

汇编语言

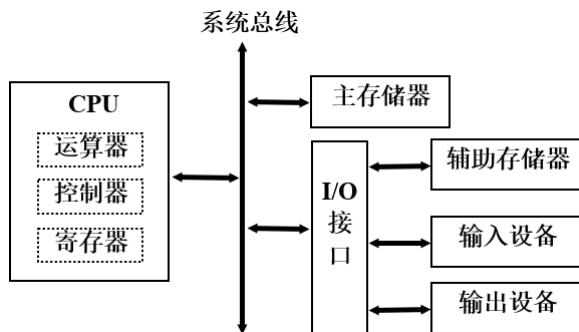
一、汇编语言基础知识

1、计算机系统概述

1.1、计算机硬件

1.1.1、计算机硬件的分类

- 中央处理单元 (CPU)
 - 控制器
 - 运算器
 - **寄存器**
- 存储器
 - 主存储器: RAM和ROM
 - 辅助存储器: 磁盘、U盘
- 外部设备 (接口电路)



1.1.2、寄存器

- 概念: CPU内部的高速存储单元, 为处理器提供各种操作所需要的数据、地址等信息
- 分类
 - 16位CPU: AX BX CX DX SI DI BP SP
 - 32位CPU: EAX EBX ECX EDX ESI EDI EBP ESP

1.1.3、存储器地址

- 概念: **存储器地址**是存储器中**存储单元的编号**

1.1.4、端口 (I/O地址)

- 概念: I/O接口电路由**接口寄存器**组成, 需要用编号区分各个寄存器, 这个编号就是端口
- 作用: **系统通过端口与外设进行通信**

1.2、计算机软件

- 系统软件
- 应用软件

1.3、计算机的程序设计语言

1.3.1、程序设计语言的分类

- 机器语言
- 汇编语言
- 高级语言

1.3.2、汇编语言概述

- 什么是汇编语言、汇编语言程序、汇编程序?
 - 汇编语言：汇编格式指令以及使用它们编写程序的规则就是汇编语言；以助记符的形式表示计算机指令
 - 汇编语言程序：用汇编语言书写的程序
 - 汇编程序：将汇编语言程序汇编成机器代码目标模块的程序
- 汇编语言与高级语言的比较?
 - 优点
 - 本质上就是机器语言
 - 可以直接有效的控制计算机硬件
 - 编写时间和空间两方面最有效的程序
 - 缺点
 - 与处理器密切相关——通用性、可以执行较差
 - 编写繁琐、调试困难

2、数据表示

2.1、数制

- 基数：N进制数，N就是基数
- 数码：N进制数，有N个数码
- 十进制转N进制
 - 整数部分：除以N取余，余数倒序排列
 - 小数部分：乘以N取整，整数正序排列
- N进制转十进制
 - 按权展开

2.2、编码

2.2.1、BCD码：二-十进制码

- 组成：用4位二进制数表示1位十进制数
- 分类
 - 压缩BCD码：一个字节表达两位BCD码
 - 非压缩BCD码：一个字节表达一位BCD码，高4位设置0，低4位表达数值

2.2.2、ASCII码

- 标准ASCII码
 - 组成：7位二进制编码，有128个
 - 常用：0---30H A---41H a---61H 回车---0DH 换行---0AH 空格---20H
- 扩展ASCII码： D_7 位置1

2.3、有符号数表示法

2.3.1补码

- 正数补码：同无符号数

- 负数补码：对应正数补码取反加1
- N位二进制补码表示的范围： $-2^{N-1} \sim 2^{N-1} - 1$

2.4、二进制数运算

2.4.1二进制和十六进制数的关系

- 每4个二进制位对应1个十六进制位

2.4.2十六进制数的加减运算

- 类似十进制，逢16进1，借1当16

3、Intel80×86系列微处理器

3.1、16位80×86微处理器

- 8086的工作方式是实方式
- 80286增加保护方式
- 80286引入了系统指令

3.2、IA-32微处理器

- 提供虚拟8086工作方式

3.3、Intel64位处理器

- 处理器集成多核技术

4、PC微型计算机

4.1、内存空间的分配（32位PC机）

- 系统RAM区：地址最低端的640KB，由DOS进行管理
- 显示RAM区：128KB，没有被完全使用
- 扩展ROM区：I/O接口电路卡上的ROM
- 系统ROM区：ROM-BIOS程序

4.2、微机的软件

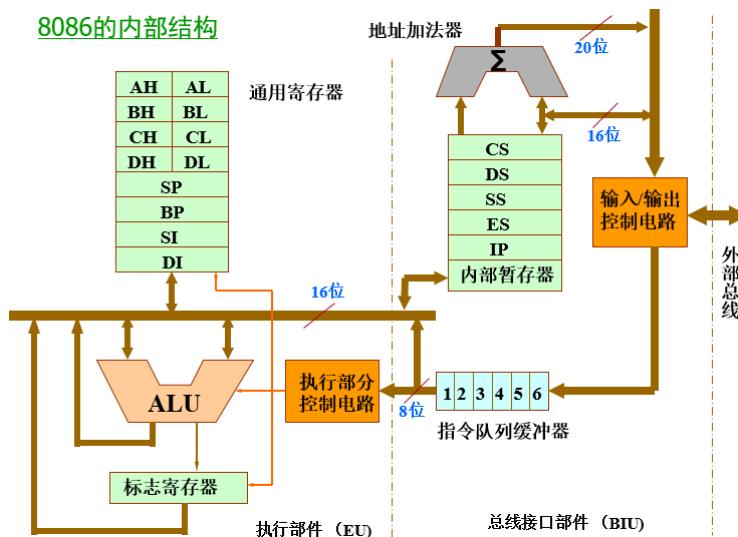
- 系统软件：DOS平台
- 应用软件
 - 文本编辑器：编写源程序
 - 汇编程序：将汇编语言源程序翻译成机器代码目标模块
 - 连接程序：将汇编后的目标模块转换为可执行程序
 - 调试程序：DEBUG程序

5、8086微处理器

5.1、概述

- 微处理器抽象为以名称存取的寄存器
- 8086内部结构的两个功能模块
 - 总线接口单元（BIU）：负责读取指令和操作数
 - 执行单元（EU）：负责指令译码和执行

5.2、8086的内部结构



5.3、8086的寄存器

5.3.1、通用寄存器

- 数据寄存器：AX BX CX DX
 - 功能：存放计算的结果和操作数，也可以存放地址
 - 各自的专用目的
 - AX：累加器，使用频度最高
 - BX：基址寄存器，存放存储器地址
 - CX：计数器，用于循环和串操作
 - DX：数据寄存器，存放双字长数据的高16位
- 变址寄存器：SI DI
 - SI：源变址寄存器
 - DI：目的变址寄存器
 - SI与DI在串操作指令有特殊的用法
- 指针寄存器：BP SP
 - 功能：寻址内存堆栈内的数据
 - SP：堆栈指针寄存器，只能用于指示栈顶的偏移地址
 - BP：基址指针寄存器，表示数据在堆栈段中的基址

5.3.2、标志寄存器

- 作用：用于反应指令执行结果或控制指令执行形式
- 状态标志：记录程序运行结果的状态信息
 - CF：进位标志，运算结果的最高有效位有进位(加法)或借位(减法)时，CF置1；否则置0
 - ZF：零位标志，若运算结果为0，则ZF = 1（不考虑进位）
 - SF：符号标志，运算结果最高位为1，则SF = 1；否则SF = 0
 - PF：奇偶标志，当运算结果最低字节中“1”的个数为零或偶数时，PF = 1；否则PF = 0
 - OF：溢出标志，若算术运算的结果有溢出，则OF = 1；否则 OF = 0
 - AF：辅助进位标志，运算时D3位（低半字节）有进位或借位时，AF = 1；否则AF = 0
- 控制标志：可由程序根据需要用指令设置，用与控制处理器执行指令的方式
 - DF：方向标志，重复串操作时，0表示正序，1表示倒序
 - IF：中断允许标志
 - TF：单步标志
- 溢出和进位的区别

- 进位标志表示**无符号数**运算结果是否超出范围，超出范围后加上进位或借位运算结果仍然正确；
- 溢出标志表示**有符号数**运算结果是否超出范围，超出范围后运算结果不正确。
- 只有当两个相同符号数相加（包括不同符号数相减），而运算结果的符号与原数据符号相反时，产生溢出

5.3.3、指令指针IP

- 作用：专用寄存器，与CS联用，CS:IP确定下一条指令的物理地址

5.3.4、段寄存器

每个段寄存器确定一个逻辑段的起始地址

- CS：代码段寄存器，指明代码段的起始地址
- SS：堆栈段寄存器，指明堆栈段的起始地址
- DS：数据段寄存器，指明数据段的起始地址
- ES：附加段寄存器，指明附加段的起始地址

5.3.5、存储器组织

- 数据信息的表达单位
 - LSB：数据的最低位，D₀位
 - MSB：最高有效位，D₇、D₁₅、D₃₁
- 数据存储格式
 - 单字节数据
 - 存放、表达：[内存单元的编号]
 - 字、双字数据
 - 存放：低字节存入低地址，高字节存入高地址
 - 表达：[低地址]
- 数据的地址对齐

同一个存储器地址可以是字节单元地址、字单元地址、双字单元地址等等

字单元安排在偶地址(xxx0B)、双字单元安排在模4地址(xx00B)等，被称为“**地址对齐**(Align)”。

- 存储器的分段管理

8086有20根地址线，**最大可寻址空间为1MB**，物理地址范围从00000H ~ FFFFFH

一个存储单元具有一个唯一的物理地址，还具有多个逻辑地址(段基地址 : 段内偏移地址)

