



河南大學  
Henan University

# 汇编语言与接口技术

## ——第 3 章 80x86 指令系统和寻址方式

---

主讲教师：舒高峰

电子邮箱：gaofeng.shu@henu.edu.cn

联系电话：13161693313

# 00 | 上节回顾-CF 和 OF 的区分

## 简答题

- CF 和 OF 分别是什么？
- 它们的区别是什么？
- 判断**无符号数**溢出的条件是什么？
- 判断**有符号数**溢出的条件是什么？

# 目录

---

- 01 80x86 指令系统概述
- 02 8086 的寻址方式
- 03 8086 的常用指令
- 04 80386 的寻址方式和指令系统

# 01 | 指令系统概述-相关概念

## 机器指令

- CPU 能直接识别并执行的由 0、1 表示的二进制代码
- 指令由计算机的电器元件状态来体现，为译码器所识别，并经一定时间周期进行执行

## 指令构成

- **机器指令 = 操作码 + 操作数**
  - 操作码：计算机要完成什么操作 (停机中断？加减乘除？)
  - 操作数：参与操作的对象是什么 (0个？1个？)
- 80x86 机器指令使用 1-6B 的变长指令

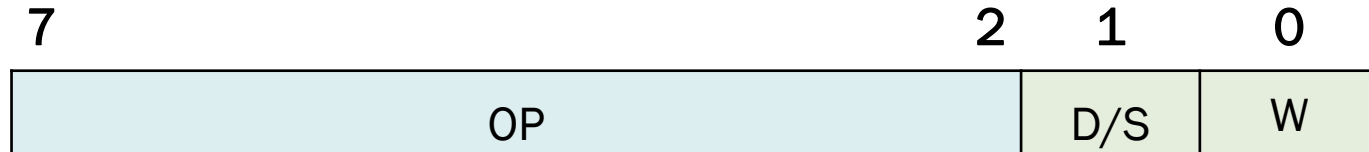
## 寻址方式

- 计算机根据指令信息**求出操作数的有效地址的方法**

# 01 | 指令系统概述-指令操作码

## 指令操作码 (Operate Code)

- 一般为指令的第一个字节，用二进制代码表示的**本指令所执行的具体操作**，常用某位某些具体操作信息



MOV:100010

ADD:000000

DIV:111101

## 特征位含义

- **OP**: 由 6 位二进制代码表示不同的操作码
- **D/S**: D=1 表示寄存器寻址的是目标操作数，D=0 表示寄存器所寻址的是源操作数；S 是符号扩展位
- **W**: W=1 表示当前指令进行字操作，W=0 表示当前指令进行字节操作

# 01 | 指令系统概述-操作数

## 寻址方式

- 一般为指令的第二个字节，表示执行指令时所用操作数的来源

## 操作数

- 一般为指令的第三至第六个字节，表示指令的操作对象
  - 一条指令可以包含 1 个操作数，也可以包含多个操作数，也可能由指令隐含地指出
  - 若位移量或立即数为 16 位，则在指令代码中，**低位字节放在前面，高位字节放在后面**
  - **单字节指令**：由一个字节构成，既包含操作码，也隐含寄存器名

# 01 指令系统概述-指令格式

## 指令格式

操作码	一字节指令 (隐含操作数)
-----	------------------

操作码	寻址方式	二字节指令
-----	------	-------

操作码	寻址方式	数据	三字节指令
-----	------	----	-------

操作码	寻址方式	数据/偏移 (低)	数据/偏移 (高)	四字节指令
-----	------	--------------	--------------	-------

操作码	寻址方式	偏移 (低)	偏移 (高)	数据	五字节指令
-----	------	--------	--------	----	-------

操作码	寻址方式	偏移 (低)	偏移 (高)	数据 (低)	数据 (高)	六字节指令
-----	------	--------	--------	--------	--------	-------

# 01 | 指令系统概述-指令举例1

## 指令举例

7	2	1	0	7	6	5	4	3	2	1	0	偏移 (低)	偏移 (高)	数据 (低)	数据 (高)
OPCODE				D	W	MOD		REG		R/M					

- 第一字节：**操作码**。D 表示指令中 REG 部分是否为目的操作数，W 表示操作数是字或字节数据
- 第二字节：MOD 表示**寻址方式**，REG 表示**所访问的寄存器**，R/M 表示**内存或寄存器**
- 后续字节：表示**地址偏移**或**数据**内容

MOD R/M, REG	存储器寻址 有效地址的计算公式			寄存器寻址	
				W=0	W=1
	MOD=00	MOD=01	MOD=10	MOD=11	
000	BX+SI	BX+SI+disp8	BX+SI+disp16	AL	AX
001	BX+DI	BX+DI+disp8	BX+DI+disp16	CL	CX
010	BP+SI	BP+SI+disp8	BP+SI+disp16	DL	DX
011	BP+DI	BP+DI+disp8	BP+DI+disp16	BL	BX
100	SI	SI+disp8	SI+disp16	AH	SP
101	DI	DI+disp8	DI+disp16	CH	BP
110	DIRECT	BP+disp8	BP+disp16	DH	SI
111	BX	BX+disp8	BX+disp16	BH	DI



# 01 | 指令系统概述-指令举例2

## 指令举例

7	2	1	0	7	6	5	4	3	2	1	0	偏移 (低)	偏移 (高)	数据 (低)	数据 (高)
OPCODE		D	W	MOD		REG			R/M						

- 第一字节：存储器到寄存器传送指令码 **100010DW**
- 第二字节：查表可得  
**MOD=10**，**R/M=001**，  
**REG=000**
- 后续字节：16 位偏移，低位字节在前，高位字节在后

MOD R/M, REG	存储器寻址 有效地址的计算公式			寄存器寻址	
				W=0	W=1
	MOD=00	MOD=01	MOD=10	MOD=11	
000	BX+SI	BX+SI+disp8	BX+SI+disp16	AL	AX
001	BX+DI	BX+DI+disp8	BX+DI+disp16	CL	CX
010	BP+SI	BP+SI+disp8	BP+SI+disp16	DL	DX
011	BP+DI	BP+DI+disp8	BP+DI+disp16	BL	BX
100	SI	SI+disp8	SI+disp16	AH	SP
101	DI	DI+disp8	DI+disp16	CH	BP
110	DIRECT	BP+disp8	BP+disp16	DH	SI
111	BX	BX+disp8	BX+disp16	BH	DI

示例: **MOV AX, [BX+DI+1122H]**

073F:0100 8B812211 MOV AX,[BX+DI+1122]

1000	1011	1000	0001	0010	0010	0001	0001
8	B	8	1	2	2	1	1

# 01 | 指令系统概述-指令的书写格式

## 指令书写格式

- [标号:] 助记符 [操作数] [;注释]
  - 标号与助记符用冒号分隔，表示本条指令的存放位置
  - 标号不是每条指令中都需要，必要时定义
  - 16 位系统中，汇编指令的操作数一般为 0-2 个

## 伪指令书写格式

- [符号名] 伪指令助记符 [操作数] [;注释]
  - 符号名与伪指令之间不需要冒号分隔
  - 伪指令的操作数至少 1 个，多个时用逗号分隔

# 01 | 指令系统概述-指令示例

## 数据传送指令

[标号:] 助记符 [操作数] [;注释]

- MOV <DST>, <SRC>
  - MOV AX, 2
  - MOV AX, BX
  - MOV AX, [BX]

## 指令功能

- 使用源操作数的值为目的操作数赋值
- 指令执行完，目的操作数的值与源操作数一样
  - 源操作数不变，目的操作数改变
- 单独写的指令一般都不包括标号部分
  - 标号在写指令序列时才会使用到

# 01 | 指令系统概述-约定符号说明

## 约定符号

符号	含义	符号	含义
DST	目的操作数 (destination)	MEM8/16	8/16 位存储单元
SRC	源操作数 (source)	IMM8/16	8/16 位立即数
REG	通用寄存器 (register)	REG8/16	8/16 位通用寄存器
SEG	段寄存器 (segment)	OPRD	操作数 (operand)
MEM	存储单元 (memory)	/	或者
IMM	立即数 (immediate)	=	赋值
DISP	偏移量 (displacement)		

# 01 | 指令系统概述-汇编源程序基本格式

## 汇编语言程序基本格式

```
DATA SEGMENT
    A DW M
    BUF DB 0dH, 0aH ;数据段
DATA ENDS

CODE SEGMENT
    ASSUME DS:DATA, CS:CODE
START:
    MOV AX, DATA
    MOV DS, AX
    .....
    MOV AX, 4C00H
    INT 21H ;代码段
CODE ENDS
END START
```

[标号] 伪指令助记符 [操作数] [;注释]

[标号:] 助记符 [操作数] [;注释]

# 目录

---

- 01 80x86 指令系统概述
- 02 8086 的寻址方式**
- 03 8086 的常用指令
- 04 80386 的寻址方式和指令系统

# 02 | 8086寻址方式-寻址方式类型

## 寻址方式类型

- **数据寻址方式**：获取指令所需的**操作数**或**操作数地址**的方式
- **程序寻址方式**：在程序中出现**转移**和**调用**时的程序定位方式
- **端口寻址方式**：获取指令所需的**外部端口地址**的方式

# 02 | 8086寻址方式-数据寻址方式

## 数据寻址方式

- 指令中，指定**操作数或操作数存放位置**的方法
- **数据寻址方式是针对操作数的**，确定寻址方式时，一定要指明是对源操作数、还是对目的操作数而言的。

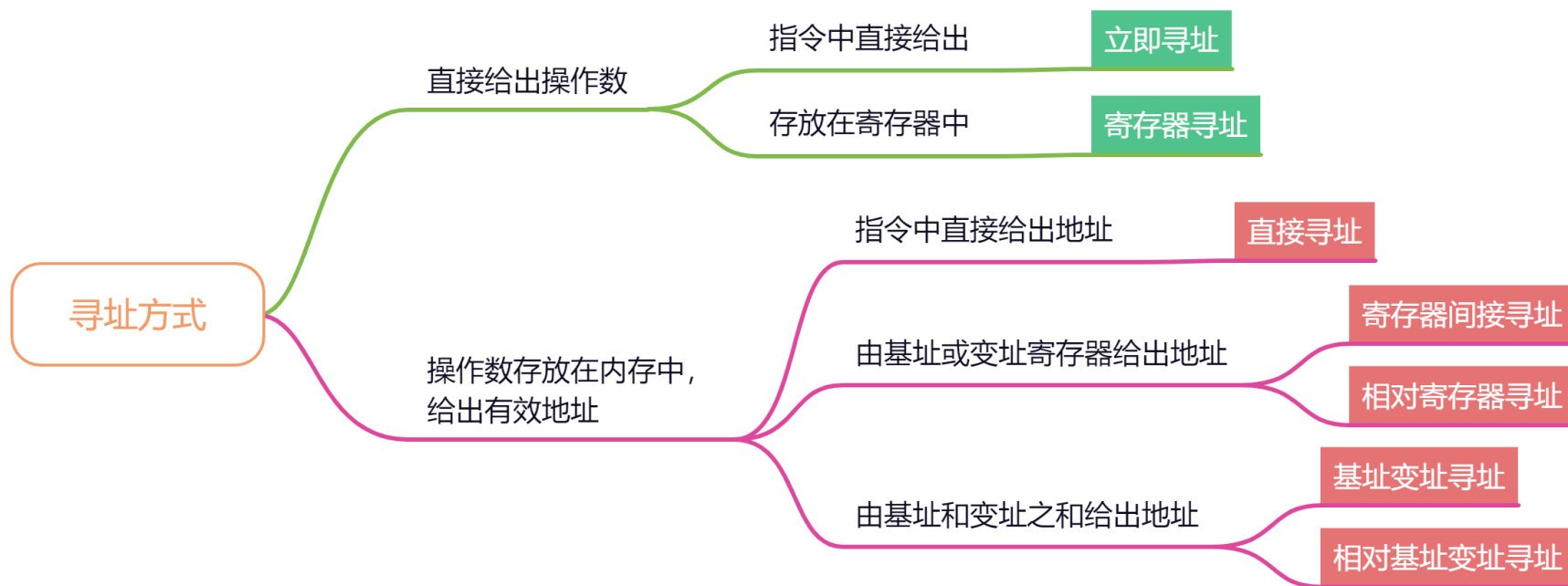
## 数据寻址方式分类

- 操作数在指令中：立即数寻址方式
  - 操作数在寄存器中：寄存器寻址方式
  - 操作数在存储单元中：存储器寻址方式 5 种
- 指令执行时，  
操作数都在  
CPU内部！



# 02 | 8086寻址方式-数据寻址方式

## 数据寻址方式分类



存储器寻址方式有（ ）种。

☐ A 2

☐ B 3

☒ C 5

☐ D 7

# 02 | 8086寻址方式-立即寻址

## 立即数寻址

- 操作数的数值紧跟在操作码之后，**直接在指令中出现**

```
MOV AX, 0A7FH
```

## 特点

- 优点：不需访存，**执行速度快**；
- 缺点：立即数不能修改，**通用性差**（适用于常数）

## 注意

- 立即数**不能作为目的操作数**
- 立即数可以为 8 位、16 位或 32 位，常用于给寄存器赋值，但**不能直接为段寄存器赋值**

# 02 | 8086寻址方式-寄存器寻址

## 寄存器寻址

- 操作数在**寄存器**中，指令的操作码之后给出该寄存器的编号或名称

```
MOV AX, BX
```

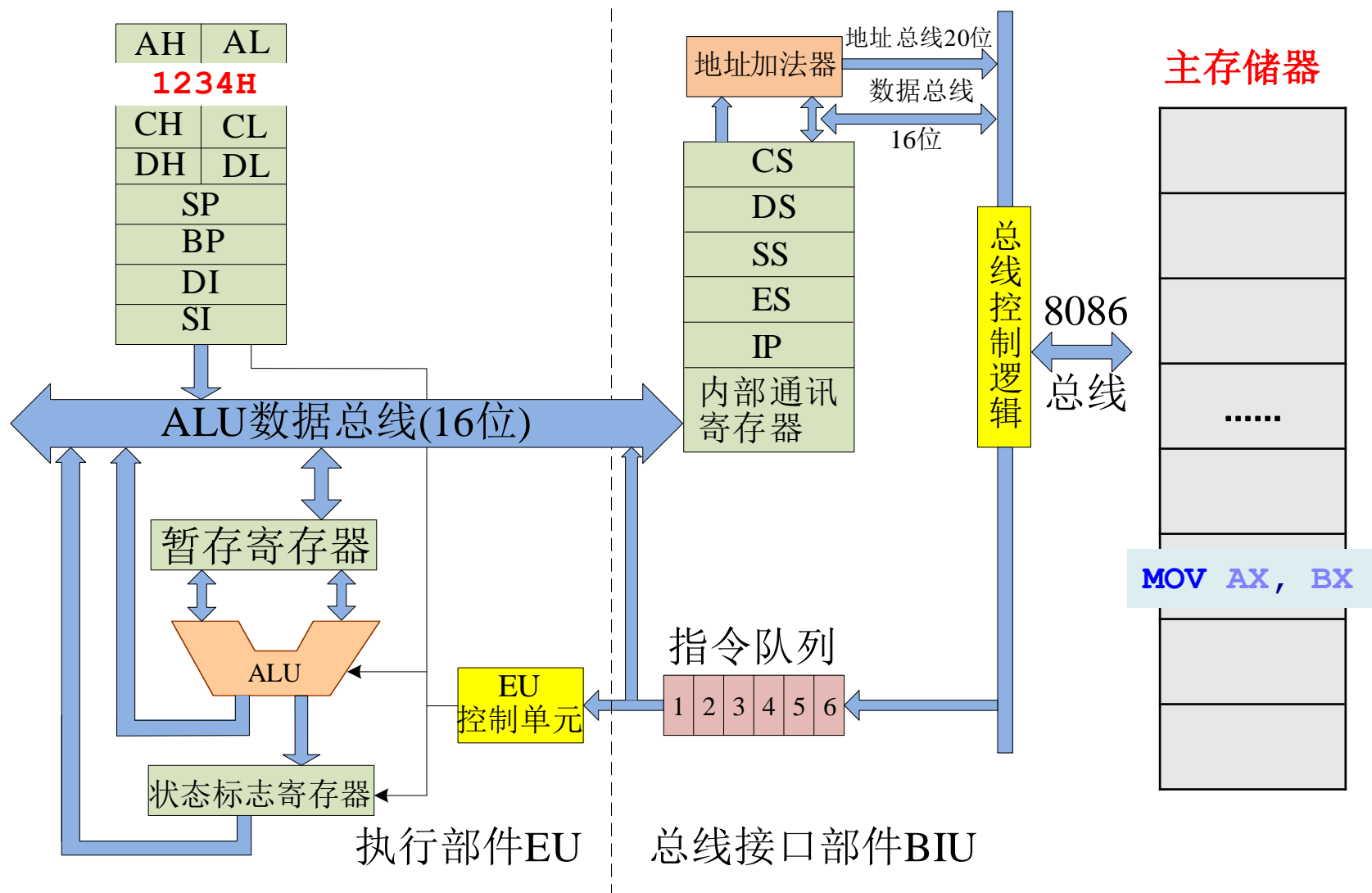
## 特点

- 由于操作数存在于 CPU 中，指令执行时不需访存，因此具有**较快的执行速度**

## 注意

- 通用寄存器和段寄存器都可以用作寄存器寻址方式
- **专用寄存器 — IP、FLAG 不可用**

# 02 | 8086寻址方式-寄存器寻址示意



# 02 | 8086寻址方式-直接寻址

## 直接寻址

- 操作数在**存储单元**中，指令的操作码之后给出该存储单元的有效地址（EA）  
`MOV AX, [2000H]`

## 注意

- 指令中，EA 可以是**数值形式**，也可以是**符号地址形式**

`MOV AX, [1000H]` ; 数值地址必须加方括号

`MOV AX, X` ; 符号地址 X 必须事先定义

`MOV [1000H], DX` ; 目的操作数采用直接寻址

- 在**默认**情况下，数据在 DS 段中
  - 物理地址  $PA = (DS) \times 16 + EA$
  - 可使用段超越前缀的方式改变默认段

`MOV AX, ES: [100H]`

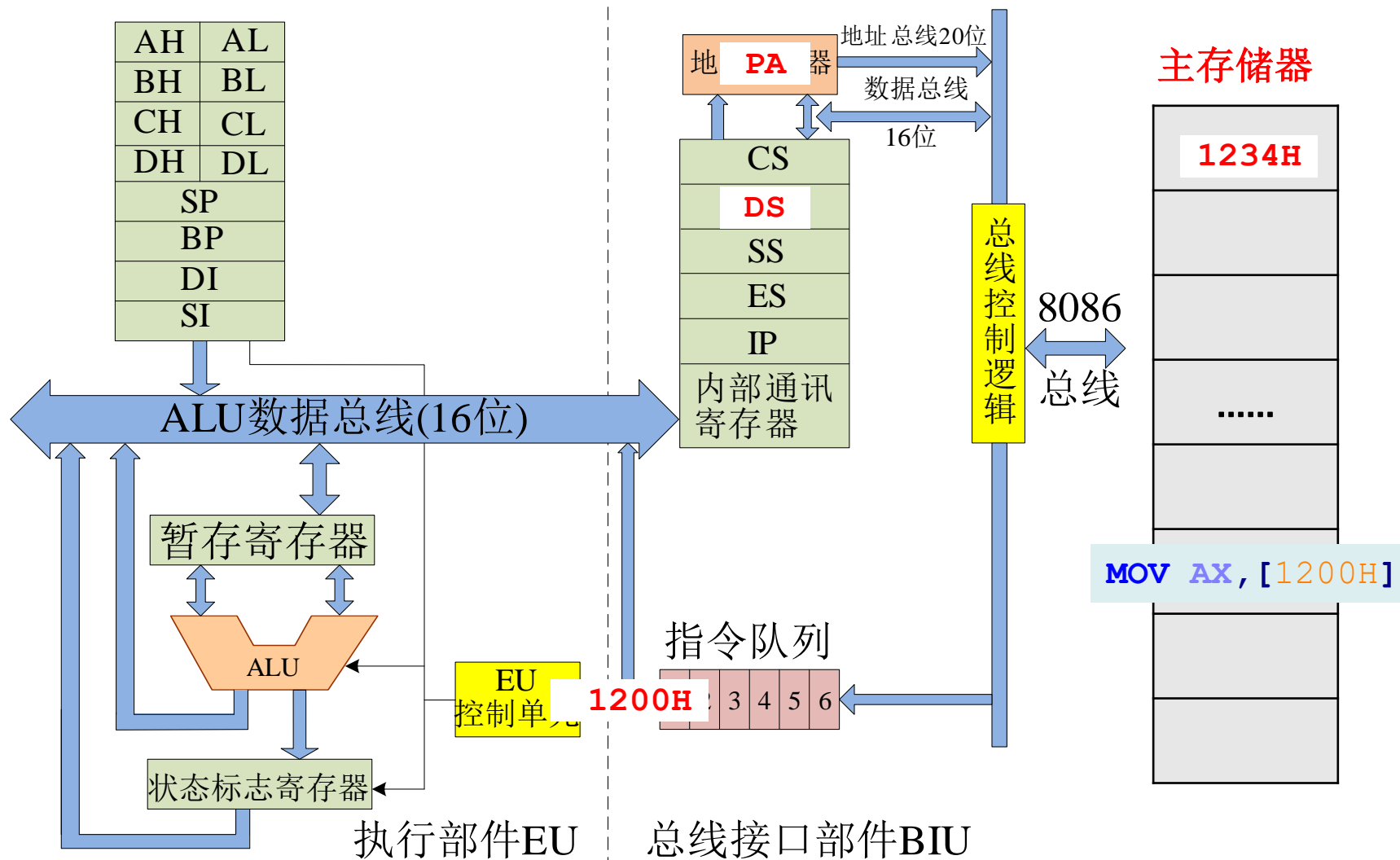
## 02 | 8086寻址方式-直接寻址举例

### 举例

- 执行指令 `MOV BX, [1234H]` 时,  $(DS)=2000H$ , 存储单元 `21234H` 的值为 `5213H`, 问执行该指令后 `BX` 的值是什么?
  - 该指令源操作数的寻址方式为**直接寻址方式**
  - 有效地址  $EA = 1234H$ , 默认为 `DS` 段
  - 物理地址  $PA = 2000H \times 16 + 1234H = 21234H$
  - 所以, 该指令执行后  **$(BX)=5213H$**

	.....
21234H	13H
21235H	52H
	.....

# 02 | 8086寻址方式-直接寻址示意





# 02 | 8086寻址方式-思考问题

## 问题 1

- 右边所示的三条指令有何不同？

- 源操作数的寻址方式不同：第一条指令为直接寻址方式；后两条指令为立即数寻址方式；
- 数据数制不同：前两条为十六进制；第三条为十进制。

```
1. MOV AX, [2000H]
2. MOV AX, 2000H
3. MOV AX, 2000
```

## 问题 2

- `MOV AX, X` 和 `MOV AX, [X]` 效果一样吗？

- 一样。符号地址加不加括号指的都是之前定义的单元

## 问题 3

- 高级语言的  $y=x$ ; 用汇编指令如何实现？

- 需要使用两条指令完成赋值

```
MOV AX, X
MOV Y, AX
```

# 02 | 8086寻址方式-寄存器间接寻址

## 寄存器间接寻址

- 操作数在**存储单元**中，指令的操作码之后给出存放该单元有效地址的**寄存器**编码或名称

```
MOV AX, [BX]  
MOV [BP], BX
```

## 注意

- 可以用于这种寻址方式的寄存器称为**间址寄存器**
  - **间址寄存器**: **BX**、**BP**、**SI**、**DI**
- **默认**情况下，数据在 DS 段或 SS 段中，由**间址寄存器**决定
  - 物理地址  $PA = (DS) \times 16 + (BX/SI/DI)$  ; DS 段
  - 物理地址  $PA = (SS) \times 16 + (BP)$  ; SS 段

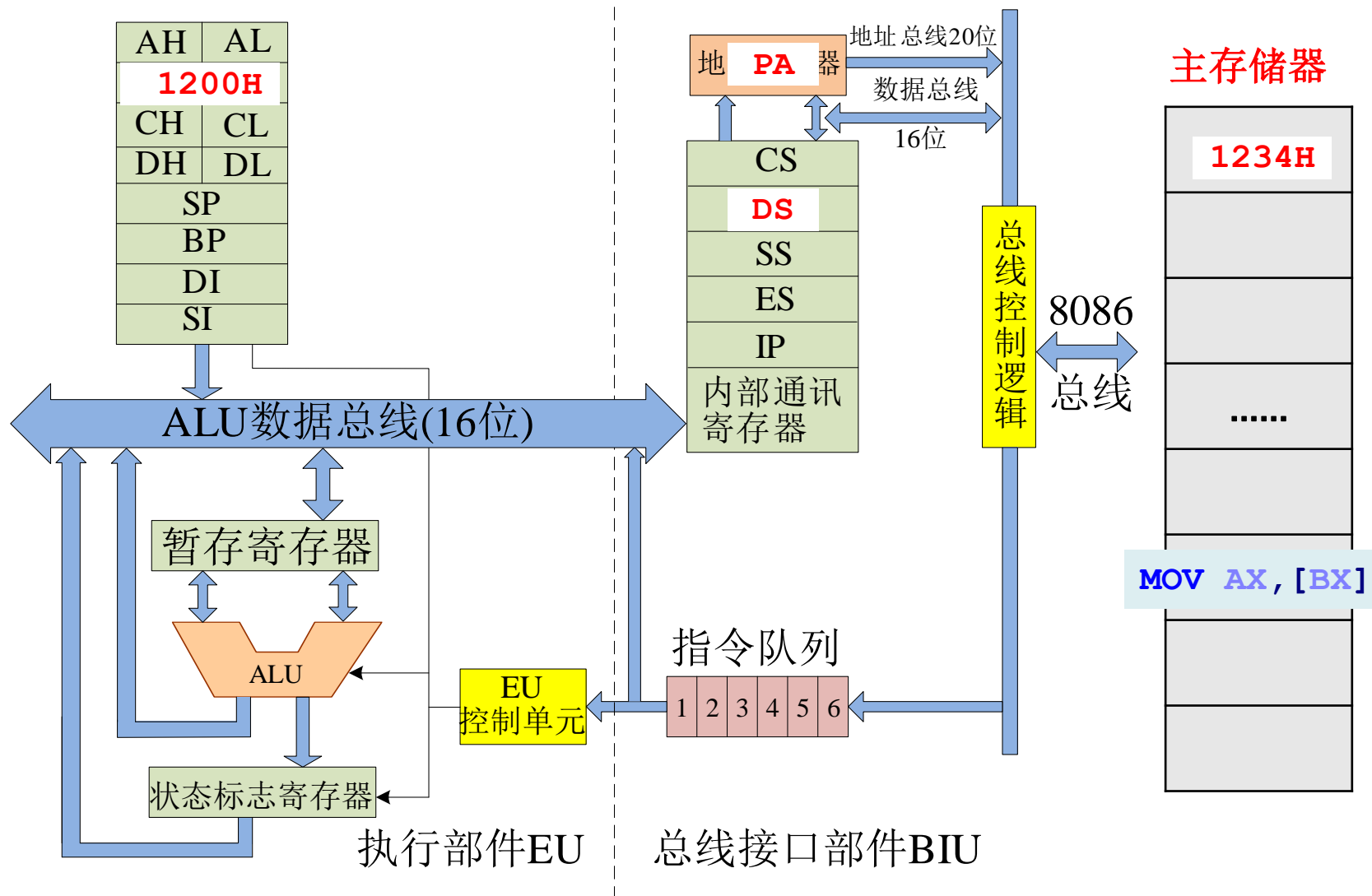
## 02 | 8086寻址方式-寄存器间接寻址举例

### 举例

- 执行指令 `MOV BX, [DI]` 时,  $(DS)=1000H$ ,  $(DI)=2345H$ , 存储单元 `12345H` 的值为 `4354H`, 则指令执行后 `BX` 的值为多少?
  - 该指令源操作数的寻址方式为**寄存器间接寻址**
  - 有效地址  $EA = (DI) = 2345H$ , 默认为 `DS` 段
  - 物理地址  $PA = (DS) \times 16 + EA$   
$$= 1000H \times 16 + 2345H = 12345H$$
  - 所以, 该指令执行后  **$(BX) = [12345H] = 4354H$**

	.....
12345H	54H
12346H	43H
	.....

