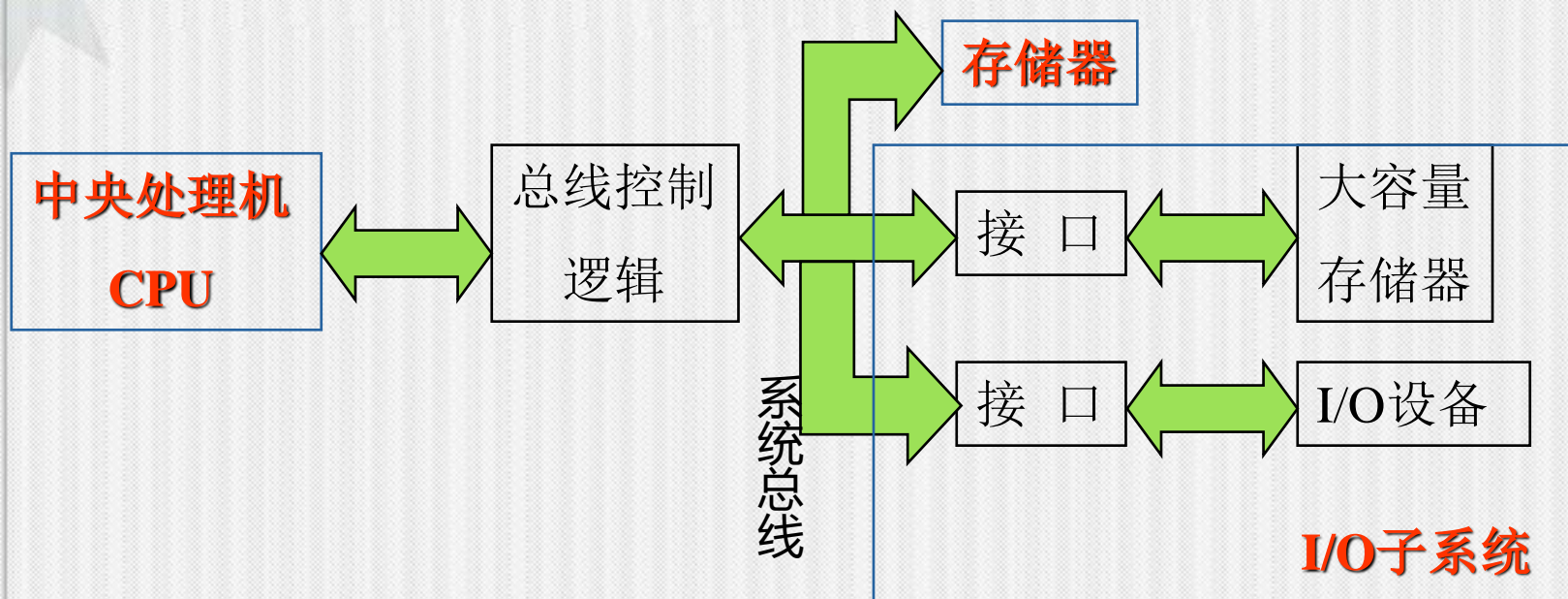


汇编语言程序设计

Assembly Language Programming

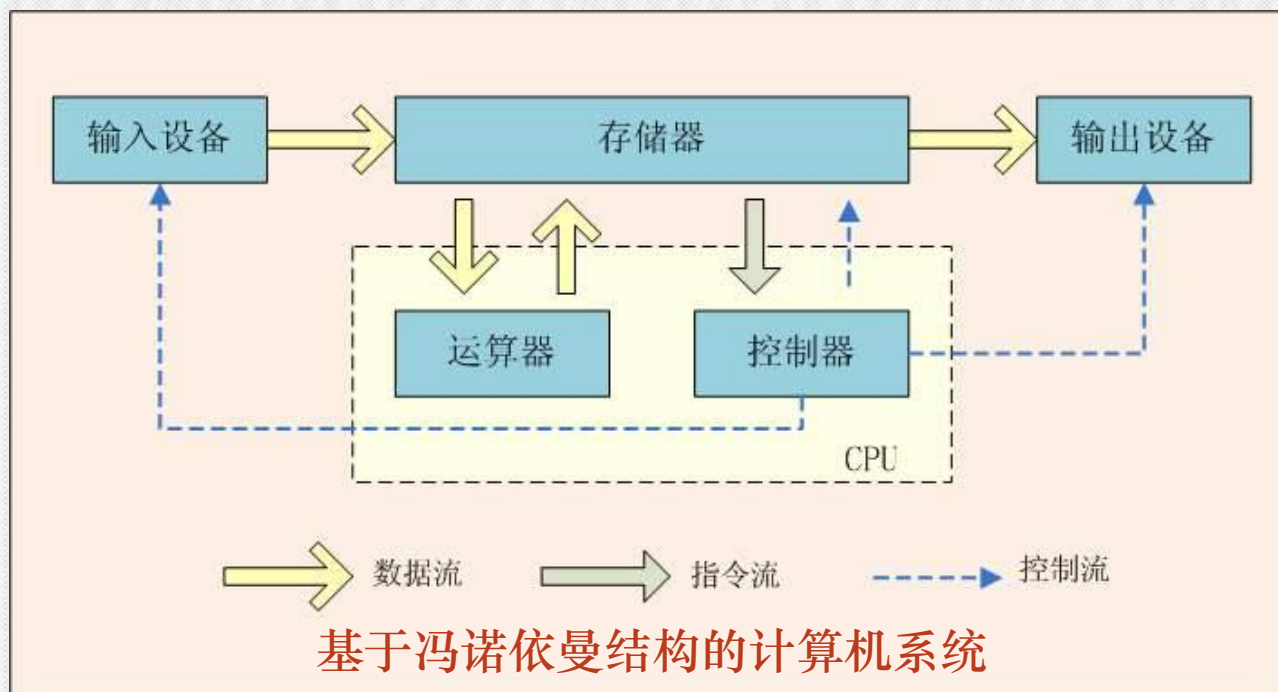
第一章 基础知识

§2 计算机基本结构



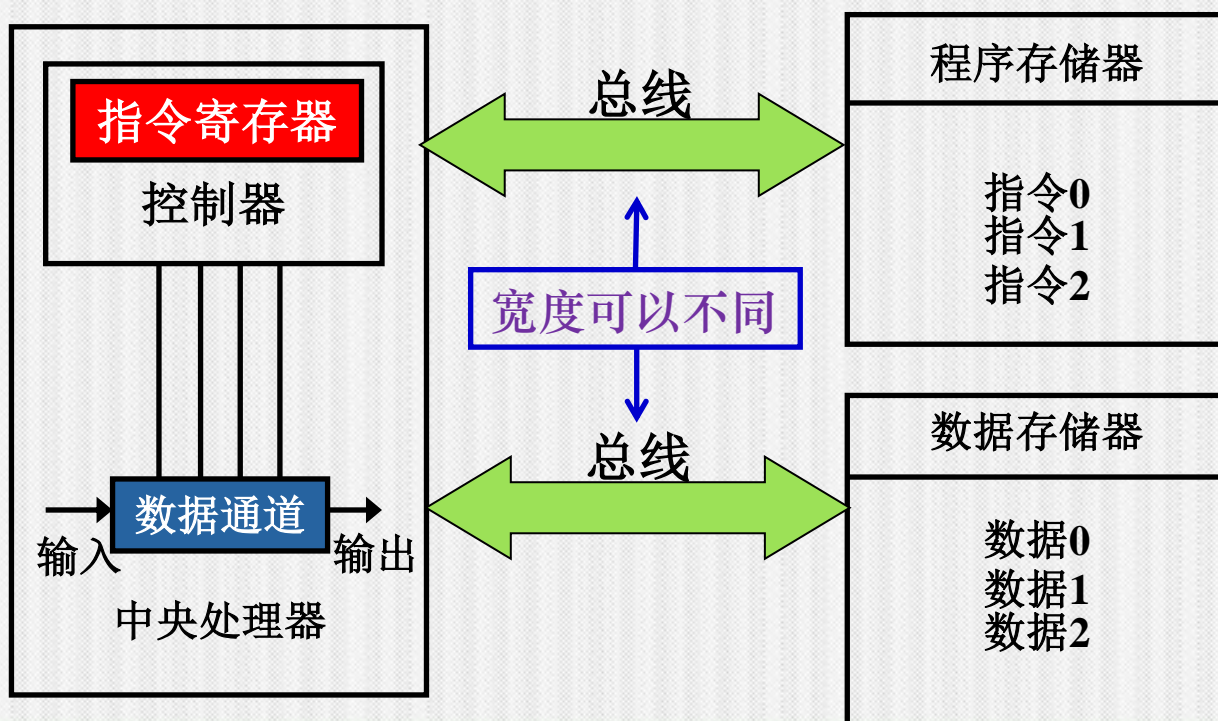
♥ 冯·诺依曼结构

- ❖ **核心思想**：将程序（指令序列的集合）和数据存放在同一存储器的不同地址；执行过程：取指令(or数据)→分析指令→执行指令。

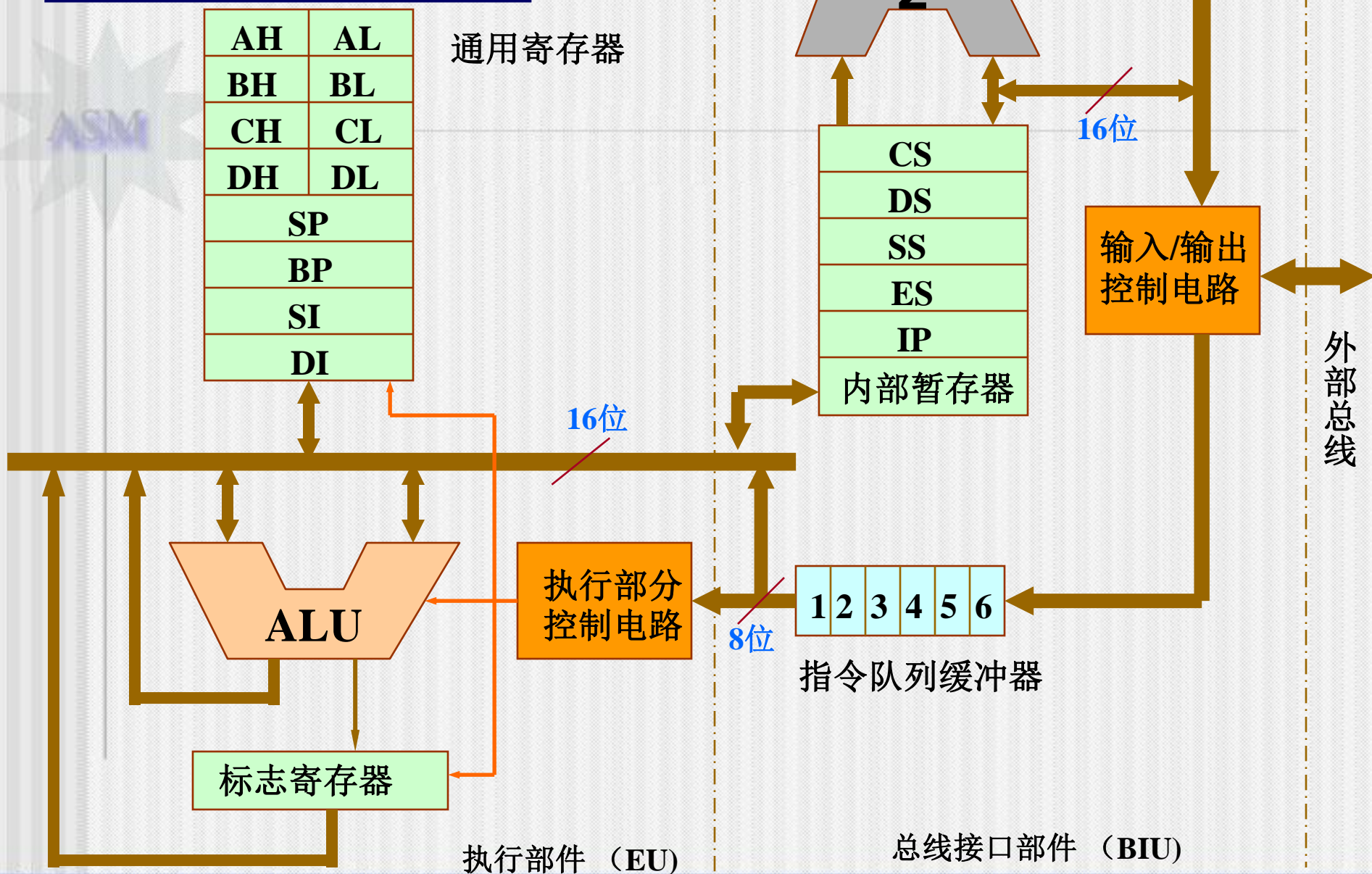


