循环指令

LOOP指令

LOOP label

;功能1：ECX←ECX－1

;功能2：若ECX≠0，转移到LABEL

;否则，顺序执行

相当于 → **DEC ECX**

相当于 → **JNZ label**

DEC ECX
JNZ label

寄存器ECX是默认的计数器

目标地址采用相对短转移

LOOP指令的应用

- **LOOP是循环指令，用于实现减量计数的循环控制**
- 典型应用形式

```
        mov ecx,num     ;设置循环的计数初值num
label:  ...             ;循环体
        loop label      ;ECX减1，未到0继续循环
                        ;到0循环结束，顺序执行
```

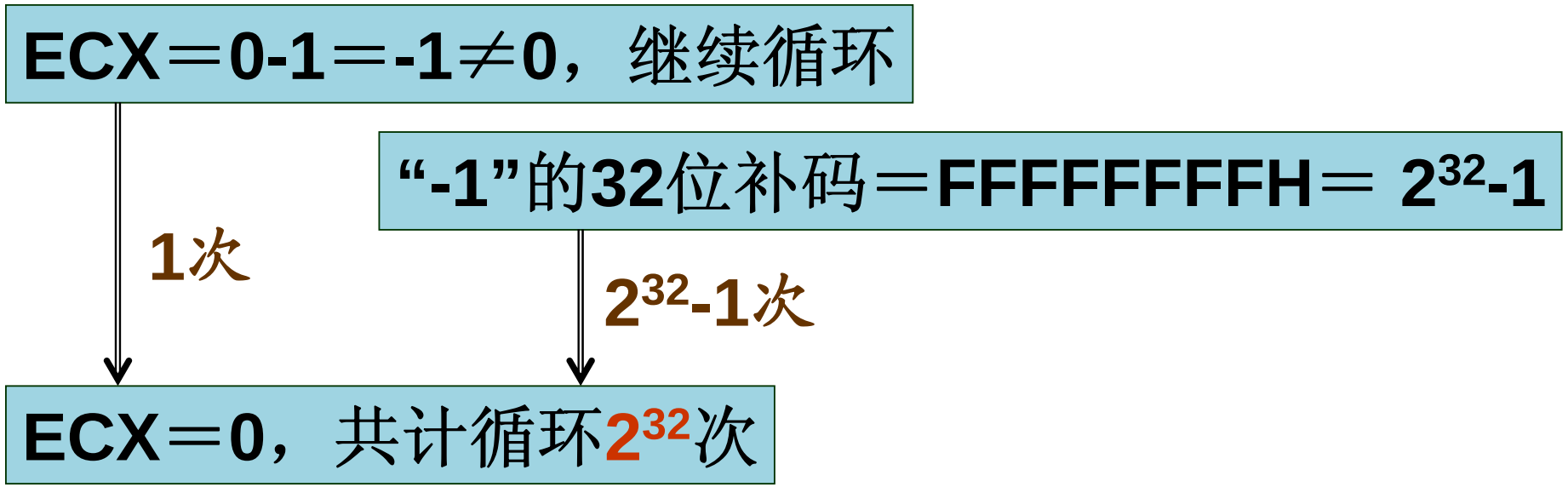
LOOP指令的循环次数

```
        mov ecx,num     ;设置循环的计数初值num
label:  ...             ;循环体
        loop label      ;ECX减1，未到0继续循环
                        ;到0循环结束，顺序执行
```

循环初值	1	2	…	2^{32}-1	0
循环次数	1	2	…	2^{32}-1	2^{32}

LOOP指令先减1后判断

```
        mov ecx,0          ;设置循环的计数初值
label:  loop label         ;ECX减1，未到0继续循环
```