# 02 | 8086寻址方式-寄存器间接寻址示意



# 02 | 8086寻址方式-寄存器相对寻址

## 寄存器相对寻址

- 操作数在**存储单元**中，操作数的有效地址为指令中指定**间址寄存器**的值与指令中给出的**偏移量 (DISP)** 之和

```
MOV AX, 3003H[SI]
MOV AL, [BP+08H]
MOV BP, [BX][100H]
```

## 注意

- 指令格式
  - 间址寄存器**: BX、BP、SI、DI
  - 偏移量**: 数值形式、符号地址形式
- 物理地址的形成与寄存器间接寻址相似
  - $PA = (DS) \times 16 + (BX/SI/DI) + DISP$  ;DS 段
  - $PA = (SS) \times 16 + (BP) + DISP$  ;SS 段

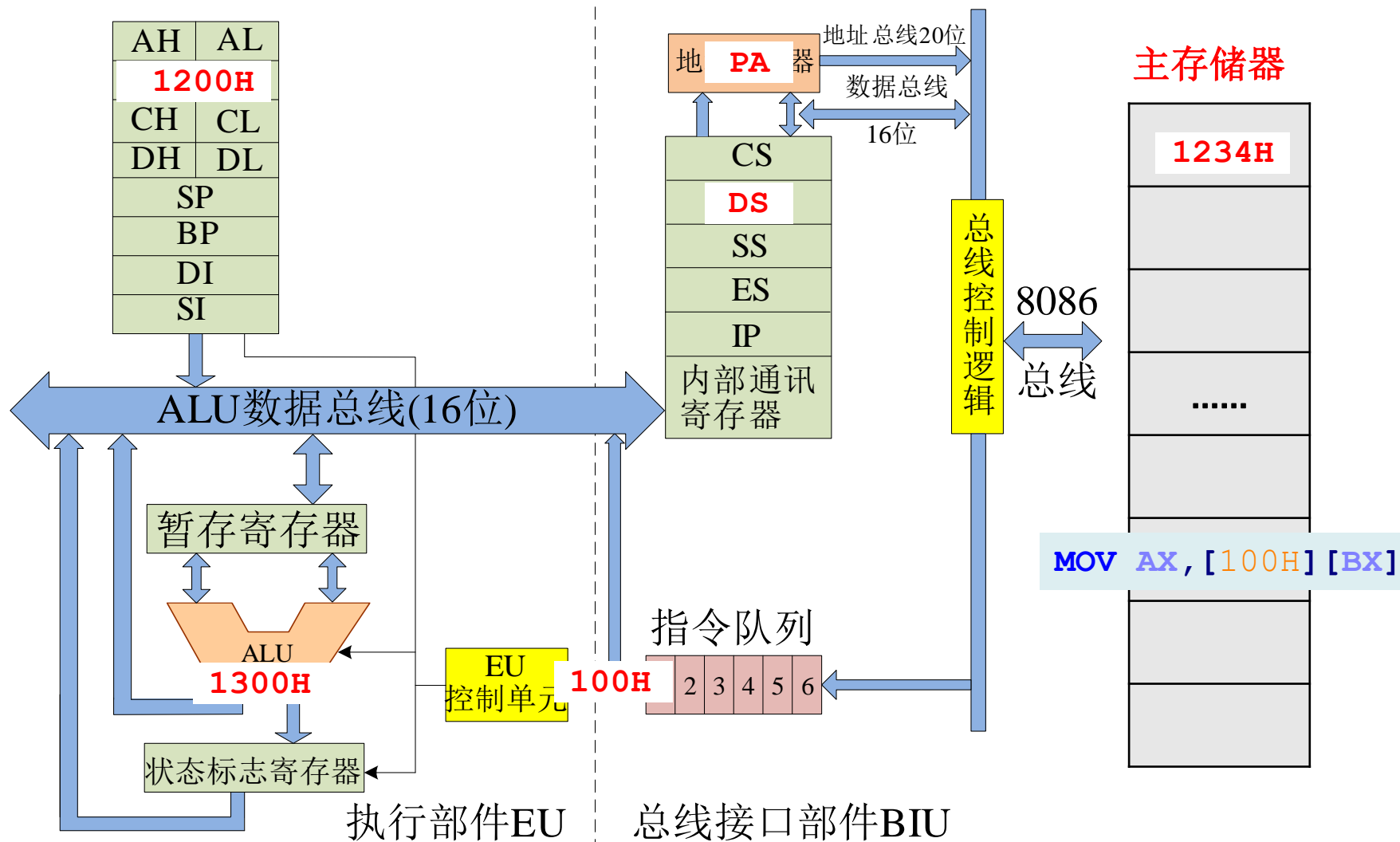
## 02 | 8086寻址方式-寄存器相对寻址举例

### 举例

- 执行指令 `MOV BX, [SI+100H]` 时,  $(DS)=1000H$ ,  $(SI)=2345H$ , 存储单元 `12445H` 的值为 `2715H`, 则指令执行后 `BX` 的值为多少?
  - 该指令源操作数的寻址方式为**寄存器相对寻址**
  - 有效地址  $EA = (SI) + 100H = 2445H$ , 默认为 `DS` 段
  - 物理地址  $PA = (DS) \times 16 + EA$   
 $= 1000H \times 16 + 2445H = 12445H$
  - 所以, 该指令执行后  **$(BX) = [12445H] = 2715H$**

	.....
12445H	15H
12446H	27H
	.....

# 02 | 8086寻址方式-寄存器相对寻址示意



# 02 | 8086寻址方式-基址变址寻址

## 基址变址寻址

- 操作数在**存储单元**中，操作数的有效地址为指令中指定的**基址寄存器**的值与**变址寄存器**的值之和

```
MOV AX, [BX][SI]  
MOV AX, [BP+SI]  
MOV AX, [BX]+[DI]
```

## 注意

- 指令格式
  - **基址寄存器**: BX、BP
  - **变址寄存器**: SI、DI
- 默认情况下，数据在 DS 段或 SS 段中，由**基址寄存器**决定
  - $PA = (DS) \times 16 + (BX) + (SI/DI)$  ; DS 段
  - $PA = (SS) \times 16 + (BP) + (SI/DI)$  ; SS 段



## 02 | 8086寻址方式-基址变址寻址举例

### 举例

- 执行指令 `MOV AX, [BX+SI]` 时,  $(DS)=1000H$ ,  $(BX)=2100H$ ,  $(SI)=0010H$ , 存储单元  $12110H$  的值为  $1234H$ , 则指令执行后  $AX$  的值为多少?
  - 该指令源操作数的寻址方式为**基址变址寻址**
  - 有效地址  $EA = (BX) + (SI) = 2110H$
  - 物理地址  $PA = (DS) \times 16 + EA$   
 $= 1000H \times 16 + 2110H = 12110H$
  - 所以, 该指令执行后  **$(AX) = [12110H] = 1234H$**

	.....
12110H	34H
12111H	12H
	.....

# 02 | 8086寻址方式-相对基址变址寻址

## 相对基址变址寻址

- 操作数在**存储单元**中，操作数的有效地址为指令中指定的**基址寄存器**的值、**变址寄存器**的值和**位移量 DISP**三者之和

```
MOV AX, 100H[BX][SI]
MOV AX, 100H[BP+SI]
MOV AX, [BX+DI+100H]
```

## 注意

- 指令格式
  - 基址寄存器：BX、BP；变址寄存器：SI、DI
  - 偏移量：数值形式、符号地址形式
- 默认情况下，数据在 DS 段或 SS 段中，由**基址寄存器**决定
  - $PA = (DS) \times 16 + (BX) + (SI/DI) + DISP$  ; DS 段
  - $PA = (SS) \times 16 + (BP) + (SI/DI) + DISP$  ; SS 段

# 02 | 8086寻址方式-相对基址变址寻址举例

## 举例

- 执行指令 `MOV AX, [BX+SI+200H]` 时,  $(DS)=1000H$ ,  $(BX)=2100H$ ,  $(SI)=0010H$ , 存储单元 `12310H` 的值为 `1234H`, 则指令执行后 `AX` 的值为多少?
  - 该指令源操作数的寻址方式为**相对基址变址寻址**
  - 有效地址  $EA = (BX) + (SI) + 200H = 2310H$
  - 物理地址  $PA = (DS) \times 16 + EA$   
 $= 1000H \times 16 + 2310H = 12310H$
  - 所以, 该指令执行后  **$(AX) = [12310H] = 1234H$**

	.....
12310H	34H
12311H	12H
	.....

# 02 | 8086寻址方式-总结1

## 三大类寻址方式

- 按操作数的存放位置分为：  
**立即数寻址**方式、**寄存器寻址**方式、**存储器寻址**方式

## 间址寄存器

- 间址寄存器又可分为**基址寄存器**（BX、BP）和**变址寄存器**（SI、DI）两类
  - 只有间址寄存器才可以出现在方括号里，**其他寄存器不行**
- 方括号“[]”中的寄存器只能是一个或两个间址寄存器，且**不能同时**为基址寄存器或变址寄存器

## 方括号 []

- 指令中凡是加有方括号“[]”的内容——**立即数或寄存器**，其值均表示为**地址信息**

■ 例如：MOV AX, [1000H]

MOV AX, [BX]

- 对于**符号地址**，则方括号可省略

## 02 | 8086寻址方式-总结2

### 有效地址计算

- 存储器寻址方式中，在取操作数之前需要**计算有效地址**
- 有效地址 EA 是将指令中**所有地址信息**相加之和
- 寻址方式中的偏移量有正有负（16位）
  - 可偏移范围为向上 32768 个单元，向下 32767 个单元
- 有效地址应为 16 位，多于 16 位应**按 64K 取模**
  - 例如，MOV AX, [BX+0FFFFH]，若 (BX)=100H，则  
$$EA = 0FFFFH + 100H = 100FFH \pmod{64K} = 0FFH$$

# 02 | 8086寻址方式-总结3

## 修改默认段

- 存储器寻址方式中，默认情况下均在 DS 段或 SS 段中
- 若需要寻址其他段，则可使用显式指明段寄存器的**段超越前缀**的方式来改变默认段寄存器
  - 指令形式：MOV **ES**: [100H], AL
  - 物理地址  $PA = (ES) \times 16 + 100H$
- 例如：MOV AX, **ES**: [BP+100H] 和 MOV AX, **SS**: [BX+100H]
- 指令 MOV AX, [100H] 与 MOV AX, SS: [100H] 执行结果是否相同？
  - **不一定相同**！所访问存储单元属于不同的逻辑段。

# 02 | 8086寻址方式-总结4

## 存储器寻址方式的通用形式

$[DISP] + [\text{基址寄存器}] + [\text{变址寄存器}]$

组合形式	寻址格式	操作数寻址方式
一个偏移量	$[DISP]$	直接寻址方式
一个间址寄存器	$[\text{基址寄存器}]$	寄存器间接寻址方式
	$[\text{变址寄存器}]$	
一个间址寄存器和一个偏移量	$[DISP] + [\text{基址寄存器}]$	寄存器相对寻址方式
	$[DISP] + [\text{变址寄存器}]$	
两个间址寄存器	$[\text{基址寄存器}] + [\text{变址寄存器}]$	基址变址寻址方式
所有都包含	$[DISP] + [\text{基址寄存器}] + [\text{变址寄存器}]$	相对基址变址寻址方式

# 02 | 8086寻址方式-练习1

## 源操作数寻址方式判断正误

- |                 |   |         |
|-----------------|---|---------|
| 1. MOV AX, [SP] |  x | 间址寄存器错误 |
| 2. MOV AX, SP   |  ✓ | 寄存器寻址   |
| 3. MOV AX, [SI] |  ✓ | 寄存器间接寻址 |
| 4. MOV AX, SI   |  ✓ | 寄存器寻址   |
| 5. MOV AX, [DS] |  x | 间址寄存器错误 |
| 6. MOV AX, DS   |  ✓ | 寄存器寻址   |










## 02 | 8086寻址方式-练习2

### 求有效地址的值

- 试说明下列指令中源操作数的寻址方式，并指出指令执行后寄存器 AX 中的值或所要寻找操作数的 EA 值

- 设  $(BX)=2000H$ ,  $(SI)=40H$

- |                          |   |
|--------------------------|---|
| 1. MOV AX, [1234H]       |  直接寻址方式, EA=1234H       |
| 2. MOV AX, 1234H         |  立即数寻址方式, (AX)=1234H    |
| 3. MOV AX, BX            |  寄存器寻址方式, (AX)=2000H    |
| 4. MOV AX, [BX]          |  寄存器间接寻址方式, EA=2000H    |
| 5. MOV AX, [BX+1234H]    |  寄存器相对寻址方式, EA=3234H  |
| 6. MOV AX, [BX+SI]       |  基址变址寻址方式, EA=2040H   |
| 7. MOV AX, [BX+SI+1234H] |  相对基址变址寻址方式, EA=3274H |

# 02 | 8086寻址方式-练习3-1

## 求 AX 的值

- 已知  $(DS)=2000H$ ,  $(SS)=2001H$ ,  $(BX)=100H$ ,  $(BP)=0F3H$ ,  $(SI)=2$ , 内存单元中的值如图所示, 试指出源操作数的寻址方式, 并给出指令执行后 AX 的值。

1. `MOV AX, 1200H`

- 立即数寻址方式,  $(AX)=1200H$

2. `MOV AX, BX`

- 寄存器寻址方式,  $(AX)=(BX)=100H$

3. `MOV AX, [BX]`

- 寄存器间接寻址方式,  $EA=100H$
- $PA=(DS) \times 16 + EA = 20000H + 100H = 20100H$
- $(AX)=3412H$

	.....
20100H	12
20101H	34
20102H	05
20103H	06
20104H	7C
20105H	10
20106H	B7
20107H	01
	.....

# 02 | 8086寻址方式-练习3-2

## 求 AX 的值

- 已知  $(DS)=2000H$ ,  $(SS)=2001H$ ,  $(BX)=100H$ ,  $(BP)=0F3H$ ,  $(SI)=2$ , 内存单元中的值如图所示, 试指出源操作数的寻址方式, 并给出指令执行后 AX 的值。

### 4. `MOV AX, 1[BX]`

- 寄存器相对寻址方式,  $EA=(BX)+1=101H$
- $PA=(DS) \times 16 + EA = 20000H + 101H = 20101H$
- $(AX)=0534H$

### 5. `MOV AX, [BX][SI]`

- 基址变址寻址方式,  $EA=(BX)+(SI)=102H$
- $PA=(DS) \times 16 + EA = 20000H + 102H = 20102H$
- $(AX)=0605H$

	.....
20100H	12
20101H	34
20102H	05
20103H	06
20104H	7C
20105H	10
20106H	B7
20107H	01
	.....

# 02 | 8086寻址方式-练习3-3

## 求 AX 的值

- 已知  $(DS)=2000H$ ,  $(SS)=2001H$ ,  $(BX)=100H$ ,  $(BP)=0F3H$ ,  $(SI)=2$ , 内存单元中的值如图所示, 试指出源操作数的寻址方式, 并给出指令执行后 AX 的值。

6. `MOV AX, [BP][SI]`

- 基址变址寻址方式,  $EA=(BP)+(SI)=0F5H$
- $PA=(SS) \times 16 + EA = 20010H + 0F5H = 20105H$
- $(AX)=0B710H$

7. `MOV AX, DS:11H[BP][SI]`

- 相对基址变址寻址方式,  $EA=(BP)+(SI)+11H=106H$
- $PA=(DS) \times 16 + EA = 20000H + 106H = 20106H$
- $(AX)=01B7H$

	.....
20100H	12
20101H	34
20102H	05
20103H	06
20104H	7C
20105H	10
20106H	B7
20107H	01
	.....

# 02 | 8086寻址方式-练习3-4

## 求 AX 的值

- 已知  $(DS)=2000H$ ,  $(SS)=2001H$ ,  $(BX)=100H$ ,  $(BP)=0F3H$ ,  $(SI)=2$ , 内存单元中的值如图所示, 试指出源操作数的寻址方式, 并给出指令执行后 AX 的值。

8. `MOV AX, 0FFFFH[BX][SI]`

- 相对基址变址寻址方式
- $EA = (BX) + (SI) + 0FFFFH = 10101H$
- $PA = (DS) \times 16 + EA = 20000H + 101H = 20101H$
- $(AX) = 0534H$

	.....
20100H	12
20101H	34
20102H	05
20103H	06
20104H	7C
20105H	10
20106H	B7
20107H	01
	.....





## 02 | 8086寻址方式-练习4

指出下列各种操作数的寻址方式

- |               |   |
|---------------|---|
| ① [BX]        |  寄存器间接寻址方式   |
| ② SI          |  寄存器寻址方式     |
| ③ 435H        |  立即数寻址方式     |
| ④ [BP+DI+123] |  相对基址变址寻址方式  |
| ⑤ [23]        |  直接寻址方式      |
| ⑥ data        |  直接寻址方式      |
| ⑦ [DI+32]     |  寄存器相对寻址方式   |
| ⑧ [BX+SI]     |  基址变址寻址方式  |
| ⑨ [BP+4]      |  寄存器相对寻址方式 |







## 02 | 8086寻址方式-练习5-1

### 判断下列操作数的寻址方式的正确性

- |              |   |
|--------------|---|
| ① [AX]       |  ✗ 16 位间址寄存器只有 BX, BP, SI, DI  |
| ② BP         |  ✓ 寄存器寻址方式                     |
| ③ [SI+DI]    |  ✗ 间址寄存器不能同时为变址寄存器             |
| ④ DS         |  ✓ 寄存器寻址; 如: MOV DS, AX        |
| ⑤ BH         |  ✓ 寄存器寻址                       |
| ⑥ [BX+BP+32] |  ✗ 间址寄存器不能同时为基址寄存器             |
| ⑦ [BL+44]    |  ✗ 16 位间址寄存器只有 BX, BP, SI, DI |

## 02 | 8086寻址方式-练习5-2

### 判断下列操作数的寻址方式的正确性






- ①  $[CX+90H]$   ✗ CX 不可作为间址寄存器
- ②  $BX+90H$   ✗ 应为  $[BX+90H]$
- ③  $[DX]$   ✗ DX 不能作间址寄存器
- ④  $SI[100H]$   ✗ 书写格式应为  $100H[SI]$  或  $[SI+100H]$
- ⑤  $[BX*4]$   ✗ 16 位系统中没有比例因子寻址方式
- ⑥  $[DX+90H]$   ✗ DX 不能作间址寄存器



# 02 | 8086寻址方式-练习6






## 求有效地址的值

- 已知寄存器 BX、DI 和 BP 的值分别为 2345H、0FFF0H 和 42H，试分别计算各操作数的有效地址

- ① [BX]  EA=2345H
- ② [DI+123H]  EA=(DI) + 123H=10113H (mod 64K)= 0113H
- ③ [BP+DI]  EA=(BP) + (DI) =10032H (mod 64K)= 0032H
- ④ [BX+DI+200H]  EA=(BX) + (DI) +200H=12535H (mod 64K)= 2535H
- ⑤ [1234H]  EA=1234H

# 02 | 8086寻址方式-练习7

指出下列各寻址方式所使用的段寄存器

- ①  $[SI+34H]$   DS
- ②  $[456H]$   DS
- ③  $ES:[BP+DI]$   ES
- ④  $[BX+DI+200H]$   DS
- ⑤  $[BP+1234H]$   SS

**[BP+DI+90H]** 所使用的段寄存器是 ( )

☐ A CS

☐ B DS

☐ C ES

☒ D SS

# 02 | 8086寻址方式-程序寻址方式

## 程序寻址方式

- 程序在存储器中的寻址方式分为**段内转移**和**段间转移**
  - 段内转移指转移地址与转移指令地址**在同一段中**
  - 段间转移指目标地址与转移指令地址**不在同一个段中**

## 程序寻址方式分类

- (段内)直接寻址方式
  - 段内间接寻址方式
  - 段间直接寻址方式
  - 段间间接寻址方式
- 在同一代码段内转移，  
只修改 **IP**，CS 不变
- 在不同代码段之间的转移，  
同时修改 **CS** 和 **IP**

# 02 | 8086寻址方式-段内寻址方式

## (段内)直接寻址方式

- 段内直接寻址方式也称为相对寻址方式，用指令本身提供的**位移量**加到 **IP** 中，形成有效目标地址

```
JMP 1000H ;转移地址在指令中给出  
CALL 1000H ;调用地址在指令中给出
```

## 段内间接寻址方式

- 程序转移的地址存放在寄存器或存储单元中，根据数据寻址方式得到转移有效地址，用它来**更新 IP** 的内容
  - 由于此寻址方式仅修改 IP 的内容，所以这种寻址方式只能在段内进行程序转移

```
JMP BX ;转移地址由 BX 给出  
CALL AX ;调用地址由 AX 给出  
JMP WORD PTR [BP+TABLE] ;转移地址由 BP+TABLE 所指的存储单元给出
```

## 02 | 8086寻址方式-段间寻址方式

### 段间直接寻址方式

- 在指令中**直接给出** 16 位的段基值和 16 位的偏移地址，用来**更新当前的 CS 和 IP** 的内容。

```
JMP 2500H:3600H  
CALL 2600H:3800H
```

### 段间间接寻址方式

- 程序转移的地址存放在**存储单元**中，根据存储器寻址方式得到转移地址的偏移量和段地址，用它来**更新 CS 和 IP** 的内容
  - 存储单元中**低位字地址单元**存放的是**偏移地址**，**高位字地址单元**中存放的是**转移段基值**

```
JMP WORD PTR [BX]
```

# 02 | 8086寻址方式-端口寻址方式

## 端口寻址方式

- 对输入输出设备的端口地址寻址

## 直接端口寻址

- 由指令直接给出端口地址，端口地址范围为 0 ~ 255

```
OUT AL, 32H
```

## 间接端口寻址

- 由 DX 寄存器指出端口地址，这种方式给出的端口地址范围为 0 ~ 65535

```
IN AL, DX
```

# 本节小结

- 了解机器指令格式（操作码与操作数）
- 掌握标识符的**写法**，能够识别并写出正确的标识符
- 理解寻址方式的**含义**，掌握 16 位系统使用的 7 种寻址方式
  - 特别是存储器寻址的 5 种方式，**EA 和 PA 形成**
- 能够**判断**寻址方式书写格式的正误
- 掌握寻址方式所使用的**段寄存器**
- 能够计算存储器寻址方式的**有效地址**、**物理地址**



# 目录

---

- 01 80x86 指令系统概述
- 02 8086 的寻址方式
- 03 8086 的常用指令**
- 04 80386 的寻址方式和指令系统

## 03 | 上节回顾-寻址方式

### 简答题

- 8086 共有几种寻址方式？分别是哪些？
- 间址寄存器有哪些？它们默认的段寄存器分别是什么？

# 03 | 8086常用指令-学习指令的关注点

## 指令的格式

- 操作数的个数，每个操作数使用的寻址方式

## 指令的功能

- 指令执行的目的和结果

## 对标志位的影响

- 指令执行时，是否需要标志位参与
- 指令执行后，对标志位的影响（改变）

## 指令的默认设置

- 指令执行时的约定设置、必须预置的参数、隐含使用的寄存器等

## 03 | 8086常用指令-对标志位的影响

### 对标志位有影响

- 按指令执行结果设置相应的标志位

### 对标志位不影响

- 指令执行不改变标志状态

### 对标志位无定义

- 指令执行后这些标志是任意的、不可预测

# 03 | 8086常用指令-约定符号

## 约定符号

符号	含义	符号	含义
DST	目的操作数 (destination)	MEM8/16	8/16 位存储单元
SRC	源操作数 (source)	IMM8/16	8/16 位立即数
REG	通用寄存器 (register)	REG8/16	8/16 位通用寄存器
SEG	段寄存器 (segment)	OPRD	操作数 (operand)
MEM	存储单元 (memory)	/	或者
IMM	立即数 (immediate)	=	赋值

# 03 | 8086常用指令 - 常用指令

## 常用指令

8086/8088常用指令

### 数据传送指令

- MOV指令<sup>①</sup>
- 交换指令XCHG
- I/O指令IN/OUT
- 取有效地址LEA
- 装入地址指令LDS/LES
- 标志传送指令LAHF/SAHF/PUSHF/POPF
- 表转换指令XLAT
- 堆栈操作指令PUSH/POP
- 符号扩展指令CBW/CBD

### 算术运算指令

- 加法类指令<sup>①</sup>
- 减法类指令<sup>①</sup>
- 乘法指令<sup>②</sup>
- 除法指令<sup>②</sup>
- BCD调整指令<sup>②</sup>

