第三章 宏汇编语言

华中科技大学

学习指导

一、学习目标与要求

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

第三章 宏汇编语言

华中科技大学

学习指导

二、学习重点

1.宏汇编语言中的表达式

(1) 符号常量

(2) 变量(数据在主存中的存储示意图)

(3) 地址表达式

(4) 属性定义算符 (PTR,跨段前缀)

(5) 属性分离算符 (SEG, OFFSET)

第三章 宏汇编语言

学习指导

二、学习重点

2. 常用的机器指令语句

(1) 数据传送指令

(2) 算术运算指令

(3) 位操作指令

要求掌握各指令的语法规定，功能，最常用指令对标志寄存器的影响。

华中科技大学

学习指导

二、学习重点

3. 常用的伪指令

(1) 数据定义伪指令

(2) 符号定义伪指令

(3) 段定义伪指令

(4) 假定伪指令

(5) 源程序结束伪指令

(6) 汇编地址计数器$

华中科技大学

学习指导

二、学习重点

4. 常用的DOS系统功能调用

1，2，9，10 号

特别注意特殊字符的显示效果：

0AH, 0DH

特别注意 ‘$’ (24H) 的作用。

第三章 宏汇编语言

学习指导

三、学习难点

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

第三章 宏汇编语言

3.1 宏汇编语言的基本语法

1. 常量与数值表达式

(1) 常量

(2) 数值表达式

2. 变量、标号与地址表达式

(1) 变量

(2) 标号

(3) 地址表达式

1. 常量与数值表达式

华中科技大学

(1) 常量

C语言中的常量定义：#define pi=3.1415926

常量的基本概念：汇编时已有确定的数值的量。

- 用途：
  - 机器指令语句中的立即操作数；
  - 也可作存贮器操作数的组成部分(位移量V)；
  - 在数据定义伪指令语句中给变量赋初值；
- 分类：

  数值常量、符号常量

1. 常量与数值表达式——常量

符号常量的定义：

等价伪指令 EQU

等号伪指令 =

例1

例2

- 使用：定义后直接引用符号名。
- 注意：

①符号常量不分配存贮单元，只建立等价代换关系，可出现在任何段。

②用EQU语句定义的符号常量在该程序中不能再重新赋值，而用“=”定义的符号常量可多次重新赋值，使用时，以最后一次定义的值为准。

符号常量

例1：

```
DATA    SEGMENT  USE16
NUMBER  EQU  4;
COUNT    =  35
TAB      DW   70, 80H, -5, NUMBER
         ⋮
MOV  AX, NUMBER
MOV  SI,  COUNT
MOV  DX, TAB
COUNT  = 10
MOV  BX, COUNT[ECX]
MOV  CX, TAB[ECX]    ；注意这两条指令的区别
```

- 符号常量特点：
  - ① 在汇编期间被代换成相应等价的数据；
  - ② 提高程序的可读性；
  - ③ 便于随时修改程序中的参数。

符号常量

例2：

```
        .386                                ;选择的处理器为386

数据段  DATA   SEGMENT  USE16              ;USE16定义了16
                                            位的段
        SUM    DW  0                        ;SUM为字变量，初值为0
        DATA   ENDS

堆栈段  STACK  SEGMENT  USE16  STACK
        DB     100  DUP(0)                  ;堆栈的大小为100个字节
        STACK  ENDS
```

符号常量


```
CODE  SEGMENT  USE16
 ASSUME  CS:CODE,  SS:STACK,  DS:DATA
 NUM1  EQU  1
 NUM2  =    50
 START: MOV  AX,          } ; 数据段首址→ DS,
        MOV  DS, AX       }   DS必须由用户程序自己设置
        MOV  CX, NUM2     ; 循环计数器置初值
        MOV  AX, 0        ; 累加器置初值
        MOV  BX, NUM1     ; 1→BX
NEXT:   ADD  AX, BX       ; (AX)+(BX)→AX
        INC  BX           }
        INC  BX           } ; (BX)+2→BX
        DEC  CX           ; (CX)-1→CX
        JNE  NEXT         ; (CX)≠0转NEXT
        MOV  SUM, AX      ; (CX)=0累加结果→SUM
        MOV  AH, 4CH      }
        INT  21H          } ; 返回DOS
CODE  ENDS
END  START
```

代码段

1. 常量与数值表达式——数值表达式

(2) 数值表达式

数值表达式：常量与运算符(算术运算、逻辑运算、关系运算)组成的有确定意义的式子。

①算术运算

+、−、*、/、MOD(模除，取余数)、SHR(右移)、SHL(左移)。

移位的特别说明：表示将二进制常量右移或左移运算符右边所规定的次数(正整数)，所空出的位数均补0。

1. 常量与数值表达式——数值表达式

②逻辑运算

逻辑乘：AND（与）

逻辑加：OR （或）

按位加：XOR（异或）

逻辑非：NOT （非）

③关系运算

相等：EQ

不等：NE

小于：LT

大于：GT

小于等于：LE

大于等于：GE

(3)数值表达式的运算时机

汇编期间进行，运算的结果为一数值常量

2. 变量、标号与地址表达式——变量

(1) 变量

变量：是数据段或附加数据段中一个数据存贮单元的名字，是这个存储单元的地址的符号表示。可代表一批存储单元的首址。

①变量的属性

段属性：定义变量所在段的段首址，当访问该变量时该段首址应在某一段寄存器中，即为CPU当前可访问段；

偏移地址：该变量所占存储单元到所在段的段首址的字节距离；

类型：类型是指存取该变量中的数据所需要的字节数，变量的类型由定义该变量时所使用的伪指令确定；

2. 变量、标号与地址表达式——变量

②变量的定义

一般在数据段或附加数据段中使用数据定义伪指令

DB、DW、DD、DQ和DT来定义

格式: **[变量名]** 数据定义伪指令 表达式[，…]

功能: 定义了一变量，并开辟了由变量属性所决定的一片连续存储区，其存储区

所占字节数=表达式个数*变量的类型。

变量的类型：存储单元的大小，由数据定义伪指令定义。

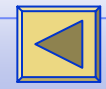
例

数据定义伪指令

数据定义伪指令：DB、DW、DD、DQ、DT指定变量的类型

.BYTE (字节)	DB
.WORD (字)	DW
.DWORD (双字)	DD
.FWORD (3个字)	DF
.QWORD (4个字)	DQ
.TBYTE (10个字节)	DT

例：
```
BUF   DB   'ABCD12EF……'  ;  BUF的类型为字节
ARR   DW   10，-60，189     ;  ARR的类型为字
TT    DD   0A57BD36H       ;  TT的类型为双字
```



表达式

变量定义中的表达式，指定了变量的初值：

注意

(i) 数值表达式

(ii) ASCII字符串

(iii) 地址表达式(只适用DW和DD两个伪指令)

(iv) ？ 变量值不确定

(v)重复子句: n DUP(表达式)，表示定义了n个数据存储单元

例

(vi)上述(i)～(v)组成的系列，各表达式之间用逗号隔开。

地址表达式

- **DW 地址表达式**

  当地址表达式含变量名时，初值取变量EA；

  例： A DW B

  A，B为变量，则A的初始值为B的偏移地址

- **DD 地址表达式**

  当地址表达式含变量名时，初始值取变量的EA，变量所在段的段首址

  例: A DW B

  B为变量，则A的初始值为B的偏移地址，段首址

重复子句：n DUP(表达式)

例：

DB 3 DUP('A', 12H) →

'A', 12H, 'A', 12H, 'A', 12H

DB 2 DUP('A', 2 DUP(3), 'B') →

'A', 3, 3, 'B', 'A', 3, 3, 'B'

表达式(注意)

①表达式的个数(包括(v)中的重复因子n)确定了存储单元的个数。

② 这一片连续的数据存储单元也称数据存储区，其类型由数据定义伪指令确定

③在定义一个数据存储区时：

- 变量仅代表该区的第一个数据存储单元;
- 整个数据存储区的类型均与变量相同;
- 建立了一个以变量为首址的数据存储区或以变量为名的数组

数据段定义的例子

例：数据段定义如下：

```
DATA  SEGMENT  USE16
A       DW    M
BUF     DB     'AB', 0DH, 0AH
CON     EQU  500H
B       DW    0FFAAH
MARK = 100H
D       DD    BUF
M       DB  2 DUP(1), 2 DUP(2, 'B')
        DB  '123', 1
DATA  ENDS
```

请画出数据段中的数据在主存中的存储形式。

问题：上例中若分别执行语句：

MOV AL, BUF

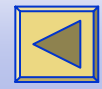
MOV AL, BUF+2 后AL的结果？

指令 MOV EDX, M 是否正确？

A	0CH
	0H
BUF	41H
	42H
	0DH
	0AH
B	0FFH
	0AAH
D	2H
	0H
	DATA

注意

M	1
	1
	2
	42H
	2
	42H
	31H
	32H
	33H
	1



本例中的注意事项

- 伪指令EQU及“=”不分配存贮单元；
- 使用直接寻址方式时，变量的类型必须与指令的要求相符；
- 变量的段必须是当前段。

2. 变量、标号与地址表达式——标号

(2) 标号

- **标号**：是机器指令语句存放地址的符号表示，也可以是子程序名，即子程序入口地址的符号表示；在代码段中定义和引用。
- **标号的属性**：

①标号的段属性：标号的段属性是指定义该标号所在段的段首址。

②标号的偏移地址：标号的偏移地址是指它所在段的段首址到该标号所代表存储单元的字节距离。

③标号的类型：分NEAR（近）和FAR（远）两类型，近标号在定义该标号的段内使用，远标号无此限制。

3. 变量、标号与地址表达式——地址表达式

(3)地址表达式

①地址表达式的定义

②接触过的地址表达式

③地址表达式的属性

④地址表达式与数值表达式区别

⑤特殊运算符

(i)类型运算符PTR

(ii)属性分离算符

⑥使用地址表达式的注意事项

①地址表达式的定义

地址表达式是由变量、标号、常量、寄存器(名加方括号)及一些运算符(数值表达式的运算符和特殊运算符)所组成的有意义的式子。

②接触过的地址表达式

接触过的地址表达式:

直接寻址方式、寄存器间接寻址方式、

变址方式、基址加变址方式

例如：

```
MOV  AX，BUF[BX+SI]
MOV  AL，BUF+2          ；源操作数为地址表达式
MOV  AL，BUF[BX]
MOV  WORD  PTR DS:[1000H], 3000H ；
                        ；目的操作数为地址表达式
```

③地址表达式的属性

地址表达式的结果是一偏移地址，因此具备段属性、偏移地址和类型。

问题：变量定义中，给变量置初始值的地址表达式可以含寄存器符号加方括号吗？

例如变量定义：　A　DW　[BX]　是否正确？

④地址表达式与数值表达式区别

- 地址表达式的结果：是一偏移地址，它具有段属性、偏移地址和类型，(一个表达式中一般只出现一个变量或标号)
- 数值表达式的结果：只有大小，无属性。
- 在特殊情况下(没有用到寄存器、不作为地址访问)，地址表达式的值也可能仅表示一个数值(没有属性)。

例:在变量/常量定义中给变量/常量赋值，或在OFFSET BUF + 2语句中)

```
NUM    DW    BUF1 - BUF2
```

(此时可以有多个变量或标号)

⑤特殊运算符——类型运算符PTR

- **格式**：类型 **PTR** 地址表达式

  类型可以是**BYTE、WORD、DWORD、FWORD、NEAR、FAR**

- **功能**：用来指明紧跟其后的地址表达式的类型属性，但保持它原来的段属性和偏移地址属性不变或者使它们临时兼有与原定义所不同的类型属性。

- 作用

例

类型运算符PTR的三个作用

作用1：使语句中类型模糊的操作数类型变得明确

```
ADD  BYTE  PTR [SI]，5
ADD  WORD  PTR [SI]，5
```

注意这两条语句的区别

作用2：临时改变某一操作数地址的类型，使得类型不一致的两地址变为一致。

例

作用3：PTR运算符还可以与EQU或等号“=”等伪指令连用，用来将同一存储区地址用不同类型的变量或标号来表示。

例

PTR作用2 例：

例 DATA1 DW 1122H，3344H

⋮

MOV AL，BYTE PTR DATA1;

将变量DATA1临时改为字节类型

问题1：将最后一条语句改为：

MOV EAX， DWORD PTR DATA1

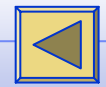
执行该语句后，(EAX)=?

比较PTR的作用与C语言的强制类型转换的不同点？

问题2：上述最后一条指令中，改变了DATA1的类型是否从此DATA1变为BYTE类型？

问题3：是否可以用该运算符改变寄存器的类型？

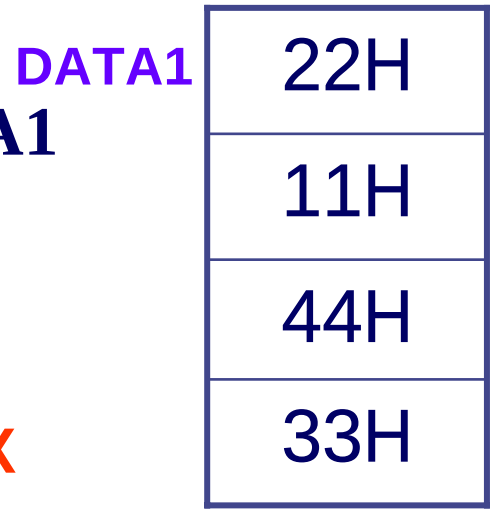
MOV EAX， DWORD PTR SI



PTR作用3 例：

例：分析下列程序的执行结果

```
DATA1 DW 1122H，3344H
DATA2 EQU BYTE PTR DATA1
  ┆
MOV AL，DATA2    ；22→AL
MOV BX，DATA1    ；1122→BX
```



用PTR算符建立了一个与变量DATA1有相同段首址和偏移地址的变量DATA2，但它的类型为BYTE

类型运算符PTR 例：

例：阅读下列程序段，指出其中的错误语句

注意

```
DATA  SEGMENT   USE16
NUM   DB   11H，22H，33H，44H
LEN   EQU  $ - NUM
DATA  END
      ┆
MOV  AX，NUM
MOV  EAX，DWORD PTR NUM
MOV  SI，OFFSET NUM
ADD  2[SI]，LEN
MOV  BYTE PTR 2[SI]，’A’
INC  [SI]
DEC  BYTE PTR［SI］
```

定义符号常量LEN，其值为以变量NUM为首址的数据存储区所占的字节数

类型不一致，出错

临时改NUM双字，

注意与C语言强制类型转换的区别

(NUM)＝44332211H→EAX

NUM的EA→SI

目的操作数类型不明确，出错

OPD＝[SI]＋2＝NUM＋2，

41H→NUM＋2

PTR指定OPD类型为字节

类型不明确，出错

OPD-[SI]＝NUM，由PTR指定为字节类型

使用PTR注意事项：

a. PTR临时赋予地址表达式的新类型只能在本语句中有效。

b. 不带方括号的寄存器符号不是地址表达式，不能用PTR改变寄存器的类型

⑤特殊运算符——属性分离算符

- 格式：属性分离算符 变量或标号
- 功能：属性分离算符可分离出变量、标号的段、偏移地址、类型的属性值。运算结果为数值常量。

a.取段址算符SEG

- 格式：SEG <变量或标号>
- 功能：分离出其后变量或标号的段首址。

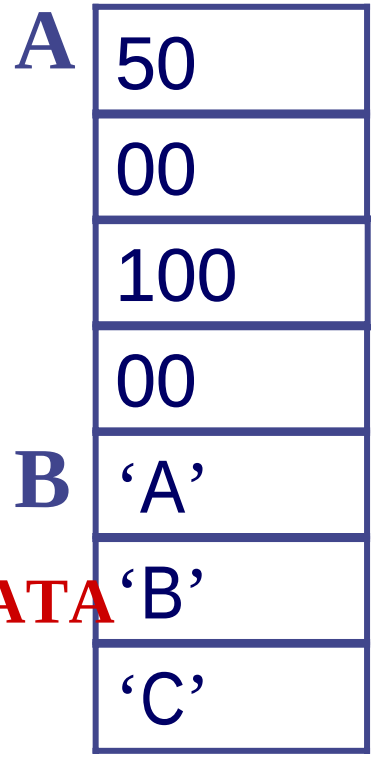
例

b.取偏移算符OFFSE

- 格式：OFFSET <变量>
- 功能：分离出其后变量或标号的偏移地址。

属性分离算符的例子

```
DATA  SEGMENT  USE16
A   DW  50，100，…
B    DB  ‘ABC…’
DATA  ENDS
   ┆
MOV  AX，SEG  B  ;<=>MOV  AX，DATA
MOV  DS，AX
MOV  BX，OFFSET A；A的EA→BX
MOV  DX，2[BX]    ；   100→DX
```



⑥使用地址表达式注意事项

(1)指令中的地址表达式不允许出现不带方括号的寄存器符号；

例：　MOV AX，SI+4　——错误语句，

　　MOV AX，[SI+4]　——正确语句

例

(2)在定义变量时，其后表达式不能带寄存器符号和方括号；

例：　A DW SI+4，[SI+4] ——错误

(3)数值表达式中如果有变量和标号，均是取其EA参加运算。

例

请指出下列程序段中的错误：

```
DW      1122H，3344H
  ⋮
MOV     SI，OFFSET  A
MOV     AX，A
MOV     BX，[SI]
MOV     DL，2[SI]
MOV     4[SI]，55H
```

改为：

```
MOV  BYTE  PTR 4[SI]，55H
MOV  WORD  PTR 4[SI]，55H
```

3.2 常用的机器指令语句

3.2.1 80X86指令集及其特点

1. 80X86指令集

- 8086　100条基本指令
- 80386　170条指令
- Pentium　300多条

2. 特点

- 原8086的16位操作指令都可扩展支持32位操作数；
- 原有16位存储器寻址的指令都可以使用32位的寻址方式；
- 在实方式和虚拟8086方式中段的大小只能为64KB，只有在保护方式下才使用32位段。

3.2.1 80X86指令集及其特点

3.分类

(1)数据传送指令

一般数据传送指令

堆栈操作指令：　PUSH、POP、PUSHF、POPF

标志传送命令：　SAHF、LAHF

I/O指令

地址传送指令

(2)算术运算指令

(3)位操作指令

(4)串操作指令

(5)程序控制指令

(6)处理机控制指令

标志的操作指令：STD、CLD；STI、CCI

- 怎样记住常用的指令?

