

第三章 指令系统

第3.1节 8086寻址方式

指令是使计算机执行某种特定操作的二进制编码

指令系统是所有指令的集合

一条指令通常包括两部分

- **操作码**：规定所要执行的操作类型
- **操作数**：所需要处理的数据或者数据的地址信息

获得数据或者数据地址信息的方式称为**寻址方式**

例：MOV AX, 1234H

指令中有两个操作数，将前者称为**目的操作数**，后者称为**源操作数**，该指令将1234H传送到寄存器AX中。

一. 数据寻址方式

1. 立即数寻址

指令中直接给出操作数，指令执行时可以立即得到，此时把操作数又称作“**立即数**”

例：MOV AL,5 ;AL←5

2. 寄存器寻址

操作数放在CPU内部的寄存器中，在指令中直接指出寄存器的名字

例：INC CX ;将CX的内容加1

3. 隐含寻址

指令已经默认是对CPU中的某个寄存器操作

例：DAA ;默认对AL进行十进制加法调整

以上三种方式均直接从CPU内部获得数据，操作速度快

4. 直接寻址

操作数放在存储器中，存储单元的有效地址直接由指令给出

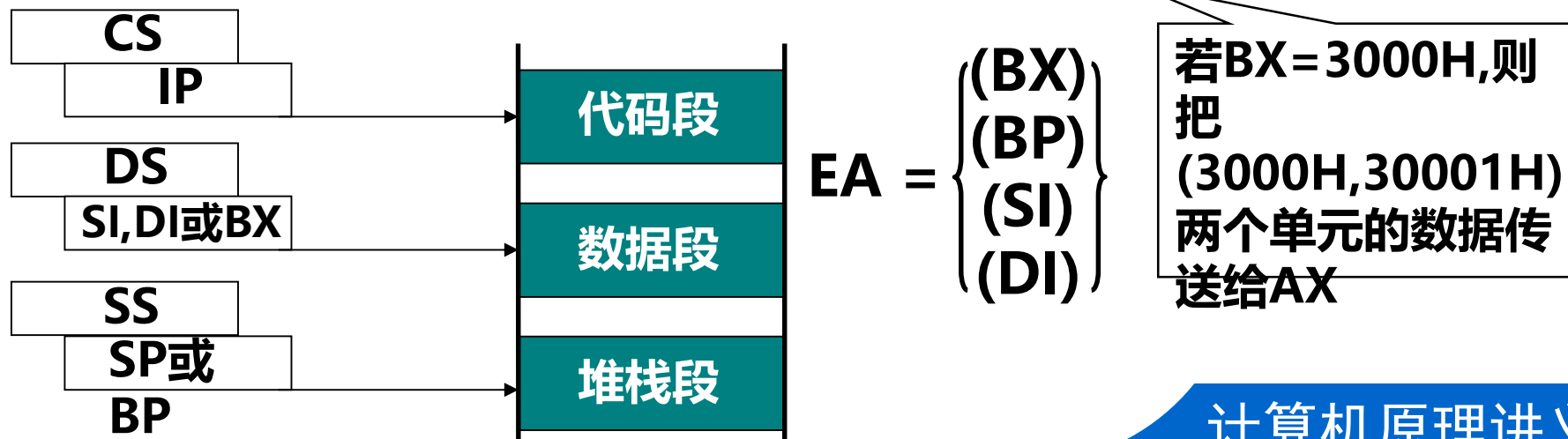
例：MOV SI, [2000H] ;SI←(2000H, 2001H)

CPU执行单元EU根据指令的寻址方式计算出16位的偏移量，称为**有效地址(EA)**，一般用 [有效地址] 表示。

5. 寄存器间接寻址

操作数一定在存储器中，存储单元的有效地址由寄存器指出，

寄存器可以为BX、BP、SI、DI， 例：MOV AX, [BX]



若需要对其它段寻址，则必须用前缀指出段寄存器名，又称“段超越”

例：MOV BL, ES:[SI] ;BL ← 扩展段1000H单元的内容

注：IP只能在代码段寻址，SP只能在堆栈段寻址，不能进行段超越

6. 寄存器相对寻址

操作数的有效地址由寄存器内容加上一个8位或16位的偏移量得到

例：MOV AX, [BX+1000H] ;AX ← BX+1000H 所指向的存储单元内容

若BX = 2000H，则将数据段(3000H, 3001H) 的内容传送给AX

指令也可书写为：MOV AX, 1000H[BX]

7. 基址变址寻址

操作数的有效地址由基址寄存器和变址寄存器的内容相加产生

例：MOV AX, [BP(BX)] ;AX ← (BP) + SI
EA = { (BP) } + { (DI) }

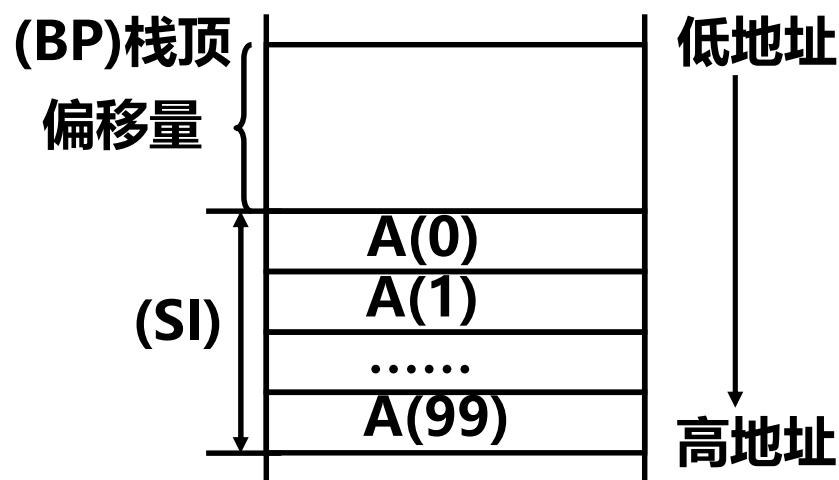
8. 相对基址变址寻址

操作数的有效地址由基址寄存器、变址寄存器和偏移量三者相加产生

例：MOV AX, [BX+SI+1000H] ;AX ← (BX+SI+1000H)

❖ 相对基址变址在数组访问中的应用

例：若访问堆栈段中的数组A[100]时，可以用BP存放栈顶地址，用偏移量表示数组中第一个元素到栈顶的距离，用变址寄存器SI（或DI）指向具体数组元素



第4~8种寻址方式，操作数一定在存储器中，由于计算EA、总线操作需要时间，因此比1~3方式慢

二. I/O端口寻址

操作数在I/O端口中时，必须通过累加器（AX或AL）实现对端口的访问

1. 直接端口寻址 — 指令直接提供8位端口的地址

例：IN AL, 63H ;AL←(63H)