### 逻辑指令

AND/OR/XOR/NOT/TEST

### 移位与循环移位指令

- 算术移位指令SAL/SAR
- 逻辑移位指令SHL/SHR
- 循环移位指令ROL/ROR/RCL/RCR

### 处理器指令

- 标志位操作指令CLC/STC/CMC/CLD/STD/CLI/STI
- 同步与控制指令<sup>①</sup>

### 03 | 8086常用指令 - 数据传送指令

#### 数据传送指令

- 数据传送指令用于实现 CPU 的内部寄存器之间、CPU 内部寄存器和存储器之间、CPU 累加器 AX 或 AL 和 I/O 端口之间的数据传输

#### 特点

- 除了 SAHF 和 POPF 指令外均不影响标志寄存器的内容
- 需要注意的是，在数据传送指令中，源操作数和目的操作数的数据长度必须一致

2024/2/5

### 03 | 8086常用指令 - 算术运算指令

#### 算术运算指令

- 算术运算指令包括加、减、乘、除 4 种基本运算指令，以及为适应进行 BCD 码十进制运算而设置的各种校正指令

#### 特点

- 算术运算指令的执行均会影响标志位的状态
- 除加 1、减 1 指令外，均为双操作数指令
- 双操作数指令的两个操作数必须有一个在寄存器中，双操作数指令的目的操作数不允许使用立即数
- 单操作数指令也不允许使用立即数

2024/2/5

# 03 | 8086常用指令 - 数据传送指令

## 数据传送指令

- 数据传送指令用于实现 CPU 的**内部寄存器**之间、CPU **内部寄存器和存储器**之间、CPU **累加器 AX 或 AL** 和 **I/O 端口**之间的数据传送

## 特点

- 除了 SAHF 和 POPF 指令外**均不影响标志寄存器的内容**
- 需要注意的是，在数据传送指令中，**源操作数和目的操作数的数据长度必须一致**

# 03 | 8086常用指令-MOV指令-格式

## MOV 指令格式

- MOV <DST>, <SRC>
  - DST: REG/MEM/SEG
  - SRC: REG/MEM/SEG/IMM

## 功能

- 使用源操作数的值为目的操作数指定的内容赋值

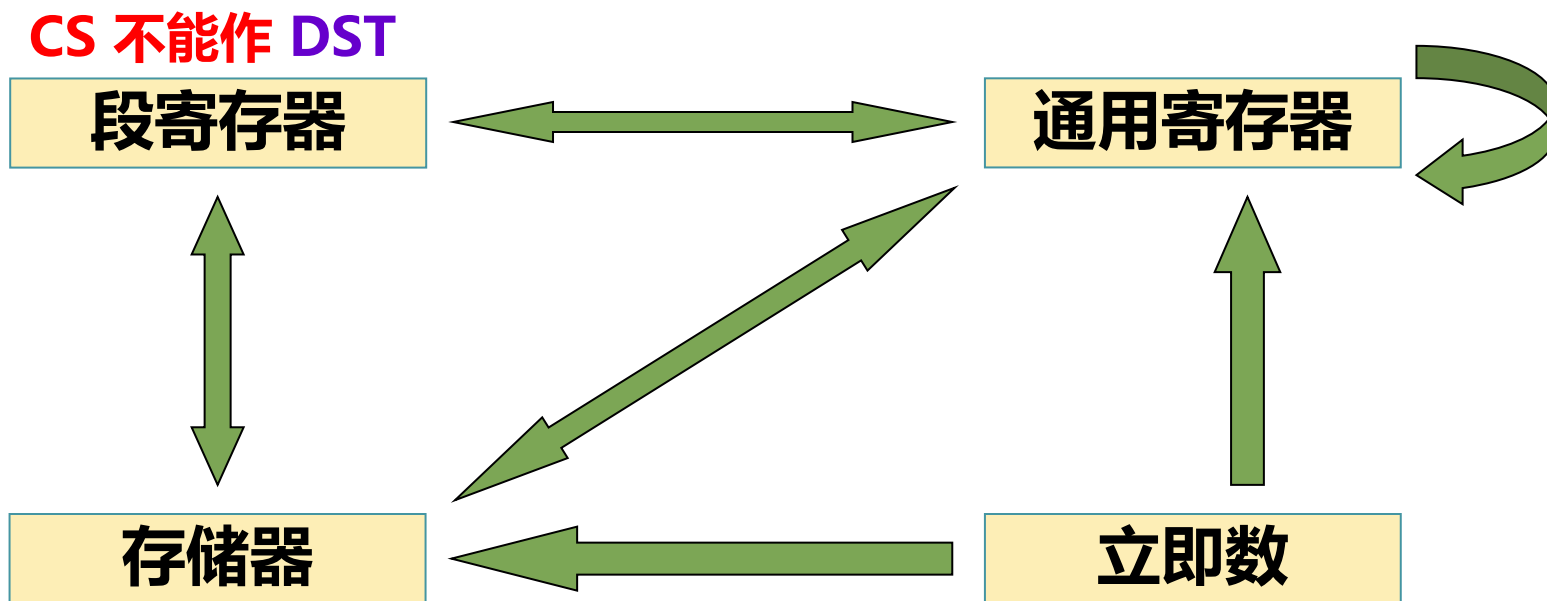
## 注意

- DST 和 SRC 不能都是存储器操作数
- 不允许在**段寄存器**之间传输数据，不能向段寄存器传送立即数
- 不能用 CS 和 IP 作为 DST，不允许用立即数作为 DST



# 03 | 8086常用指令-MOV指令-数据传送

## MOV 指令数据传送示意图



# 03 | 8086常用指令-MOV指令-举例

## 使用举例

```
1. MOV BL, 44           ; 44 → BL
2. MOV BYTE PTR [DI], 78H ; 78H → [DI]
3. MOV AX, [BX]         ; [BX] → AX
4. MOV SI, DI           ; DI → SI
5. MOV [2000H], CX      ; CX → [2000H]
6. MOV DS, BX           ; BX → DS
7. MOV DS, [2000H]      ; [2000H] → DS
8. MOV BX, ES           ; ES → BX
```

- 立即数 ⇒ 通用寄存器 (1)
- 立即数 ⇒ 存储器 (2)
- 段寄存器 ⇔ 存储器 (7)
- 通用寄存器 ⇔ 通用寄存器 (4)
- 段寄存器 ⇔ 通用寄存器 (6, 8)
- 存储器 ⇔ 通用寄存器 (3, 5)

# 03 | 8086常用指令-MOV指令-规定

## 有关操作数的规定

- 两个操作数不能同时为段寄存器
- 两个操作数不能同时为存储器寻址方式
- 两个操作数必须类型匹配
- 立即数不能作为目的操作数
- 立即数不能直接为段寄存器赋值
- CS 不能作为目的操作数，IP 不能作为操作数

# 03 | 8086常用指令 - MOV指令 - 操作数匹配问题

## 两操作数类型确定

```
MOV AX, BX
```

- 二者**类型必须相同**。同为字节型、字型、双字型等

## 其中一个不确定

```
MOV AX, 1
```

- 由确定类型的操作数决定指令的操作类型
  - **确定的类型**：寄存器、变量；
  - **不确定的类型**：立即数、非符号地址构成的存储器寻址方式

## 两个都不确定

```
MOV word PTR [2000H], 1
```

- 必须进行**强制类型**，以指明指令的操作类型
  - 两操作数状态：目的——非符号地址构成的存储单元，源——立即数
  - 对存储单元进行 PTR 的类型强制 (第 4 章中介绍)

# 03 | 8086常用指令-MOV指令-变通方法

## 错误指令

段寄存器  $\nRightarrow$  段寄存器

- `MOV ES, DS`

立即数  $\nRightarrow$  段寄存器

- `MOV DS, 100H`

存储器  $\nRightarrow$  存储器

- `MOV VARA, VARB`



## 可选的解决方法

```
MOV AX, DS
```

```
MOV ES, AX
```

```
MOV AX, 100H
```

```
MOV DS, AX
```

```
MOV AX, VARB
```

```
MOV VARA, AX
```



# 03 | 8086常用指令-MOV指令-练习1

## 指出下列指令的错误

- |                     |                       |
|---------------------|-----------------------|
| 1. MOV AX, BL       | ;两个操作数类型不匹配           |
| 2. MOV AL, 3824H    | ;不能用 16 位数据对 8 位寄存器赋值 |
| 3. MOV X, Y         | ;两个操作数不能同时为存储器单元      |
| 4. MOV DS, 5000H    | ;立即数不能直接为段寄存器赋值       |
| 5. MOV [AX], BX     | ;AX 不能作为间址寄存器         |
| 6. MOV AX, [SI][DI] | ;源操作数寻址方式不正确          |
| 7. MOV 35H, AL      | ;立即数不能作为目的操作数         |
| 8. MOV CS, AX       | ;CS 不能作为目的操作数         |
| 9. MOV [SI], [DI]   | ;两个操作数不能同时为存储器单元      |
| 10. MOV [BX], 1     | ;两操作数类型不确定, 需要强制类型    |

## 03 | 8086常用指令-MOV指令-练习2

设 b 是已定义的字节变量，判断以下指令的正误

1. MOV AX, b

;✗ 两个操作数类型不匹配

2. MOV b, 0

;✓ b 确定为字节类型

3. MOV [BX], 0

;✗ 两个操作数类型均不确定

4. MOV word ptr [BX], 0

;✓ 指令为字操作

关于指令 **MOV [BX], 0FFH** 错误原因描述正确的是 ( )

- ☐ A BX 不能作为间址寄存器
- ☒ B 两个操作数类型不确定
- ☐ C 立即数不能直接为 [BX] 赋值
- ☐ D 立即数不能作为源操作数



# 03 | 8086常用指令-XCHG指令-格式

## 交换指令 XCHG (exchange) 格式

- XCHG <OPRD1>, <OPRD2>

- OPRD1: REG/MEM

- OPRD2: REG/MEM

## 功能

- 将指定的两个操作数的值相交换

## 注意

- OPRD1 和 OPRD2 不能同时为存储器操作数
- 任意一个操作数都不能使用段寄存器, 也不能使用立即数

# 03 | 8086常用指令-XCHG指令-举例

## 使用举例

1. XCHG AL, AH ; 8 位交换
2. XCHG SI, BX ; 16 位交换
3. XCHG EAX, EBX ; 32 位交换
- 4.
5. XCHG AL, [EBX] ; AL 与由 EBX 指定的**字节**存储单元交换
6. XCHG [ESI], BX ; BX 与由 ESI 指定的**字**存储单元交换
7. XCHG EDX, [EDI] ; EDX 与由 EDI 指定的**双字**存储单元交换

# 03 | 8086常用指令-LEA指令-格式

## 取有效地址 LEA (Load Effective Address) 指令格式

● LEA <DST>, <SRC>

■ DST: REG

■ SRC: MEM

### 功能

- 把 SRC 的有效地址送给指定的 DST

### 注意

- DST 和 SRC 不能同时为存储器操作数
- 任意一个操作数都不能使用段寄存器, 也不能使用立即数


# 03 | 8086常用指令-LEA指令-举例

## LEA 示例 1


```
1. MOV EDI, 51234H           ;EDI=00051234H
2. MOV EAX, 6                 ;EAX=00000006H
3. LEA ESI, [EDI+EAX]         ;ESI=0005123AH
4. LEA ECX, [EAX*4]           ;ECX=00000018H
5. LEA EBX, [EDI+EAX*4+300H]  ;EBX=0005154CH
```

## LEA 示例 2

- BUF 定义如右, 指令 **LEA SI, BUF** 的结果?

 (SI)=0500H

- 指令 **MOV SI, BUF** 的结果?

 (SI)=1234H

	.....	
500H	34H	BUF
501H	12H	
	.....	

# 03 | 8086常用指令-LEA指令-区分1

## LEA 指令与 MOV 指令的区分

- 假设某数据段定义如下：

```
DATA SEGMENT
    TABLE DW 0040H
            DW 3000H
DATA ENDS
```

请指出下列指令的执行结果

- |                         |                |
|-------------------------|----------------|
| 1. MOV BX, TABLE        | ; (BX) = 0040H |
| 2. LEA BX, TABLE        | ; (BX) = 0000H |
| 3. MOV BX, OFFSET TABLE | ; (BX) = 0000H |

**MOV** 传送的是 **SRC** 的**内容**，且只能简单地传送内容；  
**LEA** 传送的是 **SRC** 的**偏移地址**，相当于 C 语言中的取地址符 **&**，**LEA** 指令 **SRC** 可以进行复杂的运算。

## 03 | 8086常用指令-LEA指令-区分2

右边两条指令的效果一样吗?

```
1. LEA BX, BUFFER  
2. MOV BX, OFFSET BUFFER
```

 **效果一样**，都是取 BUFFER 的有效地址送给 BX 寄存器

 两条指令的不同之处


- LEA: 3 字节**存储器寻址**指令

- MOV: 2 字节**立即寻址**指令

右边两条指令都正确吗?

```
1. LEA BX, [BX+200]  
2. MOV BX, OFFSET [BX+200]
```

 第一条正确，第二条不正确

 因为，OFFSET 是对**变量**和**标号**求偏移属性的，而不能对其他的存储单元寻址方式来用

# 03 | 8086常用指令-装入地址指令

## 装入地址指令 LDS/LES 格式

Load Pointer Using **DS**  
Load Pointer Using **ES**

● **LDS/LES** <DST>, <SRC>

■ **DST**: REG16

■ **SRC**: MEM32

## 功能

- 将 **SRC** 指定的单元中低字数据送入指定的 **DST** 中，高字数据传送给 DS/ES 寄存器

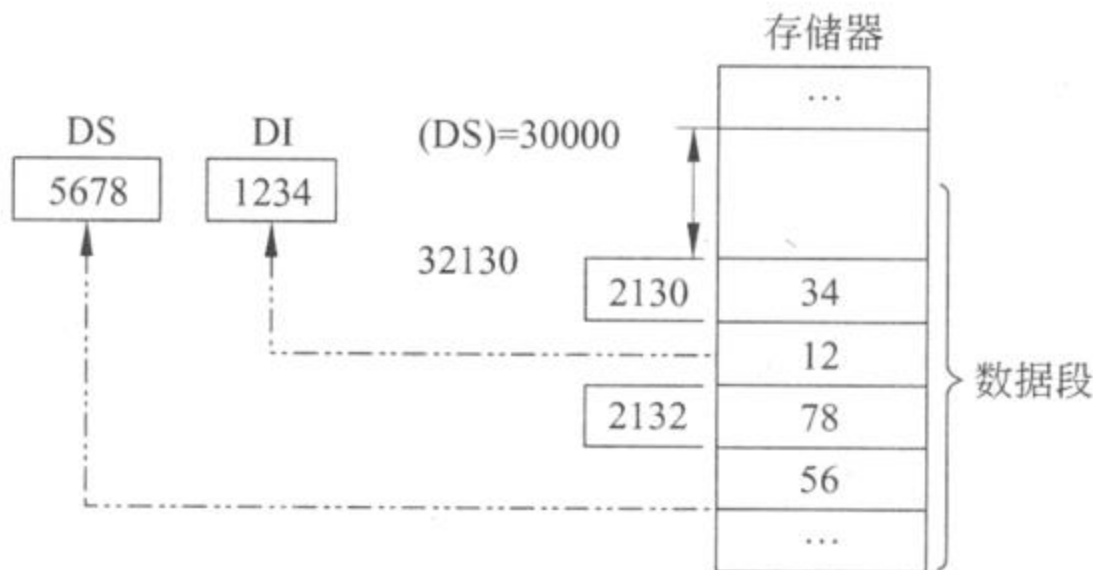
## 注意

- **DST** 必须是通用寄存器之一
- **SRC** 必须是内存操作数

# 03 | 8086常用指令-装入地址指令-示例

## 装入地址指令示例

- **LDS DI, [2130H]** ;将 2130H 和 2131H 单元的内容送 DI  
;将 2132H 和 2133H 单元的内容送 DS
- 若 [2130H] 对应的双字数据的值为 56781234H, 则  
执行该指令后 (DI)=1234H, (DS)=5678H





## 03 | 8086常用指令-堆栈

### 堆栈

- 一段内存区域，只是对它的访问操作仅限于一端进行
  - 地址较大的一端被称为**栈底**，地址较小的一端被称为**栈顶**

### 堆栈寄存器

- **堆栈段寄存器 SS** 含有当前堆栈段的段号，SS 指示堆栈所在内存区域的位置
- **堆栈指针寄存器 ESP** 含有栈顶的偏移地址 (有效地址)，ESP 指向栈顶
- **基址寄存器 EBP** 存放堆栈段中的一个数据区基址的偏移地址

# 03 | 8086常用指令 - PUSH指令

## 进栈 PUSH 指令格式

- PUSH <SRC>

- SRC: REG/MEM/SEG/IMM

- DST: 隐含, 由 SS:ESP 指定, 类型与 SRC 匹配

## 功能

- 将 SRC 压入堆栈

## 注意

- SRC 至少是 16 位数据
- 先改变 ESP 的值, 再把 SRC 数据送到 ESP 所指的存储单元
- ESP 总是指向栈顶

# 03 | 8086常用指令 - PUSH指令 - 举例

## PUSH 指令示例

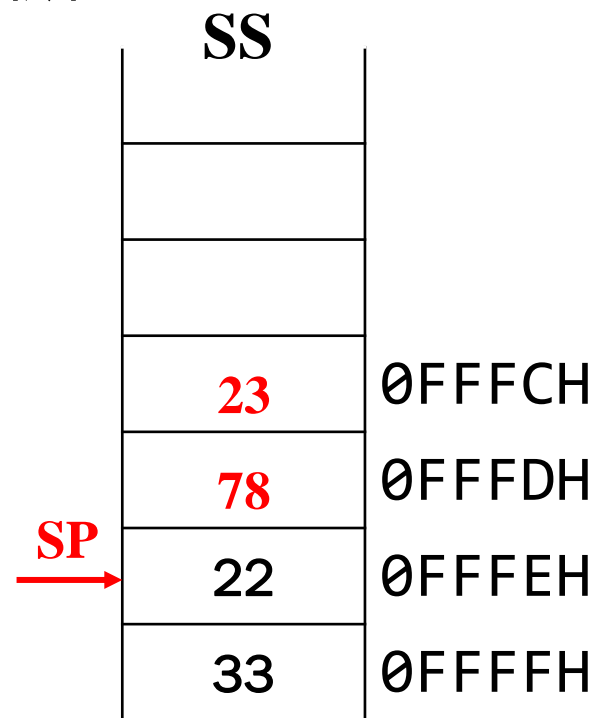
- 设  $(AX)=7823H$ ,  $(SP)=0FFFEH$   
则指令 **PUSH AX** 的执行过程如右图所示

✎ 修改指针:  $(SP) - 2 \rightarrow SP$

$(SP)=0FFFCH$

✎ 保存数据:  $7823H \rightarrow [SP]$

$[0FFFCH]=7823H$



# 03 | 8086常用指令 - POP指令

## 出栈 POP 指令格式

- POP <DST>
  - DST: REG/MEM/SEG
  - SRC: 隐含, 由 SS:ESP 指定

## 功能

- 从栈顶弹出一个双字或者字数据到 DST


## 注意

- DST 不能是立即数和 CS
- 先将 ESP 所指的存储单元内容传送到 DST 中, 再改变 ESP 的值, 至少出栈 16 位数据
- ESP 总是指向栈顶

# 03 | 8086常用指令 - POP指令 - 举例

## POP 指令示例

1. `POP ESI` ; 从堆栈弹出一个双字到 ESI
2. `POP DWORD PTR [EBX+4]` ; 从堆栈弹出一个双字到 EBX+4 所指示存储单元
3. `POP DI` ; 从堆栈弹出一个字到 DI
4. `POP WORD PTR [EDX+8]` ; 从堆栈弹出一个字到 EDX+8 所指示的存储单元

 符号 “`DWORD PTR`” 表示指定双字存储单元

 符号 “`WORD PTR`” 表示指定字存储单元

 至少出栈一个字!

# 03 | 8086常用指令 - 堆栈操作指令 - 举例

## 堆栈保护寄存器内容示例

```
1. PUSH EBP      ;保护 EBP
2. PUSH ESI      ;保护 ESI
3. PUSH EDI      ;保护 EDI
4. ....         ;其他操作
5. ....         ;其间会破坏 EBP、ESI 和 EDI 的原有值
6. POP EDI       ;恢复 EDI
7. POP ESI       ;恢复 ESI
8. POP EBP       ;恢复 EBP
```

- 必须充分注意堆栈操作“**后进先出**”，同时确保**堆栈平衡**

# 03 | 8086常用指令-全进栈出栈指令

## 通用寄存器全进栈出栈 PUSH/POPA 指令格式

Push All Registers  
onto Stack  
Pop All Registers  
onto Stack

### ● PUSH

- SRC: REG16
- DST: 隐含, 由 SS:ESP 指定

### ● POPA

- SRC: 隐含, 由 SS:ESP 指定
- DST: REG16

## 功能

- 将 8 个 16 位通用寄存器的内容按顺序压入或弹出堆栈

## 注意

- **PUSH** 将通用寄存器内容压入堆栈的顺序:
  - AX、CX、DX、BX、SP、BP、SI、DI
- **POPA** 指令从堆栈弹出内容, 以相反的顺序送入通用寄存器

# 03 | 8086常用指令 - 全进栈出栈指令 - 举例

## 堆栈保护寄存器内容示例

```
1. PUSH EBP      ;保护 EBP
2. PUSH ESI      ;保护 ESI
3. PUSH EDI      ;保护 EDI
4. ....         ;其他操作
5. ....         ;其间会破坏 EBP、ESI 和 EDI 的原有值
6. POP EDI       ;恢复 EDI
7. POP ESI       ;恢复 ESI
8. POP EBP       ;恢复 EBP
```

```
1. PUSHA         ;把 8 个通用寄存器之值全部压入堆栈
2. ....         ;其他操作
3. ....         ;其间会破坏通用寄存器的原有值
4. POPA          ;恢复 8 个通用寄存器
```



# 03 | 8086常用指令-符号扩展指令

## 字节/字扩展 CBW/CWD 指令格式

Convert Byte to Word  
Convert Word to Doubleword

### ● CBW

- SRC: AL
- DST: AH

### ● CWD

- SRC: AX
- DST: DX

## 功能

- CBW 将 AL 中的单字节数据的符号扩展到 AH 中
- CWD 将 AX 中的单字数据的符号扩展到 DX 中

## 注意

- 这两条指令进行的是**补码扩展**
  - 若  $AL < 80H$ , 则  $0 \rightarrow AH$ ; 若  $AL \geq 80H$ , 则  $0FFH \rightarrow AH$
  - 若  $AX < 8000H$ , 则  $0 \rightarrow DX$ ; 若  $AX \geq 8000H$ , 则  $0FFFFH \rightarrow DX$
- 这两条符号扩展指令在**有符号数的乘法和除法**运算中十分有用, **对标志位没有影响**

## 03 | 8086常用指令-符号扩展指令-举例

请写出下列指令序列中每条指令的执行结果

1. MOV AL, 80H ; (AL) = 80H
2. CBW ; (AX) = 0FF80H
3. ADD AL, 255 ; (AX) = 0FF7FH
4. CBW ; (AX) = 007FH
5. NEG AX ; (AX) = 0FF81H
6. CWD ; (DX:AX) = 0FFFF FF81H

**MOV AX, 8765H**

**CBW**

执行上述指令后，AX 的内容是什么？

- ☐ A 8765H
- ☐ B 6665H
- ☒ C 0065H
- ☐ D 0FF65H

# 03 | 8086常用指令 - 标志传送指令1

## 标志传送 LAHF/SAHF 指令格式

Load AH from Flags  
Store AH into Flags

- LAHF/SAHF

## 功能

- LAHF 将 EFLAGS 寄存器的低 8 位传送到 AH
- SAHF 将 AH 内容传送到 EFLAGS 寄存器低 8 位

## 注意

- 16 位 CPU 中传送的是 FLAGS 寄存器的低 8 位
  - 被传送的标志位包括：符号标志位、零标志位、辅助进位标志位、奇偶标志位和进位标志位
  - 使用这条指令，可以方便地把标志位副本保管在变量中

# 03 | 8086常用指令 - 标志传送指令2

## 标志传送 PUSHF/POPF 指令格式

Push Flags onto Stack  
Pop Flags off Stack

- PUSHF/POPF

## 功能

- PUSHF 将 FLAGS 寄存器内容压入堆栈
- POPF 将栈顶内容弹出至 FLAGS 寄存器

## 注意

- 32 位 CPU 中传送的是 EFLAGS 寄存器的低 16 位
- 32 位 CPU 中将整个 EFLAGS 寄存器内容作为双字压入或弹出堆栈的指令是 PUSHFD/POPCD

# 03 | 8086常用指令-XLAT指令

## 表转换 XLAT (Translate) 指令格式

- `XLAT <label>`
- `XLAT ;AL ← [BX+AL]`

## 功能

- 用 BX 寻址用户自定义表中的字节替换 AL 中的字节

## 注意

- AL 的原始值是转换表的索引，用于码的转换
- 描述 `XLAT` 的最佳方式是 `MOV AL, [BX+AL]`

# 03 | 8086常用指令 - I/O指令

## 输入输出端口 IN/OUT 指令格式

● IN <DST>, <SRC>

■ SRC: IMM8/DX

■ DST: AL/AX

● OUT <DST>, <SRC>

■ SRC: AL/AX

■ DST: IMM8/DX

## 功能

- IN 把外部端口 SRC 的内容传送到 AL/AX 中
- OUT 把 AL/AX 中的内容输出至指定端口 DST

## 注意

- IN 指令的 DST 和 OUT 指令的 SRC 必须是 AL/AX
- 直接寻址使用 8 位立即数, 寻址范围是 00H~0FFH
- 间接寻址使用 DX 作为间址寄存器, 寻址范围为 0000H~0FFFFH

# 03 | 8086常用指令 - 算术运算指令

## 算术运算指令

- 算术运算指令包括**加、减、乘、除** 4 种基本运算指令，以及为适应进行 BCD 码十进制运算而设置的各种**校正指令**

## 特点

- 算术运算指令的执行**均会影响标志位的状态**
- 除**加 1、减 1** 指令外，均为**双操作数指令**
- **双操作数指令**的两个操作数**必须**有一个**在寄存器中**，**双操作数指令的目的操作数不允许**使用**立即数**
- **单操作数指令也****不允许**使用**立即数**

03 | 8086常用指令 - 学习指令的关注点

指令的格式

- 操作数的个数，每个操作数使用的寻址方式

指令的功能

- 指令执行的目的和结果

对标志位的影响

- 指令执行时，是否需要标志位参与
- 指令执行后，对标志位的影响（改变）

指令的默认设置

- 指令执行时的约定设置，必须预置的参数、隐含使用的寄存器等等



# 03 | 8086常用指令-加法指令-ADD

## 不带进位的加法 ADD 指令格式

● **ADD** <DST>, <SRC> ; DST+SRC → DST

■ DST: REG/MEM

■ SRC: REG/MEM/IMM

## 功能

- SRC 和 DST 每一位相加，结果放在 DST 中

## 注意

- **不允许**两个存储器单元内容相加，也**不允许**在两个段寄存器之间相加
- 对标志位有影响，主要是 **CF**、**ZF**、**OF**、**SF** 标志位

