



2.6 控制转移类指令



2.6 控制转移类指令

- ◇ 控制转移类指令通过改变IP（和CS）值来改变程序的执行顺序，从而实现分支、循环、过程调用等程序结构。
- ◇ 重点掌握
 - ◆ JMP/Jcc
 - ◆ LOOP/JCXZ
 - ◆ CALL/RET INT n/IRET

1 目标指令地址的寻址方式

◇ 相对寻址方式

- ◆ 指令代码中提供目的地址相对于当前IP的位移量，转移的目的地址就是当前IP值加上位移量。

◇ 直接寻址方式

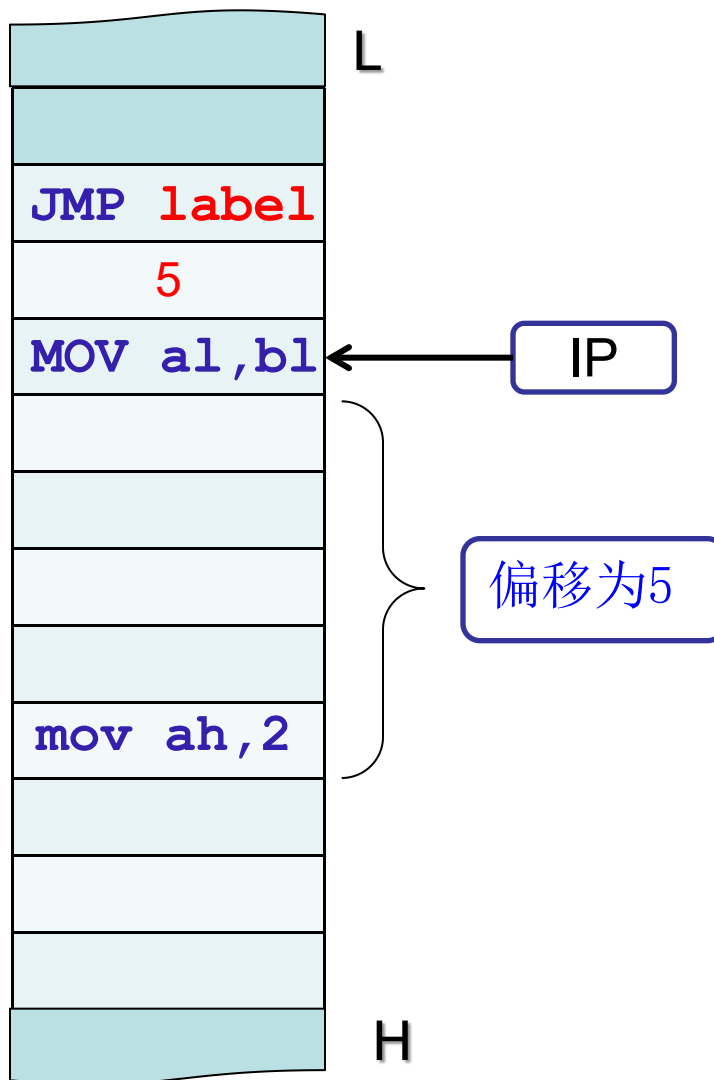
- ◆ 转移指令中提供目标指令的逻辑地址(CS:IP)。

◇ 间接寻址方式

- ◆ 从转移指令代码中指示的寄存器或存储单元中获取目标指令的地址。

相对寻址方式图示

:
:
JMP label
MOV al,b1
:
:
Label: mov ah,2
:
:



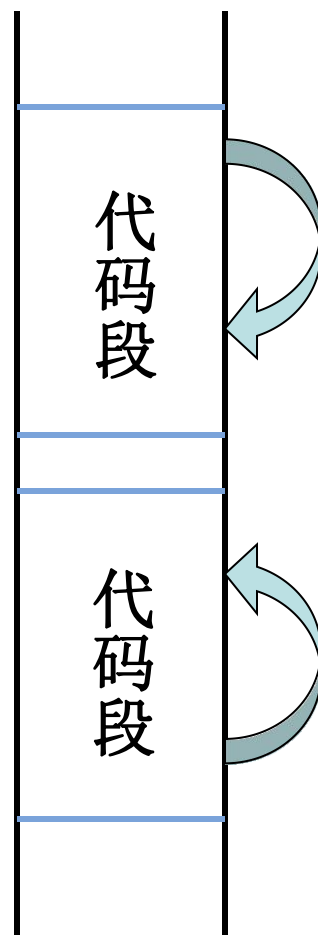
2 目标地址的寻址范围：段内寻址

◇ 段内转移——近转移 (near)

- ◆ 在当前代码段(最大**64KB**)范围内转移，且目标指令的段内偏移地址与当前**IP**的偏移在 $\pm 32\text{KB}$ 的范围内
- ◆ 转移时不需要更改**CS**，只要改变**IP**

◇ 段内转移——短转移 (short)

- ◆ 在当前代码段(最大**64KB**)范围内转移，且目标指令的段内偏移地址与当前**IP**的偏移在 $-128 \sim +127$ 的范围内



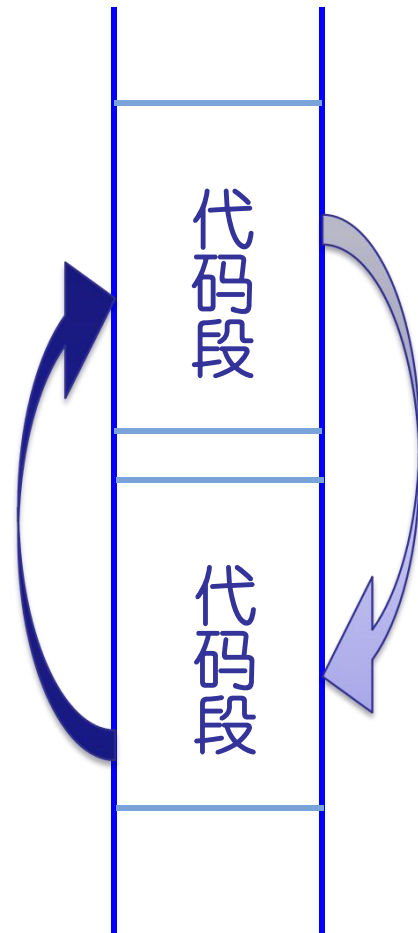
2 目标地址的寻址范围：段间寻址

◇ 段间转移——远转移（far）

- ◆ 远转移用于实现从当前代码段跳转到另一个代码段，可以支持在**1MB**范围内实现转移。
- ◆ **CS**和**IP**的值都需要修改。
- ◆ 目标地址是以**段地址:段内偏移地址**形式提供的**32位远指针**，即目标指令的逻辑地址。

