第4章　指令系统与汇编语言

本章讨论的内容主要包括：

- ✓ 指令格式
- ✓ 寻址方式
- ✓ PC指令系统
- ✓ 汇编语言程序设计
- ✓ CISC与RISC

4.1 指令格式

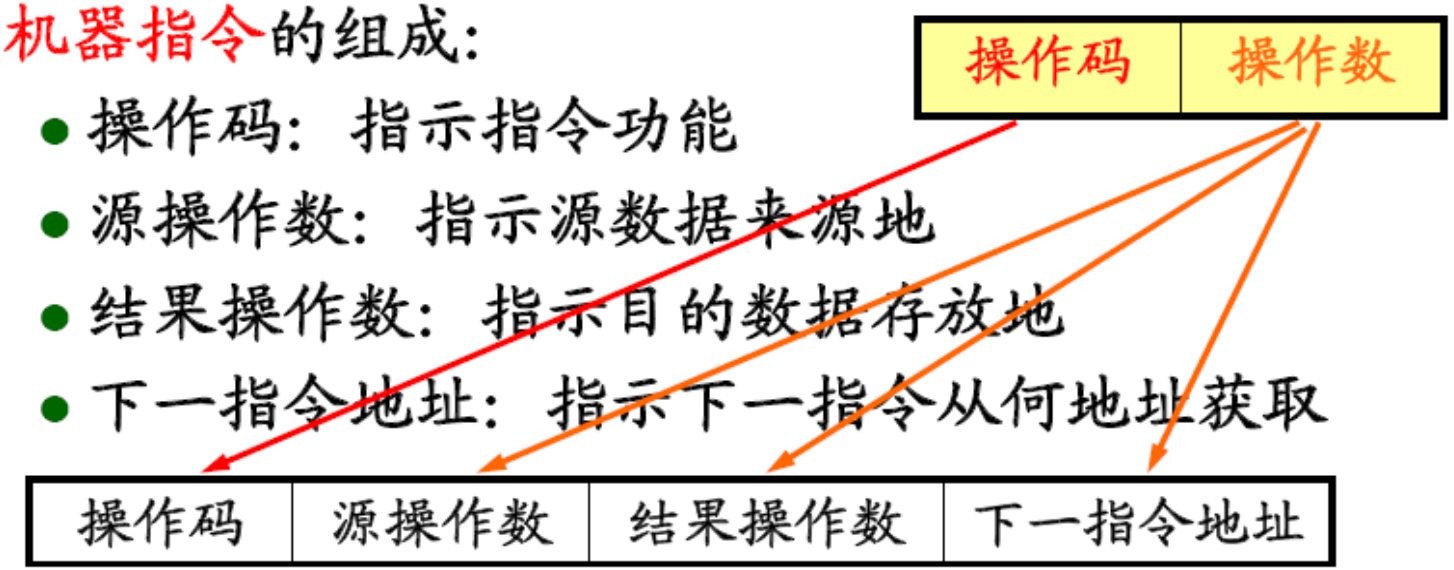
4.1.1 概述

- 机器指令与指令系统
  - ✓ 计算机软件、硬件的界面之一
  - ✓ 计算机的性能与它所设置的指令系统有很大的关系，而指令系统的设置又与机器的结构密切相关
- 指令系统的发展
  - ✓ CISC (Complex Instruction Set Computer)
    复杂指令系统计算机
  - ✓ RISC (Reduced Instruction Set Computer)
    精简指令系统计算机
- 设置指令系统的要求
  - ✓ 完备性、有效性、规整性、兼容性

4.1.2 指令格式的描述

1. 指令信息

- 机器指令的组成：
  - 操作码：指示指令功能
  - 源操作数：指示源数据来源地
  - 结果操作数：指示目的数据存放地
  - 下一指令地址：指示下一指令从何地址获取



基本的指令格式

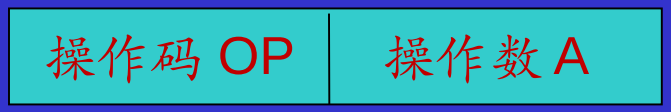
- 机器指令在计算机中是用二进制编码表示的。

00000101	34H	12H
11111111	mod 010 r/m	

汇编指令

```
ADD   AX，1234H
CALL  WORD PTR [BX]
```

2. 指令格式

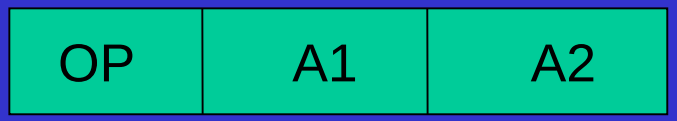


1）三地址指令格式

(A1) OP (A2) => (A3)

例如： ADDT r1，r2，r3 ； *r1+ r2 → r3*

2）二地址指令格式



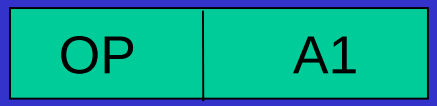
(A1) OP (A2) => (A2)

例如： ADD r1，r2 ； *r1+ r2 → r1*

MOV r3，r1 ； *r1 → r3*

2. 指令格式

3）一地址指令格式



(AC) OP (A) => (AC)

例如：
```
LOAD   r1   ;  r1 → AC
ADDS   r2   ;  AC + r2 → AC
STOR   r3   ;  AC → r3
```

4）零地址指令格式

无需操作数 或 操作数是默认的

5）多地址指令格式

4.1.3 指令设计

1. 指令长度的影响因素

- ✓ 内存大小与组织
- ✓ CPU数据总线宽度
- ✓ CPU内部寄存器的数量
- ✓ 寻址方式
- ✓ 指令数量

2. 指令长度与字长的关系

- 构成一条指令的二进制位数称为指令长度。从规整性的角度出发，指令长度都是字节（或16、32、64位的字长）的整倍数。

- 字长
- 指令长度与字长的关系
  - ✓ 短格式指令
  - ✓ 长格式指令
- 指令长度变长或定长
  - ✓ 定长操作码，变长指令码
  - ✓ 变长操作码，定长指令码

3. 哈夫曼（Huffman）编码

指令操作码 OP 用来表示指令所完成的功能。

用不同的二进制编码来设定指令操作码，每种编码对应一条指令。

- ✓ 固定长度编码
- ✓ 哈夫曼编码
- ✓ 扩展编码

- 这里用实例介绍固定长度编码，以及哈夫曼编码

例：假设一台模型计算机共有7种不同的操作码，已知各种操作码在程序中出现的概率如下表，利用固定长度编码法进行操作码编码。

指令	I_1	I_2	I_3	I_4	I_5	I_6	I_7
概率	0.45	0.30	0.15	0.05	0.03	0.01	0.01

解：由于N=7 因此，指令操作码固定长度为

[lbN]=[lb7]=3

编码结果：

指令序号	概率	编码	操作码长度
I_1	0.45	000	3位
I_2	0.30	001	3位
I_3	0.15	010	3位
I_4	0.05	011	3位
I_5	0.03	100	3位
I_6	0.01	101	3位
I_7	0.01	110	3位

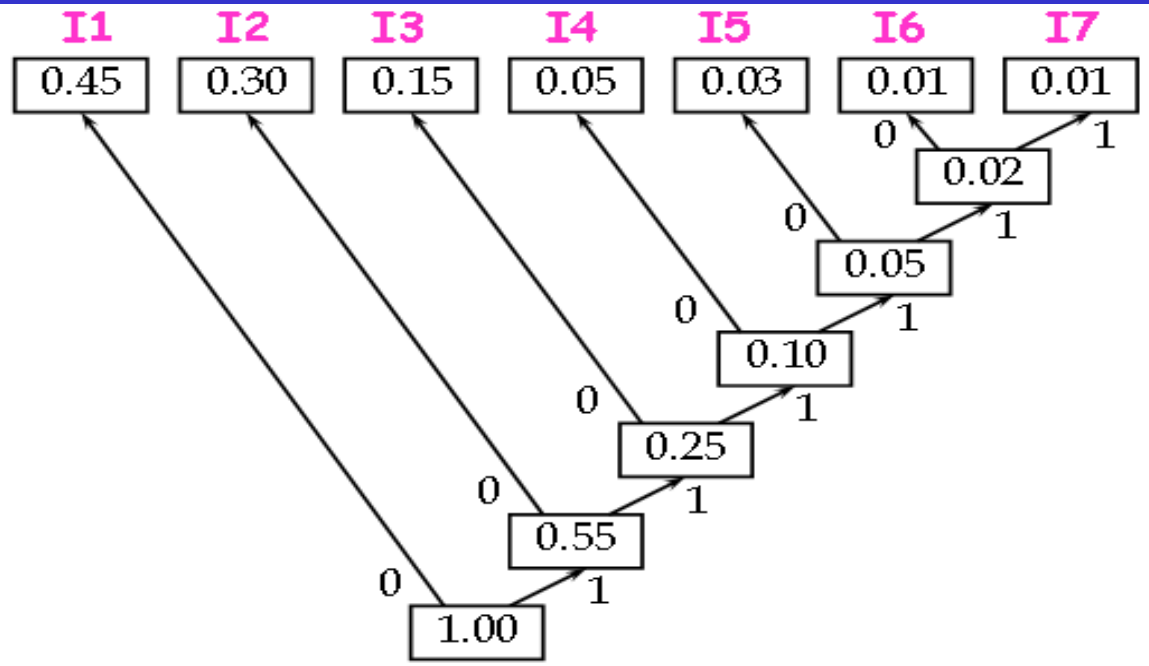
指令译（解）码：三—八译码器

- 哈夫曼编码原理：

  使用频率高的指令采用短操作码

例：利用Huffman 编码法进行操作码编码。

指令	I_1	I_2	I_3	I_4	I_5	I_6	I_7
概率	0.45	0.30	0.15	0.05	0.03	0.01	0.01



Huffman编码树生成过程

编码结果：

指令序号	概率	Huffman编码法	操作码长度
I_1	0.45	0	1位
I_2	0.30	10	2位
I_3	0.15	110	3位
I_4	0.05	1110	4位
I_5	0.03	11110	5位
I_6	0.01	111110	6位
I_7	0.01	111111	6位

采用Huffman编码法的操作码平均长度：

$0.45 \times 1 + 0.30 \times 2 + 0.15 \times 3 + 0.05 \times 4 + 0.03 \times 5 + 0.01 \times 6 + 0.01 \times 6$
$= 1.97$（位）

最优Huffman编码法的操作码平均长度计算公式：

$$H = \sum_{i=1}^{n} (P_i \cdot I_i) = -\sum_{i=1}^{n} [P_i \cdot \log_a (P_i)]$$

