第 3 章 指令系统与寻址方式

内容提要

3.1 80x86指令基本概念

3.2 80x86寻址方式

3.3 80x86指令格式

3.4 80x86指令系统

要求：

熟悉各寄存器及用法

各种寻址方式

掌握8086常用指令的功能及应用

3.1 基本概念

指令集架构ISA

ISA：Instruction Set Architecture

- 寄存器组织
  - 寄存器个数、寄存器尺寸
- 存储器组织
  - 地址空间
  - 寻址粒度
- 指令集
  - 操作码表
  - 寻址方式
  - 指令格式

复杂指令集计算机CISC体系

CISC – Complex Instruction Set Computer

- 实例
  - Intel x86, IBM Z系列主机，老式CPU体系结构
- 特点
  - 通用寄存器的数量少
  - 寻址方式多
  - 专用、复杂指令种类多
  - 指令变长

CISC体系结构的局限

- 程序员和编译器使用复杂指令的频度低
- 访问存储器是一个慢速操作，在一条指令的执行时间中占有较大的比例
- 过程/函数调用是主要瓶颈
  - 传递参数耗时

RISC的提出与发展

- Load/Store结构提出： CDC6600(1963)--CRAY1(1976)
- RISC思想最早在IBM公司提出，但那时不叫RISC，IBM801处理器是公认体现RISC思想的机器。
- 1980年，Berkeley的Patterson和Dizel提出RISC名词，并研制了RISC-I、Π实验样机。
- 1981年Stanford的Hennessy研制MIPS芯片。
- 随后都分别推出了商品化RISC： SPARC V1(1987)和MIPS1(1986)

精简指令集计算机RISC体系

RISC – Reduced Instruction Set Computer

- 实例
  - IBM Power PC, Sun Sparc, MIPS
- 指令集被简化
- 固定长度、固定格式指令字
  - 流水线操作，取指与指令执行的并行操作
- 寻址方式少
  - 降低了硬件的复杂度
- 面向寄存器的指令集
  - 降低对存储器的访问次数
- 可用寄存器的数量多
  - 降低对存储器的访问次数
  - 高效的过程调用

CISC和RISC对比

- CISC
  - 比RISC执行的指令条数少（即：代码密度高）
  - 取指操作少，降低功耗
- RISC
  - 比CISC执行的指令条数多
  - 取指操作多，功耗大
  - 数据、控制通路更简单，每条指令能耗低
  - 易于实现流水操作

现代CISC和RISC体系结构已开始相互融合

指令的构成

- 计算机指令包括：
  - 操作码字段：说明计算机要执行哪种操作
  - 操作数字段：指令操作所涉及的对象

0,1,2,3个操作数

对于CISC处理机，最常见的是单操作数和双操作数指令

3.2 寻址方式

寻址方式

寻址方式：确定操作数对象的方式

1. 与数据有关的寻址方式
2. 与转移有关的寻址方式

1. 与数据有关的寻址方式

（1）立即寻址
（2）寄存器寻址
（3）直接寻址
（4）寄存器间接寻址
（5）寄存器相对寻址
（6）基址变址寻址
（7）相对基址变址寻址
（8）比例变址寻址
（9）基址比例变址寻址
（10）相对基址比例变址寻址

要求掌握（1）～（7）

存储器寻址

386+寻址方式

（1）立即寻址

操作数包含在指令中

目的 ← 源

MOV AX，55AAH

寄存器不能使用段寄存器

（2）寄存器寻址

操作数包含在CPU内部的寄存器中

MOV AX，BX

MOV DS，AX

源和目的不能同时为段寄存器，目的不能为IP或CS寄存器

存储器寻址

由于操作数存放在存储器中，寻址时需要考虑以下三个因素：

- 有效地址
- 默认段选择规则
- 段超越

i 有效地址

操作数地址（物理地址）＝段基址＋偏移地址

偏移地址也称为有效地址EA（Effective Address）

有效地址EA为下式1项或多项的组合

EA＝基址＋（变址×比例因子）＋位移量

ii 默认段选择规则

访存类型	段寄存器	默认选择规则
指　　令	CS	用于取指令
堆　　栈	SS	堆栈操作；BP用作寄存器间接寻址的访存
局部数据	DS	除堆栈和目的串操作之外的其他数据的访问
目 的 串	ES	串处理指令的目的串

默认段寄存器选择取决于指令、寻址方式和涉及的寄存器

iii 段超越前缀

- 80x86为程序员提供了编程的灵活性，允许在程序中自行选择段寄存器，段超越的格式为

〈段寄存器〉：指令操作数

段超越前缀

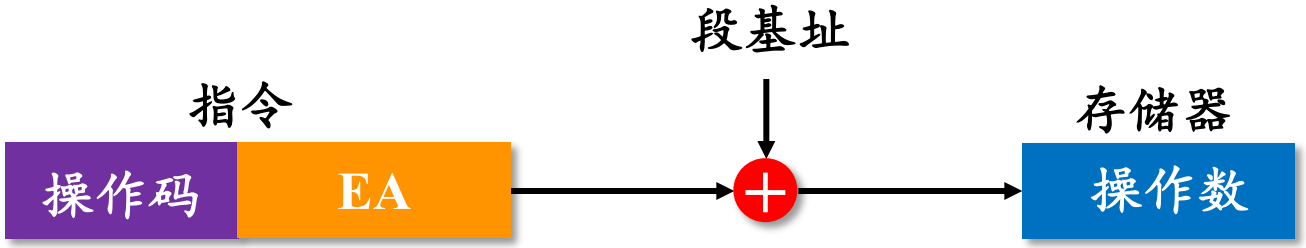
例： mov ax, ds:[bp]; BP指向的段是数据段

（3）直接寻址

操作数的有效地址EA（只有位移量）包含在指令中。

MOV AX，[2000H]

ADD [3000H]，CX

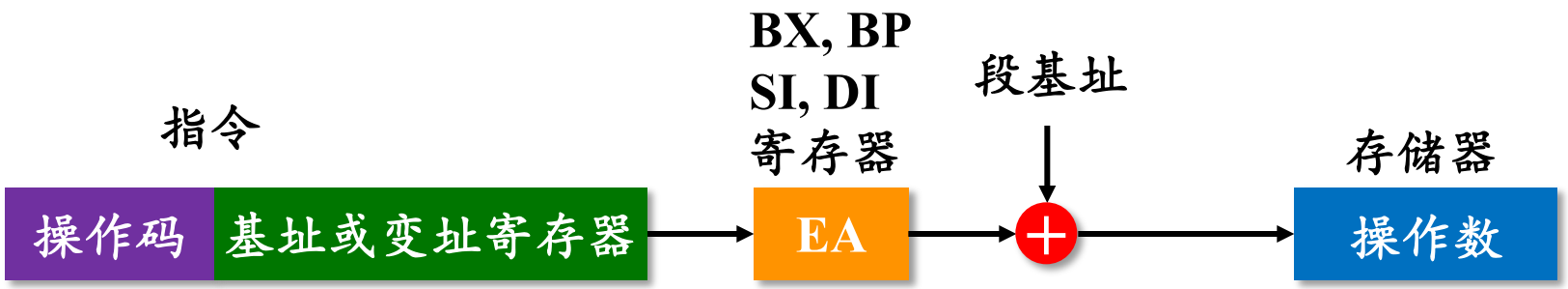


（4）寄存器间接寻址

操作数的有效地址EA包含在BX，BP，SI和DI中。

MOV CX，[BX]

ADD AX，[SI]



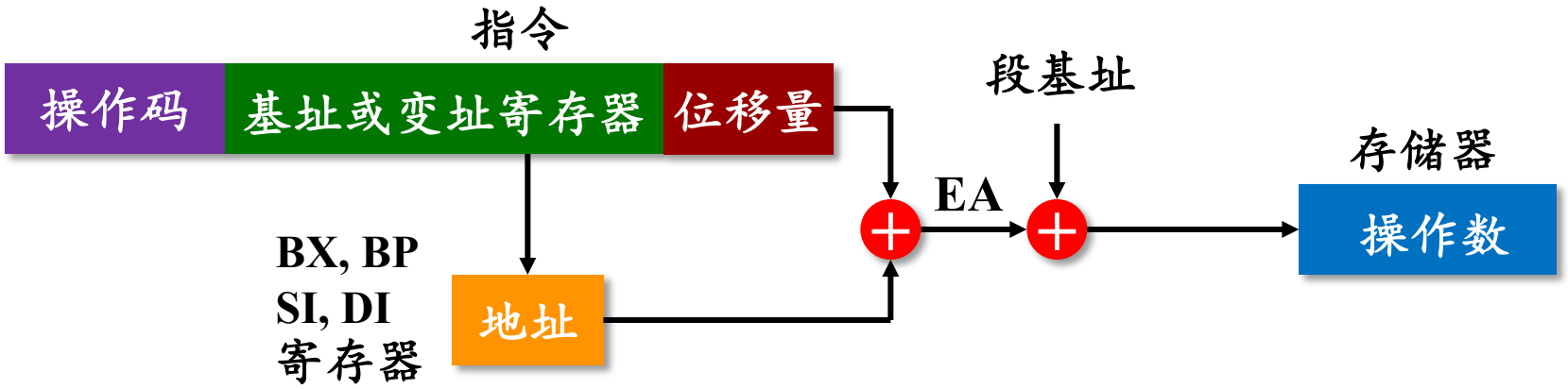
基址寄存器为BP时，默认段寄存器为SS

（5）寄存器相对寻址

操作数的有效地址EA为基址寄存器或变址寄存器的内容和指令中的位移量之和。

MOV AX，TABLE[BX]

也写成：MOV AX，[BX+TABLE]



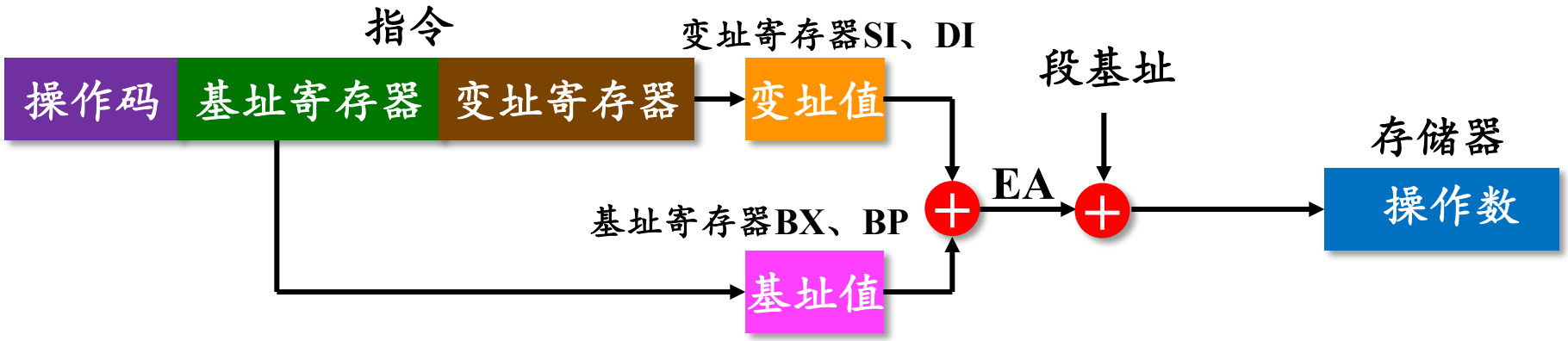
基址寄存器为BP时，默认段寄存器为SS

（6）基址变址寻址

操作数的有效地址EA为基址寄存器和变址寄存器的内容之和。

MOV AX，[BX][SI]

也写成：MOV AX，[BX+SI]



基址寄存器为BP时，默认段寄存器为SS

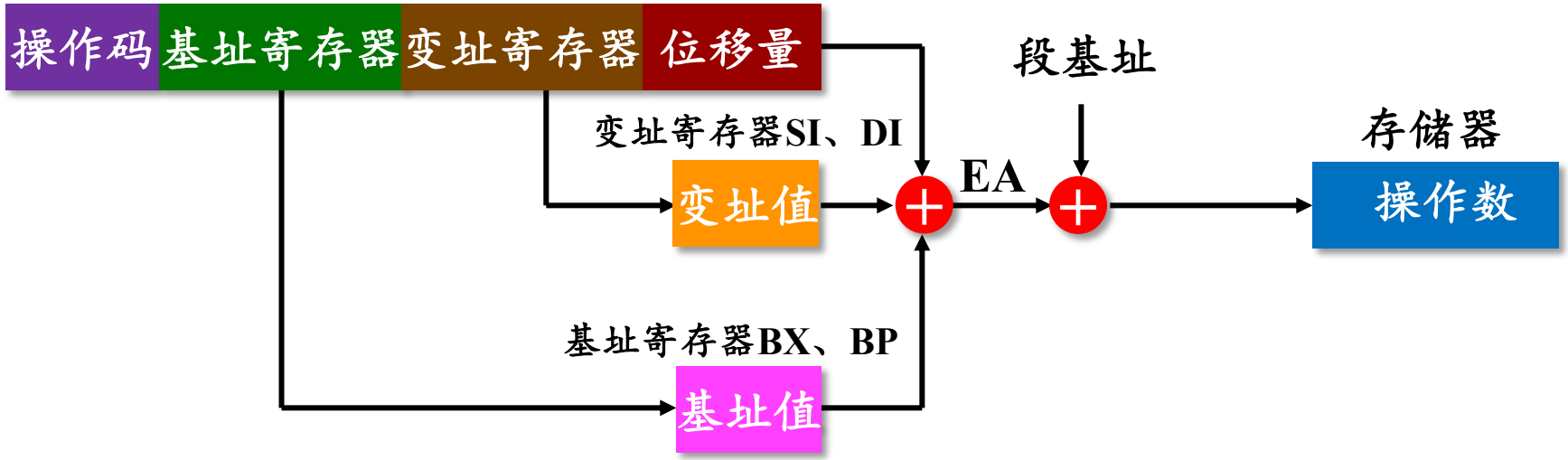
（7）相对基址变址寻址

操作数的有效地址EA为基址寄存器的内容、变址寄存器的内容和指令中的位移量之和。

MOV AX，ARRAY[BX][SI]

