E61	3062	2E63	2064	1265															
------	------	------	------	------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

(5) 按下左 Shift 键，引发键盘中断；int 9 中断例程接收左 Shift 键的通码，设置 0040:17 处的状态字节的第 1 位为 1，表示左 Shift 键按下。

(6) 按下 A 键，引发键盘中断；CPU 执行 int 9 中断例程，从 60h 端口读出 A 键的通码；检测状态字节，看看是否有切换键按下；发现左 Shift 键被按下，则将 A 键的扫描码 1Eh 和 Shift\_A 对应的 ASCII 码，即字母 “A” 的 ASCII 码 41h，写入键盘缓冲区。此时缓冲区中的内容如下。

1E61	3062	2E63	2064	1265	1E41														
------	------	------	------	------	------	--	--	--	--	--	--	--	--	--	--	--	--	--	--

(7) 松开左 Shift 键，引发键盘中断；int 9 中断例程接收左 Shift 键的通码，设置 0040:17 处的状态字节的第 1 位为 0，表示左 Shift 键松开。

(8) 按下 A 键，引发键盘中断；CPU 执行 int 9 中断例程，从 60h 端口读出 A 键的通码；然后检测状态字节，看看是否有切换键按下；发现没有切换键按下，则将 A 键的扫描码 1Eh 和 A 对应的 ASCII 码，即字母 “a” 的 ASCII 码 61h，写入键盘缓冲区。此时缓冲区中的内容如下。

1E61	3062	2E63	2064	1265	1E41	1E61													
------	------	------	------	------	------	------	--	--	--	--	--	--	--	--	--	--	--	--	--

c. DOS中断调用

DOS键盘功能调用 (INT 21H)

AH	功 能	调用参数	返回参数
1 6	从键盘输入一个字符并显示在屏幕上 读键盘字符	无 (DL) = 0FFH	(AL) = 字符码 若有字符可取,(AL) = 字符码 ZF=0 若有字符可取,(AL) = 0
7 8 A B C	从键盘输入一个字符不显示 从键盘输入一个字符不显示 检测 Ctrl+Break 输入字符到缓冲区 读键盘状态 清除键盘缓冲区， 并调入一种键盘功能	无 无 DX DS: = 缓冲区首址 无 (AL) = 键盘功能号 (1, 6, 7, 8 或 A)	ZF=1 (AL) = 字符码 (AL) = 字符码 (AL) = 0FFH, 无键入 (AL) = 00H, 有键入

- i. AH=1 单字符输入要求接受功能键或者是数字组合键必须进行两次DOS调用，第一次回送00，第二次回送扫描码
- ii. AH=0AH 输入字符串，第一个字节保存最大字符数（用户自定义），第二个字节是用户输入的字符串包括结束的回车。缓

缓冲区大小为：最大字符数+2

```
MaxLen DB 32
ActLen DB ?
String DB 32 DUP (?)
```

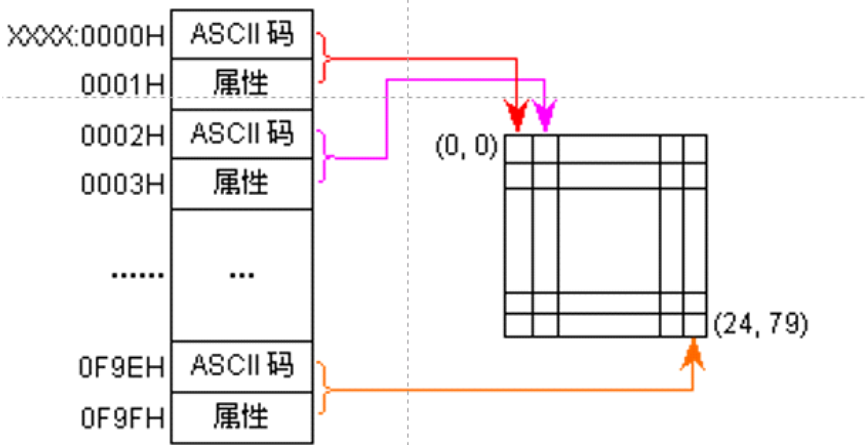
```
LEA DX, MaxLen
MOV AH, 0AH
INT 21H
```