实际编程时，汇编程序会根据目标地址的属性，自动处理成短转移、近转移或远转移

程序员可用操作符 **short ptr**、**near ptr** 或 **far ptr** 强制成为需要的转移类型

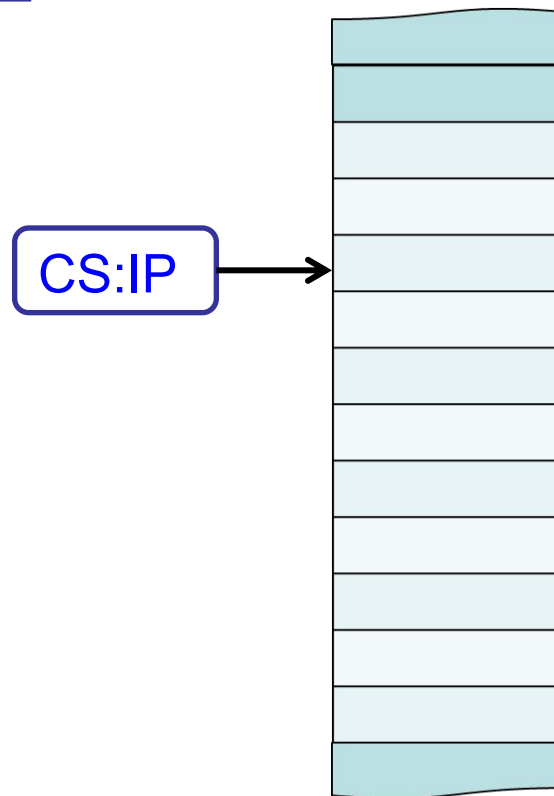


2.6.1 无条件转移指令

JMP label ; 程序的执行转向label标识的目标指令

◇ JMP指令的机内执行情况包括4种类型：

- (1) 段内转移、相对寻址
- (2) 段内转移、间接寻址
- (3) 段间转移、直接寻址
- (4) 段间转移、间接寻址



2.6.1 无条件转移指令JMP (jump)

JMP label ;段内转移、相对寻址

;IP←IP + 位移量

演示

JMP r16/m16 ;段内转移、间接寻址

;IP←r16/m16

演示

演示

JMP far ptr label ;段间转移、直接寻址

演示

;IP←偏移地址,CS←段地址

JMP far ptr mem ;段间转移, 间接寻址

演示

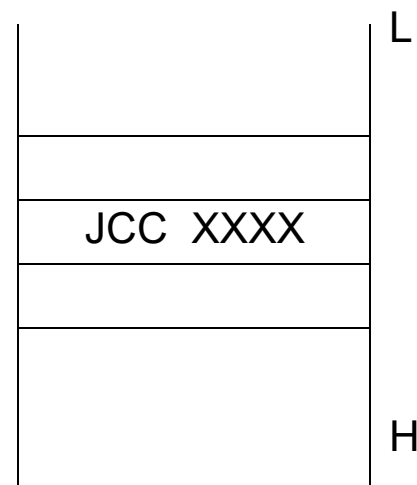
;IP←[mem],CS←[mem + 2]

2.6.2 条件转移指令

- ◇ 条件转移指令 **Jcc** 根据指定的条件确定程序是否发生转移。其通用格式为：

Jcc label ; 条件满足, 发生转移
; $IP \leftarrow IP + 8\text{位位移量}$
; 否则, 顺序执行

- ◇ **label** 只支持短转移的相对寻址方式



2.6.2 条件转移指令

条件转移指令的分类

- ◇ **Jcc**指令共有**16**条，但为了方便记忆和使用却有**30**个助记符（[表2-4](#)）
- ◇ **Jcc**指令不影响标志，但要利用标志，根据利用的标志位不同，分成三种情况：
 - (1) 根据单个标志位状态决定是否转移
 - (2) 根据无符号数的高低关系决定是否转移
 - (3) 根据有符号数的大小关系决定是否转移

2.6.2 条件转移指令

1. 判断单个标志位状态的条件转移指令

(1) JZ/JE和JNZ/JNE

利用零标志ZF，判断结果是否为零（或相等）

(2) JS和JNS

利用符号标志SF，判断结果是正是负

(3) JO和JNO

利用溢出标志OF，判断结果是否产生溢出

(4) JP/JPE和JNP/JPO

利用奇偶标志PF，判断结果中“1”的个数是偶是奇

(5) JC/JB/JNAE和JNC/JNB/JAE

利用进位标志CF，判断结果是否进位或借位

2.6.2 条件转移指令

例题2.22 判AX中的无符号数是奇数还是偶数，若是偶数直接除以2，若是奇数则加1后除以2。

◇ 问题：如何判断AX中的数据是奇数还是偶数？

解答：判断AX最低位是“0”（偶数），还是“1”（奇数）。

- 1.用逻辑与指令将除最低位外的其他位清0，保留最低位不变，如果结果是0，AX中是偶数；否则，是奇数；
- 2.先将AX最低位用移位指令移至进位标志，再判断进位标志，是0，则AX中是偶数；否则，是奇数；
- 3.将最低位用移位指令移至最高位（符号位），判断符号标志，是0，AX中是偶数；否则，为奇数。

例题2.22解答1：用JZ指令实现

test ax,01h

；测试AX的最低位D0（不用AND指令，以免改变AX）

jz even

；标志ZF = 1，即D0 = 0：AX内是偶数，程序转移

add ax,1

；标志ZF = 0，即D0 = 1：AX内的奇数，加1

even: shr ax,1 ; AX ← AX ÷ 2

用右移一位的方法实现除以2。

本例中用RCR指令比SHR指令更好。考虑边界值，如原操作数是0FFFFH的情况。

例题2.22解答2：用JNC指令实现

mov bx,ax

shr bx,1

还可使用SAR、ROR和RCR指令

；将AX的最低位D0移进CF

jnc even

；标志CF = 0，即D0 = 0：AX内是偶数，程序转移

add ax,1

；标志CF = 1，即D0 = 1：AX内的奇数，加1

even: rcr ax,1 ；AX ← AX ÷ 2

例题2.22解答3：用JNS指令实现

mov bx,ax

ror bx,1

错误！循环移位指令不影响SF等标志

；将AX的最低位D0移进最高位（符号位SF）

jns even

；标志SF = 0，即D0 = 0：AX内是偶数，程序转移

add ax,1

；标志SF = 1，即D0 = 1：AX内的奇数，加1

even: rcr ax,1 ；AX ← AX ÷ 2

ADD BX,0 ；增加一条指令

例2.23 判断寄存器AL中的字符是否为字母Y(y)