也写成：**MOV AX，[BX+SI+ARRAY]**

常用于对二维数组的寻址

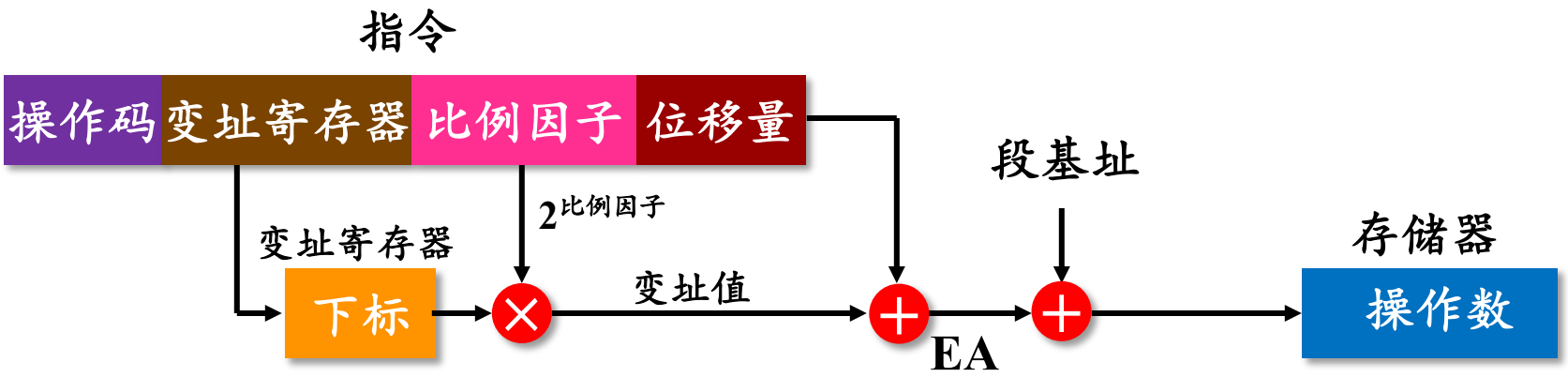


基址寄存器为BP时，默认段寄存器为SS

（8）比例变址寻址

操作数的有效地址EA是变址寄存器的内容乘以指令中2的比例因子次方再加上位移量之和。

MOV EAX，TABLE[ESI*4]

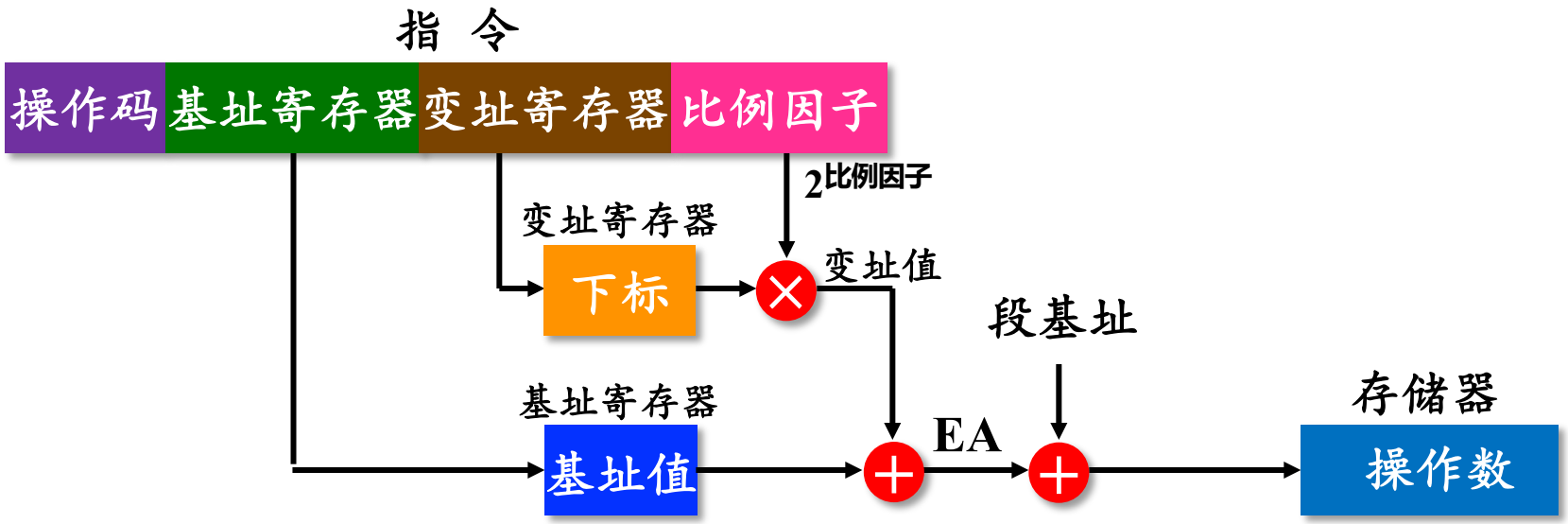


只用在80386+中

（9）基址比例变址寻址

操作数的有效地址EA是变址寄存器的内容乘以指令中2的比例因子次方再加上基址寄存器的内容之和。

MOV ECX，[EAX][EDX*8]

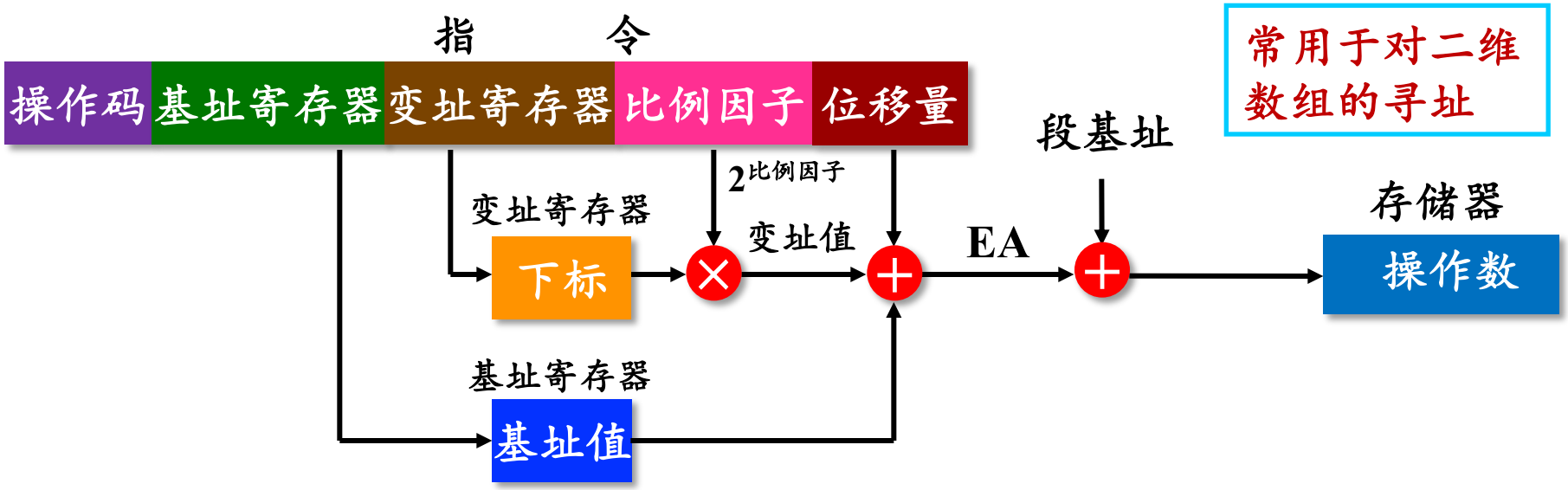


只用在80386+中

（10）相对基址比例变址寻址

操作数的有效地址EA是变址寄存器的内容乘以指令中2的比例因子次方，加上基址寄存器的内容，再加上位移量之和。

MOV EAX，TABLE[EBP][EDI*8]



只用在80386+中

2. 与转移有关的寻址方式

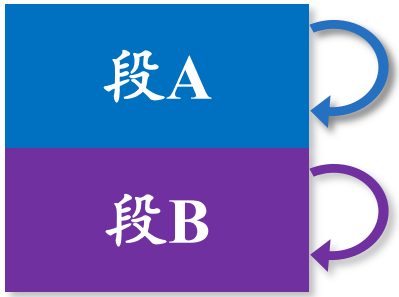
- CPU在执行程序时，利用内部寄存器记录下一条指令的地址。80x86采用代码段寄存器CS和指令指针寄存器IP保存即将执行的指令地址。其中CS保存段地址，IP保存段内偏移地址。

  下一条指令的物理地址 = (CS)*16 + (IP)

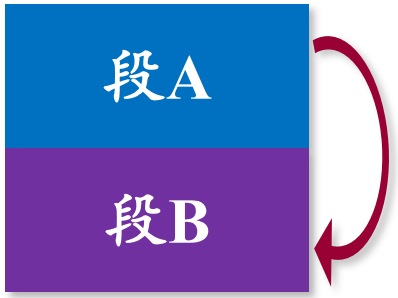
- 顺序执行时，IP的内容随指令的执行自动取得增量(当前机器指令占据的字节数)，而CS的内容不变
- 执行转移指令时，指令地址将发生跳变，即修改IP内容或修改IP和CS的内容

转移的分类

- **段内转移：** 转移的目标地址在当前代码段内。CS的内容保持不变，需要将IP的内容修改为目标地址



- **段间转移：** 转移的目标地址不在本段。CS和IP的内容都需要修改



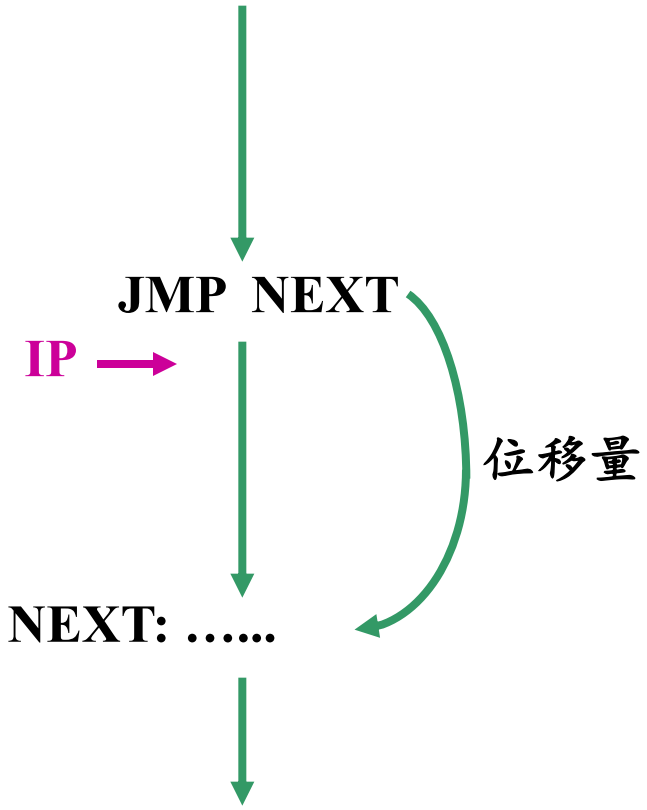
（1）段内直接寻址

EA = （IP）+ 8（16）位位移量

包含在转移指令中

转移：EA ⟶（IP）

JMP NEXT



位移量为目标地址与IP寄存器的内容(下一条指令地址)之差
8位位移量称为短转移（转移范围为-128～127字节）
16位位移量称为近转移（转移范围为-32768～32767字节）

（2）段内间接寻址

转移的有效地址EA放在寄存器或存储单元中。这个寄存器或存储单元的内容可以用除立即寻址以外的其他任意一种寻址方式获得。

转移：EA ⟶（IP）

JMP BX ;（BX）⟶（IP）

JMP WORD PTR[BP+TABLE]

（（SS）× 16 +（BP）+ 位移量）⟶（IP）

JMP WORD PTR TABLE[BX]

