



汇编语言程序设计

第三章 寻址方式与指令系统 II

张华平 副教授 博士

Email: kevinzhang@bit.edu.cn

Website: <http://www.nlpir.org/>

@ICTCLAS张华平博士

大数据搜索挖掘实验室 (wSMS@BIT)

2015-10





指令

指令告诉CPU执行什么样的操作及操作数从哪里得到。指令可以用大写、小写或大小写字母混合的方式书写。

☞ 指令格式和功能

☞ 指令所影响的标志位

☞ 指令执行周期 *

☞ 指令机器长度 *

汇编语言是面向机器的，指令和机器码基本上是一一对应的，所以它们的实现取决于硬件。





指令系统

- 数据传送指令
- 算术运算指令[2进制/10进制]
- 逻辑指令
- 程序控制指令
- 处理机控制指令
- 串操作指令
- 条件字节设置指令



3.3 数据传送指令

数据传送指令可以实现数据、地址、标志的传送。除了目标地址为标志寄存器的传送指令外，本组的其它指令不影响标志。

1. 通用数据传送指令；
2. 堆栈操作指令；
3. 输入输出指令；
4. 查表转换指令；
5. 地址传送指令；
6. 标志传送指令；





1. 通用数据传送指令

(1). 传送指令 **MOV**

格式: **MOV** DST, SRC

功能: SRC (源) → DST (目标)