1MB空间最多能分成**64K**个段，最少能分成**16**个段

5.3.6、8086的寻址方式

- 指令的组成
 - 操作码：对应机器指令的一个二进制编码
 - 操作数：可以是具体数值、寄存器、存储器地址
- 立即数(imm)寻址

指令中的操作数直接存放在机器代码中，紧跟在操作码之后（操作数作为指令的一部分存放在操作码之后的主存单元中）

- 寄存器(reg)寻址

同时指定了操作的数据是多少位

- 存储器寻址
 - 直接寻址：有效地址在指令中直接给出

```
MOV AX, [2000H]
```
 - 寄存器间接寻址：有效地址存放在基址寄存器BX或变址寄存器SI、DI中

```
MOV AX, [SI]
```
 - 寄存器相对寻址：有效地址是寄存器内容与有符号数之和，寄存器可以是BX/BP或SI/DI

```
MOV AX, [DI+06H]
```
 - 基址变址寻址：有效地址由基址寄存器(BX或BP)的内容加上变址寄存器(SI或DI)的内容构成

```
MOV AX, [BX+SI]
```
 - 相对基址变址寻址：有效地址是基址寄存器(BX/BP)、变址寄存器(SI/DI)与有符号数之和

```
MOV AX, [BX+SI+06H]
```

二、8086的指令系统

- 指令系统：能够执行的全部指令的集合
- 学习指令的关注点
 - 指令的功能
 - 指令支持的寻址方式
 - 指令对标志位的影响
 - 其他方面：预置参数、隐含使用的寄存器
- 指令格式

标号 : 指令助记符 目的操作数, 源操作数 ; 注释

1、数据传送类指令

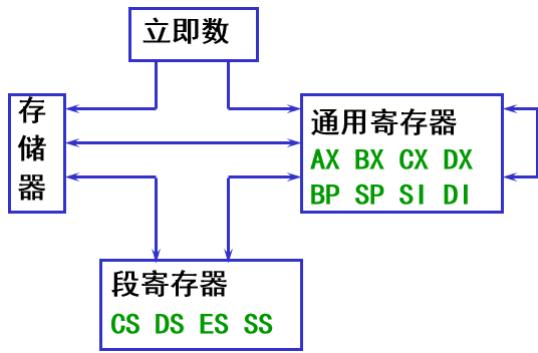
- 最基本、最重要、最常用
- 除标志寄存器传送指令外，均不影响标志位

1.1、MOV指令

- 当源操作数是立即数，目的操作数是存储器时，要明确是字节操作还是字操作

```
1 | mov word ptr [si+2],0bh
```

- 不存在存储器向存储器的传送指令
- 不允许立即数传送给段寄存器
- 不允许直接改变CS值
- 不允许段寄存器之间直接传送数据



1.2、XCHG交换指令

- 寄存器与寄存器之间对换数据
- 寄存器与存储器之间对换数据

1.3、XLAT换码指令

- 功能：将BX指定的缓冲区中、AL指定的位移处的一个字节数据取出赋给AL

```
1 | MOV al, ds:[bx+a1]
```

1.4、堆栈操作指令

- 堆栈操作的单位是字
- 堆栈的作用
 - 临时存放数据
 - 传递参数
 - 保存和恢复寄存器
- PUSH: 先使SP-2, 然后把一个字操作数存入堆栈顶部
- POP: 先把栈顶的一个字传送至指定的目的操作数, 然后使SP加2

1.5、标志传送指令

- 作用：临时保存标志位数据
- 低8位传送
 - LAHF: 将标志寄存器的低字节送寄存器AH
 - SAHF: 将AH寄存器内容送FLAGS的低字节

```
1 | ;置位CF为1, 等价于STC
2 | LAHF
3 | OR AH, 01H
4 | SAHF
```

- 16位传送
 - PUSHF: 将标志寄存器的内容压入堆栈, 同时栈顶指针SP减2
 - POPF: 将栈顶字单元内容送标志寄存器, 同时栈顶指针SP加2

```
1 ;置位TF为1
2 PUSHF
3 POP AX
4 OR AX, 0100H
5 PUSH AX
6 POPF
```

- 标志位操作

对CF操作

- CLC: CF := 0
- STC: CF := 1
- CMC: CF := ~CF

串操作指令中对DF操作，规定倒/正序操作串

- CLD: DF := 0
- STD: DF := 1

中断服务层序程序中对DF操作

- CLI: DF := 0
- STI: DF := 1

1.6、地址传送指令

- LEA: 将存储器操作数的有效地址传送至指定的16位寄存器中

```
1 | lea bx,[bx+si+0f62h]
```

- LEA和OFFSET的区别

- LEA可以用直接、间接、基址变址、相对基址变址，OFFSET只能用直接寻址
- OFFSET在汇编时由汇编程序计算偏移地址，LEA是在程序运行时由指令实现
- 在程序运行中才能得到的地址只能用LEA

- LDS\LES: 指针传送指令，双字传送

```
1 | MOV WORD PTR [3060H],0100H
2 | MOV WORD PTR [3062H],1450H
3 | LDS SI,[3060H] ;DS = 1450H,SI = 0100H
4 | LES DI,[3060H] ;ES = 1450H,DI = 0100H
```

2、算术运算类指令

- 注意算术运算类指令对标志的影响

2.1、加法指令ADD

- 功能：将源与目的操作数相加，结果送到目的操作数
- 对标志位的影响：按状态标志的定义相应设置

2.2、带进位加法指令ADC

- 功能：将源与目的操作数相加，再加上进位CF标志，结果送到目的操作数
- 对标志位的影响：按状态标志的定义相应设置

2.3、增量指令INC

- 功能：对操作数加1
- 对标志位的影响：**不影响CF**，按定义设置其他状态标志

2.4、减法指令SUB

- 功能：将目的操作数减去源操作数，结果送到目的操作数
- 对标志位的影响：按照定义相应设置状态标志

2.5、带借位的减法指令SBB

- 功能：将目的操作数减去源操作数，再减去借位CF（进位），结果送到目的操作数
- 对标志位的影响：按照定义相应设置状态标志

2.6、减量指令DEC

- 功能：指令对操作数减1
- 对标志位的影响：**不影响CF**，按定义设置其他状态标志

2.7、求补指令NEG

- 功能：对操作数执行求补运算：用零减去操作数，然后结果返回操作数；按位取反加1
- 对标志位的影响：对标志的影响与用0作减法的SUB指令一样

2.8、比较指令CMP

- 功能：将目的操作数减去源操作数，结果**不送回**目的操作数，只影响标志位
- 对标志位的影响：按照SUB指令定义相应设置状态标志

2.9、乘法指令

2.9.1、无符号数乘法MUL

- 字节乘法：其中一个乘数事先放在AL中，另一个乘数随意

MUL r8/m8

AX \leftarrow AL \times r8/m8

- 字乘法：其中一个乘数放在AX中，另一个乘数随意

MUL r16/m16

DX.AX \leftarrow AX \times r16/m16

- 对标志位的影响

若乘积的高一半 (AH或DX) 为0，则OF=CF=0，否则OF=CF=1；对其他标志没有定义

2.9.2、有符号数乘法

- 字节乘法

IMUL r8/m8

AX \leftarrow AL \times r8/m8

- 字乘法

IMUL r16/m16

DX.AX \leftarrow AX \times r16/m16

- 对标志位的影响

若乘积的高一半是低一半的符号扩展，则OF=CF=0；否则均为1；对其他标志没有定义

2.10、除法指令

- 除法指令对标志没有定义
- 除法指令会产生结果溢出（除法错中断）

2.10.1、无符号数除法

- 字节除法：被除数放在AX中，除数是一个字节

DIV r8/m8

AL \leftarrow AX \div r8/m8的商，AH \leftarrow AX \div r8/m8的余数

- 字除法：被除数放在DX.AX中，除数是一个字

DIV r16/m16

AX \leftarrow DX.AX \div r16/m16的商，DX \leftarrow DX.AX \div r16/m16的余数

2.10.2、有符号数除法

- 字节除法

IDIV r8/m8

AL \leftarrow AX \div r8/m8的商，AH \leftarrow AX \div r8/m8的余数

- 字除法

IDIV r16/m16

AX \leftarrow DX.AX \div r16/m16的商，DX \leftarrow DX.AX \div r16/m16的余数

2.11、符号扩展指令

- CBW

- AL的符号扩展至AH
- 如AL的最高有效位是0，则AH = 00
- AL的最高有效位为1，则AH = FFH。AL不变

```
1 mov al,80h ;al=80h
2 cbw          ;ax=ff80h
3 add al,255  ;al=7fh
4 cbw          ;ax=007fh
```

- CWD

- AX的符号扩展至DX
- 如AX的最高有效位是0，则DX = 00
- AX的最高有效位为1，则DX = FFFFH。AX不变

```
1 ;进行有符号数除法AX ÷ BX
2 CWD AX
3 IDIV BX
```

2.12、十进制调整指令

- 对二进制运算的结果进行十进制调整，以得到十进制的运算结果

2.12.1、压缩BCD码

- DAA：AL \leftarrow 将AL的加和调整为压缩BCD码

```

1 mov al,68h ; al=68h, 压缩BCD码表示真值68
2 mov bl,28h ; bl=28h, 压缩BCD码表示真值28
3 add al,bl ; 二进制加法: al=68h+28h=90h
4 daa         ; 十进制调整: al=96h