端口寻址时，地址不加[]，当端口地址可用一个字节来表示时，可使用直接端口寻址

2. 间接端口寻址 — 由DX寄存器给出16位端口地址

例：MOV DX, 162H

IN AX, DX

从端口162H读取一个字的数据传送到AX中，DX间接表示端口地址

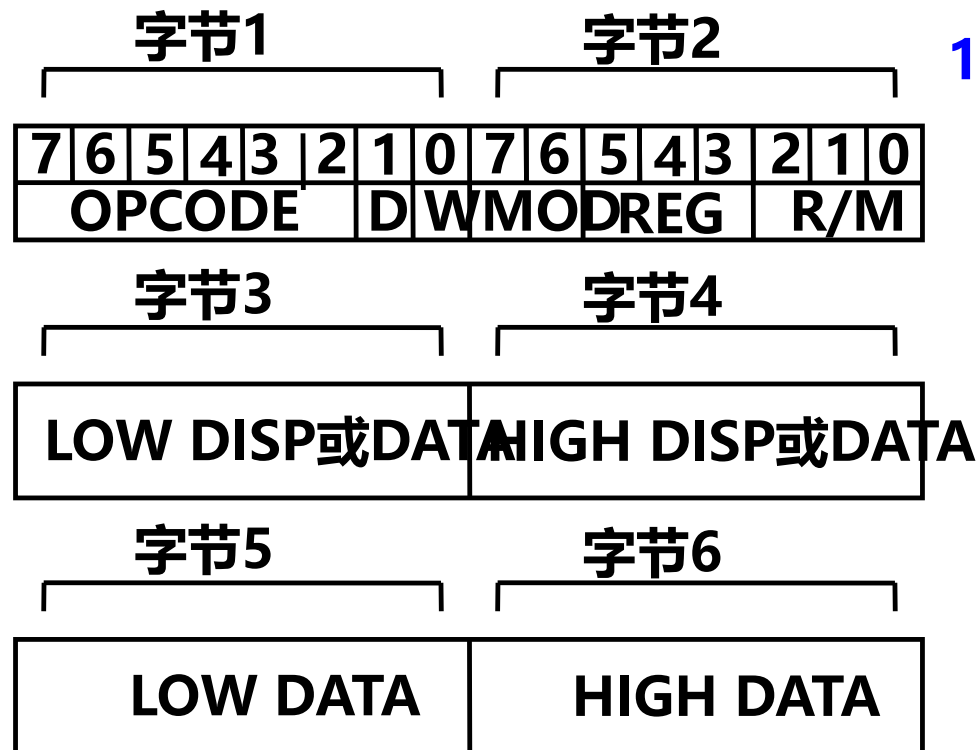
比较：MOV AX, DX ;AX←DX

IN AX, DX ;AX←(DX)

第3.2节 8086指令格式

一. 指令格式

指令一般由1 ~ 6个字节组成，具体格式如下：



1.字节1

① **OPCODE**: 指令操作码

② **D**: 操作数传输方向(立即数指令和串操作指令除外)

D=0, Reg为源操作数

D=1, Reg为目的操作数

③ **W**: 操作数字节长度

W=0, 字节操作

W=1, 字操作

2. 字节2，寻址方式

① MOD：寻址方式字段

- 00 = 存储器方式，指令中无偏移量
- 01 = 存储器方式，指令中有8位偏移量
- 10 = 存储器方式，指令中有16位偏移量
- 11 = 寄存器方式，指令中无偏移量

② Reg：寄存器编码字段

000=AL/AX 100=AH/SP

001=CL/CX 101=CH/BP

010=DL/DX 110=DH/SI

011=BL/BX 111=BH/DI

③ R/M: 寄存器/存储器字段

- 在MOD=11,寄存器模式下, R/M给出第二个操作数的寄存器编码
- 在MOD≠11,存储器模式下, R/M给出计算有效地址的方法

MOD=11			MOD ≠ 11 有效地址计算方法			
R/M	W=0	W=1	R/M	MOD=00	MOD=01	MOD=10
000	AL	AX	000	BX+SI	BX+SI+D8	BX+SI+D16
001	CL	CX	001	BX+DI	BX+DI+D8	BX+DI+D16
010	DL	DX	010	BP+SI	BP+SI+D8	BP+SI+D16
011	BL	DX	011	BP+DI	BP+DI+D8	BP+DI+D16
100	AH	SP	100	SI	SI+D8	SI+D16
101	CH	BP	101	DI	DI+D8	DI+D16
110	DH	SI	110	直接地址	BP+D8	BP+D16
111	BH	DI	111	BX	BX+D8	BX+D16

3. 字节3 ~ 字节6

- **DISP: 存储器操作数地址偏移量, 长度由MOD字段定义**
- **DATA: 指令中的立即数**

例: **MOV AX, [BX]**

指令码为: 8B 07

MOV [BX+SI+2000H], 2345H

指令码为: C7 80 00 20 45 23

MOV AX, BX

指令码为: 89 D8

第3.3节 8086指令介绍

8086 CPU共有133条指令，根据操作性质，可分为：

- **传输指令**
- **算术运算指令**
- **逻辑运算和移位指令**
- **串操作指令**
- **控制转移指令**

※ 操作数的符号表示

- ✓ **DST：目的操作数**
- ✓ **SRC：源操作数**
- ✓ **TARGET：循环、转移和调用指令中的目的操作数**

※ 操作数的符号表示

- ✓ REG: 寄存器操作数, 字节或字
- ✓ REG8: 8位寄存器操作数, 字节
- ✓ REG16: 16位寄存器操作数, 字
- ✓ MEM: 存储器操作数, 字节或字
- ✓ MEM8: 8位存储器操作数, 字节
- ✓ MEM16: 16位存储器操作数, 字
- ✓ MEM32: 32位存储器操作数, 双字
- ✓ ACC: 累加器AL或AX
- ✓ SEG_REG: 段寄存器
- ✓ IMM: 立即操作数, 字节或字
- ✓ IMM8: 立即操作数, 字节
- ✓ IMM16: 立即操作数, 字
- ✓ SHORT_LABEL: 短标号, 8位偏移量
- ✓ NEAR_LABEL: 近标号, 16位地址或偏移量
- ✓ FAR_LABEL: 远标号, 32位地址