# 03 | 8086常用指令-加法指令-ADC

## 带进位的加法 ADC (Add With Carry) 指令格式

● **ADC** <DST>, <SRC> ; DST+SRC+CF → DST

■ DST: REG/MEM

■ SRC: REG/MEM/IMM

## 功能

- SRC 和 DST 相加再加上 CF 的值，结果放在 DST 中
- 常用于多字/多字节数据的高字节加法

## 注意

- **不允许**两个存储器单元内容相加，也**不允许**在两个段寄存器之间相加
- 对标志位有影响，主要是 **CF**、**ZF**、**OF**、**SF** 标志位

# 03 | 8086常用指令-加法指令-INC

## 增量 (加 1) 指令 INC (Increment) 指令格式

- `INC <OPRD>` ; `OPRD+1 → OPRD`
  - `OPRD`: REG/MEM

## 功能

- `OPRD` 加上 1, 结果放在 `OPRD` 中
- 常用于**指针**、**计数器**的修改操作

## 注意

- `OPRD` 可以是寄存器或存储单元, 但**不能**为立即数
- 影响标志位 **AF**、**OF**、**PF**、**SF** 和 **ZF**, 但不影响 **CF** 位
- 将操作数视为无符号数

INC 指令不影响进位标志位（ ）

- ☒ A 正确
- ☐ B 错误

# 03 | 8086常用指令-加法指令-举例1

## 加法指令举例1

1. ADD CX, DX	; CX $\leftarrow$ (CX) + (DX)
2. ADD DI, [ALPHA+BX]	; DI $\leftarrow$ [ALPHA + (BX)] + (DI)
3. ADD TEMP, CL	; TEMP $\leftarrow$ (CL) + TEMP
4. ADD CL, 2	; CL $\leftarrow$ 2 + (CL)
5. ADD ALPHA, 2	; ALPHA $\leftarrow$ 2 + ALPHA
6. ADC AX, BX	; AX $\leftarrow$ (AX) + (BX) + CF
7. INC WORD PTR [BX]	; [BX] $\leftarrow$ [BX] + 1

- REG+REG (1, 6)
- MEM+REG (2, 3)
- IMM+REG (4)
- IMM+MEM (5, 7)

# 03 | 8086常用指令-加法指令-举例2

## 加法指令举例2

```
1. MOV EAX, 12345678H      ; (EAX) = 12345678H
2. MOV ECX, 01010101H      ; (ECX) = 01010101H
3. ADD EAX, ECX              ; (EAX) = 13355779H
4. ADD CX, AX                ; (ECX) = 0101587AH
5. ADD AL, CH                ; (EAX) = 133557D1H
6. ADD AL, 3                 ; (EAX) = 133557D4H
7. ADD CX, 0B000H           ; (ECX) = 0101087AH
```

- 8 位或 16 位独立操作，不影响高位数据！

# 03 | 8086常用指令-加法指令对标志位的影响1

## 加法指令对标志位的影响

加法指令	OF	SF	ZF	AF	PF	CF
ADD	修改	修改	修改	修改	修改	修改
ADC	修改	修改	修改	修改	修改	检测并修改
INC	修改	修改	修改	修改	修改	无影响

## CF 和 OF 的判断

- 和的最高有效位向高位进位，CF=1；否则，CF=0
- DST 和 SRC 最高位相同，而结果最高位 (SF) 与之相反，OF=1；否则，OF=0

# 03 | 8086常用指令-加法指令对标志位的影响2

## 示例

1. MOV AL, 0FBH	; (AL) = 0FBH	Z	S	P	A	C	O
2. ADD AL, 7	; (AL) = 02H	0	0	0	1	1	0
3. MOV BX, 1FEH	; (BX) = 1FEH						
4. ADD AL, BL	; (AL) = 00H	1	0	1	1	1	0
5. MOV word ptr [200H], 4652H	; [200H] = 4652H						
6. ADD word ptr [BX+2], 0F0F0H	; [200H] = 3742H	0	0	1	0	1	0

- 除了 AF，其他标志位均根据**当前运算的结果**改变标志位的值
  - 若参与运算是 8 位，则根据 8 位结果改变标志
  - 若参与运算是 16 位，则根据 16 位结果改变标志
  - 若参与运算是 32 位，则根据 32 位结果改变标志



# 03 | 8086常用指令-重点区分CF和OF

## 无符号数运算

- CF: 进借位标志/溢出标志
  - 溢出时, 差别仅在于高位, 低位结果正确
- OF: 无意义

## 有符号数运算

- CF: 进借位标志
- OF: 溢出标志
  - 溢出时, 数据性质发生变化, 结果不正确

# 03 | 8086常用指令-CF和OF的设置和使用

## 设置

- CPU 对两个操作数只按照**二进制规则**运算
- 产生运算结果后，设置标志位
  - CF：按实际的进位情况设置
  - OF：将其作为有符号数看待，来设置 OF

## 使用

- 判断溢出时使用哪个标志，则**由程序员来决定**
  - 若将参与运算操作数看作**无符号数**，则使用 CF，不用 OF
  - 若将参与运算操作数看作**有符号数**，则使用 OF

# 03 | 8086常用指令-加法指令-举例3

采用 16 位寄存器编写两个 32 位数据相加

```
1. MOV AX, WORD PTR d2      ; (AX) = 3412H
2. MOV DX, WORD PTR d2+2    ; (DX) = 7856H
3. ADD WORD PTR d1, AX      ; 低字和 = 0935H
4. ADC WORD PTR d1+2, DX    ; 高字和 = 01BEH
```

d1

.....
35
09
BE
01
12
34
56
78
.....

$$\begin{array}{r} 0011\ 0100\ 0001\ 0010B = 3412H \\ +\ 1101\ 0101\ 0010\ 0011B = D523H \\ \hline 1\ 0000\ 1001\ 0011\ 0101B = 0935H \\ CF=1, SF=0, OF=0, ZF=0 \end{array}$$

$$\begin{array}{r} 0111\ 1000\ 0101\ 0110B = 7856H \\ +\ 1000\ 1001\ 0110\ 0111B = 8967H \\ \hline 1B = CF \\ 1\ 0000\ 0001\ 1011\ 1110B = 01BEH \\ CF=1, SF=0, OF=0, ZF=0 \end{array}$$

d2

32 位 CPU 实现两个 32 位数据相加

```
1. MOV EAX, DWORD PTR d2
2. ADD DWORD PTR d1, EAX
```

# 03 | 8086常用指令-减法指令-SUB

## 不带借位的减法 SUB (Subtraction) 指令格式

● SUB <DST>, <SRC> ; DST-SRC → DST

■ DST: REG/MEM

■ SRC: REG/MEM/IMM

## 功能

- 将 DST 减去 SRC，结果送到 DST

## 注意

- 不允许两个存储器单元内容相减，也不允许在两个段寄存器之间相减
- 对标志位有影响，主要是 CF、ZF、OF、SF 标志位

# 03 | 8086常用指令-减法指令-SBB

## 带借位的减法 SBB (Subtraction with Borrow) 指令格式

● SBB <DST>, <SRC> ; DST-SRC-CF → DST

■ DST: REG/MEM

■ SRC: REG/MEM/IMM

### 功能

- 将 DST 减去 SRC，再减去 CF 的值，结果送到 DST

### 注意

- **不允许**两个存储器单元内容相减，也**不允许**在两个段寄存器之间相减
- 对标志位有影响，主要是 **CF**、**ZF**、**OF**、**SF** 标志位

# 03 | 8086常用指令-减法指令-DEC

## 减量 (减 1) 指令 DEC (Decrement) 指令格式

- DEC <OPRD> ;OPRD-1 → OPRD
- OPRD: REG/MEM

## 功能

- OPRD 减去 1, 结果放在 OPRD 中

## 注意

- OPRD 可以是寄存器或存储单元, 但**不能**为立即数
- 与 INC 一样影响标志位 **AF**、**OF**、**PF**、**SF** 和 **ZF**, 但不影响 **CF** 位
- 将操作数视为无符号数

# 03 | 8086常用指令-减法指令-NEG

## 取补指令 NEG (Negation) 指令格式

- $NEG \quad <OPRD>$  ;  $0-OPRD \rightarrow OPRD$
- $OPRD$ : REG/MEM

## 功能

- 将  $OPRD$  取补码后送回  $OPRD$  中
  - 将  $OPRD$  连同符号逐位取反，然后在末位加 1

## 注意

- $OPRD$  可以是寄存器或存储单元，但**不能**为立即数
- 对所有标志位有影响
- 适用于操作数在机器内用补码表示的场合

# 03 | 8086常用指令-减法指令-CMP

## 比较指令 CMP (Compare) 指令格式

● **CMP** <DST>, <SRC> ; DST-SRC

■ DST: REG/MEM

■ SRC: REG/MEM/IMM

## 功能

- DST 与 SRC 相减，不送回结果，只根据结果置标志位

## 注意

- DST 和 SRC **不能**同时为存储器单元和段寄存器
- 可根据标志位判断**两个操作数的大小关系**
- 一般情况下，**CMP** 指令后经常会有一条条件转移指令，用来检查标志位的状态是否满足某种关系



# 03 | 8086常用指令 - CMP指令判断两数大小

以 CMP A, B 为例

- 判断两个操作数是否**相等**
  - 若  $ZF=1$ , 则  $A=B$ ; 否则,  $A \neq B$
- 判断两个**无符号数**的大小
  - 若  $CF=1$ , 则  $A < B$ ; 否则,  $A \geq B$
- 判断两个**有符号数**的大小
  - 若  $SF \oplus OF=1$ , 即  $SF$  与  $OF$  不相同, 则  $A < B$ ; 否则,  $A \geq B$

无符号数		
ZF	CF	比较结果
1	---	$A=B$
0	0	$A > B$
0	1	$A < B$

有符号数		
ZF	标志位	比较结果
1	---	$A=B$
0	$SF=OF$	$A > B$
0	$SF \neq OF$	$A < B$

CMP 指令结果可作为 **JGE 指令** 的判断条件

## 03 | 8086常用指令-减法指令对标志位的影响

### 减法指令对标志位的影响

减法指令	OF	SF	ZF	AF	PF	CF
SUB/CMP	修改	修改	修改	修改	修改	修改
SBB	修改	修改	修改	修改	修改	检测并修改
DEC	修改	修改	修改	修改	修改	无影响
NEG	修改	修改	修改	修改	修改	修改

### CF 和 OF 的判断

- DST 的最高有效位向高位借位, CF=1; 否则, CF=0
- DST 和 SRC 最高位相反, 而结果最高位 (SF) 与 SRC 相同, OF=1; 否则, OF=0

## 03 | 8086常用指令-减法指令对标志位的影响

### 减法指令对标志位的影响

减法指令	OF	SF	ZF	AF	PF	CF
SUB/CMP	修改	修改	修改	修改	修改	修改
SBB	修改	修改	修改	修改	修改	检测并修改
DEC	修改	修改	修改	修改	修改	无影响
NEG	修改	修改	修改	修改	修改	修改

### NEG 对标志位的影响

- CF: 当 OPRD=0 时, CF=0; 否则 CF=1;
- OF: 当 OPRD 为负的最小值时, OF=1 ; 否则 OF=0;
  - 负的最小值: 80H、8000H、8000 0000H

# 03 | 8086常用指令-减法指令举例1

## 减法指令举例 1

1. SUB CX, DX	; CX $\leftarrow$ (CX) - (DX)
2. CMP DI, [ALPHA+BX]	; DI - [ALPHA + (BX)]
3. SUB TEMP, CL	; TEMP $\leftarrow$ TEMP - (CL)
4. SBB CL, 2	; CL $\leftarrow$ (CL) - 2 - CF
5. CMP ALPHA, 2	; ALPHA - 2
6. SBB AX, BX	; AX $\leftarrow$ (AX) - (BX) - CF
7. DEC WORD PTR [BX]	; [BX] $\leftarrow$ [BX] - 1
8. NEG AL	; AL $\leftarrow$ 0 - AL

- REG-REG (1, 6)
- REG-IMM (4)
- REG-MEM (2)
- MEM-IMM (5, 7)
- MEM-REG (3)

# 03 | 8086常用指令-减法指令举例2

## 减法指令举例 2

1. MOV AX, 0FF64H	; AX=0FF64H	Z	S	P	A	C	O
2. NEG AL	; AX=0FF9CH	0	1	1	1	1	0
3. SUB AL, 9DH	; AX=0FFFH	0	1	1	1	1	0
4. NEG AX	; AX= 0001H	0	0	0	1	1	0
5. DEC AL	; AX= 0000H	1	0	1	0	1	0
6. NEG AX	; AX= 0000H	1	0	1	0	0	0

### ● NEG 对标志位的影响

- 当且仅当操作数为零时，不产生借位 (CF=0)
- 当且仅当操作数为负的最小值时，产生溢出 (OF=1)

### ● DEC 不影响 CF 标志位

# 03 | 8086常用指令-加减法指令练习

写出下列指令序列中每条指令的执行结果

```
MOV BX, 23ABH      ; (BX) = 23ABH
ADD BL, 0ACH        ; (BL) = 0ABH + 0ACH = 57H (不是 157H)
                   ; (BX) = 2357H      OF=1, CF=1, SF=0
MOV AX, 23F5H      ; (AX) = 23F5H
ADD BH, AL          ; (BH) = 23H + 0F5H = 18H (不是 118H)
                   ; (BX) = 1857H      OF=0, CF=1, SF=0
SBB BX, AX          ; (BX) = 1857H - 23F5H - CF = 0F461H
                   ; (BX) = 0F461H     OF=0, CF=1, SF=1
ADC AX, 12H         ; (AX) = 23F5H + 12H + CF = 2408H
                   ; (AX) = 2408H      OF=0, CF=0, SF=0
SUB BH, -9          ; (BH) = 0F4H - 0F7H = 0FDH
                   ; (BX) = 0FD61H     OF=0, CF=1, SF=1
```

# 阶段小结

---

- 掌握 MOV、XCHG、LEA、PUSH/POP、CBW/CWD、ADD/ADC/INC、SUB/SBB/DEC/NEG
- 会使用 LDS/LES、PUSHA/POPA、LAHF/SAHF、PUSHF/POPF、XLAT、IN/OUT
- 掌握各指令的**格式、功能、注意事项、操作数的类型、指令执行后对标志位的影响**

## 03 | 上节回顾-数据传送指令

### 简答题

- 数据传送的源操作数与目的操作数的规定。
- 不确定的数据类型是什么？数据传送指令类型匹配问题指的是什么？
- 加减法指令对状态标志寄存器的影响是什么？



# 03 | 8086常用指令-乘法指令-MUL

## 无符号数的乘法 MUL (Multiply) 指令格式

- **MUL <OPRD>** ;AL/AX/EAX × OPRD → DST
  - **DST**: (AH:AL)/(DX:AX)/(EDX:EAX)
  - **OPRD**: REG/MEM

## 功能

- 实现两个**无符号数**的乘法运算
  - 乘数是 **OPRD**，被乘数位于 AL、AX 或 EAX 中 (由 **OPRD** 的尺寸决定，**乘数和被乘数的尺寸一致**)
  - 乘积尺寸翻倍：8 位乘积送到 AX；16 位乘积送 DX:AX；32 位乘积送 EDX:EAX

## 注意

- **OPRD** 可以是通用寄存器和存储单元，**不能**是立即数
- 仅对 **CF**、**OF** 标志位有影响，对其他标志位无定义

# 03 | 8086常用指令 - 乘法指令 - IMUL - 0

## 有符号数的乘法 IMUL (Signed Multiply) 指令

- IMUL (有符号数乘法) 指令执行有符号数乘法
- 与 MUL 指令不同, IMUL 会保留乘积的符号, 将乘积低半部分的最高位符号扩展到高半部分
- 如果乘积的高半部分不是其低半部分的符号扩展, 则 CF 和 OF 置 1, 执行了 IMUL 操作后, 必须检查这些标志位中的一个

## 有符号数的乘法 IMUL 指令格式

- IMUL <OPRD> ;单操作数
- IMUL <DST>, <SRC> ;双操作数

# 03 | 8086常用指令-乘法指令-IMUL-1

## 有符号数的乘法 IMUL 指令格式 1

- **IMUL** <OPRD> ;AL/AX/EAX × OPRD → DST
  - **DST**: (AH:AL)/(DX:AX)/(EDX:EAX)
  - **OPRD**: REG/MEM

## 功能

- 实现两个**有符号数**的乘法运算
  - 乘数是 **OPRD**，被乘数位于 AL、AX 或 EAX 中 (由 **OPRD** 的尺寸决定，**乘数和被乘数的尺寸一致**)

## 注意

- 被乘数和乘数均被视为**有符号数**
- 其他与无符号乘法指令 **MUL** 类似

# 03 | 8086常用指令-乘法指令-IMUL-2

## 有符号数的乘法 IMUL 指令格式 2

- **IMUL** <DST>, <SRC> ;  $DST \times SRC \rightarrow DST$ 
  - **DST**: REG
  - **SRC**: REG/MEM/IMM

## 功能

- 实现两个**有符号数**的乘法运算

## 注意

- **DST** 只能是 16 位或者 32 位**通用寄存器**
- **SRC** 可以是**通用寄存器**或**存储单元**（须与 **DST** 尺寸一致），可以是一个**立即数**（尺寸不能超过 **DST**）

# 03 | 8086常用指令 - 乘法指令对标志位的影响

## 乘法指令对标志位的影响

乘法指令	OF	SF	ZF	AF	PF	CF
MUL/IMUL	修改	无定义	无定义	无定义	无定义	修改

## CF 和 OF 的判断

单操作数高半部分: 8 位乘法的 AH  
16 位乘法的 DX  
32 位乘法的 EDX

- 对于 MUL 指令
  - 当乘积的高半部分为 0 时, CF=OF=0
  - 当乘积的高半部分不为 0 时, CF=OF=1
- 对于 IMUL 指令
  - 当乘积的高半部分为符号扩展时, CF=OF=0
  - 当乘积的高半部分不为符号扩展时, CF=OF=1

# 03 | 8086常用指令 - 乘法指令举例

## 乘法指令举例 1

```
1.  .DATA
2.      VAL1  WORD  2000H
3.      VAL2  WORD  0100H
4.  .CODE
5.  MOV AL, 05H
6.  MOV BL, 10H
7.  MUL BL           ; AX=0050H, CF=OF=0
8.  MOV AX, VAL1     ; AX=2000H
9.  MUL VAL2         ; DX:AX=0020 0000H, CF=OF=1
10. MOV AL, 30H
11. MOV BL, 04H
12. IMUL BL          ; AX=00C0H, CF=OF=1
13. MOV AX, 30H
14. MOV BX, 04H
15. IMUL BX          ; DX:AX=0000 00C0H, CF=OF=0
```

# 03 | 8086常用指令-除法指令-DIV

## 无符号数的除法 DIV (Divide) 指令格式

- $DIV \text{ <OPRD>}; AX/(DX:AX)/(EDX:EAX) \div OPRD \rightarrow DST$ 
  - $DST: (AH:AL)/(DX:AX)/(EDX:EAX)$
  - $OPRD: REG/MEM$

## 功能

- 实现两个**无符号数**的除法运算
  - 除数是  $OPRD$ ，被除数位于  $AX$ 、 $DX:AX$  或  $EDX:EAX$  中 (由  $OPRD$  的尺寸决定，**被除数的尺寸翻倍**)
  - 商和余数的尺寸与  $OPRD$  相同：商在  $AL$ 、 $AX$  或  $EAX$  中；余数在  $AH$ 、 $DX$  或  $EDX$  中

## 注意

- $OPRD$  可以是通用寄存器和存储单元，**不能**是立即数
- 除法指令**不影响**标志位，但需注意**除操作溢出**

# 03 | 8086常用指令 - 除法指令 - IDIV

## 有符号数的除法 IDIV (Signed Divide) 指令格式

- **IDIV** <OPRD> ;  $AX/(DX:AX)/(EDX:EAX) \div OPRD \rightarrow DST$ 
  - **DST**: (AH:AL)/(DX:AX)/(EDX:EAX)
  - **OPRD**: REG/MEM

## 功能

- 实现两个**有符号数**的除法运算
  - 商和余数的尺寸与 **OPRD** 相同：商在 AL、AX 或 EAX 中；余数在 AH、DX 或 EDX 中
  - **余数的符号与被除数一致**，且余数的绝对值小于除数的绝对值

## 注意

- 被除数和除数均被视为**有符号数**
- 其他与无符号乘法指令 **DIV** 类似



## 03 | 8086常用指令 - 除法指令对标志位的影响

### 除法指令对标志位的影响

除法指令	OF	SF	ZF	AF	PF	CF
DIV/IDIV	无定义	无定义	无定义	无定义	无定义	无定义

### 除法溢出 — 0 类中断

- 如果 **OPRD** 生成的商超出 **DST** 所表示的范围，则产生**除法溢出 (divide overflow)**，产生除法溢出会导致处理器异常并暂停执行当前程序
  - 商一般存在 AL、AX 或 EAX 中，若商的字长超过 8/16/32 位，则无法保存商，产生除法溢出
  - 可通过多个寄存器保存商的值解决这个问题
- 除数为 0，则除法无法计算

# 03 | 8086常用指令 - 除法指令举例1

## 除法指令举例 1

```
1. MOV AX, 0083H ;被除数
2. MOV BL, 2      ;除数
3. DIV BL         ;AL=41H, AH=01H
4.
5. MOV DX, 0      ;清除被除数高 16 位
6. MOV AX, 8003H ;被除数的低 16 位
7. MOV BX, 100H   ;除数
8. DIV BX         ;AX=0080H, DX=0003H
9.
10. MOV AX, 1000H
11. MOV BL, 10H
12. DIV BL        ;AL 无法容纳 100H, 产生除法溢出
```

# 03 | 8086常用指令-除法指令举例2

## 除法指令举例 2

1. `MOV AX, 0EFFCH` ;被除数的低字 (-4100D)
2. `CWD` ;AX 符号扩展到 DX, (DX:AX=FFFF EFFCH)
3. `MOV BX, 0100H` ;除数 (256D)
4. `IDIV BX` ;商AX=0FFF0H (-16D), 余数 DX=0FFFCH (-4D)
- 5.
6. `MOV AX, 1004H` ;被除数的低字 (4100D)
7. `CWD` ;AX 扩展到 DX, (DX:AX=0000 1004H)
8. `MOV BX, 0FF00H` ;除数 (-256D)
9. `IDIV BX` ;商 AX=0FFF0H (-16D), 余数 DX=0004H (4D)

# 03 | 8086常用指令 - 除法指令举例3

## 除法指令举例 3

1. `MOV AX, 0EFFCH` ;被除数的低字 (-4100D)
2. `CWD` ;AX 符号扩展到 DX, (DX:AX=FFFF EFFCH)
3. `MOV BX, 0100H` ;除数 (256D)
4. `IDIV BX` ;商AX=0FFF0H (-16D), 余数DX=0FFFCH (-4D)
- 5.
6. `MOV AX, 0EFFCH` ;被除数的低字 (-4100D)
7. `MOV DX, 0H` ;DX=0000H, (DX:AX=0000 EFFCH=61436D)
8. `MOV BX, 0100H` ;除数 (256D)
9. `IDIV BX` ;商 AX=00EFH (239D), 余数 DX=00FCH (252D)

- 如果不进行符号扩展, 除法操作结果将**出错!**

除法指令 **DIV BL** 中，被除数在寄存器（ ）中。

- ☒ A AX
- ☐ B DX
- ☐ C DX:AX
- ☐ D BX

## 03 | 8086常用指令-综合练习

- 编写程序段，完成  $(C-120+A\times B)/C$  的计算，并把所得的商和余数分别存入 X 和 Y 中，其中 A、B、C、X 和 Y 都是有符号的字变量。

解题思路：

① 先计算  $A\times B$

- 使用指令：  
`MOV AX, A`  
`IMUL B` ;得到32位积，存于 DX:AX

② 加上 C

- C 为 16 位数据，要加 32 位数据必须先进行符号扩展；
- 应执行指令：`CWD`

③ 再减 120

CWD 指令必须使用 DX、AX，故扩展之前需另存 DX:AX 中的数据

④ 然后除以 C

- 被除数已是 32 位数据，可直接使用指令：`IDIV C`

⑤ 最后保存结果

- 将 DX、AX 中的余数和商分别存入 Y 和 X 单元

## 03 | 8086常用指令-综合练习参考

- 编写程序段，完成  $(C-120+A \times B)/C$  的计算，并把所得的商和余数分别存入 X 和 Y 中，其中 A、B、C、X 和 Y 都是有符号的字变量。

不能使用指令 MOV  
DX, 0 扩展位数!