； 寄存器AL中的字符是字母Y（含大小写），则令AH=0，否则令AH=-1

	cmp al,'y'	； 比较AL与小写字母y
	je next	； 相等，转移
	cmp al,'Y'	； 不相等，继续比较AL与大写字母Y
	je next	； 相等，转移
	mov ah,-1	； 不相等，令AH=-1
	jmp done	； 无条件转移指令
next:	mov ah,0	； 相等的处理：令AH=0
done:	

2.24 偶校验

；为DL寄存器中的数据生成偶校验位，存入CF标志位

```
test dl,0ffh
```

；使CF = 0，同时设置PF标志

```
jpe done
```

；DL中“1”的个数为偶数

；正好CF = 0，转向done

```
stc
```

；DL中“1”的个数为奇数，设置CF = 1

```
done: ..... ; 完成
```

2.6.2 条件转移指令

2. 比较无符号数高低

- ◇ 无符号数的大小用高 (Above)、低 (Below) 表示，需要利用CF确定高低、利用ZF标志确定相等 (Equal)
- ◇ 两数的高低分成4种关系，对应4条指令

JB (JNAE) : 目的操作数低于 (不高于等于) 源操作数

JNB (JAE) : 目的操作数不低于 (高于等于) 源操作数

JBE (JNA) : 目的操作数低于等于 (不高于) 源操作数

JNBE (JA) : 目的操作数不低于等于 (高于) 源操作数

2.6.2 条件转移指令

3. 比较有符号数大小

- ◇ 判断有符号数的大（Greater）、小（Less），需要组合 OF、SF 标志、并利用 ZF 标志确定相等与否
- ◇ 两数的大小分成4种关系，分别对应4条指令