```

- DAS: AL←将AL的减差调整为压缩BCD码

```

1 mov al,68h ; al=68h, 压缩BCD码表示真值68
2 mov bl,28h ; bl=28h, 压缩BCD码表示真值28
3 sub al,bl ; 二进制减法: al=68h-28h=40h
4 das         ; 十进制调整: al=40h

```

- 使用DAA或DAS指令前, 应先执行以AL为目的操作数的加法或减法指令
- DAA和DAS指令对OF标志无定义, 按结果影响其他标志, 例如CF反映压缩BCD码相加或减的进位或借位状态

```

1 ;计算1234 - 4612
2 SUB AL,BL
3 DAS
4 XCHG AL,AH
5 SBB AL,BH
6 DAS
7 SCHG AL,AH

```

2.12.2、非压缩BCD码

- AAA: AL←将AL的加和调整为非压缩BCD码, AH←AH + 调整的进位

```

1 mov ax,0608h ; ax=0608h, 非压缩BCD码表示真值68
2 mov b1,09h   ; b1=09h, 非压缩BCD码表示真值9
3 add al,b1   ; 二进制加法: al=08h+09h=11h
4 aaa          ; 十进制调整: ax=0707h

```

- AAS: AL←将AL的减差调整为非压缩BCD码, AH←AH - 调整的借位

```

1 mov ax,0608h ; ax=0608h, 非压缩BCD码表示真值68
2 mov b1,09h   ; b1=09h, 非压缩BCD码表示真值9
3 sub al,b1   ; 二进制减法: al=08h-09h=ffh
4 aas          ; 十进制调整: ax=0509h

```

- AAM
- AAD
- 0~9非压缩BCD码与ASCII码的转换

```

1 ;ASCII码的0~9对应30H~39H
2 ;非压缩BCD码的0~9对应00H~09H
3
4 ;ASCII --> 非压缩BCD
5 AND AL,0FH
6
7 ;非压缩BCD --> ASCII
8 OR AL,30H

```

3、位操作类指令

3.1、逻辑运算指令

所有双操作数的逻辑指令均设置CF=OF=0，根据结果设置SF、ZF和PF状态，而对AF未定义

3.1.1、AND

- 可用于复位某些位（同0相与），不影响其他位

```
and b1, 11110110B
```

3.1.2、OR

- 可用于置位某些位（同1相或），不影响其他位

```
or b1, 00001001B
```

3.1.3、NOT

- 不影响标志位

3.1.4、XOR

- 相同结果为1，不同结果为0
- 可用于求反某些位（同1相异或），不影响其他位

```
xor b1, 00001001B
```

3.1.5、TEST

- 对两个操作数执行逻辑与运算，结果不回送到目的操作数

3.2、移位指令

3.2.1、算数左移SAL

- 算术左移，最高位进入CF，最低位补0

3.2.2、算数右移SAR

- 算术右移，最低位进入CF，最高位不变

3.2.3、逻辑左移SHL

- 逻辑左移，最高位进入CF，最低位补0

3.2.4、逻辑右移SHR

- 逻辑右移，最低位进入CF，最高位补0

3.3.5、对标志位的影响

- 按照移入的位设置进位标志CF
- 根据移位后的结果影响SF、ZF、PF
- 对AF没有定义
- 如果进行一位移动，如果移位前的操作数最高位与移位后操作数的最高位不同（有变化），则OF = 1；否则OF = 0。当移位次数大于1时，OF不确定

3.3、循环移位指令

3.3.1、不带进位循环左移ROL

3.3.2、不带进位循环右移ROR

3.3.3、带进位循环左移RCL

3.3.4、带进位循环右移RCR

3.3.5、对标志位的影响

- 按照指令功能设置进位标志CF
- 不影响SF、ZF、PF、AF
- 如果进行一位移动，如果移位前的操作数最高位与移位后操作数的最高位不同（有变化），则OF = 1；否则OF = 0。当移位次数大于1时，OF不确定

```
1 ;将DX.AX双字数据右移4位
2     MOV CL,4
3 AGAIN:SHR DX,1
4     RCR AX,1
5     DEC CL
6     JNZ AGAIN
```

4、控制转移类指令

通过改变IP（和CS）值，实现程序执行顺序的改变

4.1、无条件转移指令

- 段内转移（相对寻址、间接寻址）
 - 近转移
 - 在当前代码段±32KB范围内转移
 - 不需要更改CS段地址，只要改变IP偏移地址
 - 短转移
 - 在段内 - 128 ~ + 127位范围的转移
- 段间转移（间接寻址、直接寻址）
 - 远转移
 - 从当前代码段跳转到另一个代码段，可以在1MB范围
 - 更改CS段地址和IP偏移地址
 - 目标地址必须用一个32位数表达，叫做32位远指针，它就是逻辑地址

4.2、条件转移指令

操作数label是采用相对寻址方式的短转移标号

4.2.1、判断单个标志位状态

- JZ JNZ

```
1 ;如果AL最高位为0，设置AH=0；否则，设置AH=FFH
2     TEST AL,80H
3     JZ NEXT
4     MOV AH,0FFH
5     JMP DONE
6 NEXT:MOV AH,0
7 DONE:...
```

- JS JNS

```

1 ;计算|X-Y|, X和Y为存放于X和Y单元的16位操作数, 结果存入result中
2     MOV AX,X
3     SUB AX,Y
4     JNS NONNEG
5     NEG AX
6 NONNEG:MOV RESULT,AX

```

- JP JNP
- JO JNO

```

1 ;计算X-Y, X和Y为存放于X和Y单元的16位操作数。若溢出, 则转移到overflow处理
2     MOV AX,X
3     SUB AX,Y
4     JO OVERFLOW
5     ...
6 OVERFLOW:...

```

- JC JNC

```

1 ;记录BX中"1"的个数
2     XOR AL,AL          ;AL保存1的个数
3 AGAIN:TEXT BX,0FFFFH
4     JE NEXT
5     SHL BX,1
6     JNC AGAIN
7     INC AL
8     JMP AGAIN
9 NEXT:...

```

4.2.2、比较无符号数高低

- 低于(不高于等于) : JB (JNAE)
- 不低于(高于等于) : JNB (JAE)
- 低于等于(不高于) : JBE (JNA)
- 不低于等于(高于) : JNBE (JA)

4.2.3、比较有符号数大小

- 小于(不大于等于) : JL (JNGE)
- 不小于(大于等于) : JNL (JGE)
- 小于等于(不大于) : JLE (JNG)
- 不小于等于(大于) : JNLE (JG)

4.3、循环指令

- label操作数采用**相对短转移寻址方式**
- 循环指令默认利用CX计数器
- JCXZ label ; CX=0, 转移到标号label
- LOOP label ; CX←CX - 1; CX≠0, 循环到标号label
- LOOPZ label ; CX←CX - 1; CX≠0且ZF=1, 循环到标号label
- LOOPNZ label ; CX←CX - 1; CX≠0且ZF=0, 循环到标号label

4.4、子程序指令

主程序采用 CALL 调用指令转移到该子程序的起始处执行，运行完子程序后，采用 RET 返回指令回到主程序继续执行

- 段内调用

入栈偏移地址IP

$SP \leftarrow SP - 2, SS:[SP] \leftarrow IP$

- 段间调用

入栈偏移地址IP和段地址CS

$SP \leftarrow SP - 2, SS:[SP] \leftarrow IP$

$SP \leftarrow SP - 2, SS:[SP] \leftarrow CS$

- 无参返回

- 有参返回

`RET i16 ; 堆栈指针SP将增加，即 $SP \leftarrow SP + i16$`

4.5、中断指令

8086可以管理256个中断，各种中断用一个**向量编号**来区别

- 外部中断

- 可屏蔽中断

- 可以在CPU内部被屏蔽掉，由中断允许标志IF控制
 - 执行中断服务程序或关中断指令CLI，都使IF = 0
 - 执行开中断指令STI将使IF = 1
 - 除可屏蔽中断外其余中断不受IF控制

- 非屏蔽中断：中断向量号为02H

- 内部中断（异常）

- 除法错中断：除数为0或商超过范围
 - **指令中断**：调用INT i8
 - 断点中断：用于断点调试INT 3的3号中断
 - 溢出中断：OF = 1产生的4号中断
 - 单步中断：TF = 1在每条指令执行后产生的1号中断

- 中断过程

1. 标志寄存器入栈保存
2. 禁止新的可屏蔽中断和单步中断：IF = TF = 0
3. 断电地址入栈保存
4. 读取中断服务程序的起始地址

三、汇编语言程序格式

- 硬指令和伪指令

- 硬指令：使CPU产生动作、并在程序执行时才处理的语句

- 伪指令：不产生CPU动作、在程序执行前由汇编程序处理的说明性语句

1、汇编语言程序的开发

1.1、汇编语言程序的语句格式

1.1.1、执行性语句

- 由**硬指令**构成的语句，它通常对应一条**机器指令**，出现在程序的**代码段**中

标号：硬指令助记符 操作数,操作数 ... ; 注释

- 标号是反映硬指令位置(**逻辑地址**)的标识符，为分支、循环等指令提供转移的目的地址
- 硬指令助记符可以是**处理器指令**和**宏指令**

1.1.2、说明性语句

- 伪指令**构成的语句，它通常指示汇编程序如何汇编源程序

名字 伪指令助记符 参数,参数,... ; 注释

- 名字是反映伪指令位置(**逻辑地址**)和**属性**的标识符

1.2、汇编语言的程序格式

- 完整的汇编语言源程序由**段**组成
- 所有的**可执行性语句**必须位于某一个**代码段内**，说明性语句可根据需要位于任一段内

1.2.1、简化段定义格式

```

1 .model small ;定义程序的存储模型
2 .stack ;定义堆栈段
3 .data ;定义数据段
4     ... ;定义数据
5 .code ;定义代码段
6 .startup ;程序起始点，建立CS、DS、SS
7     ... ;指令序列
8 .exit 0 ;程序结束点，返回DOS
9     ... ;子程序代码
10 end ;汇编结束

