1. **预备知识**

*主要内容：*

* ***计算机系统概述***
* ***数据存储方式***
* ***数据表示***
* ***存储器编址***
* ***8086寄存器组***
* ***什么是汇编语言***
* ***汇编语言程序举例***

**§1.1 计算机系统概述**

1. **硬件（Hardware）**

**(1)中央处理单元CPU**

**控制器、运算器、寄存器**

**(2)存储器**

**主存储器：RAM和ROM**

**辅助存储器：磁盘、光盘、U盘**

**(3)外部设备**

**输入设备和输出设备**

**※程序员能看到的硬件:**

* **中央处理单元 CPU（Intel 80x86）**

**对汇编语言程序员，最关心其中的寄存器**

* **存储器（主存储器）**

**呈现给汇编语言程序员的，是存储器地址**

* **外部设备（接口电路）**

**汇编语言程序员看到的是端口（I/O地址）**

**※寄存器：**

**（1）寄存器是CPU内部的高速存储单元**

**（2）它们为处理器提供各种操作所需要的数据或地址等信息**

**（3）汇编语言程序采用它们各自的符号名**

**16位Intel 8086/80286 CPU中有**

**AX BX CX DX**

**SI DI BP SP**

**32位80386/80486/Pentium系列 CPU中有**

**EAX EBX ECX EDX**

**ESI EDI EBP ESP**

**※ 存储器地址：**

**（1）存储器是由大量存储单元组成，需要用编号区别**

**每个单元：编号＝地址**

**（2）存储器地址是存储器中存储单元的编号**

**（3）每个存储单元存放一个字节量的数据**

**一个字节B（Byte）＝8个二进制位b（bit）**

**（4）采用十六进制数来表达地址**

**Intel 8086具有1兆字节（1MB）存储器容量**

**存储器地址表示为：00000H ～ FFFFFH**

**其中大写H（或小写h）表示是十六进制数**

**※端口：**

**（1）I/O接口电路由接口寄存器组成，需要用编号区别各个寄存器：编号＝地址**

**（2）I/O地址是接口电路中寄存器的编号**

**（3）端口是I/O地址的通俗说法**

**（4）系统通过这些端口与外设进行通信**

**（5）采用十六进制数来表达端口**

**Intel 8086支持64K个8位端口**

**I/O地址可以表示为：0000H ～ FFFFH**

1. **软件（Sofware）**
2. **系统软件**
3. **应用软件**
4. **附图**

I/O

接

口

主存储器

系统总线

CPU

寄存器

控制器

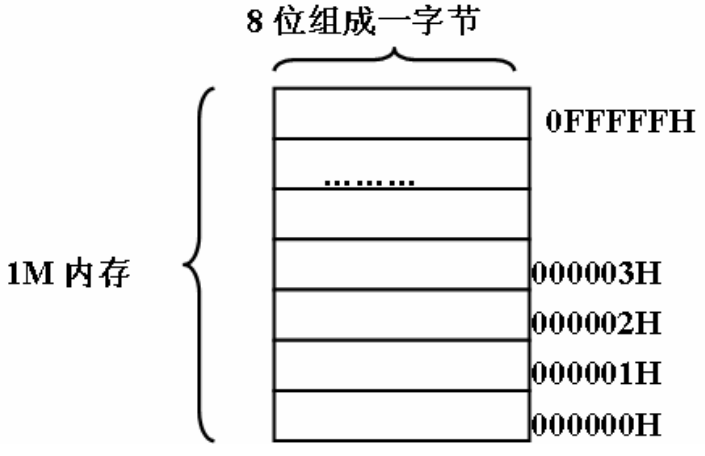
运算器

辅助存储器

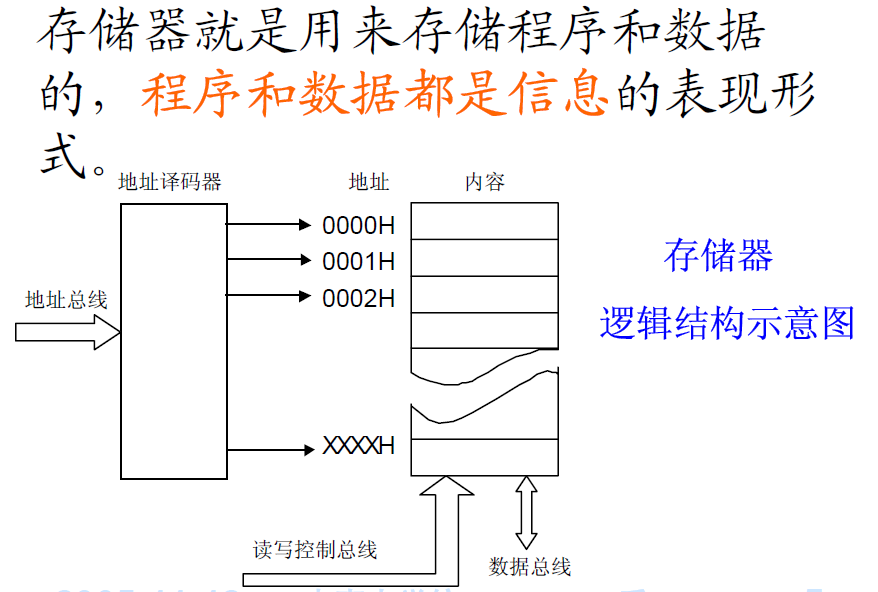
输入设备

输出设备

**(1)**

**(2)** ****

§1.2 数据存储方式

1.

2.

（1）字：由2个字节组成，字地址由两字节中地址较小的一个即低字节的地址决定。

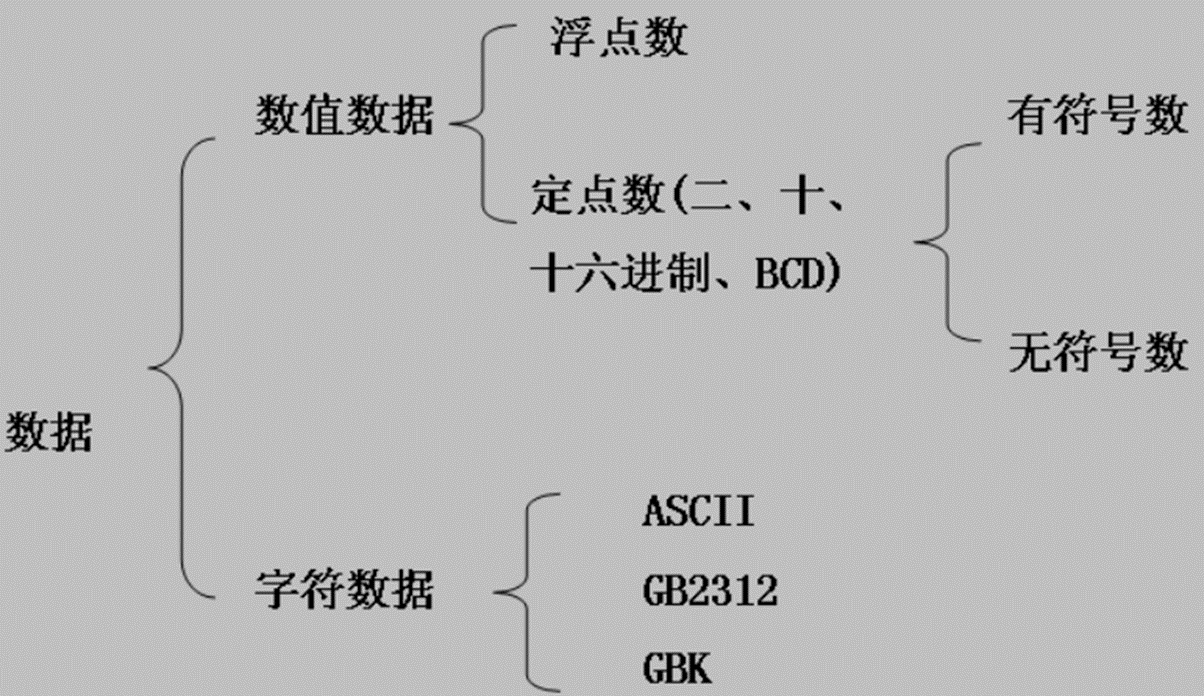
存放的方式是低8位存放低字节，高8位存放高字节。

（2）双字：双字的地址也由四个字节中的最低地址确定.



