指令寻址方式

寻址方式（Addressing）

- 通过地址访问数据或指令
- 数据寻址：

  指令执行过程中，

  访问所需要操作的数据(操作数)

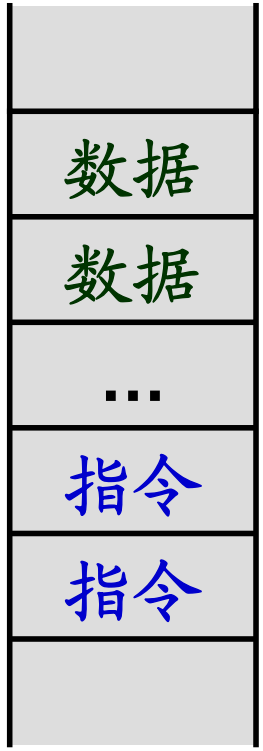
操作数在哪儿呢？

- 指令寻址：

  一条指令执行后，

  确定执行的下一条指令的位置

指令又在哪儿呢？

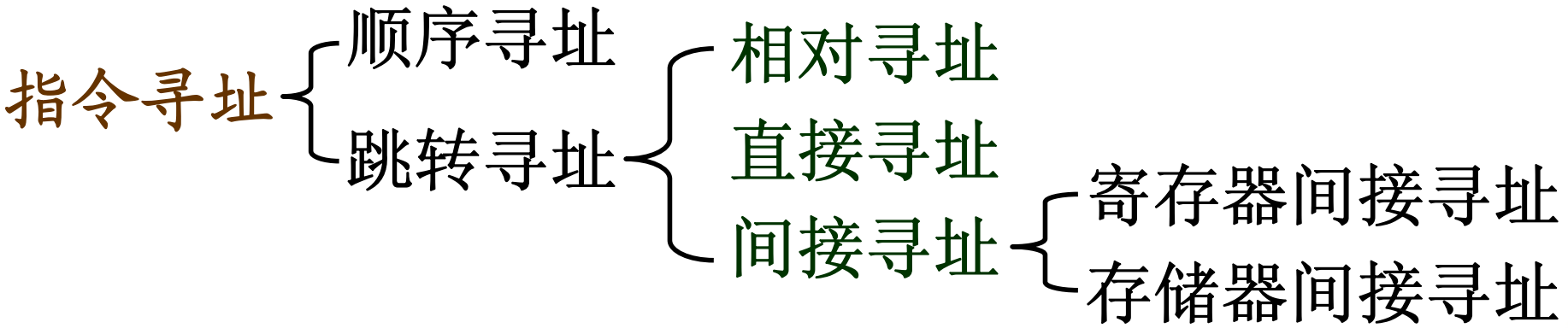


数据寻址的总结

- 数据寻址
  - 立即(数)寻址
  - 寄存器寻址
  - 存储器寻址
    - 直接寻址
    - 寄存器间接寻址
    - 寄存器相对寻址
    - ……
  - 堆栈寻址
  - 隐含寻址
  - 外设(I/O)寻址
    - 直接寻址
    - 间接寻址

指令寻址

- 顺序寻址：EIP自动增量指向下一条指令
  - 顺序执行接着的下一条指令
- 跳转寻址：控制流程跳转(转移)到指定指令位置
  - 实现程序分支、循环、调用等结构