  格式、功能、特殊规定，对标志寄存器的影响

3.2.1 80X86指令集及其特点

4.再次强调的问题

(1)大多数双操作数的指令，具有相同的语句格式和操作规定

- 格式：　[标号:] 操作符 OPD，OPS [；注释]
- 指令：数据传送指令；　算术运算指令
  部分位操作指令；　串操作指令
- 操作规定：
  ① 目的操作数与源操作数应有相同的类型。
  ② 目的操作数不能是立即操作数。
  ③ 操作结束后，运算结果送入目的地址中，源操作数并不改变。
  ④ 源操作数和目的操作数不能同时为存储器操作数。

3.2.180X86指令集及其特点

(2)某些单操作数指令也有相同的语句格式和操作规定，

- 格式：[标号：] 操作符 OPD [；注释]
- 指令：算术运算和位操作
- 操作规定：

① 操作对象为目的地址中的操作数，操作结束后，将结果送入目的地址。

② 操作数不能是立即操作数。

3.2.2 数据传送指令

华中科技大学

1. 一般数据传送指令
   (1) 传送指令
   (2) 数据交换指令
   (3) 查表转换指令XLAT
2. 地址传送指令
   (1) 传送偏移地址指令

一般数据传送指令 ——传送指令

A. 一般传送指令 MOV

- 格式：MOV OPD, OPS
- 功能：(OPS)→OPD (字或字节)
- 说明：

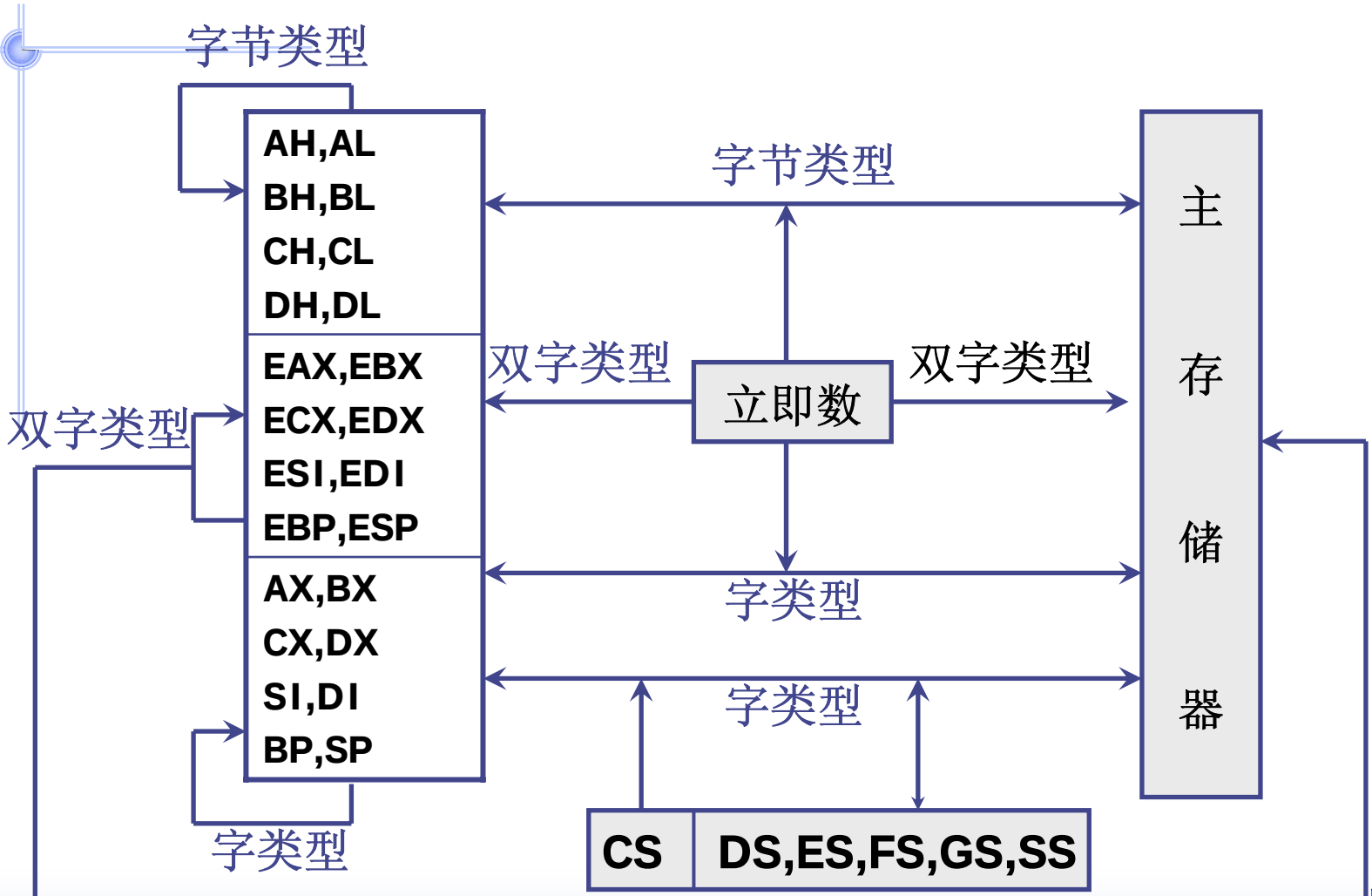
图

a. 不能实现存贮单元之间的直接数据传送，
OPS、OPD不能同时采用存贮器寻址方式。

例：将字变量BUF0中的内容传送至字变量BUF1中，只能用以下方式：

```
MOV  AX , BUF0
MOV  BUF1 , AX
```

一般传送指令 MOV



MOV指令所允许的数据传送路径及类型

一般数据传送指令 ——传送指令

b. 不能向CS送数据；IP不能在任何语句中出现。

例：“MOV CS , AX”、“MOV AX，IP”均为错误语句。

c. OPS和OPD必须类型一致。

例：MOV AX , CL 为错误语句。

d. 立即数不能直接传递至数据段或者附加数据段寄存器中；

问题：前面说过指令 MOV ECX, BL 是错误的，若确实想把BL寄存器的有符号数传送到ECX？

一般数据传送指令 ——传送指令

B. 有符号数传送指令

- 格式： MOVSX OPD，OPS (move with sign-extend)
- 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再送入目的地址。

MOVSX ECX， BL

例

C. 无符号数传送指令

- 格式：MOVZX OPD，OPS (move with zero-extend)
- 功能：将源操作数的高位全部补0，扩展成与目的操作数相同的数据类型再送入目的地址中。

有符号数传送指令

例 阅读下列程序段，指出运行结束后，EAX、EBX的值。

```
                ⋮
BYTE0       DB  0A8H
DWORD0      DD  11111111H
                ⋮
MOVSX  EAX，BYTE0
MOV    EBX，DWORD0
ADD    EBX，EAX
                ⋮
```