说明: **MOV**指令可以实现一个字节、一个字、一个双字的数据传送,

注意: 源操作数和目标操作数的数据类型匹配问题, 即应同为字节、字或双字型数据。

MOV 指令可实现的数据传送方向如下图所示。



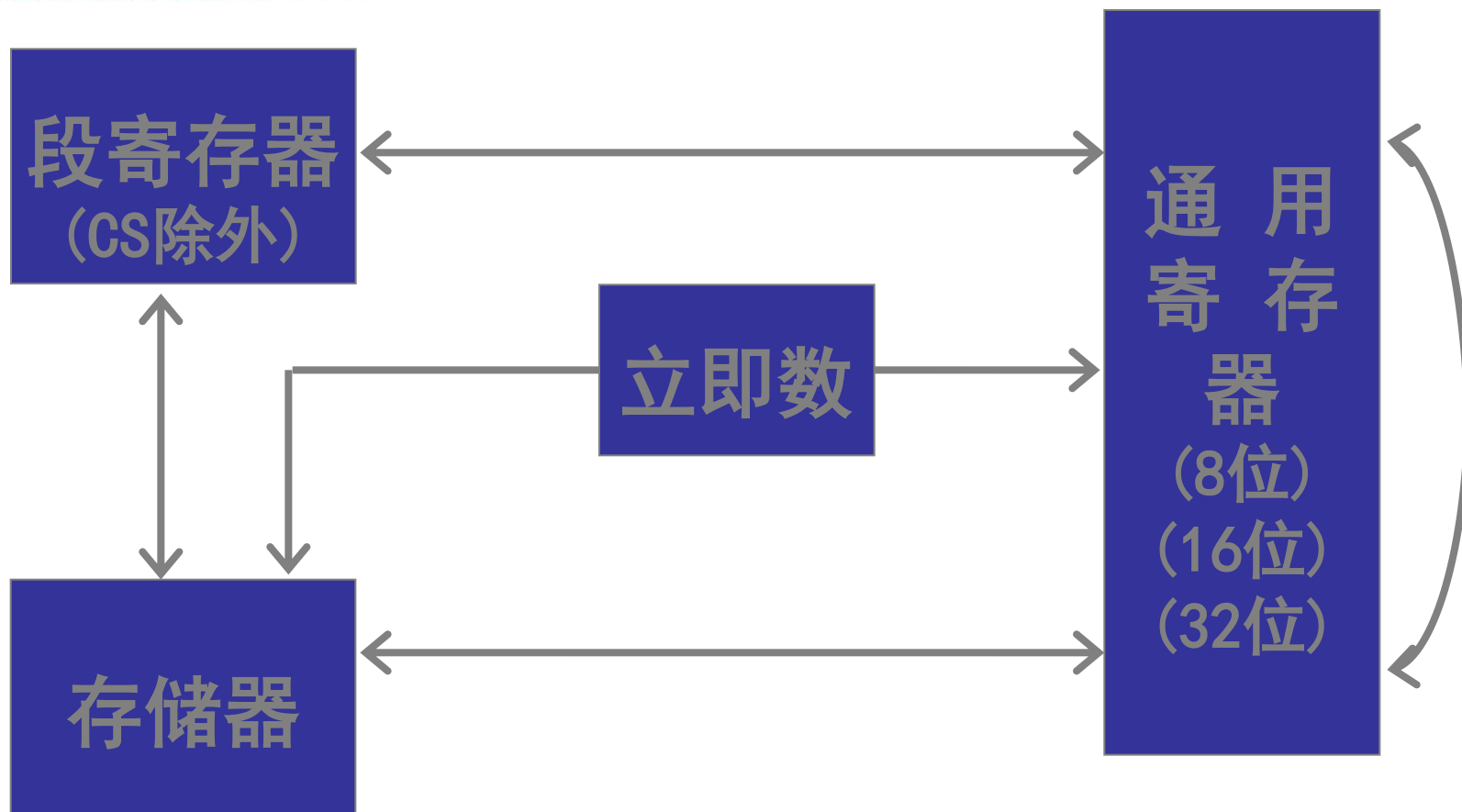


图3-12 MOV指令数据传送方向示意图



数据传送规则：

- 立即数不能作为目标操作数；
- 立即数不能直接送段寄存器；
- 目标寄存器不能是CS；
- 两个段寄存器间不能直接传送；
- 两个存储单元之间不能直接传送。





例：

```
MOV    AL, 5
MOV    DS, AX
MOV    [BX], AX
MOV    ES:VAR, 12
MOV    WORD PTR [BX], 10
MOV    EAX, EBX
```

“**WORD PTR**”，它明确指出BX所指向的内存单元为字型，立即数12被汇编为16位的二进制数。若要生成8位的二进制数，需要用

“**BYTE PTR**”，这里的类型显式说明是必须的

。





(2). 带符号扩展的数据传送指令 **MOVSX** (386以上)

格式: **MOVSX** DST, SRC

(3). 带零扩展的数据传送指令 **MOVZX**
(386以上)

格式: **MOVZX** DST, SRC





(4). 交换指令 **XCHG**

格式: XCHG OPR1, OPR2

功能: 交换两个操作数。

说明: OPR是操作数, 操作数可以是8位、16位、32位。该指令可能的组合是:

XCHG 寄存器操作数, 寄存器操作数

XCHG 寄存器操作数, 存储器操作数

XCHG 存储器操作数, 寄存器操作数



例：

设：(AX) = 1234H, (BX) = 4567H

则：XCHG AX, BX

执行后 (AX) = 4567H, (BX) = 1234H

由于系统提供了这个指令，因此，采用其他方法交换时，速度将会较慢，并需要占用更多的存储空间，编程时要避免这种情况。





2. 堆栈操作指令

堆栈数据的存取原则是“LIFO”。

堆栈段段基址→SS

堆栈栈顶地址→SP/ESP

堆栈用途：

对现场数据的保护与恢复、子程序与中断服务返回地址的保护与恢复等。



(1). 进栈指令 **PUSH**

格式: **PUSH SRC**

功能: $SP = SP - 2$

$SS:SP = (SRC)$

说明: SRC可以是16位或32位(386以上)的寄存器操作数或存储器操作数。在80286以上的机器中, SRC还可以是立即数。若SRC是16位操作数, 则堆栈指针减2; 若SRC是32位操作数, 则堆栈指针减4。





(2). 出栈指令 **POP**

格式: POP DST

功能: $(DST) = SS:SP$

$$SP = SP + 2$$

说明: DST可以是16位或32位(386以上)的寄存器操作数和存储器操作数,也可以是除CS寄存器以外的任何段寄存器。若DST是16位,则堆栈指针加2;若DST是32位,则堆栈指针加4。





常见指令序列:

PUSH AX

PUSH BX

.....

PUSH 1234H ; 80286以

上可用

POP DX

.....