※字数据存入计算机主存，依据特点：高-高，低-低（占连续字节）

§1.3 数据表示



§1.4 存储器编址

1.4.1 存储器的编址

1.8086 CPU有20条地址线

(1)最大寻址空间为**220＝1MB**

(2)物理地址范围从**00000H～FFFFFH**

CPU是通过地址总线来指定存储器单元的。地址总线上能传送多少个不同的信息，CPU就可以对多少个存储单元进行寻址。

一个CPU有N根地址线，则可以说这个CPU的总线宽度为N。这样的CPU最多可以寻找2的N次方个内存单元。

2.8086 将1MB空间分成许多逻辑段

(1)8086对逻辑段要求：

①段地址低4位均为0

②每段最大不超过64KB

(2)8086对逻辑段并不要求：  
 ①必须是64KB

②各段之间完全分开（即可以重叠）

(3)1MB分成的逻辑段的个数

①最多：

每隔16个存储单元就可以开始一个段

所以most为：**220÷16＝216＝64K 个段**

②最少：

每隔64个存储单元开始一个段

所以least为：**220÷216＝16 个段**

3.一个存储单元除具有唯一的物理地址外，还具有多个逻辑地址。

（1）物理地址：内存单元在整个内存空间中的唯一的20位地址**00000H～FFFFFH**

(2)逻辑地址： 段基地址：段内偏移地址

①段地址：说明逻辑段在主存中的起始位置

8086规定段地址必须是模16地址xxx0H

省略低4位0000B，段地址就可以用16位数据表示，就能用16位段寄存器表达段地址

“段地址”包含段的概念（为什么出现段的概念在寄存器中会有介绍），这种说法会使人 误以为内存被划分成了一个一个的段，每个段有一个段地址。

但其实，内存并没有分段，段的划分来自于CPU。

以后，在编程时可以根据需要，将若干地址连续的内存单元看作一个段。

②偏移地址：说明主存单元距离段起始位置的偏移量

(每段不超过64KB，偏移地址也可以用16位数据表示)

1.4.2地址变换

1.公式：物理地址=段基地址×16（基础地址）+偏移地址

2.本质含义：CPU在访问内存时，用一个基础地址和一个相对于基础地址的偏移地址相加，给出内存单元的物理地址。

§1.5 8086寄存器组

1.5.1 8086微处理器

1. 微处理器是微机的硬件核心，主要包含指令执行的运算和控制部件，还有多种寄存器

2. 对程序员来说，微处理器抽象为以名称存取的寄存器

3. 8086可访问主存空间为1MB，20条地址线，16条数据线。

4. 内部结构有两个功能模块，完成一条指令的取指和执行功能

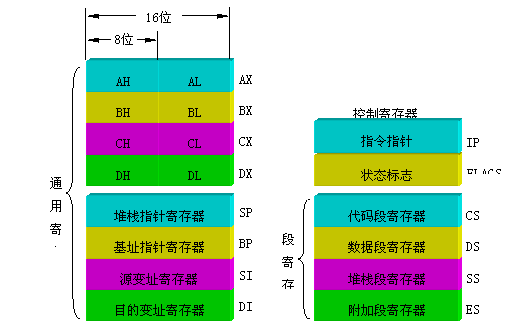
5. 总线接口单元BIU：负责读取指令和操作数

6. 执行单元EU ：负责指令译码和执行

7. 8086的内部结构图（见书P15）

1.5.2 8086寄存器组

1.



8088/8086内部有14个16位的寄存器。按其功能可分为三

大类：通用寄存器（8个），段寄存器（4个）和控制器寄

存器（2个）

2.分类

(1)通用寄存器

**①**8086的16位通用寄存器是：

AX BX CX DX

SI DI BP SP

其中前4个数据寄存器都还可以分成高8位和低8位两个独立的寄存器

②8086的8位通用寄存器是：

AH BH CH DH

AL BL CL DL

对其中某8位的操作，并不影响另外对应8位的数据

③分类

I．数据寄存器

是通用的，用来存放计算的结果和操作数，但每

个寄存器又有它们各自的专用目的，并因此而得

名。

a．AX－－累加器，使用频度最高，用于算术、逻辑运算以及与外设传送信息等；

b．BX－－基址寄存器，常用做存放存储器地址；

c．CX－－计数器，作为循环和串操作等指令中的隐含计数器；

d．DX－－数据寄存器，常用来存放双字长数据的高16位，或存放外设端口地址。

II．变址及指针寄存器

i．变址寄存器：常用于存储器寻址时提供地址

a．SI:源变址寄存器

b．DI:目的变址寄存器

※一般与DS联用，确定数据段中某一存储单元地址。

※在串指令中，SI与DS联用、DI与ES联用，分别寻址数据段和附加段：同时，在串指令中，SI和DI还具有自动增量或减量的功能。

ii．指针寄存器：用于寻址内存堆栈内数据

a．BP:基址指针（表示栈顶的偏移地址）

b．SP:堆栈指针 (表示堆栈段中的基地址)

※SP和BP寄存器均可与SS段寄存器联用，已确定堆栈段中的存储单元地址。

※堆栈是主存中一个特殊的区域，采用先进后出或后进先出操作方式，而不是随机存取操作方式。

※堆栈通常由处理器自动维持，由堆栈段寄存器SS和堆栈指针寄存器SP共同指示。

※SP不应该再用于其他目的，实际上可归类为专用寄存器。

（2）段寄存器

①“段”概念的出现

一个程序中有可以执行的指令代码，还有指令操

作的各类数据等。遵循模块化程序设计思想，希望将相关代码安排在一起，相关数据安排在一起。

②“段”的作用

一个段安排一类代码和数据。

(程序员在编写程序时，可以很自然地把程序的各部分放在相应的段中)

③ 段的分类

对应用程序来说，主要涉及三类段：

代码段：存放程序中指令代码

数据段：存放当前运行程序所用数据

堆栈段：指明程序使用的堆栈区域

④段寄存器

为了表明段在主存中的位置（或者说“提供内存单元的段地址”）

⑤段寄存器的分类

与段的分类类似

代码段寄存器CS

堆栈段寄存器SS

数据段寄存器DS

附加段寄存器ES

※附加段也是用于存放数据的数据段，专为处理数据串设计的串操作指令必须使用附加段作为其目的操作数的存放区域

⑶ 指令指针寄存器

①指令指针寄存器IP，指示代码段中指令的偏移地址

②它与代码段寄存器CS联用，确定下一条指令的物理地址

③计算机通过CS : IP寄存器来控制指令序列的执行流程

④IP寄存器是一个专用寄存器

标志寄存器

(4)CPU内部的寄存器中，有一种特殊的寄存器（对于

不同的处理机，个数和结构都有可能不同），它具有

以下三种作用：

用来存储相关指令的某些执行结果

用来为CPU执行相关指令提供行为依据

用来控制CPU的相关工作方式

在8086CPU中，这种特殊的寄存器被称为标志寄存

器。





寻址方式

第二章

指令的组成： [标号]: 指令助记符 [目的操作数],[源操作数];[注释]

①立即数寻址方式

指令中的操作数直接存放在机器代码中，紧跟在操作

码之后。这条指令汇编成代码后，操作码作为指令的一

部分存放在操作码之后的主存单元中。

②寄存器寻址方式

操作数存放在CPU内部寄存器reg中，不需要访问存储器，还可以存放在4个段寄存器中。

③存储器寻址方式（主存寻址方式）

由于8086的存储器是分段管理的，所以这里给出的地

址只是偏移地址（称为有效地址EA），而段地址在默认的或段超越前缀指定的段寄存器中。

在汇编语言中，用方括号表示存储单元的内容。

位移量采用补码表示，也可用符号表示(在iii、v中)。

i．直接寻址方式

直接包含了有效地址。

ii．寄存器间接寻址方式

有效地址存放在基址寄存器BX、BP或变址寄存器SI、DI中。其默认的段地址在DS段寄存器中，但可使用段超越前缀改变。

iii．寄存器相对寻址方式

有效地址是寄存器内容与有效号8位或16位位移量之和，寄存器可以是BX、BP或SI、DI。

操作数的有效地址应该是：

有效地址=BX/BP/SI/DI+8/16位位移量

iv．基址变址寻址方式

把一个基址寄存器的内容加上变址寄存器的内容构成有效地址EA。

EA=BX/BP+SI/DI

v．相对基址变址寻址方式

有效地址EA=BX/BP+SI/DI+8/16位位移量

第三章 指令系统

3.1 指令与指令系统

3.2 IA-32指令系统

3.3 本章小结

**指令与指令系统**

1. 概念
2. 指令：控制计算机完成某种操作的命令
3. 指令系统：处理器所能识别的所有指令的集合

**零操作数指令： 操作码**

**单操作数指令： 操作码 操作数**

**双操作数指令： 操作码 操作数，操作数**

**多操作数指令： 三操作数及以上**

3.