运行结束后，(EAX) = 0FFFFFFA8H，

(EBX) = 111110B9H

一般数据传送指令 ——数据交换指令

A. 一般数据交换指令

- 格式：**XCHG OPD，OPS (exchange)**
- 功能：**(OPD)↔(OPS)**，可作八位或十六位交换。

……

例：**XCHG AX，DI**

执行前：**(AX)=0001H (DI)=0FFFFH**

执行后：**(DI)=0001H (AX)=0FFFFH**

一般数据传送指令 ——查表转换指令

B. 查表转换指令XLAT (table look-up translation)

格式：XLAT OPS　　或者

XLAT　　不带操作数

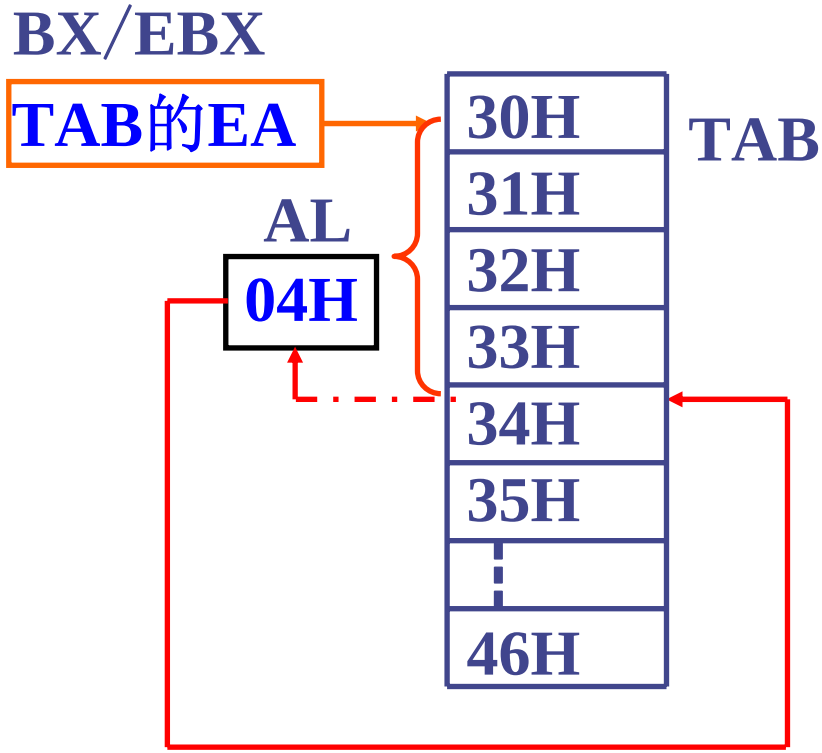
功能：([BX+AL])→ AL 或([EBX + AL])→AL

将(BX)或(EBX)为首址，(AL)为位移量的字节存贮单元中的数据→AL

例1　　例2

查表转换指令XLAT

例：将数值4转换成字符‘4’使用查表转换指令的思想



查表转换指令XLAT

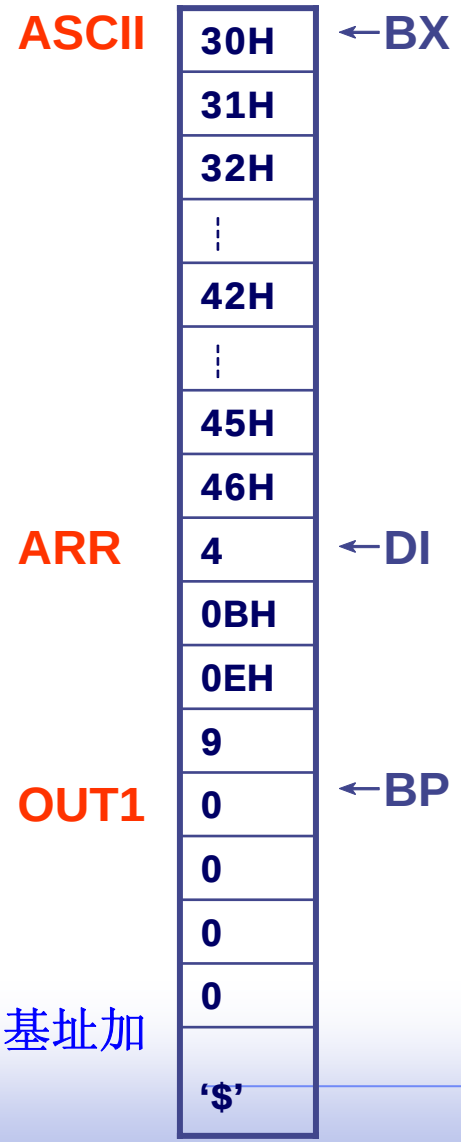
例：阅读程序

```
        ⋮
ASCII   DB '0123456789ABCDEF'
ARR     DB 4, 0BH, 0EH, 9
OUT1    DB 0, 0, 0, 0, '$'
        ⋮
        MOV BX, OFFSET ASCII
        MOV DI, OFFSET ARR
        MOV BP, OFFSET OUT1
        MOV CX, 4
NEXT:   MOV DL, [DI]
        MOV DH, 0
        MOV SI, DX
        MOV AL, [BX][SI]
        MOV DS:[BP], AL
        INC DI
        INC BP
        DEC CX
        JNE NEXT
        ⋮
```

修改：

```
MOV AL, [DI]
XLAT ASCII
```

XLAT指令简化了变址寻址和基址加变址寻址方式的使用。



地址传送指令 ——传送偏移地址指令

传送偏移地址指令

- 格式：LEA OPD，OPS (load effective address)
- 功能：按OPS的寻址方式计算EA，将EA送入指定的通用寄存器
- 注意：

a. OPD一定要是16位/32位的通用寄存器

b. OPS一定是一个存贮器地址，可是寄存器间接寻址、基址加变址、变址寻址、直接寻址。

c. 如果偏移地址为32位而OPD为16位寄存器，取低16位→OPD；

d. 如果偏移地址为16位而OPD为32位寄存器，高16位补0后→OPD

例1 例2

传送偏移地址指令

例：

MOV BX，OFFSET ARR → LEA BX，ARR

MOV SI，OFFSET PLUS → LEA SI，PLUS

MOV POIN，OFFSET MINUS → LEA POIN，MINUS ✗

LEA DI，4[SI] → MOV DI，OFFSET 4[SI] ✗

其中 ARR，PLUS，MINUS均为变量

传送偏移地址指令

例 .386

```
DATA  SEGMENT  USE16
 BUF   DB ‘ABCDEF’
NUM   DW  72，－5，100H
POIN  DW  0
DATA  ENDS
    ┆
MOV    ESI，    OFFSET  NUM      ；将NUM的EA即6→ESI
LEA    ESI，    NUM              ；与上一条语句等效，6→ESI
MOV    AX，     [ESI]            ；(［ESI］)＝72→AX
LEA    AX，     [ESI]            ；将ESI所指的存储单元的EA
                                 ；即6→AX
LEA    DI，     [ESI＋4]         ；取以NUM为首址的第三个字存
                                 ；储单元的EA即10→DI
LEA    POIN，   BUF              ；错误语句，因为OPD不是寄存器
MOV    POIN，   OFFSET BUF       ；将BUF的EA→POIN
MOV    EBX,     12345678H        ；将低16位9999H→DX
LEA    DX,      [EBX+4321H]      ；将12349999H→EAX
LEA    EAX，    [EBX+4321H]      ；将高16位补0后，
LEA    ECX，    [BX+4321H]       ；00009999H→ECX
    ┆
```

3.2.3算术运算指令

1. 加运算指令 ADD、INC
2. 减运算指令 SUB、DEC、NEG、CMP
3. 乘运算指令 IMUL、MUL
4. 符号扩展指令 CBW、CWD、CWDE、CDQ
5. 无符号乘指令 MUL
6. 除运算指令 IDIV、DIV

算术运算指令——加运算指令

加指令

- 语句格式：ADD OPD，OPS
- 功　能：　(OPD) + (OPS)→OPD

该指令对标志寄存器的标志位有影响。

例：ADD AX，-7FFFH

执行前：(AX) = 0FFFDH (即-3的补码

执 行：(AX) + [-7FFFH]补 = 0FFFDH + 8001H = 17FFEH→AX

执行后：(AX) = 7FFEH

两负数相加，结果为正，运算产生了溢出，结果是错误的，因而OF = 1。又由于从最高位向前产生了进位，CF = 1。

算术运算指令——减运算指令

减法指令

- 格式：**SUB OPD，OPS (subtract)**
- 功能：**(OPD) — (OPS) →OPD**

例：**SUB AX，5 ；(AX) — 5 →AX**

SUB AX，CX ；(AX) — (CX) →AX

减运算指令

华中科技大学

比较指令

- 格式：CMP OPD，OPS　　(compare)
- 功能：(OPD) — (OPS)

比较目的操作数与源操作数，然后根据比较的结果设置标志位，但该结果并不存入目的地址

例：CMP AX，-2
　　JGE L
　　MOV DX，AX
　　⋮

减运算指令

求补指令

- 格式：NEG OPD (two's complement negation)
- 功能：$(\overline{OPD})+1 \rightarrow OPD$ 即 0-(OPD) → OPD

例：NEG AX

执行前：(AX)=0FFFFH

执行：$(\overline{AX})+1=0FFFFH+1=0000H \rightarrow AX$

结果：(AX)=0000H

例：指出下面程序段执行后所完成的功能

```
B:   CMP  AX, 0
     JGE  EXIT
     NEG  AX
EXIT: ……
```

求(AX)的绝对值→AX

算术运算指令——乘运算指令

(1) 有符号数和无符号数的区别——复习

.数的表示范围不一样

.比较大小的标准不一样

例：对于8位16进制数，比较80H和0A8H的大小

.判断运算结果是否正确的标准也不一样，

例：对有符号数，加、减法运算结果只有OF＝0时才是正确的；对于无符号数，只有CF＝0时，运算结果才是正确的

.符号扩展不一样，有符号数的补码最高位向左延伸，得到的仍是该数的补码。

.常见的无符号数：操作数地址、循环次数、ASCII码。

乘运算指令

(2) 有符号乘指令——有三条

①双操作数的有符号乘指令

- 语句格式：IMUL OPD, OPS (signed integer multiply)
- 功能：（OPD）*（OPS）→OPD

OPD可为16/32的寄存器，OPS为同类型的寄存器、存储器操作数或立即数。

(2) 有符号乘指令——有三条

②三个操作数的有符号乘指令

- 语句格式：IMUL OPD，OPS，n
- 功能：（OPS）* n →OPD

其中，OPD可为16/32的寄存器，

OPS可为同类型的寄存器、存储器操作数，n为立即数

例：IMUL AX，BX，-10 ；(BX)*(-10)→AX

IMUL EAX，DWORD PTR [SI]，5 ；([SI])*5→EAX

IMUL BX，AX，3

(2) 有符号乘指令——有三条

③ 单操作数的有符号乘指令

- 语句格式：IMUL OPS
- 功　能：字节乘法：(AL)*(OPS)→AX

  字乘法：(AX)*(OPS)→DX，AX

  双字乘法：(EAX)*(OPS)→EDX、EAX

说明：

(a) 只需指定源操作数，另一个操作数是隐含的，被乘数和乘积都在规定的寄存器中。源操作数只能是存储器操作数或寄存器操作数而不能是立即数，乘法类型由OPS的类型决定。

(b) 如果乘积的高位(字节相乘指AH，字相乘指DX，双字相乘指EDX)不是低位的符号扩展，即在AH(或DX/EDX)中包含有乘积的有效位，则CF＝1、OF＝1、否则，CF＝0，OF＝0。