♥ 哈佛结构

- ❖ **核心思想**：将程序和数据存放在不同的存储器中；并行执行指令；执行过程：取指令 (**and**数据)→分析指令→执行指令。



8086的内部结构

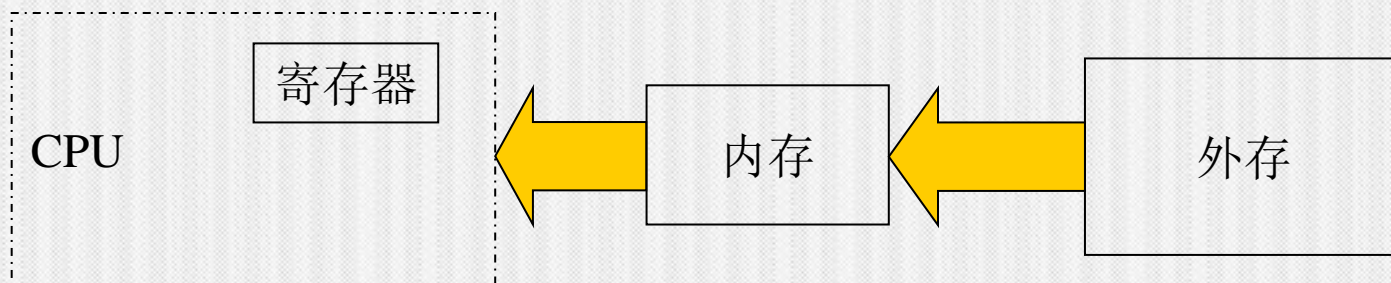


总线接口部件 (BIU)

执行部件 (EU)

2.1 8086 的寄存器组

- ♥ 8086 / 8088中共有14个16位寄存器
- ♥ 寄存器在CPU内部，所以访问速度快。但容量小





♥ 16位一分为二

AX

AH

AL

AX = 1234H

AH = 12H

AL = 34H