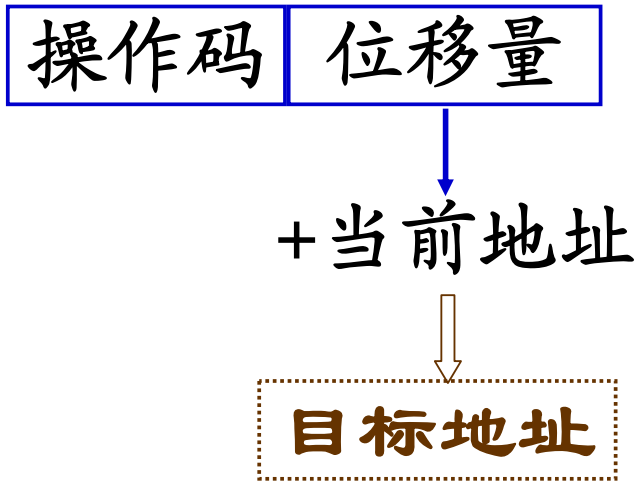


指令的相对寻址

- 提供目标地址相对于当前指令指针EIP的位移量

  目标地址(转移后的IP)＝当前EIP＋位移量

  - 相对寻址都是段内转移
  - 最常用、最灵活



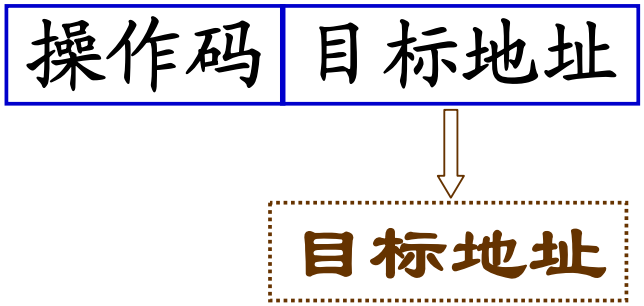
目标地址＝目的地址＝转移地址

指令的直接寻址

- 直接提供目标地址

  目标地址＝指令操作数

  - 理论上可以段内或段间转移
  - IA-32只支持段间的直接转移

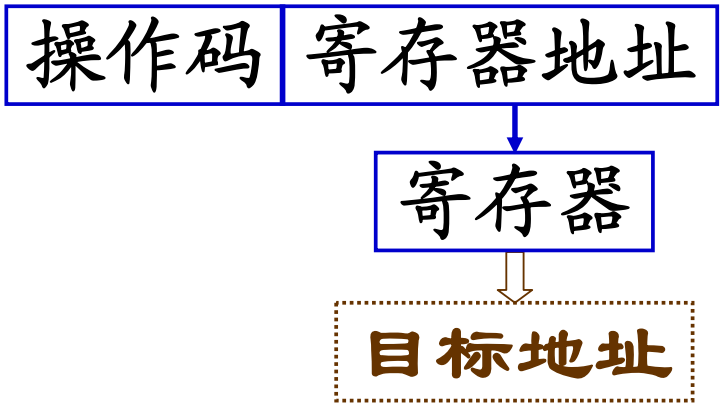


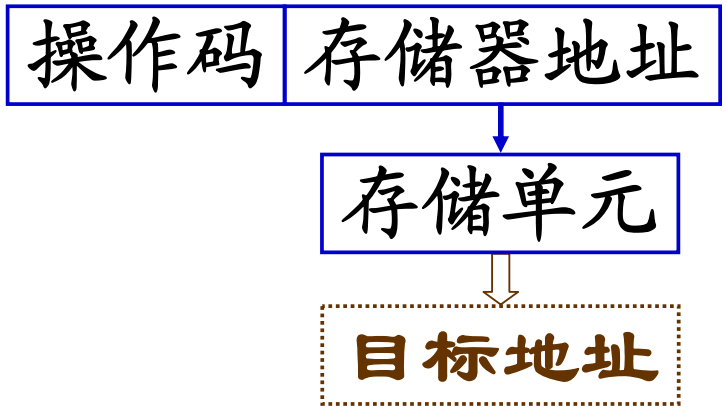
指令的间接寻址

- 指示寄存器或存储单元

  目标地址来自寄存器或存储单元、间接获得

  - 寄存器间接寻址：用寄存器保存目标地址
  - 存储器间接寻址：用存储单元保存目标地址





程序流程的控制转移

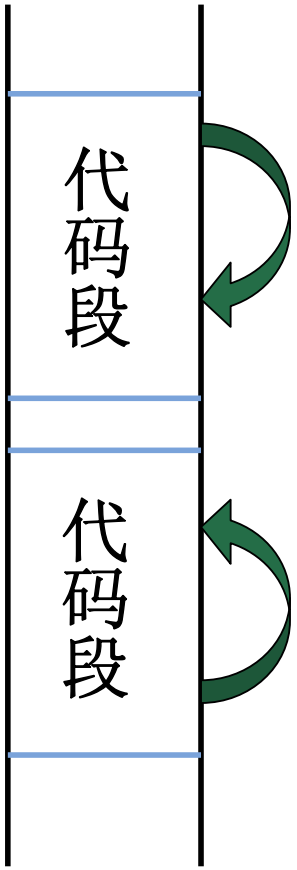
- 程序代码在代码段
  - **CS**：指明代码段在主存中的段基地址
  - **EIP**：给出将要执行指令的偏移地址
- 指令顺序寻址，程序顺序执行
  - 处理器自动增量**EIP**
- 指令跳转寻址，程序控制转移
  - **EIP**（**CS**）随之改变

控制转移类指令

JMP Jcc LOOP

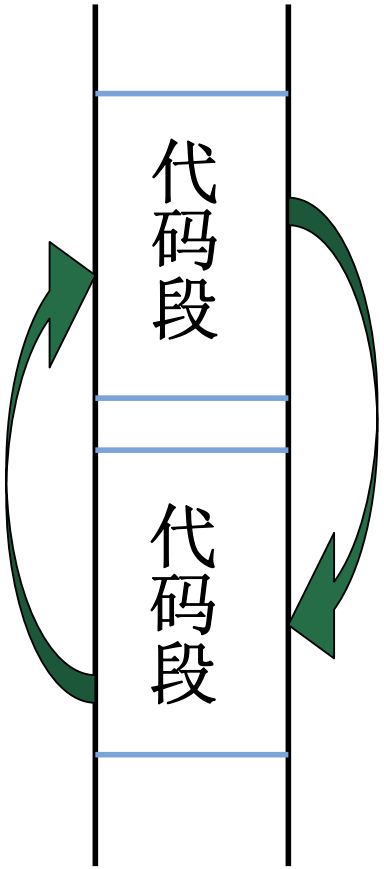
程序转移的范围：段内转移

- 在当前代码段范围内的程序转移
  - 不需更改CS，只要改变EIP（偏移地址）
- 被称为“近转移”
  - 类型属性使用“NEAR”关键字
- 如果转移范围在127～-128字节之间，（位移量使用1个字节）又称为“短转移”
  - 类型属性使用“SHORT”关键字