一. 数据传送类指令

(一) 通用数据传送指令

1. **MOV DST, SRC**

➤ **操作: $DST \leftarrow SRC$**

➤ **说明:**

① **DST和SRC的组合关系是:**

a. **$REG/MEM \leftarrow IMM$**

b. **$REG/MEM \leftrightarrow REG$**

c. **$REG/MEM \leftrightarrow SEG_REG$**

② **CS、IP和立即数不能做目的操作数**

③ **不能在两个存储单元之间直接传送**

④ **MOV指令不改变源操作数, 不影响FLAG的状态标志**

⑤ **数据位数由寄存器决定或用说明符指出 (如例所示)**

⑥ **SS修改后, 下一条指令执行完才检测中断, 修改SS和SP一定要连续进行**

例: **MOV AX, 1000**

MOV [BP], AX

MOV ES, DX

MOV word ptr [SI], 1000

MOV byte ptr [SI], AL

MOV CS, 100H 错误

MOV [1000H], [2000H] 错误

(二) 堆栈操作

SP总是指向栈顶，即最后压入堆栈的信息单元

1. PUSH SRC 进栈指令

➤ 操作: $SP \leftarrow SP-2, (SP+1, SP) \leftarrow SRC$

➤ 具体指令: PUSH REG16

PUSH SEG_REG

PUSH MEM16

例: PUSH BX

PUSH CS

PUSH [SI]

2. POP DST 出栈指令

➤ 操作: $DST \leftarrow (SP+1, SP), SP \leftarrow SP+2$

➤ 具体指令: POP REG16

POP SEG_REG

POP MEM16

例: POP AX

POP ES

POP [SI]

PUSH 1000 错误

说明: 堆栈操作以字为单位，按后进先出原则存储数据

POP出栈操作时，CS不能为目的操作数

POP CS 错误

计算机原理讲义

※ 堆栈的应用

堆栈主要用于子程序调用、中断子程序的现场保护和恢复、参数传递等。

例：保护现场（保存存储器和恢复存储器次序应一一对应）

```
Fun: PUSH  DS
      PUSH  ES
      PUSH  AX
      PUSH  BX
      ;子程序处理
      POP   BX
      POP   AX
      POP   ES
      POP   DS
      RET
```


(三) 交换指令

1. XCHG 指令

- 格式: XCHG DST, SRC
- 操作: $DST \leftrightarrow SRC$
- 说明: 两个操作数必须有一个在寄存器中, 可在寄存器之间、寄存器与存储器之间交换信息, 目的操作数DST不能为CS
- 具体指令:
 - ✓ XCHG REG, REG
 - ✓ XCHG REG, MEM
 - ✓ XCHG MEM, REG

例: XCHG AL, byte ptr [2000H] ; $AL \leftrightarrow (2000H)$

XCHG AX, DX

XCHG AX, CS 错误

(四) I/O端口传送指令

1. IN 输入指令

➤ 操作: $AL/AX \leftarrow (PORT)/(DX)$, **PORT为8位直接端口地址**

➤ 具体指令:

IN	AL, PORT	例: IN	AL, 20H
IN	AX, PORT	IN	AX, 60H
IN	AL, DX	IN	AL, DX
IN	AX, DX	IN	AX, DX

2. OUT 输出指令

➤ 操作: $(PORT)/(DX) \leftarrow AL/AX$, **PORT为8位直接端口地址**

➤ 具体指令:

OUT	PORT, AL	例: OUT	20H, AL
OUT	PORT, AX	OUT	80H, AX
OUT	DX, AL	OUT	DX, AL
OUT	DX, AX	OUT	200H, AX 错误

说明: I/O端口地址有直接寻址和间接寻址 (通过DX) 两种寻址方式

对I/O端口操作必须通过累加器AX或AL来完成

(五) 换码指令

1. XLAT 换码指令

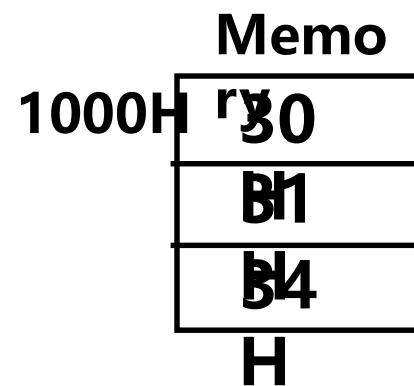
- 具体指令: XLAT
- 操作: $AL \leftarrow (BX+AL)$

例: MOV AL, 1

MOV BX, 1000H

XLAT ;将存储单元1001H的内容送给AL

结果: AL = 31H



(六) 有效地址 (EA) 传送指令

1. LEA (Load Effective Address) 取有效地址指令

- 具体指令: LEA REG16, SRC
- 操作: $REG16 \leftarrow SRC$ 的有效地址
- 说明: 该指令将源操作数的有效地址送给寄存器

例: MSG db 30H 31H 32H 33H ;定义数据

LEA BX, MSG ;将MSG的偏移地址送BX

MOV DX, MSG ;DH=31H, DL=30H
;低位在前,高位在后

MSG	30
	31
	32
	33
	H

2. LDS (Load DS With Pointer) 地址指针送寄存器和DS

- 具体指令: `LDS REG16, MEM32`
- 操作: $\text{REG16} \leftarrow (\text{MEM32}), \text{DS} \leftarrow (\text{MEM32}+2)$
- 说明: 将源操作数的4个字节, 分别传送到指定的寄存器和DS中

例: `LDS BX, [1000H]`

结果: 将(1000H,1001H)内容送BX, 将(1002H,1003H)内容送DS

3. LES (Load ES With Pointer) 地址指针送寄存器和ES

- 具体指令: `LES REG16, MEM32`
- 操作: $\text{REG16} \leftarrow (\text{MEM32}), \text{ES} \leftarrow (\text{MEM32}+2)$
- 说明: 将源操作数的4个字节, 分别传送到指定的寄存器和ES中

(七) 标志寄存器传送指令

1. LAHF (Load AH with Flag) 标志送AH
 - 格式: LAHF
 - 操作: $AH \leftarrow$ 标志寄存器的低八位
2. SAHF (Store AH into Flag) AH送标志寄存器
 - 格式: SAHF
 - 操作: 标志寄存器的低八位 $\leftarrow AH$
3. PUSHF (Push the Flag) 标志寄存器进栈
 - 格式: PUSHF
 - 操作: $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow FLAG$
 - 栈以字为单位
4. POPF (Pop the Flag) 标志寄存器出栈
 - 格式: POPF
 - 操作: $FLAG \leftarrow (SP + 1, SP), SP \leftarrow SP + 2$

注: 传送类指令除了SAHF和POPF外都不影响标志寄存器

二. 算术类指令

(一) 加法指令

1. ADD 加法

➤ 格式: **ADD DST, SRC**

➤ 操作: **DST \leftarrow SRC + DST**

➤ 具体指令

例:

✓ **ADD REG, REG**

ADD AL, DL

✓ **ADD REG, MEM**

ADD AL, [BX+DI+1000H]