BX

BH

BL

CX

CH

CL

AH = CDH

AL = ABH

DX

DH

DL

AX = CDABH

指针寄存器:SP/BP

♥ 指针寄存器用于寻址内存堆栈内的数据

- ❖ SP (Stack Pointer)为堆栈指针寄存器，指示栈顶的偏移地址
- ❖ SP 不能再用于其他目的，具有专用目的
- ❖ BP (Base Pointer)为基址指针寄存器，表示数据在堆栈段中的基地址

♥ SP和BP寄存器与SS段寄存器联合使用以确定堆栈段中的存储单元地址

段寄存器:CS/DS/SS/ES

♥ 8086CPU的 4 个16位的段寄存器

- ❖ 代码段寄存器CS (Code Segment)
- ❖ 数据段寄存器DS (Data Segment)
- ❖ 堆栈段寄存器SS (Stack Segment)
- ❖ 附加数据段寄存器ES (Extra Segment)


♥ 段寄存器用来确定该段在内存中的起始地址。

♥ 用途特定，不可分开使用。

IP (Instruction Pointer)

- ♥ 指令指针寄存器IP，指示代码段中指令的偏移地址
- ♥ 它与代码段寄存器CS联用，确定下一条指令的物理地址
- ♥ 计算机通过CS：IP寄存器来控制指令序列的执行流程
- ♥ IP寄存器是一个专用寄存器，程序一般不可直接使用该寄存器。

