



华中科技大学

# 80X86 汇编语言程序设计

## 80X86 Assembly Language Programming

### 第3章 宏汇编语言

金良海 教授

计算机学院 医学图像信息研究中心

Email: [lianghaijin@hust.edu.cn](mailto:lianghaijin@hust.edu.cn)



## 80X86 宏汇编语言 汇编语言 ?

在汇编语言中加入了一些辅助性的指令语句（如伪指令和宏指令），以帮助汇编程序更好地将汇编语言源程序翻译成二进制的机器语言源程序。这就是宏汇编语言。

# 概述 (2)



华中科技大学

80X86宏汇编  
指令语句

{ 机器指令语句  
伪指令语句  
宏指令语句

在本章及后面的章节中,将汇编指令语句称为机器指令语句



## 本章内容

3.1 宏汇编语言中的表达式

3.2 常用的机器指令

3.3 伪指令及其与机器指令的区别

3.4 常用的DOS系统功能调用 (1,2,9,10号调用)

3.5 MASM和LINK的使用

## 学习目标与要求

- ◆ 正确而熟练地使用地址表达式和数值表达式
- ◆ 熟悉常用的机器指令的使用格式、功能
- ◆ 区别机器指令语句和伪指令语句
- ◆ 常用的伪指令功能、使用方法
- ◆ 熟练掌握常用的DOS系统功能调用  
(1,2,9,10号调用)

## 学习重点 (1)

### 1. 宏汇编语言中的表达式

- ◆ 符号常量
- ◆ 变量 (数据在主存中的存储示意图)
- ◆ 地址表达式
- ◆ 属性定义算符 (PTR, 跨段前缀)
- ◆ 属性分离算符 (SEG, OFFSET)

## 学习重点 (2)

### 2. 常用的机器指令语句

- ◆ 数据传送指令
- ◆ 算术运算指令
- ◆ 位操作指令

**要求掌握各指令的语法规则，功能，最常用指令对标志寄存器的影响。**

## 学习重点 (3)

### 3. 常用的伪指令

- ◆ 数据定义伪指令
- ◆ 符号定义伪指令
- ◆ 段定义伪指令
- ◆ 假定伪指令
- ◆ 源程序结束伪指令
- ◆ 汇编地址计数器\$



## 学习重点 (4)

### 4. 常用的DOS系统功能调用

1, 2, 9, 10 号系统功能调用

特别注意特殊字符的显示效果:

0AH, 0DH

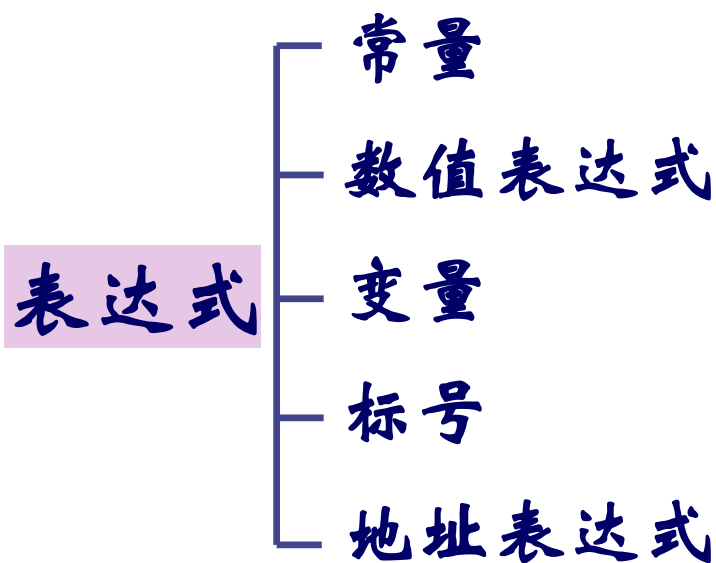
特别注意 '\$' (24H) 的作用。

## 学习难点

- ◆ 变量、地址表达式的使用
- ◆ 常用的机器指令的记忆、各指令的特殊要求
- ◆ 汇编地址计数器\$和假定伪指令
- ◆ 正确理解DOS系统功能调用、注意特殊字符的显示效果



# 3.1 宏汇编语言中的表达式





## 3.1.1 常量与数值表达式

### 常量: 数值常量和符号常量

数值常量: 就是一个数值, 如1、2、3等

符号常量: 用符号去代替数值。使用伪指令  
“EQU”和 “=”来定义符号常量。

例如:   AA EQU 50  
          BB =   100

在C语言中:

```
#define AA 50
```

```
#define BB 100
```



## (3.1.1) 1. 常量 (1)

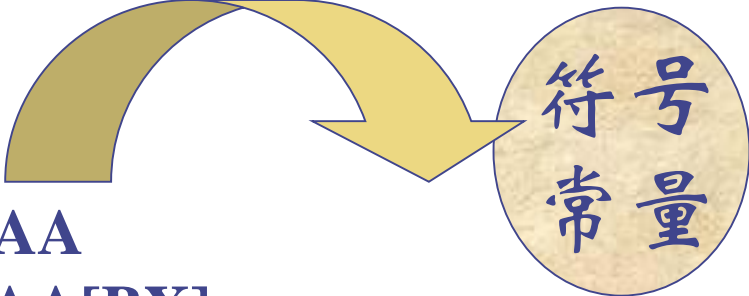
MOV CX, 10  
MOV AH, 10[BX]  
MOV DL, 10[BX][SI]



数值常量

AA EQU 10  
BB = 100

MOV CX, AA  
MOV AH, AA[BX]  
MOV DL, AA[BX][SI]



符号常量



## (3.1.1) 1.常量 (2)

### 数值常量的类型

数值常量	二进制常量	10101100 B
	八进制常量	127 Q , 127 O
	十进制常量	829 , 829 D
	十六进制常量	12AF H, 0C2AF H
	字符常量	'123Ab', "123Ab"





## (3.1.1) 1.常量 (3)

定义一个大小为10的数组，输入数据，求和，排序

```
int score[10], i, j, temp, sum=0;
for (i=0; i<10; i++) scanf("%d", score+i);
for (i=0; i<10; i++) sum += score[i];
for (i=0; i<9; i++)
    for (j=i+1; j<10; j++)
        if (score[i]>score[j]) {
            temp=score[i]; score[i]=score[j];
            score[j]=temp;
        }
```

数组的大小要改为20，或者以后还可能变化，怎么办？





## (3.1.1) 1.常量 (4)

```
#define N 20
```

N EQU 20

N = 20

```
int score[N], i, j, temp, sum=0;
for (i=0;i<N; i++) scanf("%d", score+i);
for (i=0;i<N; i++) sum += score[i];
for (i=0;i<N-1;i++)
    for (j=i+1; j<N; j++)
        if (score[i]>score[j]) {
            temp=score[i]; score[i]=score[j];
            score[j]=temp;
        }
```







## (3.1.1) 2.数值表达式 (1)

### 数值表达式

由常量和运算符 组成的有意义的式子。

- 一个常量是一个数值表达式；
- 通过运算符和括号将数值连接起来形成的表达式

算术运算

+, —, \*, /, MOD, SHR, SHL

逻辑运算

AND, OR, XOR, NOT

关系运算

EQ, NE, LT, GT, LE, GE





华中科技大学

## (3.1.1) 2.数值表达式 (2)

**MOD:** 模除，两整数相除后取余数

$$42 \text{ MOD } 10 = 2$$

**SHR:** 右移，shift right  
二进制常量右移规定的位数，空位补0。

$$11100111\text{B} \text{ SHR } 2 = 00111001$$

$$120 \text{ SHR } 2 = 30$$

**SHL:** 左移，shift left  
二进制常量左移规定的位数，空位补0。



## (3.1.1) 2.数值表达式 (3)

**AND:** 逻辑乘, 按位进行的AND运算

$$789AH \quad \text{AND} \quad 0FH = 0AH$$

$$\begin{array}{r} 0111 \ 1000 \ 1001 \ 1010 \\ \wedge \ 0000 \ 0000 \ 0000 \ 1111 \\ \hline 0000 \ 0000 \ 0000 \ 1010 \end{array}$$

按位运算

**OR:** 逻辑加

**XOR:** 按位加

$$\begin{array}{r} 0111 \ 1000 \ 1001 \ 1010 \\ \text{XOR} \ 0000 \ 0000 \ 0000 \ 1111 \\ \hline 0111 \ 1000 \ 1001 \ 0101 \end{array}$$

**NOT:** 逻辑非



## (3.1.1) 2.数值表达式 (4)

### 关系运算

(1)用文字形式, 而不用符号

即用 EQ, 而不用 =

用 LT, 而不用 <

(2)若关系成立, 则结果为 0FFH;

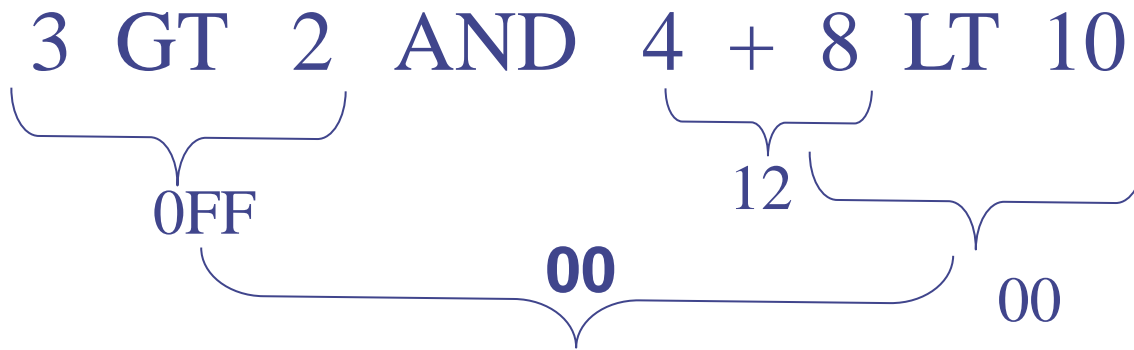
关系不成立, 则结果为 0。





华中科技大学

## (3.1.1) 2.数值表达式 (5)



算符的优先级?