可否使用指令:

ADD CX, AX  
ADC BX, DX

```
1.  MOV AX, A
2.  IMUL B           ; (DX:AX)=A×B
3.  MOV CX, AX
4.  MOV BX, DX       ; (BX:CX)=A×B
5.  MOV AX, C
6.  CWD              ; (DX:AX)=C
7.  ADD AX, CX        {
8.  ADC DX, BX         ; (DX:AX)=(C+A×B)
9.  SUB AX, 120
10. SBB DX, 0         ; (DX:AX)=(C-120+A×B)
11. IDIV C            ; 商为AX, 余数为DX
12. MOV X, AX
13. MOV Y, DX        ; 保存结果
```

# 03 | 8086常用指令 - BCD调整指令

## BCD 码

17D

0001 0111B

00000001 00000111B

17H

0107H

- **定义**：用二进制数的形式表示的十进制数
- **压缩 BCD 码**：4 位二进制数表示 1 位十进制数
- **非压缩 BCD 码**：8 位二进制数表示 1 位十进制数

## 常用的 ASCII 码 (牢记)

- 数字字符
- 大小写字母字符
- 常用控制字符
  - 数字 < 大写字母 < 小写字母

十进制	十六进制	对应字符	相关字符
10	0A	换行	
13	0D	回车	
32	20	空格	
48	30	0	1~9
65	41	A	B~Z
97	61	a	b~z



# 03 | 8086常用指令 - BCD调整指令

## BCD 调整指令

- BCD 调整指令分为**压缩 (组合)** 的和**非压缩(未组合)** 的 BCD 调整指令
  - **压缩 BCD 调整指令**：对**加减**运算后的结果进行调整，产生一个压缩的 BCD 码
  - **非压缩 BCD 调整指令**：对**加减乘除**运算后的结果进行调整，产生一个非压缩的 BCD 码

## 使用 BCD 调整指令的原因

- 8086 CPU 指令集没有专门的适用于 BCD 码的运算指令，只有普通的二进制运算指令

$$35H + 35H = 6AH \neq 70H$$

$$70H - 38H = 38H \neq 32H$$

# 03 | 8086常用指令-压缩BCD调整指令-格式

## 压缩 BCD 调整指令

Decimal Adjust for Addition  
Decimal Adjust for Subtraction

- DAA/DAS

- DST: AL

- SRC: AL

## 功能

- DAA 指令对在 AL 中的和进行调整，产生一个压缩 BCD 码
- DAS 指令对在 AL 中的差进行调整，产生一个压缩 BCD 码

## 注意

- DAA 和 DAS 指令影响标志 AF, CF, PF, SF 和 ZF, 但不影响标志 OF

## 03 | 8086常用指令-压缩BCD调整指令-方法

### DAA 指令调整方法

- 如果 AL 中的低四位在 **A – F 之间**，或 **AF 为 1**，则  
 $AL \leftarrow (AL) + 6$ ，且 AF 位置 1
- 如果 AL 中的高四位在 **A – F 之间**，或 **CF 为 1**，则  
 $AL \leftarrow (AL) + 60H$ ，且 CF 位置 1

### DAS 指令调整方法

- 如果 AL 中的低四位在 **A – F 之间**，或 **AF 为 1**，则  
 $AL \leftarrow (AL) - 6$ ，且 AF 位置 1
- 如果 AL 中的高四位在 **A – F 之间**，或 **CF 为 1**，则  
 $AL \leftarrow (AL) - 60H$ ，且 CF 位置 1

## 03 | 8086常用指令-压缩BCD调整指令-示例

### 压缩 BCD 调整指令示例

```
1. MOV AL, 34H
2. ADD AL, 47H ; AL=7BH, AF=0, CF=0
3. DAA ; AL=81H, AF=1, CF=0
4. ADC AL, 87H ; AL=08H, AF=0, CF=1
5. DAA ; AL=68H, AF=0, CF=1
6. ADC AL, 79H ; AL=E2H, AF=1, CF=0
7. DAA ; AL=48H, AF=1, CF=1
```

- 低四位调整 (3, 7)
- 高四位调整 (5, 7)

# 03 | 8086常用指令-非压缩BCD调整指令-格式

## 非压缩 BCD 调整指令

- AAA/AAS/AAM/AAD

■ SRC: AL

■ DST: AX

Ascii Adjust for Addition  
Ascii Adjust for Subtraction  
Ascii Adjust for Multiplication  
Ascii Adjust for Division

## 功能

- AAA 指令用于对 ADD 指令运算后 AL 调整；AAS 指令用于对 SUB 指令运算后 AL 调整；AAM 指令用于对 MUL 指令运算后 AX 调整。
- AAD 指令用于调整 DIV 运算前 AH、AL 中除数，以便除法所得的商是有效的非压缩 BCD 数。

## 注意

- AAA 和 AAS 指令会影响 AF 和 CF，而对 OF, PF, SF 和 ZF 没有意义

# 03 | 8086常用指令-非压缩BCD调整指令-方法

## 以 AAA 指令调整方法为例

- **AAA** 这条指令对在 AL 中的和（由两个非压缩 BCD 码相加后的结果）进行调整，产生一个非压缩 BCD 码。调整方法如下：

① 如 AL 中的低 4 位在 **0-9 之间**，且 **AF 为 0**，则转 ③。

② 如 AL 中的低 4 位在 **A-F 之间**，或 **AF 为 1**，则  
 $AL \leftarrow (AL) + 6$ ， $AH \leftarrow (AH) + 1$ ，AF 位置 1。

③ 清除 AL 中的高 4 位。

④ AF 位的值送 CF 位。

# 03 | 8086常用指令-非压缩BCD调整指令-示例

## 非压缩 BCD 调整指令示例

```
1. MOV AX, 7
2. ADD AL, 6      ; AL=0DH, AH=00H, AF=0, CF=0
3. AAA           ; AL=03H, AH=01H, AF=1, CF=1
4. ADC AL, 6      ; AL=0AH, AH=01H, AF=0, CF=0
5. AAA           ; AL=00H, AH=02H, AF=1, CF=1
6. ADD AL, 39H    ; AL=39H, AH=02H, AF=0, CF=0
7. AAA           ; AL=09H, AH=02H, AF=0, CF=0
```

# 03 | 8086常用指令 - BCD调整指令对标志位的影响

## 压缩 BCD 指令对标志位的影响

指令	OF	SF	ZF	AF	PF	CF
DAA	无定义	修改	修改	检测并修改	修改	检测并修改
DAS	无定义	修改	修改	检测并修改	修改	检测并修改

## 非压缩 BCD 指令对标志位的影响

指令	OF	SF	ZF	AF	PF	CF
AAA	无定义	无定义	无定义	检测并修改	无定义	修改
AAS	无定义	无定义	无定义	检测并修改	无定义	修改
AAD	无定义	修改	修改	无定义	修改	无定义
AAM	无定义	修改	修改	无定义	修改	无定义



# 03 | 8086常用指令-逻辑指令-格式

## 逻辑指令格式

- 逻辑 与: `AND <DST>, <SRC>`      ■ `DST`: REG/MEM
- 逻辑 或: `OR <DST>, <SRC>`      ■ `SRC`: REG/MEM/IMM
- 逻辑异或: `XOR <DST>, <SRC>`
- 逻辑 非: `NOT <DST>`
- 测试指令: `TEST <DST>, <SRC>`

## 注意

- `NOT` 指令对所有标志位**无影响**；其它指令，置 **CF=OF=0**，正常影响 **SF** 和 **ZF**
- `SRC` 和 `DST` 不能同时为存储单元
- 段寄存器不能进行逻辑运算

# 03 | 8086常用指令 - 逻辑指令对标志位的影响

## 逻辑指令对标志位的影响

逻辑指令	OF	SF	ZF	AF	PF	CF
AND	置 0	修改	修改	无定义	修改	置 0
OR	置 0	修改	修改	无定义	修改	置 0
XOR	置 0	修改	修改	无定义	修改	置 0
TEST	置 0	修改	修改	无定义	修改	置 0
NOT	不影响	不影响	不影响	不影响	不影响	不影响

## 03 | 8086常用指令-逻辑与指令的应用

若  $AL=0B7H$ ，则 **AND AL, 0F8H** 的执行过程为

	1011	0111	=	0B7H
AND	1111	1000	=	0F8H
<hr/>				
	1011	0000	=	0B0H

两位同时为 1，逻辑与结果为 1；  
任意一位为 0，结果为 0

$CF=0$ ， $SF=1$ ， $OF=0$ ， $ZF=0$

将 BL 中的 D3 和 D0 位清零，其他位不变

- 利用逻辑与指令设定源操作数：清零位为 0，其他位为 1，即 1111 0110B

**AND BL, 0F6H**

- 逻辑与指令常用于将一个操作数 (寄存器) 的某些位清零

## 03 | 8086常用指令-逻辑或指令的应用

若 AL=61H, 则 OR AL, 78H 的执行过程为

	0110	0001	=	61H
OR	0111	1000	=	78H
<hr/>				
	0111	1001	=	79H

任意一位为 1, 逻辑或结果为 1;  
两位同时为 0, 结果为 0

CF=0, SF=0, OF=0, ZF=0

将 BL 中的第 1、3、4、6 位置 1, 其他位不变

- 利用逻辑或指令设定源操作数: 置位位为 1, 其他位为 0, 即 0101 1010B

OR BL, 5AH

- 逻辑或指令常用于将一个操作数 (寄存器) 的某些位置 1

## 03 | 8086常用指令-逻辑异或指令的应用

若 AL=61H, 则 XOR AL, 78H 的执行过程为

$$\begin{array}{rcl} & 0110 & 0001 = 61H \\ \text{XOR} & 0111 & 1000 = 78H \\ \hline & 0001 & 1001 = 19H \end{array}$$

两位不同, 逻辑异或结果为 1;  
两位相同, 结果为 0

CF=0, SF=0, OF=0, ZF=0

将 BL 中的低四位取反, 其他位不变

- 利用逻辑异或指令设定源操作数: 取反位为 1, 其他位为 0, 即 0000 1111B

XOR BL, 0FH

- 逻辑异或指令常用于将一个操作数 (寄存器) 的某些位取反或所有位清零

## 03 | 8086常用指令-逻辑非指令的应用

若  $AL=61H$ ，则 **NOT AL** 的执行过程为

$$\begin{array}{r} \text{NOT} \quad 0110 \ 0001 = 61H \\ \hline 1001 \ 1110 = 9EH \end{array}$$

取操作数的反码；  
取补码操作是 **NEG**

**NOT 与 NEG**

1. **MOV AL, 61H** ;  $AL=0110 \ 0001B=97D$
2. **NOT AL** ;  $AL=1001 \ 1110B=-98D$
3. **NEG AL** ;  $AL=1001 \ 1111B=-97D$

试使用四种不同的方法将 **AX** 的内容清零

- **MOV 指令:** **MOV AX, 0**
- **AND 指令:** **AND AX, 0**
- **XOR 指令:** **XOR AX, AX**
- **SUB 指令:** **SUB AX, AX**

# 03 | 8086常用指令-测试指令的功能

## 测试 TEST 指令功能

- 通过指令执行后 **ZF 标志位** 的状态, 判断 **DST** 的某一位是 1 还是 0; 或者 **DST** 的某几位是否同时为 0

## 注意

- **TEST** 指令与 **AND** 指令操作相同, 但不会更改 **DST**, 只根据**测试位**改变标志位的值 (类似于 **CMP** 指令和 **SUB** 指令的关系)

## 03 | 8086常用指令-测试指令的应用

### 试判断 AL 的最低位是否为 1

- 利用**测试**指令设定源操作数：**测试位为 1，其他位为 0**，即 0000 0001B

**TEST AL, 01H**

- 指令执行后，若 ZF=0，则 AL 最低位为 1；否则，AL 最低位为 0

### 试判断 AL 的低 4 位是否同时为 0

- 利用**测试**指令设定源操作数：**测试位为 1，其他位为 0**，即 0000 1111B

**TEST AL, 0FH**

- 指令执行后，若 ZF=1，则 AL 低 4 位全为 0；否则，AL 低 4 位不全为 0



以下指令能对 **AX** 的内容模 8 的是 ( )

- ☐ A **OR** **AX**, 08H
- ☐ B **XOR** **AX**, 07H
- ☒ C **AND** **AX**, 07H
- ☐ D **TEST** **AX**, 08H

# 03 | 8086常用指令-移位指令

## 移位指令格式

- 逻辑移位: `SHL/SHR <DST>, <COUNT>`
- 算术移位: `SAL/SAR <DST>, <COUNT>`
- 循环移位: `ROL/ROR <DST>, <COUNT>`
- 带进位的循环移位: `RCL/RCR <DST>, <COUNT>`
  - `DST`: REG/MEM
  - `COUNT`: 1/CL (CX 低 8 位所表示的数)

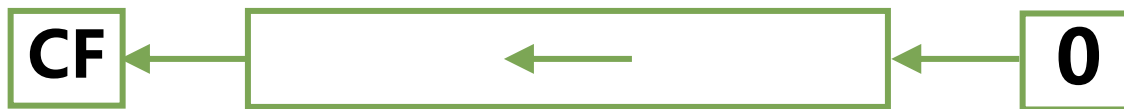
## 注意

- `DST` 可以是通用寄存器和存储单元内容
- `COUNT` 是 `DST` 移动的位数, 移 1 位 `COUNT` 为 1, 移多位 `COUNT` 可以由 CL 寄存器的值间接给出

# 03 | 8086常用指令 - 逻辑移位指令

## 逻辑左移 SHL (Shift Logical Left) 指令

SHL <DST>, <COUNT>



最高位移入 CF, 最低位补 0

## 逻辑右移 SHR (Shift Logical Right) 指令

SHR <DST>, <COUNT>



最低位移入 CF, 最高位补 0

# 03 | 8086常用指令-逻辑移位指令功能

## 逻辑移位指令功能

- 逻辑移位指令可用作**无符号数**的**乘除 2** 操作
  - 逻辑**左移一位**相当于无符号数**乘以 2**
  - 逻辑**右移一位**相当于无符号数**除以 2**

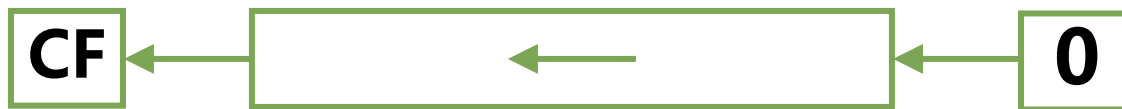
例如：设 AH=12H=18D， BL=0A9H=169D

- SHL AH, 1      00100100      (AH)=24H=36D      CF=0
- SHR AH, 1      00001001      (AH)=09H=09D      CF=0
- SHL BL, 1      01010010      (BL)=52H=82D      CF=1
- SHR BL, 1      01010100      (BL)=54H=84D      CF=1

# 03 | 8086常用指令 - 算术移位指令

算术左移 SAL (Shift Arithmetic Left) 指令

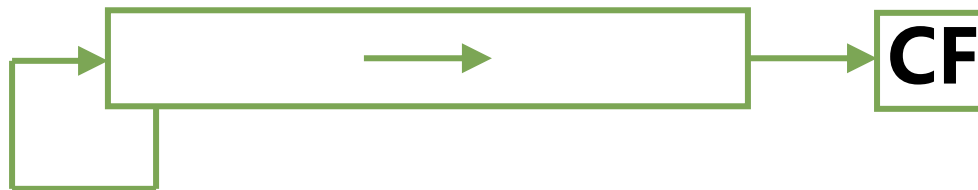
SAL <DST>, <COUNT>



最高位移入 CF, 最低位补 0

算术右移 SAR (Shift Arithmetic Right) 指令

SAR <DST>, <COUNT>



最低位移入 CF, 最高位不变

# 03 | 8086常用指令-算术移位指令功能

## 算术移位指令功能

- 逻辑移位指令可用作**有符号数**的**乘除 2** 操作
  - 算术**左移一位**相当于有符号数**乘以 2**
    - ✓ **只要符号位不改变**, 结果均正确
  - 算术**右移一位**相当于有符号数**除以 2**

例如: 设  $AH=12H=18D$ ,  $BL=0A9H=-87D$

- `SAL AH, 1`      `00100100`       $(AH) = 24H = 36D$        $CF=0$
- `SAR AH, 1`      `00001001`       $(AH) = 09H = 09D$        $CF=0$
- `SAL BL, 1`      `01010010`       $(BL) = 52H = 82D$        $CF=1$
- `SAR BL, 1`      `11010100`       $(BL) = 0D4H = -44D$        $CF=1$

# 03 | 8086常用指令-对标志位的影响

## 逻辑、算术移位指令对标志位的影响

逻辑、算术移位指令	OF	SF	ZF	AF	PF	CF
SAL/SAR/SHL/SHR 1	修改	修改	修改	无定义	修改	修改
SAL/SAR/SHL/SHR count	无定义	修改	修改	无定义	修改	修改

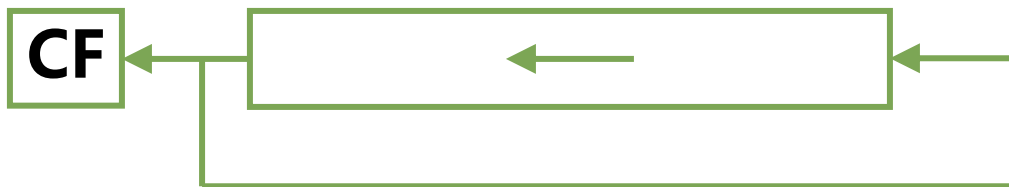
## 对标志位的影响

- CF：按照数据的移出位设置
- OF：
  - 当移位次数为 1 时，若移位操作改变了操作数的最高位——符号位时，OF=1，否则 OF=0
  - 当移位次数不为 1 时，无定义
- SF、ZF：根据移位后的结果影响

# 03 | 8086常用指令 - 循环移位指令

## 循环左移 ROL (Rotate Left) 指令

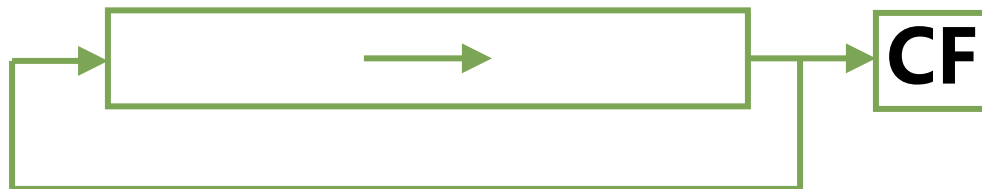
ROL <DST>, <COUNT>



最高位移入 CF，同时移入最低位

## 循环右移 ROR (Rotate Right) 指令

ROR <DST>, <COUNT>



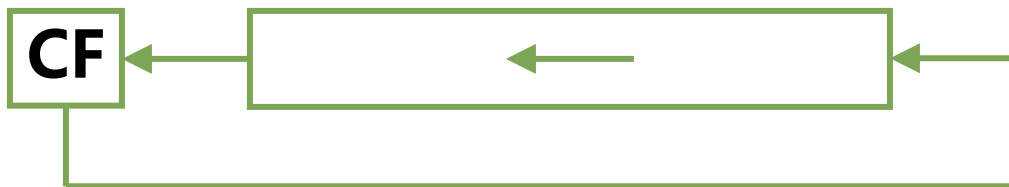
最低位移入 CF，同时移入最高位



# 03 | 8086常用指令 - 带进位的循环移位指令

带进位的循环左移 RCL (Rotate Through Carry Left) 指令

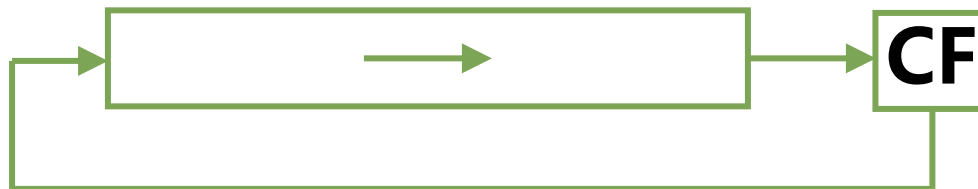
RCL <DST>, <COUNT>



CF 移入最低位, 最高位移入 CF

带进位的循环右移 RCR (Rotate Through Carry Right) 指令

RCR <DST>, <COUNT>



CF 移入最高位, 最低位移入 CF

# 03 | 8086常用指令-对标志位的影响

## 循环移位指令对标志位的影响

循环移位指令	OF	SF	ZF	AF	PF	CF
ROL/ROR 1	修改	无影响	无影响	无影响	无影响	修改
ROL/ROR count	无定义	无影响	无影响	无影响	无影响	修改
RCL/RCR 1	修改	无影响	无影响	无影响	无影响	检测并修改
RCL/RCR count	无定义	无影响	无影响	无影响	无影响	检测并修改

## 对标志位的影响

- CF: 按照数据的移出位设置
- OF:
  - 当移位次数为 1 时, 若移位操作改变了操作数的最高位——符号位时, OF=1, 否则 OF=0
  - 当移位次数不为 1 时, 无定义

# 03 | 8086常用指令-移位操作

数据 82H

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

CF

逻辑左移 SHL

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

逻辑右移 SHR

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

算术左移 SAL

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

算术右移 SAR

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

循环左移 ROL

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

循环右移 ROR

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

带进位的  
循环左移 RCL

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

1

带进位的  
循环右移 RCR

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0

# 03 | 8086常用指令-逻辑、算术移位指令-示例

## 逻辑、算术移位指令示例

			C	S	Z	O
1. MOV CL, 4						
2. MOV AL, 0F0H	; (AL) = 1111 0000B = 0F0H					
3. SHL AL, 1	; (AL) = 1110 0000B = 0E0H	1	1	0	0	
4. SHR AL, 1	; (AL) = 0111 0000B = 70H	0	0	0	1	
5. SAR AL, 1	; (AL) = 0011 1000B = 38H	0	0	0	0	
6. SAR AL, CL	; (AL) = 0000 0011B = 03H	1	0	0	x	

- 逻辑左移一位，最高位进 CF (3)
- 逻辑右移一位，最高位补 0，**改变了最高位符号**，低位移入 CF (4)
- 算术右移一位，高位补 0，低位移入 CF (5)
- 算术右移四位，高位补 0，低位移入 CF (6)

# 03 | 8086常用指令-循环移位指令-示例

## 循环移位指令示例

- 例如：设 AH=89H, CL=02H, CF=1

ROL AH, 1

0 0 0 1 0 0 1 1

CF=1

(AH) = 13H

ROR AH, CL

0 1 1 0 0 0 1 0

CF=0

(AH) = 62H

RCL AH, CL

0 0 1 0 0 1 1 1

CF=0

(AH) = 27H

RCR AH, 1

1 1 0 0 0 1 0 0

CF=1

(AH) = 0C4H

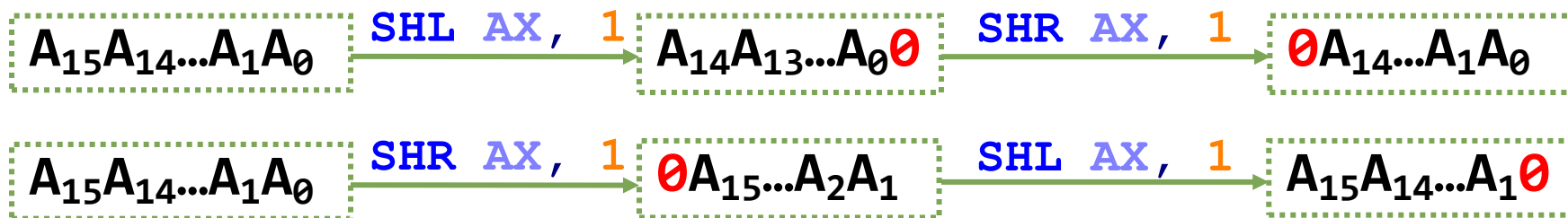
## 03 | 8086常用指令-移位指令相关问题

执行以下两组指令序列后，结果是否一定相同？

```
SHL AX, 1  
SHR AX, 1
```

```
SHR AX, 1  
SHL AX, 1
```

✎ 答：不一定相同，因为逻辑左右移指令均会改变了原操作数的值



若是以下两组指令序列呢？

```
RCL AX, 1  
RCR AX, 1
```

```
RCR AX, 1  
RCL AX, 1
```

## 03 | 8086常用指令-移位指令计算

算术右移一位相当于带符号数的除 2 操作

```
MOV AL, -15
```

```
SAR AL, 1
```

- 结果 AL=-8

可使用等价的除法指令

```
MOV AL, -15
```

```
MOV BL, 2
```

```
IDIV BL
```

- 结果 AL=-7

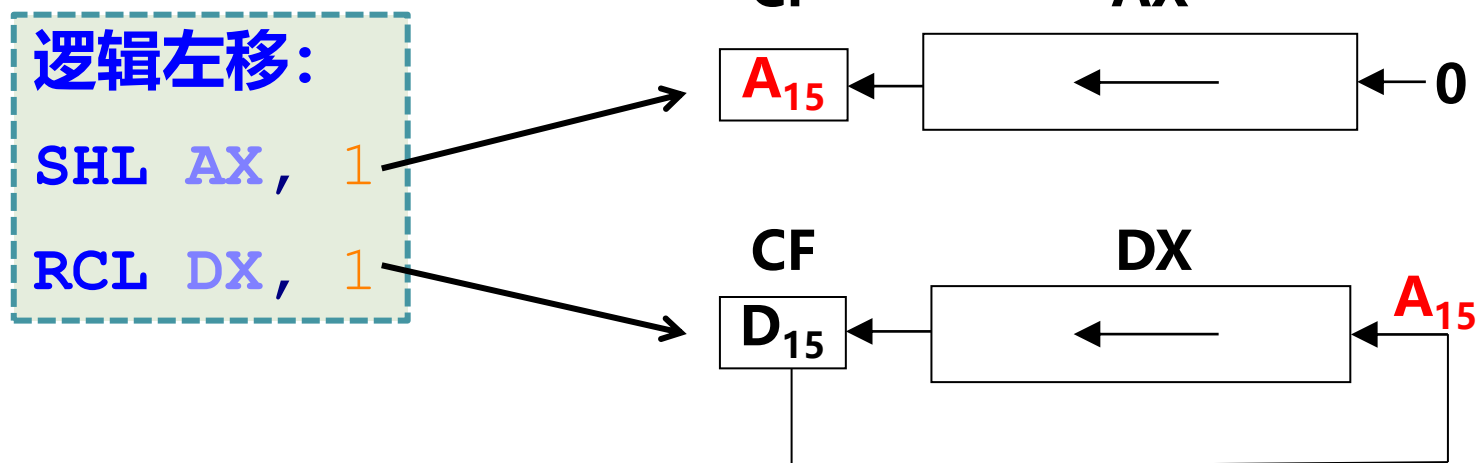
思考两个结果不一致的原因!

除法指令要求：  
余数与被除数符号相同!

