



东北大学  
Northeastern University

# 汇编语言程序设计

主讲：东北大学计算机学院 刘松冉

# 第七章 循环结构程序设计

- 一. 循环结构程序的提出
- 二. 循环结构程序的组成
- 三. 循环控制指令
- 四. 数据串操作指令
- 五. 循环程序的控制方法
- 六. 单重循环程序举例
- 七. 多重循环程序设计
- 八. 循环程序的效率



## 一. 循环结构程序的提出

---

► 在许多实际应用问题中，经常遇到某一段操作需反复进行的情况。

例7.1 设内存DATA1和DATA2开始分别存放50个无符号字数据，编制程序将两个存储区对应字数据求和并存入SUM开始的单元。

利用顺序结构编写求和程序，首先设置源操作数和目的操作数指针，设置存放结果指针，接着从源指针指出的字单元取出数据，与目的指针指出的字单元的数据相加，和存入结果的字单元，然后，修改各地址指针，使之指向下一个字数据，求和，保存结果，...直至50对字数据全部求和后，程序结束。

# 一. 循环结构程序的提出

```
1:;*****EXAM7.1.1*****  
2:SSEG SEGMENT STACK  
3:STACK DB 50 DUP(0)  
4:SSEG ENDS  
5:DATA SEGMENT  
6:DATA1 DW 15H,36H,45H,27BH,...  
7:DATA2 DW 174H,03H,5BCH,390H,26H,...  
8:SUM DW 50 DUP(0)  
9:DATA ENDS  
10:CSEG SEGMENT  
11: ASSUME CS:CSEG,DS:DATA  
12: ASSUME SS:SSEG  
13:START: MOV AX,DATA  
14: MOV DS,AX  
15: MOV AX,SSEG  
16: MOV SS,AX  
17: MOV SP,SIZE STACK  
18: LEA BX,DATA1  
19: LEA SI,DATA2  
20: LEA DI,SUM  
21: MOV AX,[SI]  
22: ADD AX,[BX]  
23: MOV [DI],AX  
24: ADD BX,2  
25: ADD SI,2  
26: ADD DI,2  
27: MOV AX,[SI]  
28: ADD AX,[BX]  
29: MOV [DI],AX  
30: ADD BX,2  
31: ADD SI,2  
32: ADD DI,2  
315: MOV AX,[SI]  
316: ADD AX,[BX]  
317: MOV [DI],AX  
318: MOV AH,4CH  
319: INT 21H  
320:CSEG ENDS  
321: END START
```

顺序结构设计

# 一. 循环结构程序的提出

1:;\*\*\*\*\*EXAM7.1.1\*\*\*\*\*

```
2:SSEG SEGMENT STACK
3:STACK DB      50 DUP(0)
4:SSEG ENDS
5:DATA SEGMENT
6:DATA1 DW      15H,36H,45H,27BH, ...
7:DATA2 DW      174H,03H,5BCH,390H,26H, ...
8:SUM DW      50 DUP(0)
9:DATA ENDS
10:CSEG SEGMENT
11:ASSUME CS:CSEG,DS:DATA
12:ASSUME SS:SSEG
13:START: MOV AX,DATA
14:MOV DS,AX
15:MOV AX,SSEG
16:MOV SS,AX
17:MOV SP,SIZE STACK
18:LEA BX,DATA1
19:LEA SI,DATA2
20:LEA DI,SUM
```

```
21:    MOV AX,[SI]
22:    ADD AX,[BX]
23:    MOV [DI],AX
24:    ADD BX,2
25:    ADD SI,2
26:    ADD DI,2
27:    MOV AX,[SI]
28:    ADD AX,[BX]
29:    MOV [DI],AX
30:    ADD BX,2
31:    ADD SI,2
32:    ADD DI,2
:
315:   MOV AX,[SI]
316:   ADD AX,[BX]
317:   MOV [DI],AX
318:   MOV AH,4CH
319:   INT 21H
320:CSEG ENDS
321:   END START
```

代码完全相同

- 编写繁琐
- 代码体积大
- 浪费内存

# 一. 循环结构程序的提出



1:;\*\*\*\*\*EXAM7.1.2\*\*\*\*\*

```
2:SSEG SEGMENT STACK
3:STACK DB      50 DUP(0)
4:SSEG ENDS
5:DATA SEGMENT
6:DATA1 DW      15H,36H,45H,27BH, ...
7:DATA2 DW      174H,03H,5BCH,390H,26H, ...
8:SUM DW      50 DUP(0)
9:DATA ENDS
10:CSEG SEGMENT
11:    ASSUME CS:CSEG,DS:DATA
12:    ASSUME SS:SSEG
13:START: MOV AX,DATA
14:    MOV DS,AX
15:    MOV AX,SSEG
16:    MOV SS,AX
17:    MOV SP,SIZE STACK
18:    LEA BX,DATA1
19:    LEA SI,DATA2
20:    LEA DI,SUM
```

```
21:        MOV CX,50
22:AGAIN:   MOV AX,[SI]
23:        ADD AX,[BX]
24:        MOV [DI],AX
25:        ADD BX,2
26:        ADD SI,2
27:        ADD DI,2
28:        DEC CX
29:        JNZ AGAIN
30:        MOV AH,4CH
31:        INT 21H
```

## 二. 循环结构程序的组成

### 1. 初始化部分

初始化部分是为循环程序做好准备、以保证循环程序能够正常运行的部分。这一部分往往在循环程序的开头，逻辑上先从这部分开始执行。这一部分一般设置地址指针的初值、计数器的初值及程序中用到的某些寄存器和某些内存单元的初值，初始化部分只执行一遍。