```

1.2.2、完整段定义格式

```

1 stack segment stack ;定义堆栈段
2     ... ;定义堆栈段大小
3 stack ends ;堆栈段结束
4 data segment ;定义数据段
5     ... ;定义数据
6 data ends
7 code segment 'code' ;定义代码段
8     assume cs:code,ds:data,ss:stack ;只是声明，并没有赋值，要靠后面的两句为DS赋值
9 start: mov ax,data ;建立DS段地址
10    mov ds,ax
11    ... ;指令序列
12    mov ax,4c00h
13    int 21h
14    ... ;子程序代码
15 code ends ;代码段结束
16    end start ;汇编结束，同时指明程序起始点
17

```

1.3、汇编语言程序的开发过程

- 编辑-----.asm文件
- 汇编-----.obj文件

将源程序翻译成由机器代码组成的目标模块文件 (.obj)

3. 连接-----.exe文件

把一个或多个目标文件和库文件合成一个可执行程序 (.exe/.com)

4. 调试

1.4、DOS系统功能的调用

- DOS提供编程资源以**中断调用**的方法使用各种子程序，Windows以**应用程序接口API**的形式提供动态链接库DLL
- **21H**号中断是DOS提供给用户的用于**调用系统功能**的中断
- 功能调用的步骤
 1. 在**AH**寄存器中设置**系统功能调用号**
 2. 在指定寄存器中设置**入口参数**
 3. 执行指令**INT 21H**实现中断服务程序的功能调用
 4. 根据**出口参数**分析功能调用执行情况

1.4.1、字符输出功能调用

- 功能号: 02H
- 入口参数: DL

```
1 mov ah,02h ; 设置功能号: ah←02h
2 mov dl,'?' ; 提供入口参数: dl←'?'
3 int 21h      ; DOS功能调用: 显示
```

1.4.2、字符串输出功能调用

- 功能号: 09H
- 入口参数: DS:DX

```
1 string db 'Hello,Everybody !',0dh,0ah,'$'    ; 在数据段定义要显示的字符串
2 ...
3 mov ah,09h          ; 设置功能号: ah←09h
4 mov dx,offset string ; 提供入口参数: dx←字符串的偏移地址
5 int 21h            ; DOS功能调用: 显示
```

1.4.3、字符输入功能调用

- 功能号: 01H
- 出口参数: AL

```
1 getkey: mov ah,01h ; 功能号: ah←01h
2     int 21h        ; 功能调用
3     cmp al,'Y'     ; 处理出口参数al
4     je yeskey      ; 是“Y”
5     cmp al,'N'
6     je nokey       ; 是“N”
7     jne getkey
```

1.4.4、字符串输入功能调用

- 功能号: 0AH
- 入口参数: 首先建立缓冲区，首字节为最大字符数，第二字节为实际字符数，后面为字符串数据，DS:DX指向缓冲区首地址

- 出口参数: DS:DX

```

1 buffer db 81      ;定义缓冲区, 第1个字节填入可能输入的最大字符数
2 db 0              ;存放实际输入的字符数
3 db 81 dup(0)      ;存放输入的字符串
4 ...
5 mov dx,seg buffer ;伪指令seg取得buffer的段地址
6 mov ds,dx          ;设置数据段DS
7 mov dx,offset buffer
8 mov ah,0ah
9 int 21h

```

1.4.5、按键判断功能调用

- 功能号: AH = 0BH
- 出口参数
 - AL = 0, 当前没有按键
 - AL = FFH, 当前已经按键。

```

1 getkey: mov ah,0bh
2     int 21h
3     or al,al    ;al=0
4     jz getkey

```

2、参数、变量和符号

- 数值型参数: 常数、表达式
- 地址性参数: 标号、名字

2.1、数值型参数

2.1.1、数值常数(N进制)

2.1.2、符号常数

- 符号常数定义方法
 - 符号名 EQU 数值表达式
 - 符号名 EQU <字符串>

```

1 ;定义
2 DosWriteChar equ 2
3 CarriageReturn = 13
4 callDOS equ <int 21h>
5 ;应用
6 mov ah,2      ;mov ah,DosWriteChar
7 mov dl,13    ;mov dl,CarriageReturn
8 int 21h      ;callDOS

```

2.1.3、字符串常数

2.2、数值表达式

- 由运算符连接的各种常数所构成的表达式
- 在汇编过程中计算表达式, 最终得到一个数值, 程序运行之前, 就已经计算出了表达式; 所以, 程序运行速度没有变慢, 但增强程序的可读性

- 注意！因为数值表达式是在汇编过程中计算的，因此不能包含在程序运行时才能得知的项

2.3、变量定义伪指令

- 格式： 变量名 伪指令助记符 初值表
- 变量名：表示初值表首元素的逻辑地址
- 伪指令助记符
 - DB：字节
 - DW：字
 - DD：双字
 - DF：二字
 - DQ：四字
 - DT：十字节
- 初值表
 - 数值、'字符串'、'字符'
 - ? (表示初值不确定)
 - 重复次数 DUP (重复参数)
- 定位伪指令ORG：下一条数据将以ORG后面的参数所表示的地址为首地址

1	ORG 100h ;从100h处安排数据或程序
2	ORG \$+10 ;使偏移地址加10，“\$”表示当前偏移地址值

2.4、变量和标号的属性

2.4.1、地址属性

- 标号和名字对应存储单元的逻辑地址，包括段地址和段内偏移地址

2.4.2、类型属性

- 标号、子程序名的类型可以是NEAR (近) 和FAR (远)，分别表示段内或段间
- 变量名的类型可以是BYTE (字节)、WORD (字) 和DWORD (双字) 等

2.4.3、地址操作符

- 取得名字或标号的段地址和偏移地址两个属性
- [] 将括起的表达式作为存储器地址
- \$ 当前偏移地址
- : 采用指定的段地址寄存器

1	OFFSET 名字/标号 ;返回名字或标号的偏移地址
2	SEG 名字/标号 ;返回名字或标号的段地址

2.4.4、类型操作符

- 类型名 PTR 名字/标号

作用：临时改变名字或标号的类型

类型名：BYTE/WORD/DWORD/FWORD/QWORD/TBYTE/NEAR/FAR

- THIS 类型名

为同一片存储单元取不同类型的变量名

```

1 A EQU THIS BYTE
2 B DW 4321H, 2255H
3 MOV AL,A      ;等价于MOV AL,byte ptr LABD

```

- TYPE 名字/标号

格式	功能
TYPE 变量名	返回该变量每个数据占用的字节数 (DB:1、DW:2、DD:4)
TYPE 标号	返回距离属性值 (SHORT:FF01H、NEAR:FF02H、FAR:FF05H)

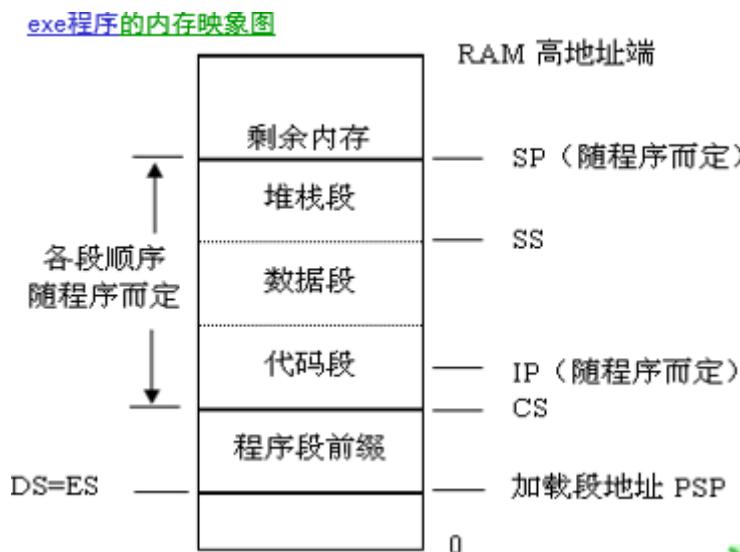
- LENGTHOF 名字
 - 返回数据项数
- SIZEOF 名字
 - 返回整个数据所占用的字节数
- $\text{SIZEOF} = \text{LENGTHOF} \times \text{TYPE}$