（（DS）×16 +（BX）+ 位移量）⟶（IP）

（3）段间直接寻址

指令中包含转移地址的段地址和偏移地址，执行时直接用段地址替代CS、偏移地址替代IP

JMP FAR PTR LABEL

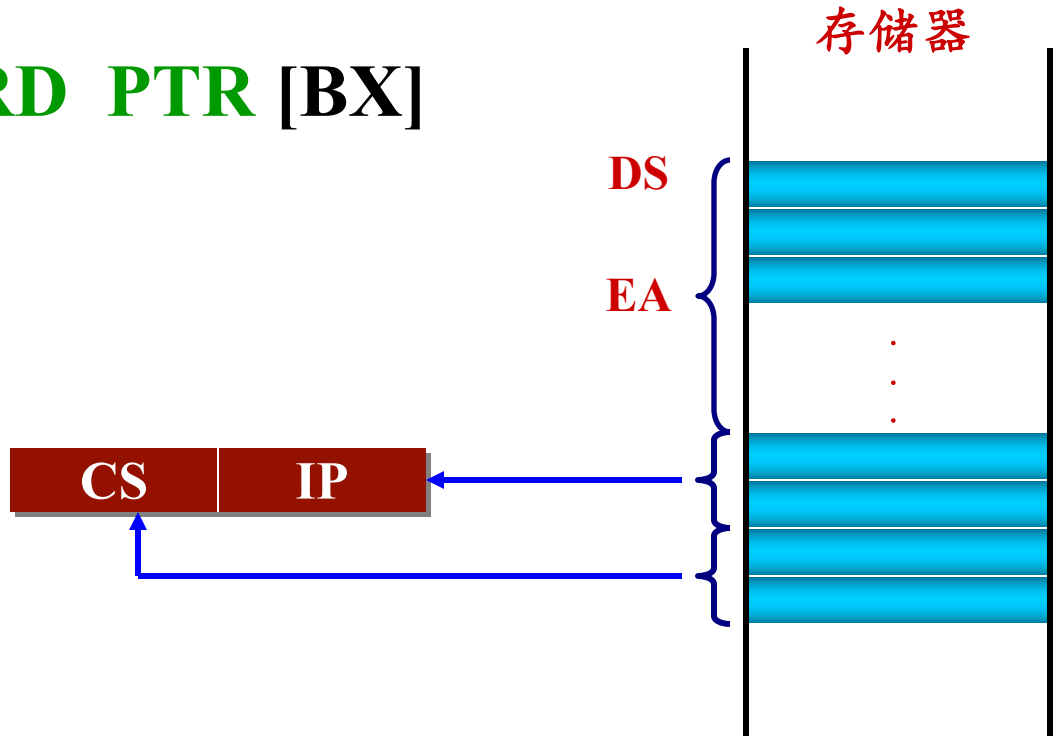
LABEL的段地址 ⟶（CS）

LABEL的偏移地址⟶（IP）

（4）段间间接寻址

- IP、CS存放在存储器中，该存储单元使用除立即寻址和寄存器寻址方式以外的任意一种寻址方式来指定

JMP DWORD PTR [BX]



寻址方式小结（1）

- 立即寻址
- 寄存器寻址
- 存储器寻址
  - 直接寻址
  - 间接寻址
    - 寄存器间接寻址
    - 寄存器相对寻址
    - 基址变址寻址
    - 相对基址变址寻址
    - 带比例因子的变址寻址

寻址方式小结（2）

- 段内寻址
  - 段内直接寻址
  - 段内间接寻址
- 段间寻址
  - 段间直接寻址
  - 段间间接寻址

动手做

1、可用作寄存器间接寻址或基址变址寻址的寄存器，正确的是（ ）。

A. AX，BX，CX，DX　　B. DS，ES，CS，SS

C. SP，BP，IP， BX　　D. SI，Dl，BP，BX

2、直接寻址方式是指在指令中给出操作数在内存中的地址，该地址是（ ）。

A. 逻辑地址　　B. 段地址

C. 偏移地址　　D. 物理地址

3、 MOV [BX+DI+16]，DX指令的源、目操作数寻址方式分别是（ ）。

A. 直接寻址、相对基址变址寻址　　B.相对基址变址寻址、寄存器寻址

C. 相对基址变址寻址、直接寻址　　D. 寄存器寻址、相对基址变址寻址


BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
北京邮电大学

3.3 指令格式

x86指令格式

8086； 80386+

指令前缀（1）

指令前缀：是一个特殊字节，用来改变已有指令的行为

- 封锁与重复前缀
  - F0H(lock)、F2H(repnz)、F3H(rep)
- 段跨越前缀
  - 2EH：CS；　　36H：SS；　　64H：FS
  - 3EH：DS；　　26H：ES；　　65H：GS
- 地址宽度前缀（Address-Size Override Prefix）
  - 67H
  - 用来确定使用16还是32位位移量/地址偏移

指令前缀（2）

- 操作数宽度前缀（Operand-Size Override Prefix）
  - 66H
  - 用来确定指令存取的操作数是字还是双字

地址宽度前缀、操作数宽度前缀和段描述符中D bit的组合决定了操作数和地址的宽度

Segment Default D = ...	0	0	0	0	1	1	1	1
Operand-Size Prefix 66H	N	N	Y	Y	N	N	Y	Y
Address-Size Prefix 67H	N	Y	N	Y	N	Y	N	Y
Effective Operand Size	16	16	32	32	32	32	16	16
Effective Address Size	16	32	16	32	32	16	32	16

操作码

- x86指令是多字节指令（最多为16字节），指令操作码（OPCODE）字节确定该指令所执行的操作
- x86指令种类多，相应的指令格式也很多。下面以一种指令格式为例：

W：=0/1 字节操作/字或双字操作。由操作数宽度前缀等来指明是字操作还是双字操作。
D：对双操作数指令有效
0：指定寄存器（reg字段）用于源操作数
1：指定寄存器（reg字段）用于目的操作数

寻址方式

reg：指定寄存器。双操作数指令中有一个操作数必须放在寄存器中

mod与r/m：确定另一操作数。用于指定寄存器或确定存储器的寻址方式

SIB：base reg + 2^{scale} × index reg

mod、reg和r/m字段

reg	w=0	w=1		r/m	mod=0		mod=1		mod=2		mod=3
		16b	32b		16b	32b	16b	32b	16b	32b	
0	AL	AX	EAX	0	addr=BX+SI	=EAX	*same*	*same*	*same*	*same*	*same*
1	CL	CX	ECX	1	addr=BX+DI	=ECX	*addr*	*addr*	*addr*	*addr*	*as*
2	DL	DX	EDX	2	addr=BP+SI	=EDX	*mod=0*	*mod=0*	*mod=0*	*mod=0*	*reg*
3	BL	BX	EBX	3	addr=BP +DI	EBX	*+d8*	*+d8*	*+d16*	*+d32*	*field*
4	AH	SP	ESP	4	addr=SI	=(sib)	SI+d8	(sib)+d8	SI+d8	(sib)+d32	"
5	CH	BP	EBP	5	addr=DI	=d32	DI+d8	EBP+d8	DI+d16	EBP+d32	"
6	DH	SI	ESI	6	addr=d16	=ESI	BP+d8	ESI+d8	BP+d16	ESI+d32	"
7	BH	DI	EDI	7	addr=BX	=EDI	BX+d8	EDI+d8	BX+d16	EDI+d32	"

r/m field depends on mod and machine mode

w from opcode

First address specifier: Reg=3 bits, R/M=3 bits, Mod=2 bits

RHK.S96 27

sc、index和base字段

	Index	Base
0	EAX	EAX
1	ECX	ECX
2	EDX	EDX
3	EBX	EBX
4	no index	ESP
5	EBP	if mod=0, d32 if mod≠0, EBP
6	ESI	ESI
7	EDI	EDI

Base + Scaled Index Mode

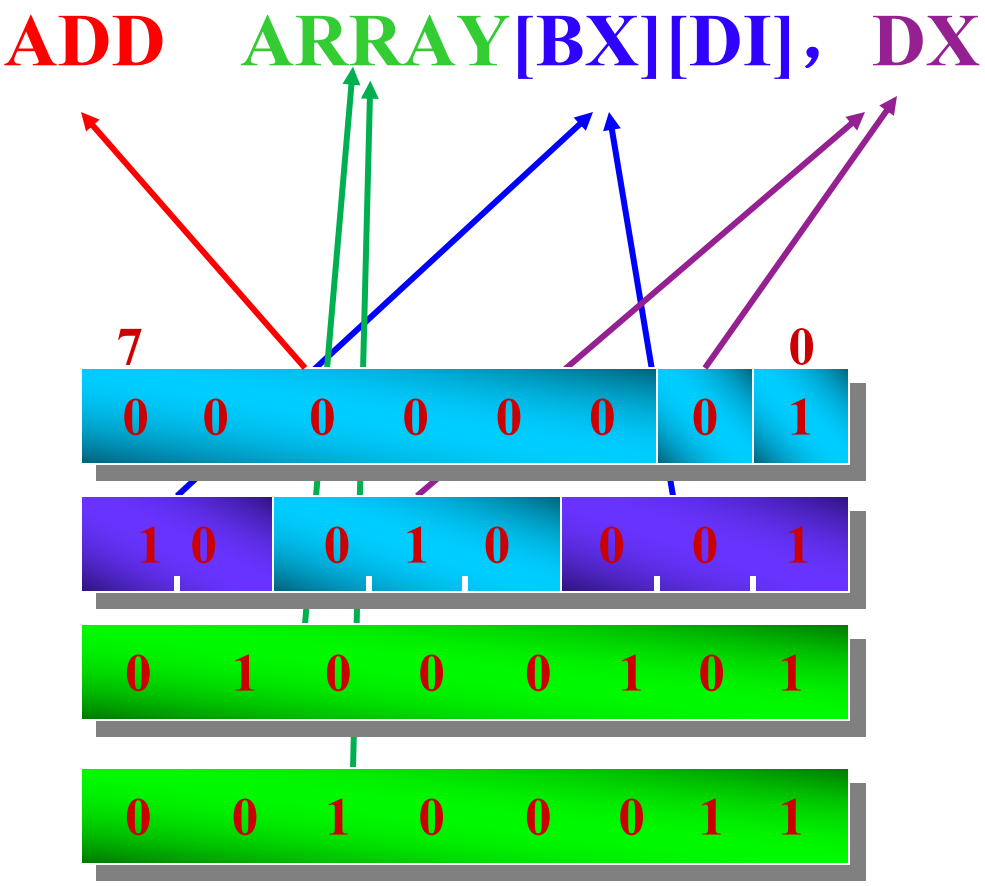
Used when:
- mod = 0,1,2
- in 32-bit mode
- AND r/m = 4!

2-bit Scale Field
3-bit Index Field
3-bit Base Field

$$\text{base reg} + 2^{\text{scale}} \times \text{index reg}$$

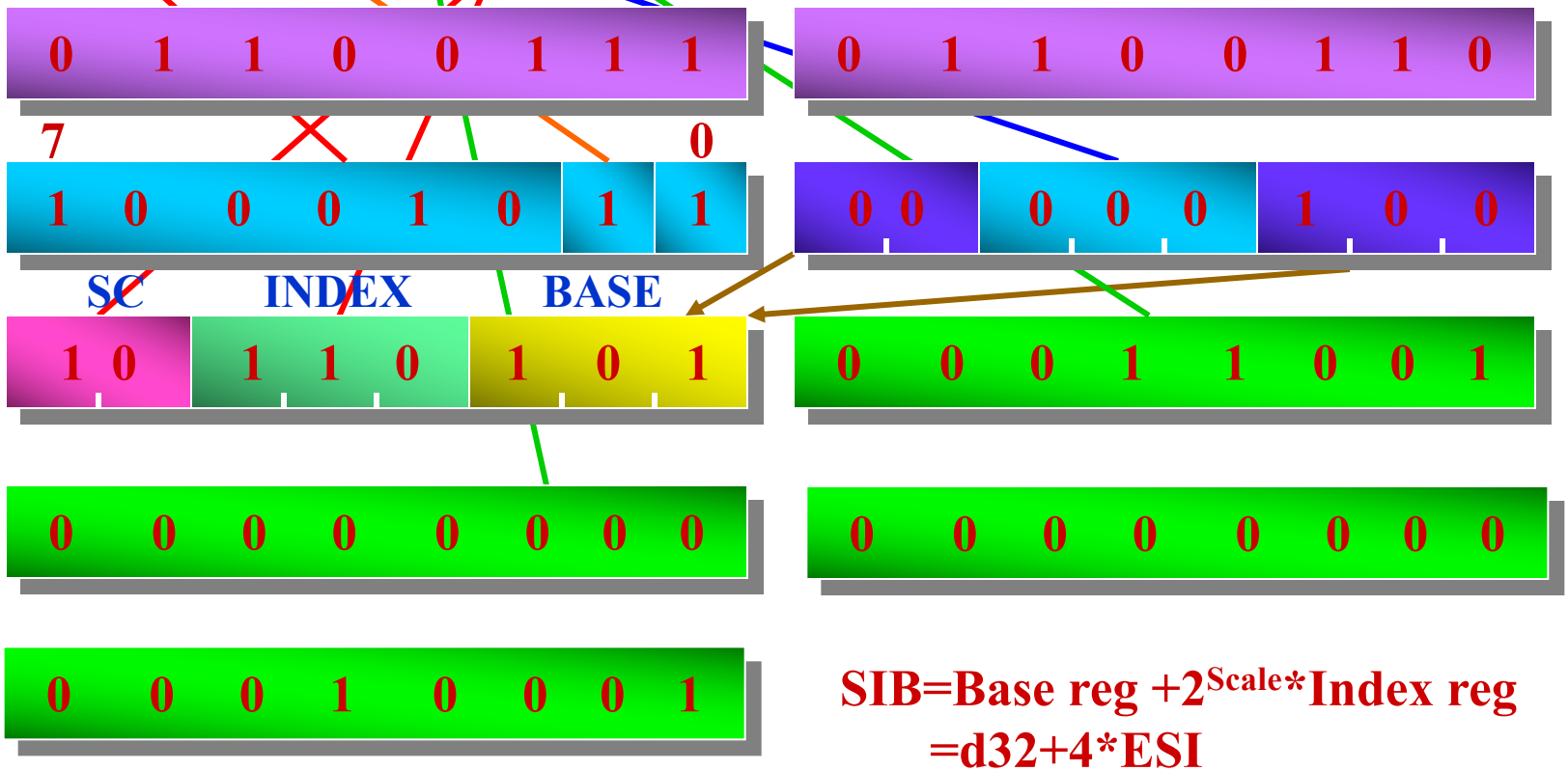
指令格式举例（1）

ADD DST，SRC；（DST）←（DST）+（SRC）



指令格式举例（2）

MOV EAX, COUNT[ESI*4]; 67668B04B519000011



SIB=Base reg +2^{Scale}*Index reg
=d32+4*ESI


BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
北京邮电大学

3.4 x86指令系统

80x86指令系统

1. 数据传送指令
2. 算术运算指令
3. 逻辑运算指令
4. 串处理指令
5. 控制转移指令
6. 处理机控制指令

1. 数据传送指令

（1）通用数据传送指令

（2）累加器专用传送指令

（3）地址传送指令

（4）标志寄存器传送指令

（5）类型转换指令

（1）通用数据传送指令

- MOV　把源操作数传送到目的操作数
- PUSH　源操作数进栈
- POP　出栈到目的操作数
- XCHG　交换源，目的操作数

80386+

- MOVSX 带符号扩展传送
- MOVZX 带零扩展传送
- PUSHA/PUSHAD 所有寄存器进栈
- POPA/POPAD 所有寄存出栈

MOV DST, SRC

执行操作：

（DST）←（SRC）

寻址方式：

MOV reg, reg

MOV mem, reg

MOV reg, mem

MOV reg, imm

MOV mem, imm

标志位不受影响

例

```
mov al,4          ;  al←4，字节传送
mov cx,0ffh       ;  cx←00ffh，字传送
mov si,200h       ;  si←0200h，字传送
mov ax,bx         ;  ax←bx，字传送
mov ah,al         ;  ah←al，字节传送
mov ds,ax         ;  ds←ax，字传送
mov [bx],al       ;  [bx]←al，字节传送
mov al,[bx]
mov dx,[bp]       ;  dx←ss:[bp]
mov es,[si]       ;  es←ds:[si]
```