### 2. 工作部分

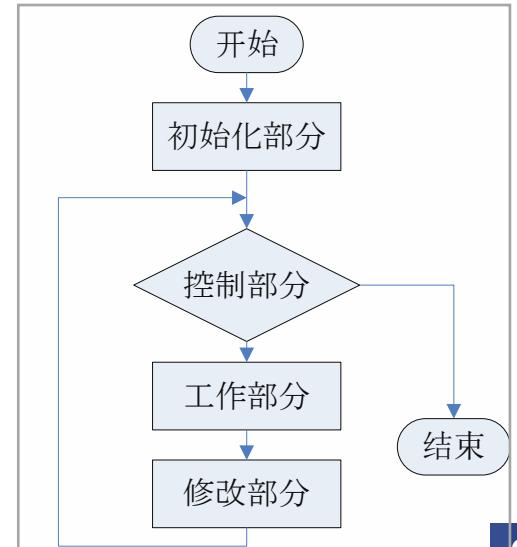
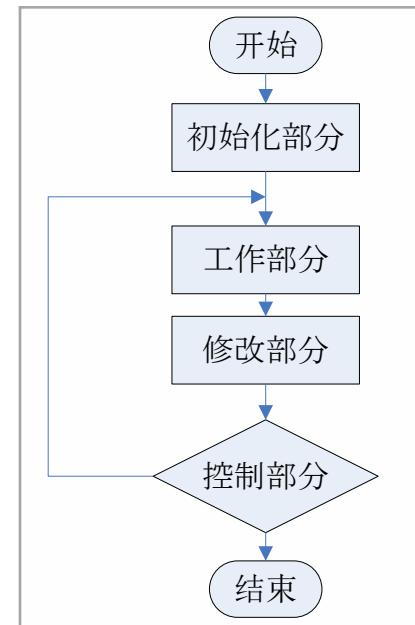
工作部分是循环程序的主体，它从初始化部分设置的初值开始，动态地反复执行相同的操作。这一部分完成循环程序所要实现的功能，即需要重复进行的工作。

### 3. 修改部分

这一部分与工作部分协调配合，对参加运算的数据或地址指针以及结果单元的地址指针进行恰当的修改，以保证循环程序在循环的过程中，每次循环都能正确地得到参加运算的数据，并正确地存放运算结果。

### 4. 控制部分

控制部分是保证循环程序按预定的循环次数或某种预定的条件正常循环，且能控制循环程序正常退出循环的部分。



### 三. 循环控制指令

---

- |▶ 1. LOOP                  重复控制
- 2. LOOPE/LOOPZ      条件重复控制
- 3. LOOPNE/LOOPNZ 条件重复控制
- 4. JCXZ                  CX为0转

### 三. 循环控制指令

---

#### |▶ 1. LOOP 重复控制

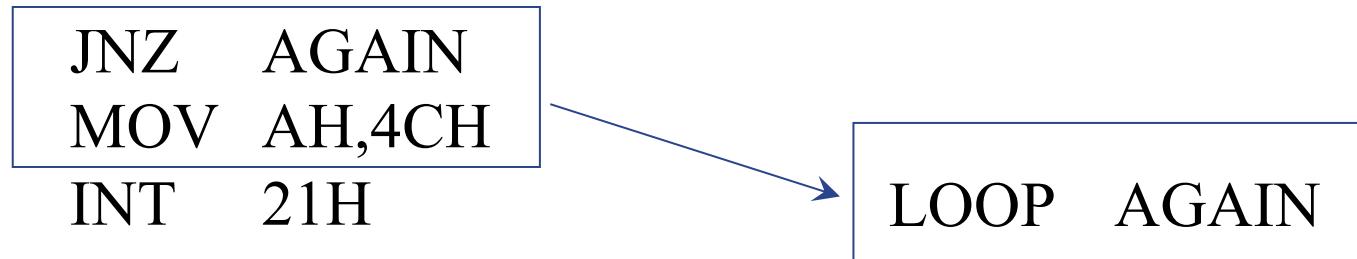
- **指令汇编格式:** LOOP shortlabel
- **操作:** 计数寄存器CX减1，如果新的CX值不为0，则转向shortlabel指定的循环入口执行；若CX=0，则退出循环，执行后续指令。
- **受影响的状态标志位:** 没有

### 三. 循环控制指令

#### ► 1. LOOP 重复控制

- 例子:

```
21:      MOV CX,50
22:AGAIN: MOV AX,[SI]
23:      ADD AX,[BX]
24:      MOV [DI],AX
25:      ADD BX,2
26:      ADD SI,2
27:      ADD DI,2
28:      DEC CX
29:      JNZ AGAIN
30:      MOV AH,4CH
31:      INT 21H
```



### 三. 循环控制指令

---

#### ► 2. LOOPE/LOOPZ 条件重复控制

- **指令汇编格式:** LOOPE/LOOPZ shortlabel
- **操作:** 计数寄存器CX减1，如果零标志位 ZF = 1 且 CX ≠ 0 时，转向shortlabel 指定的循环入口执行，如果 ZF = 0 或 CX = 0 则执行后续指令。
- **受影响的状态标志位:** 没有

### 三. 循环控制指令

#### 2. LOOPE/LOOPZ 条件重复控制

- 例子：比较两个数据串是否相等时。由第一个数据开始，依次比较，若发现比较过程中某两个数据已不相等，不必再反复比较下去而需中途退出循环。

```
DSEG SEGMENT
STR1 DB 'COMPUTER'
STR2 DB 'COMPRESS'
DSEG SEGMENT
ASSUME S:CSEG,DS:DSEG
START: MOV AX,DSEG
        MOV DS,AX
        LEA SI,STR1
        LEA DI,STR2
        MOV CX,8
AGAIN:  MOV AL,[SI]
        MOV AH,[DI]
        INC SI
        INC DI
        CMP AL,AH
        JNE STOP
        LOOP AGAIN
        .
        .
CSEG ENDS
END     START
```

STR1:	'C'
	'O'
	'M'
	'P'
	'U'
	'T'
	'E'
	'R'
STR2:	'C'
	'O'
	'M'
	'P'
	'R'
	'E'
	'S'
	'S'

### 三. 循环控制指令

---

#### ► 3. LOOPNE/LOOPNZ 条件重复控制

- **指令汇编格式:** LOOPNE/LOOPNZ shortlabel
- **操作:** 计数寄存器CX减1，如果零标志位 ZF = 0 且 CX ≠ 0 时，转向shortlabel 指定的循环入口执行，如果 ZF = 1 或 CX = 0 则执行后续指令。
- **受影响的状态标志位:** 没有