指令长度

4.指令的执行速度：

二、性质

1. 指令的兼容性：同一系列机的指令都是兼容的

2. 指令的字长影响指令的执行速度

3. 对不同的操作数，指令执行的时间不同：

存储器→立即数→寄存器（从左至右，速度逐渐增加）

三、

1.8086指令系统为16位

**IA-32指令系统**

总括

1. 分类

数据传送

算术运算

逻辑运算和移位

串操作

程序控制

处理器控制

依功能分为六个部分

Inter 8086 指令系统共有117条指令

二、 指令的一般格式

标号：指令助记符 目的操作数，源操作数 ；注释

1. 标号：该指令在主存中的逻辑地址，可有可无
2. 指令助记符：代表一种指令
3. 目的操作数和源操作数：参与操作的对象
4. 注释：对该指令或这段程序功能的说明，是为了程序便于阅读而加上的

分讲

第一类：数据传送类指令

1. 总括
2. 地位：数据传送是计算机中最基本、最重要的一种操作
3. 功能：把数据从一个位置传送到另一个位置
4. 数量：8086有14种数据传送指令
5. 执行对象：执行寄存器和寄存器间、主存与寄存器间、 AL/AX由于外设施口间的字与字节的多种传送操作
6. 特点：除标志寄存器传送指令外，均不影响标志位
7. 分类：

通用数据传送

输入输出

地址传送

标志位操作

二、分讲

**一般数据传送指令**

**堆栈操作指令**

**交换指令**

**查表转换指令**

**字位扩展指令**

**（一）通用数据传送**

1. 一般数据传送指令

⑴格式：mov dest,src

⑵功能：把1字节或字的操作数从源地址src传送至目的地址dest

⑶对象：

立即数→通用寄存器/存储单元

寄存器→寄存器/存储单元

存储单元→寄存器

段寄存器→通用寄存器/存储单元

⑷注意：

①两操作数的字长必须相同

②两操作数不允许同时为存储器操作数；两操作数不允许同时为段寄存器；在源操作数是立即数时，目标操作数不能是段寄存器；

③IP和CS不作为目标操作数，FLAGS一般也不做为操作数在指令中出现

! 虽然存在通用寄存器和存储单元向CS段寄存器传送数据的指令，却不允许执行，因为这样直接改变CS值将引起程序执行混乱

2、堆栈操作指令

⑴概念：堆栈是一个“先进后出”的主存区域，位于堆栈段中，使用SS段寄存器记录其段地址。堆栈只有一个出口，即当前栈顶。栈顶是地址较小的一端(低端)，它用堆栈指针寄存器SP指定。

⑵分类：有两种基本操作，对应于两条基本指令：进栈指令和出栈指令

⑶对象：堆栈操作的对象只能是字操作数。

⑷进栈指令

i．格式：push r16/m16/seg

ii．执行过程：先使堆栈指针SP减2，然后把一个字操作数存入堆栈顶部。进栈时，低字节存放于低地址SP，高字节存放于高地址SP-1

⑸出栈指令

i．格式：pop r16/m16/seg

ii．执行过程：把栈顶的一个字传送至指定的目的操作数，然后堆栈指针SP+2

⑹应用

i．可用来临时存放数据，以便随时恢复他们

ii．常用于在子程序间传递参数。因为在子程序中，通常要保存被修改的寄存器内容，以便在返回时恢复它们。

POP指令的顺序与PUSH指令相反，且成对出现，只有这样才能使各寄存器恢复原来内容

⑺注意

i．操作数不能是立即数

ii．不允许从栈顶弹出一个子给CS，但允许PUSH CS

iii．PUSH指令的操作方向是从高地址向低地址，而POP指令的操作正好相反。

3、交换指令

⑴格式:XCHG REG，MEM/REG

⑵对象：操作数可以是字也可以是字节

可以在通用寄存器与通用寄存器或存储器之间对换数据

⑶注意：

①两操作数必须有一个是寄存器操作数

②不能使用段寄存器！

4、查表（换码）指令

⑴格式: ①XLAT ;（AL）← (（BX）+（AL））

②XLAT src\_table； （src\_table表示要查找的表的首地址）

⑵操作：将BX+AL所指单元的内容送AL

⑶实质：首先在主存中建立1字节表格，表格的内容是要转换成目的代码，表格的首地址存放于BX寄存器中，需要转换的代码存放于AL寄存器，要求被转换的代码应是相对表格首地址的位移量。设置好后，执行换码指令，即将AL寄存器的内容转换为目标代码。

(4)注意：

①AL的内容实际上是距离表格首地址的位移量，只有8位，所以表格的最大长度为256，超过256的表格需要采用修改BX和AL的方法才能转换。

②XLAT指令默认使用BX和AL寄存器。

隐含寻址方式：采用默认操作数的方法

采用该种方法的指令有：XLAT，算术运算指令标志位操作等

5.字位扩展指令

（1）作用：①将有符号数的符号位扩展到高位

②无符号数的扩展为在高位补0

(2) 分类：①字节到字的扩展指令

i．格式：CBW

ii.操作：将AL内容扩展到AX内容中

iii．规则：若最高位=1，则执行后AH=FFH

若最高位=0，则执行后AH=00H

②字到双字的扩展指令

i．格式：CBD

ii．操作：将AX内容扩展到DX :AX，高位存放在DX中，隐含了操作数AX。

iii．规则：若最高位=1，则执行后DX=FFFFH

若最高位=0，则执行后DX=0000H

(3)注意：①这是一个含有隐含操作数方式的指令

②这是一个零操作数指令

**（二）输入输出**

1. 格式：（1）输入指令： IN acc，PORT

（2）输出指令 ：OUT PORT，acc

PORT:端口地址

1. 端口地址表现形式的不同：

根据端口地址码的长度，指令具有两种不同的端口地址表现形式。

（1）直接地址

端口地址为8位时，指令中直接给出8位端口地址（立即数方

式）；

寻址256个端口。

(2) 间接地址

端口地址为16位时，指令中的端口地址必须由DX指定；

可寻址64K个端口。

有效地址传送指令

指针传送指令

LEA指令

\*LDS指令

\*LES指令

**（三）地址传送**

1. LEA取偏移地址指令
2. 格式:LEA REG，MEM
3. 操作：将变量的16位偏移地址取出送目标寄存器
4. 作用：当程序中用符号地址表示内存偏移地址时，须使用该

指令。

(4) 注意：源操作数MEM必须是一个存储器操作数，目标操作数必须是16位通用寄存器，一般是间址寄存器（BX，BP，SI，DI）。

2.LDS与LES

(1)格式：LDS reg16，mem32

（reg16） ((mem32)+1:(mem32))

(DS/ES) ((mem32)+3:(mem32)+2)

LES reg16，mem32

(2)作用：把存储器mem32中存放的一个32位远地址指针（包括

偏移地址）送到reg16和DS/ES。4个存储单元的前两个单元的内容作为偏移地址送reg16，后两个单元的内容作为段地址装入段寄存器DS/ES

(3)注意：源操作数mem32位存储器操作数，给出的是内存中4个连续的存储单元的首地址。目标操作数reg16必须是BX、BP、SI、DI等4个间址寄存器之一

LAHF

SAHF

PUSHF

POPF



隐含操作数为FLAGS

隐含操作数为AH

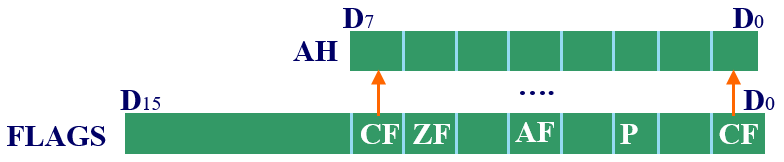
**（四）标志位操作**

1. LAHF SAHF
2. LAHF:

操作: 将FLAGS的低8位的SF，ZF，AF，PF，和CF装入AH对应位置，8位中其余3位为无效位

(2) SAHF

操作：执行与LAHF相反的操作



1. PUSHF POPF

针对FLAGS的堆栈操作指令,将标志寄存器压栈或从堆栈弹出

1. PUSHF

[SP-1] ← FLAGSH）

[SP-2] ←（FLAGSL）

（SP）←（SP）-2

注意：指令本身不影响标志位

1. POPF

（FLAGSL）← [SP]

（FLAGSH）← [SP+1]

（SP）←（SP）+2

注意：指令影响标志位

第二类 : 算术运算类指令

 加、减、乘、除

普通加法指令ADD

带进位的加法指令ADC

加1指令INC

**（一）加法指令**

注意！加法指令对操作数的要求与MOV指令相同

1. ADD
2. 格式：ADD OPRD1，OPRD2
3. 操作：OPRD1+OPRD2→OPRD1
4. 注意：① ADD指令的执行对全部6个状态标志位都产生影响

② 源操作数和目标操作数不能同时为存储器操作数；

不能对段寄存器进行运算。

1. ADC
2. 操作: OPRD1+OPRD2+CF →OPRD1
3. 作用！：ADC指令多用于多字节数相加，使用前要先将CF清零
4. 注意：指令格式、对操作数的要求、对标志位的影响与ADD指

令完全一样

1. INC
2. 格式: INC OPRD
3. 操作: OPRD+1→OPRD
4. 作用：常用于在程序中修改地址指针
5. 注意： ① INC指令不影响***CF***标志位，但对其它5个状态标志

AF、OF、PF、SF及ZF会产生影响。

②！OPRD不能是段寄存器或立即数



普通减法指令SUB

考虑借位的减法指令SBB

减1指令DEC

比较指令CMP

求补指令NEG

**（二）减法指令**

减法指令对操作数的要求与对应的加法指令相同

1. SUB
2. 格式：SUB OPRD1，OPRD2
3. 操作：OPRD1- OPRD2 →OPRD1
4. 注意：对操作数的要求以及对标志位的影响与ADD指令相同
5. SBB
6. 操作：OPRD1- OPRD2- CF →OPRD1
7. 作用：主要用于多字节减法运算
8. 注意：指令格式、对操作数的要求、对标志位的影响与SUB指令