# 03 | 8086常用指令-移位指令练习1

## 32 位组合数据移位

- 编写指令序列，完成 (DX:AX) 中 32 位数据的逻辑左移 1 位




指令练习



## 03 | 8086常用指令-移位指令练习2

用加减法和移位指令实现：  $(DX) = (AX) \times 3 + (BX) \times 7$

- 假设数据为无符号数，结果仍为 16 位，不溢出
- 编程思想：

 **逻辑左移**指令可实现无符号数的乘  $2^n$  的操作

$$(AX) \times 3 = (AX) \times 2^2 - (AX)$$

AX 先逻辑左移 2 位，结果再减去 AX 的值

$$(AX) \times 3 = (AX) \times 2 + (AX)$$

AX 先逻辑左移 1 位，结果再加上 AX 的值

$$(BX) \times 7 = (BX) \times 2^3 - (BX)$$

BX 先逻辑左移 3 位，结果再减去 BX 的值

## 03 | 8086常用指令-移位指令练习2参考

用加减法和移位指令实现:  $(DX) = (AX) \times 3 + (BX) \times 7$

```
1. MOV DX, AX      ;暂存 AX
2. SHL AX, 1
3. ADD DX, AX       ;(DX)=(AX)×3
4. MOV SI, BX       ;暂存 BX
5. MOV CL, 3
6. SHL BX, CL       ;(BX)=(BX)×8
7. SUB BX, SI       ;(BX)=(BX)×7
8. ADD DX, BX       ;(DX)=(AX)×3+(BX)×7
```

**SHL** 指令移位位数大于 1 时，需将移位位数放在寄存器（ ）中。

☐ A AL

☐ B BL

☒ C CL

☐ D DL

# 03 | 8086常用指令-处理器指令

## 标志位操作指令

- 一组无操作数指令，共 7 条指令，用于将 **CF**、**IF**、**DF** 标志位进行置 1 或清零操作

指令	英文全称	指令功能
CLC	<b>C</b> lear <b>C</b> arry	复位进位标志：CF←0
STC	<b>S</b> et <b>C</b> arry	置位进位标志：CF←1
CMC	<b>C</b> omplement <b>C</b> arry Flag	求反进位标志：CF←~CF
CLD	<b>C</b> lear <b>D</b> irection Flag	复位方向标志：DF←0
STD	<b>S</b> et <b>D</b> irection Flag	置位方向标志：DF←1
CLI	<b>C</b> lear <b>I</b> nterrupt Flag	复位中断标志：IF←0
STI	<b>S</b> et <b>I</b> nterrupt Flag	置位中断标志：IF←1

# 03 | 8086常用指令-处理器指令

## 同步与控制指令、暂停和空操作指令

- **同步与控制指令**：用于与其它协处理器同步和控制的指令，共 3 条指令
- **暂停和空操作指令**：使 CPU 进入暂停状态或空闲状态

指令	英文全称	指令功能
ESC	Escape	为某协处理器提供对数据总线的访问，协处理器开始工作
WAIT	Wait	CPU 进入等待状态，直到协处理器发出已完成操作的信号
LOCK	Lock Bus	这条指令是一个前缀，在执行下一条指令时禁止其他处理器占用总线
HLT	Halt CPU	使 CPU 暂停执行后续指令
NOP	No Operation	执行空操作

# 阶段小结

---

- 掌握 MUL/IMUL、DIV/IDIV、AND/OR/XOR/NOT/TEST、SHL/SHR/SAL/SAR/ROL/ROR/RCL/RCR
- 会使用 DAA/DAS/AAA、CLC/STC/CMC/CLD/STD/CLI/STI
- 掌握各指令的**格式、功能、注意事项、操作数的类型、指令执行后对标志位的影响**
- 掌握加减乘除等基本运算程序的编写

## 03 | 上节回顾-汇编语言程序编写相关概念

### 简答题

- 在汇编语言程序设计中，常量如何定义？变量如何定义？
- 变量和标号的属性有哪些？它们分别指的是什么？
- 常用的系统功能调用有哪些？他们的入口参数和出口参数分别是什么？

# 03 | 8086常用指令-串操作指令

## 串操作指令

教材 4.10.1 P162

- 一般情况下，串操作处理的数据不只是一个字节或字，而是在内存中地址**连续的字节串或字串**
- 只有加上**重复前缀** REP 才能实现对整个串的操作
- 串操作指令包括
  - **串传送指令**：MOVSB/MOVSW
  - **串比较指令**：CMPSB/CMPSW
  - **串扫描指令**：SCASB/SCASW
  - **串装入指令**：LODSB/LODSW
  - **串存储指令**：STOSB/STOSW
  - **重复前缀** : REP  
REPE/REPZ  
REPNE/REPNZ



# 03 | 8086常用指令-重复前缀指令

## 重复前缀 REP 指令格式

Repeat String Operation  
Repeat Equal / Repeat Zero  
Repeat Not Equal / Repeat Not Zero

- 无条件重复前缀指令: REP (CX≠0)
- 有条件重复前缀指令: REPE/REPZ (CX≠0 & ZF=1)
- 有条件重复前缀指令: REPNE/REPZ (CX≠0 & ZF=0)

## 功能

- 这 3 中重复前缀指令不能单独使用, 只能加在串操作指令之前, 用来控制重复执行串操作
- 重复前缀不影响标志位

# 03 | 8086常用指令-串传送指令

## 串传送 MOVSB 指令格式

Move String (Byte or Word)

- REP MOVSB/MOVSW

- SRC: [DS:SI]

- DST: [ES:DI]

## 功能

- 将由 DS:SI 所指向的存储单元的**字节或字**传送到由 ES:DI 所指向的存储单元

## 注意

- 该指令对标志位**无影响**
- 一般带无条件的**重复前缀**执行串操作，其过程往往相当于执行一个**循环程序**

# 03 | 8086常用指令-串传送指令说明

## 串传送指令说明

- CX 寄存器中事先存放好传送的字节数或字数
- 由方向标志确定**串传送**的方向
  - DF=0, 用 **MOVSB** 指令时, 每传送一次, 地址指针 SI 和 DI 自动增 1, 用 **MOVSW** 指令时, 每传送一次, 地址指针 SI 和 DI 自动增 2
  - 若 DF=1, 则地址指针自动减 1 或自动减 2
- 对带 **REP** 重复前缀的串传送指令来说, 每传送一次, CX 中的计数值总是减 1

# 03 | 8086常用指令-串传送指令-举例

## 串传送指令示例

1. **MOV SI, 2000H** ;源地址为 2000H
2. **MOV DI, 3000H** ;目的地址为 3000H
3. **MOV CX, 100** ;字符串长 100 个字节
4. **CLD** ;方向标志清 0, 使指针按增量方向修改
5. **REP MOVSB** ;将源地址开始的 100 个字节传送到目的地址

# 03 | 8086常用指令-串比较指令

## 串比较 CMPS 指令格式

Compare String (Byte or Word)

- REPZ CMPSB/CMPSW

- SRC: [DS:SI]

- DST: [ES:DI]

- REPNZ CMPSB/CMPSW

- SRC: [DS:SI]

- DST: [ES:DI]

## 功能

- 把 DS:SI 所指向的存储单元的**字节或字**与 ES:DI 所指向的存储单元的字节或字逐一比较，并设置相应标志位

## 注意

- 该指令对标志位均**有影响**
- 一般带有条件的**重复前缀**

# 03 | 8086常用指令-串比较指令说明

## 串比较指令说明

- CX 寄存器中事先存放好要比较的字节数或字数
- 由方向标志确定**串比较**的方向
  - DF=0, 每比较一次, SI 和 DI 加 1 或 2, CX 减 1
  - DF=1, 每比较一次, SI 和 DI 减 1 或 2, CX 减 1
- CMPSB/CMPSW 指令的前缀可以有 REPNZ/REPNE 或 REPZ/REPE
- 利用 CMPSB/CMPSW, 可以实现在两个字符串中寻找第一个不相等的元素 (使用 REPZ/REPE) 或者第一个相等的元素 (使用 REPNZ/REPNE)
- **注意**: ZF 并不因为 CX 在串操作过程中不断减 1 而受影响

# 03 | 8086常用指令-串扫描指令

## 串扫描 SCAS 指令格式

Scan String (Byte or Word)

- REPZ SCASB/SCASW

- SRC: AL/AX

- DST: [ES:DI]

- REPNZ SCASB/SCASW

- SRC: AL/AX

- DST: [ES:DI]

## 功能

- 将 AL/AX 的**字节或字**内容与 ES:DI 所指向的存储单元的字节或字逐一比较，并设置相应标志位

## 注意

- 该指令对标志位均**有影响**
- 一般带有条件的**重复前缀**

# 03 | 8086常用指令-串扫描指令说明

## 串扫描指令说明

- CX 寄存器中事先存放好要扫描的字节数或字数
- 由方向标志确定**串扫描**的方向
  - DF=0, 每扫描一次, DI 加 1 或 2, CX 减 1
  - DF=1, 每扫描一次, DI 减 1 或 2, CX 减 1
- SCASB/SCASW 指令的前缀可以有 REPNZ/REPNE 或 REPZ/REPE
- 利用 SCASB/SCASW, 可以实现在由 ES:DI 所指的字符串中寻找第一个与 AL/AX 不相等的元素 (使用 REPZ/REPE) 或者第一个与 AL/AX 相等的元素 (使用 REPNZ/REPNE)
- **注意**: ZF 并不因为 CX 在串操作过程中不断减 1 而受影响



# 03 | 8086常用指令-串装入指令

## 串装入 LODS 指令格式

Load String (Byte or Word)

- LODSB/LODSW

- SRC: [DS:SI]

- DST: AL/AX

## 功能

- 将 DS:SI 所指的存储单元的**字节或字**内容装入 (提取) 到 AL/AX 中, 因此该指令也称为**取字符串指令**

## 注意

- 该指令对标志位**无影响**
- 一般**不带**重复前缀

# 03 | 8086常用指令-串装入指令说明和举例

## 串装入指令说明

- CX 寄存器中事先存放好要提取的字节数或字数
- 由方向标志确定源字符串**串提取**的方向
  - DF=0, 每装入一次, SI 加 1 或 2
  - DF=1, 每装入一次, SI 减 1 或 2

## 串装入指令举例

```
1.      CLD                ;方向标志清 0
2.      MOV SI, 0700H      ;SI 作为地址指针
3.      MOV CX, 5          ;共处理 5 个字节
4.  LI:  LODSB             ;取 1 个字节到 AL 中, 并使地址增 1
5.      .....            ;处理字符
6.      MOV [SI-1], AL     ;送回处理结果
7.      DEC CX             ;计数值减 1
8.      JNZ LI             ;如未处理完, 则继续
```

# 03 | 8086常用指令-串存储指令

## 串存储 STOS 指令格式

Store String (Byte or Word)

- STOSB/STOSW

- SRC: AL/AX

- DST: [ES:DI]

## 功能

- 把 AL/AX 的内容存放到 ES:DI 所指的存储单元中，因此该指令也称为存字符串指令

## 注意

- 该指令对标志位无影响
- 一般带无条件重复前缀

# 03 | 8086常用指令-串存储指令说明和举例

## 串存储指令说明

- CX 寄存器中事先存放好要存储的字节数或字数
- 由方向标志确定源字符串存储的方向
  - DF=0, 每装入一次, DI 加 1 或 2, CX 减 1
  - DF=1, 每装入一次, DI 减 1 或 2, CX 减 1

## 串存储指令举例

- 使 ES 段 0404H 开始的 256 个单元清 0

1. CLD	;清除方向标志
2. LEA DI, [0404H]	;将目的地址 0404H 送 DI
3. MOV CX, 0080H	;共有 128 个串
4. XOR AX, AX	;AX 清 0
5. REP STOSW	;将 256 个字节 (128个字) 清 0

# 03 | 8086常用指令-串操作指令总结1

## 串操作指令总结

- 8086 的串操作指令的特点是，通过加重重复前缀实现串操作 (LODSB/LODSW 除外)
- 所有的串操作指令都用寄存器 SI、DI 操作数进行间接寻址
  - 用寄存器 SI 对源操作数进行间接寻址，并且规定在 DS 段中
  - 用寄存器 DI 为目的操作数进行间接寻址，并规定在 ES 段中
- 串操作时，地址的修改往往与方向标志 DF 有关

03

8086常用指令-串操作指令总结2

串操作指令总结

指令	重复前缀	对标志位的影响	操作数	地址指针寄存器
MOVS	REP	无影响	目标	[ES:DI]
			源	[DS:SI]
CMPS	REPE/REPZ	有影响	目标	[ES:DI]
			源	[DS:SI]
SCAS	REPE/REPZ	有影响	目标	[ES:DI]
			源	AL/AX
LODS	无	无影响	目的	AL/AX
			源	[DS:SI]
STOS	REP	无影响	目标	[ES:DI]
			源	AL/AX

# 03 | 8086常用指令-程序控制指令

## 程序控制指令

- 程序控制指令也称**控制转移指令**，共包括 4 类指令：
  - 无条件转移和条件转移指令：教材 4.8.1 P150
  - 循环控制指令：教材 4.9.1 P157
  - 子程序调用和返回指令：教材 4.11.1 P167
  - 中断指令：教材 8.1.2 P289
- 这类指令共同特点是**打乱了顺序程序结构**

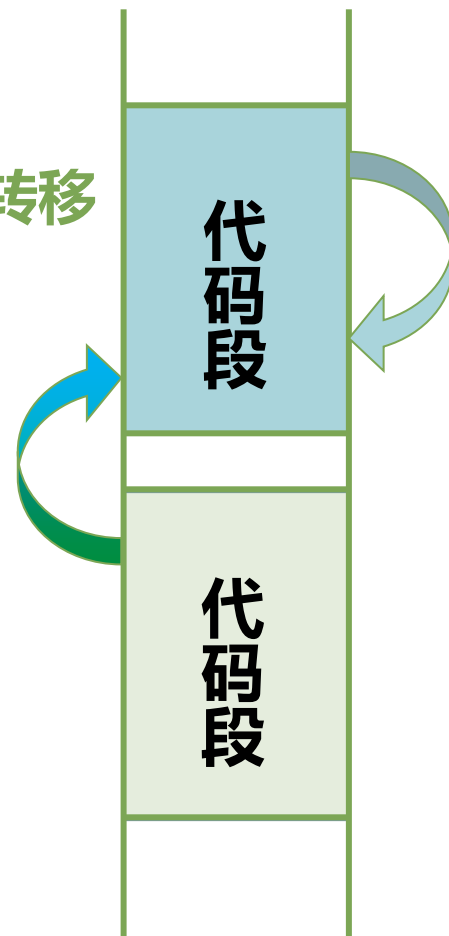
# 03 | 8086常用指令 - 跳跃寻址的转移类型

## 转移分类

- 根据转移的范围分为：**段内转移**和**段间转移**
- 根据操作数寻址方式分为：**直接转移**和**间接转移**

## 转移类型

- **段内转移**：在同一代码段内转移，只修改 IP，CS 不变
  - 段内直接转移
  - 段内间接转移
- **段间转移**：在不同代码段之间的转移，同时修改 CS 和 IP
  - 段间直接转移
  - 段间间接转移





# 03 | 8086常用指令-无条件转移指令

## 无条件转移 JMP (Unconditional Jump) 指令格式

● **JMP** <OPRD>

■ **OPRD**: REG/MEM/IMM/Label (表示的地址信息)

### 功能

- 使程序**无条件地**转向 **OPRD** 所指定的位置执行

### 注意

- **OPRD** 可用寄存器寻址和存储器寻址方式表示
- 条件转移指令和无条件转移指令对标志位**无影响**
- 无条件转移指令又分为:
  - 段内转移: 短转移 (Short)、近转移 (Near)
  - 段间转移: 远转移 (Far)

## 03 | 8086常用指令-段内直接转移

### 指令格式

- 短转移: `JMP SHORT <LABEL>`
- 近转移: `JMP NEAR PTR <LABEL>`

### 功能

- 使程序**无条件地**转向 `LABEL` 所指定的位置执行

### 转移的范围

- 短转移: 8 位偏移量 (-128~+127)
- 近转移: 16 位偏移量 (-32768~+32767)

# 03 | 8086常用指令-段内直接转移说明

## 段内直接转移说明

- 一般，**可以省去类型说明符**，直接使用标号
  - 例如：JMP EXIT
  - 由汇编程序自动判断是**短转移**，还是**近转移**
- **段内直接转移指令**，在 DEBUG 下，操作数都是**偏移量的形式**，而不是直接地址
  - 可查看对应的**机器指令**确定偏移量的位数
- 指令的执行过程：CS 不变，**IP+偏移量 → IP**

# 03 | 8086常用指令-段内间接转移

## 指令格式

- 寄存器寻址: `JMP REG`
- 存储器寻址: `JMP NEAR PTR MEM`

## 功能

- 使程序**无条件地**转向操作数所指定的位置执行
  - 指令的执行过程: CS 不变, REG、MEM 的字数据 → IP

## 转移的范围

- 16 位偏移量 (-32768~+32767) 构成的存储单元 (整个逻辑段)
  - 例如: (BX)=2010H, [2010H]=0FA34H  
则执行 `JMP NEAR PTR [BX]` 之后, (IP)=0FA34H

# 03 | 8086常用指令-无条件转移指令举例

## 无条件转移指令示例

1. <b>JMP</b> 1000H	; 段内直接转移
2. <b>JMP</b> CX	; 段内间接转移
3. <b>JMP</b> 2000H:0100H	; 段间直接转移
4. <b>JMP</b> DWORD PTR [SI]	; 段间间接转移

## 注意

- (1) 段内直接转移，转移地址的偏移量由指令给出
- (2) 段内间接转移，转移地址由 CX 指出
- (3) 段间直接转移，段地址和偏移量由指令给出
- (4) 段间间接转移，段地址和偏移量放在 SI、SI+1、SI+2、SI+3 这 4 个单元中，前 2 个单元的内容作为偏移量，后 2 个单元的内容作为段地址

# 03 | 8086常用指令-条件转移指令

## 条件转移 Jxx (Conditional Jump) 指令格式

● Jxx <OPRD>

■ OPRD: REG/MEM/IMM/Label  
(表示的地址信息)

## 功能

- 指令是**根据其要求条件的满足**情况决定是否发生转移的

## 注意

- 条件转移指令只能完成**段内短转移**，即偏移量为 8 位的
  - 若超出该范围，则可用多条转移指令完成
- 条件转移指令的类型
  - 无符号数比较的转移指令
  - 有符号数比较的转移指令
  - 单个标志位测试的转移指令

# 03 | 8086常用指令-无符号数比较的转移指令

## 无符号数比较的转移指令

指令	英文全称	检测的转移条件	功能描述
JE/JZ	Jump if Equal Jump if Zero	ZF=1	等于，则转移
JNE/JNZ	Jump if Not Equal Jump if Not Zero	ZF=0	不等，则转移
JA/JNBE	Jump if Above Jump if Not Below or Equal	CF=0 且 ZF=0	大于（不小于等于），则转移
JAE/JNB	Jump if Above or Equal Jump if Not Below	CF=0	大于等于（不小于），则转移
JB/JNAE	Jump if Below Jump if Not Above or Equal	CF=1	小于（不大于等于），则转移
JBE/JNA	Jump if Below or Equal Jump if Not Above	CF=1 或 ZF=1	小于等于（不大于），则转移

- A — 大于； B — 小于； E/Z — 等于； N — 否

# 03 | 8086常用指令 - 有符号数比较的转移指令

## 有符号数比较的转移指令

指令	英文全称	检测的转移条件	功能描述
JE/JZ	Jump if Equal Jump if Zero	ZF=1	等于，则转移
JNE/JNZ	Jump if Not Equal Jump if Not Zero	ZF=0	不等，则转移
JG/JNLE	Jump if Greater Jump if Not Less or Equal	ZF=0 且 SF=OF	大于（不小于等于），则转移
JGE/JNL	Jump if Greater or Equal Jump if Not Less	SF=OF	大于等于（不小于），则转移
JL/JNGE	Jump if Less Jump if Not Greater or Equal	SF≠OF	小于（不大于等于），则转移
JLE/JNG	Jump if Less or Equal Jump if Not Greater	ZF=1或SF≠OF	小于等于（不大于），则转移

- G — 大于； L — 小于； E/Z — 等于； N — 否



## 03 | 8086常用指令 - JGE指令的判断条件

JGE 为带符号数大于等于测试转移指令，可分 4 种情况考虑

- 正数 A - 正数 B，为“正+负”的情况，OF=0
  - 若  $A \geq B$ ，则 SF=0；若  $A < B$ ，则 SF=1
- 负数 A - 负数 B，为“负+正”的情况，OF=0
  - 若  $A \geq B$ ，则 SF=0；若  $A < B$ ，则 SF=1
- 正数 A - 负数 B，肯定  $A > B$ ，为“正+正”的情况
  - 若结果为负 SF=1，则溢出 OF=1
  - 若结果为正 SF=0，则未溢出 OF=0
- 负数 A - 正数 B，肯定  $A < B$ ，为“负+负”的情况
  - 若结果为负 SF=1，则未溢出 OF=0
  - 若结果为正 SF=0，则溢出 OF=1

参考 CMP 指令

# 03 | 8086常用指令-单个标志位测试的转移指令

## 单个标志位测试的转移指令

指令	英文全称	检测的转移条件	功能描述
JZ	Jump if Zero	ZF=1	结果为0，则转移
JNZ	Jump if Not Zero	ZF=0	结果不为0，则转移
JS	Jump if Signed	SF=1	结果为负数，则转移
JNS	Jump if Not Signed	SF=0	结果为正数，则转移
JC	Jump if Carry	CF=1	产生进/借位，则转移
JNC	Jump if Not Carry	CF=0	未产生进/借位，则转移
JO	Jump if Overflow	OF=1	结果溢出，则转移
JNO	Jump if Not Overflow	OF=0	结果未溢出，则转移
JP/JPE	Jump if Parity / Parity Even	PF=1	结果有偶数个1，则转移
JNP/JPO	Jump if No Parity / Parity Odd	PF=0	结果有奇数个1，则转移
JCXZ	Jump if Register CX is Zero	CX=0	CX为零，则转移

# 03 | 8086常用指令-条件转移指令示例1

## 示例1：用汇编语言实现以下 C 语言的 if 语句

- C 语言中的 if 语句可以看作是**判断+执行或跳过**
  - 判断的条件实际上是比较的过程
- 汇编语言程序可以通过**比较指令**和**转移指令**来实现

```
if ( op1==op2 )  
{  
    X = 1;  
    Y = 2;  
}
```

## 参考汇编程序

```
1.      MOV AX, OP1  
2.      CMP AX, OP2      ; OP1 == OP2?  
3.      JNE L1           ; 否: 跳过后续指令  
4.      MOV X, 1         ; 是: X, Y 赋值  
5.      MOV Y, 2  
6.  L1:  . . . . .
```

# 03 | 8086常用指令-条件转移指令示例2

## 示例2：将 3 个数的最小值送入 AX 寄存器中

- 假定 V1、V2、V3 为已定义的字变量

```
1.      MOV AX, V1 ;假设 V1 是最小值
2.      CMP AX, V2
3.      JBE L1      ;如果 AX ≤ V2, 跳转到 L1
4.      MOV AX, V2 ;否则, 将 V2 送入 AX
5. L1:   CMP AX, V3
6.      JBE L2      ;如果 AX ≤ V3, 跳转到 L2
7.      MOV AX, V3 ;否则, 将 V3 送入 AX
8. L2:   .....
```

# 03 | 8086常用指令-循环控制指令

## 循环控制

- 在设计循环程序时，可以用控制指令来控制**循环**是否继续
- 8086 指令系统提供了 3 种形式的循环控制指令

## 循环控制指令

- LOOP
- LOOPZ/LOOPE
- LOOPNZ/LOOPNE

## 注意

- 循环指令本身**不影响标志位**

# 03 | 8086常用指令 - LOOP指令

## 循环 LOOP 指令格式

- `LOOP Label`

## 指令执行过程

- 先将 `CX` 的内容减 1，再判断 `CX` 是否为 0，如不为 0，则继续循环；如为 0，则退出循环，执行下一条指令

## 示例 —— 最简单的延时子程序

```
1.      MOV CX, n ;设置循环次数
2.  KKK: .....
3.      LOOP KKK  ;CX 减 1, 如不为 0, 则循环
4.      .....    ;退出循环, 后续处理
```

以下程序段执行结束后，AX 的值为（ ）

```
MOV CX, 10H
MOV AX, 0
KKK: ADD AX, 1
      LOOP KKK
```

- ☐ A 0
- ☐ B 0001H
- ☒ C 0010H
- ☐ D 0016H

# 03 | 8086常用指令 - LOOPZ/LOOPE指令

## 循环 LOOPZ/LOOPE 指令格式

- LOOPZ/LOOPE Label

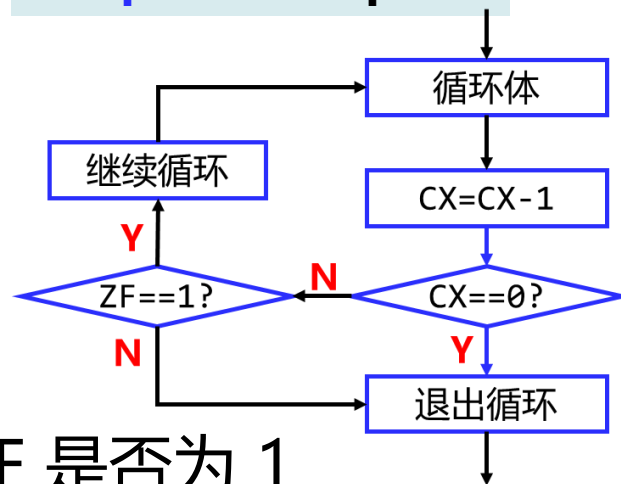
## 指令执行过程

- 先将 CX 的内容减 1
- 再判断 CX 是否为 0，并且判断 ZF 是否为 1
  - 当 ZF=1，并且 CX≠0，则继续循环
  - 当 ZF=0，或者 CX=0，则退出循环

## 注意

- CX 中的值为 0 时，并不会影响标志位 ZF。这就是说，ZF 是否为 1，是由前面其他指令的执行决定的

Loop While Zero  
Loop While Equal





# 03 | 8086常用指令 - LOOPNZ/LOOPNE指令

## 循环 LOOPNZ/LOOPNE 指令格式

- LOOPNZ/LOOPNE Label

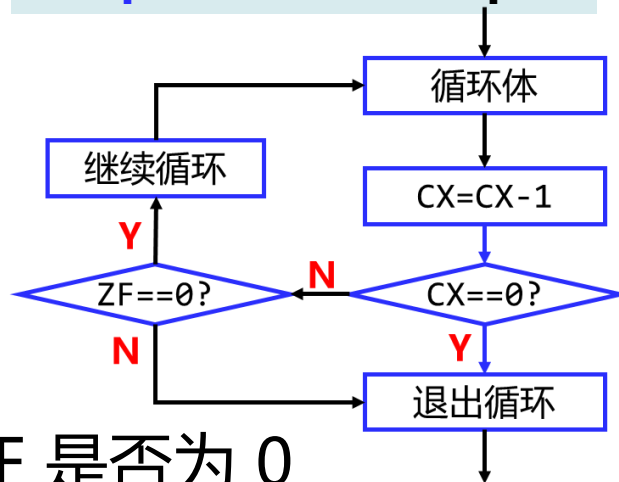
## 指令执行过程

- 先将 CX 的内容减 1
- 再判断 CX 是否为 0，并且判断 ZF 是否为 0
  - 当 ZF=0，并且 CX≠0，则继续循环
  - 当 ZF=1，或者 CX=0，则退出循环

## 注意

- 和 LOOPZ/LOOPE 的执行情况类似。一般情况下，从 LOOPNZ/LOOPNE 指令构成的循环退出以后，紧接着用 JNZ 或 JZ 指令来判断是在什么情况下退出循环的

Loop While Not Zero  
Loop While Not Equal












# 03 | 8086常用指令-指令学习常见错误

## 容易犯的错误

- 寻址方式错误
  - 编造并不存在的寻址方式
  - 错误使用间址寄存器
- 指令格式错误
  - 操作码错误，如 MOVE
  - 多个操作数之间不加逗号
  - 操作数的表示方式错误
- 操作数类型不匹配
  - 搞清楚哪些寻址方式是确定类型的，哪些是不确定的
- 隐含操作数问题
  - 忘记设置隐含操作数
  - 隐含操作数设置错误
  - 将隐含操作数显式写出
- 凭空构造指令
  - 对于指令的使用格式记忆错误
  - 区分 16 位系统的指令和 32 位系统的指令