$\alpha=2$

所以，采用最优Huffman编码法的操作码平均长度为：

$0.45 \times 1.152 + 0.30 \times 1.737 + 0.15 \times 2.737 + 0.05 \times 4.322 + 0.03 \times 5.059 +$
$0.01 \times 6.644 + 0.01 \times 6.644 = 1.95$（位）

Huffman操作码的主要缺点：

1) 操作码长度很不规整，硬件译码困难
2) 与地址码共同组成固定长的指令比较困难

解决方法之一：　指令操作码的扩展技术

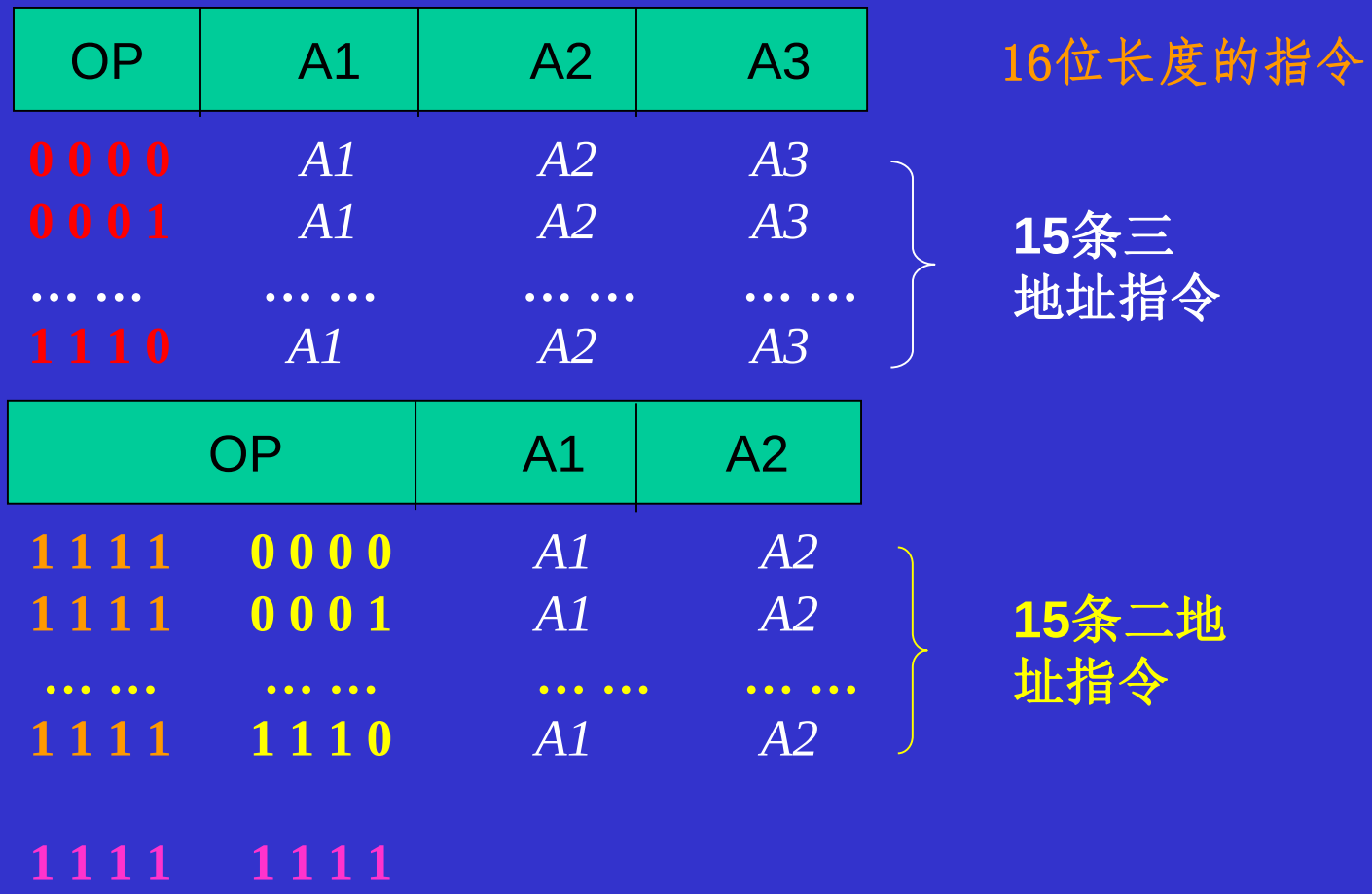
4. 扩展编码方式

OP	A1	A2	A3
0 0 0 0	*A1*	*A2*	*A3*
0 0 0 1	*A1*	*A2*	*A3*
……	……	……	……
1 1 1 0	*A1*	*A2*	*A3*
1 1 1 1			

16位长度的指令

15条三地址指令

4. 扩展编码方式


OP
A1
A2
A3
16位长度的指令
0 0 0 0
A1
A2
A3
0 0 0 1
A1
A2
A3
… …
… …
… …
… …
1 1 1 0
A1
A2
A3
15条三
地址指令
OP
A1
A2
1 1 1 1
0 0 0 0
A1
A2
1 1 1 1
0 0 0 1
A1
A2
… …
… …
… …
… …
1 1 1 1
1 1 1 0
A1
A2
15条二地
址指令
1 1 1 1
1 1 1 1

4. 扩展编码方式


OP
A1
A2
A3
16位长度的指令
0 0 0 0 A1 A2 A3
0 0 0 1 A1 A2 A3
1 1 1 0 A1 A2 A3
15条三地址指令
OP
A1
A2
1 1 1 1 0 0 0 0 A1 A2
1 1 1 1 0 0 0 1 A1 A2
1 1 1 1 1 1 1 0 A1 A2
15条二地址指令
OP
A1
1 1 1 1 1 1 1 1 0 0 0 0 A1
1 1 1 1 1 1 1 1 0 0 0 1 A1
1 1 1 1 1 1 1 1 1 1 1 1 A1
16条一地址指令

4. 扩展编码方式

OP	A1	A2	A3

16位长度的指令

0 0 0 0	*A1*	*A2*	*A3*
0 0 0 1	*A1*	*A2*	*A3*
……	……	……	……
1 1 1 0	*A1*	*A2*	*A3*

15条三地址指令

1 1 1 1	0 0 0 0	*A1*	*A2*
1 1 1 1	0 0 0 1	*A1*	*A2*
……	……	……	……
1 1 1 1	1 1 1 0	*A1*	*A2*

15条二地址指令

1 1 1 1	1 1 1 1	0 0 0 0	*A1*
1 1 1 1	1 1 1 1	0 0 0 1	*A1*
……	……	……	……
1 1 1 1	1 1 1 1	1 1 1 1	*A1*

16条一地址指令

- 这只是其中一种扩展方法，还有其他多种扩展方法。

操作码编码	说明
0000 0001 …… 1110	4位长度的操作码共有15种
1111 0000 1111 0001 …… 1111 1110	8位长度的操作码共有15种
1111 1111 0000 1111 1111 0001 …… 1111 1111 1110	12位长度的操作码共有15种

等长15/15/15...扩展法

操作码编码	说明
0000 0001 …… 0111	4位长度的操作码共有8种
1000 0000 1000 0001 …… 1111 0111	8位长度的操作码共有64种
1000 1000 0000 1000 1000 0001 …… 1111 1111 0111	12位长度的操作码共有512种

等长8/64/512...扩展法

- 扩展编码方式是一种重要的指令优化技术，可以缩短指令平均长度，减少程序总位数和增加指令字所能表示的操作信息。
- 一般用在指令字长较短的微、小型机上。

4.2 寻址方式（编址方式）

指令操作数 A 用来指出指令的操作对象。

操作数可能在运算器中的某个寄存器中或存储器中，也可能就在指令中。

指令中以什么方式提供操作数或操作数地址，称为**寻址方式**(或**编址方式**)。

- 不同的计算机有不同的寻址方式，但其基本原理是相同的。在一些计算机中，某些寻址方式还可以组合使用，从而形成更复杂的寻址方式。
- 这里以8086CPU为例介绍几种常用的寻址方式。

4.2 寻址方式（编址方式）

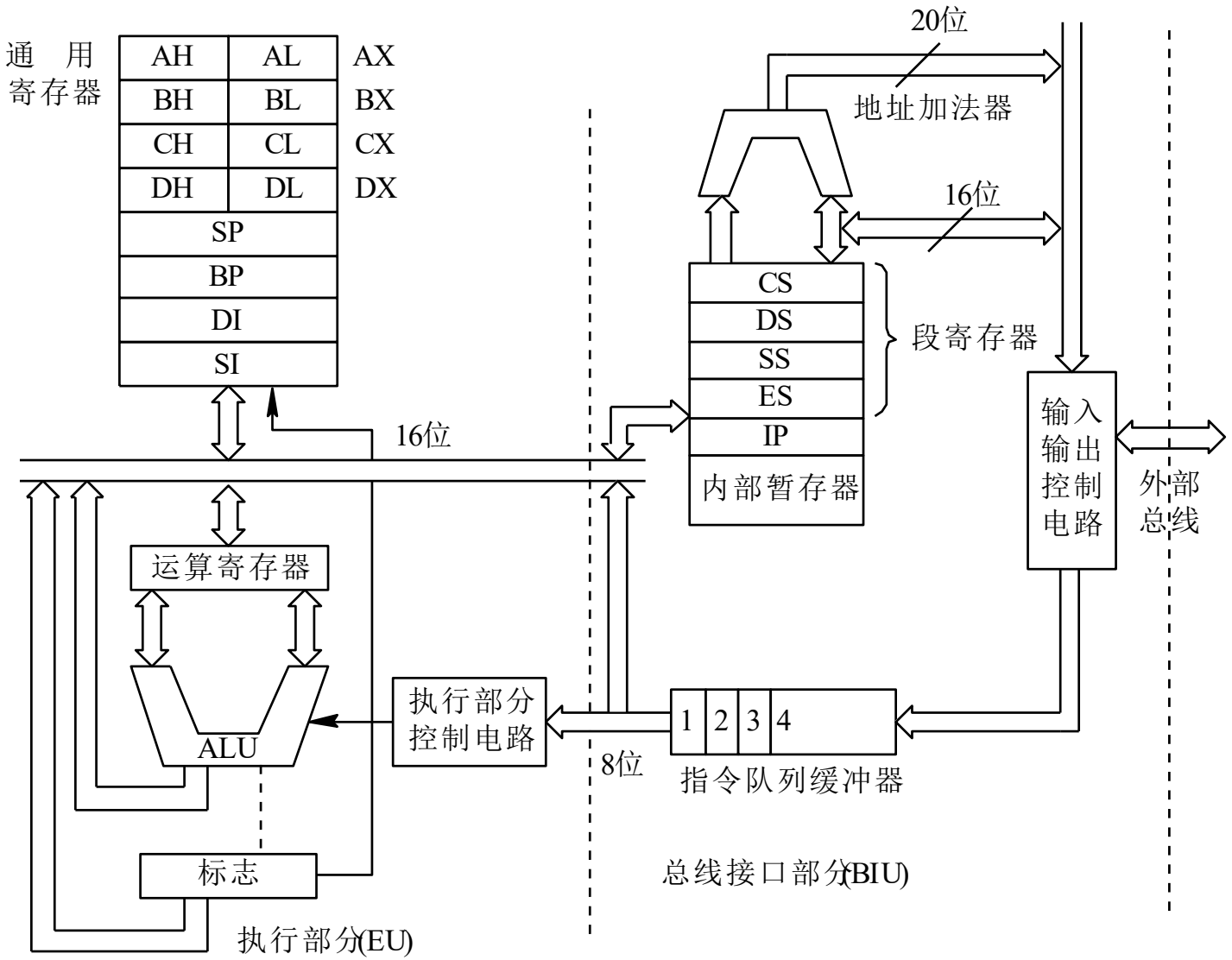
- 这里以8086/8088CPU为例介绍几种常用的寻址方式。
- 为了说明8086/8088处理器的寻址方式及指令系统，在此首先介绍处理器中与应用有关的、用户能用指令访问的**内部寄存器**。