JECXZ指令

- 为避免计数初值为0可能导致的程序错误
- 设计JECXZ指令

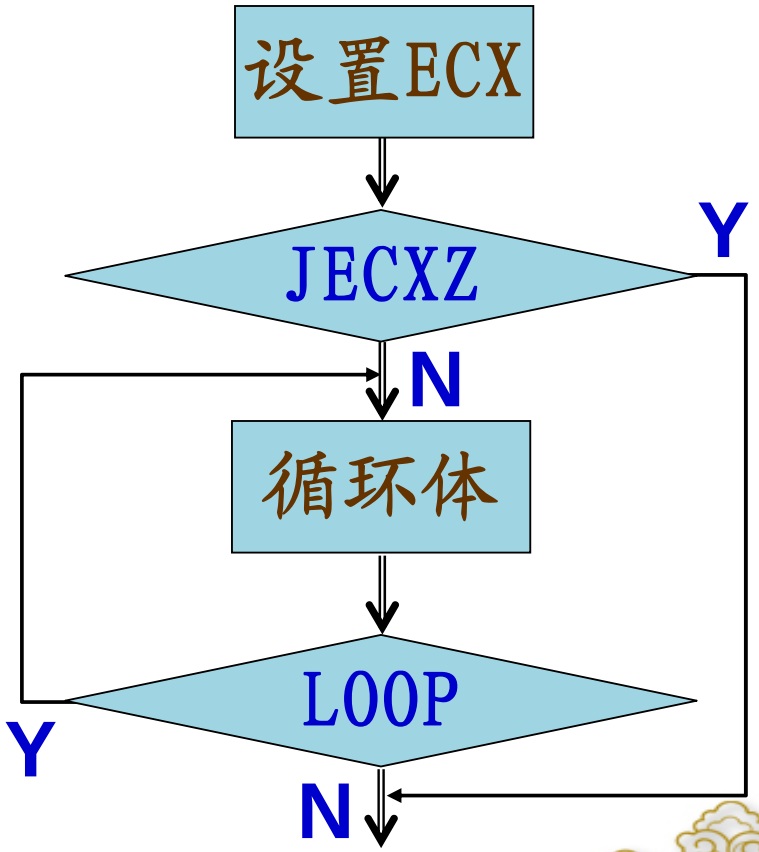
JECXZ label

;ECX＝0，转移到label

;否则，顺序执行

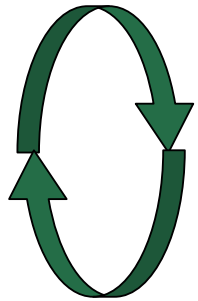
CMP ECX,0

JZ label



本讲总结

- LOOP指令，用于实现减量计数的循环控制
  - 设置ECX等于计数初值
  - ECX通常不应为0（实际表达2^{32}次循环）
- JECXZ指令用于跳过ECX=0的情况



数组求和程序

数组求和

- 数组元素逐个相加（不考虑溢出）

```
                ;数据段
array     dword 136,-138,133,130,-161   ;数组
sum       dword ?                       ;结果变量
```

- 元素逐个相加，作为循环体
- 数组元素个数已知，可用**LOOP**指令控制计数

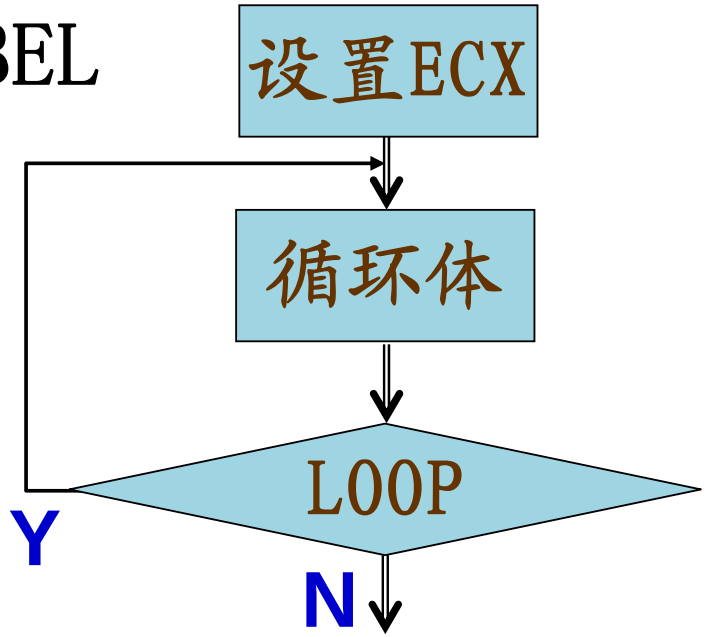
LOOP指令

LOOP label

;功能1：ECX←ECX－1

;功能2：若ECX≠0，转移到LABEL

;否则，顺序执行



LOOP指令的应用

- 循环指令**LOOP** 可用于实现减量计数的循环控制
- 典型应用形式

```
        mov ecx,num      ;设置循环的计数初值num
label:  ...              ;循环体
        loop label       ;ECX减1，未到0继续循环
                         ;到0循环结束，顺序执行
```

LOOP指令控制数组求和

```
        mov ecx,lengthof array  ;ECX＝数组元素个数
        xor eax,eax             ;求和初值为0
        …                       ;指向首个元素
again:  add eax, …              ;求和
        …                       ;指向下一个数组元素
        loop again
        mov sum,eax             ;保存结果
```