3. 程序段的定义和属性

3.1、DOS的程序结构

3.1.1、exe程序结构

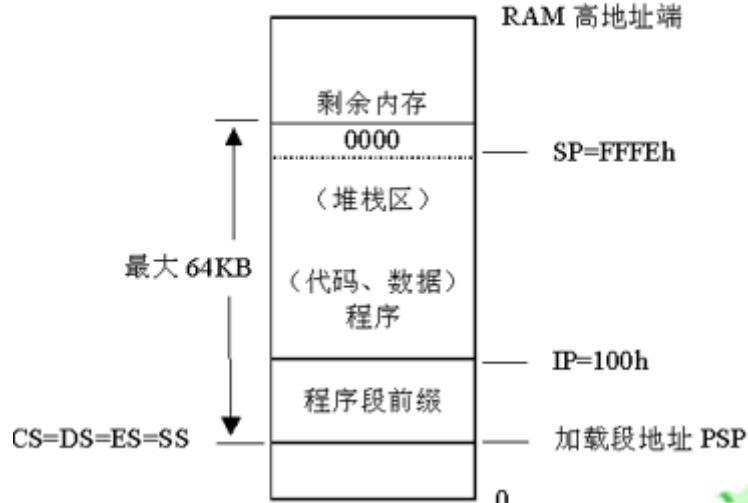
- 可以有独立的代码、数据和堆栈段，还可以有多个代码段或多个数据段，程序长度可以超过 64KB，执行起始处可以任意指定
- 当DOS装入或执行一个程序时，DOS确定当时主存最低的可用地址作为该程序的装入起始点。此点以下的区域称为程序段。在程序段内偏移0处，DOS为该程序建立一个程序段前缀控制块 PSP(Program Segment Prefix)，它占256(100h)个字节；而在偏移100h处才装入程序本身



3.1.2、com程序结构

- 将代码、数据和堆栈段合一，不超过64KB
- 在程序开发时，需要满足一定要求并采用相应参数才能正确生成COM结构的程序
- 与EXE文件相比其加载速度更快，占用的磁盘空间更少
- DOS也为COM程序建立程序段前缀PSP，但由于两种文件结构不同，所以加载到主存后各段设置并不完全一样

com程序的内存映象图



3.2、简化段定义格式

3.2.1、.MODEL存储模型

- 使用简化段定义，必须有存储模型伪指令
- .model语句必须位于所有段定义语句之前
- 七种类型：TINY、SMALL、MEDIUM、HUGE、COMPACT、LARGE、FLAT
- TINY微型模型
 - 用于创建**COM**类型程序
 - 代码段、数据段、堆栈段都在同一个段内，不大于64KB
 - 访问操作数或指令都只需要使用16位偏移地址
 - 调用类型和数据指针缺省分别为**近调用**和**近指针**
- SMALL小型模型
 - 至多只能有一个代码段和一个数据段，每段不大于64KB
 - 数据段是指数据段、堆栈段和附加段的总和，它们共用同一个段基址，总长度不可超过64KB；因此小型模式下程序的最大长度为128KB
 - 调用类型和数据指针缺省分别为**近调用**和**近指针**
- COMPACT紧凑模型
 - 适合于**数据量大但代码量小**的程序
 - **代码段**被限制在一个不大于64KB的段内；而**数据段**则可以有多个，超过64KB
 - 调用类型缺省仍为**近调用**；而数据指针缺省为**远指针**，因为必须用段地址区别多个数据段
- MEDIUM中型模型
 - 与紧凑模型互补的模式
 - 适合于**数据量小但代码量大**的程序
 - **代码段**可以超过64KB，有多个；但**数据段**只能有一个不大于64KB的段
 - 数据指针缺省为**近指针**；但调用类型缺省是**远调用**，因为要利用段地址区别多个代码段
- LARGE大型模型
 - 代码段和数据段都有多个，都可以超过64KB；但全部的静态数据（不能改变的数据）仍限制在64K字节内
 - 调用类型和数据指针缺省分别为**远调用**和**远指针**
- HUGE巨型模型
 - 与大型模型基本相同，只是静态数据不再被限制在64K字节之内
- FLAT平展模型
 - 用于创建一个**32位的程序**
 - 它只能运行在32位x86 CPU上

- DOS下不能使用FLAT模型

3.2.2、堆栈段伪指令

.STACK [大小]

- 段名是： stack
- 默认是1KB

3.2.3、数据段伪指令

.DATA

- 段名是： _DATA

.DATA?

- 创建无初值变量
- 段名是： _BSS

.CONST

- 建立只读的常量数据段
- 段名是： CONST

3.2.4、代码段伪指令

.CODE [段名]

- 如果没有给出段名，默认的代码段名是： _TEXT

3.2.5、程序开始伪指令

.STARTUP

- 按照CPU类型、存储模型、操作系统和堆栈类型，**产生程序开始执行的代码；同时还指定程序开始执行的起始点**（设置了DS的值）
- 在DOS下，还将设置DS值，调整SS和SP值

3.2.6、程序终止伪指令

.EXIT [返回参数]

- .exit 0对应的代码是：
`mov ax,4c00h
int 21h`

3.2.7、汇编结束伪指令

END [标号]

- 源程序的最后必须有一条END语句
- 标号用于指定程序开始执行点，连接程序将据此设置CS : IP值
- 采用了.startup伪指令就不需要再用“end 标号”指明开始执行点，但还要有end伪指令

3.3、完整段定义格式

3.3.1、段定义属性

```

1 [段名] segment [定位] [组合] [段字] '类别'
2     ...;语句序列
3 [段名] ends
4
5     ;堆栈段要采用stack组合类型
6     ;数据段要有'code'类别
7     ;如果不指定则采用默认值，指定必须按照顺序

```

- 段定位属性
 - BYTE 段开始为下一个可用的字节地址 (xxxx xxxx0b)
 - WORD 段开始为下一个可用的偶数地址 (xxxx xxxx0b)
 - DWORD 段开始为下一个可用的4倍数地址 (xxxxxx00b)
 - PARA 段开始为下一个可用的节地址 (xxxx 0000b)
 - PAGE 段开始为下一个可用的页地址 (0000 0000b)
 - **简化段**定义伪指令的**代码和数据段**默认采用**WORD**定位，**堆栈段**默认采用**PARA**定位
 - **完整段**定义伪指令的默认定位属性是**PARA**
- 段组合属性：指定多个逻辑段之间的关系
 - PRIVATE：与其他段没有逻辑关系，每段都有自己的段地址，不与其他段合并，**完整段**的默认段组合方式
 - PUBLIC：与所有同名同类的其他段相邻的连接在一起，制定一个**共同的段地址**，也就是**合成一个物理段**，**简化段**的默认段组合方式
 - STACK：**堆栈段必须具有的**段组合，与PUBLIC段具有相同的方式合并
- 段字属性：为了支持32位段
- 段类别属性
 - 连接程序组织段时，将所有的同类别段相邻分配
 - 段类别可以是任意名称，但必须位于**单引号**中
 - 大多数MASM程序使用 '**code**'、'**data**'和'**stack**'来分别指名代码段、数据段和堆栈段

3.3.2、ASSUME伪指令

- 功能：通知MASM用指定的段寄存器来寻址对应的逻辑段
- 并不为段寄存器设定初值，连接程序LINK将正确设置CS : IP和SS : SP
- 需要手动为DS赋值；如果使用附加段，还要赋值ES

```

1 stack    segment stack
2     dw 512 dup(?)
3 stack    ends
4 data    segment
5     ...           ;在数据段定义数据
6 data    ends
7 code    segment 'code'
8     assume cs:code,ds:data,ss:stack
9 start:  mov ax,data
10    mov ds,ax
11    ...           ;在代码段填入指令序列
12    mov ax,4c00h
13    int 21h
14    ...           ;子程序代码
15 code    ends
16 end start

```

四、基本汇编语言程序设计

1、顺序程序设计

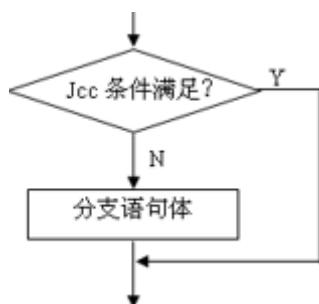
按指令书写的前后顺序执行每一条指令

```
1 ;查表法，实现一位16进制数转换为ASCII码显示
2 .model small
3 .stack
4 .data
5     ASCII    db 30h,31h,32h,33h,34h,35h
6                 db 36h,37h,38h,39h          ;0~9的ASCII码
7                 db 41h,42h,43h,44h,45h,46h ;A~F的ASCII码
8     hex      db 0bh                  ;任意设定了一个待转换的一位16进制数
9 .code
10 .startup
11     mov bx,offset ASCII ;BX指向ASCII码表
12     mov al,hex           ;AL取得一位16进制数，正是ASCII码表中位移
13     and al,0fh           ;只有低4位是有效的，高4位清0
14     xlat                ;换码： AL←DS:[BX+AL]
15     mov dl,al             ;入口参数： DL←AL
16     mov ah,2               ;02号DOS功能调用
17     int 21h               ;显示一个ASCII码字符
18 .exit 0
19 end
```

2、分支程序设计

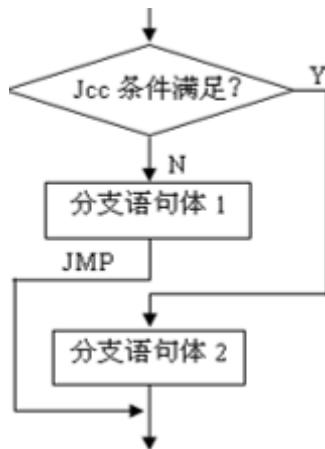
2.1、单分支程序设计

条件成立跳转，否则顺序执行分支语句体



```
1 ;将AX中存放的无符号数除以2，如果是奇数，则加1后除以2
2     test ax,01h ;测试AX最低位
3     jz even      ;最低位为0： AX为偶数
4     add ax,1     ;最低位为1： AX为奇数，需要加1
5 even: rcr ax,1 ;AX←AX÷2
6     ;如果采用SHR指令，则不能处理AX=FFFFH的特殊情况
```