✓ **ADD MEM, REG**

ADD [SI], DX

✓ **ADD REG, IMM**

ADD AX, 1000H

✓ **ADD MEM, IMM**

ADD [BP+SI], 1000H

ADD [SI], [DI] **错误**

2. ADC (Add With Carry) 带进位加法

- 格式: ADC DST, SRC
- 操作: $DST \leftarrow SRC + DST + CF$
- 具体指令: ADC REG, REG 例: ADC AX, DX
ADC REG, MEM ADC AL, [BX+DI+1000H]
ADC MEM, REG ADC [SI], DX
ADC REG, IMM ADC AX, 1000H
ADC MEM, IMM ADC [BP+SI], 1000H
ADC [SI], [DI] 错误

3. INC (Increment) 加1

- 格式: INC DST
- 操作: $DST \leftarrow DST + 1$
- 具体指令: INC REG 例: INC CX
INC MEM INC [2000H]
- 说明: INC指令影响除CF以外的所有状态标志

※加法类指令影响FLAG的状态标志

❖ 带进位加法指令ADC应用举例——实现多字节的加法运算

例：有2个4字节无符号数分别存放在数据段2000H和3000H开始的单元中，低位在前，高位在后，将两数相加，结果存放在数据段2000H开始的单元中

MOV	SI, 2000H	;SI指向第一个数首地址
MOV	DI, 3000H	;DI指向第二个数首地址
MOV	AX, [SI]	;取第一个数的低16位到AX
ADD	AX, [DI]	;两个数的低16位相加
MOV	[SI], AX	;低16位相加结果送2000H和2001H单元
MOV	AX, [SI + 2]	;取第一个数高16位
ADC	AX, [DI + 2]	;两个数高16位连同进位位相加
MOV	[SI + 2], AX	;高16位相加结果送2003H和2004H单元

(二) 减法指令

1. SUB (Subtract)减法

- 格式: SUB DST, SRC
- 操作: $DST \leftarrow DST - SRC$
- 具体指令

✓ SUB	REG, REG	例: SUB	AL, DL
✓ SUB	REG, MEM	SUB	AL, [BX+DI+1000H]
✓ SUB	MEM, REG	SUB	[SI], DX
✓ SUB	REG, IMM	SUB	AX, 1000H
✓ SUB	REG, IMM	SUB	[BP+SI], 1000H
✓ SUB	MEM, IMM	SUB	[SI], [DI] 错误

2. SBB (Subtract With Borrow) 带借位减法

- 格式: SBB DST, SRC
- 操作: $DST \leftarrow DST - SRC - CF$
- 具体指令: SBB REG, REG 例: SBB AX, DX
SBB REG, MEM SBB AL, [BX+DI+1000H]
SBB MEM, REG SBB [SI], DX
SBB REG, IMM SBB AX, 1000H
SBB MEM, IMM SBB [BP+SI], 1000H
SBB [SI], [DI] 错误

3. DEC (Decrement) 减1

- 格式: DEC DST
- 操作: $DST \leftarrow DST - 1$ ※减法类指令影响FLAG的状态标志
- 具体指令: DEC REG 例: DEC CX
DEC MEM DEC [2000H]
- 说明: DEC指令影响除CF以外的所有状态标志

4. NEG 求补运算

- 格式: NEG DST
- 操作: $DST \leftarrow \text{模} - DST$
- 说明: 求指令中给出的操作数的补码
- 具体指令: NEG REG 例: NEG AL ;将AL中的数取补码
NEG MEM NEG [2000H]
- 说明: 字节的模为100H, 字的模为10000H

5. CMP 比较指令

- 格式: CMP DST, SRC
- 操作: $DST - SRC$
- 具体指令: CMP REG, REG 例: CMP AX, DX
CMP REG, MEM CMP AL, [BX+DI+1000H]
CMP MEM, REG CMP [SI], DX
CMP REG, IMM CMP AX, 1000H
CMP MEM, IMM CMP [BP+SI], 1000H

说明: 做减法操作, 仅影响标志位, 不保存结果



算术类指令

(三) 乘法指令

1. MUL (unsigned Multiple) 无符号数乘法

- **格式：** MUL SRC
- **操作：** $AX \leftarrow AL * SRC$ 或 $\{DX, AX\} \leftarrow AX * SRC$
- **具体指令：** MUL REG 例： MUL BL
 MUL MEM MUL word ptr [2000H]

2. IMUL (signed Multiple) 有符号数乘法

- **格式：**IMUL SRC
- **操作：** $AX \leftarrow AL * SRC$ 或 $\{DX, AX\} \leftarrow AX * SRC$
- **具体指令：**IMUL REG 例：IMUL BL
IMUL MEM IMUL byte ptr [SI]

例：将2个32位无符号数DAT1和DAT2相乘，结果保存在64位无符号数Result中

		DAT1H	DAT1L
	*	DAT2H	DAT2L
		<hr/>	
		(DAT1L2L)H	(DAT1L2L)L
	(DAT1H2L)H	(DAT1H2L)L	
	(DAT2H1L)H	(DAT2H1L)L	
+	(DAT2H1H)H	(DAT2H1H)L	
		<hr/>	

(四) 除法指令

1. DIV (unsigned Divide) 无符号数除法

- 格式: DIV SRC
- 操作: $AL \leftarrow AX/SRC$, $AH \leftarrow AX\%SRC$;字节除法
 $AX \leftarrow \{DX, AX\}/SRC$, $DX \leftarrow \{DX, AX\}\%SRC$;字除法
- 具体指令: DIV REG 例: DIV BX
DIV MEM DIV word ptr [DI]

2. IDIV (signed Divide) 有符号数除法

- 格式: IDIV SRC
- 操作: $AL \leftarrow AX/SRC$, $AH \leftarrow AX\%SRC$;字节除法
 $AX \leftarrow \{DX, AX\}/SRC$, $DX \leftarrow \{DX, AX\}\%SRC$;字除法
- 具体指令: IDIV REG 例: IDIV BX
IDIV MEM IDIV byte ptr [2000H]

3. CBW (Convert Byte to Word) 字节扩展为字命令

- 格式: CBW
- 操作: 将AL扩展为AX
- 扩展方法: 使用最高位进行扩展
 - a. 若AL的最高位为0, 则扩展后AH全为0
 - b. 若AL的最高位为1, 则扩展后AH全为1

4. CWD (Convert Word to Double Word) 字扩展为双字命令

- 格式: CWD
- 操作: 将AX扩展为{DX, AX}, 扩展后DX为高16位
- 扩展方法: 使用最高位进行扩展
 - a. 若AX的最高位为0, 则扩展后DX全为0
 - b. 若AX的最高位为1, 则扩展后DX全为1