4.2.1 8088CPU的内部寄存器

1. 8088CPU的内部结构

8088微处理器内部分为两个部分：

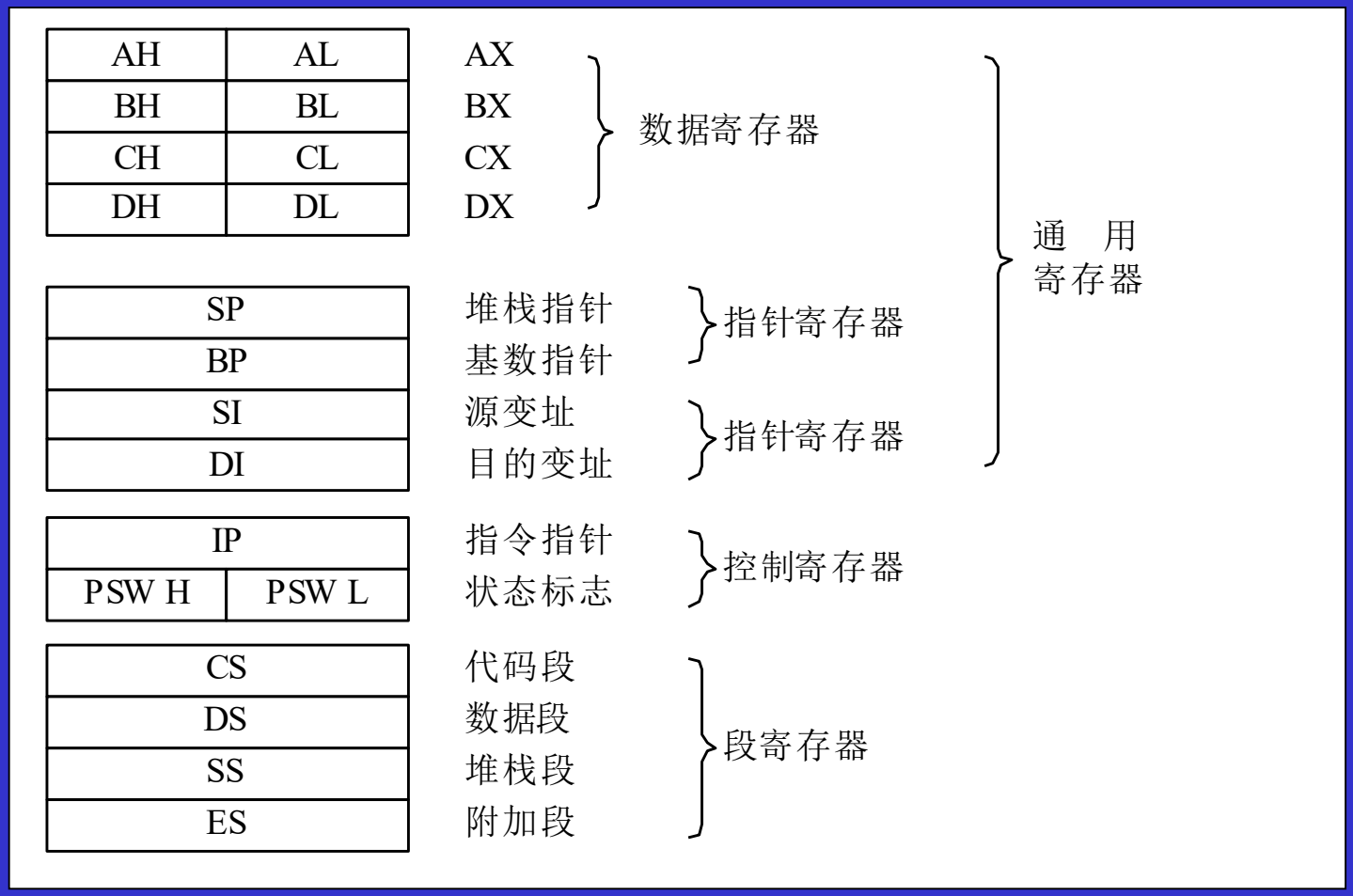
执行单元(EU)、
总线接口单元(BIU)



8088CPU的内部结构

2. 8088处理器中的内部寄存器

在8088处理器中，用户能用指令改变其内容的，主要是一组内部寄存器，其结构如图：



1) 数据寄存器

有4个16位的数据寄存器AX、BX、CX和DX，可以存放16位的操作数。

也可以作为8个8位寄存器AH、AL、BH、BL、CH、CL和DH、DL，可以存放16位的操作数。

2) 指针寄存器

有两个：SP和BP。

- ✓ SP是堆栈指针寄存器，由它和堆栈段寄存器一起来确定堆栈在内存中的位置。
- ✓ BP是基数指针寄存器，通常用于存放基地址，以使8088的寻址更加灵活。

3) 变址寄存器

SI是源变址寄存器

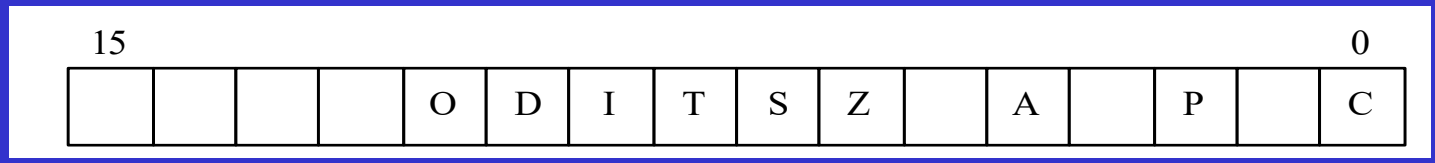
DI是目的变址寄存器，都用于指令的变址寻址。

4)控制寄存器

有两个：IP，PSW

- ✓ IP是指令指针寄存器，用来控制CPU的指令执行顺序。
- ✓ PSW是处理机状态字，也有人叫它为状态寄存器或标志寄存器，用来存放8088CPU在工作过程中的状态。

PSW各位标志如图所示:

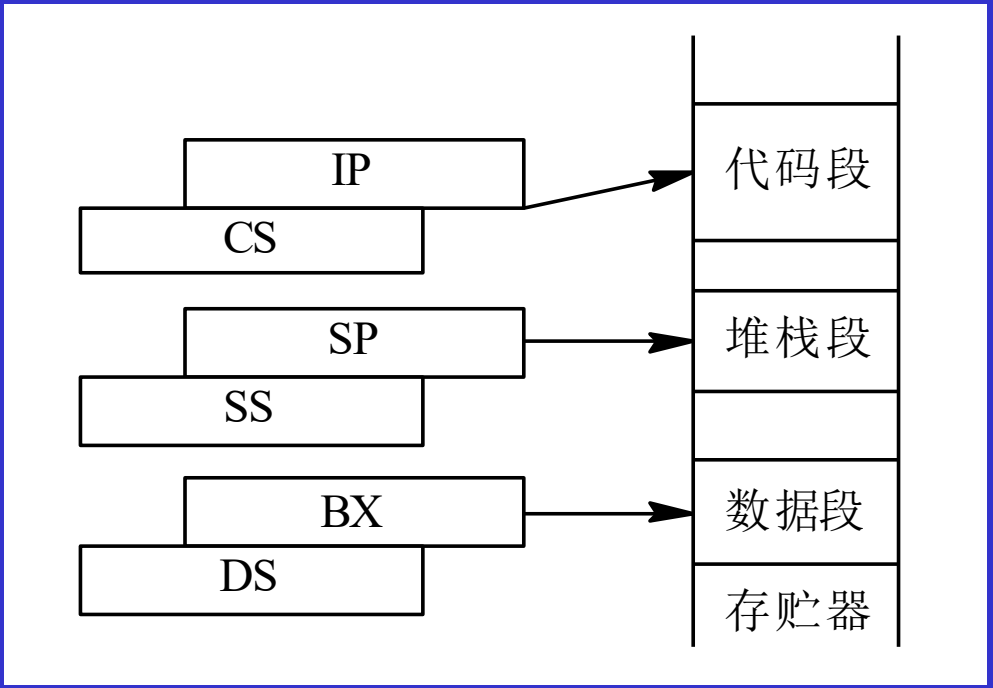


C —进位标志位
A —半加标志位
S —符号标志位
I —中断允许标志位
O —溢出标志位
P —奇偶标志位
Z — 零标志位
T —陷阱标志位(单步标志位)
D —方向标志位