### 三. 循环控制指令

#### 3. LOOPNE/LOOPNZ 条件重复控制

- 例子：在若干个数据中查找一个等于给定值的数据，并不一定要查找完所有数据才能结束，当在中间找到这个数据时，就可以中途退出。此时可用此指令控制循环结构。

```
DSEG SEGMENT
STR1: DB      'COMPUTER'
CHR:  DB      'M'

DSEG SEGMENT
ASSUME CS:CSEG,DS:DSEG
START: MOV    AX,DSEG
       MOV    DS,AX
       LEA    SI,STR1-1
       MOV    CX,8
       MOV    AL,CHR
AGAIN: INC    SI
       CMP    AL,[SI]
       JE     FOUND
       LOOP   AGAIN

        . . .
        . . .

CSEG ENDS
END   START
```

STR1:	'C'
	'O'
	'M'
	'P'
	'U'
	'T'
	'E'
	'R'
CHR:	'M'

```
LOOPNE AGAIN
        JE     FOUND
```

```
        JE     FOUND
        LOOP   AGAIN
```

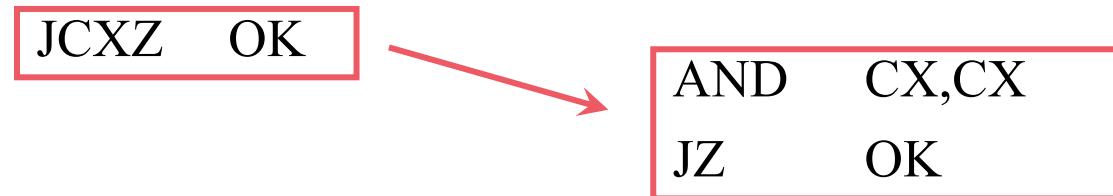
FOUNDED:

### 三. 循环控制指令

---

#### ► 4. JCXZ CX为0跳转 (Jump if CX is Zero)

- **指令汇编格式:** JCXZ shortlabel
- **操作:** 如果计数寄存器CX为0，转向shortlabel指出的程序入口执行，如果 CX≠0，则执行后续指令。
- **受影响的状态标志位:** 没有



## 四. 数据串操作指令

---

- |▶ 1. MOVS/MOVSB/MOVSW
- 2. LODS/LODSB/LODSW
- 3. STOS/STOSB/STOSW
- 4. CMPS CMPSB CMPSW
- 5. SCAS/SCASB/SCASW

## 四. 数据串操作指令

---

### ► 1. MOVS/MOVSB/MOVSW – 数据串传送

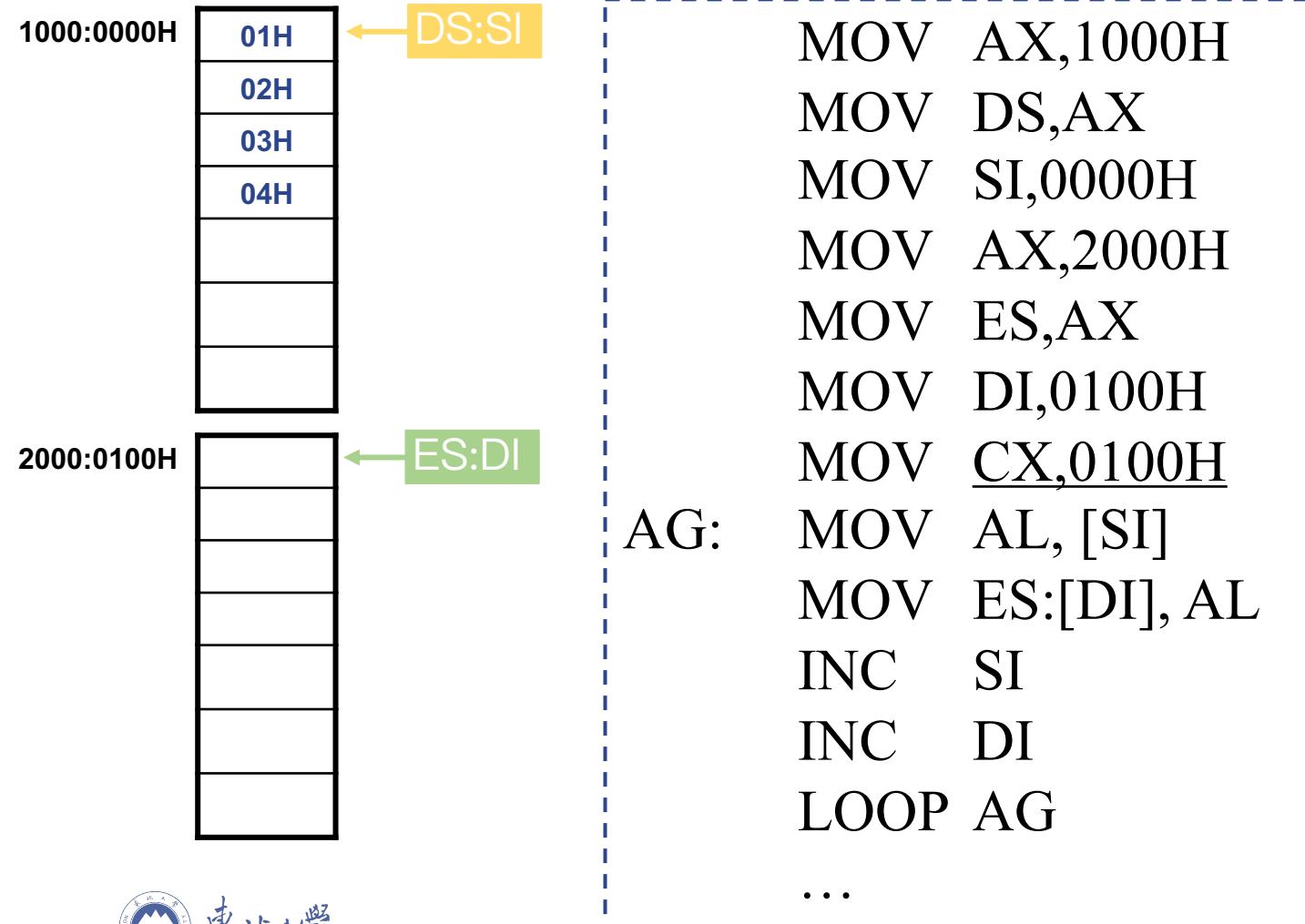
- **指令汇编格式**：MOVS/MOVSB/MOVSW [dstr，sstr]
- **操作**：将DS段SI指出的字节（或字）数据传送到ES段DI指出的单元，然后根据标志位DF的情况和操作数的类型(字节或字)修改SI，DI的地址指针。  
$$(ES:DI) \leftarrow (DS:SI)$$