※ CBW和CWD说明

CBW和CWD只对符号数进行扩展, 无符号数的扩展只需使用MOV指令将其前面AH或DX全部清0。

(五) BCD码调整指令

8086 CPU中没有专用的BCD码运算指令，是使用二进制指令进行BCD码运算，然后再用BCD码调整指令对二进制运算结果进行调整，重新得到BCD码结果。例：DAA调整规则为：

a. 低4位 >9 或 $AF=1$ ，则结果加06H

b. 高4位 >9 或 $CF=1$ ，则结果再加上60H

1. 压缩格式BCD码调整指令：

DAA	;	对加法运算结果AL进行调整
DAS	;	对减法运算结果AL进行调整
2. 非压缩格式BCD码调整指令：

AAA	;	加法调整指令
AAS	;	减法调整指令
AAM	;	乘法调整指令
AAD	;	除法调整指令

例：下列程序段执行后AL=?, CF=?

```
MOV     AL, 89H
ADD     AL, 43H
DAA
```

三. 逻辑类指令

(一) 逻辑运算指令

1. AND “与” 运算命令

- 格式: AND DST, SRC
- 操作: $DST \leftarrow DST \& SRC$
- 具体指令:
 - ✓ AND REG, REG
 - ✓ AND REG, MEM
 - ✓ AND MEM, REG
 - ✓ AND REG, IMM
 - ✓ AND MEM, IMM

2. NOT “非” 运算命令

- 格式: NOT DST
- 操作: $DST \leftarrow \text{DST取反}$
- 具体指令:
 - NOT REG
 - NOT MEM

3. OR “或” 运算命令

- 格式: OR DST, SRC
- 操作: $DST \leftarrow DST \mid SRC$
- 具体指令:
 - ✓ OR REG, REG
 - ✓ OR REG, MEM
 - ✓ OR MEM, REG
 - ✓ OR REG, IMM
 - ✓ OR MEM, IMM

4. XOR “异或” 运算命令

- 格式: XOR DST, SRC
- 操作: $DST \leftarrow DST \wedge SRC$
- 具体指令:
 - ✓ XOR REG, REG
 - ✓ XOR REG, MEM
 - ✓ XOR MEM, REG
 - ✓ XOR REG, IMM
 - ✓ XOR MEM, IMM

5. TEST 测试命令

- 格式: TEST DST, SRC
- 操作: $DST \& SRC$, 仅影响标志位, 不保存结果
- 具体指令:

✓ TEST REG, REG	TEST REG, IMM
✓ TEST REG, MEM	TEST MEM, IMM
✓ TEST MEM, REG	

※ 逻辑运算指令常见用法

- ① **清0操作数**: `XOR AX, AX`, 不仅把AX清0, 而且影响状态标志, 至少 $CF=0$, $ZF=1$ (`MOV AX, 0`不影响标志位)
- ② **把某几位取反**: 用XOR指令, 把要取反的位和1异或, 不变的位和0异或。 (例: `XOR AL, 80H` ;将AL最高位取反)
- ③ **清0或置位某几位**: 用AND指令清0, 用OR指令置位; 要清0的位就与0, 要置1的位就或1。

例: 清0 `MOV AL, 33H` ; '3' 的ASCII码

`AND AL, 0FH` ; 将高4位清0

置1 `MOV AL, 9`

(二) 逻辑移位操作命令

1. SHL 逻辑左移指令

- 格式: SHL DST, count
- 操作: 操作数整体左移, 每次的最高位移到CF, 空余位补0
- 说明: count可以为1, 也可为CL, 做移位计数
- 具体指令:

- ✓ SHL REG, 1
- ✓ SHL REG, CL
- ✓ SHL MEM, 1
- ✓ SHL MEM, CL




- 注: 1) 移位数目大于1时, 必需用CL表示;
2) 所有逻辑移位操作的具体指令均类似

2. SHR 逻辑右移指令

- 格式: SHR DST, count 
- 操作: 操作数整体右移, 最低位到CF, 空余位补0

3. SAL 算术左移指令

- 格式: SAL DST, count 
- 说明: 该指令与SHL逻辑左移指令完全一致

4. SAR 算术右移指令

- 格式: SAR DST, count 
- 操作: 操作数整体右移, 最低位到CF, 空余位用符号位补

注: 1) 逻辑移位适用于无符号数, 算术移位适用于有符号数

2) 移位指令常用来做乘以2或除以2的操作

SHL、SAL用来乘以2, SHR、SAR用来除以2

例: MOV CL,5 ;若执行前(DI)=0064H

SAR [DI],CL ;执行后(DI)=0003H, 相当于 $100/32=3$

6. ROL 循环左移指令

➤ 格式: ROL DST, count

➤ 操作: 操作数整体左移, 最高位移到CF, 最低位用原来最高位补



7. ROR 循环右移指令

➤ 格式: ROR DST, count

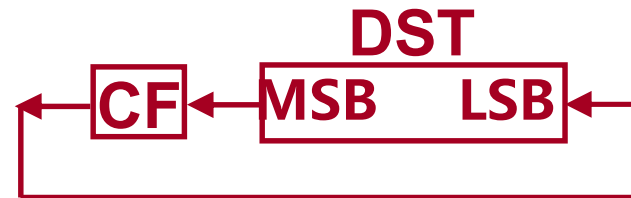
➤ 操作: 操作数整体右移, 最低位移到CF, 最高位用原来最低位补



8. RCL 带进位的循环左移指令

➤ 格式: RCL DST, count

➤ 操作: 类似ROL循环左移, 但CF加入到循环圈中



9. RCR 带进位的循环右移指令

➤ 格式: RCR DST, count

➤ 操作: 类似ROR循环左移, 但CF加入到循环圈中



四. 串操作类指令

对一串字符或数据操作，允许两个操作数均为MEM操作数，分为5种：

✓	MOVS (Move String)	串传送
✓	STOS (Store into String)	存AL中数据到串中
✓	LODS (Load from String)	从串中取数据到AL中
✓	CMPS (Compare String)	串比较
✓	SCAS (Scan String)	串扫描

※ 串操作说明

- ① 每次完成一个字节或一个字的操作，B表示字节，W表示字
- ② 多次操作需用重复前缀，如REP，重复次数由CX决定
- ③ 默认DS:SI指向存储器源操作数，ES:DI指向存储器目的操作数
- ④ SI、DI自动变化，DF指示变化方向
DF=0，SI、DI作增量变化，B: +1，W: +2
DF=1，SI、DI作减量变化，B: - 1，W: - 2

❖ 重复前缀 (CX又称作计数器)

- | | |
|------------------|-------------------|
| a. REP | 重复 |
| b. REPE / REPZ | 相等/为0 (ZF=1) 则重复 |
| c. REPNE / REPNZ | 不等/不为0 (ZF=0) 则重复 |

1. MOVS 串传送指令

- 格式: MOVSB 或 MOVSW
- 操作: MOVSB: $(ES : DI) \leftarrow (DS : SI), SI \leftarrow SI \pm 1, DI \leftarrow DI \pm 1$
MOVSW: $(ES : DI) \leftarrow (DS : SI), SI \leftarrow SI \pm 2, DI \leftarrow DI \pm 2$
- 说明: 使用前缀REP, 可完成多次数据传送