LOOP指令使用ECX计数器实现计数控制循环

逐个寻址数组元素

- 寻址存储器内的操作数，使用存储器寻址
- 存储器的直接寻址适用于访问单个变量
  - 但不方便改变地址

    故不适合访问数组元素
- 存储器的寄存器间接、相对和变址都使用寄存器
  - 通过修改寄存器内容改变地址，

    可方便地访问数组元素

寄存器间接寻址访问数组元素

```
        mov ebx, offset array    ;指向首个元素
again:  add eax, [ebx]           ;求和
        add ebx, 4               ;指向下一个数组元素
```

EBX赋值数组首个元素的地址

地址增量4，因为
数组元素是32位、占4个字节地址空间

寄存器相对寻址访问数组元素

```
        mov ebx, 0                  ;指向首个元素
again:  add eax, array[ebx]         ;求和
        add ebx, 4                  ;指向下一个数组元素
```

EBX赋值距离数组首个元素的位移量

位移量增量4，因为
数组元素是32位、占4个字节地址空间

寄存器（带比例）变址寻址访问数组元素

```
        mov ebx, 0                          ;指向首个元素
again:  add eax,array[ebx*(type array)]     ;求和
        add ebx, 1                          ;指向下一个数组元素
```

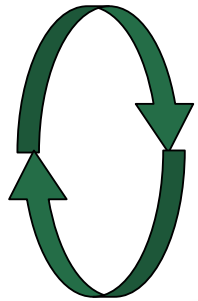
EBX赋值首个数组元素

数组元素是4个字节数据
故，type array = 4

元素个数增量1，
与数组元素的类型无关

数组求和程序

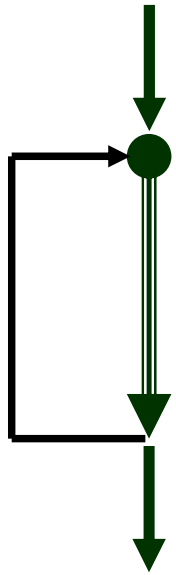
```
        mov ecx,lengthof array  ;ECX＝数组元素个数
        xor eax,eax             ;求和初值为0
        mov ebx,eax             ;数组指针为0
again:  add eax,array[ebx*(type array)]    ;求和
        inc ebx                 ;指向下一个数组元素
        loop again
        mov sum,eax             ;保存结果
```



循环程序结构

循环程序结构通常有3个组成部分

- 循环初始——为开始循环准备必要的条件
  - 如设置循环次数、循环体需要的初始值等
- 循环体——重复执行的程序代码
  - 其中包括对循环条件的修改等
- 循环控制——判断循环条件是否成立
  - 决定是否继续循环



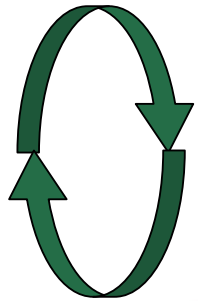
数组求和程序

```
        mov ecx,lengthof array
        xor eax,eax
        mov ebx,eax
again:  add eax,array[ebx*(type array)]
        inc ebx
        loop again
        mov sum,eax
```