程序转移的范围：段间转移

- 从当前代码段跳转到另一个代码段
- 需要更改CS（段地址）和EIP（偏移地址）
- 被称为“远转移”
  - 类型属性使用“FAR”关键字

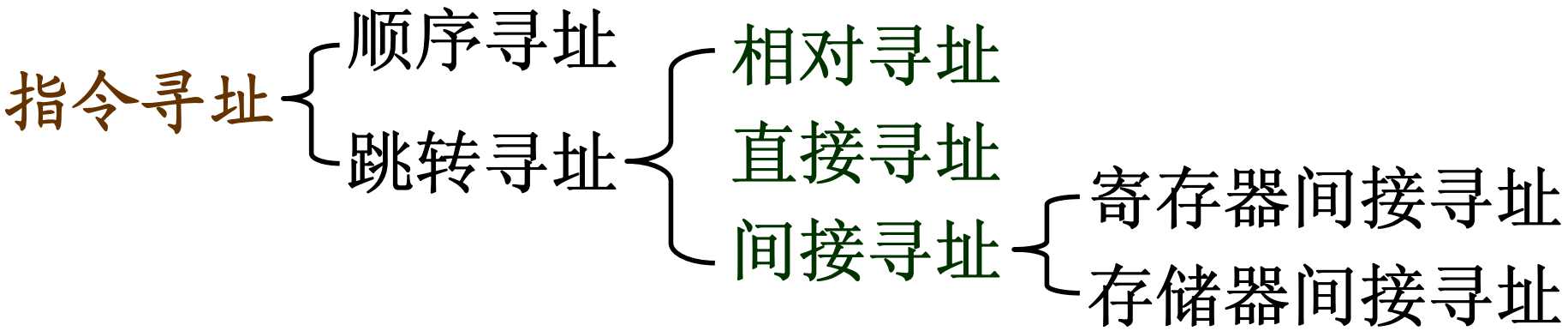


标号、子程序名等具有的类型属性

NEAR　　SHORT　　FAR

指令寻址的总结

- IA-32处理器通过改变EIP（和CS）控制程序流程
  - 实现程序分支、循环、调用等结构
- 可以实现代码段内、或代码段间跳转
  - 具有短转移short、近转移near和远转移far属性



JMP指令

无条件转移指令（JMP）

JMP label ;程序转向label标号指定的地址

;段内相对寻址，段间直接寻址

JMP reg32/reg16 ;程序转向寄存器指定的地址

;寄存器间接寻址

JMP mem48/mem32/mem16

;程序转向存储单元指定的地址

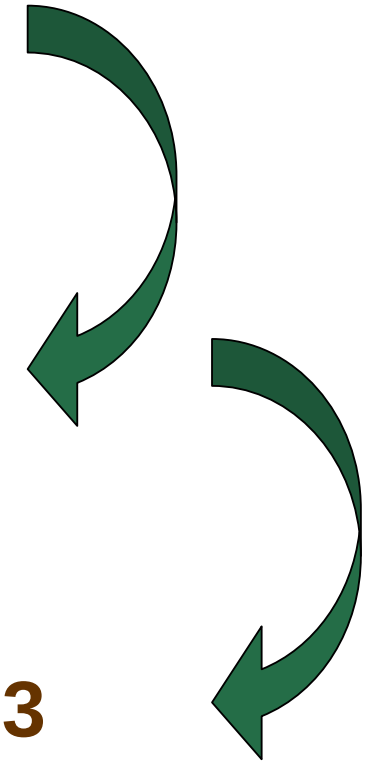
;存储器间接寻址

无条件转移程序—1

```
;代码段
        jmp labl1
        nop

labl1:  jmp near ptr labl2
        nop

labl2:  mov eax,offset labl3
```



无条件转移程序—1

```
                    ;代码段
00000000  EB 01     jmp labl1              ;相对寻址，短转移
00000002  90        nop
00000003  E9 00000001
          labl1:    jmp near ptr labl2     ;相对寻址，近转移
00000008  90        nop
00000009  B8 00000011 R
          labl2:    mov eax,offset labl3
```

位移量，1个字节

位移量，4个字节

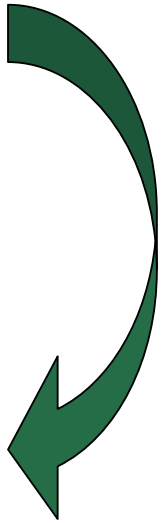
无条件转移程序—2

```
labl2:   mov eax,offset labl3

         jmp eax

         nop

labl3:   mov eax,offset labl4
```



无条件转移程序—2

```
00000009  B8 00000011 R
          labl2:   mov eax,offset labl3
0000000E  FF E0
                   jmp eax                    ;寄存器间接寻址
00000010  90
                   nop
00000011  B8 00000022 R
          labl3:   mov eax,offset labl4
```

无条件转移程序—3

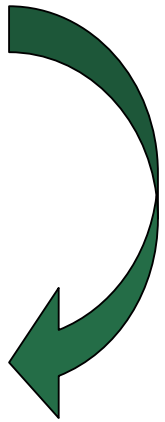
```
labl3:   mov eax,offset labl4

         mov nvar,eax

         jmp nvar

         nop
labl4:   ...
```

```
                    ;数据段
nvar    dword ?
```