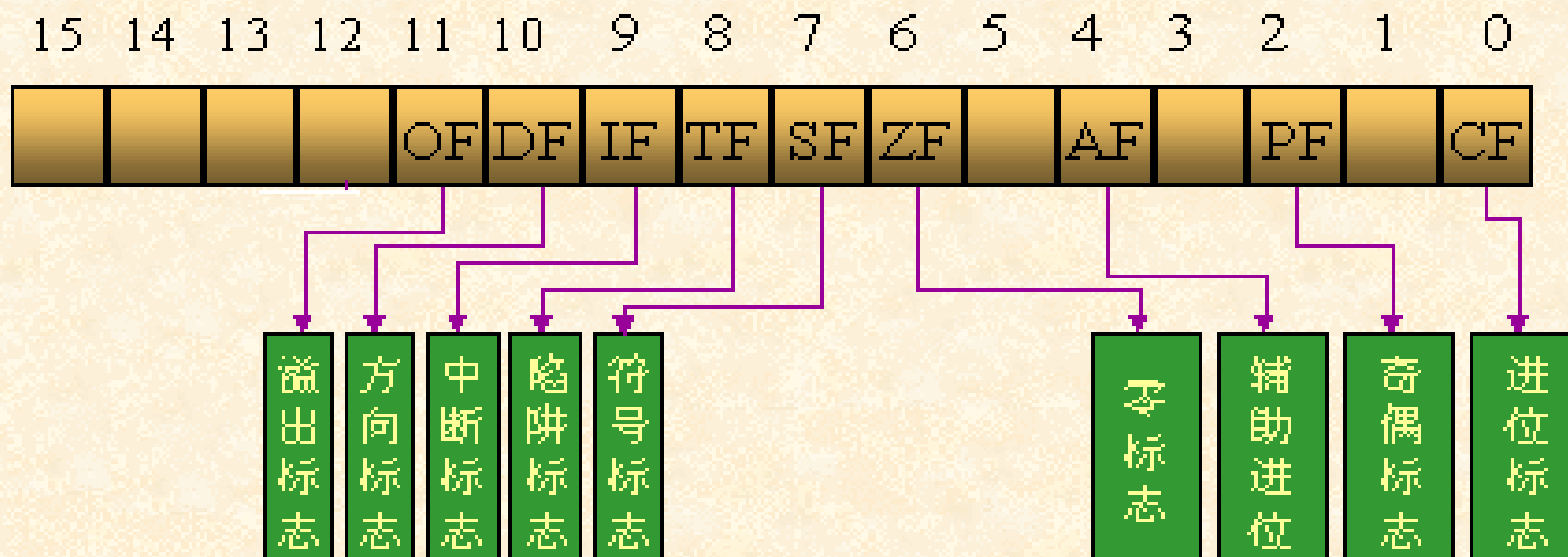
标志寄存器

- 
- 用途：标志寄存器F(FLAGS),又称程序状态字寄存器PSW, 是用以记录或存放状态标志和控制标志信息的。
- ❖ 状态标志位——用以记录当前运算结果的状态信息
 - ❖ 控制标志位——用以存放控制CPU工作方式的标志信息。

标志位

♥ 状态标志: CF, OF, ZF, SF, PF, AF

♥ 控制标志: DF, IF, TF



标志寄存器设置

- ♥ 状态标志位的设置是由CPU根据当前程序运行结果的状态**自动**完成的。状态标志位信息一般用作转移指令的转移控制条件。
- ♥ 控制标志位由程序设置，控制CPU的工作方式

进位标志CF (Carry Flag)

- ♥ 当运算结果的最高有效位有进位（加法）或借位（减法）时，进位标志置1，即 $CF = 1$ ；否则 $CF = 0$ 。

例如：

$$3AH + 7CH = B6H$$

没有进位： $CF = 0$

$$AAH + 7CH = (1) 26H$$

有进位： $CF = 1$

零标志ZF (Zero Flag)

♥ 若运算结果为0, 则ZF = 1, 否则ZF = 0

例如:

3AH + 7CH = B6H, 结果不是零:

ZF = 0

86H + 7AH = (1) 00H, 结果是零(为什么?):

ZF = 1

注意: ZF为1表示的结果是0

奇偶标志PF (Parity Flag)

♥ 当运算结果最低字节中 “1” 的个数为偶数时, $PF = 1$; 否则 $PF = 0$ 。

例如:

$3AH + 7CH = B6H = 10110110B,$

结果中有5个1, 是奇数: $PF = 0$

注意: PF标志仅反映最低8位中 “1” 的个数是偶或奇, 即使是进行16位字操作。

辅助进位标志AF (Auxiliary Carry Flag)

♥ 运算时 D_3 位（低半字节）有进位或借位时， $AF = 1$ ；否则 $AF = 0$ 。

例如：

$3AH + 7CH = B6H$

D_3 有进位： $AF = 1$

这个标志主要由处理器内部使用，用于十进制算术运算指令中，用户一般不必关心。（类似于进位标志）

符号标志SF (Sign Flag)

- ♥ 有符号数据利用最高有效位表示数据的符号。所以，最高有效位就是符号标志的状态。
- ♥ 运算结果最高位为1，则 $SF = 1$ ；否则 $SF = 0$ 。

例如：

$3AH + 7CH = B6H,$

最高位 $D_7 = 1$ ： $SF = 1$

$86H + 7AH = (1) 00H,$

最高位 $D_7 = 0$ ： $SF = 0$

溢出标志OF (Overflow Flag)

- ♥ 若算术运算的结果有溢出，则OF=1；否则 OF = 0。
- ♥ 只是对有符号数而言。对无符号数而言，OF = 1并不意味着结果出错。
- ♥ $80H - 01H = 7FH = 127$ ， OF = 1
 - ❖ 对于无符号数而言， $128 - 1 = 127$ ，正确
 - ❖ 对于有符号数而言， $-128 - 1 = 127$ ，错误