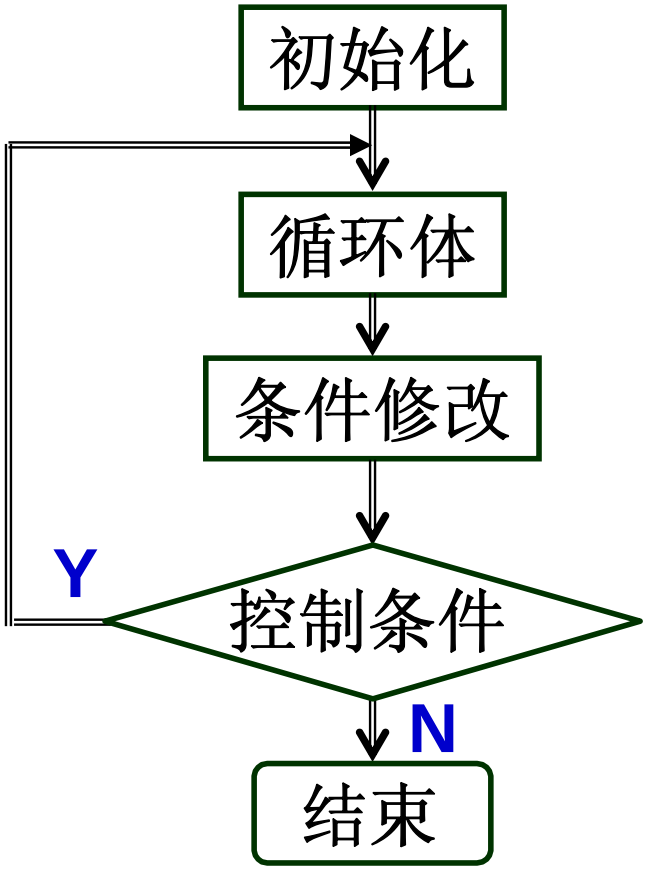
循环初始

循环体

循环控制



循环程序流程图

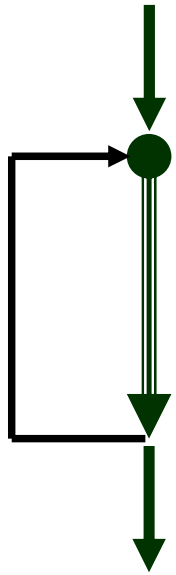


循环初始

循环体
(含条件修改)

{ 计数控制循环
  条件控制循环

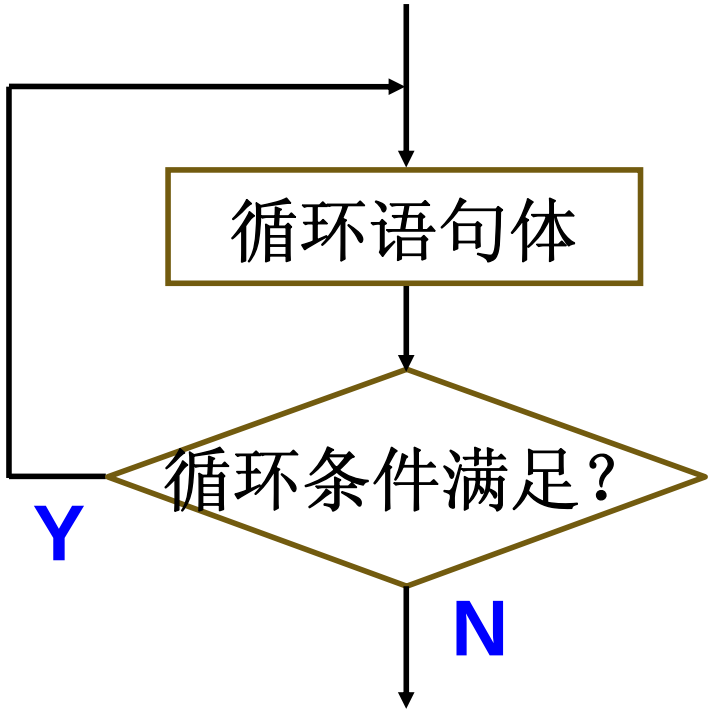
循环程序结构
的关键是如何控制循环



计数控制循环

- 通过次数控制循环
  - 计数可以减量进行，即减到0结束
  - 计数可以增量进行，即达到规定值结束
  - 类似高级语言的for语句
- 利用LOOP指令属于减量计数的循环控制
- 常见是“先循环、后判断”循环结构
  - 对应高级语言的do语句