POP BX

POP AX

注意堆栈的初始设置/堆栈异常





(3). 全部16位通用寄存器进栈指令 **PUSHA**

格式: PUSHA (286以上)

功能: 把8个16位通用寄存器的内容压入堆栈, 入栈顺序是AX、CX、DX、BX、SP (PUSHA指令执行前的值)、BP、SI、DI。入栈后SP值被减去16。

(4). 全部16位通用寄存器出栈指令 **POPA**

格式: POPA (286以上)





(5). 全部扩展通用寄存器进栈指令 **PUSHAD**

格式: **PUSHAD** (386以上)

功能: 把8个扩展通用寄存器的内容压入堆栈, 入栈顺序是EAX、ECX、EDX、EBX、ESP (PUSHAD指令执行前的值)、EBP、ESI、EDI。入栈后ESP值被减去32。

(6). 全部扩展通用寄存器出栈指令 **POPAD**

格式: **POPAD** (386以上)





3. 输入输出指令

(1). 输入指令 **IN**

格式: **IN ACR, PORT**

功能: 把外设端口 (PORT) 的内容传送给累加器 (ACR)。

说明: 可以传送8位、16位、32位, 相应的累加器选择AL、AX、EAX。





例.

IN AL, 61H ; AL ← (61H端口)

IN AX, 20H ; AX ← (20H端口)

MOV DX, 3F8H

IN AL, DX ; AL ← (3F8H端口)

IN EAX, DX

; EAX ← (DX所指向的端口)