无条件转移程序—3

```
                labl3:   mov eax,offset labl4
00000016  A3 00000000 R
                         mov nvar,eax
0000001B  FF 25 00000000 R
                         jmp nvar
00000021  90
                         nop
                labl4:   ...
```

```
                ;数据段
nvar    dword ?
```

数据的存储器直接寻址

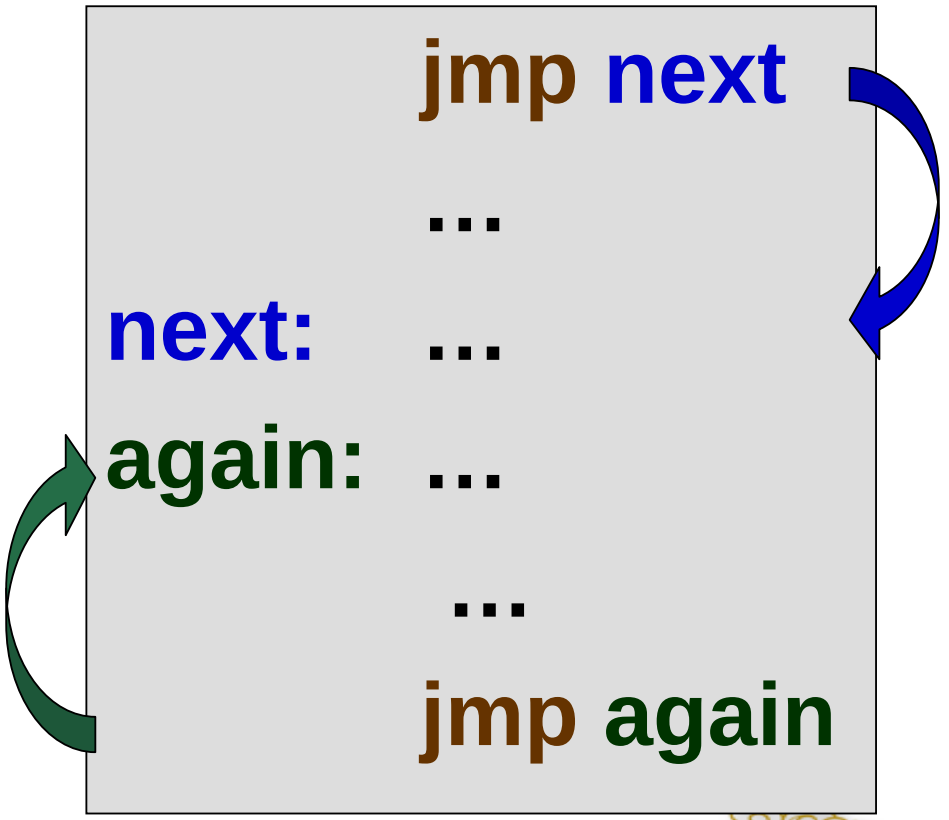
;存储器间接寻址

```
mov ebx,offset nvar
jmp near ptr [ebx]
```