5) 段寄存器

有4个段寄存器：

- ✓ CS 代码段寄存器
- ✓ DS 数据段寄存器
- ✓ SS 堆栈段寄存器
- ✓ ES 附加段寄存器



3. 存贮器寻址

补充： *8086存储器的分段管理*

- *存贮器的物理地址*

以字节为最小基本存储单元的顺序编址。（2^{20}=1024K=1MB 空 间） 从 00000H 到 0FFFFFH个单元的20位绝对地址。

- *存贮器的逻辑地址*

将1MB空间，以小于等于2^{16}=64K连续的存储器为一段，分为多个段。每个段可以独立寻址。

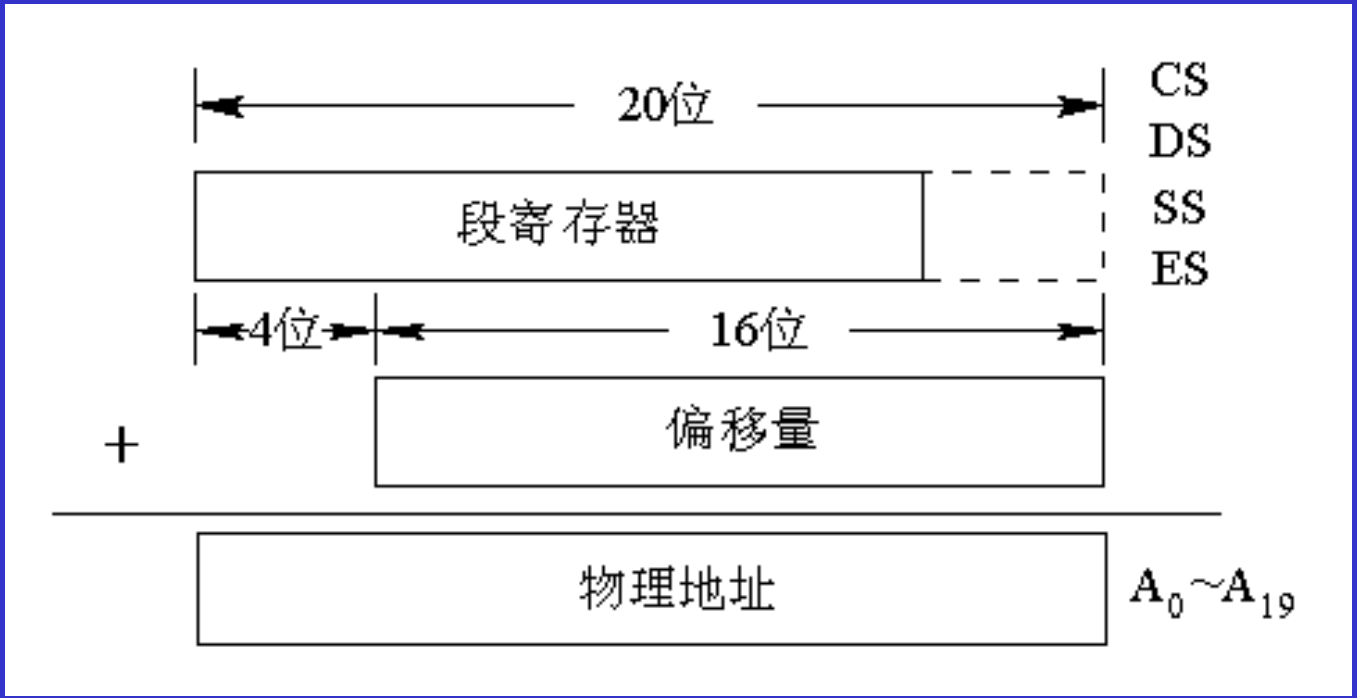
*段地址：段内偏移地址*的表示方法称逻辑地址。

如， 2000H:100H

3. 存贮器寻址

将**逻辑地址**转化为**物理地址**（绝对地址）的计算公式：

物理地址 = 段寄存器的内容×16 + 偏移地址



3. 存贮器寻址

8086存储器的分段管理

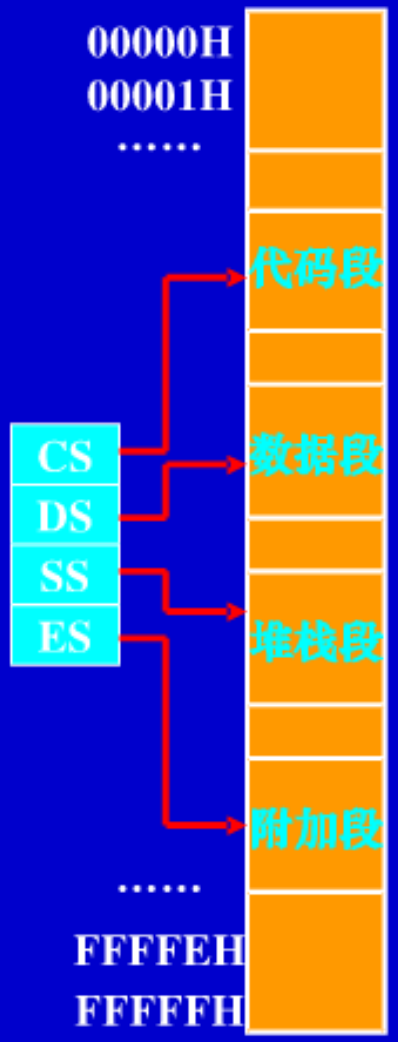
段地址：段的起始地址

偏移地址：段内地址的顺序编号。

8086同时可有4个段被激活（称当前段）。它们是CS代码段、DS数据段、SS堆栈段、ES附加数据段。CPU访问内存中这4段时，逻辑地址按以下方式提供：

代码段	CS：IP
堆栈段	SS：SP 或 SS：偏移地址
数据段	DS：偏移地址
附加数据段	ES：偏移地址

偏移地址由EU部件算出，又称为*有效地址（EA）*，计算方法与指令的寻址方式有关



3. 存贮器寻址

例：对于物理地址01023H单元

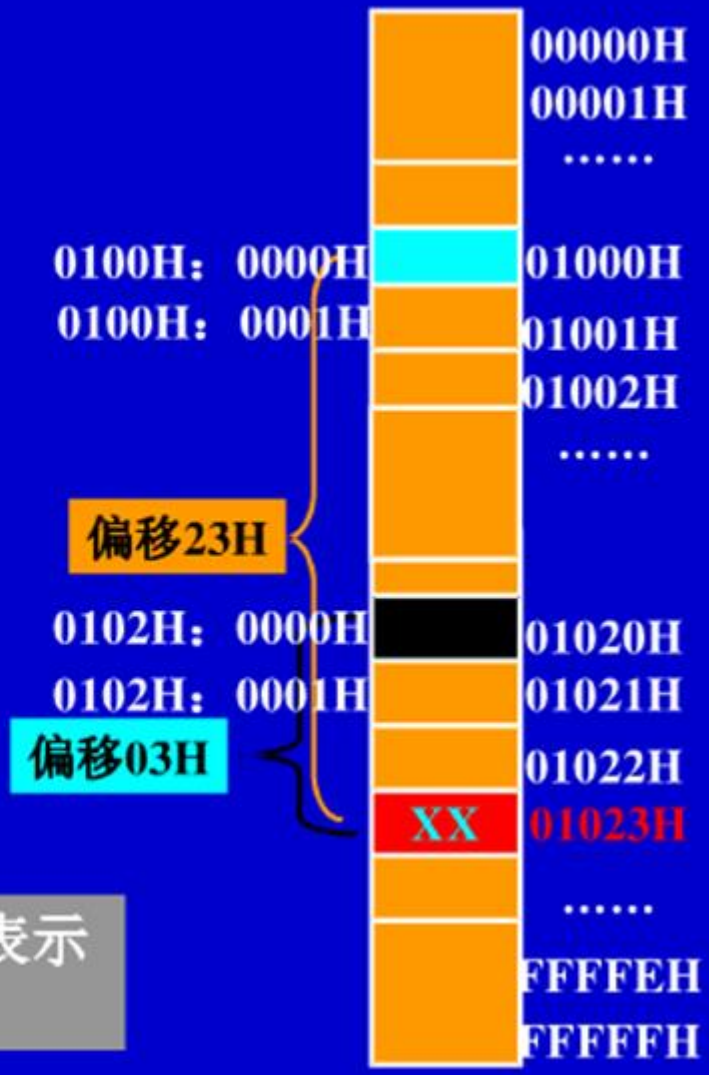
可以表为0100H：0023H

PA=0100H×10H+0023H
= 01023H

也可表为0102H：0003H

PA=0102H×10H+0003H
= 01023H

两段间距为20H单元，两种逻辑地址表示同一个物理地址单元01023H。



4.2.2 8088CPU的寻址方式

1. 操作数的寻址方式

8086/8088指令中说明操作数所在地址的寻址方式有八种（可分为三类）。

```
MOV  AX，2000H          ；立即寻址          im
MOX  AX，BX             ；寄存器寻址        r
MOX  AX，DS             ；其中，段寄存器寻址  SEG
MOV  AX，[2000H]        ；直接寻址
MOV  AX，[BX]           ；寄存器间接寻址
MOX  AX，disp[BX]       ；寄存器相对寻址
MOV  AX，[BX][SI]       ；基址变址寻址
MOV  AX，disp [BX][SI]  ；基址变址相对寻址
```

内存寻址 mem

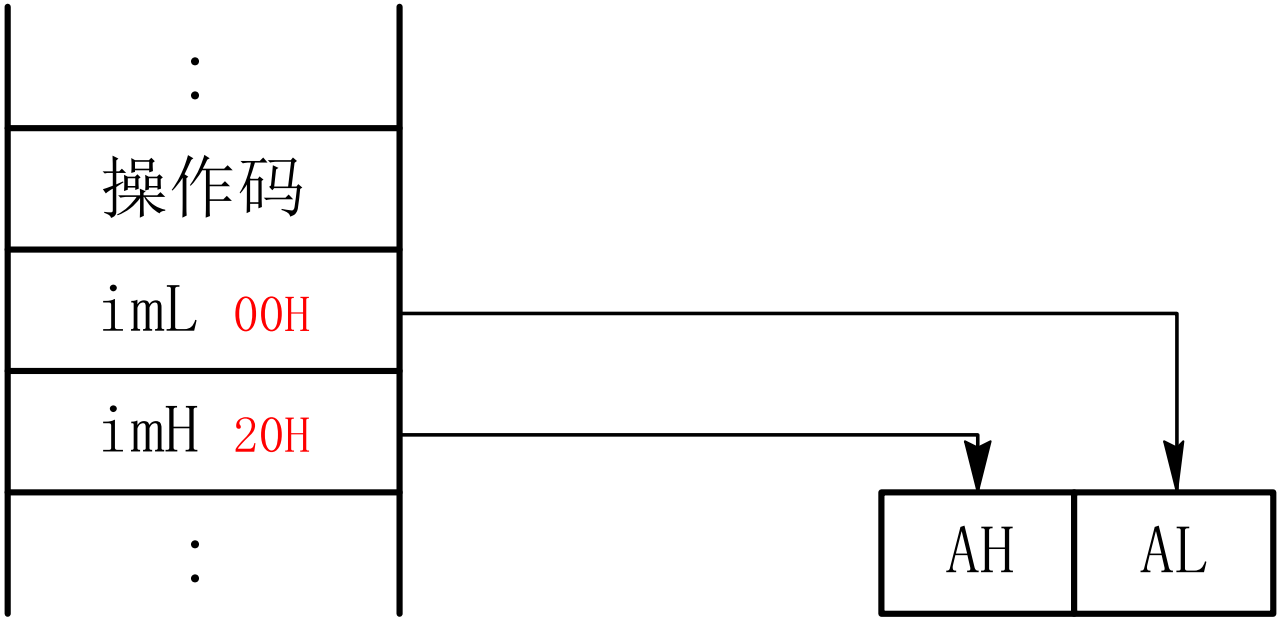
1）立即寻址

立即寻址方式所提供的**操作数直接包含在指令中**。它紧跟在操作码的后面，与操作码一起放在代码段区域中。

例： MOV CL，20H

例： MOV AX，2000H

例： MOV AX，2000H



立即寻址

（立即数是指令的一部分，即操作数）

2）寄存器寻址

寄存器寻址是指**操作数在CPU的内部寄存器中**，寄存器可以是16位寄存器，也可以是8位寄存器。例如AX、BX、CX、DX、SI、BP、AL、CH等。

例：MOV DS，AX

寄存器寻址

(指令里存放的是寄存器的编号)