* / MOD SHL SHR
+ -
EQ NE LT LE GT GE
NOT
AND
OR XOR

优先级高



优先级低





## 3.1.2 变量、标号与地址表达式

### 1. 变量

变量是一个数据存储单元的名字。

存储单元的属性:

段属性

偏移地址

单元的类型

单元中的内容

1000H: 500H



存储单元的地址

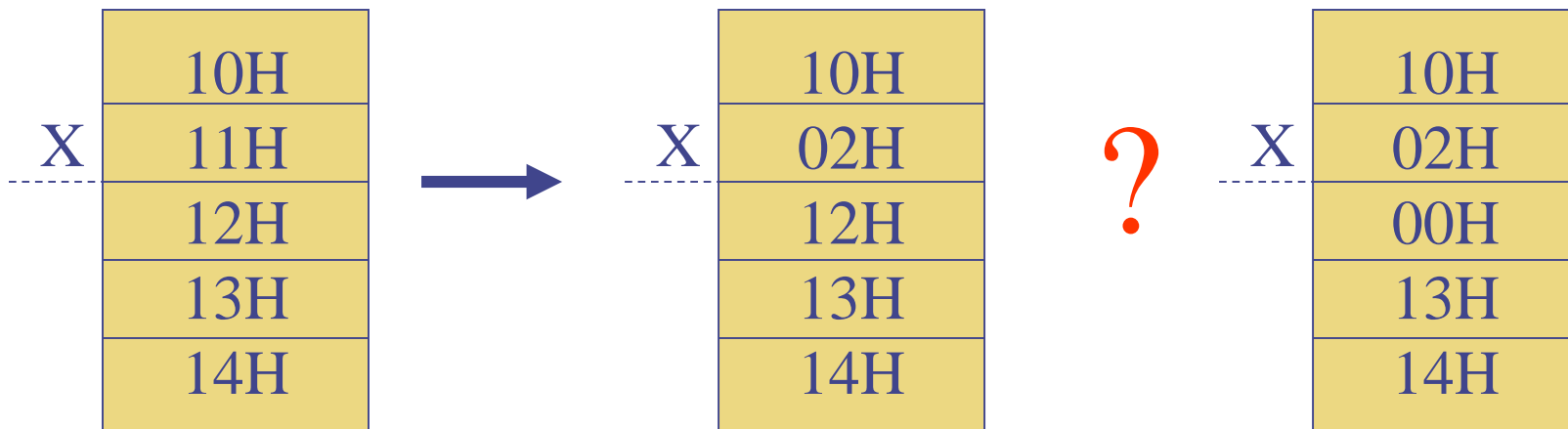


存储单元中的内容



## (3.1.2) 1.变量 (1)

正确执行 `MOV X, 2` 需要什么信息呢?



变量的三个属性:

- (1) 段属性
- (2) 偏移地址
- (3) 类型

变量所在段的段首址应在一个段寄存器中。

变量所在位置与的段首址之间的字节距离。





## (3.1.2) 1.变量 (2)

### 变量的定义:

变量名 数据定义伪指令 表达式[, ...]

{	<b>DB</b>	字节类型
	<b>DW</b>	字类型
	<b>DD</b>	双字类型
	<b>DF</b>	三字类型
	<b>DQ</b>	4字类型
	<b>DT</b>	10个字节



表达式有  
5种形式





## (3.1.2) 1.变量 (3)

**变量名 数据定义伪指令 表达式[,...]**

**数据定义伪指令: DB, DW, DD, DF, DQ, DT**

**表达式有5种形式**

(1) 数值表达式

(2) 字符串

(3) 地址表达式

(4) ?

(5) 重复子句 n DUP (表达式[, ...])



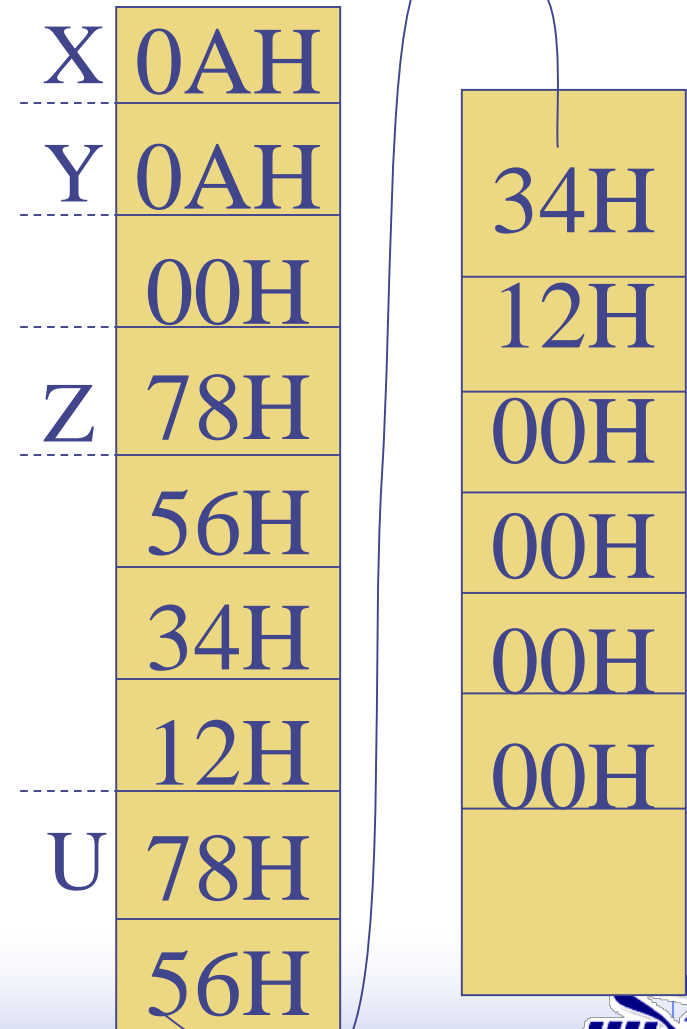
## (3.1.2) 1.变量 (4)

### (1) 变量定义: 数值表达式

```
X  DB  10
Y  DW  10
Z  DD  12345678H
U  DQ  12345678H
V  DT  12345678H
```

如下定义正确吗?

~~K DB 1234H~~





## (3.1.2) 1.变量 (5)

### (2) 变量定义: 字符串

如果字符串的长度超过2个字符,  
数据定义伪指令只能用DB

X DB 'abcd'

Y DB '12'

Z DW '12'

如下定义正确吗?

~~K DW '1234'~~

X	61H
	62H
	63H
	64H
Y	31H
	32H
Z	32H
	31H





## (3.1.2) 1. 变量 (6)

### (3) 变量定义: 地址表达式

**地址表达式: 由变量、标号、常量和运算符组成的有意义的式子。**

**在数据定义语句中, 不能出现带寄存器符号的地址表达式。地址变量定义伪指令只能用DW和DD.**

如下定义正确吗?

~~K DB X~~





## (3.1.2) 1.变量 (7)

### ➤ DW 变量或标号

取该变量或标号的偏移地址来初始化相应的字存储单元。

### ➤ DD 变量或标号

若该变量定义在16位段中，则第一个字存储其偏移地址，第二个字存放其段首址；

若该变量定义在32位段中，则只取偏移地址来初始化相应双字单元。





## (3.1.2) 1.变量 (8)

地址表达式的数据存放

**DATA SEGMENT USE16**

**W DB 34H**

**X DB 56H**

**Y DW X**

**Z DD X**

**DATA ENDS**

设DATA段的段址为  
2000H

W	34H	0000
X	56H	0001
Y	01H	0002
	00H	0003
Z	01H	0004
	00H	0005
	00H	0006
	20H	0007





## (3.1.2) 1.变量 (9)

### (4) 变量定义: ?

表示定义的变量无确定的初值

**X DB ?**

**Y DW ?**

### (5) 变量定义: 重复子句

**N DUP (表达式[, 表达式]...)**

例1: **X DB 3 DUP (2)**

**⇔ X DB 2, 2, 2**





## (3.1.2) 1.变量 (10)

例2: Y DB 3 DUP (1,2)

⇔ Y DB 1, 2, 1, 2, 1, 2

例3: Z DB 3 DUP (1, 2 DUP (2) )

⇔ Z DB 3 DUP (1, 2, 2 )

⇔ Z DB 1, 2, 2, 1, 2, 2, 1, 2, 2





## (3.1.2) 1.变量 (11)

- Q: 同一个数值表达式在不同数据定义伪指令后的表现形式?
- Q: 一条数据定义伪指令后的多个表达式的存放方法?
- Q: 长度超过2的字符串能否出现用 **DW** 定义?
- Q: 地址表达式是什么?
- Q: 地址表达式只能出现在 **DW** 和 **DD** 后。  
它们各自的意义是什么?
- Q: 重复子句的嵌套和展开?



## (3.1.2) 1.变量 (12)

```
DATA SEGMENT USE16
A      DW      M
BUF    DB      'AB', 0DH
CON    EQU     500H
B      DW      0FFAAH
D      DD      BUF
M      DB      2
DATA   ENDS
```

实验：在内存中看  
C4\_050P.asm

A	0B H	0000H
	00 H	
BUF	41H	0002H
	42H	
	0DH	
B	0AAH	0005H
	0FFH	
D	02	BUF所在段的段址
	00	
	??	
??		
M	02	



## (3.1.2) 2.标号

### 机器指令存放地址的符号表示。

标号的三个属性： 段属性

偏移地址

类型： NEAR  
FAR

标号的定义：

```
MAIN PROC NEAR
```

```
... ..
```

```
NEXT: MOV AL, [SI]
```

```
... ..
```

```
JMP NEXT
```

```
RET
```





## (3.1.2) 3.地址表达式 (1)

- **地址表达式**是由变量、标号、常量、各种存储器寻址方式中用到的寄存器和各种运算符组成的有意义的式子。
- 单个的变量、标号、常量、寄存器的内容是地址表达式的特例。

例: BUF DB 1, 2, 3, 0FFH  
X DW BUF  
... ..  
M: MOV AH, BUF+1  
MOV BX, X-1  
MOV AL, [BX]  
MOV SI, X-2

如果地址表达式中出现变量或标号, 则均取他们的EA参加运算, 不可理解为取其存储单元中的内容。





## (3.1.2) 3.地址表达式 (2)

- 先算整个表达式的值，再算物理地址，最后取存储单元内容。
- **地址表达式的值是段内偏移**，为了确定其物理地址，则还需要确定段地址。段地址由地址表达式中的变量、标号等隐含决定；若段地址不能从表达式中确定（如对于常量地址），则需用跨段前缀显式地给出其段地址：

段寄存器名：	地址表达式	ES:[2000H]	
或	段名：	地址表达式	DATA:[2000H]





## (3.1.2) 3.地址表达式 (3)

- 确定了地址表达式的物理地址（段地址和段内偏移）后，还需确定这个物理地址处的数据类别（字节？字？等等）。数据类别一般可由地址表达式中的变量等隐含决定，或由指令中其他操作数确定；若不能决定，则需要使用类型运算符PTR显式地说明：

类型    PTR    地址表达式





## (3.1.2) 3.地址表达式 (4)

$\left\{ \begin{array}{l} \text{BYTE} \\ \text{WORD} \\ \text{DWORD} \\ \text{FWORD} \\ \text{NEAR} \\ \text{FAR} \end{array} \right\} + \text{PTR} + \text{地址表达式}$

例1: `MOV BYTE PTR ES:[2000], 2`

例2: `BUF DB 1, 2`

`MOV AX, WORD PTR BUF`





## (3.1.2) 3.地址表达式 (5)

汇编程序对汇编源程序的要求:

操作数地址的类型应明确和匹配。

MOV AX, 2[BX]

MOV AL, 2[BX]

BUF DW 10

MOV AX, BUF

~~MOV DS:[2000], 2~~

分三种情况:

- (1)两个类型都明确(一致)
- (2)一个明确, 一个不明确
- (3)两个都不明确

如果不匹配, 则需要使用  
PTR强制转换。







## (3.1.2) 3.地址表达式 (6)

### 其它几个常用的运算符

(1) 取变量或标号的段地址

**SEG** 变量或标号

例: MOV AX, SEG BUF

(2) 取变量或标号的段内偏移地址

**OFFSET** 变量或标号

例: MOV AX, OFFSET BUF

已介绍的运算符:  
算术运算符  
逻辑运算符  
关系运算符





## (3.1.2) 3.地址表达式 (7)

### 其它几个常用的运算符

(3) 取常量或数值表达式的高字节

**HIGH** 常量或数值表达式

(4) 取常量或数值表达式的低字节

**LOW** 常量或数值表达式

例: NUM EQU 0FECDDH

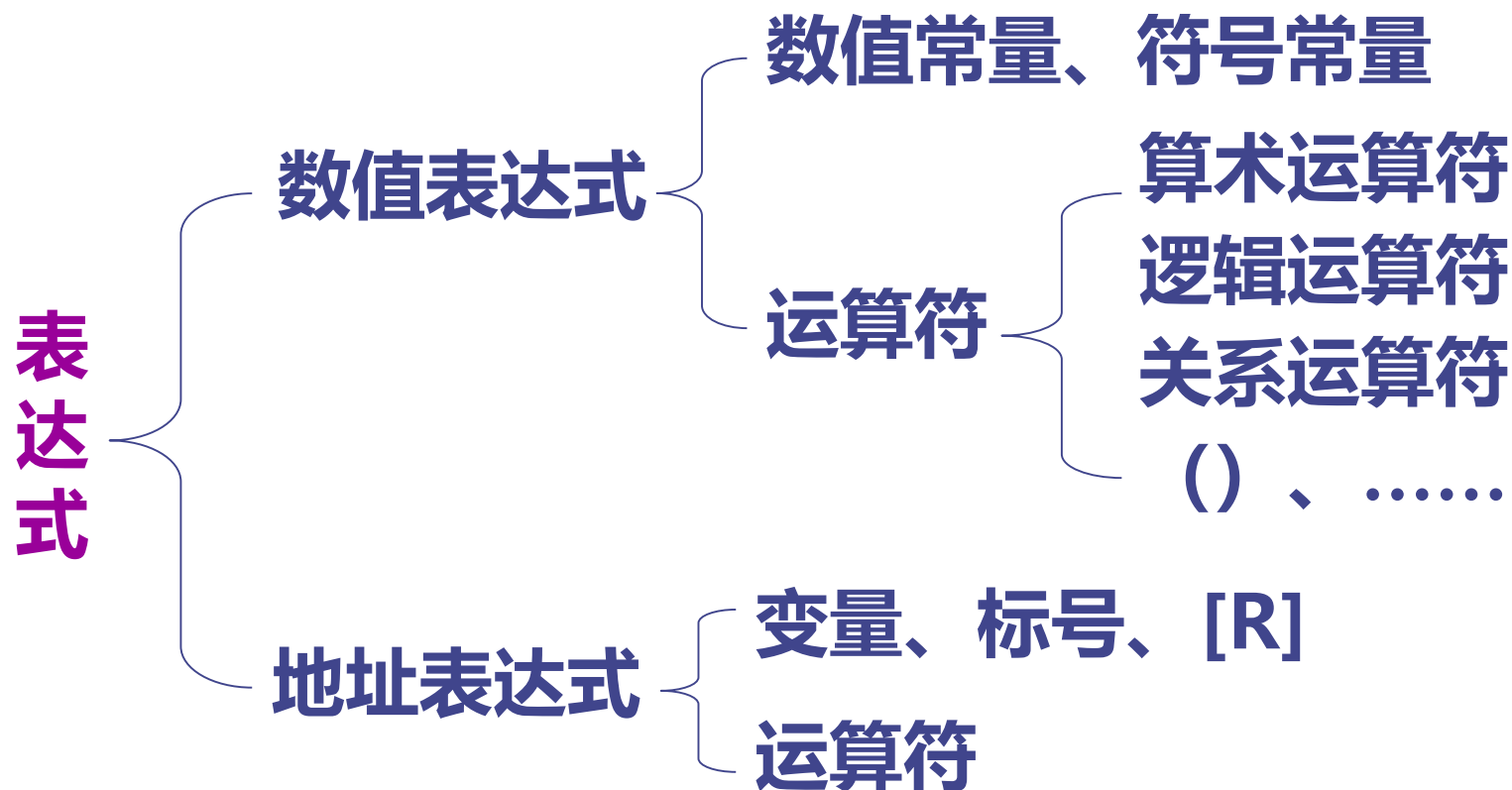
MOV AX, HIGH NUM ;AH = 0FEH

MOV AL, LOW NUM ;AL = 0DDH





## (3.1) 表达式小结





## 3.2 常用的机器指令语句 (1)

- (1) 数据传送指令 (3.2.1)
- (2) 算术运算指令 (3.2.2)
- (3) 位操作指令 (3.2.3)
- (4) 控制转移指令 (第4章)
- (5) 串操作指令 (第5章)
- (6) 处理机控制指令 (第1、5、6章)



## 3.2 常用的机器指令语句 (2)

一般情况下，指令的共同要求：

**(1) 双操作数的操作数类型必须匹配。**

~~MOV AX, BH~~

~~SUB BL, CX~~

~~MOV SI, CL~~

~~ADD BUF, AX ;其中 BUF DB 1, 2~~

~~MOV ds:[2000], 2~~





## 3.2 常用的机器指令语句 (3)

一般情况下，指令的共同要求：

- (1) 双操作数的操作数类型必须匹配。
- (2) 目的操作数一定不能是立即操作数

`CMP AX, 2`

功能是：根据  $(AX) - 2$  的结果设置标志位

~~`CMP 2, AX`~~





## 3.2 常用的机器指令语句 (4)

一般情况下，指令的共同要求：

- (1) 双操作数的操作数类型必须匹配。
- (2) 目的操作数一定不能是立即操作数。
- (3) **目的操作数和源操作数不能同时为存储器操作数。** 如果一个操作数在内存存储单元中，那么另一个必须是立即数或寄存器操作数。

~~MOV BUF1, BUF2  
ADD WORD PTR [SI], [DI]  
SUB BUF1, [SI]~~





## 3.2.1 数据传送指令

### 1、一般数据传送指令

**MOV、MOVSX、MOVZX、XCHG、XLAT**

### 2、堆栈操作指令

**PUSH、POP、PUSHA、PUSHAD、POPA、POPAD**

### 3、标志寄存器传送指令

**PUSHF、POPF、PUSHFD、POPFD、LAHF、SAHF**

### 4、地址传送指令

**LEA, LDS, LES, LFS, LGS, LSS**

### 5、输入、输出指令

**IN、OUT**

除了SAHF、  
POPF,POPFD外,  
其他不影响标志  
位。





## (3.2.1) 1.一般传送指令 (1)

- (1) 一般数据传送指令 **MOV**
- (2) 有符号数传送指令 **MOVSX**
- (3) 无符号数传送指令 **MOVZX**
- (4) 数据交换指令 **XCHG**
- (5) 查表指令 **XLAT**





华中科技大学

## (3.2.1) 1.一般传送指令 (2)

### (1) 一般传送指令

#### **MOV** 指令

语句格式: **MOV OPD, OPS**

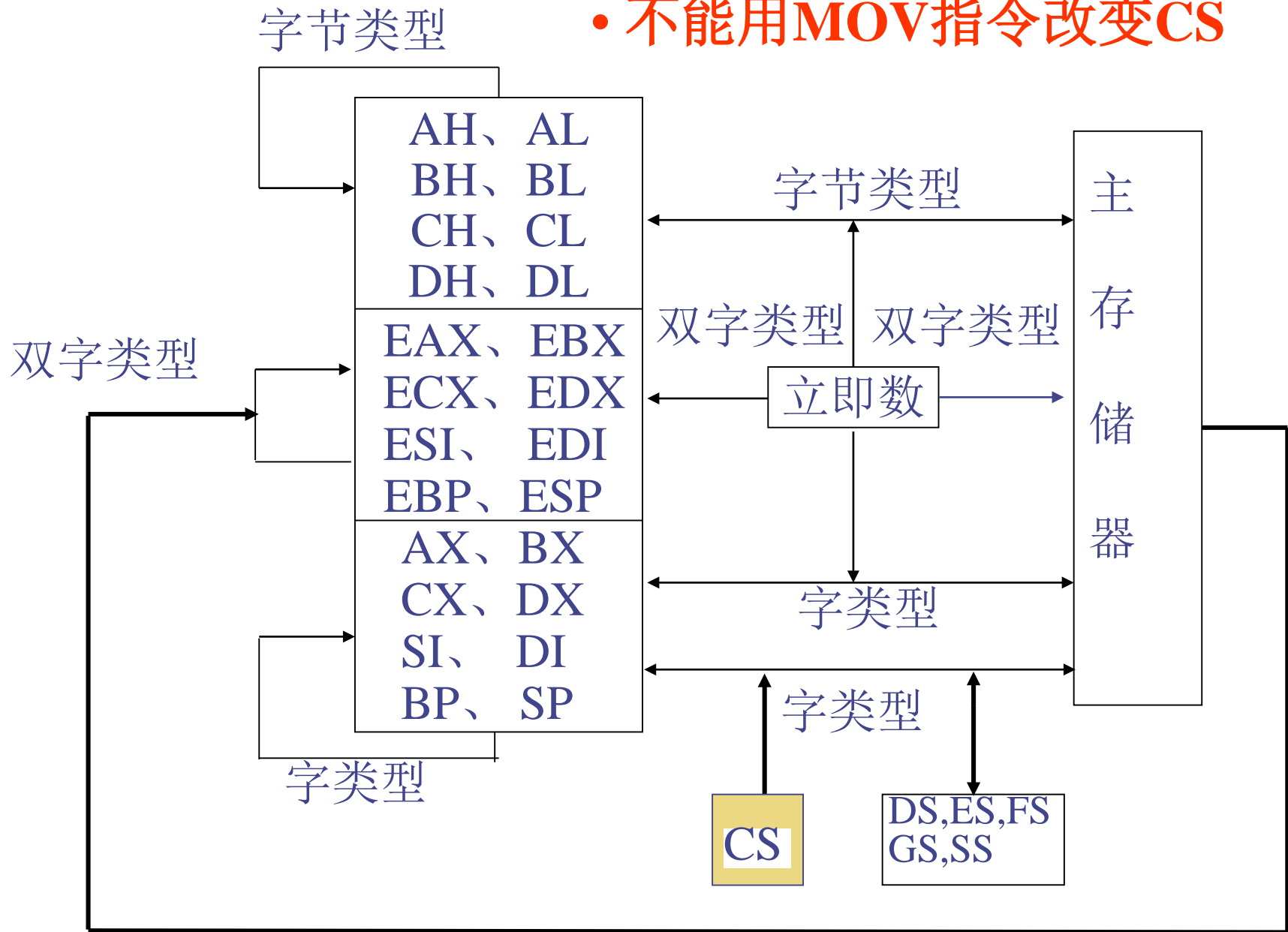
功 能: **(OPS) → OPD**

**MOV** 指令所允许的数据传送路径及类型, 见下图。



# 1.一般传送指令(3)

- 立即数不能送段寄存器
- 不能用MOV指令改变CS





## (3.2.1) 1.一般传送指令 (4)

根据传送图，分析如下语句，是否正确？

1. `MOV DS, SEG BUF` ; BUF为变量
2. `MOV DS, DATA` ; DATA为一段名  
Immediate mode illegal
3. `MOV CS, AX`  
Illegal use of CS register
4. `MOV DS, BUF` ; BUF 为字变量，其所  
; 在的段与DS建立联系。

语法正确，执行的结果可能不对。(why?)