例: MOV CX, 12

REP MOVSB ;连续传送12个字节

2. STOS 存入串指令

- 格式: STOSB 或 STOSW
- 操作: STOSB: $(ES : DI) \leftarrow AL, DI \leftarrow DI \pm 1$
STOSW: $(ES : DI) \leftarrow AX, DI \leftarrow DI \pm 2$
- 说明: 可使用REP前缀

3. LODS 从串中取数据指令

- 格式: LODSB 或 LODSW
- 操作: LODSB: $AL \leftarrow (DS : SI), SI \leftarrow SI \pm 1$
LODSW: $AX \leftarrow (DS : SI), SI \leftarrow SI \pm 2$

4. SCAS 串扫描指令

- 格式: SCASB 或 SCASW
- 操作: 在数据中查找AL或AX中的内容
SCASB: $AL \leftarrow (ES : DI), DI \leftarrow DI \pm 1$
SCASW: $AX \leftarrow (ES : DI), DI \leftarrow DI \pm 2$
- 说明: 可使用REPE/REPZ 或 REPNE/REPZ, 仅影响标志位

5. CMPS 串比较指令

➤ 格式: CMPSB 或 CMPSW

➤ 操作: 两个MEM操作数的比较, 源操作数 - 目的操作数

CMPSB: (DS : SI) — (ES : DI), SI←SI±1, DI←DI±1

CMPSW: (DS : SI) — (ES : DI), SI←SI±2, DI←DI±2

➤ 说明: 可使用REPE / REPZ 或 REPNE / REPNZ, 该指令只影响标志位, 不保存结果

例: MOV CX, 0CH
REPZ CMPSB

※ 条件重复前缀说明

a. REPE / REPZ 相等或为0时重复

➤ 操作: 若CX≠0 且 ZF=1 (相等) 时继续执行, 否则退出

b. REPNE / REPNZ 不相等或不等于0时重复

➤ 操作: 若CX≠0 且 ZF=0 (不相等) 时继续执行, 否则退出

例1：将数据段中1000H开始的10个字节移动到2000H开始的

单元

MOV SI, 1000H ;置源操作数起始地址

MOV DI, 2000H ;置目的操作数起始地址

MOV AX, DS ;将DS赋给ES, 使ES:DI指向2000H开始单元

MOV ES, AX

MOV CX, 0AH ;初始化CX为10, 移动10个字节

CLD ;使DF=0, 使地址作增量变化

REP MOVSB ;(ES:DI)←(DS:SI), SI←SI+1, (DI)←(DI)+1

INT 3 ;断点指令

例2：将数据段中1000H开始的10个字移动到1002H开始的单

元去

MOV	SI, 1012H	;置源操作数结束地址
MOV	DI, 1014H	;置目的操作数结束地址
PUSH	DS	;将DS赋给ES, 使ES:DI指向1014H单元
POP	ES	
MOV	CX, 0AH	;初始化CX为10, 移动10个字
STD		;使DF=1, 使地址作减量变化
REP	MOVSW	;(ES:DI)←(DS:SI), SI←SI-2, (DI)←(DI)-2
INT	3	;断点指令

五. 控制转移类指令

该类指令是能够使程序执行流程发生改变的指令，分为5种：

- ✓ 无条件转移和条件转移指令
- ✓ 子程序调用和返回指令
- ✓ 循环控制指令
- ✓ 中断指令
- ✓ 处理器控制指令

指令需要寻找转移地址或调用地址，将此种寻址方式称为程序寻址方式

1. 段内直接转移方式 — 指令中直接给出跳转的相对偏移量

目标转移地址 = IP + 偏移量 (8位或16位)

- ✓ 偏移量为8位，称为短跳转，跳转范围：-128 ~ +127个单元
- ✓ 偏移量为16位，称为近跳转，跳转范围：-32768 ~ +32767个单元

例：JMP 1000H ;转移地址的偏移量由指令给出

2. 段内间接转移方式 — 转移地址在寄存器或内存单元中

例：JMP CX ;转移地址由CX给出，执行时将CX值赋给IP

JMP [1000H] ;转移地址存放在 (1000H, 1001H) 单元中

3. 段间直接转移方式 — 指令中直接给出转移地址的段地址和偏移量

例：JMP 2000H:0100H ;执行时，2000H赋给CS，0100H赋给IP

注：段间直间转移又称作远跳转，该种方式为程序提供了从一个代码段转移到另一个代码段的方法

4. 段间间接转移方式 — 转移地址的段地址和偏移量在内存单元中

例：JMP DWORD PTR [SI]

(SI, SI + 1)存放偏移地址，(SI + 2, SI + 3)存放段地址

(一) 无条件转移指令和条件转移指令

A. 无条件转移指令

1. JMP TARGET

无条件直接转移指令可以跳转到内存中任何程序段，有四种形式

例：JMP 1000H ;段内直接转移

JMP CX ;段内间接转移

JMP 2000H:0100H ;段间直接转移

JMP DWORD PTR [SI] ;段间间接转移

说明：段内间接转移和段间转移只用于无条件指令

实际使用时多采用符号地址，如：

JMP BBB

.....

BBB: MOV AL, 3

.....

注：符号BBB在此称为标号，代表其
对应的指令所在的地址

B. 条件转移指令

根据判断条件，决定是否跳转，条件转移指令都是段内短跳转

1. 根据单个标志进行跳转的指令

- ✓ JS ;SF = 1, 则跳转
- ✓ JNS ;SF = 0, 则跳转
- ✓ JO ;OF = 1, 则跳转
- ✓ JNO ;OF = 0, 则跳转
- ✓ JP ;PF = 1, 则跳转
- ✓ JNP ;PF = 0, 则跳转
- ✓ JZ ;ZF = 1, 则跳转
- ✓ JNZ ;ZF = 0, 则跳转
- ✓ JC ;CF = 1, 则跳转
- ✓ JNC ;CF = 0, 则跳转

2. 根据CX中的值进行跳转的指令

- JCXZ ;CX = 0, 则跳转



3. 根据无符号数比较结果进行跳转的指令

助记符: A: 大于; B: 小于; E: 等于

- ✓ JA (JNBE) ;大于, 即不小于等于, 则跳转
- ✓ JAE (JNB) ;大于等于, 即不小于, 则跳转, 同JNC
- ✓ JE ;等于, 即结果为0, 则跳转, 同JZ
- ✓ JB (JNAE) ;小于, 即不大于等于, 则跳转, 同JC
- ✓ JBE (JNA) ;小于等于, 即不大于, 则跳转, $CF \mid ZF = 1$