- iii. AH=0CH 清除键盘缓冲区
- iv. AH=0BH 检验一个键是否被按动

3. 显示器IO

a. 字符属性

- i. 显示器文本显示模式 80\*25个字符((0,0)->(24,79))
- ii. 显存
  - 1) 不同显示适配器显存的起始地址不同
  - 2) 单色显示适配器MDA B000:0000
  - 3) CGA,EGA,VGA B800:0000
  - 4) 在80\*25的显示方式下，屏幕可有2000个字符位置，每个字符需要用两个字节，每屏显存需要有4K
  - 5) 对VGA的80列显示方式，0页的起始地址为B800:0000,1页为B800:1000, 2页为...



iii. 字符属性（分别为单色、彩色字符）

7	6	5	4	3	2	1	0
闪烁位		背景色		亮度色		前景色	
0=正常显示 1=闪烁显示		000=黑 111=白		0=正常亮度 1=加强亮度		000=黑 111=白	
属性值（二进制）		属性值（十六进制）		显示效果			
00000000		00		无显示			
00000001		01		黑底白字，下划线			
00000111		07		黑底白字，正常显示			
00001111		0F		黑底白字，高亮度			
01110000		70		白底黑字，反相显示			
10000111		87		黑底白字，闪烁			
11110000		F0		白底黑字，反相闪烁			

如：黑底白字闪烁显示，可设置属性为87H（10000111）  
不显示字符：00H

**2、彩色字符显示：**显示彩色字符时，属性字节可以选择显示字符的前景颜色和背景颜色。

◆ 前景颜色有16种可以选择，背景颜色有8种可以选择。

◆ 闪烁和亮度只应用于前景。

彩色字符显示属性字节			
位号	7	6 5 4	3 2 1 0
属性字节	BL	R G B	I R G B
	闪烁选择	背景颜色	前景颜色

b. BIOS显示中断

由于给出的功能太多，只列出例子，不做细分

**例：置光标的类型：**开始行为1，结束行为7，并把它设置到第五行、第六列。

CODES SEGMENT  
ASSUME CS:CODES

START:

mov ch,1

mov cl,7

mov ah,1

int 10h

i.

mov dh,5

mov dl,6

mov bh,0

mov ah,2

int 10h

mov ah,1 ;输入字符并显示

int 21h

MOV AH,4CH

INT 21H

CODES ENDS

END START

设置光标类型（1号功能）

入口参数：AH=1（功能号），CH=光标开始行，CL=光标结束行。

只用CH、CL的低4位，若CH的第4位为1，光标不显示。

出口参数：无。

根据CX给出光标的大小。

设置光标位置（2号功能）

入口参数：AH=2（功能号），BH=页号，DH=行号，DL=列号。

出口参数：无。

根据DX确定了光标位置。



## 例：在品红背景下，显示5个浅绿色闪烁的星号

code segment

assume cs:code

start:

```
mov ah,9
mov al,'*'
mov bh,0
mov bl,0dah
mov cx,5
int 10h
mov ah,4ch
int 21h
```

code ends

end start

**917.asm**

在当前光标位置写字符和属性（9号功能）

入口参数：AH=9，BH=页号，AL=字符的ASCII码，BL=字符属性，CX=写入字符次数。

出口参数：无。

背景颜色组合

前景颜色组合

RGB	颜色	IRGB	颜色	IRGB	颜色
000	黑	0000	黑	1000	灰
001	蓝	0001	蓝	1001	浅蓝
010	绿	0010	绿	1010	浅绿
011	青	0011	青	1011	浅青
100	红	0100	红	1100	浅红
101	品红	0101	品红	1101	浅品红
110	棕	0110	棕	1110	黄
111	白	0111	白	1111	强度白

## 显示器I/O

例：在屏幕中心建立一个20列宽和9行高的小窗口。在小窗口的最下一行输入字符，满一行就向上滚动。 48/83

data segment

```
esc_key equ 1bh ; ESC的ASCII码
win_ulc equ 30 ; 左上角列号
win_ulr equ 8 ; 左上角行号
win_lrc equ 50 ; 右下角列号
win_lrr equ 16 ; 右下角行号
win_width equ 20
```

data ends

iii. code segment

assume cs:code,ds:data

```
start: mov ah,2
mov dh,win_lrr
mov dl,win_ulc
mov bh,0
int 10h
mov cx,win_width
get_char:
mov ah,1
int 21h
```