2.2、双分支程序设计



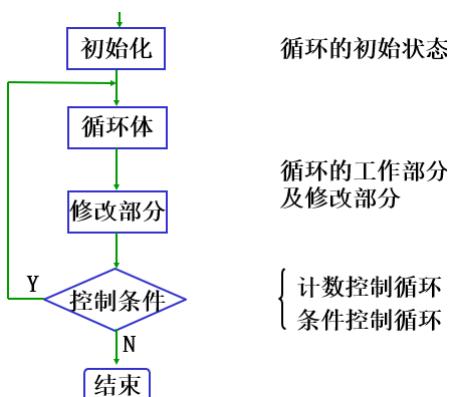
```

1 ;显示BX最高位-3
2     mov dl,'0'      ;DL←'0'
3     shl bx,1        ;BX最高位移入CF
4     jnc two          ;CF=0, 最高位为0, 转移
5     mov dl,'1'      ;CF=1, 最高位为1, DL←'1'
6 two:   mov ah,2
7     int 21h          ;显示

```

2.3、多分支程序设计

3、循环程序设计



- 三部分
 - 循环初始部分：设置循环次数、循环体需要的初值
 - 循环体部分
 - 循环控制部分：判断循环条件是否成立

```

1 ;用二进制显示从键盘输入的一个字符的ASCII码
2 .MODEL SMALL
3 .STACK
4 .DATA
5 .CODE
6 .STARTUP
7     mov ah,1      ;从键盘输入一个字符
8     int 21h
9     mov bl,al      ;DOS功能会改变AL内容，故字符ASCII码存入BL
10    mov ah,2
11    mov dl,':'
12    int 21h
13    mov cx,8      ;CX←8 (循环次数)
14 again:  shl bl,1 ;左移进CF，从高位开始显示

```

```

15    mov dl,0          ;MOV指令不改变CF
16    adc dl,30h       ;DL←0+30H+CF, CF若是0, 则DL←'0', 若是1, 则DL←'1'
17    mov ah,2
18    int 21h          ;显示
19    loop again       ;CX减1, 如果CX未减至0, 则循环
20 .EXIT 0
21 END

```

```

1 ;冒泡排序
2 mov cx,count           ;CX←数组元素个数
3 dec cx                 ;元素个数减1为外循环次数
4 outlp: mov dx,cx      ;DX←内循环次数
5     mov bx,offset array
6 inlp:  mov al,[bx]      ;取前一个元素
7     cmp al,[bx+1]        ;与后一个元素比较
8     jna next             ;前一个不大于后一个元素, 则不进行交换
9     xchg al,[bx+1]        ;否则, 进行交换
10    mov [bx],al
11 next: inc bx          ;下一对元素
12    dec dx
13    jnz inlp            ;内循环尾
14    loop outlp           ;外循环尾

```

```

1 ;现有一个以$结尾的字符串, 要求剔除其中的空格
2 .data
3 string db 'Let us have a try !','$'
4 .code
5 .startup
6     mov si,offset string
7 outlp: cmp byte ptr [di],'$'   ;外循环, 先判断后循环
8     jz done                ;为$结束
9     cmp byte ptr [si],'
10    jnz next               ;不是空格继续循环
11    mov di,si               ;是空格, 进入剔除空格分支
12    inlp: inc di           ;该分支是循环程序段
13        mov al,[di]
14        mov [di-1],al
15        cmp byte ptr [di],'$'   ;内循环, 先循环后判断
16        jnz inlp
17        jmp outlp
18 next: inc si              ;继续对后续字符进行处理
19        jmp outlp
20 done: .exit 0             ;结束

```

4、串操作指令

4.1、串数据类型

- 数据串：在连续的主存区域中，字节或字的序列

4.2、串寻址方式

- 源操作数用寄存器SI寻址，默认在数据段DS中，即DS:[SI]
- 目的操作数用寄存器DI寻址，默认在附加段ES中，即ES:[DI]
- 每执行一次串操作指令，SI和DI将自动修改（需要在串操作指令前执行CLD/STD）

- 执行指令CLD后, DF = 0, 地址指针+1或2 (正序操作)
- 执行指令STD后, DF = 1, 地址指针 -1或2 (倒序操作)

4.3、串操作指令

4.3.1、串传送MOVS

把字节或字操作数从主存的源地址传送至目的地址

- MOVSB

1	ES:[DI] <= DS:[SI]
2	SI <= SI±1
3	DI <= DI±1

- MOVSW

1	ES:[DI] <= DS:[SI]
2	SI <= SI±2
3	DI <= DI±2

4.3.2、串存储STOS

- STOSB

1	ES:[DI] <= AL
2	DI <= DI±1

- STOSW

1	ES:[DI] <= AX
2	DI <= DI±2

4.3.3、串读取LODS

- LODSB

1	AL <= DS:[SI]
2	SI <= SI±1

- LODSW

1	AX <= DS:[SI]
2	SI <= SI±2

4.3.4、串比较CMPS

- CMPSB

1	DS:[SI] -- ES:[DI]
2	SI <= SI±1
3	DI <= DI±1

- CMPSW

```
1 DS:[SI] -- ES:[DI]  
2 SI <= SI±2  
3 DI <= DI±2
```

4.3.5、串扫描SCAS

- SCASB

```
1 AL -- ES:[DI]  
2 DI <= DI±1
```

- SCASW

```
1 AX -- ES:[DI]  
2 DI <= DI±2
```

4.3.4、重复前缀指令

串操作指令前，都可以加一个重复前缀，实现串操作的重复执行。重复次数隐含在CX寄存器中

- REP

配合MOVS、STOS、LODS使用，当数据串没有结束(CX ≠ 0)，则继续执行

```
1 ;字符串传送  
2 mov ax,ds  
3     mov es,ax    ;设置附加段ES=DS  
4     mov si,offset srcmsg  
5     mov di,offset dstmsg  
6     mov cx,lengthof srcmsg  
7     cld          ;将标志位DF清零，使SI、DI自动增加，字串处理由前往后  
8     rep movsb    ;传送字符串  
9     mov ah,9      ;显示字符串  
10    mov dx,offset dstmsg  
11    int 21h
```

- REPZ

配合CMPS、SCAS使用，当数据串没有结束(CX ≠ 0)，并且串相等(ZF = 1)，则继续比较

```
1 ;字符串比较  
2 MOV CX LENGTHOF STRING1  
3 MOV SI,OFFSET STRING1  
4 MOV DI,OFFSET STRING2  
5 CLD  
6 REPZ CMPSB  
7 JNZ NZ  
8 MOV DL,'Y'  
9 JMP OUTPUT  
10 NZ:MOV DL,'N'  
11 OUTPUT:...
```

```
1 ;查找字符串
2 MOV DI,OFFSET STRING
3 MOV BL, ' '
4 MOV CX, LENGTHOF STRING
5 CLD
6 REP NZ SCASB
7 JZ FOUND
8 ...           ;没找到
9 FOUND:...     ;找到了
```

4、子程序设计

4.1、常见格式

```
1 subname proc [near|far]
2     push ax      ;保护寄存器：顺序压入堆栈
3     push bx      ;ax/bx/cx仅是示例
4     push cx
5     ...          ;过程体
6     pop cx       ;恢复寄存器：逆序弹出堆栈
7     pop bx
8     pop ax
9     ret          ;过程返回
10    subname endp ;过程结束
```

```
1 ;将AL低4位表达的一位16进制数转换为ASCII码
2 HTOASC proc
3     and al,0fh
4     cmp al,9
5     jbe htoasc1
6     add al,37h      ;是0AH~0FH, 加37H
7     ret          ;子程序返回
8 htoasc1:   add al,30h ;是0~9, 加30H
9     ret          ;子程序返回
10    HTOASC endp
```

4.2、子程序的参数传递

- 入口参数：主程序提供数据给子程序
- 出口参数：子程序返回数据给主程序
- 参数的形式
 - 传值
 - 传址：适合连续存储的数据
- 传递的方法
 - 寄存器
 - 对带有出口参数的寄存器不能保护和恢复
 - 变量
 - 采用同一个变量名共享同一个变量，实现参数的传递
 - 不同模块间共享时，需要利用PUBLIC/EXTERN声明
 - 堆栈
 - 主程序将子程序的入口参数压入堆栈，子程序从堆栈中取出参数

- 子程序将出口参数压入堆栈，主程序弹出堆栈取得它们

```

1 | add sp,n      ;主程序实现平衡堆栈
2 | ret n        ;子程序实现平衡堆栈

```

4.3、子程序的嵌套

- 嵌套深度受堆栈空间的限制，注意寄存器的保护和恢复

4.4、子程序的递归

- 必须采用**寄存器或堆栈**传递参数，递归深度受堆栈空间的限制

```

1 | ;计算N!
2 | .model small
3 | .stack
4 | .data
5 |     N      dw 3
6 |     result dw ?
7 | .code
8 | .startup
9 |     mov bx,N
10 |    push bx      ;入口参数: N
11 |    call fact    ;调用递归子程序
12 |    pop result   ;出口参数: N!
13 | .exit 0
14 | ;入口参数: 压入N
15 | ;出口参数: 弹出N!
16 fact proc
17 |     push ax
18 |     push bp
19 |     mov bp,sp
20 |     mov ax,[bp+6]      ;取入口参数 N
21 |     cmp ax,0
22 |     jne fact1          ;N>0,N!=Nx(N-1) !
23 |     inc ax              ;N=0,N!=1
24 |     jmp fact2
25 fact1: dec ax           ;N-1
26 |     push ax
27 |     call fact          ;调用递归子程序求(N-1) !
28 |     pop ax
29 |     mul word ptr [bp+6] ;求 Nx(N-1) !
30 fact2: mov [bp+6],ax    ;存入出口参数 N!
31 |     pop bp
32 |     pop ax
33 |     ret
34 fact endp
35 end