如果DF=0，则： $SI \leftarrow SI + \text{DELTA}$ ,  $DI \leftarrow DI + \text{DELTA}$

否则： $SI \leftarrow SI - \text{DELTA}$ ,  $DI \leftarrow DI - \text{DELTA}$
- **受影响的状态标志位**：没有
- **说明**：
  - 1) MOVSB，MOVSW明确指出串的类型是字节串还是字串，上面的 $\text{DELTA}$ 在字节串时为1，字串时为2。
  - 2) 使用MOVS指令时，其后可以写目标地址和源地址，目标地址一定为ES:[DI]，而源地址可以是DS:[SI]，CS:[SI]，ES:[SI]和SS:[SI]，即源地址可在非隐含段。
  - 3) 用此指令可以实现存储器 → 存储器的数据传送，这是一般mov指令不允许的。

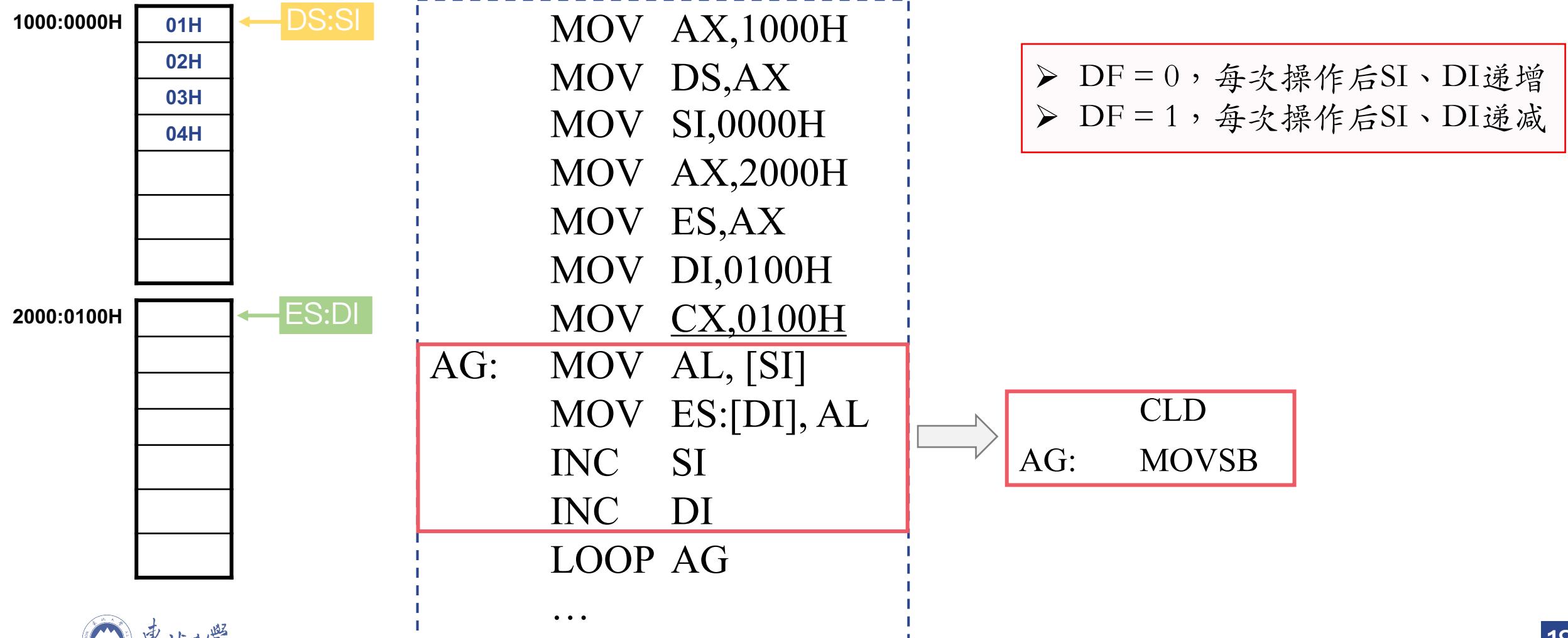
## 四. 数据串操作指令

### 1. MOVS/MOVSB/MOVSW – 数据串传送



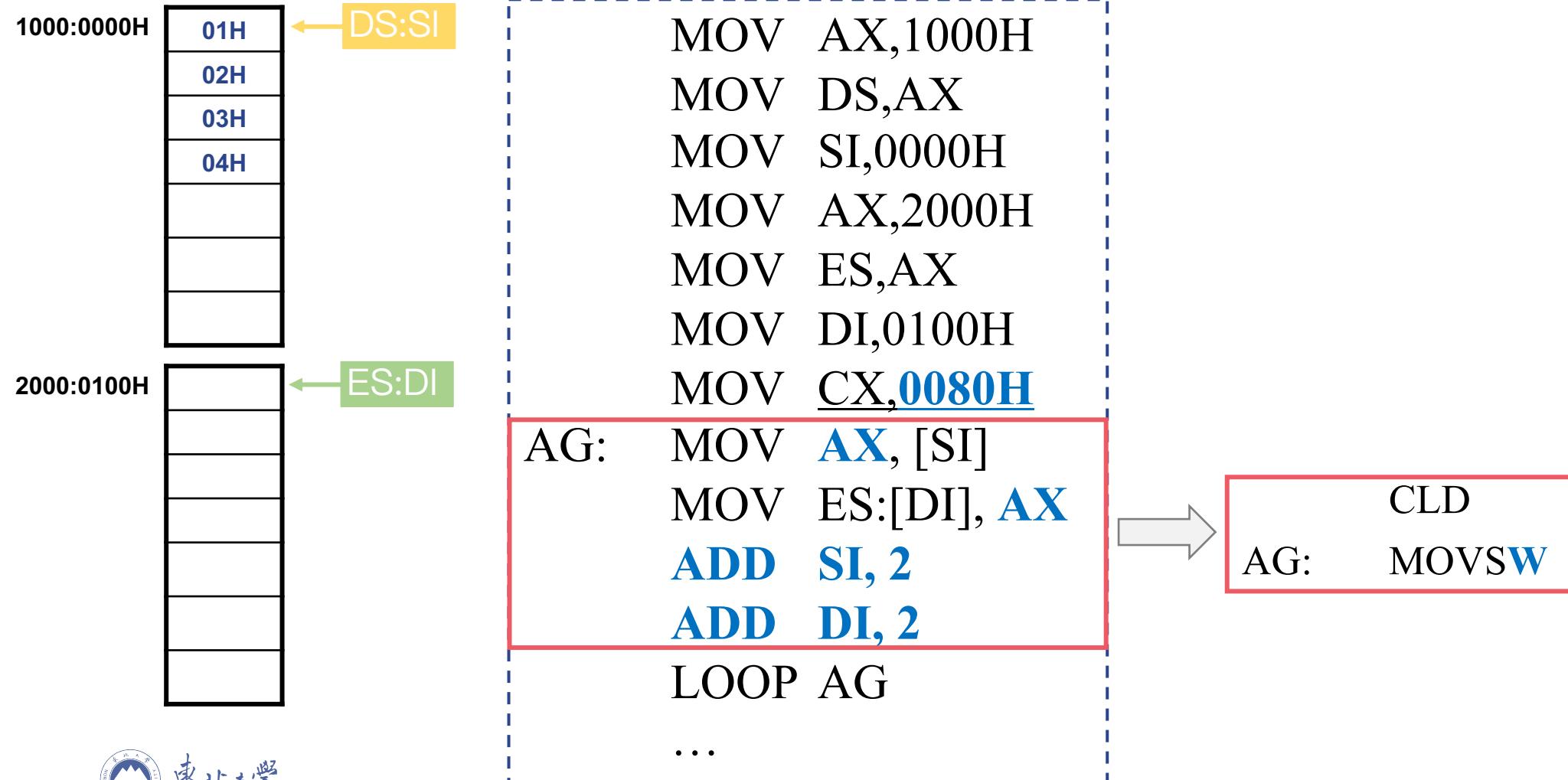
## 四. 数据串操作指令

### 1. MOVS/MOVSB/MOVSW – 数据串传送



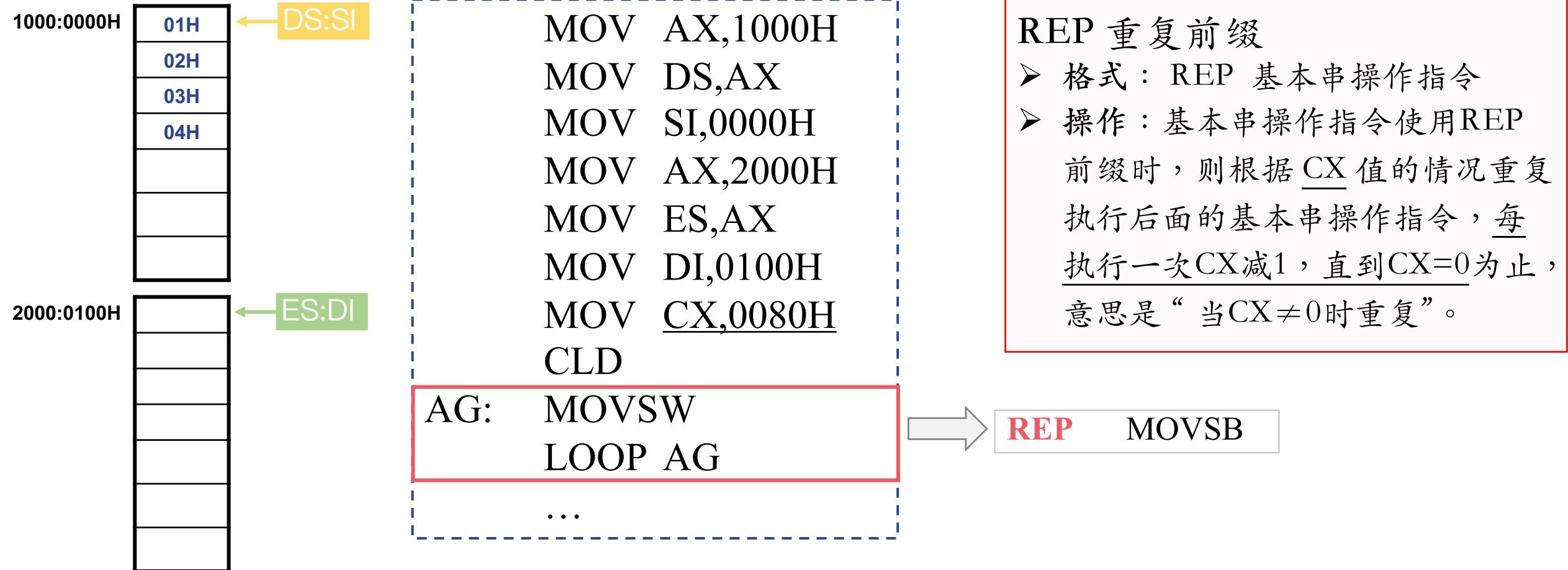
## 四. 数据串操作指令

### 1. MOVS/MOVSB/MOVSW – 数据串传送



## 四. 数据串操作指令

### 1. MOVS/MOVSB/MOVSW – 数据串传送

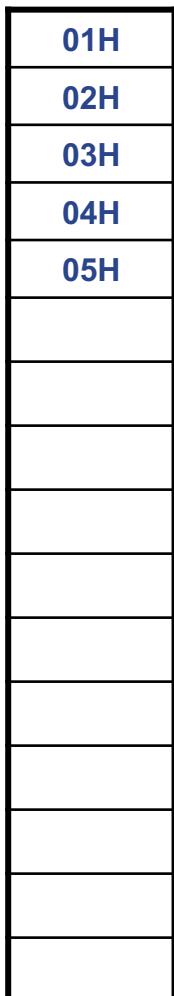


## 四. 数据串操作指令

### ► 1. MOVS/MOVSB/MOVSW – REP重复前缀

- 例子：将1000:0000H开始的100H个数据顺序下移一个位置。

1000:0000H



1000:00FFH

1000:0100H

## 四. 数据串操作指令

## ► 1. MOVS/MOVSB/MOVSW – REP重复前缀

- 例子：将1000:0000H开始的100H个数据顺序下移一个位置。

**MOV AX, 1000H**

**MOV DS, AX**

**MOV ES, AX**

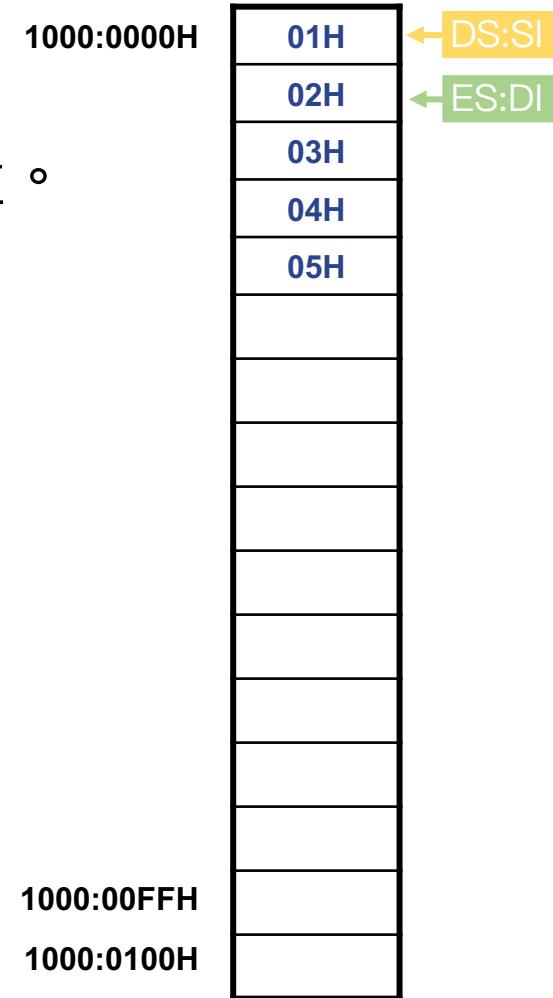