例

③ 单操作数的有符号乘指令

例：写出实现500H * 60H→SI，DI的程序段。

```
MOV  AX，500H
MOV  BX，60H
IMUL  BX
```

```
MOV EAX, 500H
IMUL EBX, EAX, 60H
```

字乘法结果高位在DX中，低位在AX中，(DI)=0001H，(SI)=0E000H

问题：若不要求将结果送SI，DI，还有其他写法吗？

例：写出实现500H *(-2) 的程序段。

```
MOV  AX，500H
MOV  BX，-2
IMUL  BX
```

```
MOV AX, 500H
IMUL -2
```

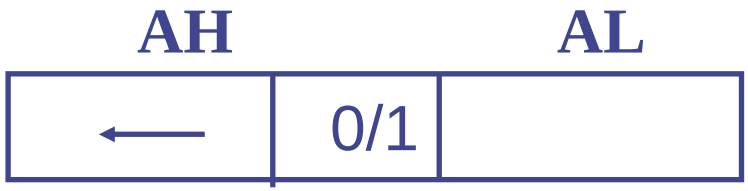
结果：(AX)=0F600H　(DX)=0FFFFH

算术运算指令——符号扩展指令

(1) 将字节转换成字指令

语句格式：CBW　　(convert byte to word)

功能：将AL中的符号扩展至AH中，操作数隐含且固定



例：MOV BX，04A7BH

MOV AL，A3H

CBW　　；将字节扩展成字

ADD BX，AX

执行后：(AX)=0FFA3H，(BX)=4A1EH

问题：若想扩展BL至BH该怎样写出指令？

符号扩展指令

(2) 将字转换成双字指令

- 语句格式：CWD (convert word to double word)
- 功 能： 将AX中的符号扩展至DX中，由 DX，AX 组成双字

(3) CWDE 将AX中的有符号数扩展为32位数→EAX

(4) CDQ 将EAX中的有符号数扩展为64位数→EDX、EAX

例：
```
MOV DX，0
MOV AX，0FFA3H
CWD
```
执行后：(AX)=0FFA3H (DX)=0FFFFH

注意：上述指令的操作数是隐含且固定的。

算术运算指令——无符号乘指令

无符号乘指令

- 语句格式：MUL OPS
- 功　能：字节乘法：(AL)*(OPS)→AX
  字乘法：(AX)*(OPS)→DX，AX
  双字乘法：(EAX)*(OPS)→EDX、EAX

说明：

(1)与有符号乘法指令之间的区别：参与运算的操作数和运算后的结果均为无符号数。

(2)如果乘积的高位不为0，即在AH(或DX/EDX)中包含有乘积的有效位，则CF＝1、OF＝1；否则，CF＝0，OF＝0。

算术运算指令——除运算指令

(1) 无符号除指令

- 语句格式：DIV OPS (unsigned divide)
- 功 能：字节除法：(AX)/(OPS)→AL(商)、AH(余数)
  字除法：(DX、AX)/(OPS)→AX(商)、DX(余数)
  双字除法：(EDX、EAX)/(OPS)→EAX(商)、EDX(余数)

说明：

(a) 除法类型由OPS的类型决定。OPS不能是立即操作数，且指令执行后，(OPS)不变。

(b) 如果除数为0或运算结果溢出，则会产生溢出中断，立即中止程序的运行。但系统未定义除法指令影响条件标志位。

注意：除法指令的被除数是隐含的。

除运算指令

(2) 有符号除指令

- 语句格式： IDIV OPS (signed integer divide)
- 功 能：字节除法:(AX)/(OPS)→AL(商)、AH(余数)
  字除法：(DX，AX)/(OPS)→AX(商)，DX(余数)
  双字除法:(EDX、EAX)/(OPS)→EAX(商)、EDX(余数)

说明：

(a)、(b)两点与DIV语句相同；

(c) 相除后，商的符号与数学上规定相同，
余数与被除数同号。

例1 例2

有符号除指令

例：写出计算4001H÷4的程序段。

```
MOV AX，4001H
CWD              ——符号位扩展到DX，
MOV CX，4
IDIV CX          —— (DX，AX)/(CX)
```

结果：(AX)=1000H , (DX)=1

问题：该题能否用字节除?

因为商是4位十六进制数，一个字节放不下。

如果将被除数改为—4001H，程序段为：

```
MOV  AX，—4001H
CWD
MOV  CX，4
IDIV CX
```

运算的结果为：

(AX) = 0F000H，(DX) = 0FFFFH

假如被除数为-4001H，除数为-4，相除后的余数也为0FFFFH。

有符号除指令 例题

例:阅读下列程序段，指出程序的功能。

```
.386
DATA  SEGMENT  USE16
X    DW  25
Y    DW  20
Z    DW  –74
V    DW  50
F    DD   2 DUP(0);
DATA ENDS
        ┆
CODE   SEGMENT  USE16
ASSUME  DS:DATA,  SS:STACK, CS:CODE
START: MOV  AX,   DATA
          MOV  DS,   AX
          MOV  AX,   X
          CWDE
          MOV  EBX, EAX
          MOV  AX,   Y
          CWDE
          IMUL   EBX, EAX
          MOV   AX,    Z
          CWDE
          ADD    EBX,  EAX
          SUB    EBX,  540
          MOV   AX,   V
          CWDE
          SUB    EAX,  EBX
          XCHG  EBX,   EAX
          MOV   AX,    X
          CWDE
          XCHG  EBX,   EAX
          CDQ
          IDIV    EBX
          MOV   F,      EAX
          MOV   F+4,    EDX
          ┆
```

计算F=(V-(X*Y+Z-540))/X

有符号除指令 例题(续)

```
DATA  SEGMENT  USE16
X    DW  25
Y    DW  20
Z    DW  –74
V    DW  50
F    DW  2 DUP(0);
DATA  ENDS
      ┆
MOV     BX,    X
IMUL    BX,    Y
ADD     BX,    Z
SUB     BX,    540
MOV     AX,   V
SUB     AX,   BX
CWD
IDIV    X
MOV     F,     AX
MOV     F+2,   DX
      ┆
```

3.2.3 位操作指令

1. 逻辑运算指令
   (1) 逻辑乘指令AND (and)
   (2) 测试指令TEST
   (3) 逻辑加指令OR
   (4) 按位加指令XOR——异或
   (5) 位操作指令的特点
2. 移位指令
   (1) 算术、逻辑移位指令
   (2) 循环移位指令

1. 逻辑运算指令——逻辑乘指令AND

逻辑乘指令

- 格式：**AND OPD, OPS (and)**
- 功能：**(OPD) ∧ (OPS) —> OPD**

例 **AND AX, 0FH**

执行前：(AX) = 0FBBAH

执行后：(AX) = 0AH

- 用途　该指令主要用来将目的操作数中清除与源操作数置0的对应位，因此可用来将存贮器或寄存器中不需要的部分去掉。将需要部分分离出来。

例：将DX中的11~8位分离出来使用以下指令：

AND DX，0F00H (即将源操作数中的11~8位置1)

1. 逻辑运算指令——测试指令TEST

华中科技大学

测试 指令

- 格式：TEST OPD，OPS (test)
- 功能：(OPD)∧(OPS) ——根据结果设置标志位

检测与源操作数中为1的位相对应的目的操作数中的那几位是否为0(或为1)，根据测试结果置OF、CF、SF、ZF位，后面往往跟着转移指令，根据测试结果确定转移方向。

例

测试指令TEST

例：要测试AX中第12位是否为0，为0转L，则要使用如下指令：

0001 0000 0000 0000
15 12 0

```
TEST  AX, 1000H
JZ    L
```

如果要同时测试第15位和第7位是否同时为0，为0转L

1000 0000 1000 0000
15 7

```
TEST  AX,  8080H
  JZ     L
```

1. 逻辑运算指令——逻辑加指令OR

逻辑加指令OR

- 格式：**OR OPD, OPS (or)**
- 功能：**(OPD) ∨ (OPS) —> OPD**

例 **OR AX, 55H**

执行前：**(AX) = 0AAAAH**

执行后：**(AX) = 0AAFFH**

该语句主要用于：在目的操作数中置位与源操作数为1的对应位，其余位不变。

三条逻辑指令用法的选择

①如果要将目的操作数中某些位清0，用AND

②如果要将目的操作数中某些位置1，用OR

③用来测试目的操作数中某一位或某几位是否为0或1，而结果不变，用TEST

④操作数自身相或、相与结果不变。

1. 逻辑运算指令——按位加指令XOR

按位加指令——异或

- 格式：**XOR OPD, OPS (exclusive or)**
- 功能：**(OPD) ∀ (OPS) → OPD**

运算法则：**0∀1=1，1∀0=1，1∀1=0，0∀0=0**

例：**XOR AX, 0AAAAH**

执行前：**(AX)=0FFFFH**

1111	1111	1111	1111
1010	1010	1010	1010
0101	0101	0101	0101

结果：**(AX)=5555H**

作用：主要用来将目的操作数中与源操作数置1的对应位取反。操作数自身异或，结果为0，CF=0

位操作指令的特点

⑴自身相或相与结果不变；

自身按位加结果为0，“XOR AX，AX”之后(AX) = 0；

⑵置位。如果(AL)=9，将(AL)+30H→AL，

用“ADD AL，30H”

或“OR AL，30H”；

⑶测试。例如，测试SI中的第3、7、11、15为是否同时为0，为0转ERR：

```
TEST  SI，8888H
JE    ERR
  ⋮
```

例

1011 0111 1000 0111

∧)