(2). 输出指令 **OUT**

格式: **OUT** **PORT, ACR**

功能: 把累加器的内容传送给外设端口。

说明: 对累加器和端口号的选择限制同 **IN** 指令。





例.

```
MOV AL, 0
```

```
OUT 61H, AL ;61H端口 ← (AL)
```

;关掉PC扬声器

```
MOV DX, 3F8H
```

```
OUT DX, AL ;3F8H端口 ← (AL)
```

;向COM1端口输出一个字

符





4. 查表转换指令XLAT

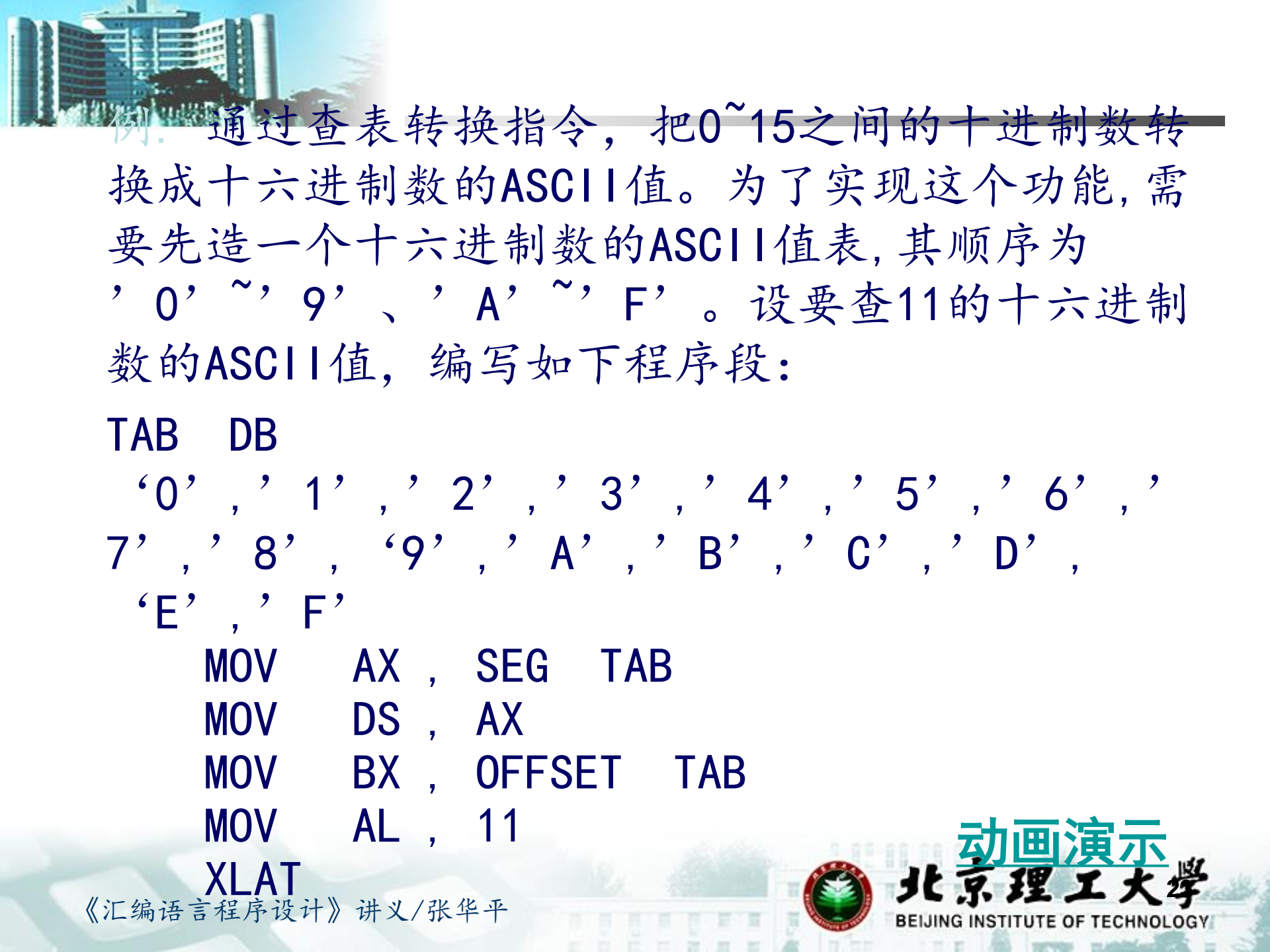
格式：XLAT

功能：通过AL寄存器中的索引值在表中查得表项内容并返回到AL中。

前提条件：

1. 数据段中应有一个字节型表；
2. 表起始地址的偏移量→BX；
3. 表索引值→AL。





例. 通过查表转换指令，把0~15之间的十进制数转换成十六进制数的ASCII值。为了实现这个功能，需要先造一个十六进制数的ASCII值表，其顺序为‘0’~‘9’、‘A’~‘F’。设要查11的十六进制数的ASCII值，编写如下程序段：

```
TAB DB
```

```
    ‘0’ , ‘1’ , ‘2’ , ‘3’ , ‘4’ , ‘5’ , ‘6’ ,  
    7’ , ‘8’ , ‘9’ , ‘A’ , ‘B’ , ‘C’ , ‘D’ ,  
    ‘E’ , ‘F’
```

```
    MOV     AX , SEG TAB
```

```
    MOV     DS , AX
```

```
    MOV     BX , OFFSET TAB
```

```
    MOV     AL , 11
```

```
    XLAT
```





5. 地址传送指令

这类指令传送的是操作数的地址，而不是操作数本身。





(1). 传送有效地址指令 **LEA**

格式: LEA REG, SRC

功能: 把源操作数的有效地址
送给指定的寄存器。

说明: 源操作数必须是存储器
操作数。

例.

LEA SI, TAB

LEA BX, TAB [SI]

LEA DI, ASCTAB [BX] [SI]

动画





(2). 加载数据段指针指令 **LDS**

格式: **LDS REG, SRC**

功能: 把源操作数中的**FAR**型指针拷贝到**DS**和指令中指定的通用寄存器。

说明:

1. 源操作数必须是存储器操作数

2. **REG是16位→源操作数必须是32位**

REG是32位→源操作数必须是48位

3. **低位→REG; 高位→DS;**

4. 可用于为处理不在当前数据段的数据做准备, 例如字符串指令的源操作数。





例. `LDS SI, ADDR`

若 $(DS:ADDR) = 78563412H$

则这条指令的执行结果是:

$(DS) = 7856H$

$(SI) = 3412H$





(3). 加载附加数据段指针指令 LES

格式: LES REG, SRC

(4). 加载FS数据段指针指令LFS (386以上)

格式: LFS REG, SRC

(5). 加载GS数据段指针指令LGS (386以上)

格式: LGS REG, SRC

(6). 加载堆栈段指针指令 LSS (386以上)