# 03 | 8086常用指令-指令练习1


判断下列指令的正误，并说明原因

- |                     |  |
|---------------------|--|
| 1. MOV BL, CX       |  x 数据类型不匹配        |
| 2. MOV DS, SS       |  x 两操作数不能同时为段寄存器  |
| 3. MOV [BX], [DI]   |  x 两操作数不能同时为存储单元  |
| 4. MOV AL, [BX][SI] |  ✓                |
| 5. MOV ES, AL       |  x 数据类型不匹配        |
| 6. MOV DS, DX       |  ✓                |
| 7. MOV CS, AX       |  x CS 不能作为目的操作数 |
| 8. MOV BX, CS       |  ✓              |
| 9. MOV DS, 1230H    |  x 段寄存器不能用立即数赋值 |


# 03 | 8086常用指令-指令练习1

判断下列指令的正误，并说明原因

10. XCHG BX, 3

 x 操作数不能是立即数

11. POP CS

 x CS 不能作为目的操作数

12. MOV IP, SI

 x IP 指针不能作为指令的操作数

13. PUSH CS

 ✓

14. PUSH BL

 x PUSH 指令不能对 8 位数据入栈

15. MOV [SP], BX

 x SP 不能作为间址寄存器

16. MOV AX, BX+3

 x 源操作数的表示有误

17. MOV AX, [BX+3]

 ✓

18. MOV BX, [BX]

 ✓

# 03 | 8086常用指令-指令练习1

判断下列指令的正误，并说明原因

19. MOV BH, [BL]

 ✗ BL 不能作为间址寄存器

20. XCHG ES, AX

 ✗ 操作数不能使用段寄存器

21. LEA AX, [BX+SI]

 ✓

22. MUL 10H

 ✗ MUL 指令单操作数不能为立即数

23. IMUL DX

 ✓

24. DIV 10

 ✗ DIV 指令不能使用立即数

25. IDIV DX, 10H

 ✗ 没有此格式的除法指令

26. SHL AX, CX

 ✗ 循环次数只能为 1 或 CL

27. SHR BX, CH

 ✗ 移位次数只能是 1 或 CL

# 03 | 8086常用指令-指令练习1

判断下列指令的正误，并说明原因

28. ROL BX, 20

 x 循环指令次数只能为 1 或 CL

29. RCR AX, CL

 ✓

30. CMP AX, 1234H

 ✓

31. CMP 12H, CL

 x 立即数不能作为目的操作数

32. JCXZ next

 ✓

33. JEBXZ next

 x 无此指令助记符

# 03 | 8086常用指令-指令练习2

## 将 DX:AX 中的双字转换成其相反数

### ● 思路:

- ① **双字数据的减法**: 用 0 减去 DX:AX 的值
- ② **位操作**: 对 DX:AX 表示的 32 位数据所有位取反, 末位加 1
  - ◆ 当 (AX)=0 时, 需要将 DX 各位取反末位加 1, AX 保持不变

### ● 相关指令:

- NOT 指令: 各位取反
- NEG 指令: 各位取反, 末位加 1
- SUB 指令: 0-OPRD

### ● 注意: 指令序列中应考虑特例情况

- NEG AX 时, (AX)=0, 则 CF=0; 但实际应使 DX 取反加 1
- NEG AX 时, (AX)≠0, 则 CF=1; 但实际 DX 只取反即可

# 03 | 8086常用指令-指令练习2

将 DX:AX 中的双字转换成其相反数

- 解法一 (0-DX:AX):

```
MOV CX, DX
MOV BX, AX
XOR DX, DX
XOR AX, AX
SUB AX, BX
SBB DX, CX
```

- 解法二 (各位取反, 末位加一):

```
NOT AX
NOT DX
ADD AX, 1
ADC DX, 0
```

- 解法三 (低位取补, 高位取反加进位):

```
NOT DX
NEG AX
CMC
ADC DX, 0
```

- 解法四 (各位取补, 高位减借位):

```
NEG DX
NEG AX
SBB DX, 0
```



# 03 | 8086常用指令-指令练习3

## 将 DX:AX 中的双字数据逻辑左移 4 位

### ● 思路1:

#### 移位练习

- 逻辑左移 1 位, 可以通过 SHL 和 RCL 配合完成
- 左移 4 位, 可通过循环 4 次完成

```
MOV CX, 4  
SHF1: SHL AX, 1  
      RCL DX, 1  
      LOOP SHF1
```

### ● 思路2:

- 分别将 DX、AX 逻辑左移
- 使用 AX 元数据的高 4 位, 设置 DX 的低 4 位

```
MOV CL, 4  
SHL DX, CL  
MOV BL, AH  
SHL AX, CL  
SHR BL, CL  
OR DL, BL
```

# 03 | 8086常用指令-指令练习4

## 将 CL 的第 5 位置为 AL 的第 0 位

### ● 思路:

- 将 CL 第 5 位清零
- 提取 AL 第 0 位, 并左移 5 位
- 用 OR 指令设置两个对应位

### ● 指令序列:

```
AND CL, 1101 1111B    ;C5 位置零
MOV CH, CL             ;将 CL 复制到 CH, CL 要循环计数
AND AL, 0000 0001B    ;提取 A0 位, 其他位置零
MOV CL, 5
SHL AL, CL             ;将 AL 左移 5 位
OR CH, AL              ;将 C5 位置为 A0 位
MOV CL, CH             ;将数据移回 CL
```

# 本节小结

---

- 掌握标识符的写法，能够识别并写出正确的标识符
- 理解寻址方式的含义，掌握 16 位系统使用的 7 种寻址方式
  - 特别是存储器寻址的 5 种方式，EA 和 PA 形成
- 熟练掌握 16 位系统的常用汇编指令，并能应用
  - 指令格式：操作码、操作数的个数和形式
  - 指令功能，包括指令对各状态标志位的影响
  - 指令应用特点，指令序列之间的关系

# 本节小结-传送类指令

指令格式	功能
MOV    DST, SRC	将源操作数的值传送至目的操作数
PUSH   SRC	将源操作数入栈
POP    DST	将堆栈栈顶元素出栈至目的操作数
XCHG   OPRD1, OPRD2	将两操作数的值相互交换
LEA    DST, SRC	将源操作数的有效地址送目的寄存器

# 本节小结-算术运算类指令1

指令格式	功能
ADD    DST, SRC	$(DST) + (SRC) \rightarrow DST$
ADC    DST, SRC	$(DST) + (SRC) + CF \rightarrow DST$
INC    DST	$(DST) + 1 \rightarrow DST$
SUB    DST, SRC	$(DST) - (SRC) \rightarrow DST$
SBB    DST, SRC	$(DST) - (SRC) - CF \rightarrow DST$
CMP    OPRD1, OPRD2	$(OPRD1) - (OPRD2)$
DEC    DST	$(DST) - 1 \rightarrow DST$
NEG    DST	$0 - (DST) \rightarrow DST$

# 本节小结-算术运算类指令2

指令格式	功能
MUL/IMUL SRC	SRC 为 8 位数据时, $(SRC) \times (AL) \rightarrow AX$ SRC 为 16 位数据时, $(SRC) \times (AX) \rightarrow DX:AX$
DIV/IDIV SRC	SRC 为 8 位数据时, $(AX) \div (SRC) \rightarrow AL...AH$ SRC 为 16 位数据时, $(DX:AX) \div (SRC) \rightarrow AX...DX$
CBW	AL 中的 8 位数据符号扩展为 16 位, 保存于 AX 中
CWD	AX 中的 16 位数据符号扩展为 32 位, 保存于 DX:AX 中

# 本节小结-处理器控制指令

指令格式	功能
STC	CF 置 1
CLC	CF 清 0
CMC	CF 取反
STD	DF 置 1
CLD	DF 清 0
STI	IF 置 1
CLI	IF 清 0

# 本节小结-串操作指令

指令	重复前缀	功能
MOVS	REP	将由 DS:SI 所指向的存储单元的字节串或字串 <b>传送到</b> 由 ES:DI 所指向的存储单元
CMPS	REPE/REPZ	把 DS:SI 所指向的存储单元的字节或字与 ES:DI 所指向的存储单元的字节或字相 <b>比较</b> ，并设置相应标志位
SCAS	REPE/REPZ	将 AL/AX 的字节或字内容与 ES:DI 所指向的存储单元的字节或字相比较，并设置相应标志位
LODS	无	将 DS:SI 所指的存储单元的字节或字内容 <b>装入</b> (提取) 到 AL/AX 中
STOS	REP	把 AL/AX 的内容 <b>存放</b> 到 ES:DI 所指的存储单元中



# 本节小结-程序控制指令

指令格式	功能
JMP OPRD	无条件转移到 OPRD 所制定的位置执行
JE/JZ/JNE/JNZ JA/JNBE/JAE/JNB JB/JNAE/JBE/JNA	根据无符号数比较结果进行转移
JE/JZ/JNE/JNZ JG/JNLE/JGE/JNL JL/JNGE/JLE/JNG	根据有符号数比较结果进行转移
JZ/JNZ JS/JNS JC/JNC JO/JNO	根据单个标志位结果进行转移
LOOP Label	CX≠0 则继续循环
LOOPZ/LOOPE Label	CX≠0 且 ZF=1 则继续循环
LOOPNZ/LOOPNE Label	CX≠0 且 ZF=0 则继续循环

条件转移指令 JNE 的测试为真的条件为 ( )

- ☒ A ZF=0
- ☐ B ZF=1
- ☐ C CF=0
- ☐ D CF=1

# 目录

---

- 01 80x86 指令系统概述
- 02 8086 的寻址方式
- 03 8086 的常用指令
- 04 80386 的寻址方式和指令系统**

# 04 | 80386寻址方式-寻址方式

## 80386 寻址方式

- 80386 CPU 继续采用分段的方法管理主存储器
- 在实模式下，80386 与 8086 寻址完全相同
  - 存储单元的物理地址仍然是**段寄存器内的段值乘上 16 加上段内偏移**
- 在保护模式下，段寄存器内所含的是指示段基址的**选择子**
  - 存储单元的地址是**段基址加上段内偏移**，但不再是段寄存器之值乘 16 加上偏移
  - 80386 CPU 新增了两个数据段寄存器 FS 和 GS
- 80386 CPU 中**立即寻址方式**和**寄存器寻址方式**与 8086 相同，且操作数可扩展为 32 位宽
- **存储器寻址方式**不仅位数扩展为 32 位，寻址范围和方式更加灵活

# 04 | 80386寻址方式-存储器寻址方式

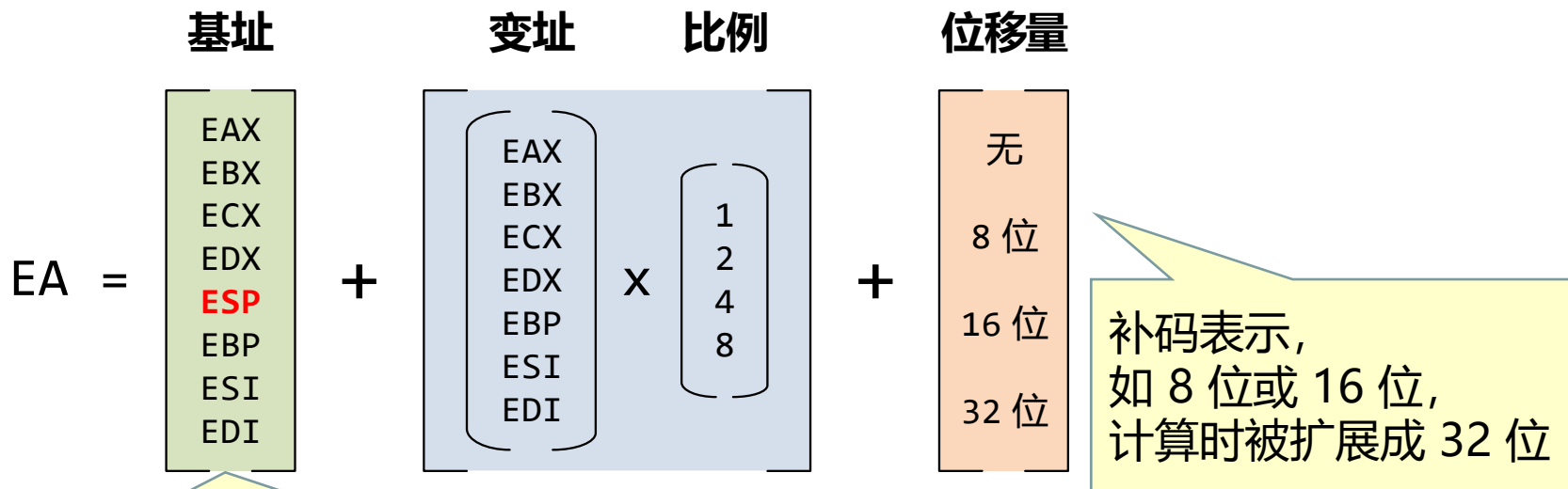
## 存储器寻址方式

- 80386 允许偏移地址可以由**三部分内容相加**构成
  - 一个 32 位**基址寄存器**、一个可乘上比例因子 1、2、4 或 8 的 32 位**变址寄存器**、一个 8 位到 32 的常数**偏移量**
- 存储器寻址方式中的有关名词概念
  - **基址**：任何通用寄存器都可作为基址寄存器，其内容为基址
  - **位移量**：在指令操作码后面的 32 位、16 位或 8 位的数
  - **变址**：除了 ESP 寄存器外，任何通用寄存器都可以作为变址寄存器，其内容即为变址值
  - **比例因子**：变址寄存器的值可以乘以一个比例因子，根据操作数的长度可为 1 字节、2 字节、4 字节或 8 字节，比例因子相应地可为 1、2、4 或 8
- 由上面 4 个分量计算有效地址的方法为

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

# 04 | 80386寻址方式-存储器寻址方式

## 存储器寻址方式



8 个 32 位通用寄存器都可以作为**基址寄存器**

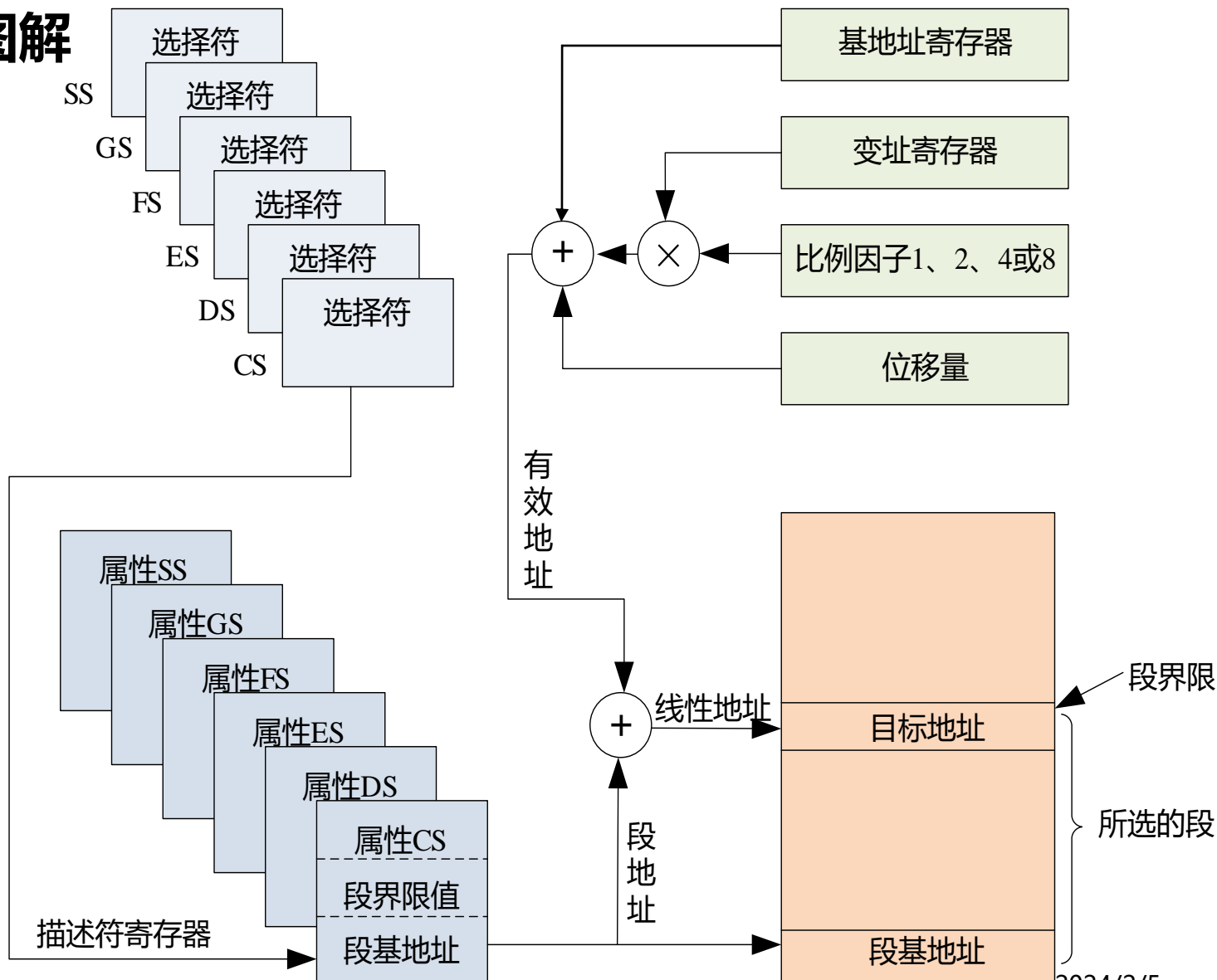
除 ESP 寄存器外, 其他 7 个通用寄存器都可以作为**变址寄存器**

- 基址、带比例因子的变址、位移量可省去任意的两部分

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

# 04 | 80386寻址方式-有效地址计算

## 寻址计算图解



# 04 | 80386寻址方式-具体的寻址方式

## 80386 寻址方式

- 立即寻址方式
  - 寄存器寻址方式
  - 直接寻址方式
  - 寄存器间接寻址方式
  - 寄存器相对寻址方式
  - 基址变址寻址方式
  - 相对基址变址寻址方式
  - 基址比例变址寻址方式
  - 相对基址比例变址寻址方式
- 与 8086 相同，操作数可以是 32 位
- 存储器寻址方式



# 04 | 80386寻址方式-直接寻址方式

## 直接寻址方式

- 操作数在**存储单元**中，指令的操作码之后给出该存储单元的偏移地址
- 指令中，EA 可以是**数值形式**，也可以是**符号地址形式**

## 示例

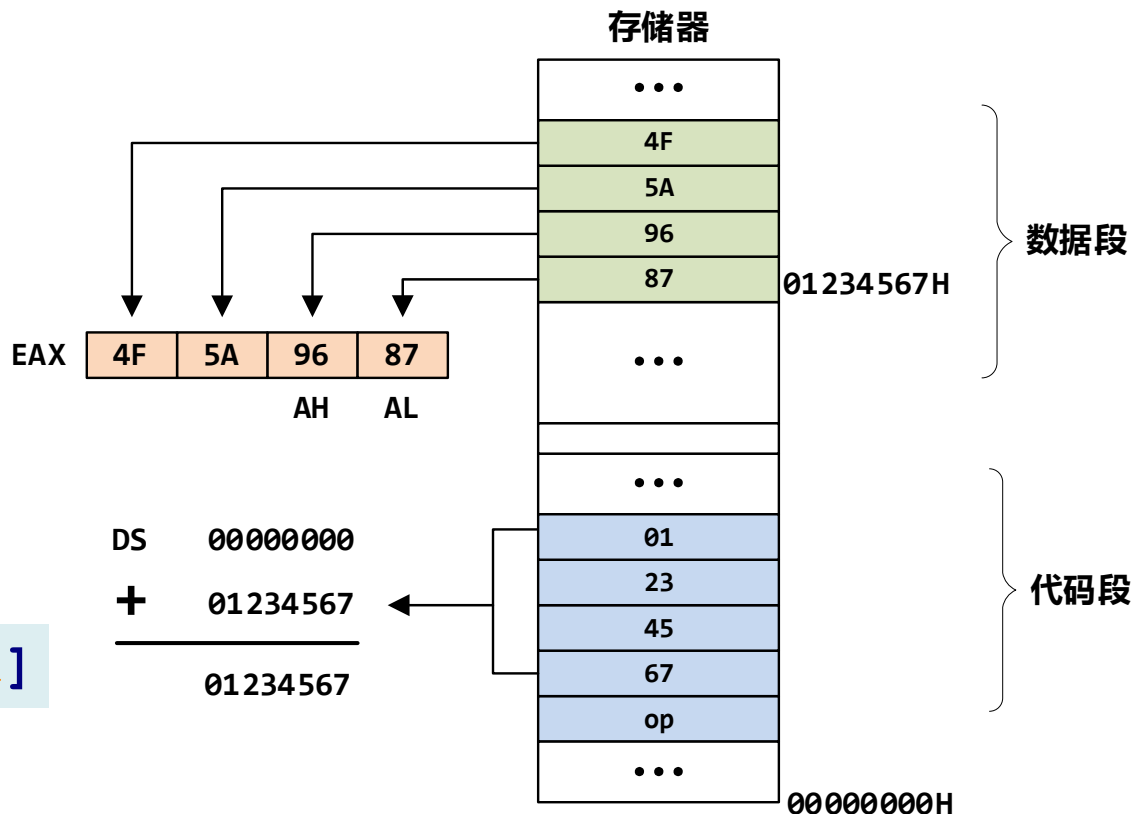
1. MOV ECX, [95480H]	;源操作数采用直接寻址
2. MOV [9547CH], DX	;目的操作数采用直接寻址
3. ADD BL, [95478H]	;源操作数采用直接寻址

# 04 | 80386寻址方式-直接寻址方式举例

## 示例

- 假设数据段和代码段重叠，段起始地址都是 0
- 有效地址为 01234567H 的双字存储单元中内容是 4F5A9687H

```
MOV EAX, [01234567H]
```



# 04 | 80386寻址方式-寄存器间接寻址方式

## 寄存器间接寻址方式

- 操作数在存储器中，由八个 32 位的通用寄存器之一给出操作数所在存储单元的有效地址。把这种通过寄存器间接给出存储单元有效地址的方式称为寄存器间接寻址方式

## 示例

```
1. MOV EAX, [ESI] ;源操作数寄存器间接寻址, ESI 给出有效地址
2. MOV [EAX], CL  ;目的操作数寄存器间接寻址, EAX 给出有效地址
3. SUB DX, [EBX]  ;源操作数寄存器间接寻址, EBX 给出有效地址
```

# 04 | 80386寻址方式-寄存器间接寻址方式

## 寄存器间接寻址方式与寄存器寻址方式的区别

- 寄存器间接寻址与寄存器寻址有**本质区别**
- **寄存器间接寻址的寄存器出现在方括号中**
- 寄存器间接寻址方式中，给出操作数所在存储单元有效地址的寄存器，相当于 C 语言中的指针变量，它含有要访问存储单元的地址

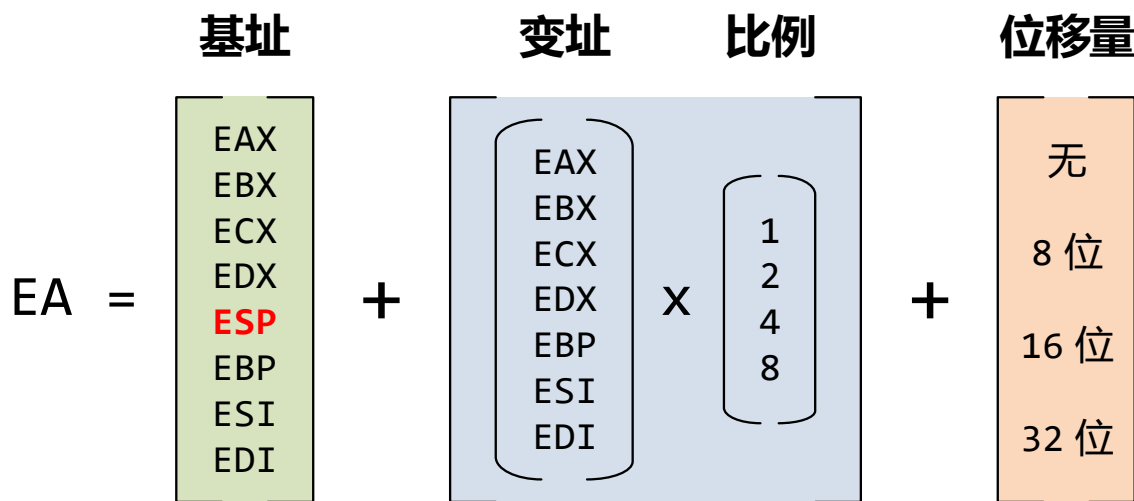
## 示例

1. `MOV EAX, [ESI]` ;源操作数寄存器间接寻址, ESI 给出有效地址
2. `MOV EAX, ESI` ;源操作数采用寄存器寻址, ESI 给出操作数内容

# 04 | 80386寻址方式-存储器寻址方式通用表示

## 存储器寻址方式的通用表示

- 默认的段首址由基址寄存器决定，与变址寄存器无关
- 当基址寄存器为 EBP 或 ESP 时，默认的段首址由 SS 指定
- 当基址寄存器为其他时，默认的段首址由 DS 指定



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

# 04 | 80386寻址方式-存储器寻址方式通用表示

## 寄存器相对寻址方式

1. `MOV EAX, [EBX+12H]` ;源操作数有效地址是 EBX 值加上 12H
2. `MOV [ESI-4], AL` ;目的操作数有效地址是 ESI 值减去 4
3. `ADD DX, [ECX+5328H]` ;源操作数有效地址是 ECX 值加上 5328H

## 基址变址寻址方式

1. `MOV EAX, [EBX+ESI]` ;源操作数有效地址是 EBX 值加上 ESI 值
2. `SUB [ECX+EDI], AL` ;目的操作数有效地址是 ECX 值加上 EDI 值
3. `XCHG [ESP+ESI], DX` ;目的操作数有效地址是 ESP 值加上 ESI 值