**MOV SI, 0000H**

**MOV DI, 0001H**

MOV CX, 0100H

CLD

→ REP MOVS B



## Iteration 0

- CX = 0100H



## 四. 数据串操作指令

### 1. MOVS/MOVSB/MOVSW – REP重复前缀

- 例子：将1000:0000H开始的100H个数据顺序下移一个位置。

MOV AX, 1000H

MOV DS, AX

MOV ES, AX

MOV SI, 0000H

MOV DI, 0001H

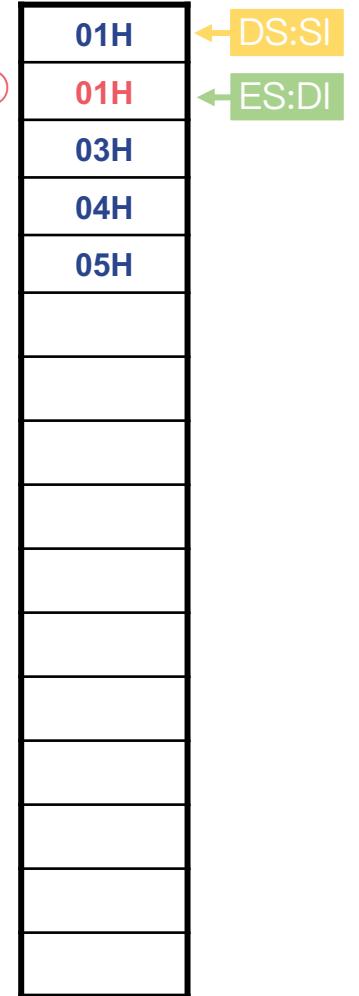
MOV CX, 0100H

CLD

→ REP MOVSB

1000:0000H

①



1000:00FFH

1000:0100H

Iteration 0  
– CX = 0100H

## 四. 数据串操作指令

### 1. MOVS/MOVSB/MOVSW – REP重复前缀

- 例子：将1000:0000H开始的100H个数据顺序下移一个位置。

MOV AX, 1000H

MOV DS, AX

MOV ES, AX

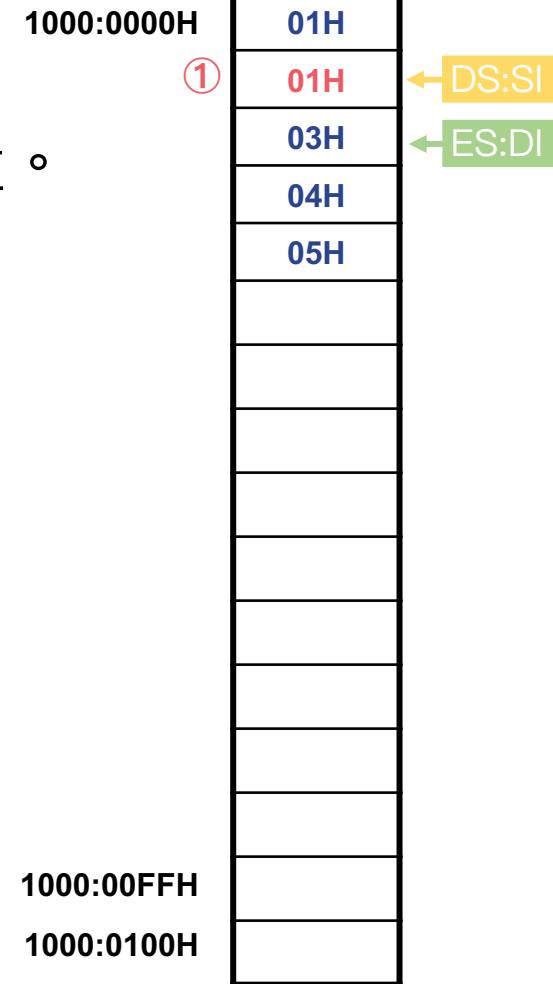
MOV SI, 0000H

MOV DI, 0001H

MOV CX, 0100H

CLD

→ REP MOVSB



Iteration 0  
– CX = 0100H

## 四. 数据串操作指令

### 1. MOVS/MOVSB/MOVSW – REP重复前缀

- 例子：将1000:0000H开始的100H个数据顺序下移一个位置。

MOV AX, 1000H

MOV DS, AX

MOV ES, AX

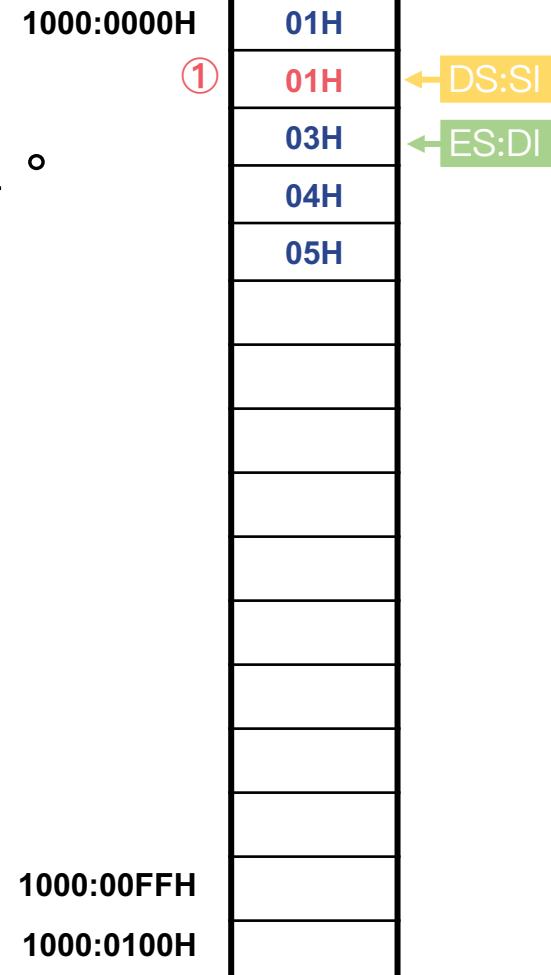
MOV SI, 0000H

MOV DI, 0001H

MOV CX, 0100H

CLD

→ REP MOVSB



③ Iteration 1  
– CX = 00FFH

## 四. 数据串操作指令

## ► 1. MOVS/MOVSB/MOVSW – REP重复前缀

- 例子：将1000:0000H开始的100H个数据顺序下移一个位置。

MOV AX, 1000H