;数据的寄存器间接寻址

本讲总结

- JMP指令实现无条件的程序流程转移
- 对应C语言的goto语句
- 高级语言慎用goto语句
- 处理器必不可少JMP指令



Jcc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

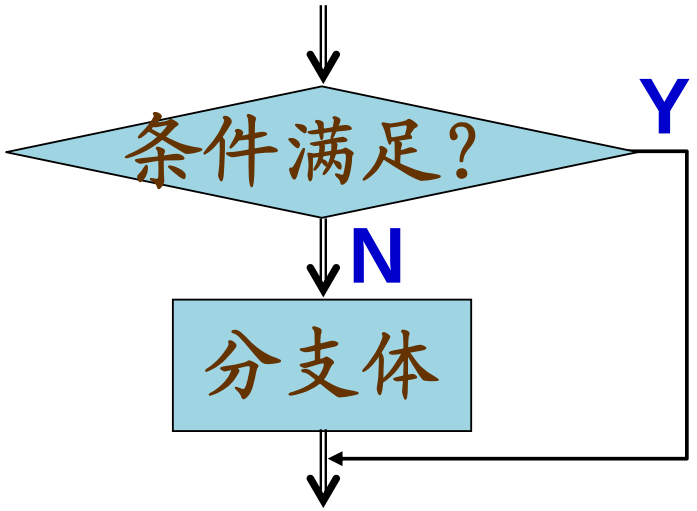
  Jcc label

  ;条件满足，发生转移；

  ;否则，顺序执行下条指令

- label表示目标地址

  采用段内相对寻址



Jcc: jump with condition

判断的标志条件cc

- 共16条指令，分成两类
  - 单个标志状态作为条件

    5个状态标志ZF、CF、SF、OF和PF的10种状态
  - 两数大小关系作为条件
    - 比较无符号整数大小

      4种情况：低于、不低于、低于等于、高于
    - 比较有符号整数大小

      4种情况：小于、不小于、小于等于、大于

利用零位标志ZF的条件转移指令

- 判断条件：运算结果为0、两数相等（标志ZF=1）

JZ label ;Jump if Zero

JE label ;Jump if Equal

- 判断条件：结果不为0、不相等（标志ZF=0）

JNZ label ;Jump if Not Zero

JNE label ;Jump if Not Equal

多个助记符方便记忆

利用进位标志CF的条件转移指令

- 判断条件：运算结果有进位(借位)（标志CF=1）

  JC label　　;Jump if Carry

- 判断条件：结果没有进位(借位)（标志CF=0）

  JNC label　　;Jump if Not Carry

JC还等于JB和JNAE指令

JNC还等于JNB和JAE指令

利用溢出标志OF的条件转移指令

- 判断条件：运算结果有溢出（标志OF=1）

  JO label　　;Jump if Overflow

- 判断条件：结果没有溢出（标志OF=0）

  JNO label　　;Jump if Not Overflow

溢出标志OF针对有符号整数加减运算

进位标志CF针对无符号整数加减运算

利用符号标志SF的条件转移指令

- 判断条件：运算结果是负、最高位为1（标志SF=1）

  JS label　　;Jump if Sign

- 判断条件：结果是正、最高位为0（标志SF=0）

  JNS label　　;Jump if Not Sign

有符号整数采用补码，最高位是符号位

符号位为1，表示负数；符号位为0，表示正数

利用奇偶标志PF的条件转移指令

- 判断条件：低8位结果中1的个数为偶或0（标志PF=1）

JP label ;Jump if Parity

JPE label ;Jump if Parity Even

- 判断条件：低8位结果中1的个数为奇（标志PF=0）

JNP label ;Jump if Not Parity

JPO label ;Jump if Parity Odd

多个助记符方便记忆

两个无符号整数大小关系的条件转移指令-1

- 判断条件：低于、不高于等于（标志CF=1）

  JB ;Jump if Below

  JNAE ;Jump if Not Above or Equal

JC

- 判断条件：不低于、高于等于（标志CF=0）

  JNB ;Jump if Not Below

  JAE ;Jump if Above or Equal

JNC

无符号数大小用高(Above)、低(Below)助记符

两个无符号整数大小关系的条件转移指令-2

- 判断条件：低于等于、不高于（标志CF=1或ZF=1）

  JBE ;Jump if Below or Equal

  JNA ;Jump if Not Above

- 判断条件：不低于等于、高于（标志CF=0且ZF=0）

  JNBE ;Jump if Not Below or Equal

  JA ;Jump if Above

无符号数大小用高(Above)、低(Below)助记符

两个有符号整数大小关系的条件转移指令-1

- 判断条件：小于、不大于等于（标志SF≠OF）

  JL ;Jump if Less

  JNGE ;Jump if Not Greater or Equal

- 判断条件：不小于、大于等于（标志SF=OF）

  JNL ;Jump if Not Less

  JGE ;Jump if Greater or Equal

有符号数大小用大(Greater)、小(Less)助记符

两个有符号整数大小关系的条件转移指令-2

- 判断条件：小于等于、不大于(标志SF≠OF或ZF=1)

  JLE ;Jump if Less or Equal

  JNG ;Jump if Not Greater

- 判断条件：不小于等于、大于(标志SF=OF且ZF=0)

  JNLE ;Jump if Not Less or Equal

  JG ;Jump if Greater

有符号数大小用大(Greater)、小(Less)助记符

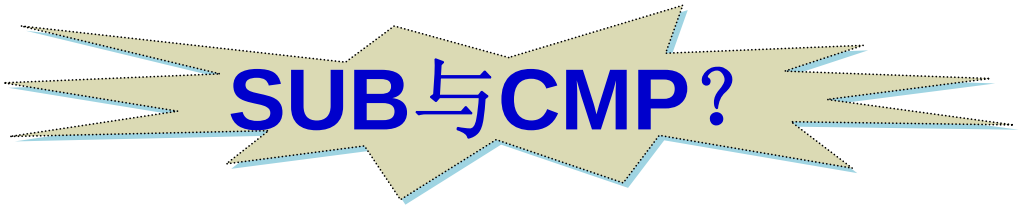
产生条件的指令

- 常用指令1：比较指令CMP
  - 进行减法运算
  - 用于判断两个数据大小、是否相等
- 常用指令2：测试指令TEST
  - 进行逻辑与运算
  - 用于判断某位为0或为1等
- 其他指令：能够影响状态标志的指令

加减运算指令、逻辑运算指令、移位指令等

比较指令CMP（compare）

- 将目的操作数减去源操作数
  - 差值不回送目的操作数
  - 按照减法结果影响状态标志



CMP reg,imm/reg/mem　　;reg－imm/reg/mem

CMP mem,imm/reg　　;mem－imm/reg

- 根据标志状态获知两个操作数的大小关系
- 给条件转移等指令使用其形成的状态标志

测试指令TEST

- 按位进行逻辑与运算，不返回逻辑与结果

  TEST reg,imm/reg/mem　　;reg ∧ imm/reg/mem

  TEST mem,imm/reg　　;mem ∧ imm/reg

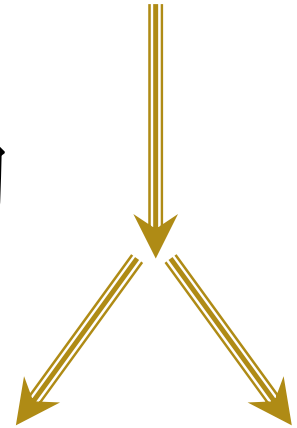
- **TEST指令像AND指令一样来设置状态标志**
- 常用于检测一些条件是否满足，一般后跟条件转移指令，目的是利用测试条件转向不同的分支