1000 1000 10001000

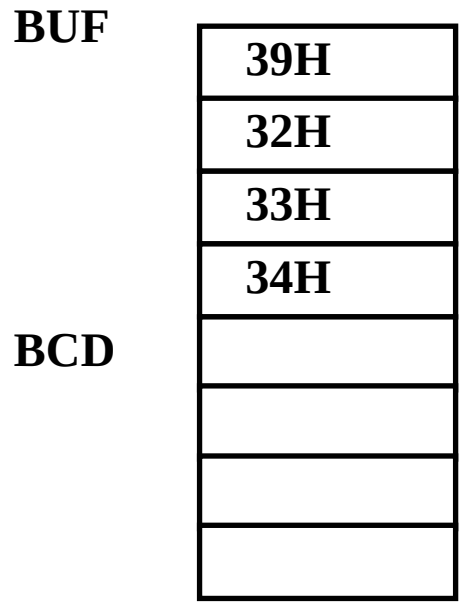
1000 0000 1000 0000

15 11 7 3 0

位操作指令的特点

例：阅读下列程序指出程序所完成的功能。

```
.386
DATA  SEGMENT  USE16
BUF  DB  '9234'
BCD  DB  4 DUP(0)
DATA ENDS
        ┆
START:MOV  AX,  DATA
        MOV  DS,  AX
        MOV  CX，4
        LEA  SI，BUF    ；取BUF缓冲区→SI
        LEA  DI，BCD+3  ；取BCD缓冲区→DI
   L:   MOV  AL，[SI]   ；取BUF中一字符→AL
        AND  AL，0FH ；清除AL高位，使其ASSCII码变为数字→AL
```



位操作指令的特点

```
        MOV  [DI]，AL    ；(AL)→BCD区
        INC   SI
        DEC  DI
        DEC  CX
        JNE   L
        MOV  AH, 4CH
        INT   21H
CODE  ENDS
        END START
```

功能：将BUF中四位数字字符的ASSCII码转换成非压缩的BCD码→BCD区。

2. 移位指令

(1)算术移位指令

(2)逻辑移位指令

(3)循环移位指令

(4)双精度移位指令

X代表	含义
H	逻辑移位(logical)
A	算术移位(arithmetic)
O	循环移位(Rotate)
C	带进位的循环移位 carray

前三类指令有统一的语句格式：

操作符 OPD，n

算术逻辑移位操作符： SXL， SXR (X=H, A)

循环移位操作符：RXL，RXR (X=O, C)

移位指令的特点

- 将目的操作数中的所有位按操作符所规定的方式移动n所规定的次数(0～31)，然后将结果送入目的地址中。
- 目的操作数是由各种寻址方式所提供的8位、16位或32位的寄存器数据或存储器数据。
- 不管哪种方式的移位都会将所移出的最后一位放入CF位

2. 移位指令——算术、逻辑移位指令

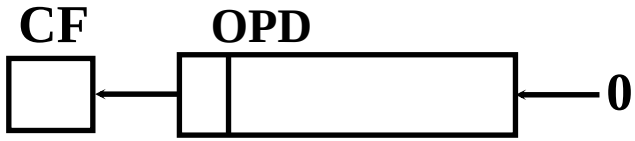
①算术左移和逻辑左移指令SAL/SHL

- 格式：SHL OPD, n 或 SAL OPD, n

  (shift logical left/shift arithmetic left)

- 功能：

将OPD的内容向左移动n指定的位数，低位补入相应个数的0。CF的内容为最后移入位的值，移动方式为：

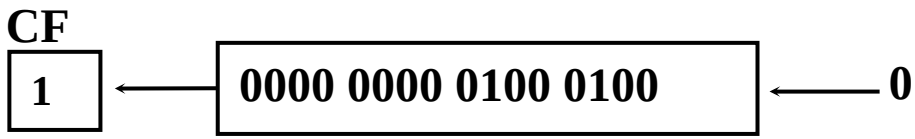


例

①算术左移和逻辑左移指令SAL/SHL

例：SAL AX, 1

执行前：(AX) = 0044H，CF = 1



结果：CF = 0，(AX) = 0088H

注意：

当一个数乘2的N次方时，可以用算术左移或逻辑左移N位的方法实现，比用乘法指令效率高。但若发生溢出则可能得不到正确的结果。

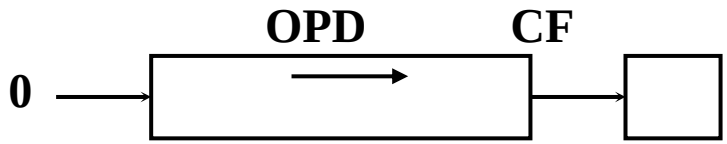
问题：设(SI)=00FFH，现需要计算(SI)*8->SI，

请写出实现语句。

2. 移位指令——算术、逻辑移位指令

②逻辑右移指令SHR

- 格式：SHR OPD，n (shift logical right)
- 功能：将(OPD)向右移动n规定的次数，最高位补入相应个数的0，CF的内容为最后移入位的值。



- SHR指令右移n位，实现无符号数除2^n运算
- 将一个字(或字节/双字)中的某一位(或几位)移动到指定的位置，从而达到分离出这些位的目的.

2. 移位指令——算术、逻辑移位指令

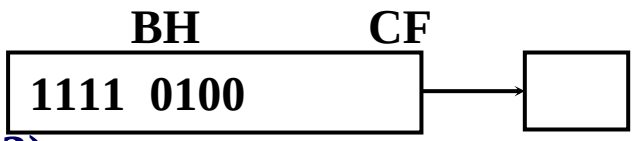
③算术右移指令SAR

- 格式：SAR OPD, n (shift arithmetic right)
- 功能：将OPD中的操作数移动n所指定的次数。且最高位保 持不变。CF的内容为最后移入位的值。

在移位过程中符号位保持不变！！！

问题：指令SAR BH，2

执行前(BH)=0F4H (—12)，CF=1，执行该指令后(BH)=？(CF)=？



(BH)=0FDH 即(—3)

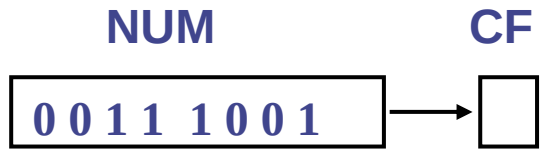
例

利用算术右移指令可以方便地实现对有符号数除2的N次方的运算。

算术右移指令SAR

例：阅读下列程序指出程序所完成的功能

```
        ⋮
NUM  DB  00111001B
BUF  DB  '(NUM) ='
BUF0 DB  0，0，'H'，0AH，0DH，'$'
        ⋮
MOV  AL, NUM
SHR  AL, 4
OR   AL, 30H
MOV  BUF0, AL
MOV  AL, NUM
AND  AL, 0FH
OR   AL, 30H
MOV  BUF0+1, AL
LEA  DX, BUF
MOV  AH, 9
INT  21H
        ⋮
```



取NUM的高四位转换为ASCII码→BUF0区

取NUM的低四位转换为ASCII码→BUF0区

输出"NUM=39H"并回车换行

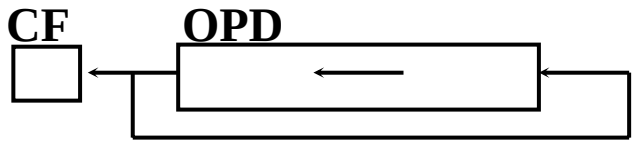
讨论：可以将上述逻辑移位指令改为算术移位指令吗？

若将NUM的值改为11001001呢？

2. 移位指令——循环移位指令

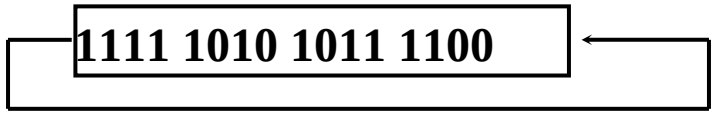
①循环左移指令

- 格式：ROL OPD, n (rotate left)
- 功能：将目的操作数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所规定的次数。CF的内容为最后移入位的值。



例：ROL DX, 4

执行前：(DX)=0FABCH

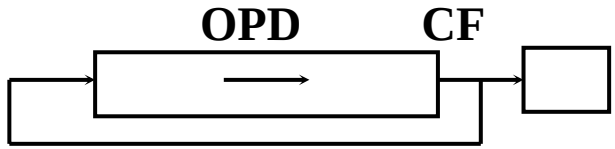


结果：(DX)=0ABCFH CF=1

2. 移位指令——循环移位指令

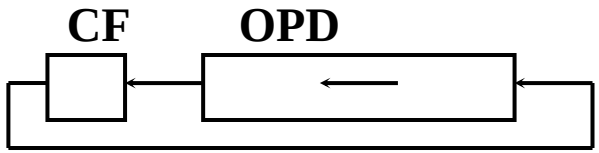
②循环右移指令

- 格式：ROR OPD，n (rotate right)
- 功能：该指令的移动方式完全同ROL，只是向右移动。



③带进位位的循环左移指令

- 格式：RCL OPD，n (rotate left through carry)
- 功能：将目的操作数连同CF标志一起向左循环移动所规定的次数。



④带进位的循环右移指令

- 格式：RCR OPD，n (rotate right through carry)
- 功能：该指令的移动方式完全同RCL，只是向右移动。

3.3 伪指令语句

华中科技大学

1. 伪指令的基本概念
2. 处理器选择伪指令
3. 段定义伪指令
   - (1)段定义伪指令
   - (2)假定伪指令
   - (3)置汇编地址计数器伪指令
4. 源程序结束伪指令