4. 根据有符号数比较结果进行跳转的指令

助记符: G: 大于; L: 小于; E: 等于

- ✓ JG (JNLE) ;大于, 即不小于等于, 则跳转
- ✓ JGE (JNL) ;大于等于, 即不小于, 则跳转
- ✓ JE ;等于, 与无符号数比较时相同
- ✓ JL (JNGE) ;小于, 即不大于等于, 则跳转
- ✓ JLE (JNG) ;小于等于, 即不大于, 则跳转

注: 根据比较结果跳转时, 必须区分有符号数和无符号数, 如:

1111 1111b 与 0000 0000b比较

(二) 子程序调用和返回指令

1. CALL TARGET

子程序调用指令类似转移指令，也有4种形式

- ✓ CALL 1000H ;段内直接转移，调用地址偏移量在指令中给出
- ✓ CALL AX ;段内间接转移，调用地址在AX中给出
- ✓ CALL 2000H:0200H ;段间直接转移，指令中直接给出调用地址的段地址和偏移量
- ✓ CALL DWORD PTR [DI] ;段间间接转移，调用地址在内存单元中

执行该指令时，会将下一条指令的地址压入堆栈，该地址称为返回地址

- 段内调用：只将返回地址的偏移量压入堆栈
- 段间调用：将返回地址的段地址和偏移量压入堆栈，先压CS，后压IP

实际使用时CALL指令后面直接跟子程序名，例：

```
CALL    DISP      DISP proc
BBB: MOV    AL, 3      .....
        .....      ret
                        DISP endp
```

注：执行CALL时，会将BBB所代表的地址压入堆栈

2. RET — 子程序返回指令，应与调用指令相对应

执行时，从堆栈顶部弹出返回地址

- ✓ 段内返回：仅从堆栈顶部弹出返回地址偏移量

$$IP \leftarrow (SP), \quad SP \leftarrow SP + 2$$

- ✓ 段间返回：从堆栈顶部弹出返回地址的段地址和偏移量

$$IP \leftarrow (SP), \quad SP \leftarrow SP + 2; \quad CS \leftarrow (SP), \quad SP \leftarrow SP + 2$$

3. RET IMM16 — 带立即数返回指令

执行时，从堆栈顶部弹出返回地址，再使SP加上立即数，例：RET 4

(三) 循环控制指令

循环控制指令均是段内短跳转

1. **LOOP TARGET** — 循环指令，循环次数由CX决定

执行时，CX先减1，若为0则退出，否则继续循环，例：

```
        MOV     CX, 10          ;设置循环次数为10次
BBB:    LOOP    BBB            ;CX减1，若不为0，则循环
        .....                ;后续处理
```

2. **LOOPZ / LOOPE TARGET** — 条件循环指令

LOOPZ 和LOOPE是同一条指令的不同助记符，执行时，在LOOP指令的基础上，还需判断ZF，若CX != 0 且 ZF = 1则循环

3. **LOOPNZ / LOOPNE TARGET**

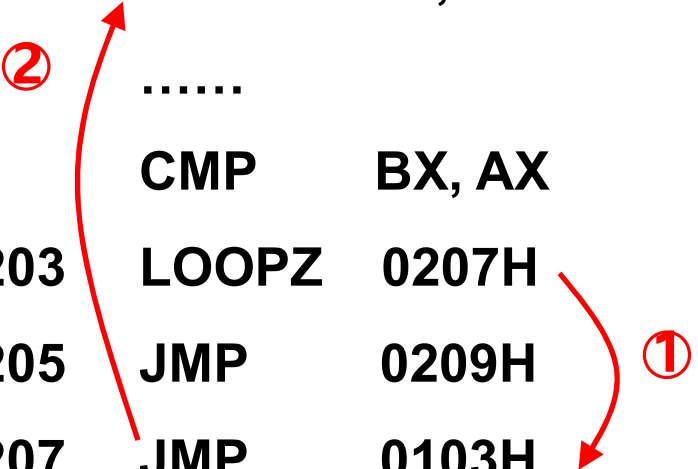
执行时，在LOOP指令的基础上，还需判断ZF

若CX != 0 且 ZF = 0则循环

※ 循环调用指令说明

- ✓ 循环指令不影响状态标志，即CX减为0时不影响FLAG
- ✓ 如果目标位置超出短跳转的范围，则用JMP辅助完成循环

例: 126C: 0100 MOV CX, 0010H
 126C: 0103 MOV AX, CX
 ② ;循环体中程序
 CMP BX, AX
 126C: 0203 LOOPZ 0207H
 126C: 0205 JMP 0209H ①
 126C: 0207 JMP 0103H
 126C: 0209 ;循环体以后的程序



(五) 中断指令和中断返回指令

1. INT n — 中断调用指令

8086指令系统为程序员提供软件中断，利用INT n指令实现

执行时，依次将FLAG，当前CS和IP（即断点地址）压入堆栈，并清除IF和TF标志，执行中断类型码为n的中断程序。

2. IRET — 中断返回指令

执行时，从堆栈中依次弹出断点地址的IP、CS和FLAG

3. INTO — 溢出中断指令

执行时，判断OF，若为1，则执行中断指令INT 4，否则无操作

(六) 控制指令

1. 标志处理指令

- | | |
|-------------------|-----------------|
| ✓ CLC / STC / CMC | 对CF 清0 / 置位 /取反 |
| ✓ CLD / STD | 对DF 清0 / 置位 |
| ✓ CLI / STI | 对IF 清0 / 置位 |

2. 处理器控制指令

- **NOP** — 空操作指令，占用一个字节的机器码，不执行任何操作
- **HLT** — Halt，停机指令，该指令使处理器处于停机状态，以便等待一次外部中断到来
- **WAIT**— 等待指令，该指令使处理器处于空转状态，也可用来等待外部中断的到来
- **ESC** — Escape，换码指令，用作前缀，例：ESC MEM