```

4.5、子程序的重入

- 子程序被中断后又被中断服务程序所调用
- 利用**寄存器和堆栈**存放参数和临时数据，而不要使用固定的存储单元(变量)，就能够实现重入
- 重入是**被动地**进入，而递归是**主动地**进入
- 重入的调用间往往没有关系，而递归的调用间却是密切相关的
- 递归子程序也是可重入子程序（因为递归程序只能采用**寄存器或堆栈**传递参数）

- DOS功能调用不可重入

```
1 .data
2     count    = 10
3     array    dw count dup(0) ;预留数据存储空间
4 .code
5 .startup
6     mov cx, count
7     mov bx, offset array
8 again: call read ;调用子程序输入一个数据
9     mov [bx], ax ;将出口参数存放缓冲区
10    inc bx
11    inc bx
12    call dpcrlf ;调用子程序，光标回车换行以便输入下一个数据
13    loop again
14 .exit 0
15 ;输入有符号10进制数的通用子程序
16 ;出口参数：AX=补码表示的二进制数值
17 ;说明：负数用“-”引导，正数用“+”引导或直接输入；数据范围是+32767~-32768
18 read    proc
19     push bx
20     push cx
21     push dx
22     xor bx, bx ;BX保存结果
23     xor cx, cx ;CX为正负标志，0为正，-1为负
24     mov ah, 1 ;输入一个字符
25     int 21h
26     cmp al, '+' ;是“+”，继续输入字符
27     jz read1
28     cmp al, '-' ;是“-”，设置-1标志
29     jnz read2 ;非“+”和“-”，转read2
30     mov cx, -1
31 read1: mov ah, 1 ;继续输入字符
32     int 21h
33 read2: cmp al, '0'
34     jb read3 ;不是0~9之间的字符，则输入数据结束
35     cmp al, '9'
36     ja read3
37     sub al, 30h ;是0~9之间的字符，则转换为二进制数
38     shl bx, 1 ;利用移位指令，实现数值乘10：BX←BXX×10
39     mov dx, bx
40     shl bx, 1
41     shl bx, 1
42     add bx, dx
43     mov ah, 0
44     add bx, ax ;已输入数值乘10后，与新输入数值相加
45     jmp read1 ;继续输入字符
46 read3: cmp cx, 0
47     jz read4
48     neg bx ;是负数，进行求补
49 read4: mov ax, bx ;设置出口参数
50     pop dx
51     pop cx
52     pop bx
53     ret ;子程序返回
54 read    endp
55 ;使光标回车换行的子程序
```

```
56 dpcrlf proc  
57     ... ;省略  
58 dpcrlf endp  
59 end
```

4.6、ASCII码与二进制数的转换

4.6.1、ASCII码转为二进制数

1. 首先判断输入为正或负数，并用一个寄存器记录
2. 接着输入0~9数字（ASCII码），并减30H转换为二进制数
3. 然后将前面输入的数值乘10，并与刚输入的数字相加得到新的数值
4. 重复2、3步，直到输入一个非数字字符结束
5. 负数进行求补，转换成补码；否则直接保存数值

4.6.2、二进制数转为ASCII码

1. 首先判断数据是零、正数或负数，是零显示“0”退出
2. 是负数，显示“-”，求数据的绝对值
3. 接着数据除以10，余数加30H转换为ASCII码压入堆栈
4. 重复③步，直到商为0结束
5. 依次从堆栈弹出各位数字，进行显示

4.7、有符号二进制数求和如何避免溢出

- 被加数要进行符号扩展，16位二进制数表示数据个数，最大是 2^{16} ，这样扩展到32位二进制数表达累加和，不再会出现溢出

五、高级汇编语言程序设计

1、高级语言特性

1.1、条件控制伪指令

- .IF
- .ELSEIF
- .ELSE
- .ENDIF

条件表达式中的变量（用DB、DW、DD等声明）、寄存器、常数默认是无符号数；若采用有符号数，则可以利用SBYTE PTR或SWORD PTR操作符指明，变量可用SBYTE、SWORD声明

```
1 ;有根判断  
2 .MODEL SMALL  
3 .STACK  
4 .DATA  
5     _A SBYTE ?  
6     _B SBYTE ?  
7     _C SBYTE ?  
8 .DATA  
9     XOR AX,AX  
10    MOV AL,_B  
11    IMUL AL  
12    MOV BX,AX  
13    MOV AL,_A
```

```
14     IMUL _C
15     MOV CX,4
16     IMUL CX
17     .IF SWORD PTR BX >= AX
18         MOV TAG,1
19     .ELSE
20         MOV TAG,0
21     .ENDIF
22 .EXIT 0
23 END
```

1.2、循环控制伪指令

1.
 - .WHILE
 - .ENDW

```
1 ... ;预置参数
2 .WHILE ;条件表达式
3     ...
4 .ENDW
5
6 ;1~100的和
7 XOR AX,AX
8 MOV CX,100
9 .WHILE CX > 0
10    ADD AX,CX
11    DEC CX
12 .ENDW
13 MOV SUM,AX
```

2.
 - .REPEAT
 - .UNTIL

```
1 ... ;预置参数
2 .REPEAT
3     ...
4 .UNTIL ... ;条件表达式
5
6 ;1~100的和
7 XOR AX,AX
8 MOV CX,100
9 .REPEAT
10    ADD AX,CX
11    DEC CX
12 .UNTIL CX == 0
```

3.
 - .REPEAT
 - .UNTILCXZ (直到条件表达式满足或CX = 0)
4.
 - .CONTINUE
 - .BREAK

```
1 ;array是100字数组, 计算前若干个非负数之和, 并存入result
2 .MODEL SMALL
3 .STACK
4 .DATA
```

```

5     RESULT DW ?
6     .CODE
7     .STARTUP
8     XOR AX,AX
9     MOV CX,100
10    LEA BX,ARRAY
11    .WHILE CX > 0
12        .IF SWORD PTR [BX] >= 0
13            ADD AX,[BX]
14        .ELSE
15            .BREAK
16        .ENDIF
17        INC BX
18        INC BX
19        DEC CX
20    .ENDW

```

1.3、过程声明和过程调用伪指令

- 过程声明伪指令

过程名 PROTO [调用距离] [语言类型] [,参数:[类型]]

- 过程定义伪指令

```

1 | 过程名    PROC  [调用距离] [语言类型] [作用范围] [<起始参数>] [USES寄存器列表]
2 |           [,参数:[类型]]
3 |           LOCAL 参数表      ;声明局部变量
4 |           ...             ;汇编语言语句
|   过程名    ENDP

```

- 调用距离: NEAR、FAR; 默认值由.MODEL伪指令指定
- 语言类型: 可以由.MODEL伪指令指定
- 作用范围: PUBLIC、PRIVATE、EXPORT; 默认是PUBLIC
- 起始参数: 自动创建过程的起始代码和收尾代码, 用于传递堆栈参数及清除堆栈等
- 寄存器列表: 通用寄存器名, 空格分隔, 保护寄存器(入栈出栈)
- 参数:类型: 表示该过程使用的形参及其类型; 16位段默认是WORD, 逗号分隔
- LOCAL: 定义局部变量, 利用堆栈存放

LOCAL 变量名[个数][:类型][, ...]

- 过程调用伪指令

INVOKE 过程名[,参数,...]

```

1 | .MODEL SMALL
2 | CHECKSUMD PROTO C,:WORD,:WORD
3 | .STACK
4 | .DATA
5 |     COUNT EQU 10
6 |     ARRAY DB 12H,25H,0F0H,0A3H,3,68H,71H,0CAH,0FFH,90H
7 |     RESULT ?
8 | .CODE
9 |     INVOKE CHECKSUMD,COUNT,OFFSET ARRAY
10    MOV RESULT,AL
11    .EXIT 0

```

```
12
13 CHECKSUMD PROTO C USES BX CX,COUNTP:WORD,ARRAYP:WORD
14     MOV BX,ARRAYP
15     MOV CX,COUNTP
16     XOR AL,AL
17     SUMD:ADD AL,[BX]
18     INC BX
19     LOOP SUMD
20     RET
21 CHECKSUMD ENDP
22
23 END
```

2、宏结构程序设计

2.1、宏汇编

宏指令实际上是一段代码序列的缩写。在汇编时，汇编程序用对应的代码序列替代宏指令

2.1.1、宏的定义和调用

- 原则：先定义，后调用
- 宏定义一般格式

```
1 | 宏名 macro[形参表]
2 | 宏定义体
3 | endm
```

- 宏调用一般格式

```
1 | 宏名 [实参表]
```

- 宏调用的实质是在汇编过程中进行**宏展开**，即用相应的**宏定义体**取代源程序的**宏指令**，同时用位置匹配的**实参对形参**进行取代

```
1 ;显示字符串
2 DISPMSG MACRO MESSAGE
3     LEA DX,MESSAGE
4     MOV AH,09H
5     INT 21H
6     ENDM
7 ;调用宏指令显示字符串
8 DISPMSG <Hello world!>
```

2.1.2、宏的参数

参数可以是常数、变量、存储单元、指令、表达式

- &：替换操作符，用于将参数与其他字符分开

```
1 dstring macro string
2     db '&string&',0dh,0ah,'$'
3     endm
```