JL (JNGE)：目的操作数小于（不大于等于）源操作数

JNL (JGE)：目的操作数不小于（大于等于）源操作数

JLE (JNG)：目的操作数小于等于（不大于）源操作数

JNLE (JG)：目的操作数不小于等于（大于）源操作数

例2.25 求两数中的较大值

cmp ax,bx ; 比较AX和BX

jae next ; 若 $AX \geq BX$, 转移

xchg ax,bx ; 若 $AX < BX$, 交换

next: mov wmax,ax

如果AX和BX存放的是有符号数,

则条件转移指令应采用JGE指令

else

mov wmax,bx

例2.25 求较大值（另解）

`cmp ax,bx` ; 比较AX和BX

`jae next`

`mov wmax, bx`

; 若 $AX < BX$, $wmax \leftarrow BX$

`jmp done`

`next: mov wmax,ax`

; 若 $AX \geq BX$, $wmax \leftarrow AX$

`done:`

If $ax > bx$

`mov wmax,ax`

else

`mov wmax,bx`

例2.25 求较大值（另解对比）

`cmp ax,bx` ; 比较AX和BX

`jbe next`

`mov wmax, ax`


; 若 $AX > BX$, $wmax \leftarrow AX$

`jmp done`

`next: mov wmax,bx`

; 若 $AX \leq BX$, $wmax \leftarrow BX$

`done:`



学习了无条件转移和条件转移指令之后，就可以学习分支结构程序设计了。

2.6.3 循环指令

- ◇ 一段代码序列多次重复执行就构成循环。
- ◇ 8088设计有利用CX计数的计数循环指令。

LOOP label ; 循环指令

用法

——; 首先 $CX \leftarrow CX - 1$; 然后判断; 若 $CX \neq 0$, 转移

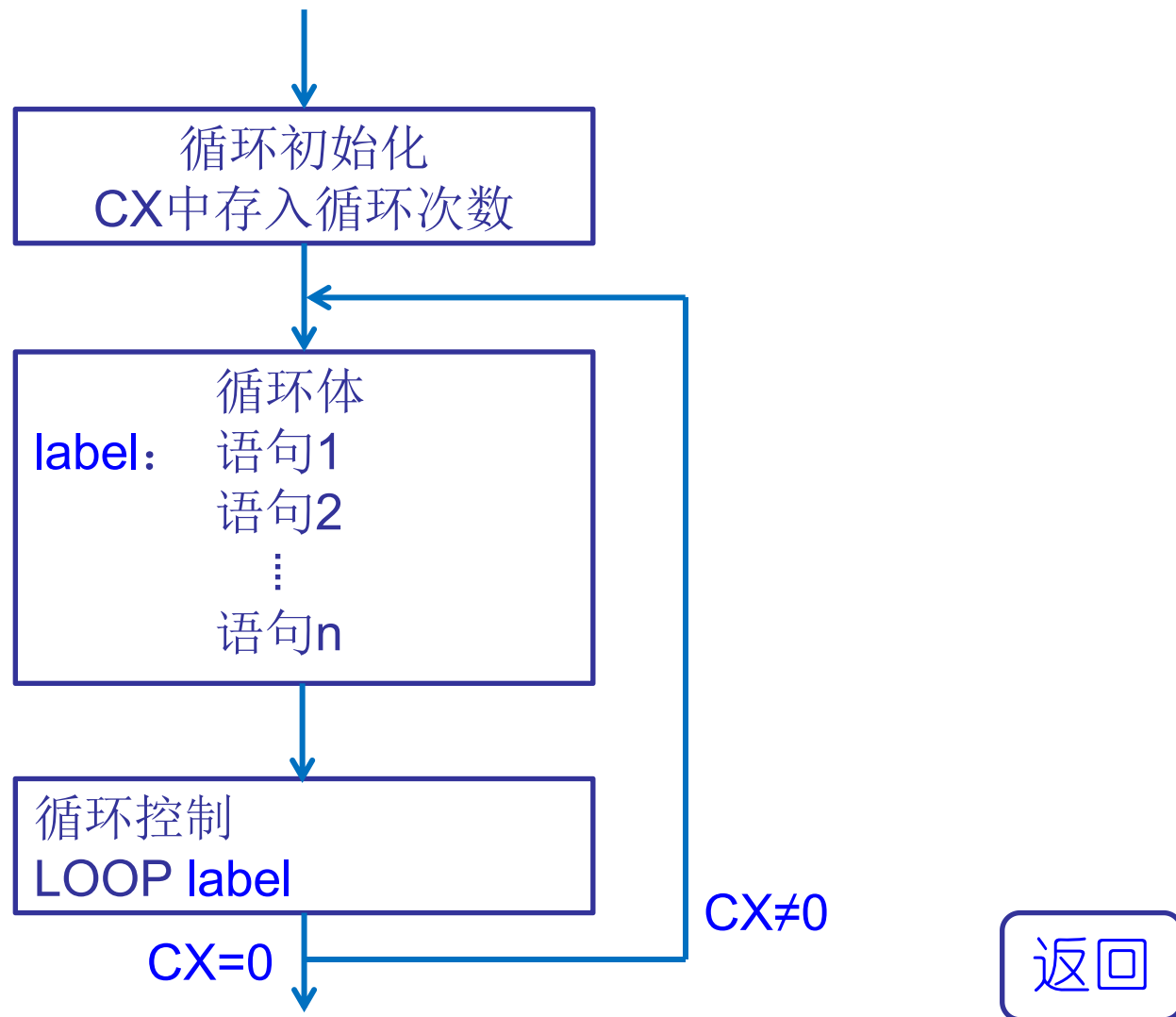
JCXZ label ; 为0循环指令

; 如果 $CX = 0$, 则转移

- ◇ label操作数采用相对短寻址方式
- ◇ 还有**LOOPZ/LOOPE**和**LOOPNZ/LOOPNE**两条指令

dec cx
jnz label

LOOP指令用法



(3) 循环控制指令 (3条)

指令格式

①循环

示例

LOOP label ; (CX) $\neq 0$ 循环, (CX) = 0 则顺序执行

②为零/相等循环 (比较)

LOOPZ/LOOPE label ; 当ZF=1且 (CX) $\neq 0$ 则循环,
当ZF=0或 (CX) = 0 退出循环

③不为零/不相等循环 (搜索)

LOOPNZ/LOOPNE label ; 当ZF=0且 (CX) $\neq 0$ 循环,
当ZF=1或 (CX) = 0 退出循环

LOOPZ

◇ 比较两个串是否相等

Str1 : "ABCDEFGFGHIJK"

Str2 : "ABCDEFGSHIJK"

(CX)=11

返回

LOOPNZ

◇ 到串中搜索是否存在某个字符

例：搜索串str1中是否存在字符H

Str1 : "ABCDEFGHIJK"

(CX)=11

返回



注意：

- ①使用循环控制指令之前，必须在寄存器**CX**中预置循环次数的初值。
- ②执行循环控制指令时，将完成 $(IP) \leftarrow (IP) + 8$ 位移量的操作。
- ③循环控制指令不影响状态标志位。
- ④循环控制指令主要用于数据块比较、查找关键字等操作。

例2.26 数据块传送（字节）

将数据段sbuf缓冲区的1KB数据送到dbuf缓冲区

```
mov cx,400h
```

； 设置循环次数：1K = 1024 = 400H

```
mov si,offset sbuf
```

； 设置循环初值：SI指向数据段源缓冲区开始

```
mov di,offset dbuf
```

； DI指向附加段目的缓冲区开始（附加段）

again:

```
mov al,[si]
```

； 循环体：实现数据传送

```
mov [di],al
```

； 每次传送一个字节

```
inc si
```

； SI和DI指向下一个单元

```
inc di
```

```
loop again
```

； 循环条件判定：循环次数减1，不为0转移（循环）

图示

WJ0226

例2.26 数据块传送（字）

`mov cx,200h`

； 设置循环次数： $1K \div 2 = 200H$

`mov si,offset sbuf`

； 设置循环初值：**SI**指向数据段源缓冲区开始

`mov di,offset dbuf`

； **DI**指向附加段目的缓冲区开始（附加段）

`again: mov ax,[si]` ； 循环体：实现数据传送

`mov es:[di],ax` ； 每次传送一个字

`add si,2` ； 指向下一个（字）单元

`add di,2`

`loop again`

； 循环条件判定：循环次数减1，不为0转移（循环）



接下来 -----

循环结构程序设计

2.6.4 子程序调用和返回指令

- ◇ 子程序是完成特定功能的一段程序。
- ◇ 当某个程序（主程序）需要执行这个功能时，采用**CALL**调用指令转移到该子程序的起始处执行。
- ◇ 当运行完子程序功能后，采用**RET**返回指令回到主程序继续执行。

CALL与RET过程演示

- 转移指令有去无回；
- 子程序调用需要返回，其中利用堆栈保存返回地址。

1. 子程序调用指令CALL

◆ CALL指令分成4种类型（类似JMP）

CALL label ; 段内调用、直接寻址

CALL r16/m16 ; 段内调用、间接寻址

CALL far ptr label ; 段间调用、直接寻址

CALL far ptr mem ; 段间调用、间接寻址

◆ CALL指令需要保存返回地址

■ 段内调用——入栈偏移地址IP

$SP \leftarrow SP - 2, SS:[SP] \leftarrow IP$

■ 段间调用——入栈偏移地址IP和段地址CS

$SP \leftarrow SP - 2, SS:[SP] \leftarrow CS$

$SP \leftarrow SP - 2, SS:[SP] \leftarrow IP$

2. 子程序返回指令RET

- ◆ 根据段内和段间、有无参数，分成4种类型

RET ; 无参数段内返回

RET i16 ; 有参数段内返回

RET ; 无参数段间返回

RET i16 ; 有参数段间返回

- ◆ 需要弹出CALL指令压入堆栈的返回地址

- 段内返回——出栈偏移地址IP

$IP \leftarrow SS:[SP], \quad SP \leftarrow SP + 2$

- 段间返回——出栈偏移地址IP和段地址CS

$IP \leftarrow SS:[SP], \quad SP \leftarrow SP + 2$

$CS \leftarrow SS:[SP], \quad SP \leftarrow SP + 2$

例2.27 十六进制数转换为ASCII码的子程序

；子程序：将DL低4位的一位16进制数转换成ASCII码

```
htoasc    proc
            and dl,0fh          ; 只取DL的低4位
            or   dl,30h         ; DL高4位变成3
            cmp  dl,39h         ; 是0~9, 还是0Ah~0Fh ?
            jbe  htoend         ; 是0~9, 转移
            add  dl,7           ; 是0Ah~0Fh, 加上7
htoend:    ret                  ; 子程序返回
htoasc     endp
```