完全一样

1. DEC
2. 格式：DEC OPRD
3. 操作：OPRD - 1 →OPRD
4. 作用: 指令常用于在程序中修改计数值
5. 注意：指令对操作数的要求与INC相同（不能是段寄存器或立即

数）

4. NEG

(1) 格式：NEG OPRD

(2) 操作：0 - OPRD →OPRD

(3) 作用：用0减去操作数，相当于对该操作数**求补码**

(4) 注意：①8/16位寄存器或存储器操作数

②NEG指令对6个状态标志位均有影响，且：

i．执行NEG指令后，一般情况都会使CF为1，除非给定

操作数为0，CF才会为0。

ii．当指定的操作数为80H（-128）或为8000H（-32768），

则执行NEG指令后，结果不变，但OF置1，其它情况下OF均值0

5. CMP

(1) 格式: CMP OPRD1，OPRD2

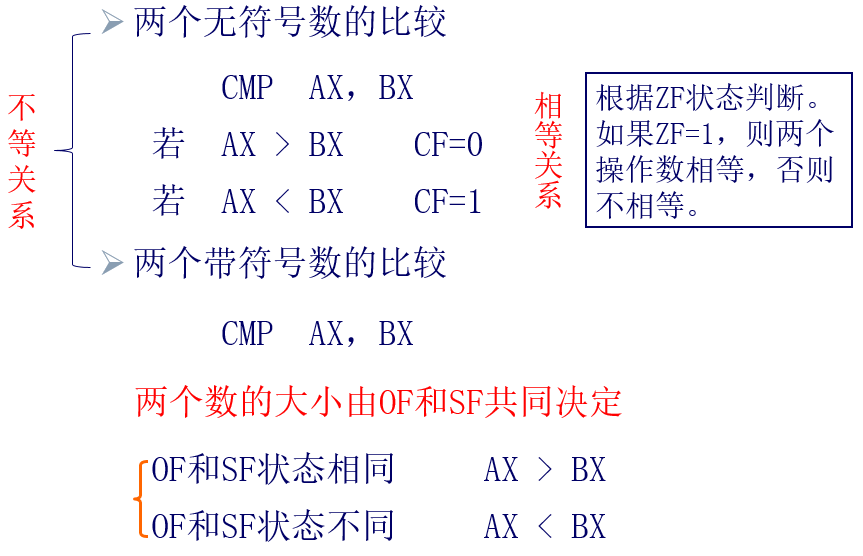
(2) 操作: OPRD1- OPRD2

(3) 作用: 用于比较两个数的大小，可作为条件转移指令转移的条件

(4) 注意：①！指令执行的结果不影响目标操作数，仅影响标志位

②指令对操作数的要求及对标志位的影响与SUB指令相同

1. 关于比较：





**！**乘法指令采用隐含寻址，隐含的是存放被乘数的累加器AL或AX及存放结果的AX，DX.

无符号的乘法指令MUL

带符号的乘法指令IMUL

**（三）乘法指令**

1. MUL
2. 格式：MUL OPRD
3. 操作:①OPRD为字节数→AL×OPRD→AL

② OPRD为16位数→AX×OPRD→DX:AX

1. 注意：两操作数字长必须相等，且不能为立即数
2. IMUL
3. 格式：IMUL OPRD
4. 操作：:①OPRD为字节数→AL×OPRD→AL

② OPRD为16位数→AX×OPRD→DX:AX

(DX为高16位，AX为低16位)

1. 注意: ①IMUL指令在格式上和功能上都与MUL指令类似，只是有如下区别：

i.要求两乘数都须为有符号数

ii.若乘积的高半部分是低半部分的符号位的扩展，则CF=OF=0，否则CF=OF=1

iii指令中给出的源操作数应满足带符号数的表示范围

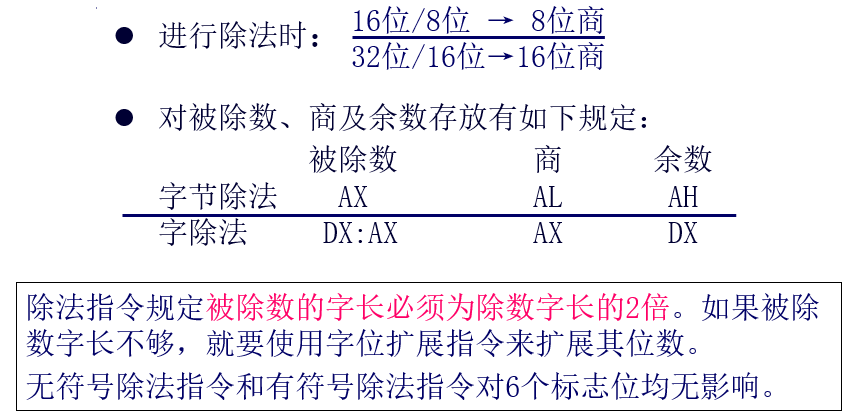
②有符号数乘法时，给出的数一般都是补码，必须先转换成源码才能相乘

无符号除法指令：DIV

有符号除法指令：IDIV



**(四) 除法指令**

 注意：

1. DIV
2. 格式：DIV OPRD

OPRD的长度决定除法类型

1. 操作: 字节操作 (AL)  (AX)/(OPRD)的商

(AH)  (AX)/(OPRD)的余数

 字操作 (AX)  (DX, AX)/(OPRD)的商

(DX)  (DX, AX)/(OPRD)的余数

1. 注意: 若除数为零或AL中商大于FFH,(或AX中商大于FFFFH)，则CPU产生一个类型0的内部中断
2. IDIV

注意:(1) 指令除要求操作数为有符号数外，在格式和功能上都和DIV

指令类似

(2)IDIV指令的结果，商和余数均为带符号数，且余数符号与被除数符号相同。

**(五) BCD码调整**

1. 总括
2. BCD码：用四个二进制位表示一个十进制数字；最常用的是8421 BCD码。

压缩BCD码：每一个10进制位用4位二进制表示，一个字节可以表示2个10进制数。

非压缩BCD码：每一个10进制位用1个字节二进制表示。其中，高4位为0，低4位存放对应的二进制数。

 (2) BCD码运算调整指令

加法10进制调整指令：DAA AAA

减法10进制调整指令：DAS AAS

乘法10进制调整指令：AAM

除法10进制调整指令：AAD

（3）注意:

①BCD调整指令不能单独使用，必须跟在算术运算指令之后（除法例

外）

②均为隐含寻址方式，隐含的操作数是AL或AL、AH

③BCD码本质上是十进制数，即应遵循逢十进一的规则。而计算机是按二进制（十六进制）进行运算，并未按十进制规则进行运算

2. 分类

(1) 压缩BCD码加法十进制调整指令DAA

①作用：用于对两个压缩BCD码相加之后的和（结果必须存入AL）进

行调整，产生正确的压缩BCD码。

②调整规则



若（AL）中低4位>9或AF=1，则（AL）+06H →（AL），并使AF=1

若（AL）中高4位>9或CF=1，则（AL）+60H →（AL），并使CF=1

1. 非压缩BCD码加法十进制调整指令AAA

①作用：对两个非压缩BCD数相加之和（结果必须存入AL）进行调

整，形成一个正确的非压缩BCD码，调整后的结果低位存入AL，高位存入AH

②具体步骤：

i．若（AL）中低4位>9或AF=1，则（AL）+6,（AH）+1，AF=1；

ii．屏蔽掉（AL）高4位，即（AL）←（AL）∩0FH；

iii．CF←AF

(3) 压缩BCD码减法十进制调整指令DAS

①作用： DAS用于对两个压缩BCD码相减后的结果（存入AL）进行

调整，产生正确的压缩BCD码

②调整规则



若（AL）中低4位>9或AF=1，则（AL）-06H，AF=1

若（AL）中高4位>9或CF=1，则（AL）-60H，CF=1

（4）非压缩BCD码减法十进制调整AAS

①作用：对两个非压缩BCD码相减后的结果（存入AL）进行调整，产

生正确的非压缩BCD码，其低位存入AL，高位存入AH

②调整规则

i．若（AL）中低4位>9或AF=1，则（AL）-06H，（AH）-1，AF=1

ii．屏蔽掉（AL）高4位，即（AL）←（AL）∩0FH；

iii．CF←AF

AAS对标志位的影响与AAA的影响类似

DAS对标志位的影响与DAA的影响类似

（5）乘法的十进制调整指令AAM

①作用：对两个非压缩BCD码数相乘的结果（存入AX）进行调整，

以得到正确的结果

②具体步骤： （AH）←（AL）/0AH

（AL）←（AL）%0AH

（把AL寄存器的内容除以OAH，商存入AH，余数存入AL）

③注意：i. AAM的实质是把AL中的二进制数转换为十进制数，所以

对于不超过99的二进制数，只用一条AAM指令即可实现

二-十进制转换

ii．AAM指令影响PF、SF和ZF标志位

iii．执行AAM指令前须有一条MUL指令（BCD码总视为无

符号数）将两个非压缩BCD码相乘，结果放入AL，然后用AAM指令进行调整

（6）除法的十进制调整指令AAD

①使用原则：在两个非压缩BCD码相除之前，先用一条AAD指令进

行调整，然后再用DIV指令

②具体步骤：（AL）←（AH）\*10+（AL）

（AH）←0

③操作：把AX中的非压缩BCD码（十位数放AH，个位数放AL）调

整为二进制数，并将结果放入AL中。AAD的操作实质上是把AX中的两位十进制数转换为二进制数

④注意：i.对于不超过99的十进制数，只用一条AAD指令即可实现十-二进制转换

ii．AAD指令影响PF、SF和ZF标志位

第三类 : 逻辑运算和移位指令

一、概括

1.逻辑运算指令和移位指令包括在位操作指令里，位操作指令对二进制数

的各位进行操作

2.逻辑运算指令：and，or，not，xor，test

3.移位指令：非循环移位，循环移位

4.注意：（1）逻辑运算指令对操作数的 要求大多与MOV指令相同

（2）除“非”运算指令 外，其余逻辑运算指令的执行都会使标志

位OF=CF=0,AF值不定，并对SF，PF和ZF有影响

二、分讲

**（一）逻辑运算指令**

1. AND “与”
2. 格式：AND OPRD1，OPRD2
3. 操作：两操作数相“与”，结果送目标地址
4. 注意：源操作数OPRD2可以是寄存器、存储器或立即数，但目标操

作数OPRD1只能是寄存器或存储器

(4) 应用：①实现两操作数按位相与的运算

Eg.AND BL，[SI]

②使目标操作数的某些位不变，某些位清零

Eg .AND AL，0FH

③在操作数不变的情况下使CF和OF清零

Eg .AND AX，AX

2. OR “或”

（1）格式： OR OPRD1，OPRD2

(2)操作：两操作数相“或”，结果送目标地址

(3)注意：①对操作数的要求以及对标志位的影响和AND指令一样

②常用于奇偶校验

(4)应用：①实现两操作数相“或”的运算

Eg .OR AX，[DI]

②使某些位不变，某些位置“1”

Eg. OR CL，0FH

③在不改变操作数的情况下使OF=CF=0

Eg. OR AX，AX

3.NOT “非”

(1)格式：NOT OPRD

(2)操作：操作数按位取反再送回原地址

(3)注意：①操作数为8位或16位的寄存器或者存储器操作数，但不能是立即数

②指令中的操作数不能是立即数

③指令的执行对标志位无影响

4.XOR “异或”