MOVSX DST, SRC

执行操作：

（DST）← 符号扩展（SRC）

寻址方式：

MOVSX reg1, reg2

MOVSX reg, mem

标志位不受影响

源操作数可以是8位或16位，目的操作数必须是16位或32位的寄存器

例：MOVSX AX，BL

若：（BL）= 82H 则执行后 （AX）= FF82H

MOVZX DST, SRC

执行操作：

（DST）← 零扩展（SRC）

寻址方式：

MOVZX reg1, reg2

MOVZX reg, mem

标志位不受影响

源操作数可以是8位或16位，目的操作数必须是16位或32位的寄存器

例：MOVZX AX，BL

若：（BL）= 82H 则执行后 （AX）= 0082H

PUSH SRC（1）

执行操作：

16位指令：

(SP) ← (SP)-2

((SP)+1, (SP))← (SRC)

32位指令：

(ESP) ← (ESP)-4

((ESP)+3, (ESP)+2, (ESP)+1, (ESP)) ← (SRC)

标志位不受影响

PUSH SRC（2）

寻址方式

PUSH mem/reg

PUSH imm

例：PUSH [BX]；PUSH [EBX]；PUSH AX；PUSH EAX

PUSH segreg

例：PUSH DS

PUSH data

例：PUSH 8F00H；

PUSHA：AX,CX,DX,BX SP,BP,SI和DI依次进栈

PUSHAD：EAX,ECX,EDX,EBX,ESP,EBP,ESI和EDI依次进栈

PUSHF：16位标志寄存器进栈

PUSHFD： 32位标志寄存器进栈

POP DST（1）

执行操作：

16位指令：

(DST) ← ((SP)+1, (SP))

(SP) ← (SP)+2

32位指令：

(DST) ← ((ESP)+3, (ESP)+2, (ESP)+1, (ESP))

(ESP) ← (ESP)+4

标志位不受影响

POP DST （2）

寻址方式：

POP mem/reg

例：POP [BX]； POP[EBX]； POP AX； POP EAX

POP segreg

例： POP DS

POPA： DI,SI,BP,SP,BX,DX,CX和AX依次出栈

POPAD：EDI,ESI,EBP,ESP,EBX,EDX,ECX和EAX依次出栈

POPF：16位标志寄存器出栈

POPFD： 32位标志寄存器出栈

堆栈的应用

例：在子程序中保护和恢复现场

```
    ⋮
push    ax
push    bx
push    cx
push    dx
    ⋮
    ⋮                    ；使用ax，bx，cx，dx

pop     dx
pop     cx
pop     bx
pop     ax
ret
```

保护现场

恢复现场

堆栈溢出问题

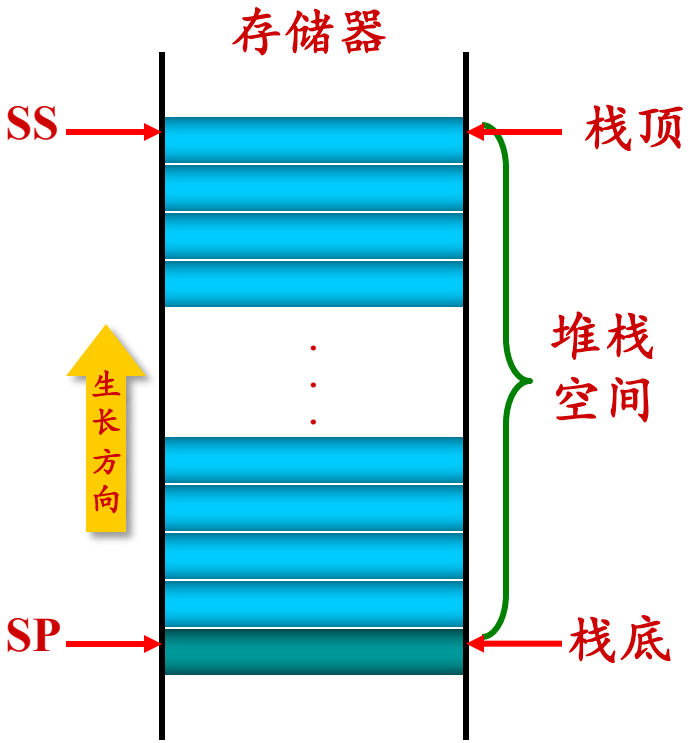
通常，CPU不自动判断堆栈是否溢出。堆栈是否溢出应由应用程序来判定。

堆栈满 ⟷ (SP) = 0

如果定义堆栈大小为100字节，则程序装载器自动将SP初始化为100。

如果把堆栈设置为足够大，则可不判断堆栈是否溢出

? 能否访问不在栈顶的数据？



XCHG OPR1, OPR2

执行操作：

（OPR1）↔（OPR2）

寻址方式：

XCHG mem/reg, mem/reg

例：XCHG BX, [BP+SI]

标志位不受影响

（2）累加器专用传送指令

- IN　从端口输入
- OUT　向端口输出
- XLAT　换码

只限于使用累加器EAX，AX或AL
传送信息该类指令不影响标志位

IN和OUT指令的长格式

IN AL,〈端口地址〉

IN AX,〈端口地址〉

IN EAX,〈端口地址〉

OUT 〈端口地址〉, AL

OUT 〈端口地址〉, AX

OUT 〈端口地址〉, EAX

长格式中的端口地址必须是8位的，
即端口地址为0～0ffh

IN和OUT指令的长格式

执行操作：

字节： IN (AL) ←(port)
OUT (port) ←(AL)

字： IN (AX) ←(port+1, port)
OUT (port+1, port) ←(AX)

双字： IN (EAX) ←(port+3, port+2, port+1, port)
OUT (port+3, port+2, port+1, port) ←(EAX)

例： IN AL, 20H
OUT 60H, AL

IN和OUT指令的短格式

```
IN  AL, DX
IN  AX, DX
IN  EAX, DX
OUT  DX, AL
OUT  DX, AX
OUT  DX,  EAX
```

端口地址必须放在DX寄存器中，
端口地址范围为0000～0ffffh

IN和OUT指令的短格式

执行操作：

字节： IN (AL) ←((DX))
OUT ((DX)) ←(AL)

字： IN (AX) ←((DX)+1, (DX))
OUT ((DX)+1, (DX)) ←(AX)

双字：

IN (EAX) ←((DX)+3, (DX)+2, (DX)+1, (DX))
OUT ((DX)+3, (DX)+2, (DX)+1, (DX)) ←(EAX)

例：MOV DX, 3F8H
OUT DX, AL

XLAT换码指令

指令格式：

XLAT

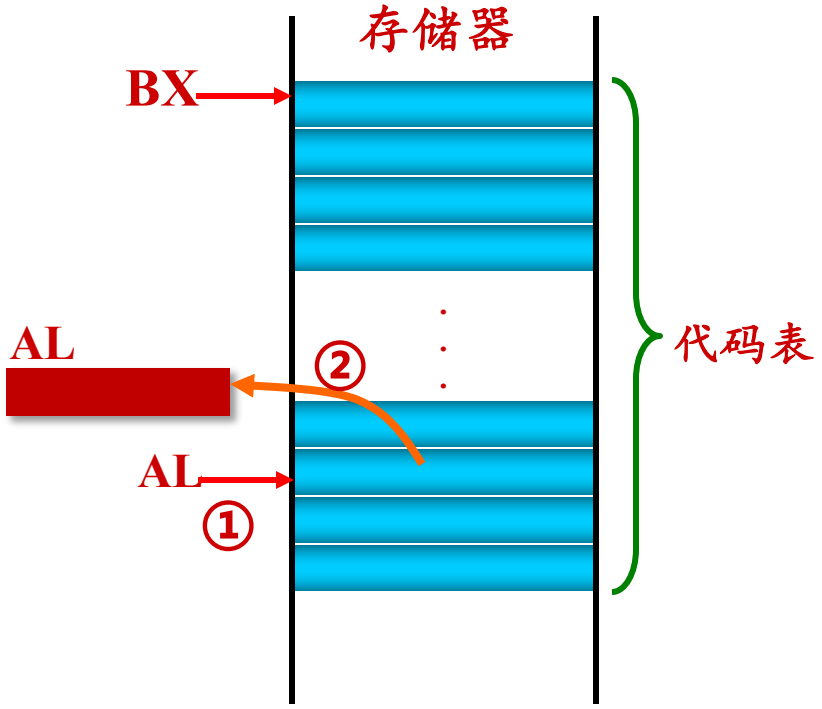
执行操作:

16位指令：

(AL) ←((BX) + (AL))

32位指令：

(AL) ←((EBX) + (AL))



BX指向代码表首地址，AL中保存代码表中字符序号（偏移量），执行后，AL取得对应的码值（表项内容）

（3）地址传送指令

- LEA　将有效地址送寄存器
- LDS　将存储器中的双字/三字送寄存器和DS
- LES　将存储器中的双字/三字送寄存器和ES

386+机型

- LFS　将存储器中的双字/三字送寄存器和FS
- LGS　将存储器中的双字/三字送寄存器和GS
- LSS　将存储器中的双字/三字送寄存器和SS

标志位不受影响

LEA reg, SRC

- 执行操作：源操作数的有效地址送寄存器

  (reg) ←SRC的有效地址EA

- 如：

  LEA BX, TABLE ；将TABLE的有效地址送BX

  LEA EAX, TABLE ；将32位地址送EAX

LDS、LES、LFS、LGS和LSS

源操作数的寻址方式为存储器寻址，将存储器的内容（即：远指针）送寄存器和段寄存器。段寄存器用指令助记符指明。（也称为指针传送指令）以LDS指令为例：

指令格式：

LDS reg, SRC

执行操作：

(reg) ←(SRC)

(DS) ←(SRC+2) 或 (DS)←(SRC+4)

例：

LDS EDI, [BX]；将BX指向的4字节存储单元送

；EDI，后续的2字节存储单元送DS

（4）标志寄存器传送指令

- LAHF　　标志寄存器送AH
- SAHF　　AH送标志寄存器
- PUSHF/PUSHFD　　标志寄存器进栈
- POPF/POPFD　　标志寄存器出栈

LAHF与SAHF

指令格式：

LAHF

执行操作：

(AH) ←(FLAGS的低字节）

指令格式：

SAHF

执行操作：

(FLAGS的低字节）← (AH)

PUSHF与PUSHFD

指令格式：

PUSHF/PUSHFD

执行操作：

(SP) ←(SP) - 2/ (ESP) ←(ESP) - 4

((SP)+1, (SP)) ←(FLAGS)/

((ESP)+3, (ESP)+2, (ESP)+1, (ESP)) ←((EFLAGS) ∧ 0FCFFFFH) (即清除VM和RF位）

POPF与POPFD

指令格式：

POPF/POPFD

执行操作：

(FLAGS) ←((SP)+1, (SP)) /

(EFLAGS) ← ((ESP)+3, (ESP)+2, (ESP)+1, (ESP))

(SP) ←(SP) + 2/

(ESP) ←(ESP) + 4

（5）类型转换指令（1）

- **CBW** 字节转换为字

  AL的符号位扩展到AH，转换成AX中的字

- **CWD /CWDE** 字转换为双字

  AX的符号位扩展到DX，转换成DX:AX中的双字

  AX的内容符号扩展到EAX，转换成EAX中的双字

- **CDQ** 双字转换为四字

  EAX的符号位扩展到EDX，转换成EDX:EAX中的四字

标志位均不受影响

2. 算术运算指令

（1）加法指令

（2）减法指令

（3）乘法指令

（4）除法指令

（5）十进制调整指令

（1）加法指令

- ADD　加法
- ADC　带进位加法
- INC　加1指令
- XADD　交换并相加

加法的CF位和OF位的说明

- CF位：对无符号数有意义
  - CF=1 最高有效位向高位有进位
  - CF=0 最高有效位向高位无进位
- OF位：对有符号数有意义
  - OF=1 若两个操作数的符号相同，而结果的符号与之相反
  - OF=0 其他情况

溢出问题

（1）无符号数加法

```
  00001001    00000010
+ 01111100  + 11111111
----------  ----------
  10000101   100000001
   CF=0        CF=1
   OF=1        OF=0
    ✓           ✗
```

（2）有符号数加法

```
  00001001    00000010
+ 01111100  + 11111111
----------  ----------
  10000101   100000001
   CF=0        CF=1
   OF=1        OF=0
    ✗           ✓
```