- <>：用于传递的字符串实参中含有逗号、空格等间隔符号

```
1 | dstring <This is an example>
```

- !: 转义操作符，用于实参中含有<>

```
1 | dstring <0 !< number !< 10>
```

- %: 表达式操作符，表示将后跟的一个表达式的值作为实参，而不是表达式本身

```
1 | dstring %(1024-1)
```

2.1.3、与宏有关的伪指令

- 局部标号伪指令LOCAL

当宏定义中使用标号时，防止因多次宏展开而造成标号重复的现象发生。每次宏展开时，汇编程序将对其中的标号自动产生一个唯一的标识符；注意区分子程序中LOCAL指令的作用：
定义局部变量

LOCAL 标号列表

2.1.4、宏与子程序

- 主要不同点：调用方式
- 宏调用在汇编时进行程序语句的**展开**，**不需要返回**；简化了**源程序**，并不减小目标程序，执行速度不变；子程序调用转向子程序体，需要执行RET指令返回，子程序还是**目标程序**级的简化，但是需要利用堆栈，占用一定的时空开销。

2.2、重复汇编

2.2.1、按参数值重复伪指令REPEAT

```
1 | REPEAT 重复次数  
2 |     重复体  
3 | ENDM  
4 |  
5 | ;定义26个大写字母  
6 | CHAR EQU 'A'  
7 | AZTABLE EQU THIS BYTE  
8 | REPEAT 26  
9 |     DB CHAR  
10|     CHAR = CHAR + 1  
11| ENDM
```

2.2.2、按参数个数重复伪指令FOR

```
1 | FOR 形参,<实参数表>  
2 |     重复体  
3 | ENDM  
4 |  
5 | ;保护寄存器  
6 | FOR REG,<AX,BX,CX,DX>  
7 |     PUSH REG  
8 | ENDM
```

2.2.3、按参数字符个数重复伪指令FORC

```

1  FORC 形参,字符串
2      重复体
3  ENDM
4
5  ;恢复寄存器
6  FORC REG,DCBA
7      POP &REG&X
8  ENDM

```

2.3、条件汇编

根据条件确定是否汇编某段语句序列，**汇编阶段完成**，汇编时判断，**只汇编其中一个分支，执行时不再判断**，直接执行此分支；

而条件控制伪指令对**两支都要汇编，执行时再次判断**，选择执行其中一个分支

```

1 宏名 MACRO 形参
2 IFXX 表达式
3 分支语句体
4 [ELSE
5     分支语句体2]
6 ENDIF
7 ENDM

```

格 式	功 能 说 明
IF 表达式	汇编程序求出表达式的值,此值不为 0 则条件满足
IFE 表达式	汇编程序求出表达式的值,此值为 0 则条件满足
IFDEF 符号	符号已定义(内部定义或声明外部定义),则条件满足
IFNDEF 符号	符号未定义,则条件满足
IFB <形参>	用在宏定义体。如果宏调用没有用实参替代该形参,则条件满足
IFNB <形参>	用在宏定义体。如果宏调用用实参替代该形参,则条件满足
IFIDN <字符串 1>,<字符串 2>	字符串 1 与字符串 2 相同则条件满足,区别大小写
IFIDNI <字符串 1>,<字符串 2>	字符串 1 与字符串 2 相同则条件满足,不区别大小写
IFDIF <字符串 1>,<字符串 2>	字符串 1 与字符串 2 不相同则条件满足,区别大小写
IFDIFI <字符串 1>,<字符串 2>	字符串 1 与字符串 2 不相同则条件满足,不区别大小写

3、模块化程序设计

使程序具有模块化结构的几种基本方法

- 程序分段
- 子程序
- 宏
- 源文件包含
- 目标代码文件连接

3.1、源程序文件的包含

INCLUDE 文件名

- 可将常用的子程序形成.**ASM**汇编语言源文件
- 可将常用的宏定义存放在.**MAC**宏库文件中
- 可将常量定义、声明语句组织在.**INC**包含文件中

汇编程序对INCLUDE伪指令进行汇编时，将它指定的文本文件内容插入到该伪指令所在位置，与其他部分**同时汇编**，实质上仍然是一个源程序，所以合并的源程序之间的各种标识符不能冲突

- 源文件包含的操作步骤
 1. 分别编辑生成各个文件
 2. 汇编、连接主程序文件

3.2、目标代码文件的连接

- 步骤
 1. 把常用子程序写成独立的源程序文件，**单独汇编**，形成子程序的目标文件.OBJ
 2. 主程序也经过独立汇编之后形成**目标文件**
 3. 连接程序将所有目标文件**连接起来**，最终产生**可执行文件**
- 需要遵循的原则
 1. 声明共用的变量、过程等

PUBLIC 变量名/过程名	说明变量/过程可以在其他模块中使用
EXTERN 变量名/过程名:类型	说明变量/过程来自其他模块
 2. 实现正确的段组合
 - 由于各文件独立汇编，因此**子程序文件必须定义在代码段中**，也可以具有局部的数据变量
 - 采用简化段定义格式，只需采用**相同的存储模型**
 - 完整段定义格式中，各自定义的段名、类别必须相同，组合类型都是public，定义成远调用
 3. 处理好参数传递的问题
 - 少量数据：寄存器、堆栈
 - 大量数据：缓冲区，用寄存器或堆栈传送数据的存储地址；或用PUBLIC/EXTERN声明的变量

3.3、子程序库的调入

- 步骤
 1. 把常用子程序写成独立的源文件，单独汇编形成OBJ文件后，存入子程序库
 2. 主程序也单独汇编形成OBJ文件
 3. 主程序连接时，调入子程序库中的子程序模块，产生最终的可执行文件
- 子程序库的概念
 - 子程序库文件 (.LIB) 是子程序模块的集合，其中存放着各子程序的名称、**目标代码及有关定位信息**
- 格式

LIB 库文件名 + 子程序目标文件名

4、输入输出程序设计

- 输入指令**IN**从外设读取数据
- 输出**OUT**指令向外设提供数据
- 处理器不直接操纵外设，而是通过I/O接口控制外设

4.1、输入输出指令

- | |
|---|
| 呈现给程序员的外设是端口 (Port) 即I/O地址 |
| 寻址外设端口的地址线为 16 条，端口最多为64K个，端口号为0000H~FFFFH |
| 每个端口传送一个字节的数据 |

- 输入指令IN

1. 直接寻址: 00H < 端口号 < FFH

```
IN AL/AX,i8
```

2. 间接寻址: FFH < 端口号 < FFFFH

```
1 MOV DX,i8/i16  
2 IN AL/AX, DX
```

- 输出指令OUT

1. 直接寻址: 00H < 端口号 < FFH

```
OUT i8,AL/AX
```

2. 间接寻址: FFH < 端口号 < FFFFH

```
1 MOV DX,i8/i16  
2 OUT DX,AL/AX
```

4.2、程序直接控制输入输出

```
1 .model tiny ;形成com格式的程序  
2 .code  
3 .startup  
4 call speaker_on ;打开扬声器声音  
5 mov ah,1 ;等待按键  
6 int 21h  
7 call speaker_off ;关闭扬声器声音  
8 .exit 0  
9 speaker_on proc ;扬声器开子程序  
10 push ax  
11 in al,61h ;读取原来控制信息  
12 or al,03h ;D1D0=PB1PB0=11b  
13 out 61h,al ;直接控制发声  
14 pop ax  
15 ret  
16 speaker_on endp
```

4.3、程序查询输入输出

- **处理器主动**执行程序检测外设状态，然后进行数据交换
- 实际的输入输出程序可以规定一个超时参数

4.4、中断服务程序

- 需要交换数据的**外设**，采用**中断请求**向处理器提出要求，即**外设主动**
- 处理器执行事先设计好的**中断服务程序**，在中断服务程序当中实现数据交换
- 8086可以处理**256**种中断，分为**内部**、**外部**两种类型
- **外部可屏蔽中断**用于与外设进行数据交换

4.4.1、内部中断服务程序

- 主程序通过中断调用指令 `INT n` 执行内部中断服务程序，其实质相当于**子程序调用**
- 进入中断服务程序后通常要执行 `STI` 指令开放可屏蔽中断源，在执行 `IRET` 指令返回
- 通常采用寄存器传递参数

- 主程序调用内部中断服务程序前，必须修改中断向量，使其指向相应的中断服务程序

```

1 | MOV AH,35H          ;获取中断向量
2 | MOV AL,中断向量号    ;入口参数:AL = 中断向量号
3 | INT 21H            ;出口参数:ES:BX = 中断服务程序的入口地址

```

```

1 | MOV AH,25H          ;设置中断向量
2 | MOV AL,中断向量号    ;入口参数:AL = 中断向量号
3 | INT 21H            ;出口参数:ES:BX = 中断服务程序的入口地址

```

4.4.2、驻留中断服务程序

| 程序执行结束，但保留在系统主存中，可以让其他程序使用

```

1 | ;实现程序驻留
2 | MOV AH,31H
3 | MOV AL,返回代码
4 | MOV DX,程序驻留的容量（单位为节，1节 = 16字节）
5 | INT 21H

```

4.4.3、外部可屏蔽中断服务程序

| 外设采用中断方式与处理器交换信息是利用**外部可屏蔽中断**实现的

- 注意点

1. 发送中断结束命令

```

1 | MOV AL,20H
2 | OUT 20H,AL

```

2. 不能采用寄存器传送参数

3. 不能使用INT 21H

4. 中断服务程序尽可能短小

5. 主程序除需要修改中断向量外，还要控制中断允许标志IF，设置中断