(1)格式：XOR OPRD1，OPRD2

(2)操作：两操作数相“异或”，结果送目标地址

(3)作用：某一操作数和自身相“异或”，结果为0，在程序中常利用

这一特性，使某寄存器清零

5.TEST 测试

(1)格式: TEST OPRD1，OPRD2

(2)操作：执行“与”运算，但运算的结果不送回目标地址，只是影响

标志位

(3)应用：常用于测试某些位的状态

**（二） 移位指令**



非循环移位指令

循环移位指令

注意：

移动一位时由指令直接给出；

移动两位及以上，则移位次数由CL指定



逻辑左移SHL

算术左移SAL

逻辑右移SHR

算术右移SAR

1. 非循环移位指令

左移可实现乘法运算

右移可实现除法运算

(1)SHL SAL

注意：①移位指令按照移入的位设置进位标志CF，根据移位后的结果影响SF、ZF、PF，对AF没有定义

②

i.如果进行1位移动，则按照操作数的最高符号位是否改变，

相应设置溢出标志OF:如果移位前的操作数最高位与移位后

操作数的最高位不同（有变化），则OF=1,否则OF=0

ii.当移位次数大于1时，OF不确定

③逻辑左移和算术左移实际上是同一条指令的两种助记符，两者完全相同，建议采用SHL



（2）SHR

①格式：SHR OPRD，1

SHR OPRD，CL

②注意：i. 它将目标操作数视为无符号数，其操作是将目的操作数顺序向右移1位或CL指定的位数，每右移1位，右边的最低位移入标志位CF，而在左边的最高位补零

ii.SHR指令也影响标志位CF和OF。如果移动次数为1，且移位之后新的最高位和次高位不相等，则标志位OF=1，否则OF=0。若移位次数不为1，则OF状态不定

iii.逻辑右移指令SHR执行1位移位，相当于无符号数除以2，商在操作数中，余数由CF标志反映



（3）SAR

①格式：SAR OPRD，1

SAR OPRD，CL

②注意：i. SAR指令将目标操作数视为有符号数，指令的操作是将目标操作数顺序向右移1位或CL指定的位数，操作数最低位移入标志位CF

ii. 与SHR指令的区别是：算术右移时最高位不是补零，而是保持不变。

iii.对标志位CF、OF、PF、SF和ZF有影响，但使AF值不定

iv.执行1位移位时，相当于有符号数除以2。

v.当操作数为负（最高位为1）且最低位有1位移出时，SAR指令产生的结果与等效的IDIV指令的结果不同



 2.循环移位指令

左移 ROL

右移 ROR



左移 RCL

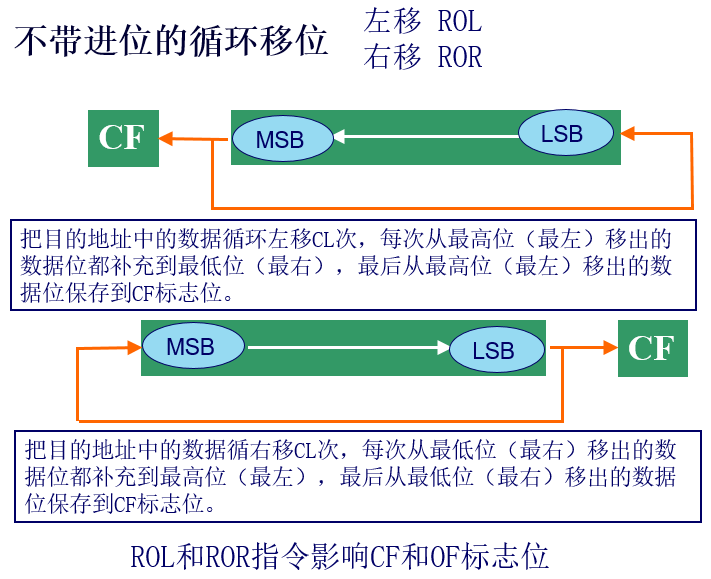
右移 RCR

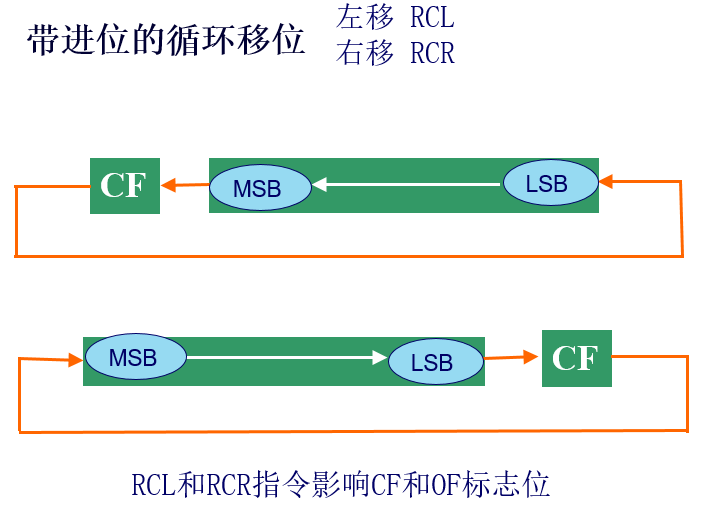
不带进位的循环移位

带进位的循环移位

指令格式、对操作数的要求与非循环移位指令相同

循环移位指令类似移位指令，但要从一端移出的位返回到另一端形成循环





注意：（1）ROL、ROR不将进位CF纳入循环位中；RCL、RCR将进位标志CF纳入循环位中，与操作数一起构成的9位或17位二进制数一起移位

（2）循环移位指令按照指令功能设置进位标志CF，不影响SF、ZF、PF、AF标志。对OF标志的影响，循环移位指令与前面介绍的移位指令一样。

（3）利用移位或循环移位指令可以方便地实现BCD码转换。

第四类：程序控制指令

一、总括

转移指令

循环控制

过程调用

中断控制

1.分类：

2. 程序控制类指令的本质是：控制程序的执行方向

3.决定程序执行方向的因素：CS，IP

4.控制程序执行方向的方法：

修改CS和IP，则程序转向另一个代码段执行；

仅修改IP，则程序将改变当前的执行顺序，转向本代码段内其它某处执行。

二、分讲

无条件转移指令

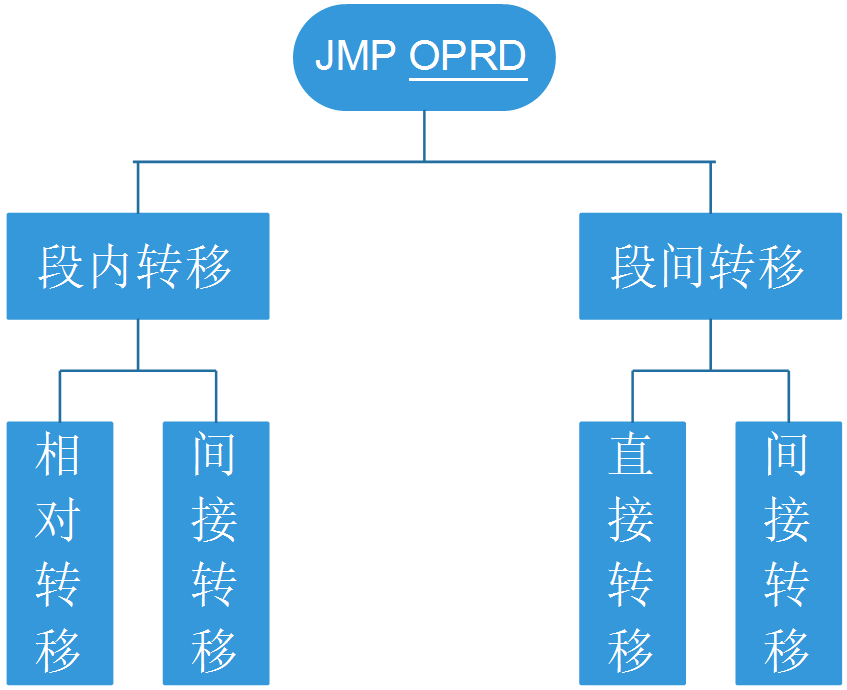
（无条件转移到目标地址，执行新的指令）

有条件转移指令

（在具备一定条件的情况下转移到目标地址）

**（一）转移指令**

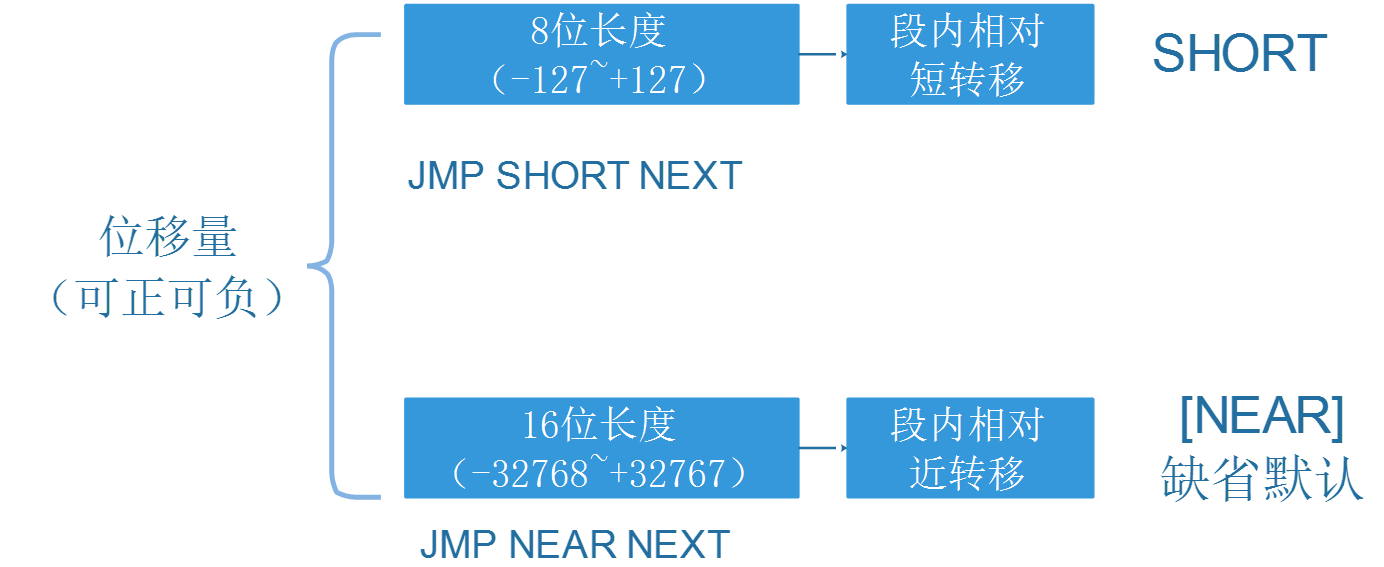
1.无条件转移 JMP



JMP指令可以将程序转移到1MB存储空间的任何位置。

(1）段内转移

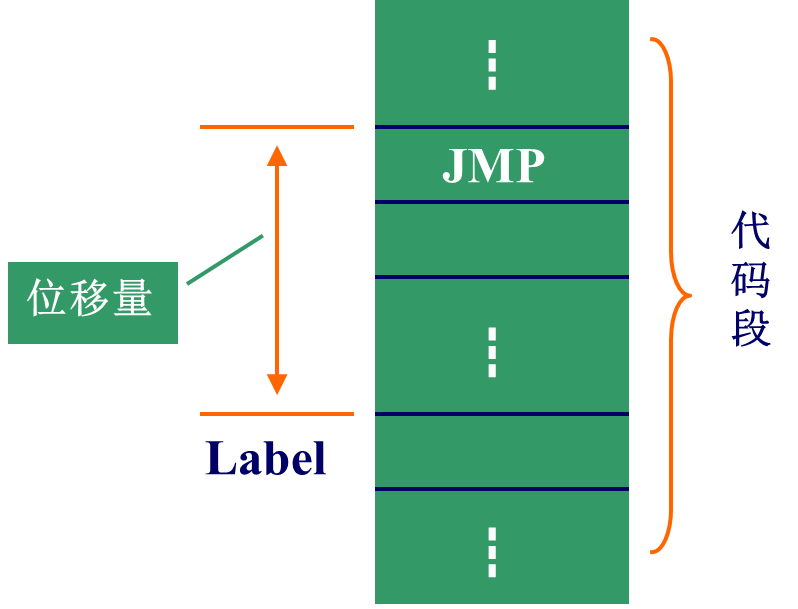
指在当前代码段64KB范围内转移，因此不需要更改CS段地址，

只要改变IP偏移地址

①段内转移，相对寻址

i.格式：jmp label ；IP←IP+偏移量

ii.注意：指令代码中的位移量是指，紧接着JMP指令后那条指令的偏移地址到目标指令的偏移地址的地址位移。



②段内转移，间接寻址

i.格式：jmp r16/m16

ii.操作：将一个16位寄存器或主存单元内容送入IP寄存器，作为新的指令指针，但不修改CS寄存器的内容。

iii.作用：测是否字节可以进行

（2）段间转移

指从当前代码段跳转到另一个代码段，此时需要更改CS段地址和IP偏移地址，这种转移也称为“远转移far jump”。转移的目标地址必须用一个32位数表达，叫做32位远指针，它就是逻辑地址。

①段间转移，直接寻址

i.格式：JMP FAR Label

IP←label的偏移地址，CS←label的段地址

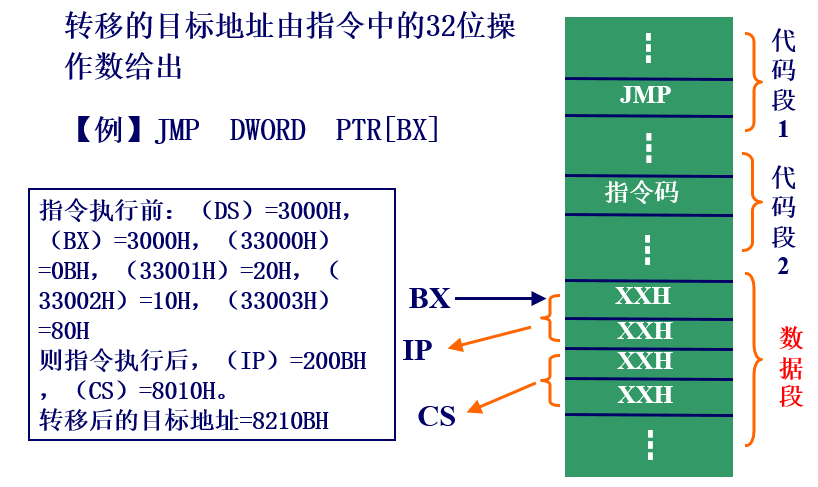
ii.操作：将标号所在段的段地址作为新的CS值，标号在该段的偏移地址作为新的IP值