## (3.2.1) 1.一般传送指令 (5)

### (2) 有符号数传送指令

#### MOVSX 指令 (386指令)

语句格式: MOVSX OPD, OPS

功 能: 将源操作数的**符号向前扩展**成与目的操作数相同的数据类型后,再送入目的地址对应的单元中。

说 明:

- **OPS 不能为立即数;**
- **OPD 必须是 16/32位的寄存器;**
- **源操作数的位数必小于目的操作数的位数。**





## (3.2.1) 1.一般传送指令 (6)

### (3) 无符号数传送指令

#### MOVZX 指令 (386指令)

语句格式: MOVZX OPD, OPS

功 能: 将源操作数的高位补0, 扩成与目的操作数相同的数据类型后, 再送入目的地址对应的单元中。

说 明:

- OPS 不能为立即数;
- OPD 必须是 16/32位的寄存器;
- 源操作数的位数必小于目的操作数的位数。





## (3.2.1) 1.一般传送指令 (7)

### MOVSX, MOVZX示例

例1: MOV BL, 0E3H

MOVSX EBX, BL

(EBX) = ?

0FFFFFFFE3H

若将最后一条指令换成

MOVZX EBX, BL

000000E3H

(EBX) = ?

例2: BYTE0 DB 0A8H

MOV BL, BYTE0

MOVSX ECX, BL

MOVSX EBX, BYTE0

;寄存器无对应限制





## (3.2.1) 1.一般传送指令 (8)

### (4) 数据交换指令

语句格式: XCHG OPD, OPS

功能 : (OPD)  $\rightarrow$  OPS (OPS)  $\rightarrow$  OPD

将源、目的地址指明的单元中的内容互换。

例3: XCHG AH, AL

执行前: (AX) = 1234H 执行后: (AX) = 3412H

注: 不能使用段寄存器

XCHG DS, AX ;

Improper use of segment register

XCHG BUF1, BUF2 ; 指令是否正确?







## (3.2.1) 1.一般传送指令 (9)

### (5) 查表转换指令

语句格式: XLAT

功能 :  $[(BX+AL)] \rightarrow AL$  或  
 $[(EBX+AL)] \rightarrow AL$

- 将(BX)或(EBX)为首址, (AL)为位移量的字节存储单元中的数据传送给AL。
- 用BX、EBX取决于16位段还是32位段。





## (3.2.1) 1.一般传送指令(10)

设有一个16进制数码 (0~9, A~F) 在(AL)中, 现将该数码转换为对应的ASCII。

一般的算法: 判断(AL)是否小于等于9,  
若是: 则将  $(AL) + '0' \rightarrow AL$ ;  
否则: 将  $(AL) - 10 + 'A' \rightarrow AL$ ;

```
MYTAB DB '0123456789ABCDEF'  
MOV    BX, OFFSET MYTAB  
XLAT
```

XLAT 可用来对文本数据进行编码和译码, 从而实现简单的加密和解密。





## (3.2.1) 1.一般传送指令 (11)

习题：判断下列指令的正确性。

MOV	AX, BX
MOV	AH, BX
MOV	[SI], [BX]
MOV	AX, BL
MOVSX	AX, BL
MOVSX	BL, AX
XCHG	AX, BYTE PTR [SI]
MOVZX	AX, BH
MOVZX	CH, AX





华中科技大学

## (3.2.1) 2.地址传送指令 (1)

- (1) 偏移地址传送指令 **LEA**
- (2) 数据段地址传送指令 **LDS**
- (3) 附加段地址数传送指令 **LES**





## (3.2.1) 2.地址传送指令 (2)

### (1) 传送偏移地址指令

语句格式: **LEA OPD, OPS**

功 能: 计算 OPS 的偏移地址, 并将其送入 OPD 中。

- **OPD** 16 / 32位的通用寄存器;
- **OPS** 存储器地址;
- 如果偏移地址为32位, 而OPD为16位, 则取低16位→ OPD; 如果偏移地址为16位, 而OPD为32位, 则高16位补0后→ OPD



## (3.2.1) 2.地址传送指令 (3)

.386

DATA SEGMENT USE16

BUF DB 'ABCDEF'

NUM DW 72, -5, 100H

POIN DW 0

DATA ENDS

CODE SEGMENT USE16

ASSUME CS:CODE,

DS:DATA

BEGIN: MOV AX, DATA

MOV DS, AX

MOV ESI, OFFSET NUM

LEA ESI, NUM

MOV AX, [ESI]

LEA AX, [ESI]

LEA DI, [ESI+4]

LEA POIN, BUF ✗

MOV POIN, OFFSET BUF

MOV EBX, 12345678H

LEA DX, [EBX+4321H]

LEA EAX, [EBX+4321H]

LEA ECX, [BX+4321H]

MOV AX, 4C00H

INT 21H

CODE ENDS

END BEGIN

参见 C3\_060P.asm



华中科技大学

## (3.2.1) 2.地址传送指令 (4)

**MOV SI, OFFSET NUM 与  
LEA SI, NUM 等效;**

找不到与 **LEA DI, [SI+4]**等效的MOV语句,

找不到与 **MOV POIN, OFFSET BUF** 等效的  
**LEA**语句





## (3.2.1) 2.地址传送指令 (5)

### (2) 传送偏移地址和数据段首址指令

格式: **LDS OPD, OPS**

功能: **(OPS) → OPD, (OPS+2/4) → DS**

- **OPD** 一定是一个16/32位的通用寄存器;
- **OPS**所提供的一定是一个存储器地址,类型为 **DWORD/FWORD**。

**C3\_061J.asm**

观察: (1) **(DS)**的改变

(2) 比较**(DS)** 改变前后相同访问指令的执行结果





.386

**DATA1** SEGMENT USE16

T1 DW - 50H

T2 DD F

T3 DB '1234567'

**DATA1** ENDS

DATA2 SEGMENT USE16

BUF DB 'ABCDEFGH'

F DW 70H

DATA2 ENDS

CODE SEGMENT USE16

ASSUME

CS:CODE, DS:DATA1

START:

MOV AX, DATA1

MOV DS, AX

MOV SI, 6

MOV AX, [SI]

MOV AX, T1

LDS SI, T2

MOV AX, [SI]

MOV AX, T1

MOV AH, 4CH

INT 21H

CODE ENDS

END START

C3\_061J.asm

比较和分析(AX)的异同

**(DS)** 改变前后相同访问指令的执行结果不同



## (3.2.1) 2.地址传送指令 (7)

### (3) 传送偏移地址和附加段首址指令

格式: **LES OPD, OPS**

功能:  $(OPS) \rightarrow OPD$ ,  $(OPS+2/4) \rightarrow ES$

- **OPD** 一定是一个16/32位的通用寄存器;
- **OPS**所提供的一定是一个存储器地址, 类型为 **DWORD/FWORD**。





## (3.2.1) 数据传送指令小结 (1)

一般传送	<b>MOV</b>	OPD, OPS
有符号数传送	<b>MOVSX</b>	R16/R32, OPS
无符号数传送	<b>MOVZX</b>	R16/R32, OPS
一般数据交换	<b>XCHG</b>	OPD, OPS
查表转换	<b>XLAT</b>	
传送偏移地址	<b>LEA</b>	R16/R32, OPS
传送偏移地址及段首址	<b>LDS</b>	R16/R32, OPS
	<b>LES, LFS, LGS, LSS</b>	





## (3.2.1) 数据传送指令小结 (2)

**思考题：同学们已经学过6种寻址方式，能否能以MOV指令为例，写出各种寻址方式的MOV指令？**

MOV AX, BX

MOV [BX], AX

MOV 8[DI], AX

MOV AX, 8[BX][SI]

MOV AX, 8

MOV AX, ES:[1000H]





## 3.2.2 算术运算指令

一般对标志位都有影响

### 1、加法指令

INC、ADD、ADC

### 2、减法指令

DEC、NEG、SUB、SBB、CMP

### 3、乘法指令

IMUL、MUL

### 4、除法指令

IDIV、DIV

### 5、符号扩展指令

CBW、CWD、CWDE、CDQ





华中科技大学

## (3.2.2) 1.加法指令 (1)

- |             |            |
|-------------|------------|
| (1) 加1指令    | <b>INC</b> |
| (2) 加指令     | <b>ADD</b> |
| (3) 带进位的加指令 | <b>ADC</b> |





## (3.2.2) 1.加法指令 (2)

### (1) 加1指令 (不影响CF)

INC OPD ; (OPD) +1 → OPD

### (2) 加指令

ADD OPD, OPS ; (OPD)+(OPS) → OPD

例: MOV AX, 0FFFDH

ADD AX, -7FFFH

执行上述语句后, (AX)= ?

OF= ? CF=? ZF= ? SF=?

(AX)=7FFEH OF=1 CF=1 ZF=0 SF=0





## (3.2.2) 1.加法指令 (3)

### (3) 带进位加指令

语句格式: `ADC OPD, OPS`

功能:  $(OPD) + (OPS) + CF \rightarrow OPD$

`MOV AX, 03F0H`

`ADD AL, 28H`

`ADC AH, 0`

问: 此程序段的意义是什么?