3）直接寻址

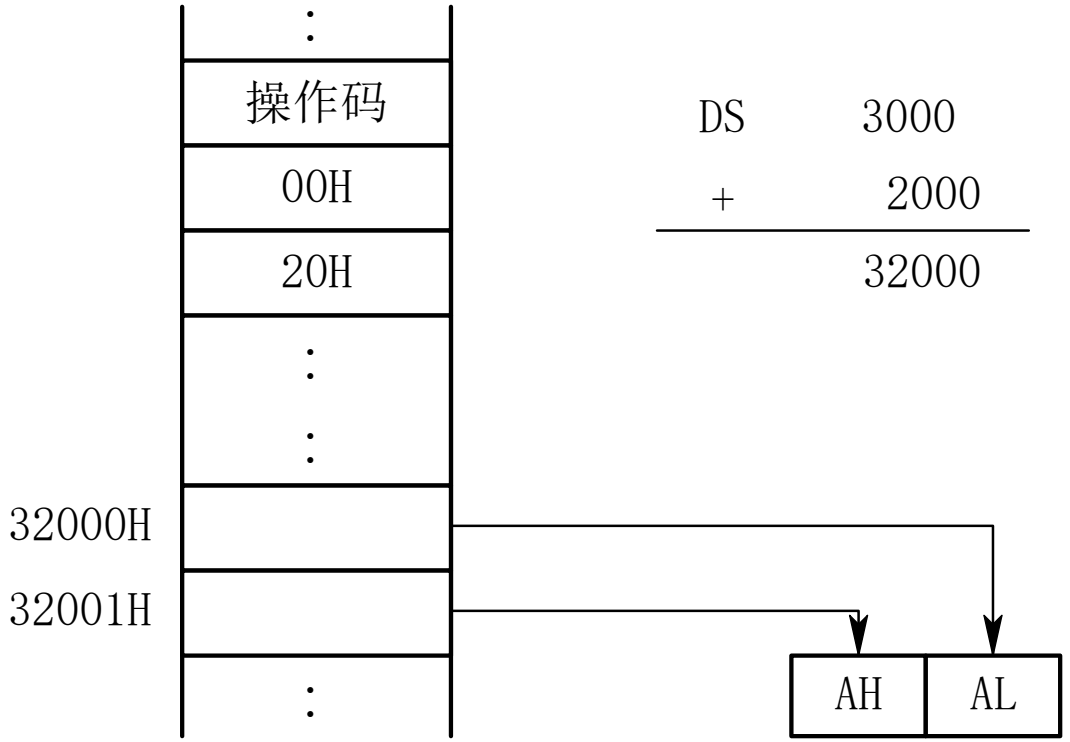
在直接寻址方式中，**操作数存放在存储器中，**而操作数地址的16位段内偏移地址直接包含在指令中，它与操作码一起存放在代码段区域。

操作数一般在数据段区域中，它的地址为数据段寄存器DS加上这16位的段内偏移地址。

例： MOV AX，[2000H]

例：MOV AX，[2000H]

指令中的16位段内偏移地址的低字节在前，高字节在后



直接寻址

(指令里存放的是16位的段内偏移地址，即EA)

4）寄存器间接寻址

在寄存器间接寻址方式中，**操作数存放在存储器中**，操作数的16位段内偏移地址放在SI、DI、BP、BX这4个寄存器中之一。

由于上述4个寄存器所默认的段寄存器不同，这样又可以分成两种情况：

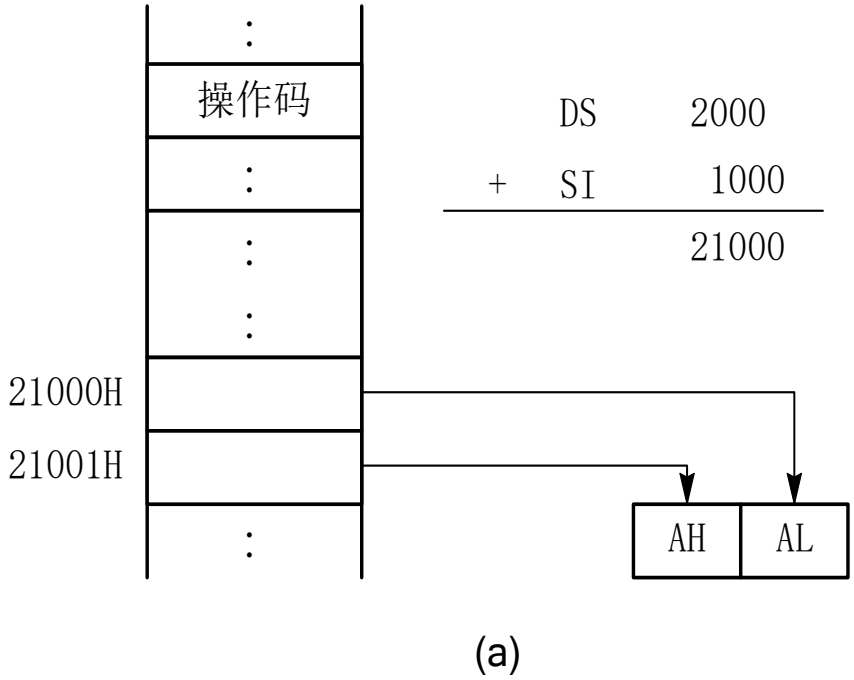
① 若以SI、DI、BX进行间接寻址，则操作数通常存放在现行数据段中。

② 若以寄存器BP间接寻址，则操作数存放在堆栈段区域中。

例：MOV AX，[SI]

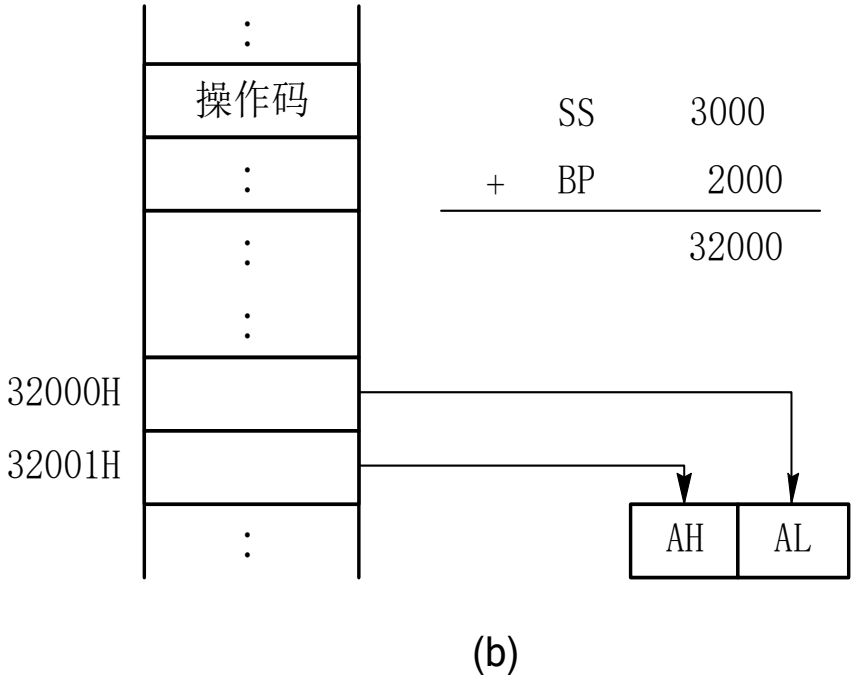
例：MOV AX，[BP]

例：MOV AX，[SI]



(a)

例：MOV AX，[BP]



(b)

寄存器间接寻址

(指令里存放的是间接寄存器的编号)