1. 伪指令的基本概念

- **定义**：汇编源程序中控制汇编程序应如何工作的命令是**伪指令**，或称**汇编控制命令**。
- **工作原理**：
  - 只为汇编程序所识别
  - 每一条汇编控制命令都对应着一段处理程序，
  - 汇编程序每遇到汇编控制命令，即转入对应的处理程序执行，执行完该处理程序，也就实现了这条汇编控制命令的功能。
- **结果**：
  - 可以申请分配一部分存贮空间用作数据区和堆栈
  - 没有对应的机器代码；
  - 在将源程序翻译成目标程序后，伪指令就不存在了。

1. 伪指令的基本概念

华中科技大学

问题

- 伪指令分类：
  - 处理器选择伪指令
  - 数据定义伪指令
  - 符号定义伪指令
  - 段定义伪指令
  - 过程定义伪指令
  - 程序模块的定义与通讯伪指令
  - 宏定义伪指令
  - 条件汇编伪指令
  - 格式控制、列表控制及其它功能伪指令

伪指令问题

- 问题：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，伪指令控制汇编程序工作。

(2)格式不同，机器指令标号后面带冒号，而伪指令的名字后面没有。

(3)被执行时CPU所处状态不同，用户程序在运行时执行机器指令，汇编程序运行时，执行伪指令。

(4)机器指令是用硬件线路来实现其功能的，它有目的代码。而伪指令是用来控制汇编程序操作的，是用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。

2. 处理器选择伪指令

- 格式： .X86[P], .MMX
- 作用：

.在源程序中，告诉汇编程序选择何种CPU所支持的指令系统；

.一般放在程序的开始处，表示后面的段使用该处理器所支持的指令系统；

.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；

.缺省情况下是.8086

不同版本的宏汇编程序对指令系统的支持不同：

.MASM 5.X只支持到.386;

.MASM 6.11支持到.586

.MASM 6.12支持到.686

3. 段定义伪指令——段定义伪指令

- **格式：**

  段名 SEGMENT [使用类型][定位方式][组合方式]['类别']

  ⋮

  段名 ENDS

- **功能：**

  定义一个以SEGMENT伪指令开始、ENDS伪指令结束的、给定段名的段。

  其中，段名为该段的名字，用来指出汇编程序为该段分配存贮区的起始位置。

段定义应注意的问题

- 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；
- 段的定义还可以嵌套，但不能交叉；
- “使用类型”只有对使用.386及以上处理器选择伪指令的段才起作用。 —— USE16 USE32
- 在实方式和虚拟8086方式中段的大小只能为64KB。

3. 段定义伪指令——假定伪指令

- **格式：** ASSUME 段寄存器：段名[，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。

例

假定伪指令 例题

例： .386

```
DATA1  SEGMENT  USE16
T1          DW    -50H
T2          DD    F
DATA1       ENDS
DATA2       SEGMENT  USE16
BUF         DB    'ABCDEF'
F           DW     70H
DATA2        ENDS
STACK       SEGMENT  USE16  STACK
            DB  200  DUP(0)
STACK ENDS
```

假定伪指令 例题(续)

```
CODE   SEGMENT  USE16
ASSUME  DS:DATA1,  CS:CODE,  SS:STACK
START:MOV        AX,    DATA1
        MOVDS,     AX
        MOVAX,     T1
       ASSUME  DS:DATA2
       MOV   BL，BUF+2

       LDS    SI，T2
       MOV   BL, BUF+2

CODE  ENDS
END    START
```

注意

若未用ASSUME，则必须加DS，否则汇编程序报错；
用了ASSUME指出对应关系则该指令不带DS不报错，但是结果不对，因没有正确设置DS

该指令在未用ASSUME时，必须加DS否则汇编程序报错；若用了ASSUME则该指令在设置DS后，结果正确

假定伪指令 注意

注意：

- 在代码段的开始，就要用ASSUME语句建立CS、SS与代码段、堆栈段的对应关系，否则就会出错
- ASSUME语句不可能将段首址置入对应的段寄存器中，这一工作要到目标程序最后投入运行时CS和SS的内容将由系统自动设置，不用用户程序处理

假定伪指令 注意(接上页)

- 对于数据段和附加数据段，若用ASSUME语句建立它们与DS、ES的关系，则其后语句如需访问这些段内的变量，均可直接使用段内寻址，而不必带跨段前缀；——ASSUME语句之后用户设置DS、ES段
- 对于数据段和附加数据段，若不用ASSUME语句建立它们与DS、ES的对应关系，则其后语句如需访问这些段内的变量，都必须带跨段前缀才可使用段内寻址

3. 段定义伪指令—置汇编地址计数器伪指令

华中科技大学

① 汇编地址计数器 $

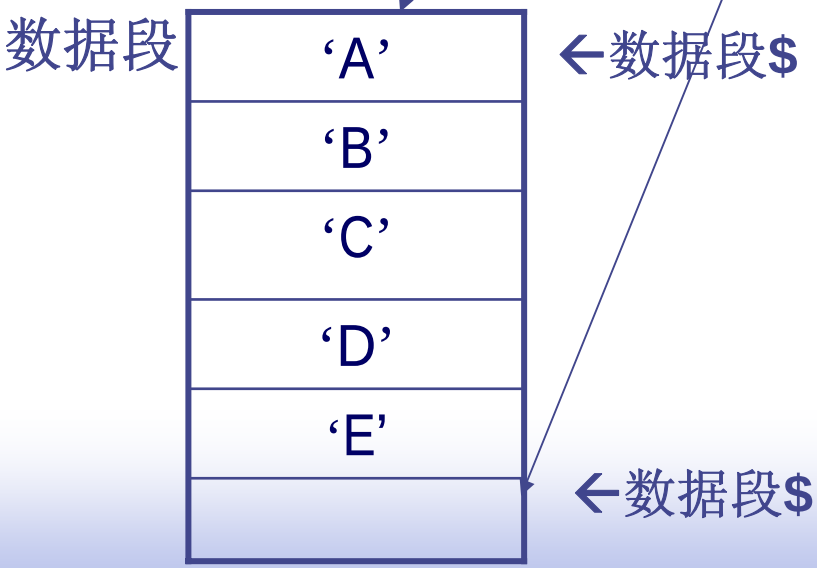
② 设置汇编地址计数器的值

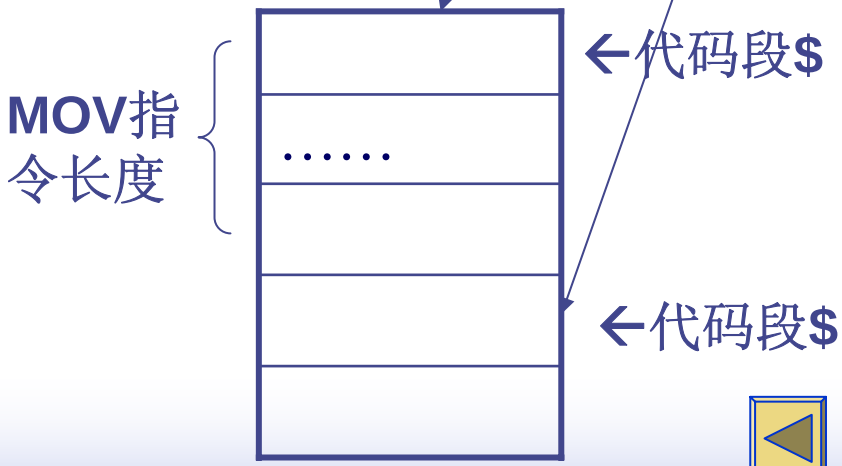
① 汇编地址计数器 $

```
DATA  SEGMENT  USE16
A  DB  'ABCDE'
B  DW  12, 34
C  DB  $-B
……
DATA  ENDS
```

```
CODE  SEGMENT  USE16
MOV   AX,  1234H
ADD    BX,  AX
……
CODE  ENDS
```

例





汇编地址计数器 $ 例题

例：

```
DATA  SEGMENT  USE16
A   DB  'ABCD'
B   EQU  $-A
C   DB   $-A DUP(0)
D   DB   $-C DUP(0)
E   DB   'MOV  DX, BUF'
         'ADD   AL, BUF+1'
         'MOV   CX, AX'
F   DB   'MOV'
DATA    ENDS
```

B=当前汇编地址计数器的值-A的偏移地址

问题:若需要统计E中'MOV'出现的次数，怎样计算E中字符串的长度，用$计算

注意：该例子中，两个$代表的值是不同的

② 设置汇编地址计数器的值

●格式：ORG <数值表达式>

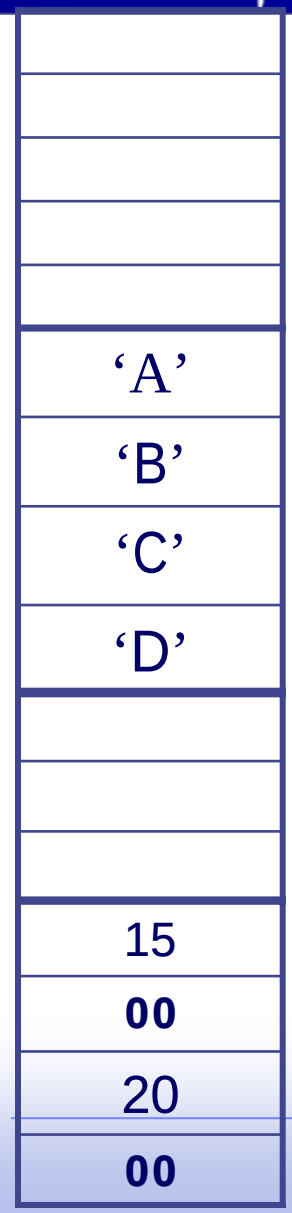
表达式的值为0～65535(16位段)