**MOV DS, AX**

**MOV ES, AX**

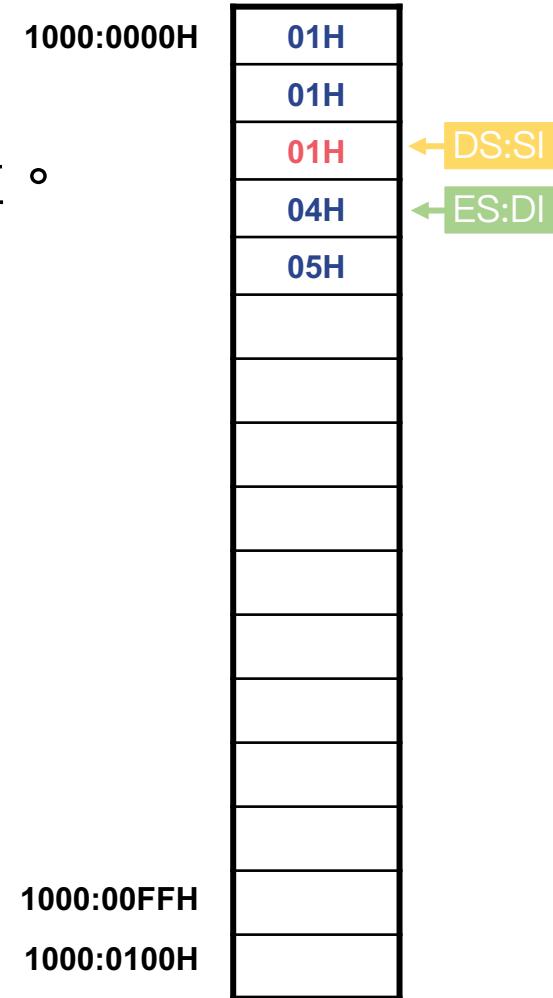
**MOV SI, 0000H**

**MOV DI, 0001H**

MOV CX, 0100H

CLD

→ REP MOVS B



## Iteration 2

- CX = 00FEH



## 四. 数据串操作指令

### 1. MOVS/MOVSB/MOVSW – REP重复前缀

- 例子：将1000:0000H开始的100H个数据顺序下移一个位置。

MOV AX, 1000H

MOV DS, AX

MOV ES, AX

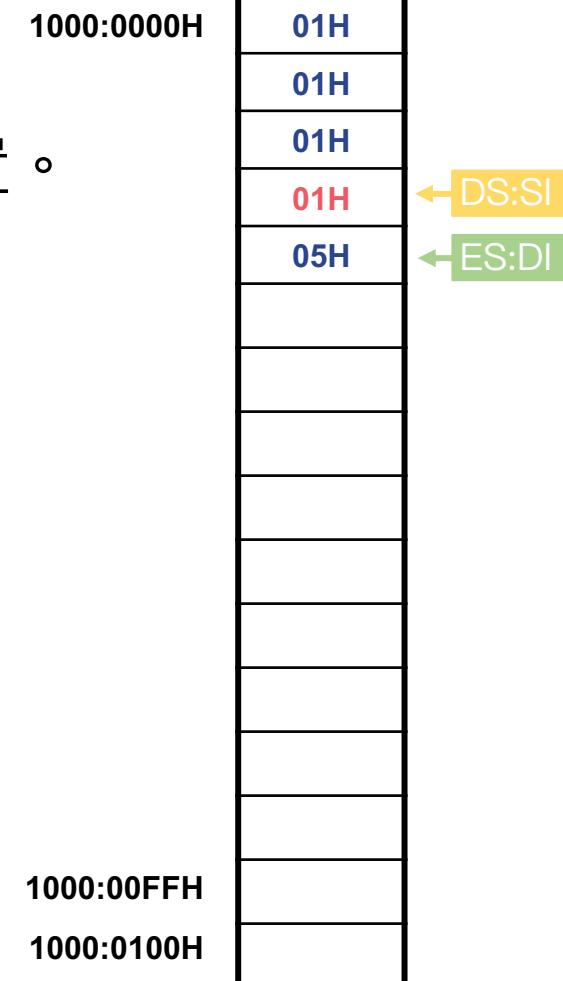
MOV SI, 0000H

MOV DI, 0001H

MOV CX, 0100H

CLD

→ REP MOVSB



Iteration 3  
– CX = 00FDH

## 四. 数据串操作指令

### 1. MOVS/MOVSB/MOVSW – REP重复前缀

- 例子：将1000:0000H开始的100H个数据顺序下移一个位置。

```
MOV AX, 1000H
```

```
MOV DS, AX
```

```
MOV ES, AX
```

```
MOV SI, 0000H
```

```
MOV DI, 0001H
```

```
MOV CX, 0100H
```

```
CLD
```

```
REP MOVS
```

```
MOV SI, 00FFH
```

```
MOV DI, 0100H
```

```
MOV CX, 0100H
```

```
STD
```

```
REP MOVSB
```

1000:0000H

01H  
02H  
03H  
04H  
05H

DS:SI  
ES:DI

1000:00FFH

1000:0100H

## 四. 数据串操作指令

---

### ► 2. LODS/LODSB/LODSW – 取数据串

- **指令汇编格式**： LODS/LODSB/LODSW [sstr]
- **操作**：将DS段SI指出的字节（或字）数据加载到累加器AL（或AX）中，然后修改SI的地址指针。  
$$AL \text{ (或) } AX \leftarrow (DS:SI)$$