在汇编程序中，运算是否正确需要由程序员进行判断！

ADD和ADC

指令格式：

ADD　　DST, SRC

ADC　　DST, SRC

标志位均受影响

执行操作：

ADD：　(DST) ← (DST)+(SRC)

ADC：　(DST) ← (DST)+(SRC)+CF

寻址方式：

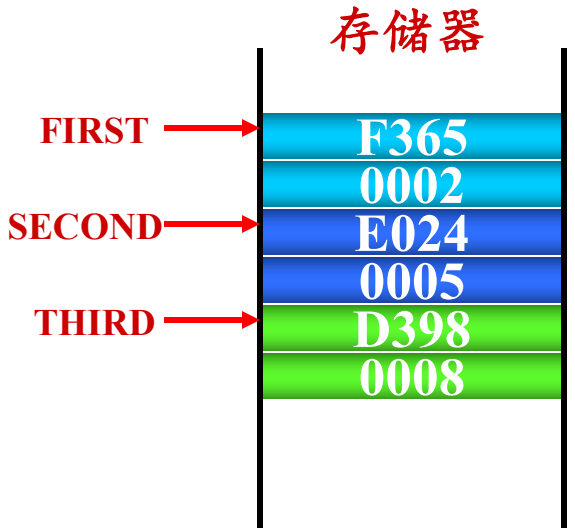
ADD/ADC　　reg, mem/reg/imm

ADD/ADC　　mem, imm /reg

举例

用16位的操作数指令完成32位的加法

```
LEA BX，FIRST
LEA SI，SECODN
LEA DI，THIRD
MOV AX，[BX]
ADD AX，[SI]
MOV [DI]，AX
MOV AX，[BX+2]
ADC AX，[SI+2]
MOV [DI+2]，AX
```



INC

指令格式：

INC　OPR

执行操作：

(OPR) ← (OPR)+1

寻址方式：

INC　mem/reg

除CF标志外，其他标志位均受影响

XADD

指令格式：

XADD DST, SRC

执行操作：

TEMP← (DST)+(SRC)

(SRC) ← (DST)

(DST) ← TEMP

寻址方式：

XADD mem/reg, reg

（2）减法指令

- SUB　减法
- SBB　带借位减法
- CMP　比较
- DEC　减1指令
- NEG　翻转有符号数的符号
- CMPXCHG 比较并交换
- CMPXCHG8B 比较并交换8字节

标志位均受影响

减法的CF位和OF位的说明

- CF位：对无符号数有意义
  - CF=1 表示无符号数运算中有借位产生。即：被减数＜减数
  - CF=0 表示无符号数运算中无借位产生
- OF位：对有符号数有意义
  - OF=1 若两个操作数的符号相反，而结果的符号与减数相同
  - OF=0 其他情况

减法中CF位的生成

减法指令在机器中实际上是用补码通过加法运算实现的

例：3 – 5=[3]$_{补}$+[-5]$_{补}$

C1

00000011 求补

+11111011 ← 10000101

0111111110

CF=C1 ⊕ Sub=0 ⊕ 1=1

减法指令置Sub=1

例：7 - 5=[7]$_{补}$+[-5]$_{补}$

C1

00000111 求补

+11111011 ← 10000101

100000010

CF=C1 ⊕ Sub=1 ⊕ 1=0

减法指令置Sub=1

SUB、SBB和CMP

指令格式：

SUB/SBB/CMP　　DST, SRC

目的操作数-源操作数
注意顺序！

执行操作：

SUB:　　(DST)← (DST) - (SRC)

SBB:　　(DST)← (DST) - (SRC) - CF

CMP:　　(DST) - (SRC)

只根据结果设置各条件标志位

寻址方式：

SUB/SBB/CMP　reg, mem/reg/imm

SUB/SBB/CMP　mem, imm /reg

DEC

指令格式：

DEC　　OPR

执行操作：

(OPR)← (OPR) - 1

寻址方式：

DEC　mem/reg

除CF标志外，其他标志位均受影响

NEG（Negate）

指令格式：

NEG　　OPR

执行操作：

(OPR) ← – (OPR)

寻址方式：

NEG　mem/reg

指令执行结果就是把一个正数变成负数或把一个负数变成正数。即：把操作数“按位求反后末位加1”，也可写成为：（OPR）← FFFFH -（OPR）+1

- 对标志位的影响
  - CF：只有操作数为0时CF才为0，否则CF为1
  - OF：只有操作数为最小负数时OF才为1，其他情况为0。

例：对80H（-128）或8000H（-32768）等求补时，操作数不变，OF=1。

CMPXCHG

指令格式：

CMPXCHG　　DST, SRC

执行操作：累加器AC与DST比较

若　　(AC)= (DST)

则　ZF ←1，(DST) ←(SRC)

否则　ZF ←0，(AC) ←(DST)

寻址方式：

CMPXCHG　reg/mem，reg8/16/32

仅适用于80486

CMPXCHG8B

指令格式：

CMPXCHG8B　　DST

执行操作：EDX，EAX与DST比较

若　(EDX，EAX)= (DST)

则　ZF ←1，(DST) ←(ECX，EBX)

否则　ZF ←0，(EDX, EAX) ←(DST)

寻址方式：源操作数存放在EDX，EAX中

CMPXCHG　mem

只有ZF标志位受影响
适用于Pentium及其后继机型

（3）乘法指令

- MUL　无符号数乘法
- IMUL　有符号数乘法

对除CF和OF位以外的条件码位无定义

MUL和IMUL

指令格式：有符号数乘法

MUL/IMUL　SRC

执行操作：

字节操作数：(AX) ←(AL) × (SRC)

字操作数：(DX, AX) ←(AX) × (SRC)

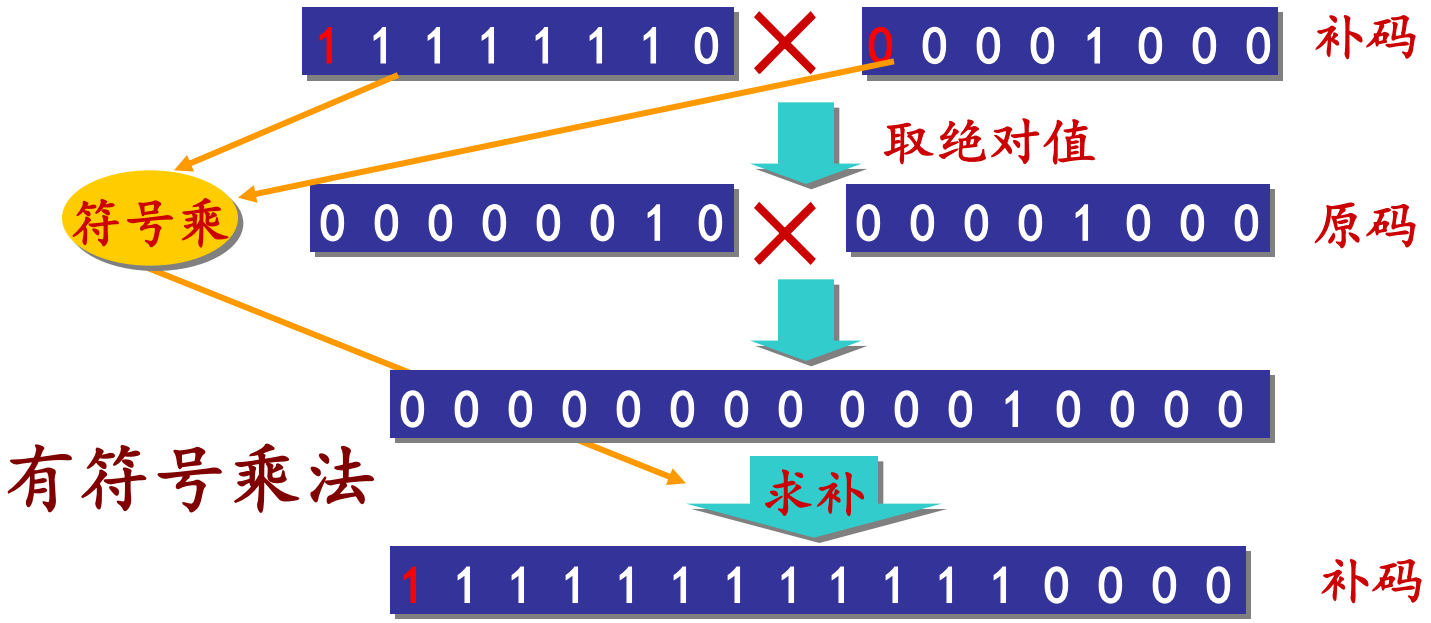
双字操作数：(EDX, EAX) ←(EAX) × (SRC)

寻址方式：目的操作数必须是16位或32位寄存器

MUL/IMUL　mem/reg

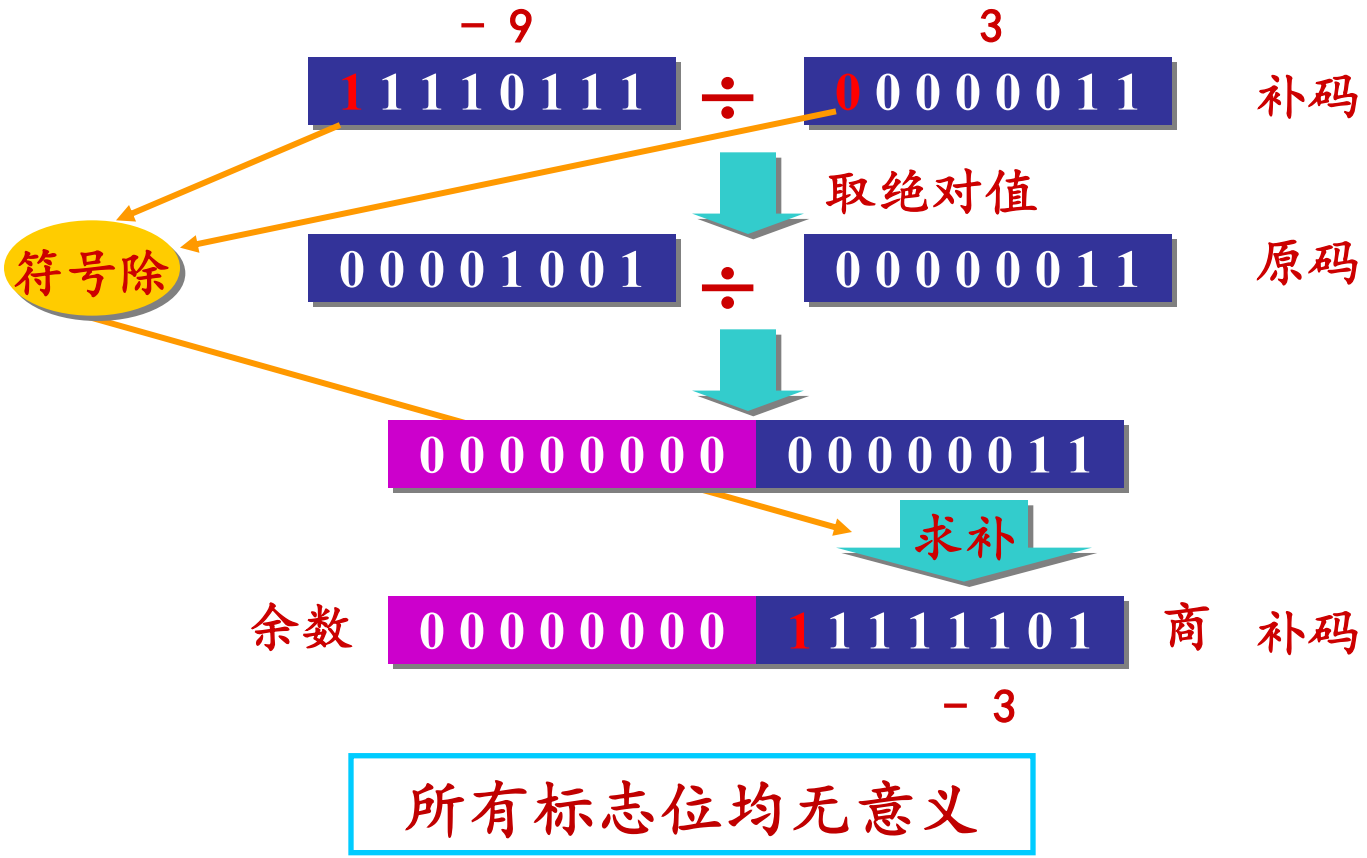
若乘积的高一半为0，则CF位和OF位均为0；否则为1。因此，可用CF或OF位来检测乘积结果的宽度

IMUL有符号数乘法


1 1 1 1 1 1 1 0
0 0 0 0 1 0 0 0
补码
取绝对值
符号乘
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
原码
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
有符号乘法
求补
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
补码

（4）除法指令

- DIV　无符号数除法
- IDIV　有符号数除法



DIV和IDIV

指令格式：

DIV/IDIV　SRC

执行操作：

字节操作数：(AL) ←(AX) / (SRC)的商

(AH) ←(AX) / (SRC)的余数

字操作数：(AX) ←(DX, AX) / (SRC)的商

(DX) ←(DX, AX) / (SRC)的余数

双字操作数：(EAX) ←(EDX, EAX) / (SRC)的商

(EDX) ←(EDX, EAX) / (SRC)的余数

寻址方式：DIV/IDIV　mem/reg

（5）十进制调整指令

压缩的BCD码调整指令

- DAA（Decimal Adjustment after Addition）BCD码的加法十进制调整
- DAS（Decimal Adjustment after Subtraction ）BCD码的减法十进制调整

非压缩的BCD码调整指令

- AAA（ASCII Adjust after Addition）非压缩BCD码的加法十进制调整
- AAS（ASCII Adjust after Subtraction）非压缩BCD码的减法十进制调整
- AAM（ASCII Adjust after Multiplication）非压缩BCD码的乘法的十进制调整
- AAD（ASCII Adjust for Division）非压缩BCD码的除法的十进制调整