$(AX) = ?$







华中科技大学

## (3.2.2) 1.加法指令 (4)

例1: 计算 1234 F00FH + 56780F0H  
(只允许使用16位寄存器)

$$\begin{array}{r} 1234 \text{ F00F} \\ + 0567 \text{ 80F0} \\ \hline 179\text{C}70\text{FF} \end{array}$$

C3\_062j.asm





## (3.2.2) 1.加法指令 (5)

例1: 计算 1234 F00FH + 1234 80F0H  
(只允许使用16位寄存器)

```
data segment
```

```
dn1 dd 12340f00fh
```

```
dn2 dd 056780f0h
```

```
sum dd ?
```

```
data ends
```

```
code segment
```

```
assume
```

```
cs:code,ds:data
```

```
start: mov ax, data
```

```
mov ds, ax
```

```
mov ax, word ptr dn1
```

```
add ax, word ptr dn2
```

```
mov word ptr sum, ax
```

```
mov ax, word ptr dn1+2
```

```
adc ax, word ptr dn2+2
```

```
mov word ptr sum+2, ax
```

```
mov ah, 4ch
```

```
int 21h
```

```
code ends
```

```
end start
```





华中科技大学

## (3.2.2) 1.加法指令 (6)

例2：考虑第2章2.2节5个数相加的例子：  
设 BUF DB 40,50,60,70,80,即以BUF为  
首址的字节区中存放有5个数据，求它们  
的和。





## (3.2.2) 1.加法指令 (7)

```
.386
STACK SEGMENT USE16 STACK
    DB 200 DUP(0)
STACK ENDS
```

```
SEG1 SEGMENT USE16
BUF   DB 40,50,60,70,80
RES   DW ?
SEG1 ENDS
```

```
CODE SEGMENT USE16
    ASSUME CS:CODE,
           DS:SEG1,SS:STACK
START:
    MOV AX, SEG1
    MOV DS, AX
```

```

XOR CX, CX
MOV AX, 0
MOV BX, OFFSET BUF
LP:  CMP CX,5
     JGE EXIT
     ADD AL, [BX]
     ADC AH, 0
     INC BX
     INC CX
     JMP LP

EXIT: MOV RES, AX
      MOV AX, 4C00H
      INT 21H
CODE ENDS
      END START
```





## (3.2.2) 2.减法指令 (1)

- |            |            |
|------------|------------|
| (1) 减1指令   | <b>DEC</b> |
| (2) 求补指令   | <b>NEG</b> |
| (3) 一般减指令  | <b>SUB</b> |
| (4) 带借位减指令 | <b>SBB</b> |
| (5) 比较指令   | <b>CMP</b> |

DEC对OF,SF,ZF,PF,AF有影响;  
其它指令对**CF**,OF,SF,ZF,PF,AF有  
影响; (**DEC不影响CF**)





## (3.2.2) 2.减法指令 (2)

### (1) 减1指令 (不影响CF)

**DEC OPD ; (OPD) -1 → OPD**

### (2) 求补指令

**NEG OPD ; (OPD)求反加1 → OPD**

执行如下程序段后, (AX)=?

**MOV AX, 20H**

**NEG AX**

**!!! 注意NEG指令与  
一般数的补码的区别**

**(AX)= 0FFE0 H**





## (3.2.2) 2.减法指令 (3)

### (3) 一般减指令

**SUB OPD, OPS ;(OPD)-(OPS) → OPD**

例: **A DW 50**

**B DW 100**

**MOV AX, 200**

**SUB AX, A ; (AX)=?**

; 下面要 **(B) - (A) → B**

~~**SUB B, A**~~

**MOV BX, A**  
**SUB B, BX**





## (3.2.2) 2.减法指令 (4)

### (4) 带借位减指令

**SBB OPD, OPS**

**$(OPD) - (OPS) - CF \rightarrow OPD$**

例: **MOV AX, 1234H**  
**SUB AL, 0F0H**  
**SBB AH, 0**

问: 此程序段的意义是什么?  
 **$(AX) = ?$**







## (3.2.2) 1.加法指令 (5)

例1: 2469 70FF - 1234 F00FH  
(只允许使用16位寄存器)

data segment

**dn1 dw 70FFh, 2469h**

**dn2 dd 1234F00FH**

sum dw 0,0

data ends

code segment

assume

cs:code,ds:data

start: mov ax, data

mov ds, ax

mov ax, dn1

sub ax, word ptr dn2

mov sum, ax

mov ax, dn1+2

**sbb ax, word ptr dn2+2**

mov sum+2,ax

mov ah, 4ch

int 21h

code ends

end start





## (3.2.2) 2.减法指令 (6)

### (5) 比较指令

**CMP OPD, OPS ; (OPD) – (OPS)**

比较指令的作用是什么？

```
CMP AX, 0  
JNE T
```

```
... ..
```

```
T: ... ..
```





## (3.2.2) 2.减法指令 (7)

例2：分析下面的程序片段，指出其功能。

```
CMP      AX,  0
JGE      EXIT
NEG      AX
... ..
EXIT:    ... ..
```

程序功能：求(AX)的绝对值



## (3.2.2) 2.减法指令 (8)

例3：分析下面的程序片段，指出每条语句的执行结果。

	<b>XOR</b>	<b>AX, AX</b>
	<b>DEC</b>	<b>AX</b>
	<b>JC</b>	<b>EXIT</b>
	<b>INC</b>	<b>AX</b>
	<b>SUB</b>	<b>AX, 1</b>
	<b>JC</b>	<b>EXIT</b>
	...	...
<b>EXIT:</b>	...	...





华中科技大学

## (3.2.2) 3.乘法指令 (1)

(1) 有符号数乘法 **IMUL**

(2) 无符号数乘法 **MUL**





## (3.2.2) 3.乘法指令 (2)

### (1) 有符号乘法

#### ■ 双操作数的有符号乘指令

语句格式: **IMUL OPD, OPS**

功 能: **(OPD) \* (OPS) → OPD**

说 明: **OPD** 为 16/32位寄存器

**OPS** 为同类型的寄存器、存储器  
操作数或立即数。

例: **IMUL AX, BX**

**IMUL EAX, DWORD PTR [SI]**

**IMUL AX, 3**





## (3.2.2) 3.乘法指令 (3)

### ■ 3个操作数的有符号乘指令

语句格式: **IMUL OPD, OPS, n**

功 能:  **$(OPS) * n \rightarrow OPD$**

说 明: **OPD** 为 16/32位寄存器

**OPS**为同类型的寄存器、存储器  
操作数或立即数。

例 : **IMUL AX, BX, -10**

**IMUL EAX, DWORD PTR [SI], 5**

**IMUL BX, AX, 3**





## (3.2.2) 3.乘法指令 (4)

### ■单操作数的有符号乘法

语句格式: **IMUL OPS**

字节乘法: **(AL)\*(OPS) → AX**

字乘法: **(AX)\*(OPS) → DX, AX**

双字乘法: **(EAX)\*(OPS) → EDX, EAX**

说明: **OPS**不能是立即数

如果乘积的高位(**AH,DX,EDX**)不是低位(**AL,AX,EAX**)的符号位扩展, 而是包含其他二进制位, 则 **CF=1, OF=1**.







## (3.2.2) 3.乘法指令 (5)

### (2) 无符号乘法

语句格式: **MUL OPS**

**字节乘法:**  $(AL) * (OPS) \rightarrow AX$

**字乘法:**  $(AX) * (OPS) \rightarrow DX, AX$

**双字乘法:**  $(EAX) * (OPS) \rightarrow EDX, EAX$

说明: **OPS**不能是立即数

如果乘积的高位(**AH,DX,EDX**)  
不是全**0**, 则 **CF=1, OF=1**.





## (3.2.2) 3.乘法指令 (6)

### 无符号乘法与有符号乘法的比较

```
code segment
    assume cs:code
begin:
```

```
    mov al, 10H
    mov bl, 0FEH
    imul bl
```

**(ax)=0FFE0H**, 结果高  
字节无有效位, **NC, NV**

```
    mov al, 10H
    mul bl
```

**(ax)=0FE0H**, 结果高字  
节有有效位, **CY, OV**

```
mov al, 0F0h
mov bl, 2
imul bl
```

**(ax) = 0FFE0H**

```
mov al, 0F0h
mov bl, 2
mul bl
```

**(ax) = 01E0H**

```
mov ah, 4ch
int 21h
```

```
code ends
end begin
```





## (3.2.2) 3.乘法指令 (7)

### 乘法指令小结

IMUL R16/R32, OPS

IMUL R16/R32, OPS, n

**OPS与OPD  
类型相同**

IMUL OPS

MUL OPS

字节乘法

AX

字乘法

DX, AX

双字乘法

EDX, EAX





## (3.2.2) 4.除法指令 (1)

### (1) 有符号除法

**IDIV OPS**

字节除法:  $(AX) / (OPS) \rightarrow AL \text{ (商)}, AH \text{ (余)}$

字除法:  $(DX, AX) / (OPS) \rightarrow AX \text{ (商)}, DX \text{ (余)}$

双字除法:  $(EDX, EAX) / (OPS) \rightarrow EAX \text{ (商)}, EDX$

### (2) 无符号除法

**DIV OPS**

字节除法:  $(AX) / (OPS) \rightarrow AL \text{ (商)}, AH \text{ (余)}$

字除法:  $(DX, AX) / (OPS) \rightarrow AX \text{ (商)}, DX \text{ (余)}$

双字除法:  $(EDX, EAX) / (OPS) \rightarrow EAX \text{ (商)}, EDX$

**OPS**不能是立即操作数, 操作类型由**OPS**决定。





## (3.2.2) 4.除法指令 (2)

例：写出 4001H 除以 4 的程序段

```
MOV  AX, 4001H
MOV  DX, 0
MOV  CX, 4
DIV  CX
```

结果：

(AX) = 1000H

(DX) = 1

Question: 能否将上述程序段写成：

```
MOV  AX, 4001H
MOV  CL, 4
DIV  CL
```





华中科技大学

## (3.2.2) 5.符号扩展指令 (1)

### (1) 将字节转换成字

**CBW**

将AL中的符号扩展至AH中。

### (2) 将字转换成双字

**CWD**

将AX中的符号扩展至DX中。





华中科技大学

## (3.2.2) 5.符号扩展指令 (2)

(3) 将AX中的有符号数扩展为32位送EAX  
CWDE

(4) 将EAX中的有符号数扩展为64位数  
送 EDX, EAX  
CDQ

问题: 请用MOVSX指令实现CBW和CWDE





## (3.2.2) 5.符号扩展指令 (3)

例：指出下面的语句执行结果

**MOV AX, 0**

**MOV AL, -2**

**CBW** ;AX=?

**CWD** ;?

**CWDE** ;?







## (3.2.2) 6.一个综合例子 (1)

```
DATA    SEGMENT
AA      DW  a
BB      DW  b
CC      DW  c
DDO     DW  2 DUP(0)
DATA    ENDS
STACK   SEGMENT STACK
        DB 200 DUP(0)
STACK   ENDS
CODE    SEGMENT
        ASSUME DS:DATA,
        SS:STACK,
        CS:CODE
```

```
START:  MOV  AX, DATA
        MOV  DS, AX
        MOV  AX, AA
        IMUL AX, BB
        ADD  AX, CC
        SUB  AX, 70
        CWD
        IDIV AA
        MOV  DDO, AX
        MOV  DDO+2, DX
        MOV  AX, 4C00H
        INT  21H
CODE    ENDS
END     START
```

程序功能：计算  $(a*b+c-70)/a$ ，并将  
结果保存到DDO，余数存入DDO+2





## (3.2.2) 6. 一个综合例子 (2)

设**a**变量是用**dw**定义的16位有符号数，**b**和**c**变量用**db**定义的8位有符号数，编写程序段计算**a\*b/c**，将商和余数分别保存在变量**D**和**R**中。

提示：需要考虑溢出的情况

在上面的例子中，若**a, b, c**是相应的无符号数呢？

```
a  dw  ?  
b  db  ?  
c  db  ?  
D  dd  ?  
R  db  ?
```

```
.....  
movsx  eax, a  
movsx  ebx, b  
imul    ebx    ;edx:eax  
movsx  ebx, c  
idiv    ebx  
mov     D, eax  
mov     R, dl  
.....
```





## (3.2.2) 6. 一个综合例子 (3)

设a变量是用dw定义的16位无符号数，b和c变量用db定义的8位无符号数，编写程序段计算 $(a*b - c) / a$ ，将商和余数分别保存在变量D和R中。

提示：需要考虑溢出的情况

```
a    dw    ?
b    db    ?
c    db    ?
D    dd    ?
R    db    ?

.....
movzx    eax, a
movzx    ebx, b
mul      ebx    ;edx:eax
movzx    ebx, c
sub      eax, ebx
sbc      edx, 0
movzx    ebx, a
div      ebx
mov      D, eax
mov      R, dl
.....
```





## 3.2.3 位操作指令

**1、逻辑运算指令** NOT AND OR XOR  
TEST BT

**2、移位指令** SHL SAL SHR SAR  
ROL ROR RCL RCR  
SHLD SHRD





华中科技大学

## (3.2.3) 1.逻辑运算指令 (1)

### (1) 求反

**NOT OPD** ; (OPD)求反 $\rightarrow$ OPD

### (2) 与(逻辑乘)

**AND OPD, OPS** ; (OPD) $\wedge$ (OPS)  $\rightarrow$ OPD

### (3) 或(逻辑加)

**OR OPD, OPS** ; (OPD) $\vee$ (OPS)  $\rightarrow$ OPD

### (4) 异或(按位加)

**XOR OPD, OPS** ; (OPD)异或(OPS)  $\rightarrow$ OPD





华中科技大学

## (3.2.3) 1.逻辑运算指令 (2)

例1：指出各指令执行后，(AX)=?