5）寄存器相对寻址

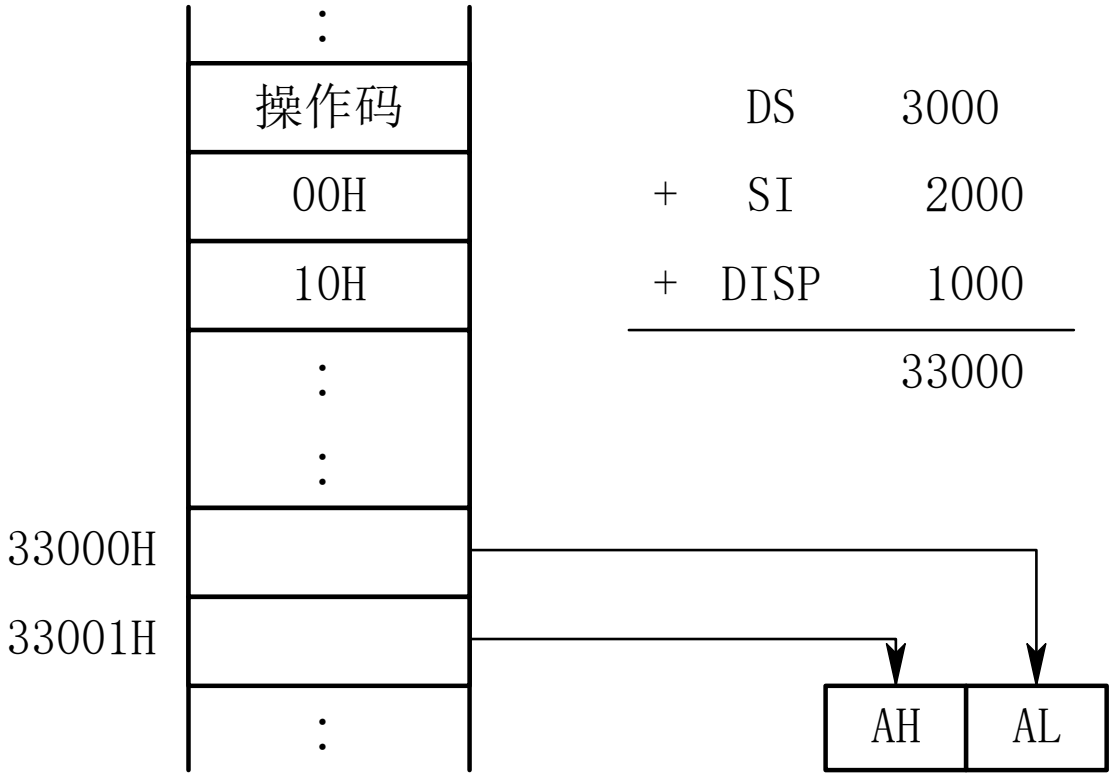
在寄存器相对寻址方式中，**操作数存放在存储器中，**操作数的16位段内偏移地址是由SI、DI、BX、BP之一的内容，再加上由指令中所指出的8位或16位相对地址偏移量而得到的。

在一般情况下，若用SI、DI或BX进行相对寻址时，以数据段寄存器DS作为地址基准，而用BP寻址时，则以堆栈段寄存器SS作为地址基准。

例：MOV AX，DISP[SI]

例：MOV AX，DISP[SI]

其过程如图所示



寄存器相对寻址

(指令里存放的是间接寄存器编号和相对地址偏移量DISP)

6）基址、变址寻址

在8086/8088中，通常把BX和BP作为基址寄存器，而把SI、DI作为变址寄存器。将这两种寄存器联合起来进行的寻址就称为**基址、变址寻址**。

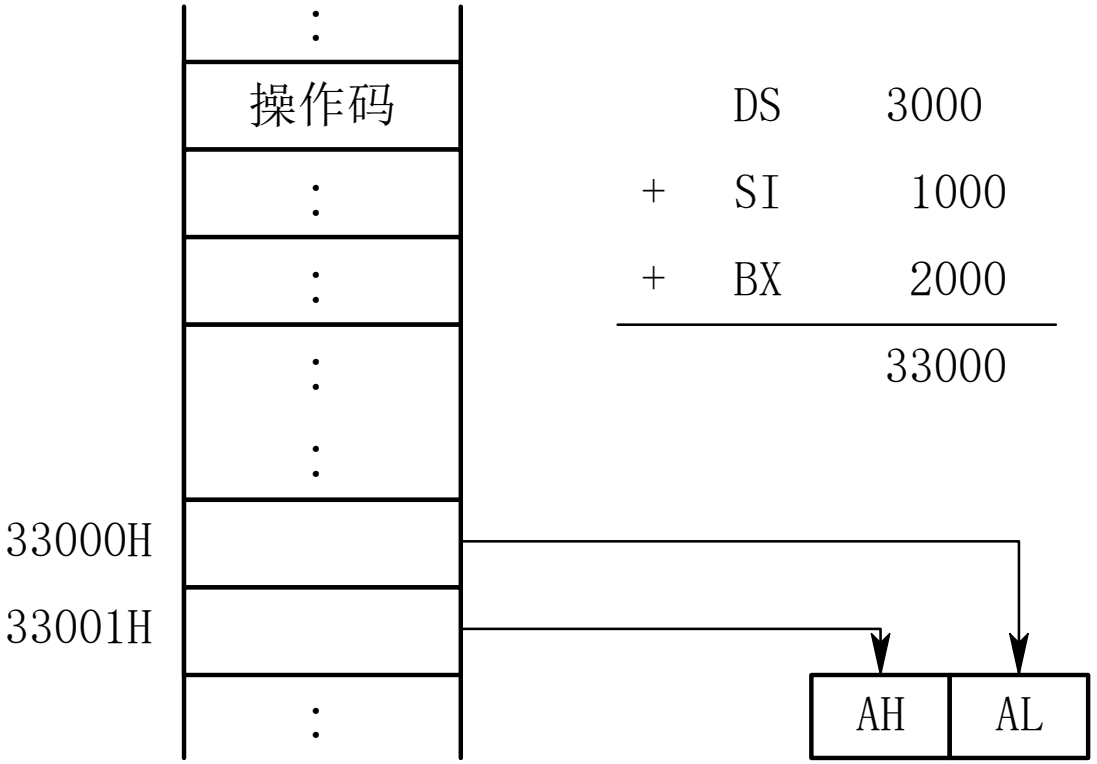
在基址、变址寻址方式中，**操作数存放在存储器中**，操作数的16位段内偏移地址是由基址寄存器内容(BX或BP内容)，再加上变址寄存器内容(SI或DI内容)而得到的。

同理，若用BX作为基地址，则操作数应放在数据段DS区域中；若用BP作为基地址，则操作数应放在堆栈段SS区域中。

例：MOV AX，[BX][SI]

例：MOV AX，[BX][SI]

其过程如图所示



基址、变址寻址

(指令里存放的是基址寄存器、变址寄存器编号)

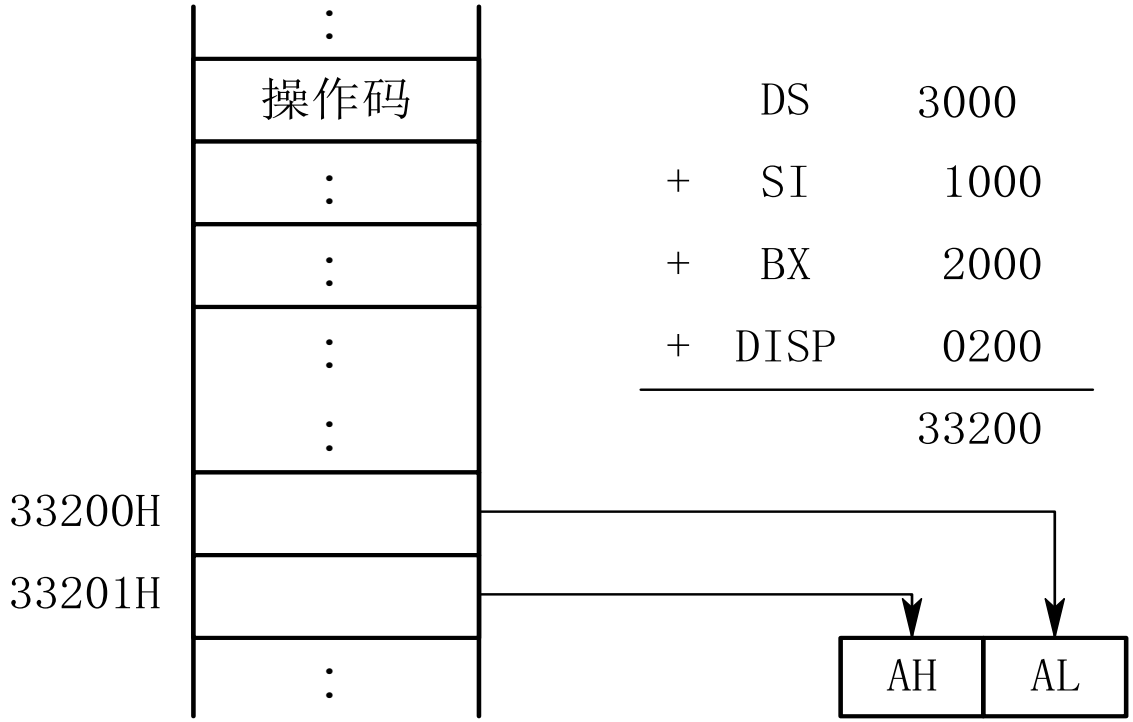
7）基址、变址、相对寻址

基址、变址、相对寻址方式实际上是基址、变址寻址方式的扩充。即**操作数存放在存储器中，**操作数的16位段内偏移地址是由基址、变址方式得到的地址再加上由指令指明的8位或16位的相对偏移地址而得到的。

例：MOV AX，DISP[BX][SI]

例：MOV AX，DISP[BX][SI]

其过程如图所示



基址、变址、相对寻址

(指令里存放的是基址寄存器、变址寄存器编号和相对地址偏移量DISP)

8）隐含寻址

在有些指令的指令码中，不仅包含有操作码信息，而且还隐含了操作数地址的信息。例如乘法指令MUL的指令码中只需指明一个乘数的地址，另一个乘数和积的地址是隐含固定的。

这种**将操作数的地址隐含在指令操作码中**的寻址方式称为隐含寻址。

例：MUL BL ; $AL \times BL \rightarrow AX$

DIV BL ; $AX \div BL \rightarrow AL$

CLC ; $0 \rightarrow CF$

操作数寻址方式总结

MOV AX，[2000H] ；直接寻址

MOV AX，[BX] ；寄存器间接寻址

MOX AX，disp[BX] ；寄存器相对寻址

MOV AX，[BX][SI] ；基址变址寻址

MOV AX，disp [BX][SI] ；基址变址相对寻址

} 内存寻址 mem

MOV AX，2000H ；立即寻址 im

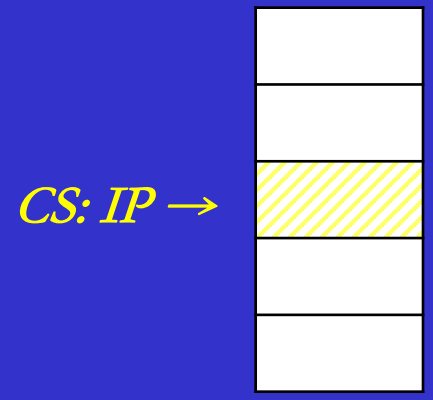
MOX AX，BX ；寄存器寻址 r

MOX AX，DS ；其中，段寄存器寻址 SEG

2. 转移地址的寻址方式

- 转移地址的寻址方式，也就是找出程序转移的地址（**下一条指令的地址**），而不是操作数。
- 8086/8088中，**CS:IP**为CPU当前将要读取的指令的地址，改变CS:IP 的内容就会程序转移。
- 转移地址可以在段内，也可以跨段转移（称段间转移）。寻求转移地址的方法有以下四种：