AND与TEST？

TEST与CMP？

本讲总结

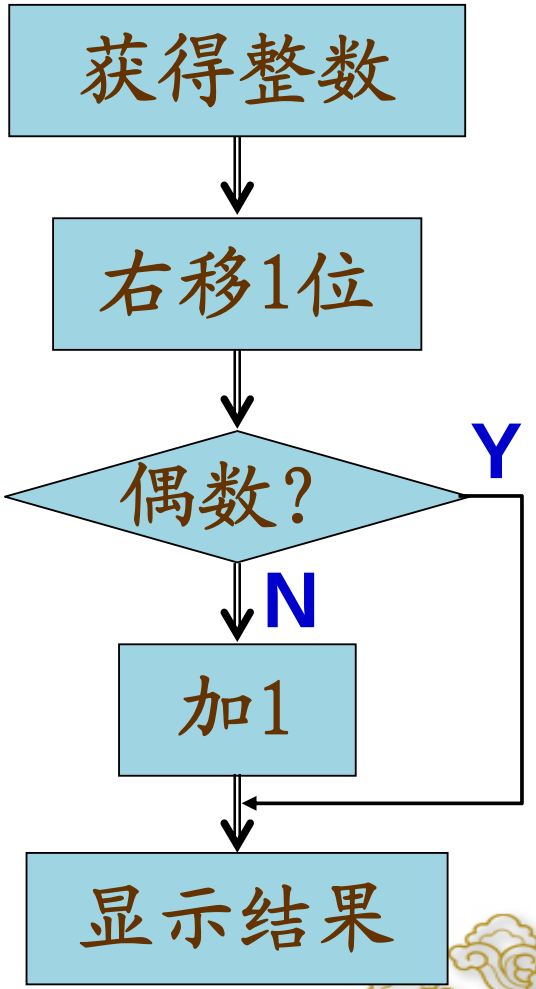
- 条件转移指令
  - 在满足条件的情况才实现转移
  - 条件不满足，则顺序执行
- 利用**Jcc**指令实现分支、循环程序结构
- 其作用对应**C**语言的**if**语句



个数折半程序

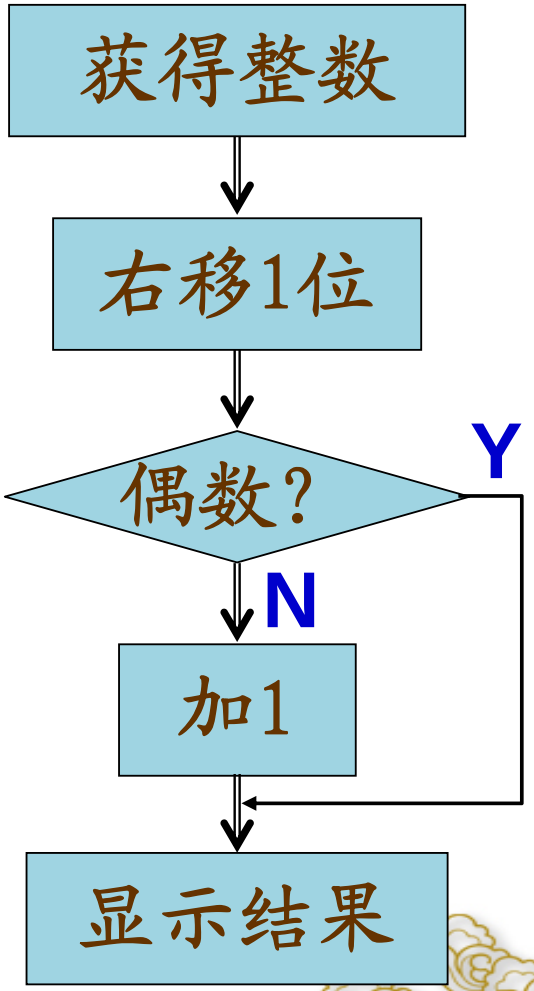
个数折半

- 对一个（无符号）整数折半
  - 就是对数据除以2
  - 数据右移1位实现除以2更方便
- 如果是偶数，完成
- 如果是奇数，需要加1
- 显示折半后的结果