什么是溢出

- ♥ 处理器内部以补码表示有符号数
 - ❖ 8个二进制位能够表达的整数范围是：+127 ~ -128
 - ❖ 16位表达的范围是：+32767 ~ -32768
- ♥ 如果运算结果超出了这个范围，就是产生了溢出
- ♥ 有溢出，说明有符号数的运算结果不正确
- ♥ 无符号数有溢出吗？
 - ❖ $\text{FFH} + 01\text{H} = 00\text{H}$, $\text{CF} = 1$, 进位溢出

溢出的判断

♥ 方法一：

- ❖ 两个正补码数相加结果为负
- ❖ 两个负补码数相加结果为正
- ❖ 正补码数 - 负补码数为负
- ❖ 负补码数 - 正补码数为正
- ❖ 其他情况

OF = 1

OF = 0

♥ 方法二（CPU）：

- ❖ 如果最高位与次高位**同时**向前有或无进/借位，则
OF = 0;
- ❖ 如果最高位与次高位**不同时**向前有或无进/借位，则
OF = 1; (?)

如何运用溢出和进位

- ♥ 处理器对两个操作数进行运算时，并不知道操作数是有符号数还是无符号数，所以全部设置，按各自规则。
- ♥ 应该利用哪个标志，则由程序员来决定。
 - ❖ 将参加运算的操作数是无符号数，就应该关心CF；
 - ❖ 将参加运算的操作数是有符号数，则要注意是否溢出。
- ♥ 我怎么知道是什么数？
 - ❖ 除了你没人知道，☺

例: MOV AX, 1

MOV BX, 2

ADD AX, BX

指令执行后, (AX)=3, OF=0, CF=0, ZF=0, SF=0

例: MOV AX, 0FFFFH

MOV BX, 1

ADD AX, BX

指令执行后, (AX)=0, OF=0, CF=1, ZF=1, SF=0

方向标志DF (Direction Flag)

♥ 用于串操作指令中，控制地址的变化方向：

- ❖ 设置DF = 0，串操作的存储器地址自动增加；
- ❖ 设置DF = 1，串操作的存储器地址自动减少。

中断允许标志IF (Interrupt-enable Flag)

♥ 用于控制外部可屏蔽中断是否可以被处理器响应：

- ❖ 设置 $IF = 1$ ，则允许中断；
- ❖ 设置 $IF = 0$ ，则禁止中断。

陷阱标志TF (Trap Flag)

♥ 用于控制处理器是否进入单步操作方式：

- ❖ 设置TF = 0，处理器正常工作；
- ❖ 设置TF = 1，处理器单步执行指令。

♥ 单步执行指令——处理器在每条指令执行结束时，便产生一个编号为1的内部中断。这种内部中断称为单步中断，所以TF也称为单步标志。

- ❖ 利用单步中断可对程序进行逐条指令的调试。
- ❖ 这种逐条指令调试程序的方法就是单步调试。

2.2 8086 存储器组织

- ♥ 内存是存放指令和数据的部件，由若干内存单元构成。
- ♥ 80x86的内存以字节编址：每个内存单元有唯一的地址，可存放1个字节。
- ♥ 内存单元的2个要素：地址（编号）与值（内容）。
 - ❖ $(100H) = 34H$
- ♥ 1个字占据2个相邻的内存单元；
 - ❖ 低字节在低地址单元，高字节在高地址单元；字的地址由其低地址来表示。双字也类似。
 - ❖ 同一地址可以看作是字节、字或双字单元的地址。（?）

地址组织方式

物理地址=绝对地址: **15** (第**15**号房间)

21	22	23	24	25	26	27	28	29	30
11	12	13	14	15	16	17	18	19	20
01	02	03	04	05	06	07	08	09	10

逻辑地址=相对地址: **205** (2层**05**号房间)

301	302	303	304	305	306	307	308	309	310
201	202	203	204	205	206	207	208	209	210
101	102	103	104	105	106	107	108	109	110

分段管理

- ♥ 将存储器分成若干个逻辑段
- ♥ 段首地址必须为：****0H。其有效地址 “****H” 存放在段寄存器中，称为段地址。
- ♥ 段中某一个单元相对于段首的距离称为偏移地址，偏移地址存放在偏移地址寄存器中。
- ♥ 段的长度不超过 $2^{16}=64\text{K}$ 。（仅能访问 $4*64\text{K}$? ）