十进制调整指令的引入

- 80x86算术运算指令都是面向二进制运算的，但人们习惯使用十进制，自然希望输入，输出都是十进制数。利用二进制指令实现十进制数运算有下述两种解决方案
  - 先将十进制数转换为二进制，进行运算，最后把结果再转换为十进制
  - 用BCD（Binary Coded Decimal）码表示十进制数，直接使用二进制指令对其进行算术运算，然后再对结果进行调整。80x86提供十进制调整指令支持该方案

BCD码

- 压缩的BCD码

例：45 → 01000101B

- 非压缩的BCD码

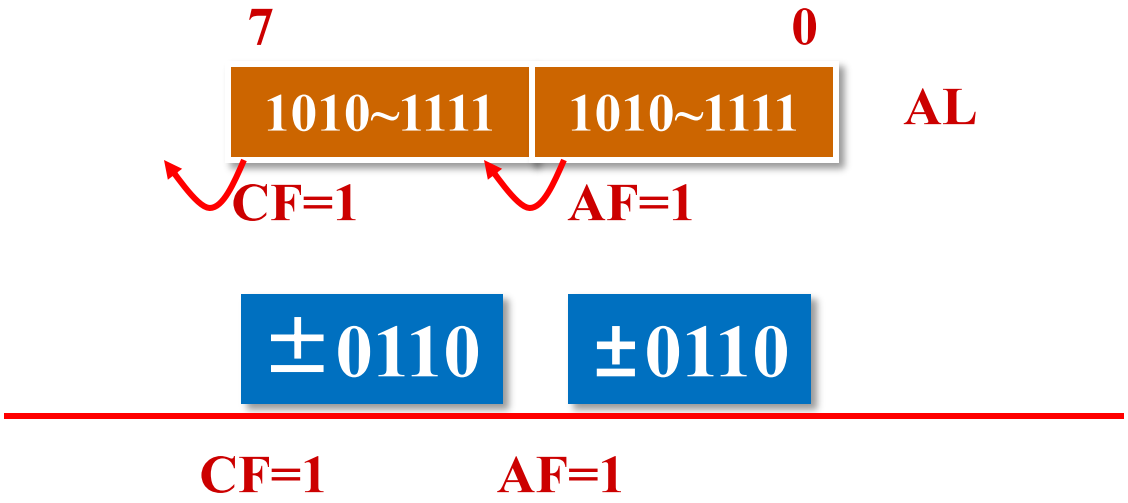
- 数字的ASCII可视为一种非压缩的BCD码

例：“5” → 00110101B

DAA和DAS指令

执行操作：DAA和DAS分别放在加法指令（ADD和ADC）和减法指令之后（结果应在AL中），对BCD码的运算结果进行调整。

调整规则：若运算结果为1010~1111或向高位有进/借位（AF=1或CF=1）时，则加6或减6。



3. 逻辑指令

（1）逻辑运算指令

（2）移位指令

（1）逻辑运算指令

- **AND** 按位逻辑与
- **OR** 按位逻辑或
- **XOR** 按位逻辑异或
- **TEST** 按位逻辑与，但不保存结果
- **NOT** 按位取反

AND、OR和XOR

指令格式：

AND/OR/XOR　　DST, SRC

执行操作：

AND:　　(DST)← (DST) ∧ (SRC)

OR:　　(DST)← (DST) ∨ (SRC)

XOR:　　(DST)← (DST) ∀ (SRC)

寻址方式：

AND/OR/XOR　　reg, mem/reg/imm

AND/OR/XOR　　mem, reg/imm

CF=0和OF=0，影响PF、ZF和SF；AF无意义

TEST

指令格式：

TEST　　OPR1, OPR2

执行操作：

(OPR1) ∧ (OPR2)

寻址方式：

TEST　　reg, mem/reg/imm

TEST　　mem, reg/imm

CF=0和OF=0，影响PF、ZF和SF；AF无意义

NOT

指令格式：

NOT　　OPR

执行操作：

$(OPR) \leftarrow (\overline{OPR})$

寻址方式：

NOT　　reg/mem

对标志位无影响

（2）移位指令

- SAL　算术左移
- SAR　算术右移
- SHL　逻辑左移
- SHR　逻辑右移
- ROL　循环左移
- ROR　循环右移
- RCL　带进位循环左移
- RCR　带进位循环右移

移位指令

指令格式：

移位指令　　OPR，CNT

寻址方式：

移位指令　　reg/mem，1/CL/imm

移动1位时可用1直接指明，移动多位时移动的位数可放在CL寄存器中

移位指令执行操作

SAL：算术左移
SHL：逻辑左移

SHR：逻辑右移

SAR：算术右移

当CNT=1时，
　若移位前后最高位发生变化，则OF=1
　若移位前后最高位无变化，则OF=0
当CNT>1时，OF无意义
影响CF、PF、ZF和SF；AF无意义

循环移位指令执行操作

ROL：循环左移

ROR：循环右移

RCL：带进位循环左移

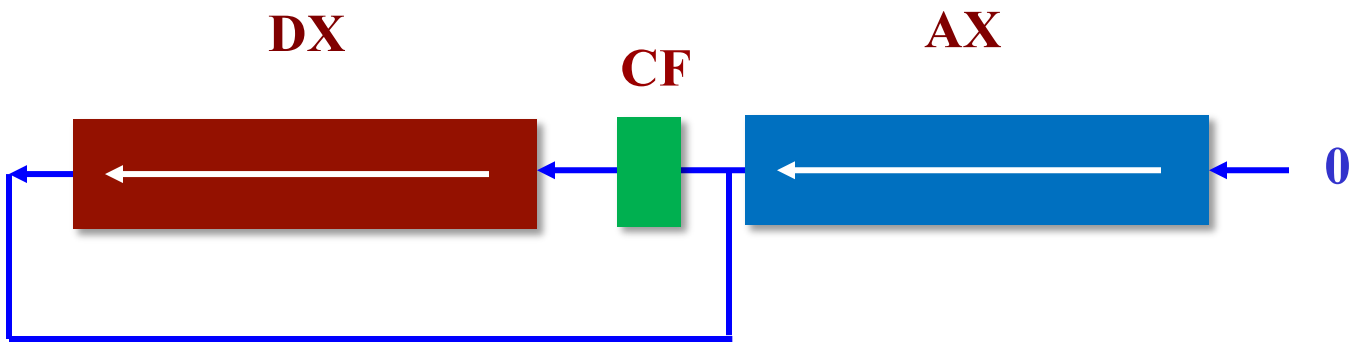
RCR：带进位循环右移

只影响CF和OF（OF位只当CNT=1才有效）

举例

- 将DX:AX中的32位数左移1位

```
SHL   AX, 1
RCL   DX, 1
```



4. 串处理指令

（1）串操作指令

（2）重复前缀

（3）类型后缀

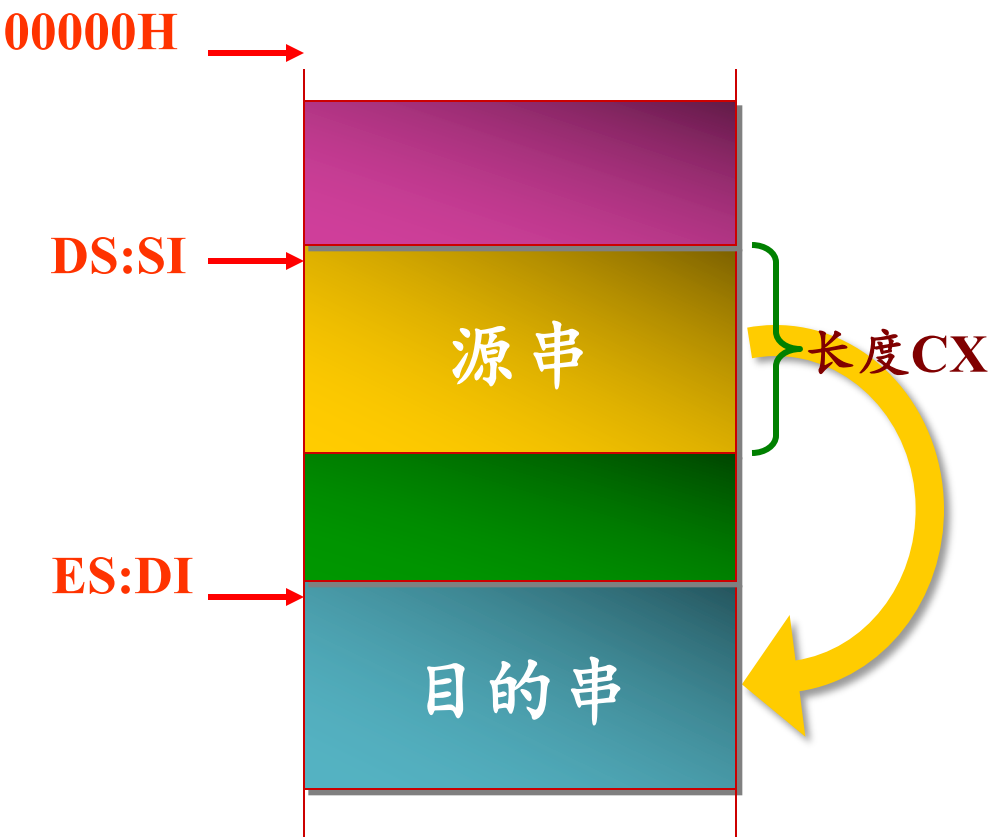
（4）串处理方向

数据串（String）:在内存中，连续存放的字节或字序列

（1）串操作指令

- MOVS　串传送指令
- STOS　串存储指令
- LODS　串加载指令
- CMPS　串比较指令
- SCAS　串扫描指令
- INS　串输入指令
- OUTS　串输出指令

串操作指令



DF的取值0或1决定SI、DI做递增还是递减操作

串传送指令：MOVSB/W/D

ES:DI 指向目的串，DS:SI指向源串。

执行操作：

(i) ((DI)) ←((SI))

(ii) (SI) ← (SI) ±1/2/4, (DI) ←(DI) ±1/2/4

加或减操作由标志位DF=0或1决定，加减1、2或4取决于串操作类型，由指令后缀/B/W/D决定

常与REP联合使用，例如：

```
REP MOVSB
REP MOVSW
REP MOVSD
```

串传送举例

把DS段中block1的50个连续的字节传送到ES段的block2中。

```
        lea si, block1
        lea di, block2
        mov cx, 50          ;  cx←传送次数
        cld                 ;  置DF=0，地址递增
lp:     movsb               ;  传送一字节
        dec cx              ;  传送次数－1
        jnz lp              ;  结果不为零转跳
```

串存储指令：STOSB/W/D

将累加器内容存入目的串，ES:DI指向目的串。

执行操作：

(i) ((DI)) ← (AL/AX/EAX)

(ii) (DI) ←(DI) ±1/2/4

加或减操作由标志位DF=0或1决定，加减1、2或4取决于串操作类型，由指令后缀B/W/D决定

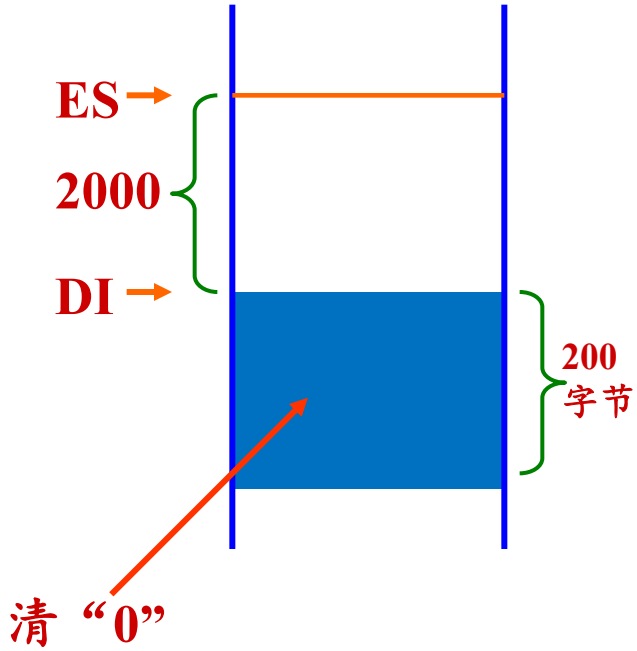
常与REP联合使用，例如：

```
REP STOPSB
REP STOPSW
REP STOPSD
```

串存储举例

把ES段中由DI指向的内存中的100个连续字单元清0。

```
        mov  ax,0
        mov  di,2000  ;  在ES段中
        mov  cx,100   ;  cx←传送次数100
        cld           ;  DF=0，地址递增
lp:     stosw         ;  存储一个字
        dec  cx       ;  次数减1
        jnz  lp       ;  结果不为零转跳
```



串加载指令：LODSB/W/D

将源串内容存入累加器，DS:SI指向源串。

执行操作：

(i) (AL/AX/EAX) ←((SI))