格式: LSS REG, SRC





MOV 指令传送地址

MOV 指令除了可以实现数据传送外,还可实现地址传送,方法是借助于**SEG**和**OFFSET**操作符。





例：设TAB为一条语句的符号地址，则可以有
以下指令：

MOV AX, SEG TAB

；把TAB的段基址送给AX寄存器

MOV DI, OFFSET TAB

；把TAB的偏移量送给DI寄存器

MOV DI, TAB MOV DI, TAB+2

LEA DI, TAB





6. 标志传送指令

指令的操作过程

考虑这些指令对**标志位**的影响！





(1). 16位标志进栈指令 **PUSHF**

格式: PUSHF

功能: SP减2; FLAGS→栈

顶单元。

(2). 16位标志出栈指令 **POPF**

格式: POPF





(3). 32位标志进栈指令 **PUSHFD**

格式: PUSHFD

功能: ESP减4; EFLAGS→栈顶

。

(4). 32位标志出栈指令 **POPFD**

格式: POPFD

(5). 标志送AH指令 **LAHF** [低8位]

格式: LAHF

(6). AH送标志寄存器指令 **SAHF**

格式: SAHF





3.4 算术运算指令

⑩ 二进制算术运算指令

⑩ 十进制算术运算指令

对于其中的双操作数指令，
其两个操作数寻址方式的限定同MOV指令

。





1. 二进制算术运算指令

1. 实现二进制算术运算。
2. 操作数和运算结果为二进制。
3. 操作数及计算结果:8位、16位、32位无符号或带符号二进制数(在书写指令时可以用十进制形式表示)。
4. 带符号数在机器中用补码形式表示, 最高位为符号位, 0表示正数, 1表示负数。





一、类型转换指令

这类指令实际上是把操作数的最高位进行扩展，用于处理带符号数运算的操作数类型匹配问题。这类指令均不影响标志。





(1). 字节扩展成字指令 **CBW**

格式:**CBW**

功能:把AL寄存器中的符号位值扩展到AH中

```
MOV    AL, FFH
```

例.

```
MOV    AL, 5
```

```
CBW                ; (AH) = 0, AL值不变
```

```
MOV    AL, 98H
```

```
CBW                ; (AX) = 0FF98H, AL值
```

不变

《汇编语言程序设计》讲义/张华平



北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY



(2). 字扩展成双字指令 CWD

格式: **CWD**

功能: 把AX的符号位值扩展到DX中

(3). 双字扩展成四字指令 CDQ

格式: **CDQ** (386以上)

功能: EAX符号位扩展到EDX中

(4). AX符号位扩展到EAX指令 CWDE

格式: **CWDE** (386以上)

功能: AX寄存器符号位扩展到EAX高16位





二、二进制加法指令

任何一条二进制加、减法指令均适用于带符号数和无符号数运算。





(1). 加法指令 **ADD**

格式: `ADD DST, SRC`

功能: $(DST) + (SRC) \rightarrow DST$

说明: 对操作数的限定同MOV指令。

标志: 影响**OF**、SF、ZF、AF、PF、**CF**标志

例. `ADD AL, BL`

`ADD CL, 6`

`ADD WORD PTR[BX], 56`

`ADD EDX, EAX`

动画





ADD运算的说明:

对于两个二进制数进行加法运算，如果把数解释为无符号数，其结果可能是溢出的，而如果把数解释为带符号数，其结果可能是不溢出的，反之也一样。

判定条件:

无符号数相加结果若使CF置1，则表示溢出；

带符号数相加结果若使OF置1，则表示溢出。

一旦发生溢出，结果就不正确了。表3-2以8位数为例说明了这种情况。



	二进制数加法	解释为无符号数	解释为带符号数
a. 带符号数 和无符号 数不溢出	$\begin{array}{r} 00000100 \\ + 00001011 \\ \hline 00001111 \end{array}$	$\begin{array}{r} 4 \\ + 11 \\ \hline 15 \\ CF=0 \end{array}$	$\begin{array}{r} +4 \\ + (+11) \\ \hline +15 \\ OF=0 \end{array}$
b. 无符号 数溢出	$\begin{array}{r} 00000111 \\ + 11111011 \\ \hline 1\ 00000010 \end{array}$ <div>CF ← 1</div>	$\begin{array}{r} 7 \\ + 251 \\ \hline 258 \text{ (错)} \\ CF=1 \end{array}$	$\begin{array}{r} +7 \\ + (-5) \\ \hline +2 \\ OF=0 \end{array}$
c. 带符号 数溢出	$\begin{array}{r} 00001001 \\ + 01111100 \\ \hline 10000101 \end{array}$	$\begin{array}{r} 9 \\ + 124 \\ \hline 133 \\ CF=0 \end{array}$	$\begin{array}{r} +9 \\ + (+124) \\ \hline -123 \text{ (错)} \\ OF=1 \end{array}$
d. 带符号数 和无符号 数都溢出	$\begin{array}{r} 10000111 \\ + 11110101 \\ \hline 1\ 01111100 \end{array}$ <div>CF ← 1</div>	$\begin{array}{r} 135 \\ + 245 \\ \hline 124 \text{ (错)} \\ CF=1 \end{array}$	$\begin{array}{r} -121 \\ + (-11) \\ \hline +124 \text{ (错)} \\ OF=1 \end{array}$