设置光标位置（2号功能）

入口参数：AH=2（功能号），BH=页号，DH=行号，DL=列号。

出口参数：无。根据DX确定了光标位置。

cmp al, esc\_key

jz exit

loop get\_char

mov ah,6

mov al,1

mov ch,win\_ulr

mov cl,win\_ulc

mov dh,win\_lrr

mov dl,win\_lrc

mov bh,7

int 10h

jmp start

exit: mov ah,4ch

int 21h

code ends

end

初始窗口或向上滚动（6号功能）

入口参数：AH=6，AL=上滚行数，CX=上滚窗口左上角的行、列号。DX=上滚窗口右下角的行、列号。BH=空白行的属性。

出口参数：无。当滚动后，底部为空白输入行。

**913.asm**

[luguangm@gmail.com](mailto:luguangm@gmail.com)

c. DOS显示功能调用

由于给出的功能太多，只列出例子，不做细分

例：编程显示字符串

data SEGMENT ;定义显示的子字符串

stri DB 'Harbin Institute of Technology (Shenzhen)', '\$'

data ENDS

code SEGMENT

ASSUME CS:code, DS:data

start: MOV AX, data ;置缓冲区地址于DS:DX

MOV DS, AX

LEA DX, stri

### 例：编程显示字符串

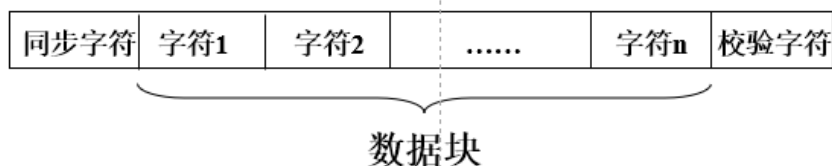
```
data    SEGMENT           ;定义显示的子字符串
stri    DB  'Harbin Institute of Technology (Shenzhen)', '$'
data    ENDS
code    SEGMENT
        ASSUME  CS:code, DS:data
start:  MOV  AX, data      ;置缓冲区地址于DS:DX
i.      MOV  DS, AX
        LEA  DX, stri
        MOV  AH, 09H      ;调显示功能
        INT  21H
        MOV  AH, 4CH      ;返回DOS
        INT  21H
code    ENDS
        END      start
```

#### 4. 串行通信口IO

a. 通信过程分类：单工、半双工、全双工

b. 串口通信分类

- 异步通信：一个字符一个字符的传输，字符一位一位的传输。传输字符步骤：起始位、字符数据、校验位、停止位。该步骤构成帧(Frame)，帧与帧之间可以有任意的空闲位，起始位后是数据的最低位。
- 同步通信：相比较于异步通信，不加起始位和停止位，直接连接起来形成数据块（信息帧）。在开始加上同步字符，结束加上差错检验