如果DF=0，则： $SI \leftarrow SI + \text{DELTA}$

否则： $SI \leftarrow SI - \text{DELTA}$
- **受影响的状态标志位**：没有
- **说明**：
  - 1) 此条指令一般不加重复前缀，因为加入重复前缀之后累加器中新加载的内容不断取代原来的结果，将只留下最后一个加载的元素。

## 四. 数据串操作指令

---

### ► 3. STOS/STOSB/STOSW — 存数据串

- **指令汇编格式**： STOS/STOSB/STOSW [dstr]
- **操作**：将AL（目标串为字节串时）或AX（目标串为字串时）的内容存储到ES段DI指出的地址单元中，然后修改DI的地址指针。  
 $(ES:DI) \leftarrow AL(\text{或} AX)$   
如果DF=0，则： $DI \leftarrow DI + \text{DELTA}$   
否则： $DI \leftarrow DI - \text{DELTA}$
- **受影响的状态标志位**：没有
- **说明**：
  - 1) 借助STOS 指令使用重复前缀可以方便地用一个常数对一块存储区初始化。

## 四. 数据串操作指令

### 3. STOS/STOSB/STOSW — 存数据串

- 例1：用0720H填满B800:0000开始的2000个字。

```
MOV AX,0B800H  
MOV ES,AX  
MOV DI,0000H  
MOV CX,07D0H ;2000  
MOV AX,0720H
```

```
CLD  
AG: STOSW  
LOOP AG
```

```
REP STOSW
```

...

## 四. 数据串操作指令

### 3. STOS/STOSB/STOSW — 存数据串

- 例2：将1000:0000H开始的100H个字节的字符串移动到2000H:0100H开始的单元，要求在移动的过程中将所有小写字符改变为大写字符。

1000:0000H

01H  
02H  
03H  
04H

2000:0100H

A : 41H      a : 61H  
Z : 5AH      z : 7AH

## 四. 数据串操作指令

### 3. STOS/STOSB/STOSW — 存数据串

- 例2：将1000:0000H开始的100H个字节的字符串移动到2000H:0100H开始的单元，要求在移动的过程中将所有小写字符改变为大写字符。