**MOV AX, 1234H**

**NOT AX ; (AX) = ? 0EDCBH**

**AND AX, 0FH; ; (AX)= ? 000BH**

**OR AX, 1255H ; (AX)= ? 125FH**

**XOR AX, AX ; (AX)= ? 0000H**





## (3.2.3) 1.逻辑运算指令 (3)

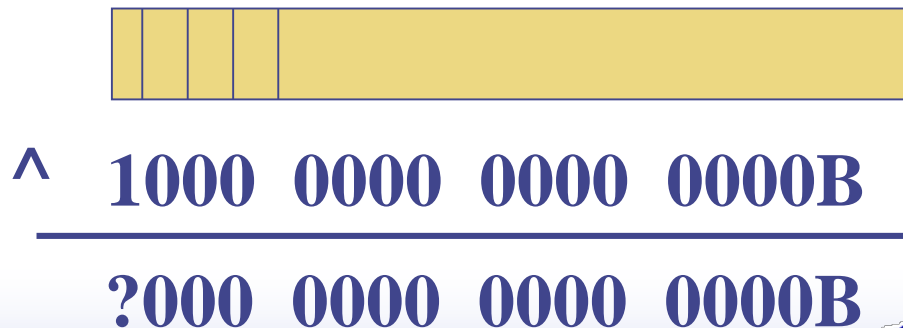
### (5) 测试指令(1) TEST OPD, OPS

功能：根据  $(OPD) \wedge (OPS)$  设置标志位，  
(OPD)、(OPS)不变。

例：判断(AX)的最高位是否为0，若为0，转L

TEST AX, 8000H

JZ L





## (3.2.3) 1.逻辑运算指令 (4)

### 测试指令(2) BT OPD, OPS

功能：将(OPD)中的第(OPS)位  $\rightarrow$  CF,  
(OPD)、(OPS)不变。

说明：

- OPD 必须为16/32位的寄存器或存储单元；
- OPS 为立即数或与OPD同类型的寄存器操作数；
- OPS的值应在-128~ 255之间；
- 当OPS的绝对值大于OPD的位数时，位数为OPS除16或32后的余数。

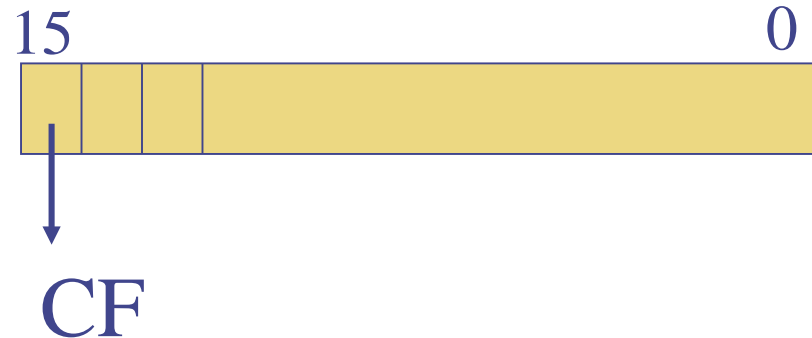




## (3.2.3) 1.逻辑运算指令 (5)

例：判断(AX)的最高位是否为0，若为0，转L

```
BT  AX, 15  
JNC L
```





## (3.2.3) 2.移位指令 (1)

- (1) 算术左移 **SAL** Shift **A**rithmetic **L**eft
- (2) 逻辑左移 **SHL** Shift Logical **L**eft
- (3) 逻辑右移 **SHR** Shift Logical **R**ight
- (4) 算术右移 **SAR** Shift **A**rithmetic **R**ight
- (5) 循环左移 **ROL** Rotate **L**eft
- (6) 循环右移 **ROR** Rotate **R**ight
- (7) 带进位的循环左移 **RCL** Rotate left through Carry
- (8) 带进位的循环右移 **RCR**
- (9) 双精度左移 **SHLD**
- (10) 双精度右移 **SHRD**





## (3.2.3) 2.移位指令 (2)

语句格式:

**操作符 OPD n 或 CL**

功能: 将(OPD)中的所有位按**操作符**规定的方式移动, 结果存在**OPD**对应的单元中。

特别说明:

**OPD**可以是寄存器, 也可以是地址表达式。

在**8086**指令系统中, **n**只能是**1**, 当移动位数操过**1**时, 要使用**CL**。



## (3.2.3) 2.移位指令 (3)

(1) **算术左移** SAL OPD, n

(2) **逻辑左移** SHL OPD, n

(OPD)向左移动n位, 低位补0



SAL AX, 3 等价于

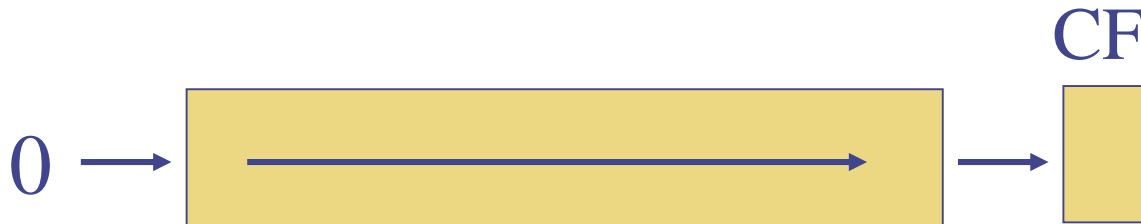
SAL AX, 1

SAL AX, 1

SAL AX, 1 ; CF为执行最后一次移位时送入的值

## (3.2.3) 2.移位指令 (4)

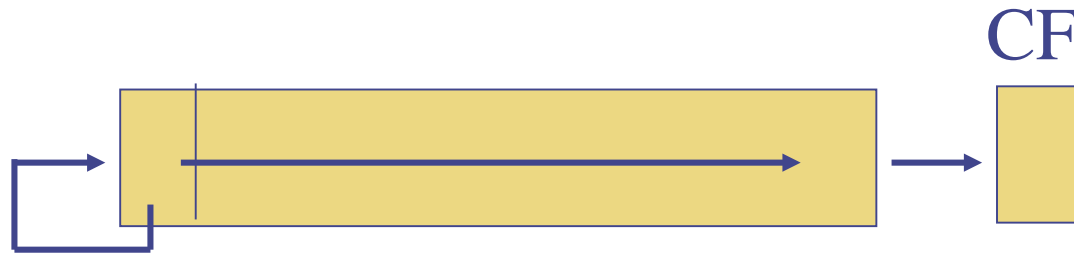
**(3)逻辑右移 SHR OPD, n**  
(OPD) 向右移动n位, 高位补0



```
MOV    AH, 5
SHR     AH, 1
      ; (AH)= ?      2
      ; (CF)= ?      1
```

## (3.2.3) 2.移位指令 (5)

**(4)算术右移 SAR** OPD, n  
(OPD)向右移动n位, 最高位不变。



MOV AH, 0F5H

SHR AH, 1

; (AH)= ? **7AH**

; (CF)= ? **1**

MOV AH, 0F5H

SAR AH, 1

; (AH)= ? **0FAH**

; (CF)= ? **1**

比较两种指令, 其结果说明什么?

## (3.2.3) 2.移位指令 (6)

### (5) 循环左移 ROL OPD, n

将(OPD)的最高位与最低位连接起来，组成一个环。将环中的所有位一起向左移动n位，CF的内容为最后移入位的值。



```
MOV DL, 0EAH
```

```
ROL DL, 4
```

(DL) = ?

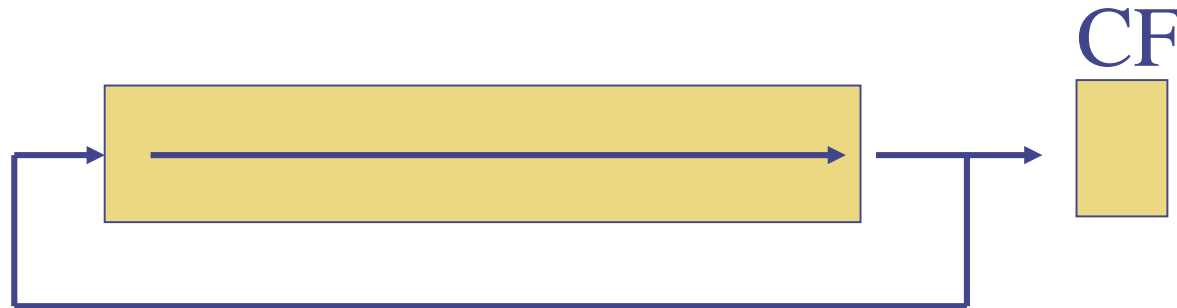
CF = ?