c. 波特率与传输率

串行通信中，传输速率是用波特率来表示。所谓波特率是指单位时间内传送二进制数据的位数(简称为bps)。在计算机里，每秒传输多少位和波特率的含义是完全一致的。

收、发双方的波特率必须一致。

例：计算串行传输5页，每页80x25个字符总共需要多少位？假设每个字符8位，1位起始位和1位终止位。计算传输上述五页所需要的时间。若数据传输率为9600bps。

解：每个字符10b

总数据量为：10\*80\*25\*5=100000b

所需时间：100000/9600=10.4秒

d. DOS串行通信口功能调用

使用DOS命令可以设置串行通信参数，如波特率，校验位，字长和终止位。

常用的波特率：2400、4800、9600、19200、38400等。

格式：MODE COMx:b, d, s

使用DOS命令可以设置串行通信参数，如波特率，校验位，字长和终止位。

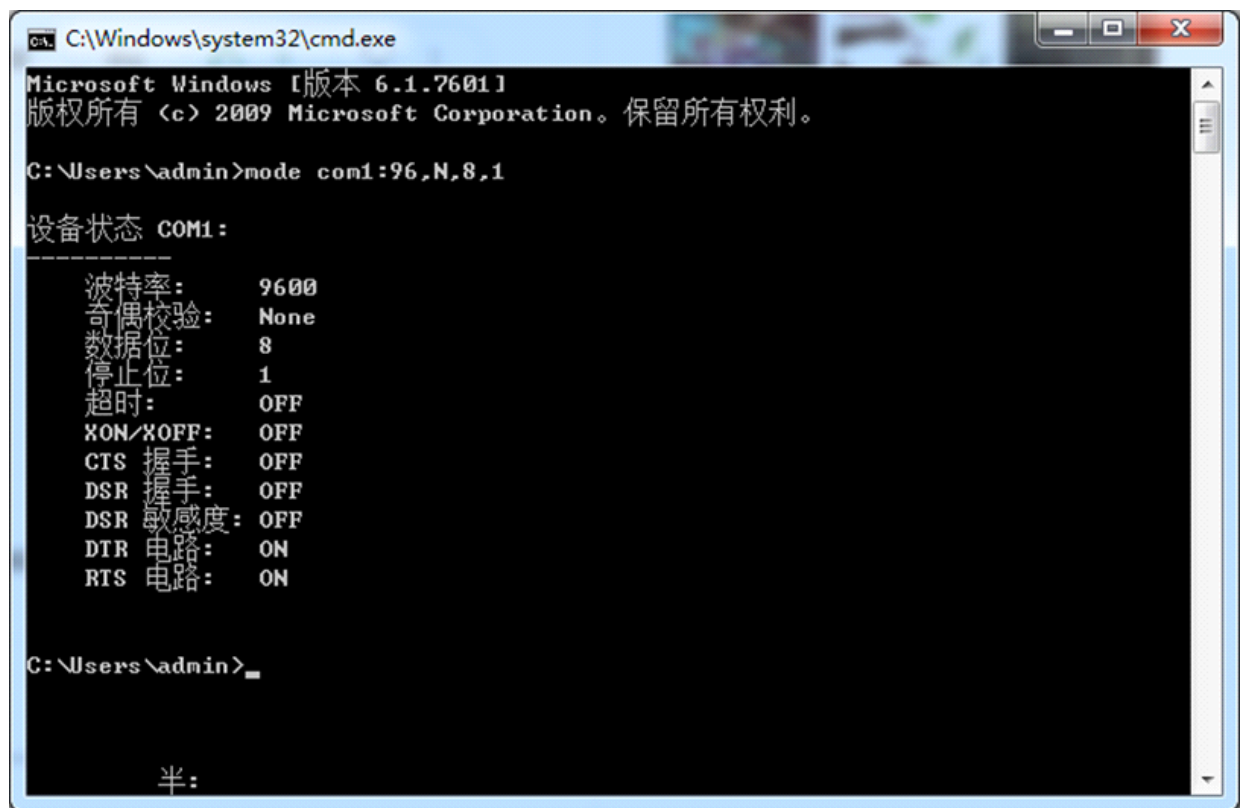
常用的波特率：2400、4800、9600、19200、38400等。

格式：MODE COMm:b,p,d,s

例如：MODE COM1:96,O,8,1

说明：b：波特率，用波特率的最高两位来表示；P：校验位（N：无校验，O：奇校验，E：偶校验）；d：数据的字长（5,6,7,8，默认值是7）；s：终止位位数（1，1.5，或2）。

端口号的分配：BIOS位 0 — 3，DOS为 1 — 4



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\admin>mode com1:96,N,8,1

设备状态 COM1:
-----
波特率:      9600
奇偶校验:    None
数据位:      8
停止位:      1
超时:        OFF
XON/XOFF:    OFF
CTS 握手:    OFF
DSR 握手:    OFF
DSR 敏感度:  OFF
DTR 电路:    ON
RTS 电路:    ON