转换原理

；主程序调用子程序
mov dl,28h
call htoasc

2.6.5 中断指令和系统功能调用

- ◆ 中断 (Interrupt) 是另一种改变程序执行顺序的方法。
- ◆ 8088CPU支持256个中断，每个中断有一个8位的编号称为中断类型码。
- ◆ 中断指令有3条：

INT i8

IRET

INTO

中断

第2章：1. 中断指令

INT i8

- ； 中断调用指令：产生i8号中断
- ； 主程序使用，其中i8表示中断向量号

IRET

- ； 中断返回指令：实现中断返回
- ； 中断服务程序使用

2.7 处理器控制类指令

- ◇ 处理器控制类指令用来控制CPU的状态，使CPU暂停、等待或执行空操作等

NOP ; 空操作指令

HLT ; HLT执行后，使机器暂停工作，CPU处于停机状态，以等待一次外部中断到来，中断结束后，程序继续执行，CPU继续工作。

- ◇ 还有其他指令：

LOCK ; 封锁总线

ESC ; 与浮点协处理器有关的交权指令

WAIT ; 等待指令

LOCK

lock是一个指令前缀，Intel的手册上对其的解释是：

Causes the processor's LOCK# signal to be asserted during execution of the accompanying instruction (turns the instruction into an atomic instruction). In a multiprocessor environment, the LOCK# signal insures that the processor has exclusive use of any shared memory while the signal is asserted.

也就是说lock会使紧跟在其后面的指令变成 **atomic instruction**。暂时的锁一下总线，指令执行完了，总线就解锁了

返回

ESC

- ◇ ESC，即Escape，若CPU收到该指令，则协处理器接管系统控制权并执行下一条机器代码，即包括三角运算、指数、对数运算等复杂的数学运算。

返回

WAIT

- ◇ 当CPU执行WAIT指令时，他将在每个时钟周期对TEST*引脚进行测试：如果无效，则程序踏步并继续测试；如果有效，则程序恢复运行
- ◇ 也就是说， WAIT指令使CPU产生等待，直到TEST*引脚有效为止
- ◇ 在使用协处理器8087时，通过该引脚和WAIT指令，可使8086与8087的操作保持同步

返回

第2章：总结

- ◇ 本章学习了 8088 CPU 的常用指令
- ◇ 希望大家就如下几个方面进行一下总结
 - ◆ 操作数寻址方式
 - ◆ 转移指令目的地址的寻址方式
 - ◆ 指令支持的操作数形式
 - ◆ 常用指令的助记符和功能
 - ◆ 指令对标志的影响
- ◇ 通过复习整理，形成指令系统的整体概念，进而掌握常用指令

8088CPU内部操作演示

第2章：作业

习题2（第56 ~ 59页）：

2.3, 2.5, 2.8, 2.11, 2.12, 2.14, 2.17, 2.20

(1) ~ (5)



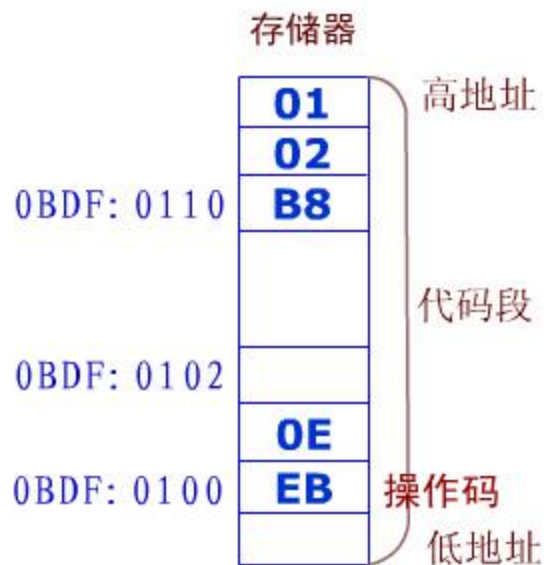
本节到此结束
谢谢！

目的地址相对寻址方式

段内相对转移 JMP 指令

IP:0100H

JMP 110



播放



停止

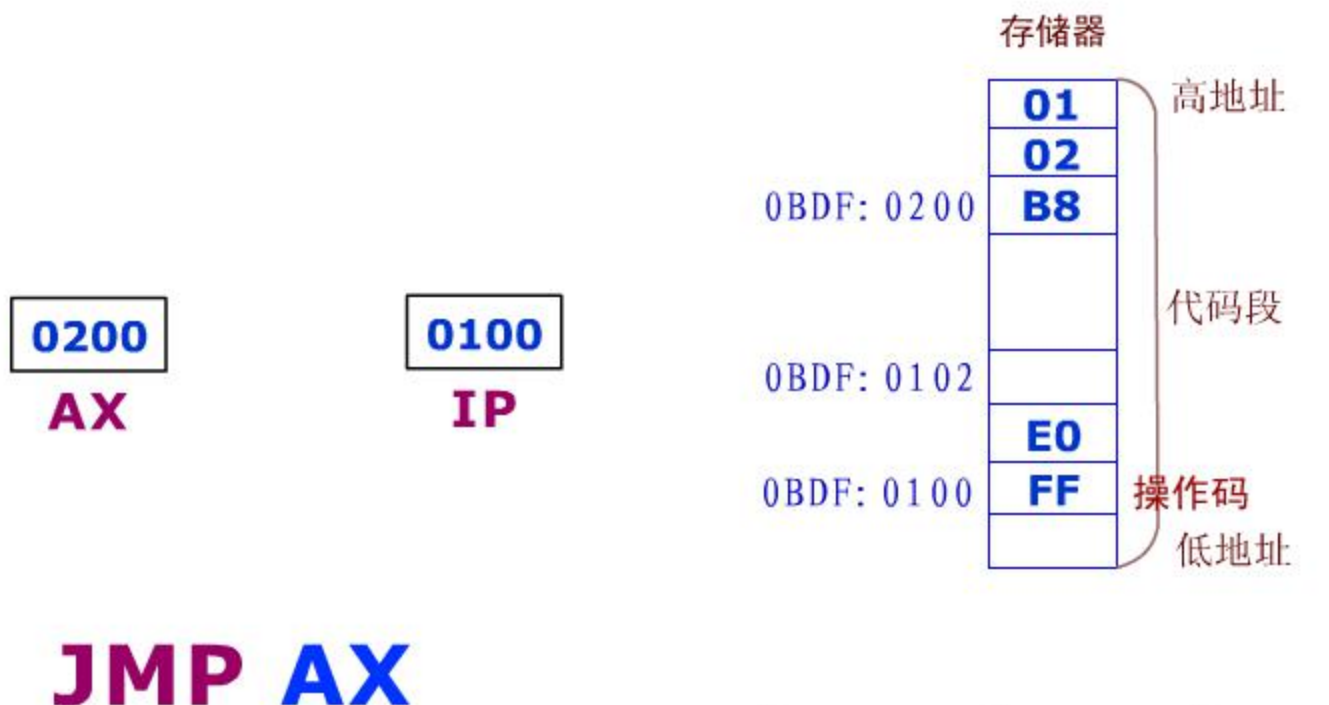


返回



目的地址寄存器段内间接寻址方式

段内寄存器间接转移JMP指令



播放



停止



返回



目的地址存储器段内间接寻址方式

段内存储器间接转移JMP指令

IP 0100

JMP WORD PTR[2000]