**0AEH 0**

## (3.2.3) 2.移位指令 (7)

### (6) 循环右移 ROR OPD, n

将(OPD)的最高位与最低位连接起来, 组成一个环。将环中的所有位一起向右移动n位, CF的内容为最后移入位的值。



```
MOV DL, 0EAH
```

```
ROR DL, 4
```

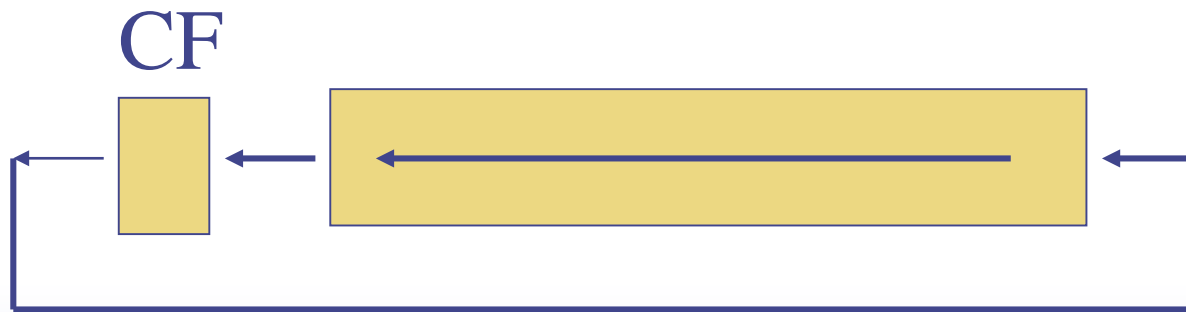
(DL) = ?    CF = ?    **0AEH**    **1**



## (3.2.3) 2.移位指令 (8)

### (7) 带进位的循环左移 RCL OPD, n

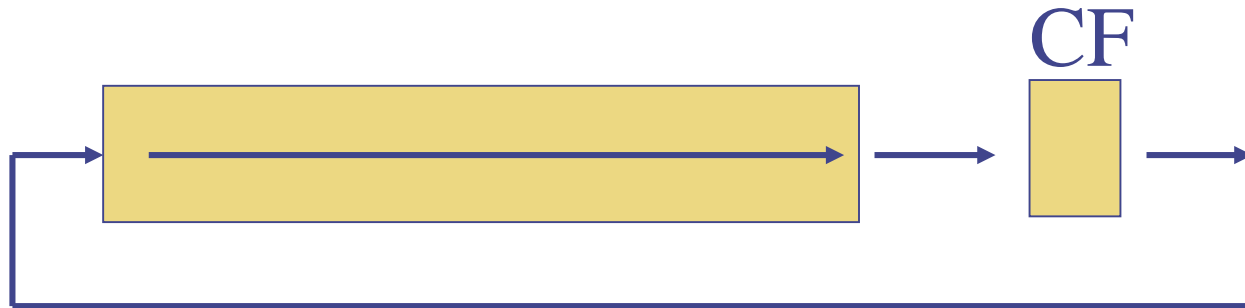
将 (OPD) 的最高位、CF、(OPD) 最低位连接起来，组成一个环。将环中的所有位一起向左移动n位，CF的内容为最后移入位的值。



## (3.2.3) 2.移位指令 (9)

### (8) 带进位的循环右移 RCR OPD, n

将(OPD)、CF连接起来，组成一个环。将环中的所有位一起向右移动n位，CF的内容为最后移入位的值。

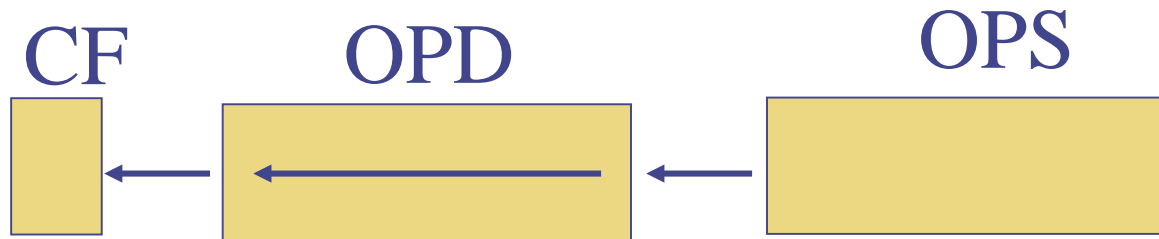


## (3.2.3) 2.移位指令 (10)

### (9) 双精度左移指令

SHLD OPD, OPS, n

将(OPD)向左移动n位, 其最右端的数来自(OPS)的高n位, CF的内容为最后移入位的值;  
(OPS)不变。



说明: OPS只能为与OPD类型相同的16位  
或32位的寄存器

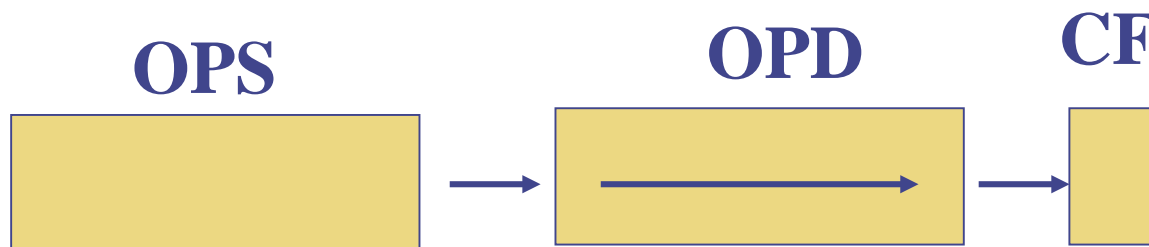


## (3.2.3) 2.移位指令 (11)

### (10) 双精度右移指令

SHRD OPD, OPS, n

将(OPD)向右移动n位，其最左端的数来自(OPS)的低n位，CF的内容为最后移入位的值。(OPS)不变。



说明：OPS只能为与OPD类型相同的16位或32位的寄存器





## 3.2 常用的机器指令语句 (小结-1)

### 一、数据传送指令

#### 1、一般数据传送指令

**MOV、MOVSX、MOVZX、XCHG、XLAT**

#### 2、堆栈操作指令

**PUSH、POP、PUSHA、PUSHAD、POPA、POPAD**

#### 3、标志寄存器传送指令

**PUSHF、POPF、PUSHFD、POPFD、LAHF、SAHF**

#### 4、地址传送指令

**LEA, LDS, LES, LFS, LGS, LSS**

#### 5、输入、输出指令

**IN、OUT**



## 3.2 常用的机器指令语句 (小结-2)



华中科技大学

### 二、算术运算指令

#### 1、加法指令

**INC、ADD、ADC**

#### 2、减法指令

**DEC、NEG、SUB、SBB、CMP**

#### 3、乘法指令

**IMUL (三种形式)、MUL**

#### 4、除法指令

**IDIV、DIV**

#### 5、符号扩展指令

**CBW、CWD、CWDE、CDQ**



## 3.2 常用的机器指令语句 (小结-3)



华中科技大学

### 三、位操作指令

#### 1、逻辑运算指令

**NOT、AND、OR、XOR、TEST**

#### 2、移位指令

**SAL、SHL**

**SAR、SHR**

**ROL、ROR**

**RCL、RCR**



## 3.2 常用的机器指令语句 (小结-4)



华中科技大学

试用不同指令将 (AX)置0。

```
MOV  AX, 0
SUB  AX, AX
AND  AX, 0
XOR  AX, AX
SHL  AX, 16
```

试用不同的指令，将AX的高、低字节内容互换。

```
XCHG AH, AL
ROL  AX, 8
ROR  AX, 8
```







## 3.3 伪指令语句 (1)

**伪指令：**告诉汇编程序（编译器）如何把汇编语言源程序翻译成二进制的机器语言目标程序的辅助性指令。它告诉汇编程序如何工作，但它本身不产生任何目标代码。如DB、DW、EQU等等。





## 3.3 伪指令语句 (2)

### 伪指令与机器指令的区别:

#### (1) 功能不同

机器指令控制CPU的工作，并产生目标代码；  
伪指令 控制汇编程序工作，不产生目标代码。

#### (2) 执行的时间不同

用户程序在运行时，执行机器指令；  
汇编程序在运行时，执行伪指令。

#### (3) 实现方式不同

机器指令是用硬件线路来实现其功能的；  
伪指令是用汇编程序来实现其功能的，它在汇编期间被执行，在目的代码中已不存在了。





## 3.3 伪指令语句 (3)

- ① 处理器选择伪指令
- ② 数据定义伪指令
- ③ 符号定义伪指令
- ④ 段定义伪指令
- ⑤ 过程定义伪指令
- ⑥ 程序模块的定义与通讯伪指令
- ⑦ 宏定义伪指令
- ⑧ 条件汇编伪指令
- ⑨ 格式控制、列表控制等伪指令

伪指令  
分类



## 3.3.1 处理器选择伪指令 (1)



华中科技大学

告诉汇编程序选择何种**CPU**所支持的指令系统

伪指令	功 能	伪指令	功 能
. 8086	接受8086指令 (缺省方式)	. 586	接受Pentium指令 (除特权指令)
. 386	接受80386指令 (除特权指令)	. 586P	接受全部Pentium指令
. 386P	接受全部80386指令	. 686	接受Pentium Pro指令 (除特权指令)
. 486	接受80486指令 (除特权指令)	. 686P	接受全部Pentium Pro指令
. 486P	接受全部80486指令	. MMX	接受MMX指令



## 3.3.1 处理器选择伪指令 (2)



华中科技大学

- 一个程序一般只选择一种指令系统;
- 处理器选择伪指令一般放在程序的开始处;
- 如果整个源程序都未使用处理器选择伪指令, 则汇编程序将选择 **缺省值 .8086**;
- MASM不同版本支持的指令系统不同。
- 处理器选择伪指令也可以放在段中, 表示从紧接着的语句开始, 使用新指定的处理器指令系统, 直到遇到一个新的处理器选择伪指令为止。





华中科技大学

## 3.3.2 数据定义伪指令

语句格式:

[变量名] **数据定义伪指令** 表达式 [, .....]

功能: 定义一数据存储区, 其类型由所使用的数据定义伪指令指定。

数据定义伪指令有: **DB、DW、DD、**  
**DF、DQ、DT。**





### 3.3.3 符号定义伪指令

等价伪指令 EQU

等号伪指令 =

格式：符号名 EQU 表达式

符号名 = 表达式

功能：用来为常量、表达式及其它符号定义一个等价的符号名，但它**并不占用**存储单元。

例： A EQU 20

B = 20

**EQU 和 = 的区别：**一个符号名只能在程序中用 EQU 定义一次，而可以用 = 重新定义多次。



### 3.3.4 段定义伪指令 (1)



华中科技大学

#### 1. 段定义伪指令

段名 **SEGMENT** [使用类型] [定位方式]  
[组合方式] ['类别']

.....

段名 **ENDS**

- 一个程序模块可以由若干段组成;
- 段名可以各不相同, 也可以重复;
- 汇编程序将一个程序中的同名段处理成一个段
- 段的定义可以嵌套, 但不能交叉。





### 3.3.4 段定义伪指令 (2)



华中科技大学

**DATA** SEGMENT

USE16

AA DW 10H, - 20H

**DATA** ENDS

CODE SEGMENT

USE16

.....

**DATA** SEGMENT USE16

BUF DB 'ABC'

**DATA** ENDS

.....

CODE ENDS

AA

10H

00H

0E0H

0FFH

BUF

41H

42H

43H



### 3.3.4 段定义伪指令 (3)



华中科技大学

“使用类型” 包括两种：

- **USE16**: 表示该段为**16位段**，段的最大长度为**64 KB**，地址的形式是**16位段地址**和**16位偏移地址**，寻址方式为**16位寻址方式**；
- **USE32**: .....
- 使用了伪指令 **.386(或以上)**，“使用类型”才起作用
- 如果无“使用类型”，而又使用了**.386**及以上处理器选择伪指令，则**默认该段为32位段**；
- 如果仅在该段内使用**.386**及以上处理器选择伪指令，则汇编程序仍然认为该段为**16位段**。
- 在实方式和虚拟**8086**方式中，段的大小只能为**64 KB**，因此，运行**.386**以上的汇编源程序必须使用**USE16**。**只有在保护方式下才使用32位段。**



### 3.3.4 段定义伪指令 (4)



华中科技大学

处理器选择伪指令	使用类型	16位段/32位段
无	不起作用	16位段
.386	无	默认32位段
<b>.386</b>	<b>USE16</b>	<b>16位段</b>
.386	USE32	32位段



### 3.3.4 段定义伪指令 (5)



华中科技大学

#### 2. 假定伪指令

格式: **ASSUME** 段寄存器:段名 [, .....]