- ✓ 段内直接（相对）转移
- ✓ 段内间接转移
- ✓ 段间直接转移
- ✓ 段间间接转移



(1) 段内转移 —— CS不变，IP改变

✓ 段内直接（相对）转移

- 新的IP = 当前IP + 指令中的8位或16位位移量
- 当位移量是8位时称为短程转移；位移量是16位时称为近程转移。

✓ 段内间接转移

- 程序转移的地址存放在寄存器或存储器单元中。
- 指令执行使用寄存器或存储器单元的内容来更新IP的内容。

例如：JMP BX ; IP←BX

（2）段间转移 —— CS、IP均改变

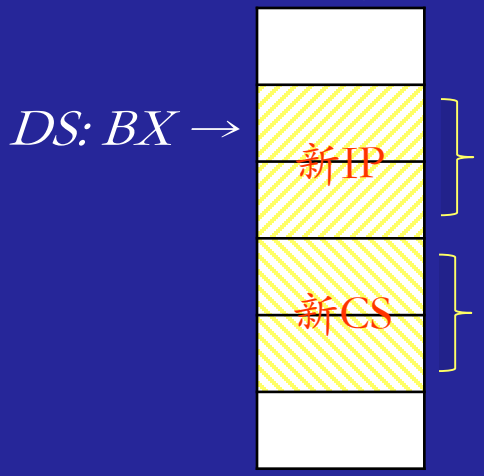
✓ 段间直接转移

- 指令码中直接给出16位的段地址和16位的偏移地址用来更新当前的CS和IP内容。

✓ 段内间接转移

- 由指令码的寻址方式字节求出存放转移地址的内存地址。其低位字中存放的是偏移地址，高位字中存放的是转移段基址。

例如：JMP DWORD PTR [BX]



注意：从应用（汇编语言）角度如下分类

（1）直接转移
- ✓ 段内直接（相对）转移
- ✓ 段间直接转移

（2）间接转移
- ✓ 段内间接转移
- ✓ 段间间接转移

应用（汇编语言）角度分类

（1）段内转移
- ✓ 段内直接（相对）转移
- ✓ 段内间接转移

（2）段间转移
- ✓ 段间直接转移
- ✓ 段间间接转移

原理（机器语言）角度分类

直接转移示例（符号地址）

```
        MOV    AX,  DATA
        MOV    DS,  AX
        MOV    SI,  OFFSET  BUF
        MOV    RES,[SI]
        INC    SI
        MOV    CX, 99
AGAIN:  MOV    AL, [SI]
        CMP    RES, AL
        JBE    NEXT
        MOV    RES, AL
NEXT:   INC    SI
        LOOP   AGAIN
        HLT
```

本章作业-1

第 2、4、10题

注：第10题针对第5章模型机，寻址方式不考虑段地址

4.3 指令系统 ---- 以 X86 汇编指令为例

- ✓ 数据传送指令
- ✓ 算术运算指令
- ✓ 逻辑运算和移位指令
- ✓ 串操作指令
- ✓ 程序控制指令
- ✓ 处理器控制指令
- ✓ 输入输出指令

- ➢ 指令的格式、功能
- ➢ 掌握不同操作数的寻址方式
- ➢ 掌握常用的指令

4.3.1 传送指令

1. MOV 指令

格式：　MOV　OPRD1，　OPRD2
　　目的　　源
功能：　（OPRD2）→（OPRD1）

其中：　**目的操作数 OPRD1**

存储器　mem
通用寄存器 r
段寄存器　SEG（除CS）

源操作数 OPRD2

立即数　im
存储器　mem
通用寄存器 r
段寄存器　SEG（含CS）

操作数可以为上述源、目的寻址方式的任意搭配，但4种情况除外。

- 例　CPU内部寄存器之间的数据传送

```
MOV AL, DH          ; AL←DH      (8位)
MOV DS, AX          ; DS←AX      (16位)
MOV AX, SI          ; AX←SI      (16位)
```

- 例　CPU内部寄存器和存储器之间的数据传送

```
MOV [BX], AX          ; 间接寻址          (16位)
MOV AX, [BX][SI]      ; 基址变址寻址      (16位)
MOV AL, BLOCK         ; BLOCK为变量名，直接寻址(8位)
```

- 例　立即数送通用寄存器、存储器

```
MOV AX, 1234H         ; AX←1234H          (16位)
MOV [BX], 12H         ; 间接寻址          (8位)
MOV AH, 12H           ; AH←12H            (8位)
```

例：

若DS=1000H,SS=2000H,BX=0100H,SI=0200H,BP=0300H,

内存（10200H）= 01H,　（10201H）=02H,
（10310H）= 03H,　（10311H）=04H,
（20500H）= 05H,　（20501H）=06H,

分析如下指令的功能

```
MOV AL, [SI]              ; 字节传送，AL=01H
  ; 源操作数为寄存器间接寻址
  ;          → DS:SI
  ;          → 逻辑地址 1000H : 0200H
  ;          → 物理地址 10200H
```

例：

若DS=1000H,SS=2000H,BX=0100H,SI=0200H,BP=0300H,

内存（10200H）= 01H,　（10201H）=02H,
（10310H）= 03H,　（10311H）=04H,
（20500H）= 05H,　（20501H）=06H,

分析如下指令的功能

```
MOV AX, [SI]              ; 字传送，AX=0201H
  ; 源操作数为寄存器间接寻址
  ;          → DS:SI
  ;          → 逻辑地址 1000H : 0200H
  ;          → 物理地址 10200H
```

例：

若DS=1000H,SS=2000H,BX=0100H,SI=0200H,BP=0300H,

内存（10200H）= 01H,　（10201H）=02H,
（10310H）= 03H,　（10311H）=04H,
（20500H）= 05H,　（20501H）=06H,

分析如下指令的功能

```
MOV AL, [SI]          ; AL=01H
MOV AX, [SI]          ; AX=0201H
MOV DX, 10H[BX][SI]   ; DX=0403H
MOV CX, [BP][SI]      ; CX=0605H
MOV [BX],CX           ; （10100H）= 05H,　（10101H）=06H,
```

➢ 操作数可以为上述源、目的寻址方式的任意搭配，但4种情况除外。

- 目的操作数不能为立即数 或CS、IP

```
MOV 1000H, AX      ;(错误)
MOV IP, 0100H      ;(错误)
MOV CS, 2000H      ;(错误)
```

- 内存单元之间不能进行数据直接传送

```
MOV [DI], [SI] ;(错误)
可改为: MOV AX, [SI]
        MOV [DI], AX
```

- 立即数不能直接传送到段寄存器中

```
MOV DS, 0100H   ;(错误)
可改为: MOV DX, 0100H
        MOV DS, DX
```

- 段寄存器之间的不能进行数据直接传送

```
MOV ES, DS      ;(错误)
可改为: MOV DX, DS
        MOV ES, DX
```

例：1000H：0100H为首地址的1000字节，传送到2000H：0000H开始的内存区。

```
        MOV   AX, 1000H
        MOV   DS, AX
        MOV   SI, 0100H      ; DS:SI = 1000H:0100H
        MOV   AX, 2000H
        MOV   ES, AX
        MOV   DI, 0000H      ; ES:DI = 2000H:0000H
        MOV   CX, 1000
NEXT:   MOV   AL, [SI]       ; (DS:SI)→AL
        MOV   ES:[DI], AL    ; AL→(ES:DI)
        INC   SI
        INC   DI
        LOOP  NEXT
        HLT
```

2. 交换指令

格式：　XCHG　OPRD1，　OPRD2
　　目的　　源
功能：　（OPRD2）←→(OPRD1)

其中：这种交换能在通用寄存器与累加器之间、通用寄存器之间、通用寄存器与存储器之间进行。

- 例：数据交换指令

```
XCHG AL, DH          ; AL ↔ DH      (8位)
   等效于：  MOV BL,AL
             MOV AL,DH
             MOV DH,BL
XCHG DX, AX          ; DX ↔ AX      (16位)
XCHG BX, CX          ; BX ↔ CX      (16位)
XCHG BX, [SI]        ; BX ↔ 内存单元DS:SI  (16位)
```

3. 地址传送指令

格式：　LEA　OPRD1，　OPRD2
　　目的　　源
功能：　OPRD2的EA → (OPRD1)

其中：OPRD1为16位通用寄存器，OPRD2为内存操作数。

➢ 例：地址传送指令

```
LEA  SI, [1000H]      ; SI ← 1000H
   ; 源操作数为直接寻址，逻辑地址 DS:1000H
   ; 其有效地址EA（即段内偏移地址）为 1000H
   ; 1000H → SI
```

➢ 例：地址传送指令

```
LEA   SI, [1000H]      ; SI ← 1000H
LEA   AX, [BX]         ; AX ← BX
LEA   DI, [BX][SI]     ; DI ← BX+SI
LEA   BX, BLOCK        ; BLOCK为变量名，BX ← BLOCK的EA
```

● 注意区分：

例：SI＝1000H，DS＝5000H，（51000H）＝1234H

执行指令　LEA　BX，[SI]后，BX＝1000H

执行指令　MOV　BX，[SI]后，BX＝1234H

4. 堆栈指令

格式： PUSH OPRD

；r、mem、SEG（含CS），以字为单位

POP OPRD

；r、mem、SEG（除CS），以字为单位

➢ 例：堆栈指令

```
PUSH   1000H        ;(错误)
PUSH   AL           ;(错误)
PUSH   AX           ;(正确)
PUSH   CS           ;(正确)
PUSH   DS           ;(正确)
PUSH   [BX]         ;(正确)
POP    CS           ;(错误)
POP    DS           ;(正确)
POP    [SI]         ;(正确)
```