1492: 2000

02
00

数据段

0BDF: 0200

E0
FF

代码段

0BDF: 0100

20
00
26
FF



播放



停止



返回

02:53 PM
12



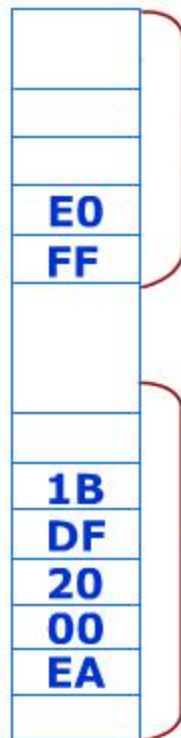
目的地址段间直接寻址方式

段间直接转移JMP指令

CS
0BDF

IP
0100

1BDF: 2000



代码段2

代码段1

0BDF: 0100

JMP 1BDF:2000



播放



停止



返回



目的地址存储器段间间接寻址方式

段间间接转移JMP指令

CS
0BDF

IP
0100

1492: 2000

1BDF: 0200

0BDF: 0100



JMP FAR PTR[2000]

播放 停止 返回

条件转移指令的含义

JZ/JE	Jump on Zero/Equal
JNZ/JNE	Jump on Not Zero/Equal
JS	Jump on Sign
JNS	Jump on Not Sign
JP/JPE	Jump on Parity/Parity Even
JNP/JPO	Jump on Not Parity/Parity Odd
JO	Jump on Overflow
JNO	Jump on Not Overflow
JC/JB/JNAE	Jump on Carry/Below/Not Above or Equal
JNC/JNB/JAE	Jump on Not Carry/Not Below/Above or Equal
JBE/JNA	Jump on Below or Equal/Not Above
JNBE/JA	Jump on Not Below or Equal/Above
JL/JNGE	Jump on Less/Not Greater or Equal
JNL/JGE	Jump on Not Less/Greater or Equal
JLE/JNG	Jump on Less or Equal/Not Greater
JNLE/JG	Jump on Not Less or Equal/Greater