其中MEM指出一个存储单元，ESC指令把该存储单元的内容送到数据总线去。

- **LOCK** — 封锁指令，用作前缀

该指令与其他指令联合，用来维持总线的封锁信号直到与其联合的指令执行完为止。

第3.4节 80386扩充指令介绍

80386为32位CPU，具有独立的32位数据线(D31~D0)和34位地址线(A31~A2, /BE0~/BE3)，可寻址4GB存储空间，内部寄存器多为32位

一. 80386的内部寄存器

1. 通用寄存器

80386有8个32位通用寄存器：

EAX EBX ECX EDX ESP EBP ESI EDI

为与8086兼容，每个寄存器低16位可独立访问：

AX BX CX DX SP BP SI DI

前4个寄存器的低8位和高8位又可独立访问

AH AL BH BL CH CL DH DL

2. 32位指令指针寄存器EIP

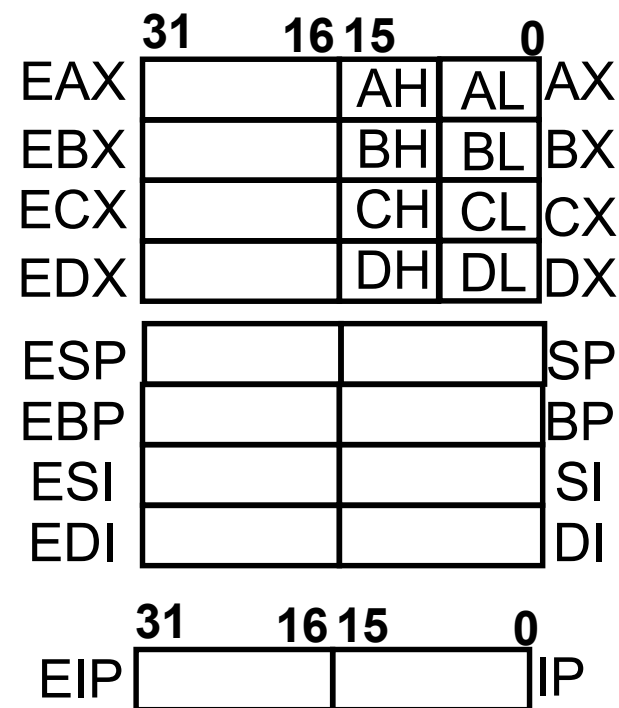
低16位取名IP，可独立访问

3. 32位标志寄存器EFLAG

增加了部分标志

EFLAG

31		0
----	--	---



4. 段寄存器：80386内部设置了6个16位段寄存器CS DS SS ES FS GS

二. 80386寻址方式

在8086寻址方式基础上，80386的所有通用寄存器既可作为基址寄存器，也可作为变址寄存器(ESP除外)，存放地址。

$EA = \text{基址} + \text{变址} \times \text{比例因子} + \text{位移量}$

比例因子可以为：1、2、4或8，位移量可以为8位、16位或32位

例：MOV ECX, [EAX + EDX * 4 + 1000H]

三. 80386扩充指令 — 1.数据传送指令

1) 使用MOV指令时，两操作数位数必须相同，若不同，则用**MOVZX**或**MOVSX**

MOVZX AX, BL ;若BL=80H，则AX=0080H，MOVZX表示零扩展BL

MOVSX AX, BL ;若BL=80H，则AX=FF80H，MOVSX表示符号扩展BL

2) 立即数入栈指令 例：**PUSH 1000H**

3) **PUSHA** 将全部16位寄存器AX~DI压入堆栈，**POPA** 则全部出栈

- 4) **LSS** ESP, MEM ;将MEM开始的存储单元内容送ESP和SS
LFS EDX, MEM ;将MEM开始的存储单元内容送EDX和FS
LGS EAX, MEM ;将MEM开始的存储单元内容送EAX和GS
- 5) **CWDE** ;将AX扩展为EAX
CDQ ;将EAX扩展为(EDX, EAX)

2. 算数运算指令扩充

1) **IMUL SRC**

$AX \leftarrow AL * SRC$;字节乘法

$EAX \leftarrow AX * SRC$;字乘法, 高16位同时也放在DX中, 兼容8086

$(EDX, EAX) \leftarrow EAX * SRC$;32位数乘法

另外操作数可为2个8位数、2个16位数、2个32位数, 例:

IMUL DX, [1000H], 300 ;DX = (1000H) * 300

IMUL ECX, 2000 ; ECX = ECX * 2000

IMUL BX, CX ;BX = BX * CX

2) **MUL SRC** ;与IMUL指令类似



3) **DIV与IDIV指令** — 可用AX, (DX, AX), (EDX, EAX)作为被除数, 例:

DIV BL ;AL = AX / BL, AH = AX % BL

DIV BX ;AX = (DX, AX) / BX, DX = (DX, AX) % BX

DIV ECX ;EAX = (EDX, EAX) / ECX, EDX = (EDX, EAX) % ECX

3. 逻辑指令扩充

1) 逻辑操作与8086一样, 只是逻辑指令还可用于32位操作数

2) 增加了2条可对64位操作数进行移位的指令SHLD和SHRD

SHLD EDX, EAX, 5 ;EDX左移5位, 低5位由EAX的高5位填充
;且EAX中内容不变

SHRD EAX, EBX, CL ;EAX右移CL位, EAX高位由EBX的低位
;填充, EBX的值保持不变

4. 串操作指令扩充

1) 增加了32位数的串操作 例: **MOVSD** ;表示传送一个双字

2) 增加字符串输入**INS**和字符串输出**OUTS**指令, 必须使用DX存放端口号

MOV DX, 160H ;从160H端口连续读10个双字存放到ES:EDI指向的存储单元

MOV CX, 10

REP INSD

5. 转移、循环和调用指令

- 1) **JECXZ** TARGET ;若ECX = 0, 则跳转
- 2) 使用CALL指令时 段内调用: 将EIP压入堆栈, 保存4个字节
段间调用: 将CS:EIP共6字节压入堆栈
- 3) RET和RET n指令 出栈时, 操作应与CALL指令相对应

6. 条件设置指令 — 80386新增指令

格式: **SETcond** REG8/MEM8 ;cond为设置条件

例: **SETZ** AL ;若ZF = 1, 则置AL为1, 否则置0

SETGE CL ;有符号数比较大小时, 若大于等于, 则置CL为1

SETO [100H] ;若OF = 1, 则置[1000H]存储单元为1, 否则置0

7. 位处理指令 — 80386新增指令

BTS ;将指定位置1 **BTR** ;将指定位清0

BTC ;对指定位取反 **BT** ;对指定位进行测试

BSF ;从最低位往最高位扫描, 若全为0, 则ZF置1, 若某位为1, 则
;ZF清0, 且将此位的位号放入目的寄存器

BSR ;从最高位往最低位扫描, 若全为0, 则ZF置1, 若某位为1, 则
;ZF清0, 且将此位的位号放入目的寄存器

例: BTC AX, 2 ;将AX的D2位装入CF, 再对AX的D2位取反
 BTS [100H], 4 ;将[100H]单元的D4位装入CF, 再将D4位置1
 BTR BL, CL ;将CL的值作为位的序号, 将BL的此位送CF,
 ;再将BL的此位清0, 若CL的值大于8, 则将CL
 ;的值对8取余得到位序号
 BT AL, CL ;将CL的值作为序号, 对AL的此位进行测试
 BSF AX, [2000H] ;对[2000H]开始的一个双字, 从D0往D31
 ;扫描, 若全为0, 则ZF = 1, 否则ZF = 0,
 ;且将此位序号送AX
 BSR EAX, ECX ;对ECX从D31往D0扫描, 若全为0, 则置
 ;ZF=1, 否则ZF=0, 并将此位序号送EAX