0~ 4G (32位段)

●功能：将$的值置成数值表达式的值

例：

```
DATA  SEGMENT  USE16
 ORG     5       空了五个字节
 A  DB   'ABCD'  A的EA为5
 B  EQU  $—A     设B的值为4
 ORG     $+3     空三个字节
 C  DW  15，20，……C的EA为12
 DATA   ENDS
```



4. 源程序结束伪指令

格式：**END** [表达式]

功能：该语句为汇编源程序的最后一个语句，用以标志源程序的结束。即告诉汇编程序翻译到此为止。

表达式指出了该程序运行时，第一条被执行指令的地址。如果不带表达式，说明该程序是一个子模块，不能单独执行，往往供另外的程序调用。

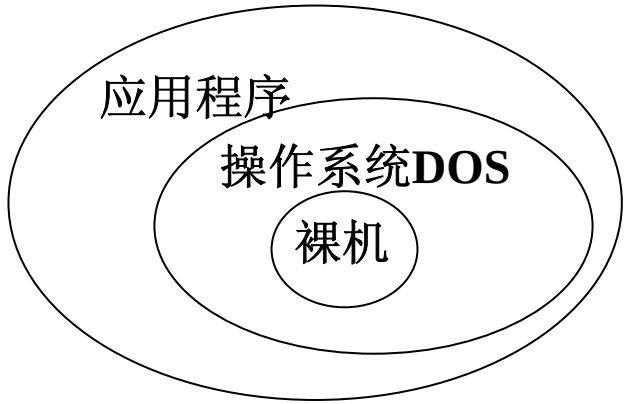
注意：不可将END语句错误地安排在程序中间

3.4 常用的系统功能调用

1. 什么是系统功能调用？
2. DOS的系统功能调用分类
3. DOS系统功能调用的一般过程
4. 常用的输入/输出系统功能调用

1. 什么是系统功能调用？

1. 什么是系统功能调用？



DOS的系统功能调用：系统将对计算机外部设备(键盘输入、屏幕输出)的控制过程编写成程序，事先存放在系统盘上，用户在需要时只要按规定的格式设置好参数，直接调用。

2. DOS的系统功能调用分类

(1)设备管理

(2)文件管理

(3)目录管理

(4)其他功能调用

3.DOS系统功能调用的一般过程

(1) 置入口参数

(2) 子程序编号(功能号)→AH

(3) INT 21H

(4) 成功调用返回，CF=0；否则CF=1且错误码→AX

注意：使用时保存好AL寄存器的内容

4.常用的输入/输出系统功能调用

(1)键盘输入(1号调用)

- 格式：MOV AH，1
  INT 21H
- 功能：等待从键盘输入一个字符→AL，同时将此字符在屏幕上显示出来。

(2)显示输出(2号调用)

- 格式：MOV DL，待显示字符的ASCII码
  MOV AH，2
  INT 21H
- 功能：将DL中的字符在屏幕上显示出来。

例如：
```
MOV DL，0AH
MOV AH，2
INT 21H
```
} 输出换行符

问题1：输出换行符是什么效果？

问题2：输出回车符是什么效果？

4.常用的输入/输出系统功能调用

(3) 输出字符串（9号调用）

- 格式：LEA DX，字符串首址偏移地址

  MOV AH，9

  INT 21H

- 功能:将当前数据段中指定的(DS:DX)字符串输出
  (该字符串必须以'$'为结束符，且字符'$'不输出)

例

(3) 输出字符串（9号调用）

例：
```
DATA  SEGMENT USE16
BUF   DB ‘A字符的ASCII码是41H’，0AH，0DH
      DB ‘B字符的ASCII码是42H’，0AH，0DH，‘$’
DATA  ENDS
      ⋮
LEA   DX，  BUF  (或者MOV  DX，OFFSET BUF)
MOV   AH，9
INT   21H
      ⋮
```

则输出：A字符的ASCII码是41H，光标回车换行，再显示：
B字符的ASCII码是42H，光标回车换行。

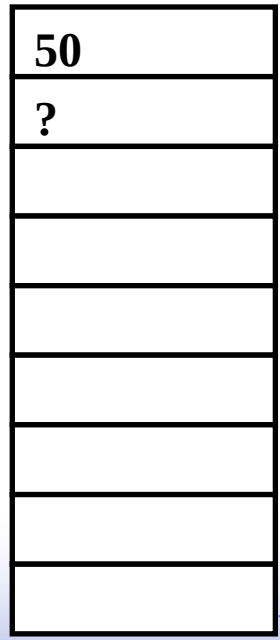
问题：如果输出的字符串中，没有‘$’会产生什么结果？
显示的字符串中有‘$’该怎样处理？

4.常用的输入/输出系统功能调用

(4)字符串输入（0AH号调用）

- 格式：

```
DATA  SEGMENT  USE16
BUF     DB  50          ；缓冲区大小
        DB  ?           ；记录填入实际的字符个数
        DB  50 DUP(0)   ；输入缓冲区，最多可装49个字符
DATA  ENDS
        ┇
        LEA   DX，BUF50?
        MOV  AH，10
INT    21H
```



(4)字符串输入（0AH号调用）

- **功能：** 从键盘接收一个以回车为结束的字符串到当前数据段输入缓冲区中，并在屏幕上回显示，同时将输入字符实际个数n填入缓冲区第二字节中。如n>49则响铃，多余字符被丢掉。

- **注意：** 一串字符的最后必须要输入回车作结束，但该结束只表示输入工作的结束，回车符ASCII码送入到缓冲区内但不计入字符个数，且光标只回到本行行首。

若输入:ABCDEF↙

请问:最后一个字符的存放位置的偏移地址？怎样计算？怎样表示？

问题　　例

(4) 字符串输入——问题

1.若缓冲区定义为：

```
BUF   DB   80
DB   ?
DB   200 DUP（0）
```

请问实际的缓冲区大小？

2. 若缓冲区定义为：

```
BUF   DB   80
DB   ?
DB   30 DUP（0）
```

实际缓冲区的大小又怎样？

(4) 字符串输入 例题

例：阅读下列程序，并指明它所完成的功能

```
DATA  SEGMENT
BUF   DB  50
 DB  0
      DB  50 DUP（0）
CRLF  DB  0DH，0AH，’$’
DATA  ENDS
        ⋮
START：MOV  AX，DATA
       MOV  DS，AX
       LEA   DX，BUF
       MOV  AH，10
       INT   21H
```

输入缓冲区

输入字符串

(4)字符串输入 例题（续）

```
    LEA  DX，CRLF
    MOV  AH，9
    INT  21H
    MOV  BX，BUF+1
    MOV  BYTE PTR BUF+2[BX]，‘$’
    LEA  DX，BUF+2
    MOV  AH，9
    INT  21H
    MOV  AX，4C00H
    INT  21H
CODE  ENDS
    END  START
```

输出回车换行

在下一行输出前面输入的字符串

3.5 本章小结

本章主要掌握的内容：

1. 数值表达式和地址表达式
2. 机器指令语句
3. 伪指令语句
4. DOS功能调用
5. 在IBM-PC机上建立、调试、运行汇编源程序的方法。

1.数值表达式和地址表达式

a. 存贮器寻址方式均属地址表达式形式(直接、间接、变址、基址加变址)

b. 在地址表达式不允许出现不带方括号的寄存器符号。

c. 地址表达式中的标号和变量均是取其偏移地址参加运算。

算符： PTR SEG OFFSET

2. 机器指令语句

(1)数据传送指令：

- 一般数据传送指令： MOV、XCHG、XLAT、MOVSX、MOVZX、BSWAP、XADD
- 堆栈操作指令： PUSH、POP、PUSHF、POPF
- 标志传送命令： SAHF、LAHF
- 地址传送指令： LEA、LDS、LES

2. 机器指令语句

(2)算术运算指令：

- 加指令：　ADD、INC
- 减指令：　DEC、SUB、CMP
- 乘除法指令：MUL、IMUL、CBW、CWD、CWDE、CDQ、DIV、IDIV

(3)位操作指令：

- 逻辑运算指令：NOT、AND、TEST、OR、XOR
- 移位指令：　SHL/SAL、SHR、SAR、ROL、ROR、RCL、RCR

3. 伪指令语句

(1) 与机器指令语句的区别

(2) 数据定义伪指令： DB、DW

(3) 符号定义伪指令： EQU、 =

(4) 段定义伪指令：SEGMENT、ENDS、ASSUME

(5) 源程序结束伪指令：END

(6) 汇编地址计数器： $ 和ORG <表达式>

4. DOS功能调用

华中科技大学

1号、2号、9号、10号DOS功能调的格式、使用条件。

作业

教材P94 3.1, 3.3

P95 3.5, 3.6

P95 3.7, 3.8