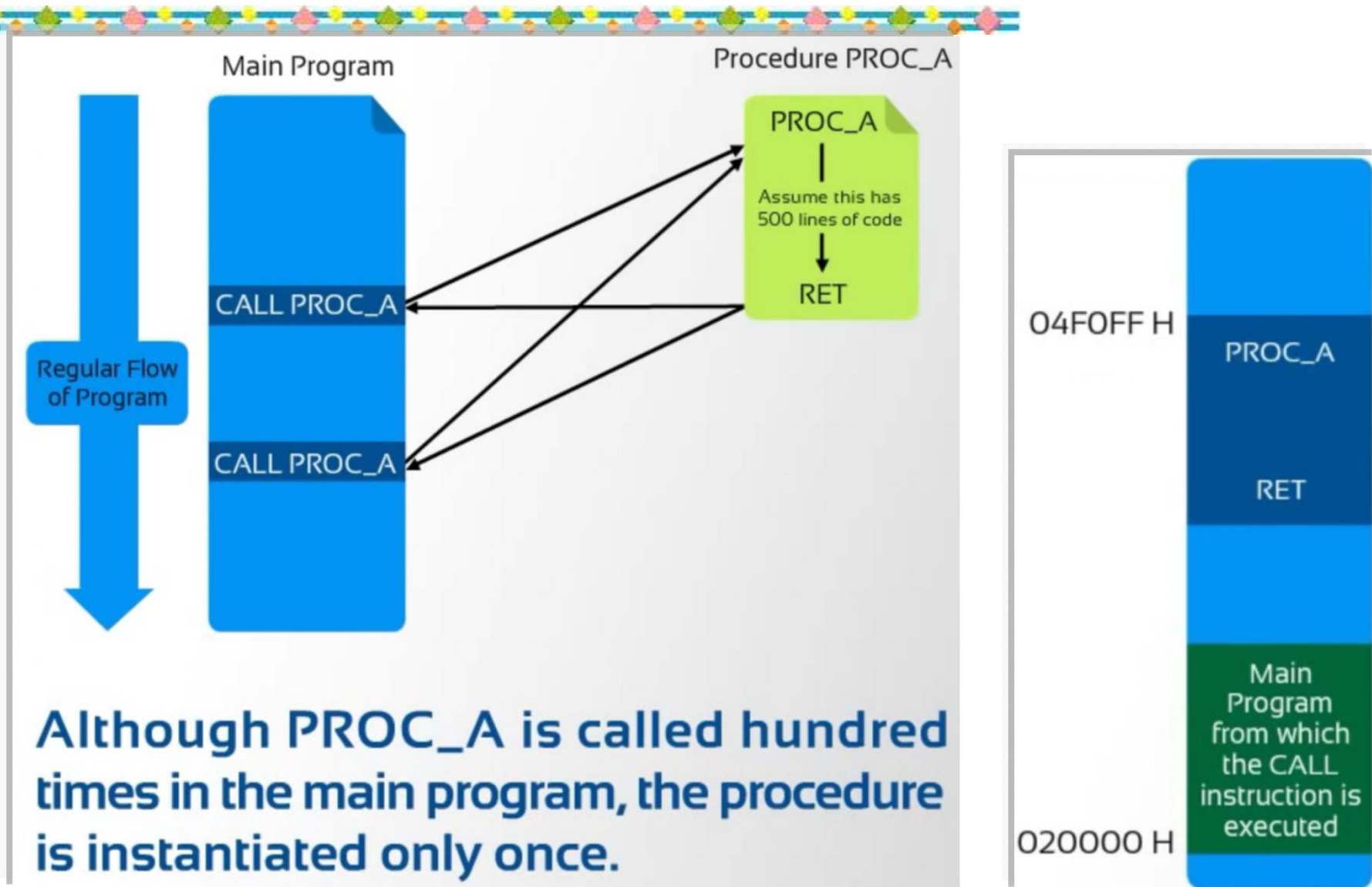


奇偶校验

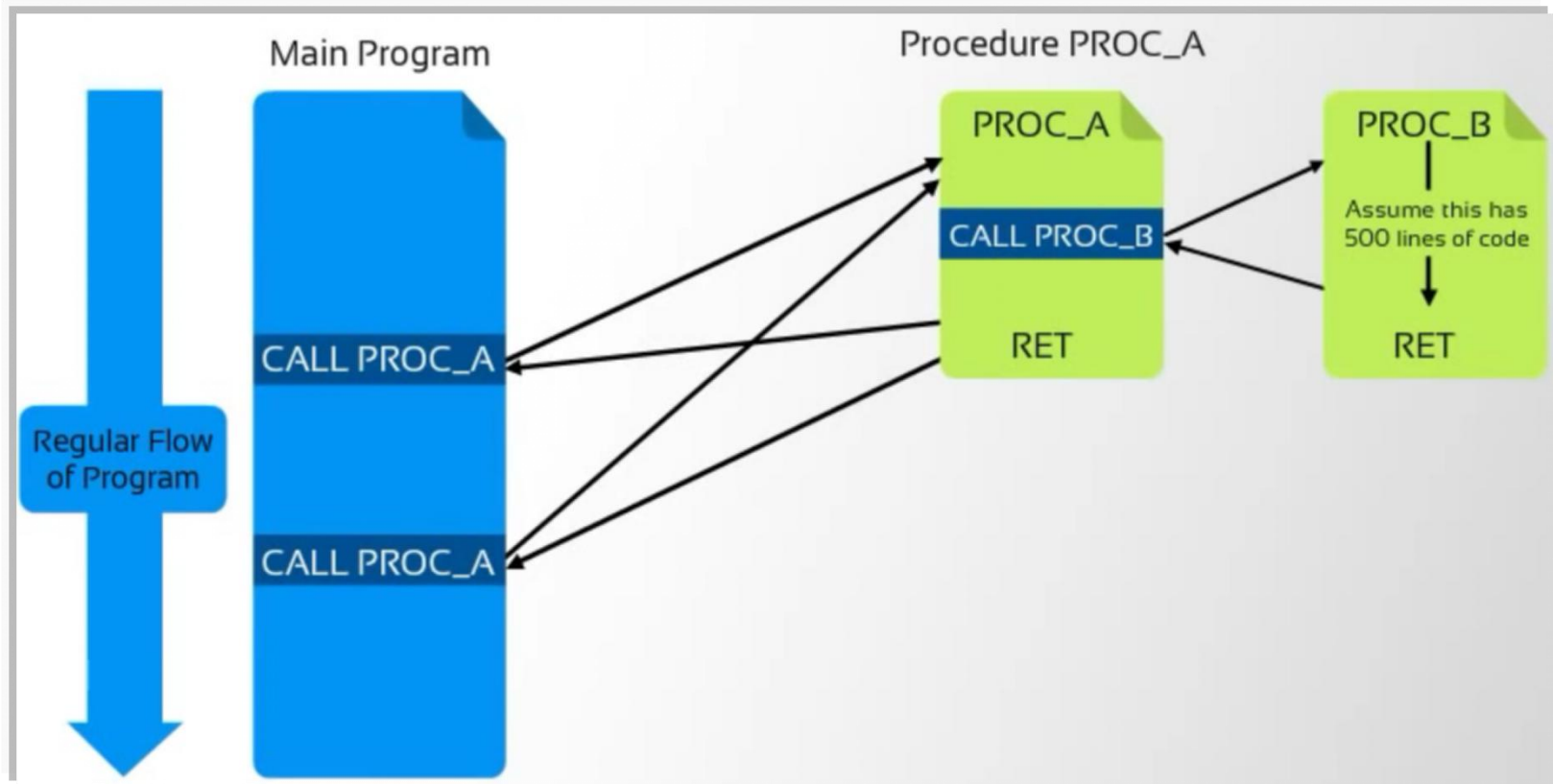
- ◇奇偶校验是计算机中最常使用的校验方法，因为不论用硬件还是用软件都很容易实现
- ◇偶校验：如果数据中“1”的个数不是偶数，则校验位是1，使得包括校验位在内的数据中“1”的个数为偶数；否则，校验位为0
- ◇奇校验：如果数据中“1”的个数不是奇数，则校验位是1，使得包括校验位在内的数据中“1”的个数为奇数；否则，校验位为0