C:\Users\admin>
```

e. DOS调用示例

例：将字符串“HELLO”通过串口输出。

```
BUFFER DB 'HELLO'
BUF_LEN EQU $-BUFFER

...
MOV BX, OFFSET BUFFER
MOV CX, BUF_LEN

NEXT:
    MOV DL, [BX]
    MOV AH, 4
    INT 21H
    LOOP NEXT

...
```

f. BIOS调用示例

串行通信口 BIOS功能调用(int 14h)

AH	功能	调用参数	返回参数
0	初始化串行口	AL=初始化参数(书上图9.12) DX=通信口号 (COM1=0,COM2=1 etc)	AH=通信口状态 (书上图9.13) AL=调制解调器状态
1	向串行口写字符	AL=所写字符 DX=通信口号 (COM1=0,COM2=1 etc)	写成功: AH=0AL=字符。 失败: (AH)7=1 (AL)1-6=通信口状态
2	从串行口读字符	DX=通信口号 (COM1=0,COM2=1 etc)	写成功: AH=0AL=字符。 失败: (AH)7=1 (AL)1-6=通信口状态
3	取串行口状态	DX=通信口号 (COM1=0,COM2=1 etc)	AH=通信口状态 AL=调制解调器状态

例：两台PC机通过COM2端口进行串行数据通信，编写一个汇编语言程序，要求从一台PC 机上键盘上输入的字符传送到另一台PC机，若按下ESC键，则退出程序。在程序中，COM2端口初始化为 4800BPS，8位数据，无校验，1位终止位。

<pre> .data message db 'serial communication via com2,4800,no p,1stop,8 bit data ',0ah,0dh,'\$'  db 'any key press is sent to other PC ',0ah,0dh db 'press esc to exit','\$'  .code main proc     mov ax,@data     mov ds,ax     mov ah,09     mov bx,offset message     int 21h      mov ah,0     mov dx,1     mov al,03ch     int 14h  again: mov al,01     int 16h     jz next     mov ah,0     int 16h </pre>	<pre> cmp al,1bh je exit mov ah,1 mov dx,1 int 14h next: mov ah,3 mov dx,1 int 14h and ah,1 cmp ah,1 jne again mov ah,2 mov dx,1 int 14h mov dl,al mov ah,2 int 21h jmp again exit: mov ah,4ch int 21h main endp end </pre>
---	---

69/80

## 5. 文件存储IO

### a. 概念

DOS INT 21H

#### i. 文件代号：16位二进制控制字

- 1) 预定义文件代号0-4
- 2) 对于建立和打开的文件，代号从6开始排序，最多同时打开5个文件
- 3) 程序执行时，调用的每一个文件都必须分配唯一——个文件

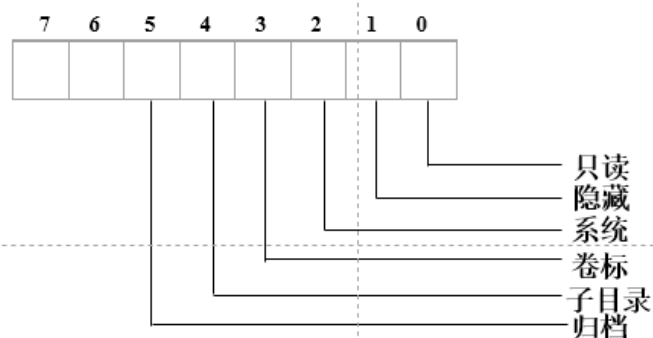
#### ii. 路径名与ASCIZ串：ASCIZ串最后字节为0，其它为指示文件位置的ASCII码字符串

```
filename1 DB 'C:\SAMPLE.TXT',00
```

#### iii. 返回错误码

CF=1表示操作失败，CF=0表示操作成功，其它错误代码不做介绍

#### iv. 文件属性：一个字节



一般情况下，用户文件只具有一种属性，如属性代码为00H的普通文件，属性代码为01H的只读文件，属性代码为02H的隐藏文件。重要的系统文件通常有多种属性，如属性代码为07H的文件，就具有只读、隐藏、系统三种属性。属性字节存放到CX寄存器中。

#### v. 文件指针

- 1) 建立文件或者打开文件成功之后，DOS将提供一个文件指针指示文件的当前位置
- 2) 文件指针为32位，初值为0指示文件的开始位置

3) 每次对文件的读写操作时，系统都会自动修改文件指针指向下一个要读写的位置

b. 调用

由于内容比较多，只列出例子

i. 例一：建立磁盘文件

例：建立一个有正常属性文件的指令序列

```
PATHNM1 DB 'E:\ACCOUNTS.FIL',00H
HANDLE1 DW ?
...
MOV AH,3CH
MOV CX,00
LEA DX, PATHNM1
INT 21H
JC ERROR
MOV HANDLE1, AX
...
```

建立文件 (3CH)

功能：按指定文件名建立文件。

入口参数：

(AH)=3CH,

DS:DX指向ASCIZ字符串的段地址和偏移地址

(CX)=文件属性。

出口参数：

若成功，则CF=0, (AX)=文件代号；

若失败，则CF=1, (AX)=错误代码。

ii. 例二：打开和关闭磁盘文件

二、打开和关闭磁盘文件