地址概念

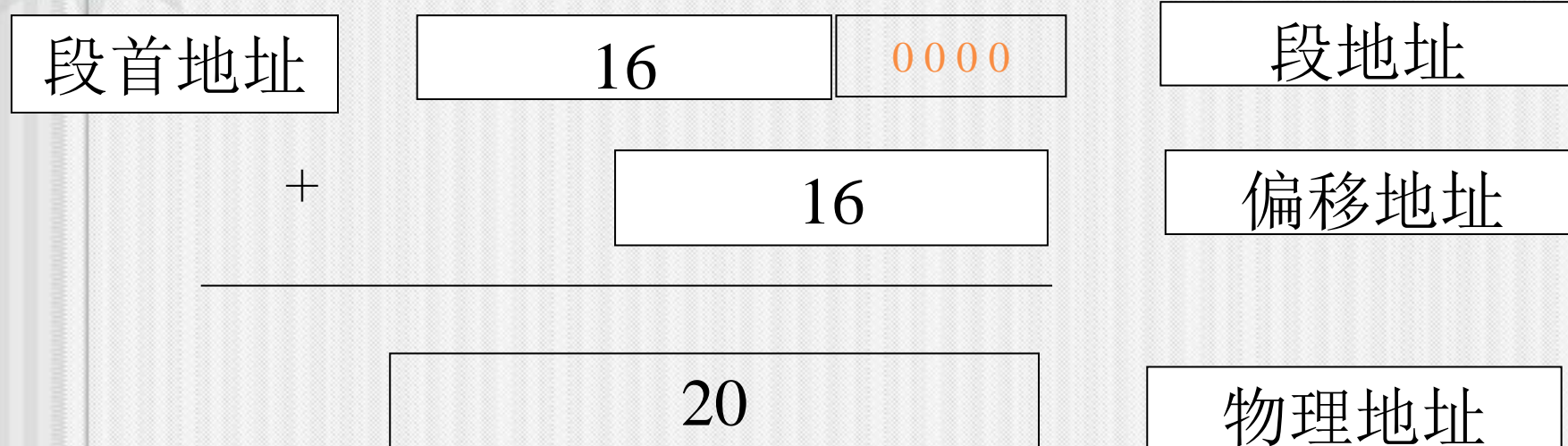
♥ 物理地址

- ❖ 20位，20位地址总线上产生的唯一的地址。

♥ 逻辑地址 - - 程序指令中引用和操作的地址

- ❖ 段地址：16位，段首地址的有效地址
- ❖ 偏移地址：段中某一个单元相对于段首的距离，16位

对比



♥ 物理地址 = 10H × 段地址 + 偏移地址

CPU形成物理地址的过程

- ♥ 地址加法器
- ♥ 段地址一般在程序开始时预定
- ♥ 程序中给出16位偏移地址
- ♥ CPU几种典型的操作
 - ❖ 取指令：指令单元地址 = $(CS) \times 10H + IP$
 - ❖ 堆栈操作：堆栈数据地址 = $(SS) \times 10H + \text{偏移}$
 - ❖ 内存数据：内存数据地址 = $(DS) \times 10H + \text{偏移}$

§ 3 寻址方式

♥ 机器码格式：

是将指令以2进制数0和1进行编码的形式

操作码

Mod Reg R/M

位移量

立即数

- ❖ 操作码说明计算机要执行哪种操作，
- ❖ **Mod Reg R/M** : 表明寻找操作数的方式。

3.1 指令格式

♥ 指令的一般格式

【标号：】	操作码	操作数1	操作数2	【； 注释】
-------	-----	------	------	--------

- ❖ **操作码**说明计算机要执行哪种操作，它是指令中不可缺少的组成部分
- ❖ **操作数**是指令执行的参与者，即各种操作的对象，为数据及数据所在地址。

3.1 指令格式

♥ 操作数的形式

- ❖ 立即操作数：指令的操作数是立即数（常量），只能是源操作数。
- ❖ 寄存器操作数：操作数存放在寄存器中值，指令中使用寄存器名。
- ❖ 内存操作数：操作数存放在内存中，指令中给出内存地址，通常为有效地址EA，段地址在某个段寄存器中。

3.2 内存操作数寻址方式

- ♥ 直接寻址方式 (direct addressing)
- ♥ 寄存器间接寻址方式 (register indirect)
- ♥ 寄存器相对寻址方式 (register relative)
- ♥ 基址变址寻址方式 (based indexed..)
- ♥ 相对基址变址方式 (relative based indexed..)