4. 堆栈指令

格式： PUSH OPRD

；r、mem、SEG（含CS），以字为单位

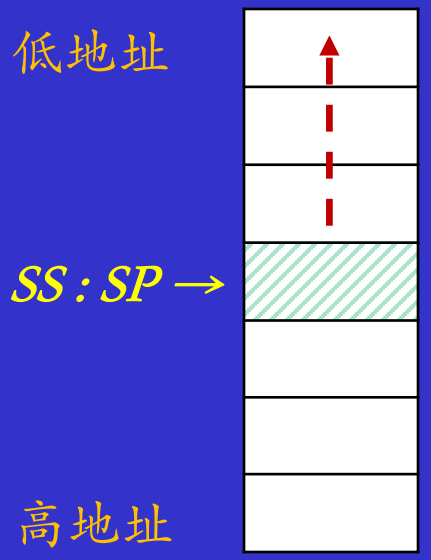
POP OPRD

；r、mem、SEG（除CS），以字为单位

- *8088系统堆栈是在存储器中开辟的一个特定区域，**SS:SP** 始终指向栈顶，反向生长。*
- *开辟堆栈的目的主要有以下两点：*

  (1) 存放指令操作数(变量)。

  (2) 保护断点和现场。



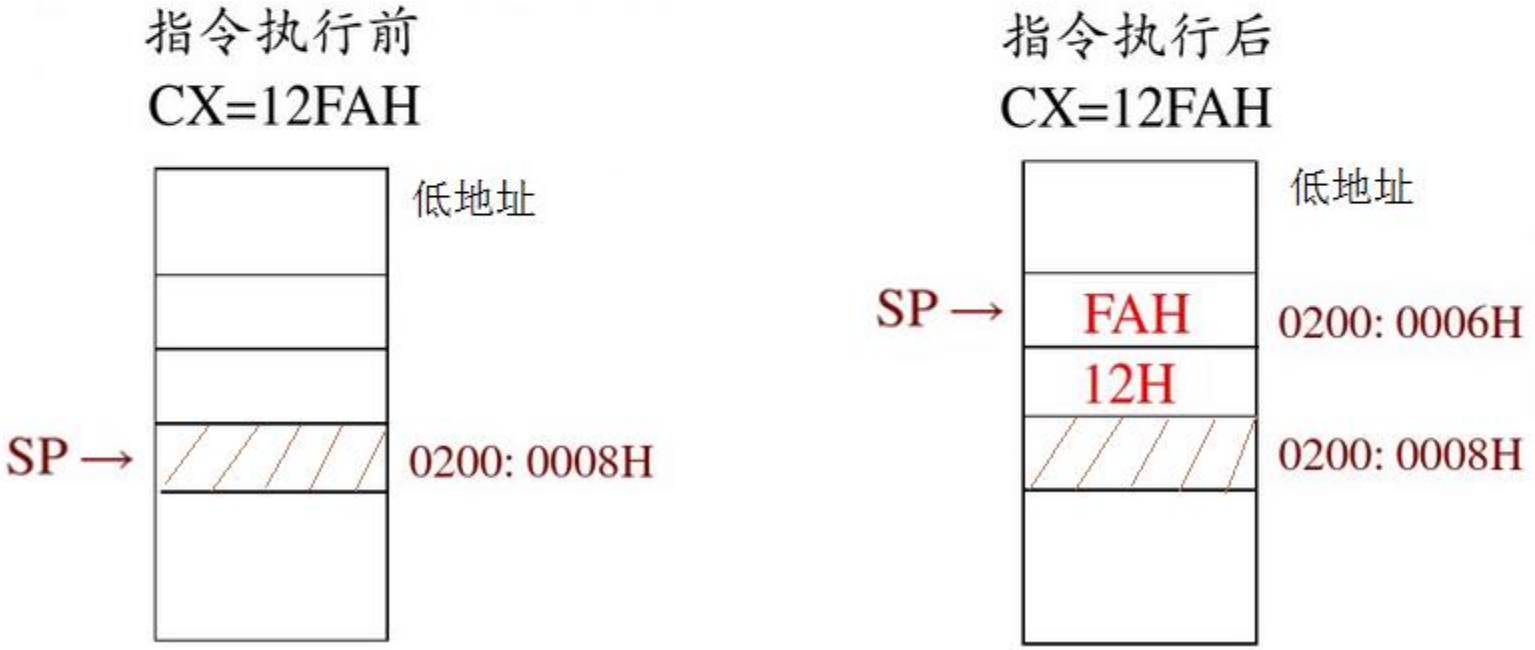
堆栈指令功能演示：

① 进栈指令　　PUSH　OPRD

功能：将一个字的源操作数传送至由SP所指向的堆栈的顶部.

操作：PUSH 操作时，先修改SP 的值，使 SP－2 ⇨SP 后，把源操作数（字）压入堆栈中 SP 指示的位置上。

例：设SS=0200H, SP=0008H, CX=12FAH, PUSH　CX 指令执行过程：



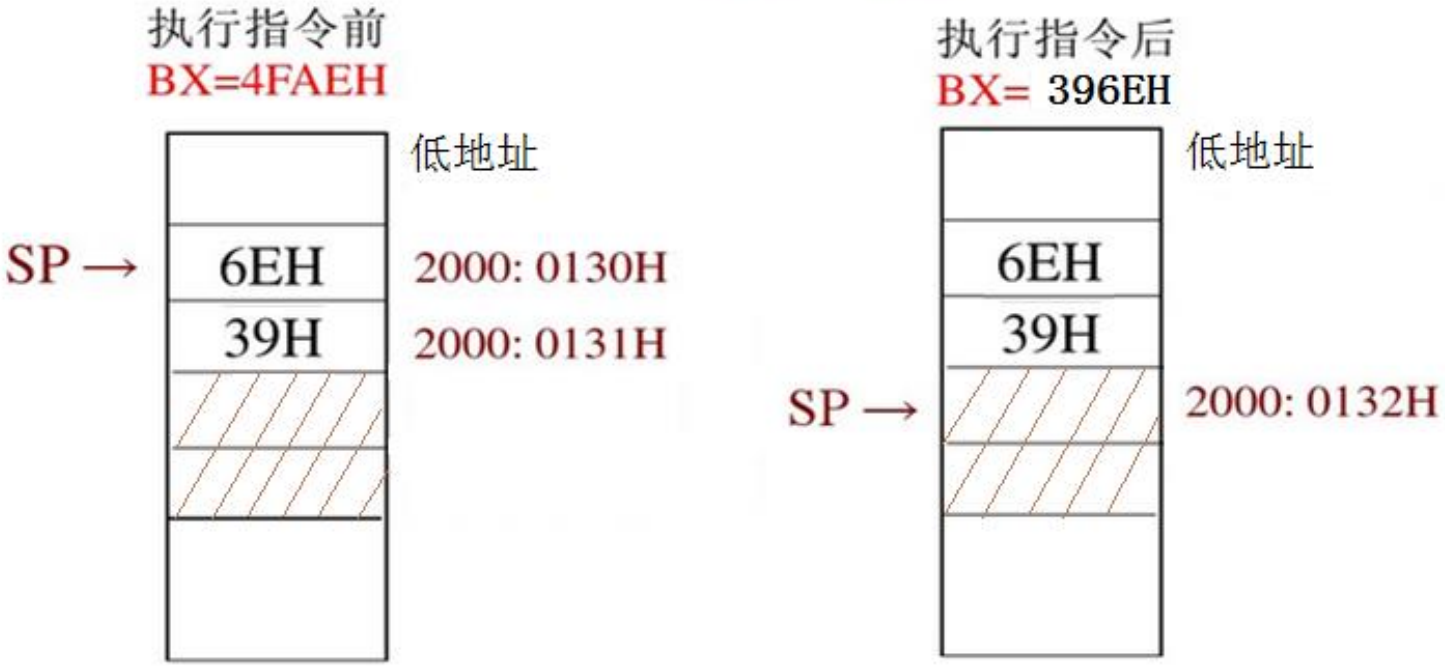
堆栈指令功能演示：

出栈指令　POP　OPRD

功能：把当前 SP 所指向的堆栈顶部的一个字送到指定的目的操作数中。（CS不能作为目的操作数）

操作：每执行一次出栈操作，SP +2 → SP，指向新的栈顶。

例：设SS=2000H，SP = 0130H，（20130H）= 396EH，BX=4FAEH，执行指令POP　BX的过程如下图所示：



- ✓ 数据传送指令
- ✓ 算术运算指令
- ✓ 逻辑运算和移位指令
- ✓ 串操作指令
- ✓ 程序控制指令
- ✓ 处理器控制指令
- ✓ 输入输出指令

其它指令简介

4.4 汇编语言及其程序设计

伪指令与汇编语言

4.5 精简指令系统计算机（RISC）

- ✓ CISC 复杂指令系统计算机
  （Complex Instruction Set Computer）

- ✓ RISC 精简指令系统计算机
  （Reduced Instruction Set Computer）

4.5.1 CISC

- RISC与CISC是目前指令系统的两种不同设计思路。
  - Neumann语义差距问题

    语义(semantics)差距:即高级语言与机器语言的间隙.

    高级语言发展→靠拢人类自然语言→语义差距↑

    要使语义差距↓ { 1.增加编译程序功能 / 2.扩大CPU指令功能 } →硬件复杂 { 芯片工作不稳定 / 投资大 }

  - 系列计算机的兼容性

CISC设计风格的主要特点

- 指令系统复杂：指令数多、寻址方式多、指令格式多。
- 绝大多数指令执行需要多个时钟周期。
- 多种指令可访问存储器。
- 采用微程序控制。
- 有专用寄存器。
- 难以优化编译生成高效的目标代码程序。

4.5.2 RISC

- RISC的由来

  - Neumann语义差距问题 →硬件复杂 { 芯片工作不稳定 / 投资大 }
  - 80-20规律:

    即典型程序中的80%语句仅使用CPU中20%的指令(大多是+、取数、转移等简单指令)。80X86中最常用的10种指令(+,-,×,call,条件转,存取,寄存器间传送,比较等)占96%。

PC上的统计实例

- 启示:能否仅使用最常用的20%的简单指令,重新组合不常用的80%的指令功能? →引发RISC技术

- 思想：留下最常用的20%的简单指令,通过优化硬件设计,提高时钟频率,实现高性能。

  靠速度取胜！但并不是简单的精简指令集。

- 若一程序总执行时间为:P=N × C × T,其中

  N--待执行的指令总条数, C—CPU平均周期数/指令

  T--每个CPU周期的时间

$$P\downarrow = N \times C \times T \begin{cases} \text{CISC: } N\downarrow \rightarrow C\uparrow \ T\uparrow & 4\sim10 \\ \text{RISC: } C\downarrow \ T\downarrow \rightarrow N\uparrow & 1.3\sim1.7 \end{cases}$$

- 结论：RISC性能=2 ~ 5倍的CISC

RISC的特点

- 只设置使用频度高的指令
- 指令长度固定，指令种类少，寻址方式种类少
- 访存指令很少，仅通过LOAD和STORE指令安排访问存储器，大多数操作均在寄存器之间进行
- CPU中设置大量的通用寄存器
- 控制器以硬布线逻辑方式直接实现
- 采用流水技术，多数指令在单机器周期内完成
- 有利于优化编译程序
- 可简化硬件设计，降低设计成本

CISC与RISC正逐渐融合……

本章作业-2

第12、18、21题