```
PATH DB 'E:\ACCOUNTS.FIL',00H
HANDLE DW ?
...
;Open the file
MOV AH,3DH
MOV AL,0
LEA DX, PATH
INT 21H
JC ERROR
MOV HANDLE, AX
...
;Close the file
MOV AH,3EH
MOV BX, HANDLE
INT 21H
JC ERROR
...
```

打开文件 (3DH)

功能：打开由ASCII Z串指定的文件。

入口参数：

(AH)=3DH, DS:DX指向ASCIZ字符串的段地址和偏移地址, (AL)=存取代码 (0: 读文件, 1: 写文件, 2: 读、写文件)。

出口参数：

若成功，则CF=0, (AX)=文件代号；

若失败，则CF=1, (AX)=错误代码。

关闭文件 (3EH)

功能：关闭文件代号指定的文件。

入口参数：

(AH)=3EH, (BX)=文件代号。

出口参数：

若操作成功，则CF=0；

若操作失败，则CF=1, (AX)=错误代码。

iii. 例三：读磁盘文件

例：从文件readme.txt中读512字节到缓冲区。

```
PATH DB 'E:\readme.txt',00H
HANDLE DW ?
Data DB 512 DUP(?)
...
;Open the file
MOV AH,3DH
MOV AL,0
LEA DX, PATH
INT 21H
JC ERROR
MOV HANDLE, AX
...
;Read the file
MOV AH,3FH
MOV BX, HANDLE
MOV CX, 512
LEA DX, data
INT 21H
JC ERROR
CMP Ax, 0
JE EndFile ;表示文件为空
...
```

读文件 (3FH)

入口参数：

(AH)=3FH, (BX)=文件代号, (CX)=要读取的字节数；

DS:DX指向接收数据缓冲区的段地址和偏移地址。

出口参数：

若成功，则CF=0, (AX)=实际读入字节数, (AX)=0, 文件结束；

若失败，则CF=1, (AX)=错误代码。

当文件读入操作完成后，要及时关闭。



iv. 例四：写磁盘文件

例：把OUTREC数据区中的256个字节写入磁盘文件

```
HENDLE DW ?
OUTREC DB 256 DUP(?)
...
MOV AH,40H
MOV BX,HANDLE
MOV CX,256
LEA DX,OUTREC
INT 21H
JC ERROR1
...
;Close the file
MOV AH,3EH
MOV BX,HANDLE
INT 21H
JC ERROR2
...
```

写文件 (40H)

入口参数:

(AH) = 40H, (BX) = 文件代号, (CX) = 要写入的字节数;

DS: DX指向存放写入信息数据缓冲区的段地址和偏移地址。

出口参数:

若成功, 则CF = 0, (AX) = 写入字节数;

若失败, 则CF = 1, (AX) = 错误代码。

当文件写入操作完成后, 必须用DOS功能调用3EH来关闭文件, 以确保操作系统将文件记录在磁盘上。

v. 例五：从文件file1中读取十个字符到file2文件中

```
data segment
    fname db 'c:\file1.dat',00
    fname1 db 'c:\file2.dat',00
    dta db 80h dup(0)
data ends
code segment
    assume cs:code,ds:data
start:mov ax,data
    mov ds,ax
    mov es,ax
    mov dx,offset fname ;Open source file
    mov al,0 ;read file
    mov ah,3dh ;Open the file
    int 21h
    mov si,ax ;file number
    mov bx,si
    mov dx,offset dta ;read from the source file
    mov cx,10 ;read 10 bytes
    mov ah,3fh ;read the file
    int 21h
```

```
mov di,ax ;read bytes
mov ah,3eh ;close the source file
int 21h
mov dx,offset fname1 ;make the new file
mov cx,0
mov ah,3ch
int 21h
mov si,ax
mov dx,offset dta ;write into the file
mov cx,di ;write bytes
mov bx,si ;file number
mov ah,40h ;write
int 21h
mov bx,si ;close the file
mov ah,3eh
int 21h
mov ah,4ch
int 21h
code ends
end start
```