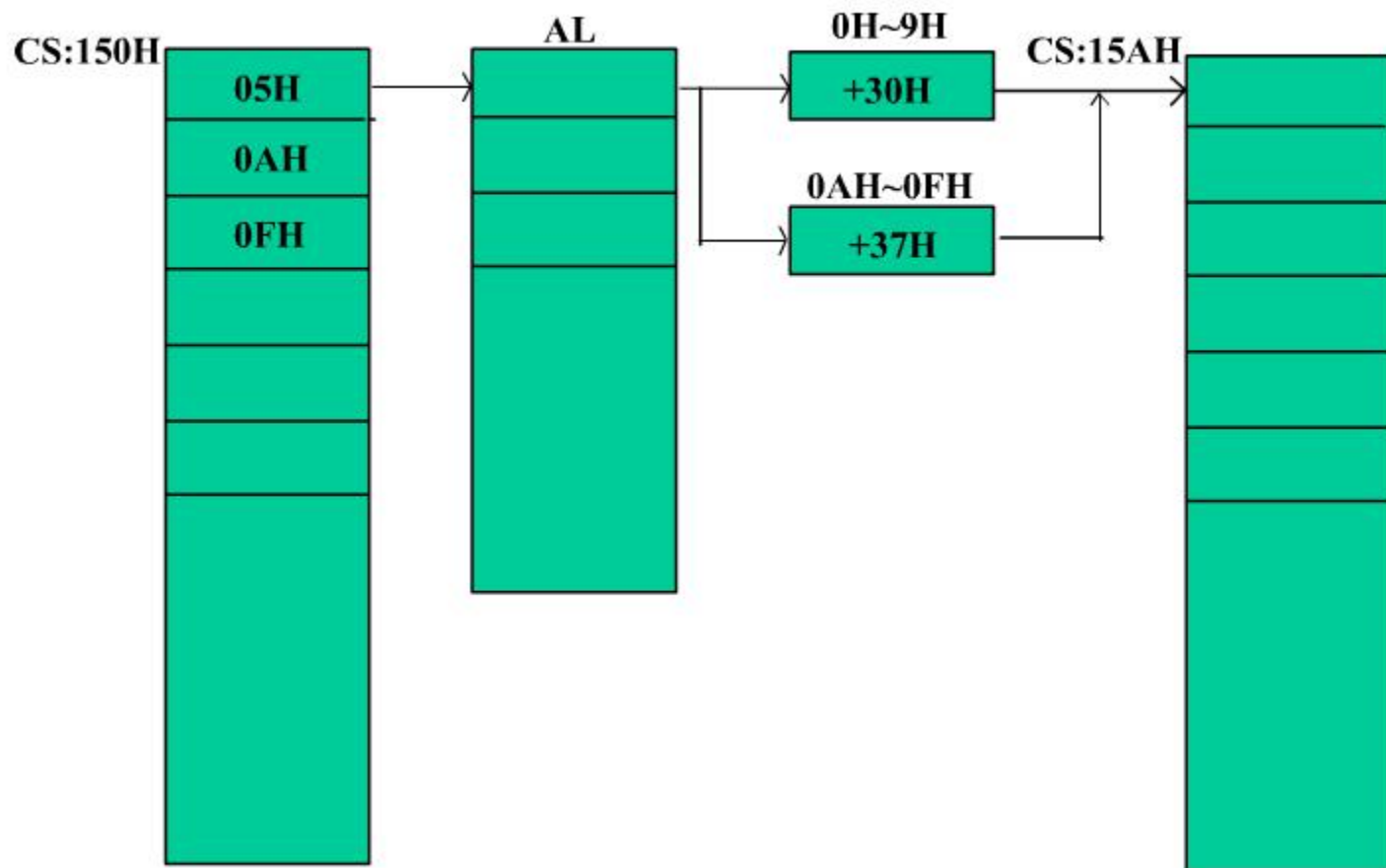
子程序调用与返回



子程序嵌套调用



十六进制数转换为ASCII码的原理



主程序与中断服务程序

主程序

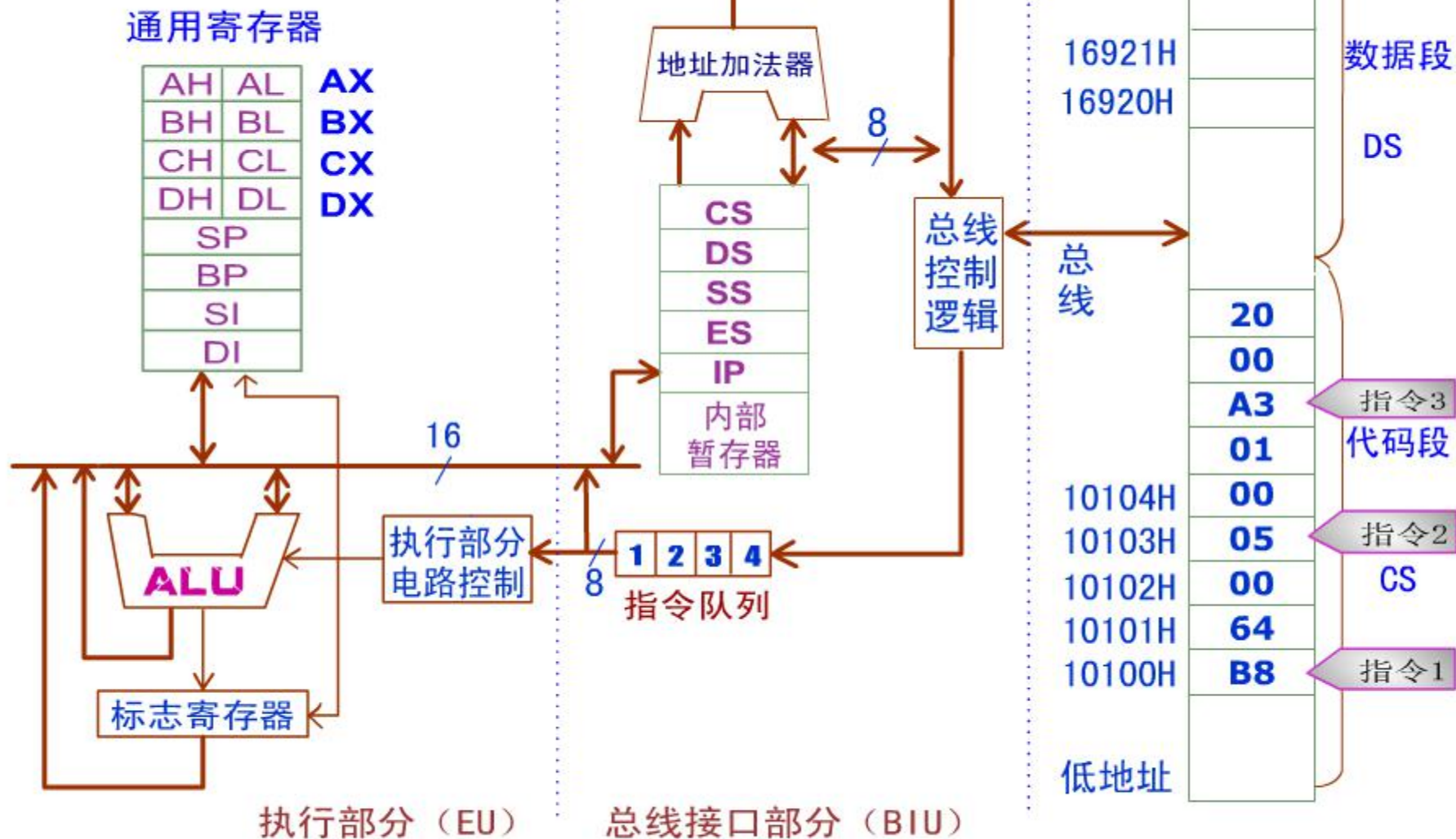
中断服务程序

中断请求

断点

IRET

中断请求可以来自处理器外部的中断源，
也可以由处理器执行指令引起：
例如执行 **INT i8** 指令。




播放



停止

8088的指令执行



- 
- ◇ 没有溢出($OF=0$)时, 若 $SF=0$, 则 $a>b$;
若 $SF=1$, 则 $a<b$;
 - ◇ 产生溢出($OF=1$)时, 若 $SF=0$, 则 $a<b$;
若 $SF=1$, 则 $a>b$;