## 基址比例变址寻址方式、相对基址变址寻址、相对基址比例变址寻址

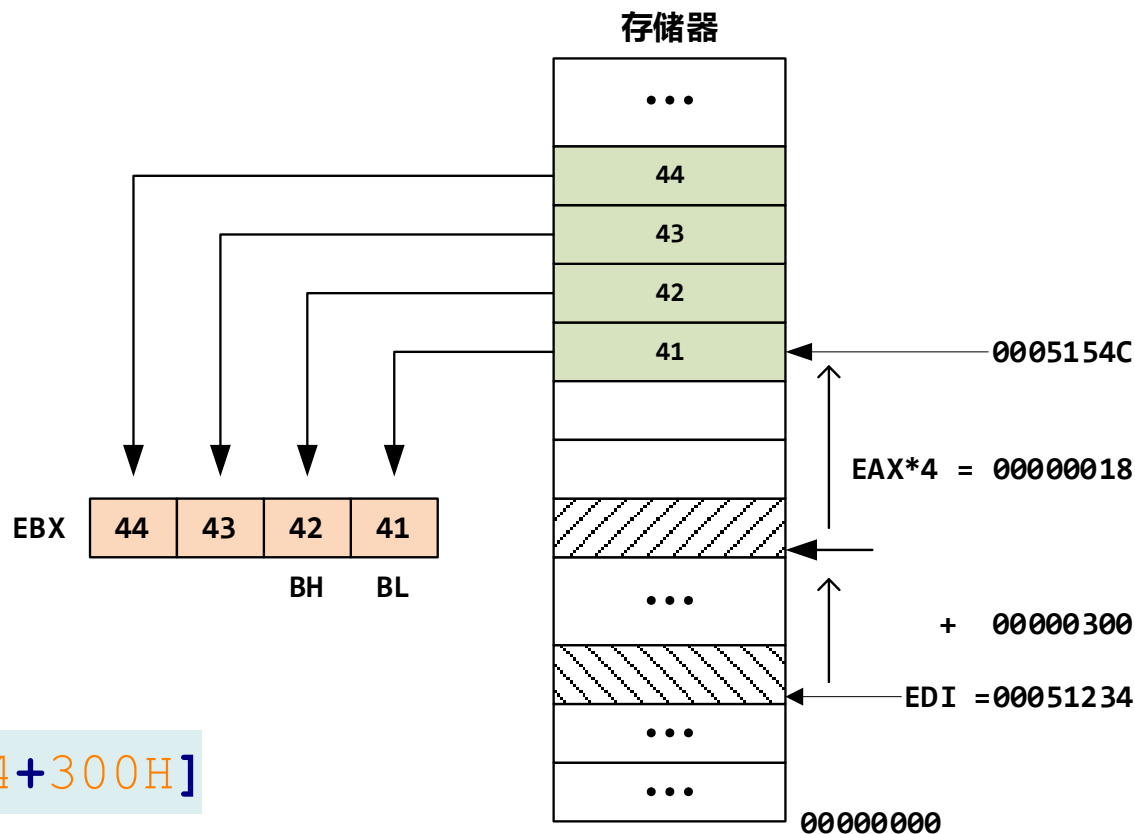
1. `MOV EAX, [ECX+EBX*4]` ;EBX 作为变址寄存器, 放大因子是 4
2. `MOV [EAX+ECX*2], DL` ;ECX 作为变址寄存器, 放大因子是 2
3. `ADD EAX, [EBX+ESI*8]` ;ESI 作为变址寄存器, 放大因子是 8
4. `SUB ECX, [EDX+EAX-4]` ;EAX 作为变址寄存器, 放大因子是 1
5. `MOV EBX, [EDI+EAX*4+300H]` ;EAX 作为变址寄存器, 放大因子是 4

# 04 | 80386寻址方式-存储器寻址方式通用表示

## 示例

- 假设由 **DS** 的段起始地址是 **0**，寄存器 **EDI** 的内容是 **51234H**，寄存器 **EAX** 的内容是 **6**，并且有效地址为 **0005154CH** 的双字存储单元的内容是 **44434241H**

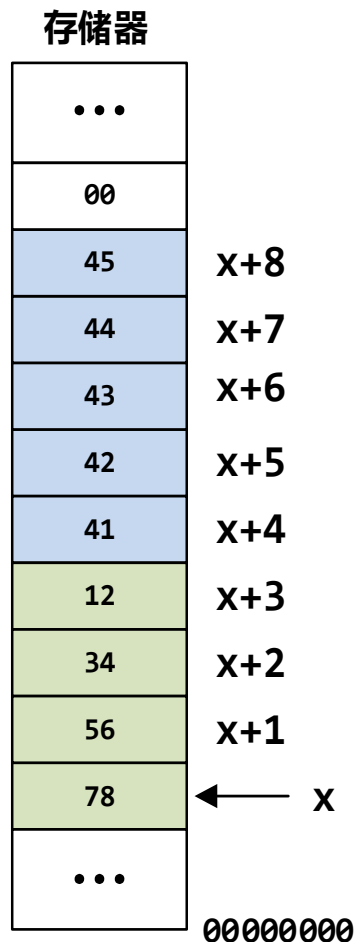
```
MOV EBX, [EDI+EAX*4+300H]
```



# 04 | 80386寻址方式-存储器寻址方式通用表示

## 示例

```
1. LEA EBX, x
2. MOV EAX, [EBX]           ; 寄存器间接寻址
3. MOV dv1, EAX             ; dv1=12345678H
4. MOV EAX, [EBX+1]         ; 寄存器相对寻址
5. MOV dv2, EAX             ; dv2=41123456H
6. ;
7. MOV ECX, 2
8. MOV AX, [EBX+ECX]        ; 基址变址寻址
9. MOV dv3, EAX             ; dv3=41121234H
10. ;
11. MOV AL, [EBX+ECX*2+3]   ; 相对基址比例变址寻址
12. MOV dv4, EAX           ; dv4=41121244H
```





# 04 | 80386寻址方式-存储器寻址方式的说明

## 存储器寻址方式的说明

- 如果指令的操作数允许是存储器操作数，那么**各种存储器寻址方式都适用**
  - 存储器操作数的尺寸可以是**字节**、**字**或者**双字**
- 在某条具体的指令中，如果有**寄存器操作数**，那么其尺寸是确定的
  - 通常要求一条指令中的多个操作数的尺寸一致
  - 指令中的**寄存器操作数的尺寸**就决定了存储器操作数的尺寸
  - 少数情况下，需要**显式地指定**存储器操作数的尺寸

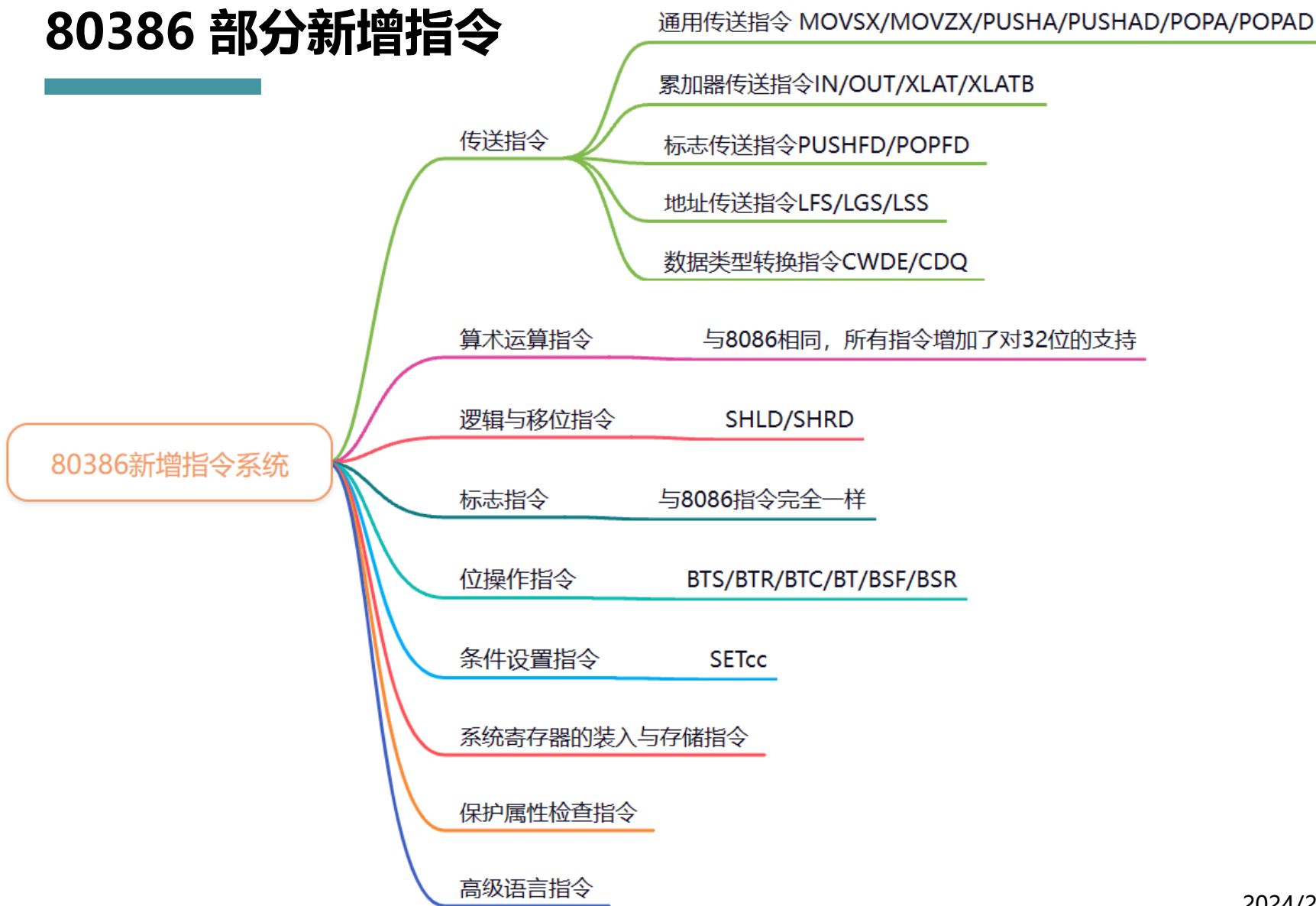
## 04 | 80386寻址方式-存储器寻址方式的说明

### 存储器寻址方式的说明

- 如果**基址寄存器**不是 **EBP** 或者 **ESP**，那么缺省引用的段寄存器是 **DS**
- 如果**基址寄存器**是 **EBP** 或者 **ESP**，那么缺省引用的段寄存器是 **SS**
- 当 **EBP** 作为**变址寄存器**使用 (**ESP** 不能作为变址寄存器使用) 时，缺省引用的段寄存器仍然是 **DS**
- 无论存储器寻址方式简单或者复杂，如果由基址寄存器、带比例因子的变址寄存器和位移量这三部分相加所得超过 32 位，那么**有效地址仅为低 32 位**

# 04 | 80386指令系统-新增常用指令

## 80386 部分新增指令



# 03 | 80386指令系统-传送类指令

## 通用传送指令

Move with Sign Extend  
Move with Zero Extend

● **MOVSX/MOVZX** <DST>, <SRC>

■ **DST**: REG

■ **SRC**: REG/MEM

## 功能

- 将 8 位有符号数 **SRC** 符号扩展 (或零扩展 **MOVZX**) 成 16 位或 32 位、或者将 16 位有符号数 **SRC** 符号扩展 (或零扩展 **MOVZX**) 成 32 位数再传送至 **DST**
  - 缓解 **MOV** 指令操作数不匹配的问题

## 注意

- **DST** 为 16 位或 32 位 REG, **SRC** 为 8 位或 16 位 REG/MEM

# 03 | 80386指令系统-传送类指令

## 其他通用传送指令

- **XCHG** 指令除了可进行字节交换、字交换外，还可以实现**双字交换**
- **PUSH** 指令的操作数除了可以是寄存器或存储器外，还可以是**立即数** (**POP** 指令无此功能)
- **PUSHAD** 一条指令则可将**全部的 32 位寄存器**压入堆栈，压入堆栈的次序为：EAX、ECX、EDX、EBX、ESP、EBP、ESI、EDI
- **POPAD** 则进行相反的弹出操作

# 03 | 80386指令系统-传送类指令举例

## 其他通用传送指令

```
1. MOV BL, 92H
2. MOVSX AX, BL           ;AX←FF92H
3. MOVSX ESI, BL          ;ESI←FFFFFF92H
4. MOVZX EDI, AX          ;EDI←0000FF92H
5.
6. XCHG EAX, EDI          ;寄存器和寄存器进行双字交换
7. XCHG ESI, MEM_DWORD   ;寄存器和存储器进行双字交换
8. PUSH 0807H            ;将立即数 0807H 压入堆栈
```

# 03 | 80386指令系统-传送类指令

## 累加器传送指令

- **IN**、**OUT**、**XLAT** 与 8086 相同，**XLATB** 是以 32 位的 **EBX** 为基址

## 标志传送指令

- 新增 **PUSHFD**、**POPFD** 将标志寄存器内容作为一个双字压入堆栈，或从堆栈顶弹出双字到标志寄存器
  - **LAHF**、**SAHF**、**PUSHF**、**POPF** 与 8086 相同

## 数据类型转换指令

Convert **Word** to **Extended Doubleword**  
Convert **Double** to **Quad**

- 新增 **CWDE**、**CDQ** 将 **AX** 中的字进行双字扩展，或将 **EAX** 中的双字进行四字扩展，结果存放在 **EDX:EAX**
  - **CBW**、**CWD** 与 8086 相同

# 03 | 80386指令系统-传送类指令

## 地址传送指令

- LEA 允许 32 位操作数
- LDS/LES 指令允许从数据段中取出 32 位的偏移量送给 DST，取 16 位的段基址送 DS/ES
- 新增 LFS/LGS/LSS 指令

LFS/LGS/LSS REG16/REG32, MEM

- 从当前数据段中取 32/48 位的地址指针送段寄存器和通用寄存器，其中高 16 位送段寄存器，低位送通用寄存器

## 示例

1. TABLE DD TABLE1 ;定义变量 TABLE1 的类型为双字(4字节)
2. DATA DF DATA1 ;定义变量 DATA 的类型为长字(6字节)
3. LFS SI, TABLE ;2 字节偏移地址送 SI, 2字节送段地址 FS
4. LGS ESI, DATA ;4 字节偏移地址送 ESI, 2字节送段地址GS



# 03 | 80386指令系统-算术运算指令

## 算术运算指令

- 80386 指令系统的算术运算指令与 8086 并没有很大的区别，最大的改变只是对 32 位数据的支持
  - 除法运算指令 **DIV/IDIV** 与 8086 一致
  - 乘法运算指令 **MUL/IMUL** 在 16 位运算时乘积的高半部分**存放在 EAX 的高16 位，且同时存放在 DX 中**
  - 支持 3 操作数指令格式

**IMUL** <DST>, <SRC1>, <SRC2> ;  $SRC1 \times SRC2 \rightarrow DST$

**DST**: REG     **SRC1**: REG/MEM;     **SRC2**: IMM

## 示例

1. **IMUL DX, BX, 500H** ;将BX中内容乘以500H, 结果送DX
2. **IMUL ECX, EDX, 1000H** ;将EDX内容乘以1000H, 结果送ECX
3. **IMUL EDX, DWORD, 30H** ;将存储器中双字乘以30H, 结果送EDX

# 03 | 80386指令系统-逻辑指令

## 双精度移位指令

Double Precision Shift Left  
Double Precision Shift Right

- SHLD/SHRD <DST>, <SRC>, <COUNT>  
DST: REG/MEM      SRC: REG      COUNT: CL/IMM8

## 功能

- 将指令中的两个操作数连起来进行移位。在移位操作中，将 SRC 内容移入 DST，而 SRC 本身不变。进位位 CF 中的值为 DST 移出的最后一位

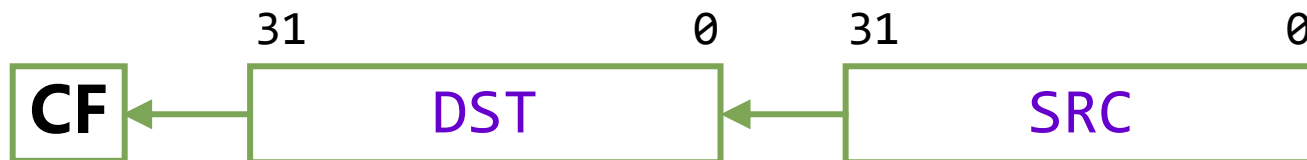
## 注意

- DST 为 REG/MEM, SRC 为 REG, COUNT 为 CL 或 8 位 IMM, COUNT 最大为 31

# 03 | 80386指令系统-逻辑指令

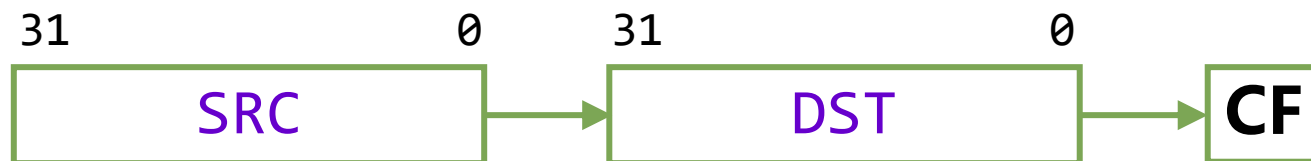
## 双精度移位指令

SHLD <DST>, <SRC>, <COUNT>



DST 最高位移入 CF，最低位由 SRC 高位补入

SHRD <DST>, <SRC>, <COUNT>



DST 最低位移入 CF，最高位由 SRC 低位补入

# 03 | 80386指令系统-位串操作指令

## 新增串操作指令

Input String from Port  
Out String from Port

- **INS** <SRC>

- SRC: DX

- DST: [ES:(E)DI]

- **OUTS** <DST>

- SRC: [DS:(E)SI]

- DST: DX

## 功能

- **INS** 从一个由 DX 给定的输入端口读入数据送入由 ES:(E)DI 指定的一连续的存储单元
- **OUTS** 将由 DS:(E)SI 指定的一连续的存储单元的数据输出到由 DX 给定的输出端口

## 注意

- **INS** 和 **OUTS** 要求用 DX 存放端口号, 不能用立即数
- **MOVS/CMPS/SCAS/LODS/STOS** 新增对 32 位的支持

# 03 | 80386指令系统-位测试指令

## 位测试指令

Bit Test  
Bit Test and Set  
Bit Test with Reset/Compliment

- BT/BTS/BTR/BTC <DST>, <SRC>
  - DST: REG/MEM
  - SRC: REG/IMM
- 将 DST 特定的位 (由 SRC 指定) 送入 CF 保存, 并测试该位

## 位扫描指令

Bit Scan Forward  
Bit Scan Reverse

- BSF/BSR <DST>, <SRC>
  - DST: REG
  - SRC: REG/MEM
- 扫描 SRC, 将 SRC 中第一个 1 的位置放入 DST, 并设置 ZF

# 03 | 80386指令系统-位测试指令

## 位测试指令

指令	英文全称	指令功能
BTS	Bit Test and Set	将测试位的值送 CF，并把指定的测试位置 1
BTR	Bit Test with Reset	将测试位的值送 CF，并把指定的测试位置 0
BTC	Bit Test with Compliment	将测试位的值送 CF，并把指定的测试位取反
BT	Bit Test	测试指定的位，并将测试位的值送 CF
BSF	Bit Scan Forward	从低到高扫描 SRC，如果全为 0 则 ZF 置 1，否则 ZF 置 0，并把为 1 位的序号放入 DST
BSR	Bit Scan Reverse	从高到低扫描 SRC，如果全为 0 则 ZF 置 1，否则 ZF 置 0，并把为 1 位的序号放入 DST

# 03 | 80386指令系统-位测试指令示例

## 位测试指令示例

```
1. MOV DX, 3456H ; DX=3456H
2. BTC DX, 5      ; CF=0, DX=3476H, 0101→0111
3. MOV CX, 18
4. BTR DX, CX     ; CF=1, DX=3472H, 0110→0010
5. MOV ECX, 3
6. BTS EDX, ECX   ; CF=0, DX=347AH, 0010→1010
7. BT EDX, 38     ; CF=1, DX=347AH
8.
9. MOV EBX, 12345678H
10. BSR EAX, EBX  ; ZF=0, EAX=1CH=28D
11. BSF DX, AX    ; ZF=0, DX=2
12. BSF CX, DX    ; ZF=0, CX=1
```

0011 0100 0101 0110B = 3456H

↑ ↑ ↑ ↑  
38 5 3 18

0001 1100B = 1CH

0010B = 2H

# 03 | 80386指令系统-LOCK前缀指令

## LOCK 前缀指令

- 带 LOCK 前缀指令执行期间，CPU 会对总线加锁，使其其他协处理器不能使用总线
- 由于 LOCK 前缀毫无间隙地长期封锁总线而妨碍操作系统将所需要的页面调入内存，实际上，32 位微处理器对可以接受 LOCK 前缀的指令作了限制，如下表

ADD MEM, REG	BT MEM, REG	OR MEM, IMM	BTC MEM, IMM
ADC MEM, REG	BTR MEM, REG	SBB MEM, IMM	DEC MEM
AND MEM, REG	BTS MEM, REG	SUB MEM, IMM	INC MEM
OR MEM, REG	BTC MEM, REG	XOR MEM, IMM	NEG MEM
SBB MEM, REG	ADD MEM, IMM	BT MEM, IMM	NOT MEM
SUB MEM, REG	ADC MEM, IMM	BTR MEM, IMM	XCHG REG, MEM
XOR MEM, REG	AND MEM, IMM	BTS MEM, IMM	XCHG MEM, REG



# 03 | 80386指令系统-条件设置指令

## 条件设置指令

- 80386 指令系统新增了条件设置指令，用于支持编译程序和高级语言生成的代码。这些指令根据当前的 EFLAGS 中的标志，来设置对应的寄存器或存储器

## 指令格式

- SET<sub>cc</sub> <DST>
- DST 只能为 REG 或 MEM
- 其中 <sub>cc</sub> 泛指所有的设置条件。若条件满足，则设置 DST 为指定的值

03

80386指令系统-条件设置指令功能1

条件设置指令功能

指令	英文全称	指令功能
SETZ/SETE	Set if Equal Set if Zero	ZF=1则置 DST 为1
SETS	Set if Signed	SF=1则置 DST 为1
SETO	Set if Overflow	OF=1则置 DST 为1
SETP/SETPE	Set if Parity Set if Parity Even	PF=1则置 DST 为1
SETB/SETNAE/SETC	Set if Below Set if Not Above or Equal Set if Carry	CF=1则置 DST 为1
SETBE/SETNA	Set if Below or Equal Set if Not Above	CF=1或ZF=1则置 DST 为1
SETL/SETNGE	Set if Less Set if Not Greater or Equal	SF≠OF则置 DST 为1
SETLE/SETNGE	Set if Less or Equal Set if Not Greater or Equal	SF≠OF或ZF=1则置 DST 为1

03

80386指令系统-条件设置指令功能2

条件设置指令功能

指令	英文全称	指令功能
SETNZ/SETNE	Set if Not Equal Set if Not Zero	ZF=0则置 DST 为1
SETNS	Set if Not Signed	SF=0则置 DST 为1
SETNO	Set if Not Overflow	OF=1则置 DST 为1
SETNP/SETPO	Set if Not Parity Set if Parity Odd	PF=0则置 DST 为1
SETNB/SETAE/SETNC	Set if Not Below Set if Above or Equal Set if Not Carry	CF=0则置 DST 为1
SETNBE/SETA	Set if Not Below or Equal Set if Above	CF=0且ZF=0则置 DST 为1
SETNL/SETGE	Set if Not Less Set if Greater or Equal	SF=OF则置 DST 为1
SETNLE/SETG	Set if Not Less or Equal Set if Greater	SF=OF且ZF=0则置 DST 为1

03

80386指令系统

- 系统寄存器的装入与存储指令

系统寄存器的装入与存储指令

- 系统控制、测试寄存器可与通用寄存器之间传送
- 以下指令可以将**系统寄存器**的内容**保存**在指定**内存单元**，或从指定内存单元内容**装入**系统寄存器中

指令	英文全称	指令功能
LMSW/SMSW	Load/Store Machine Status Word	装入/保存机器状态字
LIDT/SIDT	Load/Store Interrupt Descriptor Table	装入/保存中断描述符表寄存器
LGDT/SGDT	Load/Store Global Descriptor Table	装入/保存全局描述符表寄存器
LLDT/SLDT	Load/Store Local Descriptor Table	装入/保存局部描述符表寄存器
LTR/STR	Load/Store Task Register	装入/保存任务状态段寄存器

03

80386指令系统-保护属性检查指令

保护属性检查指令

- 对选择符相应描述符中的**访问权限**、**段限制**、**可读性**、**可写性**这些访问属性进行检查，对段的特权级进行调整
  - 此类指令只能在保护模式下使用

指令	英文全称	指令功能
LAR REG, SEL	Load Access Rights	取出 SEL 中的访问权限部分送入 REG
LSL REG, SEL	Load Segment Limit	取出 SEL 中的段限制部分送入 REG
VERR SEL	Verify Read	验证 SEL 对应的段是可读的
VERW SEL	Verify Write	验证 SEL 对应的段是可写的
ARPL SEL1, SEL2	Adjusted Requested Privilege Level of Selector	将 SEL1 的特权等级降低至 SEL2 的特权等级相同的水平

# 03 | 80386指令系统-其他指令

## 程序控制指令

- 条件转移指令的相对转移地址不受范围限制
- 循环指令 `LOOP`、`LOOPZ/LOOPE`、`LOOPNZ/LOOPNE` 的用法与 8086 完全一致

## 处理器指令

- 标志位操作指令 `CLC/STC/CMC/CLD/STD/CLI/STI` 和 8086 完全一样

## 高级语言指令

- 80386 指令系统新增了 3 个高级语言支持指令：`BOUND`、`ENTER` 和 `LEAVE`

# 本章小结

- 掌握标识符的写法，能够**识别并写出正确的标识符**
- 理解寻址方式的含义，掌握 16 位系统使用的 7 种寻址方式
  - 特别是存储器寻址的 5 种方式，**EA 和 PA 形成**
- 熟练掌握 **16 位系统的常用汇编指令**，并能应用
- 掌握 80386 寻址方式及其与 16 位寻址方式的**异同**
- 了解 80386 常用指令及其与 16 为常用指令的异同
- 了解 80386 保护属性检查指令、高级语言指令



河南大學  
Henan University

```
ConvertNum: ;Converts the number from hex to individual digits as ascii values
;Start - Get digit
mov ah,00h
mov bh,10
div bh ;al/bh remainder stored in ah quotient stored in al
;End - Get digit

add ah,30h ;Convert digit to ascii

;Start - Put digit on stack since we are getting numbers in reverse order
mov bx,00h
mov bl,ah
push bx
;End - Put digit on stack since getting numbers in reverse order

inc cx ;Increase the number of digits that need to be popped
cmp al,00h ;If end of number
jne ConvertNum

string: ;Puts the digits into a string to be outputted
pop ax
mov [di],al
```

# ASSEMBLY

## Q&A

主讲教师: 舒高峰

电子邮箱: [gaofeng.shu@henu.edu.cn](mailto:gaofeng.shu@henu.edu.cn)

联系电话: 13161693313