功能: 用来设定段寄存器与段之间的对应关系,  
即告诉汇编程序, 该段中的变量或标号  
用哪个段寄存器作段首址指示器。

Q: 为什么要建立段与段寄存器之间的联系?

汇编程序遇到一个变量时, 就会在ASSUME所指定的段中寻找, 如果ASSUME没有指令定义该变量的段, 就会出错。



### 3.3.4 段定义伪指令 (6)



华中科技大学

**ASSUME 段寄存器: 段名 [, .....]**

说明:

- 该语句一般出现在代码段中。
- 段名是程序中某一已定义段的名称
- 段名也可以是表达式 “**SEG 变量或标号**”。
- 该语句不一定出现在第一行
- **ASSUME 段寄存器: NOTHING**



## 3.3.4 段定义伪指令 (7)



华中科技大学

.386

**A** SEGMENT USE16

AA DW 0FFH

**A** ENDS

**B** SEGMENT USE16 STACK  
DB 100 DUP(?)

**B** ENDS

**D** SEGMENT USE16

DD DW 11H

**D** ENDS

**E** SEGMENT USE16

ASSUME CS:E,  
SS:B, DS:A, ES:D

MOV AX, A

MOV DS, AX

MOV ES, AX

MOV AH, BYTE PTR AA+1

MOV BX, DD

MOV CX, D

MOV ES, CX

MOV BX, DD

.....



### 3.3.4 段定义伪指令 (8)



华中科技大学

#### 何时将段首址置入对应的段寄存器？

- 目标程序运行时才能给段寄存器置值。
- **CS和SS**的内容将由操作系统**自动设置**；
- **DS和ES**的内容须由用户程序**指令设置**；
- 一定要将段首址送入DS或ES中，才能保证目标程序运行时，能正确地产生数据存储单元的物理地址。



### 3.3.4 段定义伪指令 (9)



华中科技大学

#### 3. 置汇编地址计数器伪指令

- 汇编地址计数器：\$  
表示当前位置的偏移地址
- 汇编程序在翻译程序时，每遇到一个新的段，就将汇编地址计数器置0。

#### 4. 地址定义伪指令

##### ORG 地址表达式

地址表达式的值表示紧跟它后面的指令/语句将从这个地址偏移处开始。其意义也就是将汇编地址计数器\$的值设置成地址表达式的值。





## 3.3.4 段定义伪指令 (10)



华中科技大学

**DATA SEGMENT**

**BUF1 DB 'AB'**

**X DW \$, BUF2 - \$**

**ORG \$ + 10H**

**BUF2 DB '1', 1, -1**

**Y DB \$ - BUF2**

**Z DB \$ - BUF1**

**DATA ENDS**

41H	00H, BUF1
42H	01H
02H	02H, X
00H	03H
12H	04H
00H	05H
.....	06H
31H	16H, BUF2
01H	17H
0FFH	18H
03H	19H, Y
1AH	1AH, Z





## 3.3.5 源程序结束伪指令

格式: **END** [表达式]

功能: 遇到该语句时, 汇编工作停止。

- 如果END后面带有表达式, 其值必须是一存储器地址。该地址为程序的启动地址, 即该程序运行时第一条被执行指令的地址。
- 如果不带表达式, 则说明该程序不能单独运行, 这时, 它往往是作为一个子模块供另外的程序调用。

注意: 不可将END语句错误地安排在程序中间。





华中科技大学

## 3.4 常用的DOS系统功能调用

### 一、什么是DOS系统调用？

调用操作系统提供的功能。提供的功能调用包括：设备管理、文件管理、目录管理及其它功能。参见P326。

### 二、DOS系统功能调用的一般过程

将调用号放入寄存器AH中，置好入口参数，然后执行 INT 21H。





## 3.4 常用的DOS系统功能调用

### 三、常用的输入/输出系统功能调用

程序退出并返回操作系统

键盘输入 (输入一个字符)

显示输出 (输出一个字符)

显示字符串

键盘输入字符串





华中科技大学

## (3.4) 1.程序退出并返回操作系统

调用格式: **MOV AH, 4CH**

**MOV AL, 退出码 (如0、0FFH等)**

**INT 21H**

功能: 终止当前程序并返回操作系统, 其中**AL**是返回值 (退出值)。





## (3.4) 2.键盘字符输入(1号调用)

调用格式: **MOV AH, 1**  
**INT 21H**

功能: 等待从键盘输入一个字符并将输入字符的**ASCII**码送入寄存器**AL**中。

**说明:** 首先检查按键是否为Ctrl + Break, 若是, 则从本调用的执行中退出; 否则, 将按键的**ASCII**码→**AL**, 同时将该字符送显示器显示。





## (3.4) 3.显示字符(2号调用)

调用格式:   MOV   DL, 待显示字符的ASCII码  
              MOV   AH, 2  
              INT   21H

功能: 将DL中的字符送显示器显示, 若DL中为〈CTRL〉+〈Break〉的ASCII码, 则从本调用的执行中退出。

例1   MOV   DL, '3'                               ; 显示字符3  
      MOV   AH, 2  
      INT   21H





## (3.4) 4.显示字符串(9号调用)

调用格式: LEA DX, 字符串首偏移地址  
MOV AH, 9  
INT 21H

功能: 将从数据区DS: DX所指向的单元开始,  
依次显示字符, 直到遇到 ' \$' 为止。

Q: 此处的 '\$' 与汇编地址计数器\$有何不同?







## (3.4) 4.显示字符串(9号调用)

```
.386
DATA SEGMENT USE16
BUF DB 0AH, 0DH, 'I WISH YOU SUCCESS! $'
DATA ENDS
CODE SEGMENT USE16
    ASSUME CS: CODE, DS: DATA
START: MOV AX, DATA
        MOV DS, AX
        LEA DX, BUF
        MOV AH, 9
        INT 21H
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

C3\_085P.asm





## (3.4) 4.显示字符串(9号调用)

```
STACK SEGMENT STACK
        DB 200 DUP(0)
STACK ENDS
```

```
DATA SEGMENT
T1  DB 'abcdef'
T2  DB 31H,32H,33H,'$',34H,35H
DATA ENDS
```

```
CODE SEGMENT
        ASSUME
CS:CODE,DS:DATA,SS:STACK
```

```
BEGIN:
        MOV AX,DATA
        MOV DS,AX
```

```
        LEA DX,T1
        MOV AH,9
        INT 21H
        MOV AH,4CH
        INT 21H
```

```
CODE ENDS
        END BEGIN
```

C3\_085J2.asm



## (3.4) 5.键盘输出字符串(10号调用)



华中科技大学

格式: **LEA DX, 缓冲区首偏移地址**  
**MOV AH, 10**  
**INT 21H**

功能: 从键盘上往**DS : DX**所指的输入缓冲区输入字符串（以回车键结束）并送显示器显示。



## (3.4) 5.键盘输出字符串(10号调用)



华中科技大学

### 输入缓冲区的定义及输入数据的存放：

- 第一个字节规定了最多能够输入的字符数（包括回车键），此字节不能是0。
- 第二个字节用于输出实际输入的**有效字符个数**（不包括回车键）。
- 从键盘输入的字符串从第三个字节存放，最后以回车(0DH)作结束。回车符的ASCII码也被送入缓冲区，但不计入输入的字符个数之中。如果输入的字符个数超过了缓冲区的大小，则多余字符被删除且扬声器响。



## (3.4) 5.键盘输出字符串(10号调用)



华中科技大学

### 缓冲区的定义

<b>BUF</b>	<b>DB</b>	<b>80</b>	;	最多可以输入79个有效字符	
	<b>DB</b>	<b>?</b>	;	用于返回输入的有效字符数	
	<b>DB</b>	<b>80</b>	<b>DUP(0)</b>	;	用于存放字符串的ASCII码,
				;	最后一个字节是回车键0DH



## (3.4) 5.键盘输出字符串(10号调用)



华中科技大学

**DATA SEGMENT USE16**

**BUF DB 50**

**DB 0**

**DB 51 DUP(0)**

**CRLF DB 0DH, 0AH, '\$'**

**DATA ENDS**

**STACK SEGMENT USE16 STACK**

**DB 100 DUP(?)**

**STACK ENDS**

**CODESEGMENT USE16**

**ASSUME DS:DATA,**

**CS:CODE, SS:STACK**

**START: MOV AX, DADA**

**MOV DS, AX**

**LEA DX, BUF**

**MOV AH, 10H**

**INT 21H**

**LEA DX, CRLF**

**MOV AH, 9**

**INT 21H**

**MOV BL, BUF+1**

**MOV BH, 0**

**MOV BYTE PTR BUF+2[BX], '\$'**

**LEA DX, BUF+2**

**MOV AH, 9**

**INT 21H**

**MOV AX, 4C00H**

**INT 21H**

**CODE ENDS**

**END START**





## 3.5 上机操作 (1)

**一个汇编语言源程序需要经过汇编（编译）、连接才能形成可执行的目标程序（.EXE）。对目标程序进行调试需要用源码级的调试程序。**

- **汇编程序：将.ASM的汇编语言源程序编译成.OBJ文件。**
- **连接程序：将.OBJ文件转换成可执行的.EXE文件。**
- **调试程序：对可执行的.EXE目标程序进行调试。**





## 3.5 上机操作 (2)

汇编程序、连接程序和调试程序有：

- MicroSoft公司的MASM、LINK、CV
- Borland公司的TASM、TLINK、TD

本课程选用：**MASM、LINK、TD**

假设test.asm是源程序，命令格式如下：

<b>MASM</b>	<b>/l test.asm;</b>	<b>;产生 test.obj 、 test.lst文件</b>
<b>LINK</b>	<b>test;</b>	<b>;根据 test.obj 产生 test.exe</b>
<b>TD</b>	<b>test.exe</b>	<b>;调试 test.exe</b>

若源程序有错误，则用记事本将 test.lst 打开。







## 3.5 上机操作 (3)

### 操作流程:

- (1) 将MASM6.0 软件 (包含MASM、LINK和TD) (可以从<http://202.114.1.86> 下载) 安装到D盘的MASM目录;
- (2) 用记事本建立汇编源程序 (后缀为.ASM) ;
- (3) 切换到DOS操作系统 ( “开始→程序→附件→命令提示符” ; 或 “开始→运行→ cmd ” ) ;
- (4) 在C>状态下输入: “D: 回车” 和 “cd \MASM”
- (5) 用前面介绍的MASM、LINK、TD命令编译、连接和调试程序。