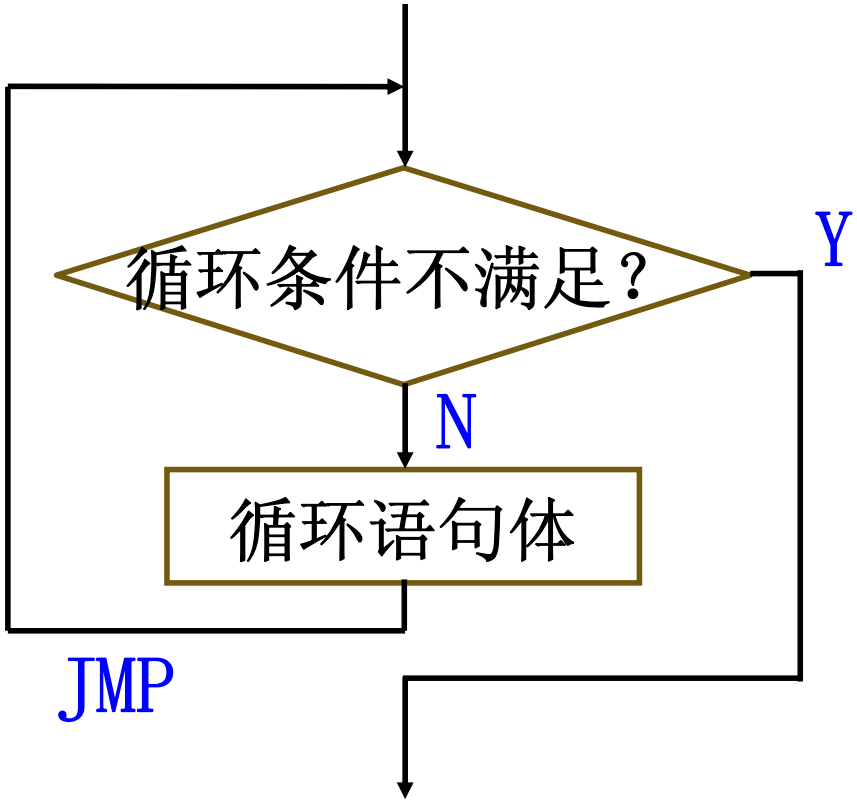
“先循环、后判断”结构



条件控制循环

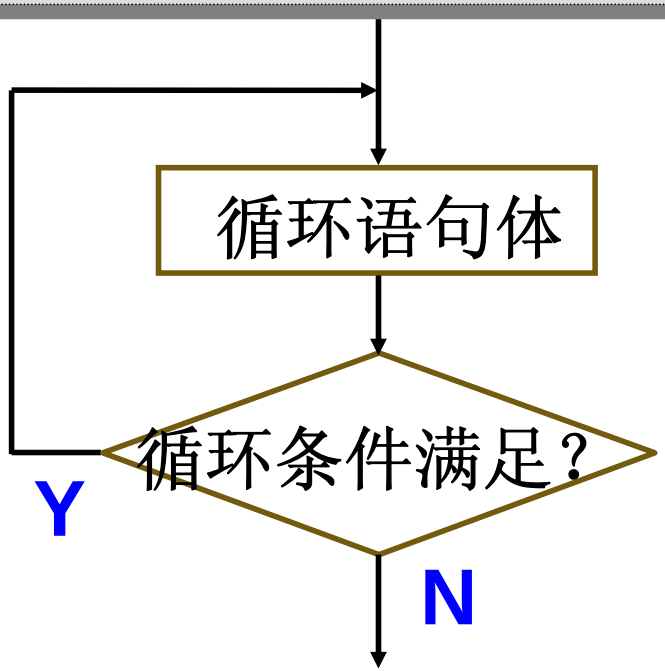
- 根据条件决定是否进行循环
  - 使用比较、测试等指令设置状态标志、产生条件
  - 使用条件转移指令实现循环控制
  - 常需要使用无条件转移指令配合实现循环
- 多见是“先判断、后循环”循环结构
  - 对应高级语言的while语句

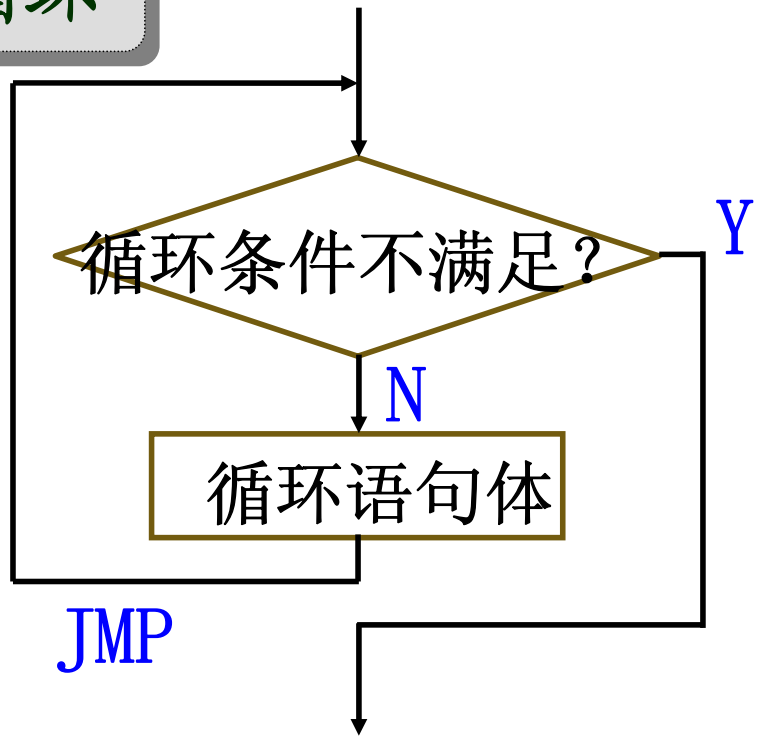
“先判断、后循环”结构



本讲总结

LOOP指令仅实现了简单的计数循环





条件转移指令才能实现复杂的循环控制