偶数判断

- （SHR指令）右移1位对整数折半
- 如果是偶数、最低位是0
  - 即，移入CF标志的位为0，完成
- 如果是奇数，需要加1
- 显示折半后的结果



条件转移指令（Jcc）

- 根据指定的条件确定程序是否发生转移

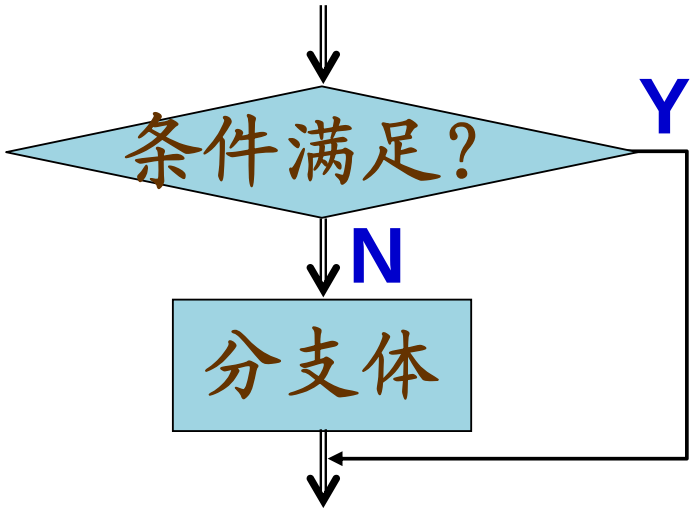
  Jcc label

  ;条件满足，发生转移；

  ;否则，顺序执行下条指令

- label表示目标地址

  采用段内相对寻址



Jcc: jump with condition

利用进位标志CF的条件转移指令

- 判断条件：运算结果有进位(借位)（标志CF=1）

  JC label　;Jump if Carry

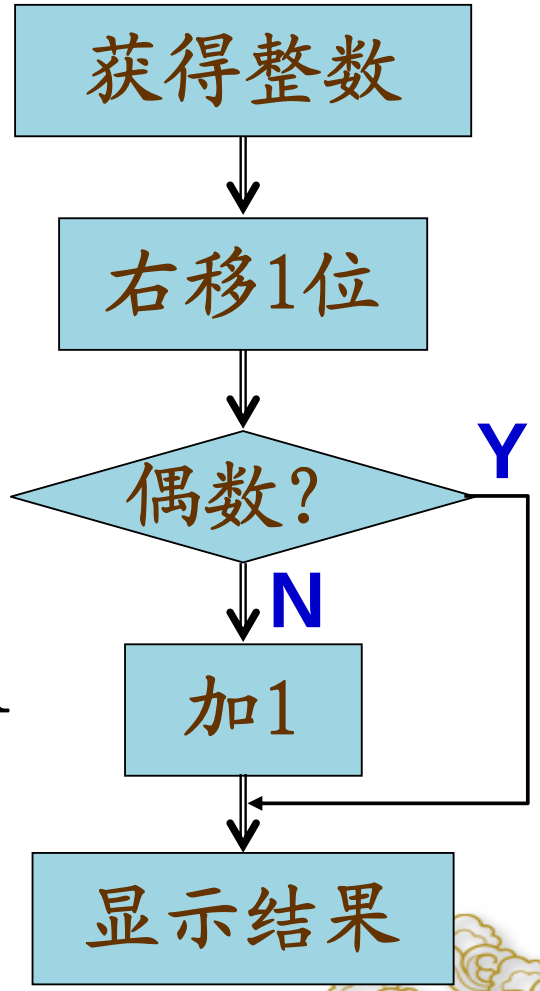
- 判断条件：结果没有进位(借位)（标志CF=0）

  JNC label　;Jump if Not Carry

判断CF=0，选用JNC指令

个数折半程序

```
        mov eax,885
        shr eax,1     ;右移1位
        jnc goeven
        ;CF＝0条件成立，转移
        add eax,1
        ;条件不成立（CF＝1），加1
goeven: call dispuid      ;显示结果
```



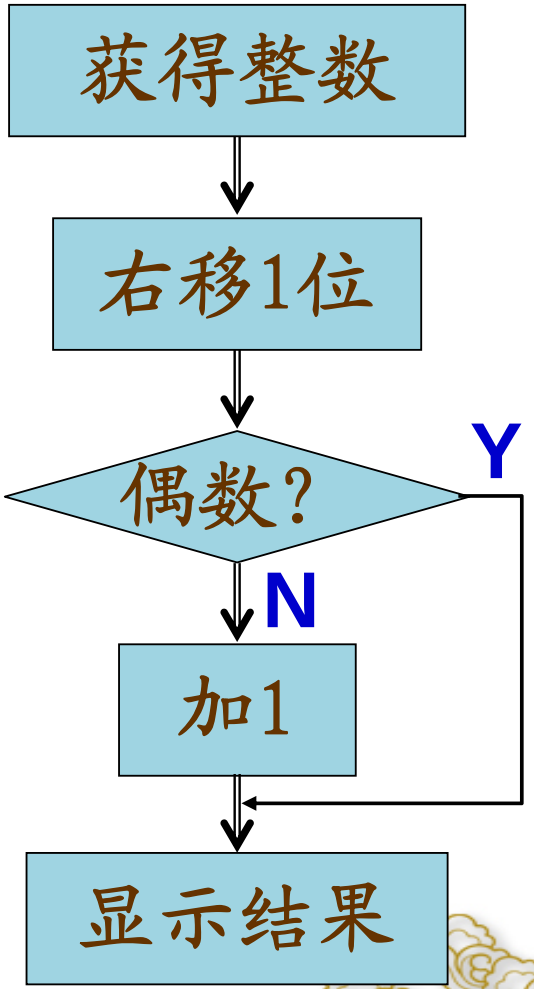
误用高级语言思维

- 习惯使用高级语言if语句

  if(CF标志=1) 加1;