直接寻址方式(direct addressing)

♥ 内存操作数的偏移地址由指令直接给出

```
MOV AX, [2000H]
```

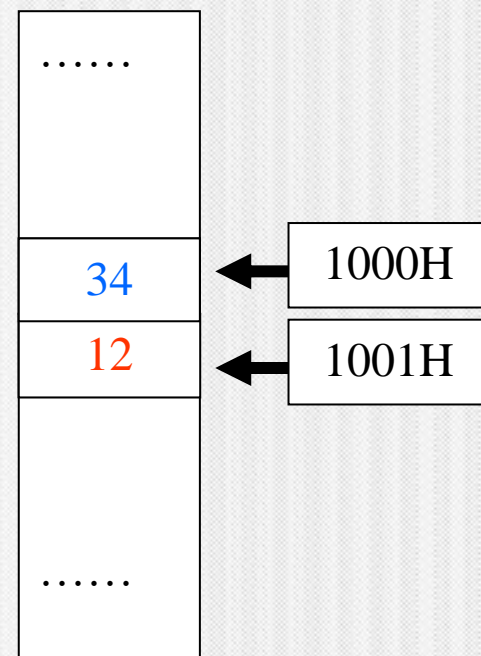
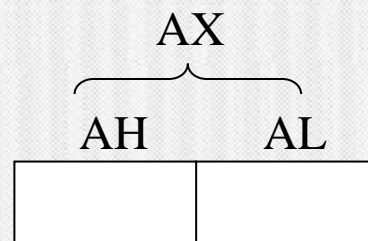
```
MOV WORD PTR [1000H], -1
```

```
MOV AX, Y
```


比 较

♥ 比较1

- ❖ MOV AL, [1000H]
- ❖ AL=34H;
- ❖ MOV AX, [1000H]
- ❖ AX=1234H



♥ 比较2

- ❖ MOV AX, 1000H
- ❖ AX=1000H;
- ❖ MOV AX, [1000H]
- ❖ AX=1234H

寄存器间接寻址方式(register indirect)

- ♥ 指定某个地址寄存器 (SI、DI、BX、BP) 的内容作为内存操作数的偏移地址

MOV AX, [BX]

MOV [BP], AL

- ♥ 使用场合：表格、字符串、缓冲区处理

寄存器相对寻址方式(register relative)

- 指令中指定地址寄存器 (SI、DI、BX、BP) 与一个位移量相加作为内存操作数的偏移地址

$$\text{偏移地址} = \begin{cases} (\text{BX}) \\ (\text{BP}) \\ (\text{SI}) \\ (\text{DI}) \end{cases} + \begin{cases} 8\text{位} \\ 16\text{位} \end{cases} \text{位移量}$$

MOV AX, [SI+2]

MOV [BP-6H], AL

- 使用场合：适于表格、字符串、缓冲区的处理；
一维数组方式($\text{DATA}[\text{DI}] = [\text{DATA} + \text{DI}]$)

基址变址寻址方式(based indexed..)

- ♥ 指定基址寄存器(**BX,BP**)、变址寄存器(**SI,DI**)内容相加作为**内存操作数**的地址。

MOV [BX+DI],DX MOV AL, [BP+SI]

- ♥ 使用场合：适于数组、字符串、表格的处理，更加灵活

- ♥ 注意：

- ❖ 必须是一个基址寄存器和一个变址寄存器的组合

- MOV AX, [BX][BP] (×)
- MOV AX, [SI][DI] (×)

相对基址变址方式(relative based indexed..)

- 指定基址寄存器(**BX, BP**)、变址寄存器(**SI, DI**)、位移量之和作为**内存操作数**的地址

$$\text{偏移地址} = \begin{cases} (\text{BX}) \\ (\text{BP}) \end{cases} + \begin{cases} (\text{SI}) \\ (\text{DI}) \end{cases} + \begin{cases} 8\text{位} \\ 16\text{位} \end{cases} \text{位移量}$$

MOV AL, [SI+BX+2] MOV AL, 2[SI+BX]

MOV [BX+DI-16H], DX

- 使用场合：适于二维数组的寻址
(Buffer[BX][SI] = [Buffer+BX+SI])

3.3 段超越

- ♥ 隐式段地址——8086/8088指令系统对存储单元的访问，其段地址都是从系统事先约定好的段寄存器中获取
- ♥ 若出现BP (SP)，默认在SS中，否则所有的操作都默认在DS中。
- ♥ (显式段地址) 段超越——不是按照系统的约定，而是在指令中显式指定某一段寄存器作为存储器操作数的段地址。