iii.一个标号在同一个段内还是在另一个段中，汇编程序能够识别。如果要强制一个段间远转移，则可以用汇编伪指令far ptr

②段间转移，间接寻址

i,使用：用一个双字存储单元表示要跳转的目标地址。这个目标地址存放在主存中连续的两个字单元中，其中低位字送IP寄存器，高位字送CS寄存器·

ii.for exmple：



2.条件转移指令

◎条件转移指令JCC根据指定条件确定程序是否发生转移；

在满足一定条件下，程序转移到目标地址继续执行

◎通用格式：jcc label

条件满足：发生转移：IP←IP+8位位移量

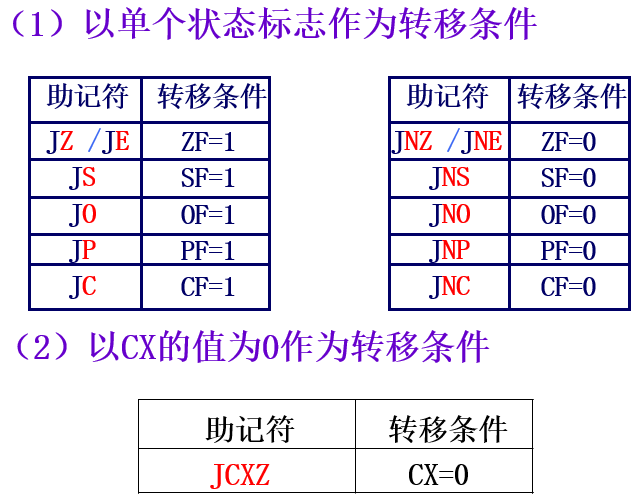
否则，顺序执行：IP←IP+2

其中，label表示目标地址（8位位移量），因为jcc指令为2字节，所以顺序执行就是当前指令偏移地址指针IP+2

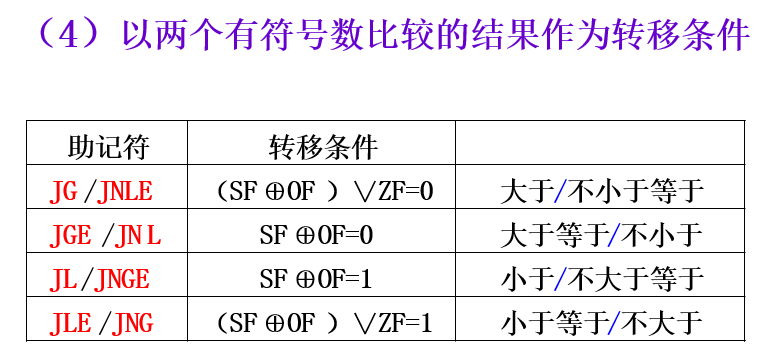
◎条件转移指令跳转的目标地址只能用前面介绍的段内相对短转移，即目标地址只能在同一段中，且在当前IP地址-128到127个单元范围内

◎与其他控制转移指令一样，条件转移指令不影响标志，但是它要利用标志！

◎判断条件共有16种，表中斜线分隔了同一条指令的多个助记符形式：

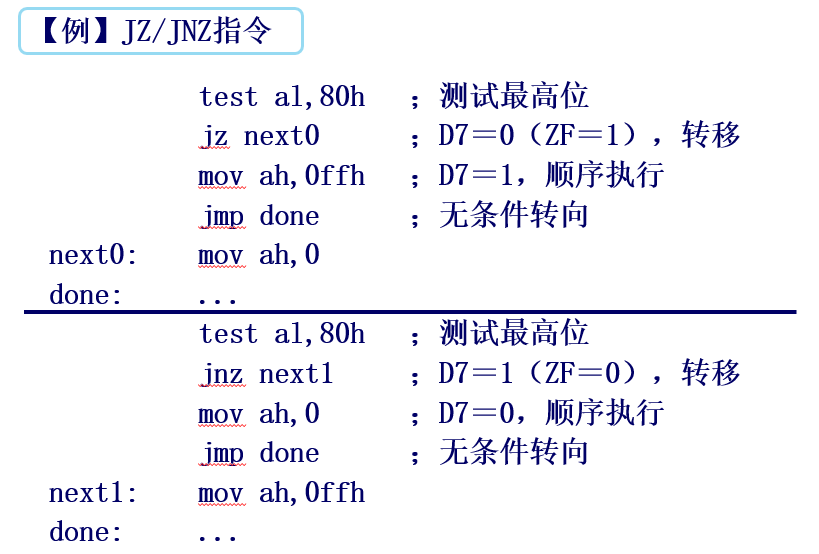




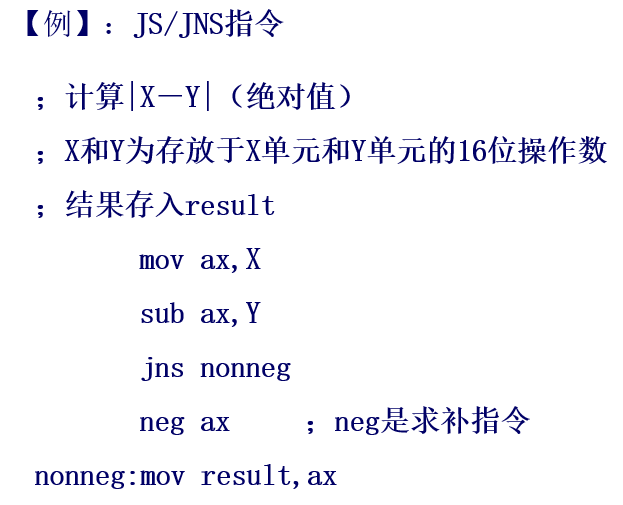


（1）判断单个标志位状态

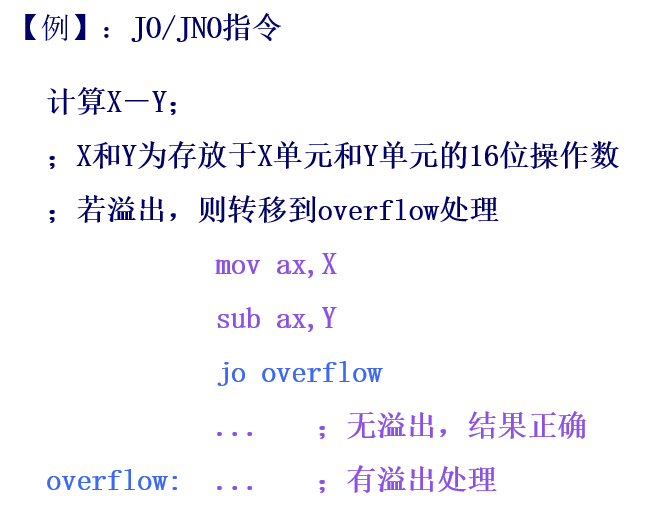
①JZ/JE和JNZ/JNE利用零标志ZF，判断结果是否为零



②JS/JNS利用符号标志SF，判断结果是正是负



③JO和JNO利用溢出标志OF，判断结果是否溢出



④JP/JPE和JNP/JPO利用奇偶标志PF，判断结果中的“1”的个数是偶数还是奇数

⑤JC/JB/JNAE和JNC/JNB/JAE，利用进位标志CF，判断解说是否进位或借位

（2）比较无符号数高低

为了区别有符号数的大小，无符号数的大小用高、低表示，需要利用CF确定高低，利用ZF标志确定相等。两数的高低分成4种关系：低于，不低于，低于等于，不低于等于，也就分别对应4条指令：JB(JNAE),JNB(JAE),JBE(JNA),JNBE(JA)

（3）比较有符号数大小

需要组合OF,SF标志，并用ZF标志确定相等与否。俩数的大小分成4中关系:小于，不小于，小于等于，不小于，也就对应4条指令：JL(JNGE),JNL(JGE),JLE(JNG),JNLE(JG)