(ii) (SI) ←(SI) ±1 / 2 / 4

加或减操作由标志位DF=0或1决定，加减1、2或4取决于串操作类型，由指令后缀W/B/D决定

常与REP联合使用，例如：

```
REP LODSB
REP LODSW
REP LODSD
```

串比较指令：CMPSB/W/D

ES:DI 指向目的串，DS:SI指向源串。

执行操作：

(i) ((DI)) -((SI)) 不保存结果，但结果反映在标志位

(ii) (SI) ← (SI) ±1/2/4 , (DI) ←(DI) ±1/2/4

加或减操作由标志位DF=0或1决定，加减1、2或4取决于串操作类型，由指令后缀B/W/D决定

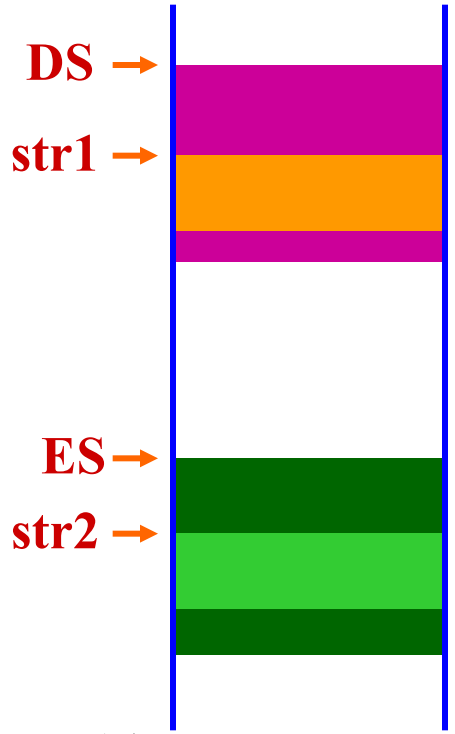
常与REPZ/REPNZ联合使用，例如：

```
REPZ  CMPSB        REPNZ  CMPSB
REPZ  CMPSW        REPNZ  CMPSW
REPZ  CMPSD        REPNZ  CMPSD
```

串比较举例

比较两个长度为strlen的字符串str1和str2

```
        lea si,str1
        lea di,str2
        mov cx,strlen
        cld             ; DF=0，地址递增
lp1:    cmpsb           ; 比较两个字符
        jnz lp2         ; 字符不相同，转移
        dec cx
        jnz lp1         ; 比较下一个字符
        mov al,0        ; 0表示字符串相等
        jmp lp3         ; 转向lp3
lp2:    mov al,0ffh         ; 255表示字符串不等
lp3:    ...
```



串扫描指令：SCASB/W/D

ES:DI 指向目的串，

执行操作：

(i) ((DI)) -(累加器) 不保存结果，结果反映在标志位

(ii) (DI) ←(DI) ±1 / 2 / 4

加或减操作由标志位DF=0或1决定，加减1、2或4取决于串操作类型，由指令后缀B/W/D决定

常与REPZ/REPNZ联合使用，例如：

REPZ SCASB	REPNZ SCASB
REPZ SCASW	REPNZ SCASW
REPZ SCASD	REPNZ SCA

串扫描举例

在长度为strlen字符串string中查找字母C

```
        lea di,string
        mov al,43h              ; C的ASCII码，mov al, 'C'
        mov cx,strlen
        cld                     ; DF=0，地址递增
lp:     scasb                   ; 搜索
        jz found                ; 为0（ZF=1），发现C
        dec cx                  ; 不是C
        jnz lp                  ; 搜索下一个字符
        ...                     ; 不含C，则继续执行
        ...
found:...
```

串输入/出指令：INS/OUTSB/W/D

INS：把端口号在DX中的I/O空间的字节、字或双字送到附加段中的由目的变址寄存器DI所指向的存储单元中

OUTS：把源变址寄存器SI所指向的数据段中的字节、字或双字单元送到端口号在DX中的I/O的I/O端口中

串输入指令：INSB/W/D

ES:DI 指向目的串，

执行操作：

(i) ((DI)) ← ((DX))

(ii) (DI) ←(DI) ±1 / 2 / 4

加或减操作由标志位DF=0或1决定，加减1、2或4取决于串操作类型，由指令后缀B/W/D决定

串输出指令：OUTSB/W/D

DS:SI 指向源串，

执行操作：

(i) ((DX)) ← ((SI))

(ii) (SI) ←(SI) ±1 / 2 / 4

加或减操作由标志位DF=0或1决定，加减1、2或4取决于串操作类型，由指令后缀B/W/D决定

（2）重复前缀

与串操作指令配合工作，分为两种：

- **REP** 无条件重复
- **REPE（REPZ）**
- **REPNE（REPNZ）**

} 有条件重复

REP

格式：

REP　　MOVS/LODS/STOS/INS/OUTS

执行操作：

①如果(CX/ECX)=0，则退出REP，否则执行②

②(CX/ECX) ←(CX/ECX) - 1

③执行其后的串指令

④重复①～③

REP举例

把block1中的50个连续的字节传送到block2中

```
lea si,block1
lea di,block2
mov cx,50                ; cx←传送次数
cld                      ; 置DF=0，地址递增
rep movsb                ; lp: movsb
                         ;     dec cx
                         ;     jnz lp
```

REPE（REPZ）

格式：

REPE(REPZ)　　CMPS/SCAS

执行操作：

①如果(CX/ECX)=0或ZF=0,则退出REP;否则执行②

②(CX/ECX) ←(CX/ECX) - 1

③执行其后的串指令

④重复①～③

REPE举例

比较两个长度为strlen的字符串str1和str2

```
        lea si,str1
        lea di,str2
        mov cx,strlen
        cld                 ; DF=0，地址递增
        repe cmpsb          ; 比较两个字符
        jnz lp2             ; 字符不相同，转移
                            ; dec cx
                            ; jnz lp1
        mov al,0            ; 0表示字符串相等
        jmp lp3             ; 转向lp3
lp2:    mov al,0ffh         ; 255表示字符串不等
lp3:    ...
```

REPNE（REPNZ）

格式：

REPNE(REPNZ)　　CMPS/SCAS

执行操作：

①如果(CX/ECX)=0或ZF=1,则退出REP;否则执行②

②(CX/ECX) ←(CX/ECX) - 1

③执行其后的串指令

④重复①～③

动手做

1、rep lodsw指令的写法是否正确？

2、repnz cmpsb指令执行动作是重复执行cmpsb指令，直到比较的结果相等退出。是否正确？

5. 控制转移指令

（1）无条件转移指令

（2）条件转移指令

（3）循环指令

（4）子程序调用指令

（5）中断指令

（1）无条件转移指令

指令格式

JMP 〈转移地址〉

程序无条件转移到〈转移地址〉指定的位置继续执行

转移分为两类

- 段内转移：只修改IP或EIP寄存器的内容

  转移范围为+32K-1 ~ - 32K或+2G -1~ -2G
- 段间转移：不仅修改IP或EIP寄存器的内容，还要修改CS寄存器的内容。

段内直接短转移

指令格式：

JMP SHORT label（标号）

执行操作：

(EIP)←(EIP) + 8位位移量

转移范围为+127~-128

段内直接近转移

指令格式：

JMP [NEAR PTR] label（标号） 注:默认为近转移

执行操作：

(IP)←(IP) + 16位位移量

转移范围为+32767~ -32768

或(EIP)←(EIP) + 32位位移量

转移范围为+2G -1~ -2G

段内间接近转移

指令格式：

JMP WORD PTR reg/mem

执行操作：

(IP)或(EIP)←(EA)

其中有效地址EA由寄存器或存储器寻址方式确定

转移范围为+32K -1 ~ - 32K或+2G -1~ -2G

段间直接远转移

指令格式：

JMP FAR PTR label（标号）

执行操作：

(IP)或(EIP)←label的段内偏移地址

(CS)←label所在段的段地址

段间间接远转移

指令格式：

JMP DWORD PTR mem

执行操作：

(IP)或(EIP)← (EA)

(CS)← (EA+2/4)

其中有效地址EA由存储器寻址方式确定

例如：mov word ptr [bx],100 ；置段内偏移值

mov word ptr [bx+2],2000h；置段基址

jmp dword ptr [bx] ；转跳到2000h:100

（2）条件转移指令

条件转移指令用于实现最基本的程序结构之一：分支结构。

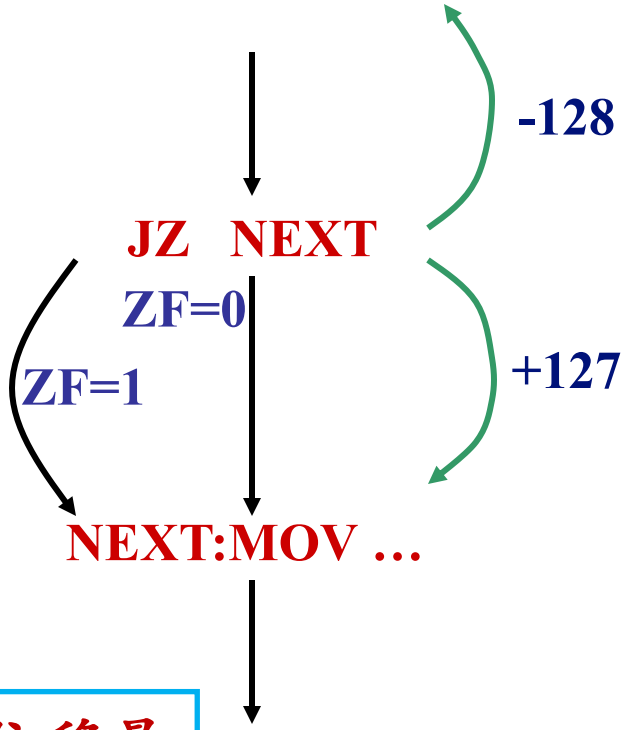
指令格式：

条件转移指令　label（标号）

标号的位移量只能是 8 位的

执行操作：

(IP) ←(IP) + 8位位移量



由汇编程序计算出标号的位移量，它是目标地址与IP寄存器的内容（下一条指令地址）之差

单个测试条件

J*指令的含义：

条件标志*表示 当* =1时，则转移

JN*指令的含义

条件标志N*表示当 * =0时，则转移

JZ/JE	JNZ/JNE
JS	JNS
JO	JNO
JP/JPE	JNP/JPO
JC/JB/JNAE	JNC/JNB/JAE

两数比较

无符号数比较：

JC/JB　　OPR；小于，CF=1

JNC/JNB　OPR；大于等于，CF=0

JBE/JNA　OPR；小于等于，CF∨ZF=1

JNBE/JA　OPR；大于，　CF∨ZF=0

有符号数比较：

JL/JNGE　OPR；小于，SF ⊕ OF=1

JNL/JGE　OPR；大于等于，SF ⊕ OF=0

JLE/JNG　OPR；小于等于，(SF ⊕ OF)∨ZF=1

JNLE/JG　OPR；大于，　(SF ⊕ OF)∨ZF=0

测试计数寄存器

指令格式：

JCXZ　OPR

执行操作：

如果(CX)=0，则转移

80386+指令

指令格式：

JECXZ OPR

执行操作：

如果(ECX)=0，则转移

（3）循环指令

（1）LOOP　OPR　　类似　　DEC　CX
　　JNZ　OPR

测试条件：(CX)≠0

（2）LOOPZ/LOOPE　OPR

测试条件：ZF=1且(CX)≠0

（3）LOOPNZ/LOOPNE　OPR

测试条件：ZF=0且(CX)≠0

执行操作：

①(CX)←(CX) - 1

②测试条件，若条件成立，则

(IP)←(IP) + 8位位移量

不影响条件标志位

举例1

求首地址为ARRAY、长度为N的字数组的内容之和。

```
      MOV   CX,N
      MOV   AX,0          ;xor ax,ax
      MOV   SI,AX
LP:   ADD   AX,ARRAY[SI]
      ADD   SI,2
      LOOP  LP
      MOV   TOTAL,AX
```

举例2

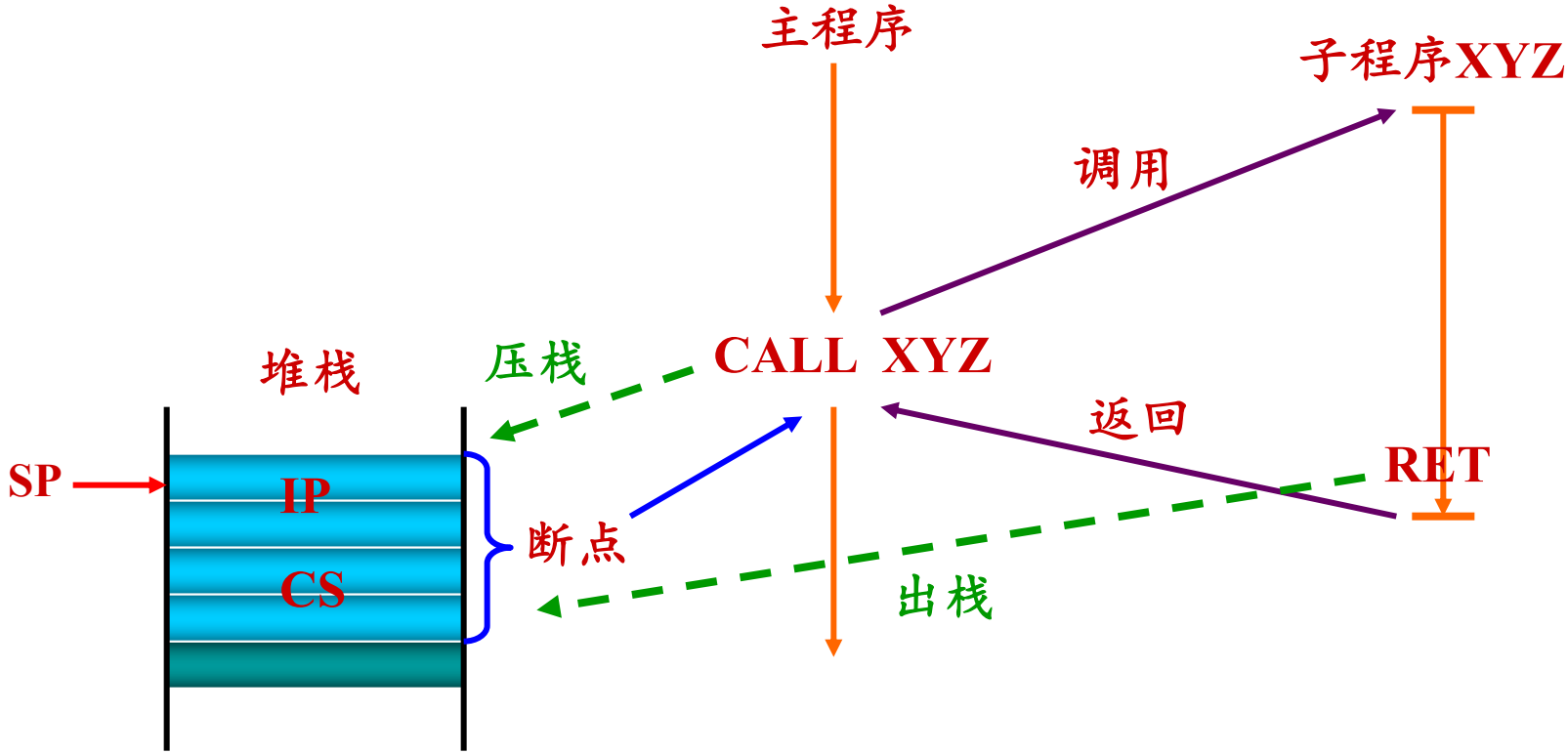
在首地址为ASCII_STR的L个字符的字符串中查找空格字符。

```
        MOV  CX,L
        MOV  SI,-1
        MOV  AL,20H     ；空格符的ASCII码
NEXT:INC SI
        CMP  AL,ASCII_STR[SI]；相等ZF置1
        LOOPNE  NEXT
        JNZ  NOT_FOUND
        ...
```