表3-2 对二进制数加法结果的解释



对于情况b:

最高位向前有进位，该进位记录在CF中，7加251应该等于258，但在有效的8位结果中，只看到2，这是因为丢掉了 $2^8=256$ ，所以对于无符号数来讲，通过判断 $CF=1$ ，便知该结果是错的。

[返回](#)





对于情况c:

因为两同号数相加，和的符号却与加数符号相反. 所以对于带符号数来讲，该结果是错的，发生这种情况后系统会置 $OF=1$ ，所以可通过 OF 来判断。

[返回](#)





(2). 带进位加法指令 **ADC**

格式: **ADC** DST, SRC

功能: $(DST) + (SRC) + \text{CF} \rightarrow DST$

说明: 对操作数的限定同**MOV**指令, 该指令
适用于多字节或多字的加法运
算。

标志: 影响**OF**、**SF**、**ZF**、**AF**、**PF**、**CF**标志

例. **ADC** AX, 35

; $(AX) = (AX) + 35 + CF$





(3). 加1指令 **INC**

格式: **INC DST**

功能: $(DST) + 1 \rightarrow DST$

标志: 除**不影响CF标志**外, 影响其它五个算术运算特征标志。

例. **INC BX**

例. 实现+2操作: **ADD AX, 2**

INC AX

INC AX





(4). 互换并加法指令 **XADD** (486以上)

格式: **XADD DST, SRC**

功能: $(DST) + (SRC) \rightarrow TEMP$

$(DST) \rightarrow SRC$

$TEMP \rightarrow DST$

说明: **TEMP**是临时变量。该指令执行后, 原
DST的内容在SRC中, 和在DST中

。

标志: 影响OF、SF、ZF、AF、PF、CF。



三、二进制减法指令

(1). 减法指令 SUB

格式: SUB DST, SRC

功能: $(DST) - (SRC) \rightarrow DST$

标志: 影响OF、SF、ZF、AF、PF、CF标志。

例. SUB AX, 35

SUB WORD PTR[BX], 56

减法指令执行后若使 $CF=1$ ，则对无符号数而言发生了溢出。若使 $OF=1$ ，则对带符号数而言发生了溢出。





(2). 带借位减法指令 **SBB**

格式: **SBB DST, SRC**

功能: $(DST) - (SRC) - CF \rightarrow DST$

说明: 除了操作为减外, 其它要求同ADC, 该指令适用于多字节或多字的减法运算。

标志: 影响OF、SF、ZF、AF、PF、CF标志

例. **SBB AX, 35**

; $(AX) = (AX) - 35 - CF$





(3). 减1指令 **DEC**

格式: **DEC DST**

功能: $(DST) - 1 \rightarrow DST$

说明: 使用本指令可以很方便地实现地址指针或循环次数的减1修改。

标志: 除不影响CF标志外, 影响其它五个算术运算特征标志。

例. **DEC BX**





(4). 比较指令 **CMP**

格式: **CMP** DST, SRC

功能: (DST) — (SRC), 影响标志位。

说明: 这条指令执行相减操作后只根据结果设置标志位, 并不改变两个操作数的原值, 其它要求同SUB。CMP指令常用于比较两个数的大小。

标志: 影响OF、SF、ZF、AF、PF、CF

标志

例. **CMP** AX, [BX]





(5). 求补指令 **NEG**

格式: **NEG DST**

功能: 对目标操作数 (含符号位) 求反加1, 并且把结果送回目标。即: 实现 $0 - (DST) \rightarrow DST$

说明: 利用**NEG**指令可实现求一个数的相反数。

标志: 影响**OF**、**SF**、**ZF**、**AF**、**PF**、**CF**标志。其中

对CF和OF的影响如下:

- a. 对操作数所能表示的最小负数 (例若操作数是8位则为-128) 求补, 原操作数不变, 但**OF**被置1
- b. 当操作数为0时, 清0 **CF**。
- c. 对非0操作数求补后, 置1 **CF**。





例. 实现 $0 - (AL)$ 的运算

NEG AL

例. EAX 中存放一负数, 求该数的绝对值

NEG EAX





例. 试编写
两个三字节
长的二进制
数加法程序,
加数FIRST、
SECOND及和
SUM的分配情
况如图所示
。



图3-17 三字节长
加法程序示意图



北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY



分析：为便于理解和方便讨论

，设 $(FIRST) = 112233H$, $(SECOND) = 445566H$, 存放顺序如上页图所示。

算法类似于手工计算, 从最低字节数据开始加起, 计算高字节时要考虑低字节的进位问题。为了简化讨论, 假设和不会超过三字节, 即最高位不会有进位产生。

为了最大限度地用到我们所学的指令和寻址方式, 本程序不使用循环结构, 不考虑优化问题。因为是三字节数据, 所以不能用字或双字加法指令实现。





经过分析，编程如下：

```
LEA  DI, SUM           ;建立和的地址指针DI
ADD  DI, 2              ;DI指向和的低字节
MOV  BX, 2
MOV  AL, FIRST[BX]      ;取FIRST的低字节（本例为33H）
ADD  AL, SECOND+2        ;两个低字节相加，和①在AL中，进位反映在CF中
MOV  [DI], AL           ;把低字节和存到DI指向的单元（本例为SUM+2单元）
DEC  DI                 ;修改和指针，使其指向中字节
DEC  BX                 ;修改加数指针，使其指向中字节
MOV  AL, FIRST[BX]      ;取FIRST的中字节（本例为22H）
ADC  AL, SECOND+1        ;两个中字节相加且加CF，和②在AL中，进位反映在CF中
MOV  [DI], AL           ;把中字节和存到DI指向的单元（本例为SUM+1单元）
DEC  DI                 ;修改和指针，使其指向高字节
DEC  BX                 ;修改加数指针，使其指向高字节
MOV  AL, FIRST[BX]      ;取FIRST的高字节（本例为11H）
ADC  AL, SECOND          ;两个高字节相加且加CF，和③在AL中，进位反映在CF中
MOV  [DI], AL           ;把高字节和存到DI指向的单元（本例为SUM单元）
```




讨论:

① 两个ADC指令能否换为ADD

?

② DEC DI和SUB DI, 1指令

的功能从表面上看是等价的, 是否可以互换

?





③ 多字节或多字加减时，CF始终有意义(前边的反映进 / 借位，最后一次反映溢出情况)，而OF只有最后一次的才有意义。

④ 溢出情况判断：若是两个无符号数相加，则当最后一次的CF被置1时，表示溢出，结果不正确；若是两个带符号数相加，则当最后一次的OF被置1时，表示溢出，结果不正确。

从该例可以看出，选取合适的算法、恰当的指令、灵活的寻址方式对汇编语言程序设计人员十分重要，它可以达到事半功倍的效果。



四、二进制乘法指令

(1). 无符号乘法指令 **MUL**

格式: **MUL** $\text{SRC}_{\text{reg} / \text{m}}$

功能: 实现两个无符号二进制数乘。

说明: 该指令只含一个源操作数, 另一个乘数必须事前放在累加器中。可以实现8位、16位、32位无符号数乘。



具体操作为：

字节型乘法： $(AL) \times (SRC)_8 \rightarrow AX$

字型乘法： $(AX) \times (SRC)_{16} \rightarrow DX:AX$

双字型乘法： $(EAX) \times (SRC)_{32} \rightarrow EDX:EAX$

标志：影响CF、OF、SF、ZF、AF、PF，而只有CF、OF有意义，其它标志不确定。

对CF和OF的影响是：若乘积的高半部分(例字节型乘法结果的AH)为0则对CF和OF清0，否则置CF和OF为1。





例.