- 于是，选用JC指令

判断CF=1，选用JC指令



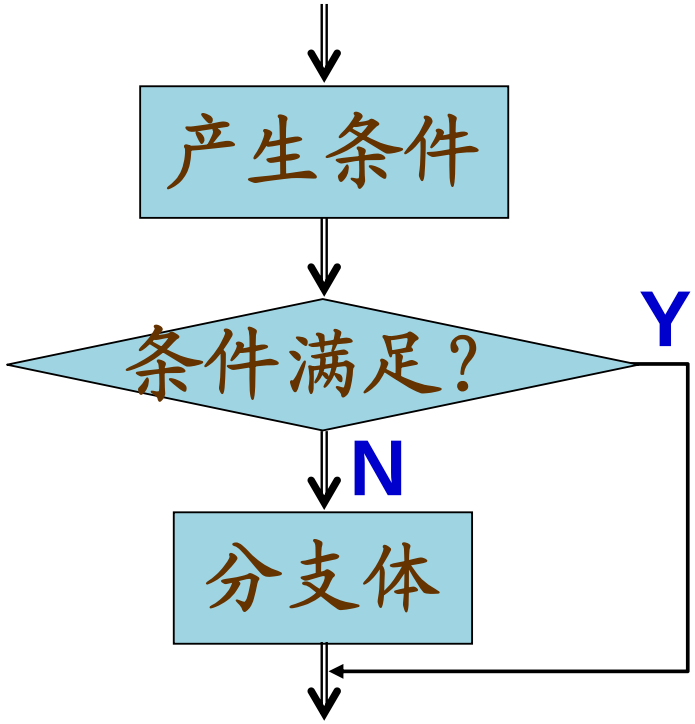
个数折半程序（误用JC）

```
        mov eax,886
        shr eax,1     ;右移1位
        jc goodd
        ;CF＝1条件成立，转移
        jmp goeven        ;CF＝0，转移到显示！
goodd:  add eax,1     ;加1
goeven: call dispuid      ; 显示结果
```

```
jnc goeven
```

本讲总结

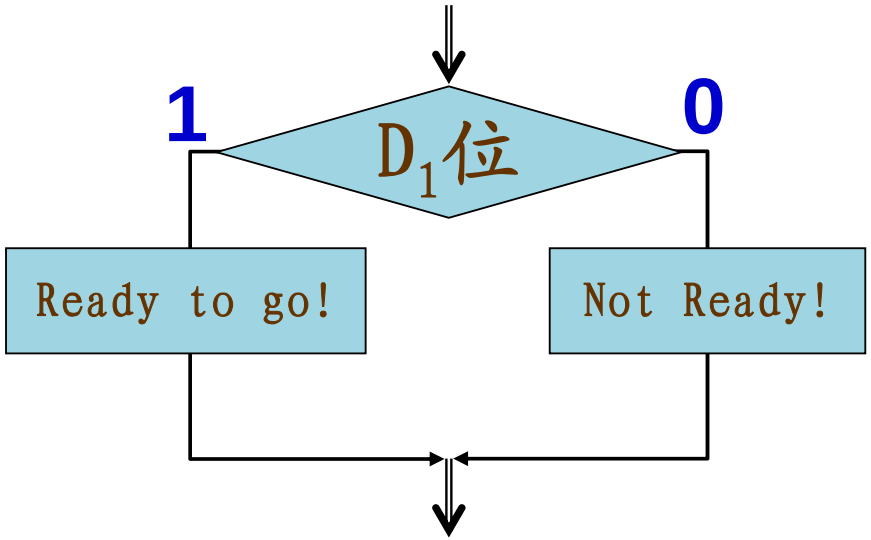
- 分析应用问题
- 考虑产生条件的指令
- 选择最适合的条件转移指令
- 形成分支程序结构



位测试程序

位测试

- 底层程序设计，常需测试某个位（bit）为0或为1
  - 为0，表示一种状态
  - 为1，表示相反状态
- 例如，测试数据D_1位为0或为1
  - 为0，显示：**Not Ready！**
  - 为1，显示：**Ready to go！**




```
no_msg   byte  'Not Ready!',0
yes_msg byte  'Ready to go!',0
```

如何进行位测试

- 使用位操作类指令：逻辑指令、移位指令等
- 典型应用：使用测试指令（TEST）
  - 将要测试位除外的其他位“逻辑与”为0，这样
    - “逻辑与”结果为0，表示测试位为0
    - “逻辑与”结果为1，表示测试位为1
  - 使用零标志条件转移指令（JZ JNZ）进行分支

位测试程序（选用JZ指令）

```
        ;代码段
        mov eax,56h          ;假设一个数据
        test eax,02h         ;测试D1位(D1＝1，其他位为0)
        jz nom               ;D1＝0条件成立，转移
        mov eax,offset yes_msg       ;D1＝1，准备好
        jmp done             ;跳转过另一个分支体!
nom:    mov eax,offset no_msg        ;没有准备好
done:   call dispmsg         ;显示信息
```

位测试程序（选用JNZ指令）

```
        ;代码段
        mov eax,56h          ;假设一个数据
        test eax,02h         ;测试D1位(D1＝1，其他位为0)
        jnz yesm             ;D1＝1条件成立，转移
        mov eax,offset no_msg    ;D1＝0，没有准备好
        jmp done             ;跳转过另一个分支体!
yesm:   mov eax,offset yes_msg   ;准备好
done:   call dispmsg         ;显示信息
```

本讲总结

- 分析应用问题
- 考虑产生条件的指令
- 选择最适合的条件转移指令
- 形成分支程序结构