（4）子程序调用

（1）CALL　调用

（2）RET　返回



CALL指令

（1）段内直接调用
（2）段内间接调用
（3）段间直接调用
（4）段间间接调用

段内直接调用

指令格式：

CALL SUBROUTINE（子程序名）

执行操作：

EIP的内容入栈

(EIP)←(EIP) + 16位移量/32位移量

段内间接调用

指令格式：

CALL reg或WORD PTR [reg]

执行操作：

EIP的内容入栈

(EIP)←(EA)

其中有效地址EA在寄存器或存储单元中。

段间直接调用

指令格式：

CALL FAR PTR SUBROUTINE（子程序名）

（子程序名）

执行操作：

CS的内容入栈

EIP的内容入栈

(EIP)←SUBROUTINE指定的偏移地址

(CS)←SUBROUTINE指定的段地址

段间间接调用

指令格式：

CALL DWORD PTR [reg] 执行操作：

CS的内容入栈

EIP的内容入栈

(EIP)←(EA)

(CS)←(EA+2/4)

其中有效地址EA使用除立即寻址和寄存器寻址方式以外的任意一种寻址方式来指定。

RET

- **RET** 段内返回
- **RET n** 段内带立即数返回
- **RET** 段间返回
- **RET n** 段间带立即数返回

段内RET和RET n

RET 执行操作：

把从栈顶弹出的数送 EIP，即

(EIP) ← Pop ( )

RET n 执行操作：

把从栈顶弹出的数送 EIP，即

(EIP) ← Pop ( )

(ESP) ←(ESP) + n

n为非负偶数

段间RET和RET n

RET 执行操作：

把从栈顶弹出的数分别送 EIP 和CS，即

(EIP) ← Pop ( )

(CS) ← Pop ( )

RET n 执行操作：

把从栈顶弹出的数分别送 EIP 和CS，即

(EIP) ← Pop ( )

(CS) ← Pop ( )

(ESP) ←(ESP) + n

n为非负偶数

段内与段间RET的区别

- 段内RET与段间RET的机器码是不一样的
- 汇编程序会自动将属性为NEAR或FAR的过程（即子程序）中的RET指令汇编成段内RET或段间RET

（5）中断指令

- **INT n**　　中断指令
- **INTO**　　溢出中断
- **IRET**　　中断返回

中断的基本概念（1）

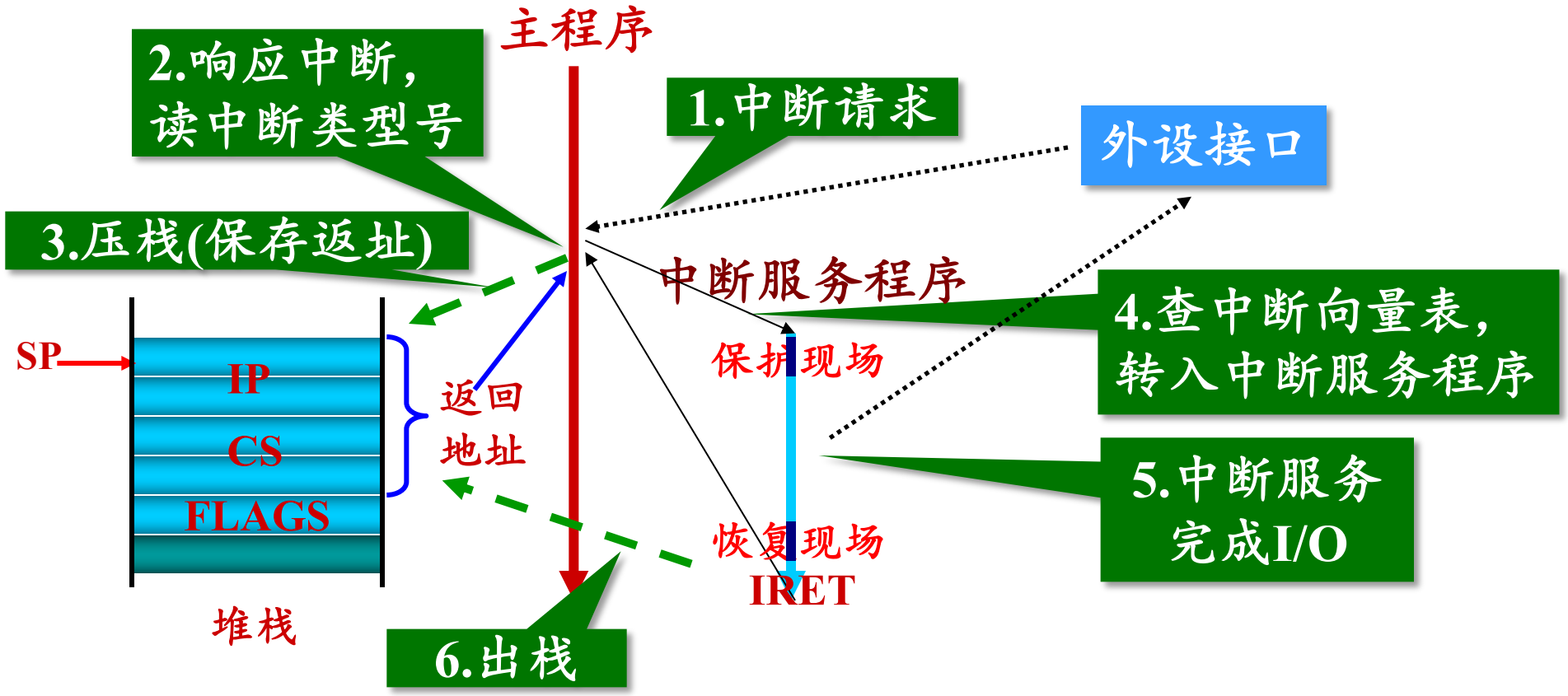
- 当硬件或外设备有意外情况发生（或遇到一条软中断指令）时，需要CPU暂停当前程序，转去执行一段特定程序（称为中断服务程序）来进行处理，这种情况叫做中断。

两类中断：

- 软中断
  - 是用软中断指令来激活的
- 硬中断
  - 中断的产生具有随机性，改变程序的执行顺序

中断的基本概念（2）

- 硬中断示意



中断向量表

在实模式下，存储器的前1024个字节（0~3FFH）为中断向量表区，可存放256个中断服务程序的入口。每个地址占4字节，低位字存放服务程序所在段的段内偏移，高位字存放服务程序所在段的段地址。

软中断指令

指令格式：

INT n

INTO

n为中断类型号，取值在 0～255 之间。溢出中断INTO指令的中断类型号隐含为 4（当OF=1时执行该指令产生溢出中断）

执行操作：

FLAGS入栈

IF, TF, AC ← 0

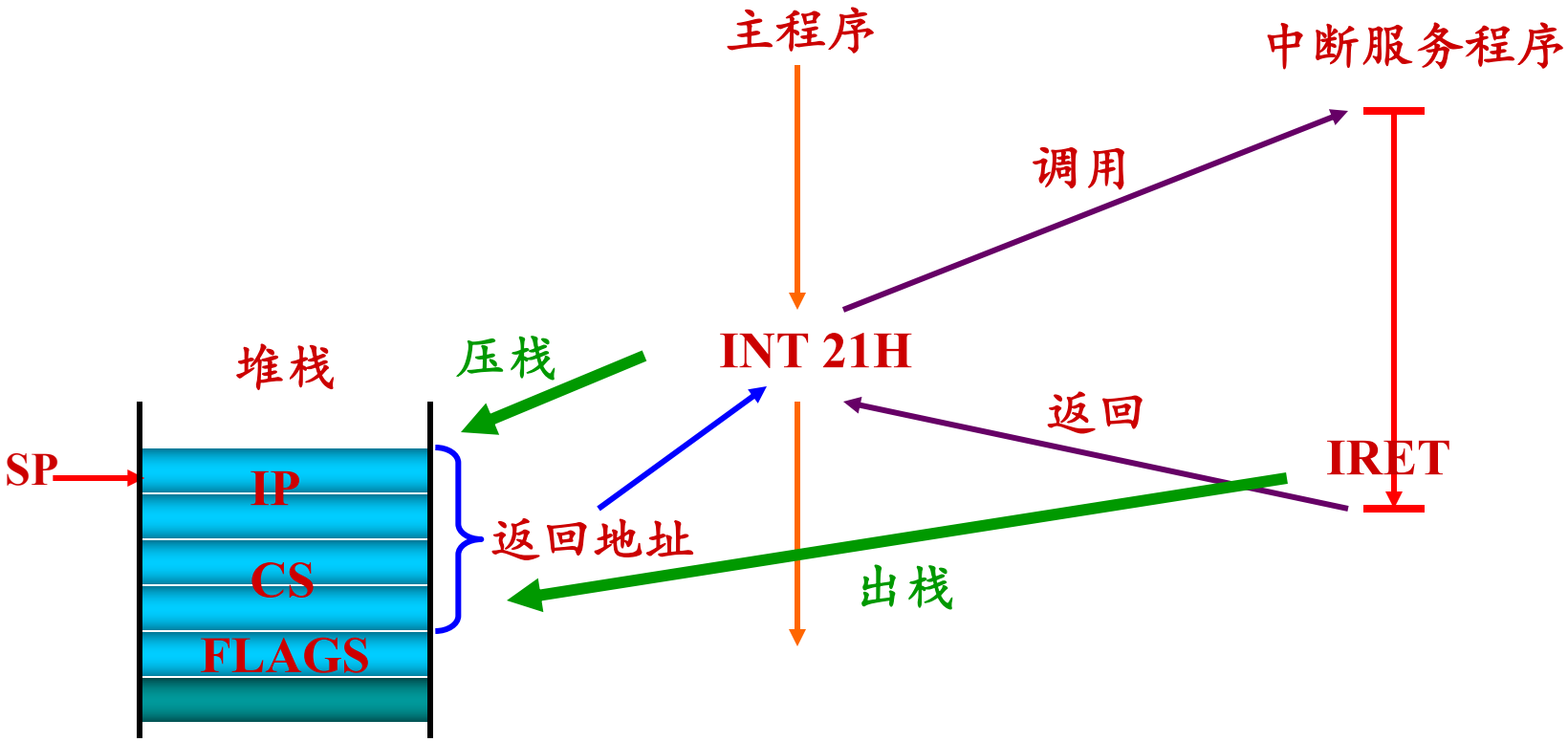
CS, IP入栈

(IP) ← (n * 4)

(CS) ← (n * 4 + 2)

INT n 指令

- 软中断示例



IRET和IRETD指令

指令格式：

IRETD

执行操作：

(EIP) ← Pop ( )

(CS) ← Pop ( )

(EFLAGS) ← Pop ( )

6. 处理机控制指令

（1）标志处理指令

（2）其他处理机控制指令

（1）标志处理指令

- CLC　清进位标志，CF←0
- CMC　进位标志取反，　CF←$\overline{CF}$
- STC　置进位标志，CF←1
- CLD　清方向标志，DF←0
- STD　置方向标志，DF←1
- CLI　清中断标志，　IF←0
- STI　置中断标志，　IF←1

（2）其他处理机控制指令

- NOP：无操作，用于预占存储单元或软件延时
- HLT：停机指令，等待中断，中断返回后继续执行下一条指令。
- WAIT：等待指令，利用WAIT指令和测试（TEST）引脚实现与8087同步运行。
- ESC op, reg/mem：交权指令，用于指示协处理器执行指令OP。在486之后，成为未定义指令。
- LOCK： 是指令前缀，与其它指令配合使用，产生总线封锁信号，直至指令执行完毕。
- BOUND reg, mem；检查reg 中的数是否在上下界之内指令（286+）
- ENTER imm16, imm8：建立堆栈帧指令（286+）
- LEAVE：释放堆栈帧指令（286+），等价于下面2条指令：

  mov esp, ebp、pop ebp

本章小结

- 指令基本概念，ISA、CISC、RISC
- 指令格式，操作码、操作数
- 寻址方式
  - 与数据有关的寻址方式，重点：前7种寻找方式
  - 与转移有关的寻址方式，段内寻址、段间寻址
- 指令系统
  - 数据传送指令
  - 算术/逻辑运算指令
  - 串处理指令
  - 控制转移指令
  - 处理机控制指令