**（二）循环指令**

1. 循环范围：以当前IP为中心的-128～+127范围内循环。

2. 循环次数由CX寄存器指定。

3. 循环指令：

LOOP （无条件转移指令）

\*LOOPZ

\*LOOPNZ

4.循环指令中的操作数采用相对寻址的方式，表示循环的目标地址是一个8位位移量，另外，循环指令不影响标志。

5. ! LOOP

(1) 格式：LOOP LABEL

(2) 循环条件：CX ≠ 0

(3) 操作： 先将计数值CX-1，再判断计数值CX是否为零

DEC CX

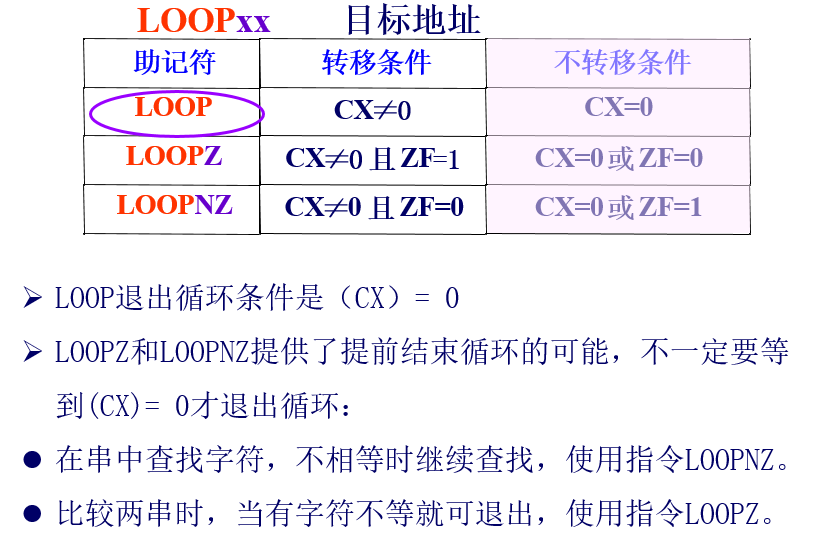
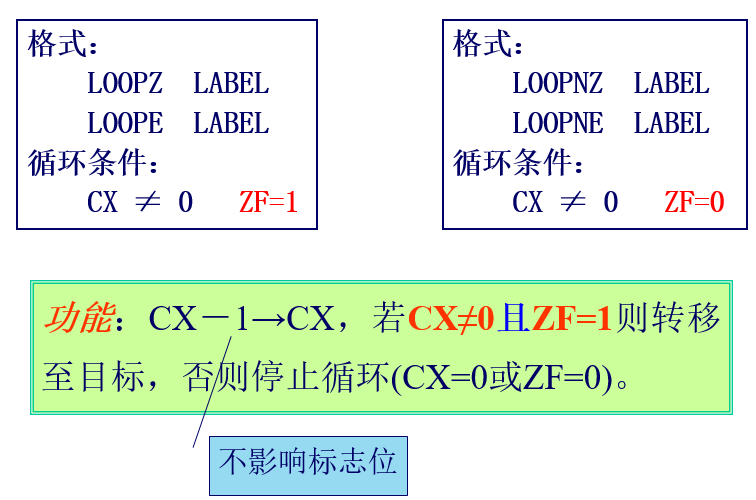
JNZ 符号地址

(4)要点：i.在CX中存放循环次数

ii.LOOP指令的标号一般在前面

iii.要执行的循环程序段英卸载标号和LOOP指令之前

6.LOOPZ LOOPNZ

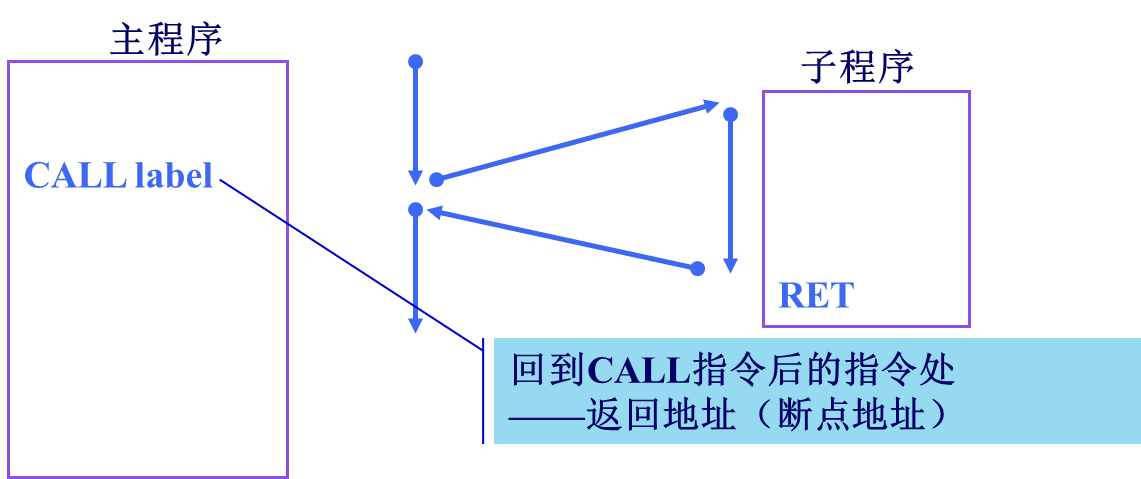
 

7.区别

**（三）过程调用与返回**

◎子过程（程序）是完成特定功能的一段程序

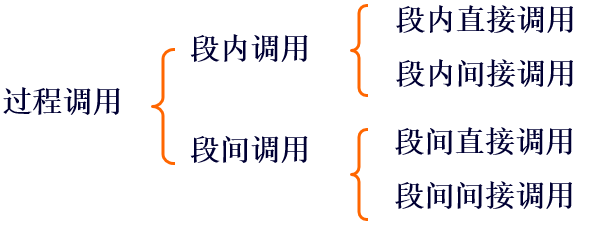
◎当主程序（调用程序）需要执行这个功能时，采用CALL调用指令转移该子程序的起始处执行

◎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

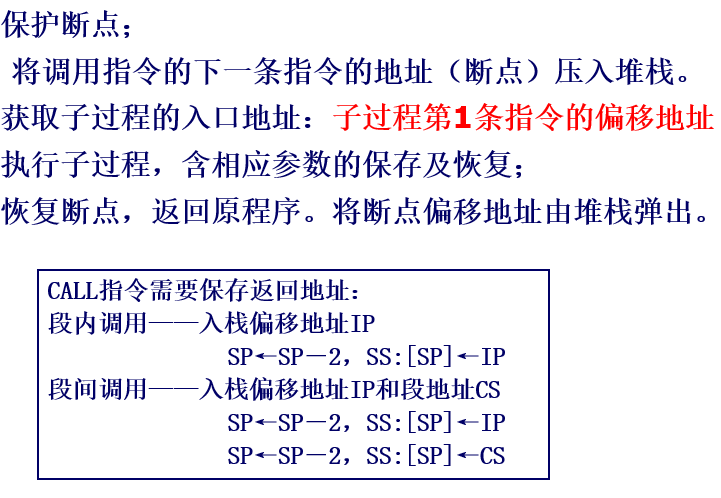
1.过程调用

（1）格式：CALL OPRD

（2）分类：



（3）调用过程：



（4）

①段内调用

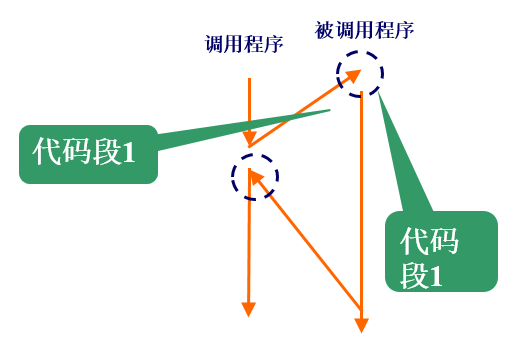
i.格式： CALL NEAR PROC（近过程名）

ii.执行过程：将断点的偏移地址压入堆栈

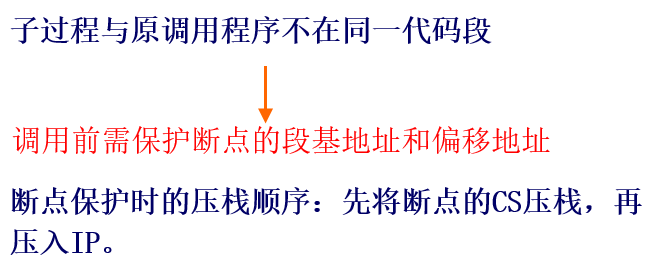
根据过程名找子程序入口

iii.要求：被调用程序与调用程序在同一代码段

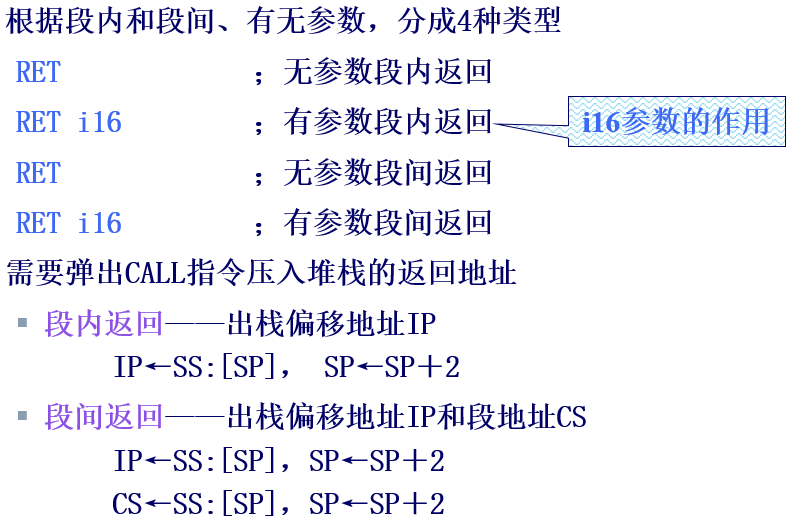
调用前只需保护断点的偏移地址



②段间调用



2.过程返回



**（四）中断指令**

1.中断与过程调用

①中断是随机事件或异常事件引起，调用则是事先已在程序中安排好；

②响应中断请求不仅要保护断点地址，还要保护FLAGS内容；

③调用指令在指令中直接给出子程序入口地址，中断指令只给出中断向量

码，入口地址则在向量码指向的内存单元中。

2.中断指令

格式：INT n （n=0~255，n为中断类型码）

3.中断过程

①将FLAGS压入堆栈；

②将INT指令的下一条指令的CS、IP压栈；

③由n×4得到存放中断向量的地址；

④将中断向量（中断服务程序入口地址）送CS和IP寄存器；

⑤转入中断服务程序。