MOV AL, 8

MUL BL ; $(AL) \times (BL)$, 结果在AX中

MOV AX, 1234H

MUL WORD PTR [BX]

; $(AX) \times ([BX])$, 结果在

DX:AX中

MOV AL, 80H

SUB AH, AH ; 清0 AH

MUL BX ; $(AX) \times (BX)$, 结果在DX:AX中





(2). 带符号乘法指令 **IMUL**

功能：实现两个带符号二进制数乘。

格式1: **IMUL SRC_{reg / m}**

说明：这种格式的指令除了是实现两个带符号数相乘且结果为带符号数外，其它与**MUL**指令相同。所有的**80X86 CPU**都支持这种格式。





具体操作为：

字节型乘法： $(AL) \times (SRC)_8 \rightarrow AX$

字型乘法： $(AX) \times (SRC)_{16} \rightarrow DX:AX$

双字型乘法： $(EAX) \times (SRC)_{32} \rightarrow EDX:EAX$

标志：

影响CF、OF、SF、ZF、AF、PF，而只有CF、OF有意义，其它标志不确定。对CF和OF的影响是：若乘积的高半部分为低半部分的符号扩展，则对CF和OF清0，否则置CF和OF为1。





格式2: **IMUL** **REG, SRC**_{reg / m} (286以上)

说明: REG和SRC的长度必须相同, 目标操作数
REG必须是16位或32位通用寄存器, 源操作数
SRC可以是寄存器或存储器操作数。

具体操作为:

$$(\text{REG})_{16} \times (\text{SRC})_{16} \rightarrow \text{REG}_{16}$$

$$(\text{REG})_{32} \times (\text{SRC})_{32} \rightarrow \text{REG}_{32}$$

格式3: **IMUL** **REG, imm₈** (286以上)

格式4: **IMUL** **REG, SRC**_{reg / m}, **imm₈** (286以上)





五、二进制除法指令

(1). 无符号除法指令 **DIV**

格式: **DIV** $\text{SRC}_{\text{reg} / \text{m}}$

功能: 实现两个无符号二进制数除法。

说明: 该指令只含一个源操作数, 该操作数作为除数使用, 注意它不能是立即数。被除数必须事前放在隐含的寄存器中。可以实现8位、16位、32位无符号数除。





具体操作为：

字节型除法： $(AX) \div (SRC)_8 \rightarrow \begin{cases} \text{商：AL} \\ \text{余数：AH} \end{cases}$

字型除法： $(DX:AX) \div (SRC)_{16} \rightarrow \begin{cases} \text{商：AX} \\ \text{余数：DX} \end{cases}$

双字型除法： $(EDX:EAX) \div (SRC)_{32} \rightarrow \begin{cases} \text{商：EAX} \\ \text{余数：EDX} \end{cases}$

标志：不确定。

例. 实现 $1000 \div 25$ 的无符号数除法。

```
MOV  AX, 1000
```

```
MOV  BL, 25
```

```
DIV  BL
```

; (AX) \div (BL)、商在AL中、余数在AH中

例. 实现 $1000 \div 512$ 的无符号数除法。

```
MOV  AX, 1000
```

```
SUB  DX, DX      ; 清0 DX
```

```
MOV  BX, 512
```

```
DIV  BX
```

; (DX:AX) \div (BX)、商在AX中、余数在DX中





(2). 带符号除法指令 **IDIV**

格式: **IDIV** $\text{SRC}_{\text{reg} / \text{m}}$

功能: 实现两个带符号二进制数除。

说明: 除了是实现两个带符号数相除且商和余数均为带符号数、余数符号与被除数相同外, 其它与 **DIV** 指令相同。

具体操作同无符号数除法。





例. 实现 $(-1000) \div (+25)$ 的带符号数除法。

MOV AX, -1000

MOV BL, 25

IDIV BL

; (AX) \div (BL)、商在AL中、余数在AH中

例. 实现 $1000 \div (-512)$ 的带符号数除法。

MOV AX, 1000

CWD ; AX的符号扩展到DX

MOV BX, -512

IDIV BX

; (DX:AX) \div (BX)、商在AX中、余数在DX中



注意: 若除数为0或商超出操作数所表示的范围 (例如字节型除法的商超出8位) 会产生除法错中断, 此时系统直接进入0号中断处理程序, 为避免出现这种情况, 必要时在程序中应事先对操作数进行判断。



感谢关注聆听！



张华平

Email: kevinzhang@bit.edu.cn

微博: @ICTCLAS张华平博士

实验室官网:

<http://www.nlpir.org>



大数据千人会