MOV	AX,1000H
MOV	DS,AX
MOV	SI,0000H
MOV	AX,2000H
MOV	ES,AX
MOV	DI,0100H
MOV	CX,0100H
AG:	MOV AL,[SI]
	CMP AL,'a'
	JB OK
	CMP AL,'z'
	JA OK
	SUB AL,20H
OK:	MOV ES:[DI],AL
	INC SI
	INC DI
	LOOP AG
	.

1000:0000H

01H

02H

03H

04H

2000:0100H

A : 41H

a : 61H

Z : 5AH

z : 7AH

## 四. 数据串操作指令

### 3. STOS/STOSB/STOSW – 存数据串

- 例2：将1000:0000H开始的100H个字节的字符串移动到2000H:0100H开始的单元，要求在移动的过程中将所有小写字符改变为大写字符。

MOV	AX,1000H
MOV	DS,AX
MOV	SI,0000H
MOV	AX,2000H
MOV	ES,AX
MOV	DI,0100H
MOV	CX,0100H
AG:	MOV AL,[SI]
	CMP AL,'a'
	JB OK
	CMP AL,'z'
	JA OK
	SUB AL,20H
OK:	MOV ES:[DI],AL
	INC SI
	INC DI
	LOOP AG

```
AG:    CLD  
       LODSB  
       CMP   AL,'a'  
       JB    OK  
       CMP   AL,'z'  
       JA    OK  
       SUB   AL,20H  
       STOSB  
       LOOP  AG
```

A : 41H      a : 61H  
Z : 5AH      z : 7AH

1000:0000H

01H  
02H  
03H  
04H

2000:0100H

## 四. 数据串操作指令

---

### ► 4. CMPS/CMPSB/CMPSW – 串比较

- **指令汇编格式**： CMPS/CMPSB/CMPSW [dstr, sstr]
- **操作**：将DS段SI指出的数据与ES段DI指出的数据相减比较，产生标志位的变化，然后修改SI、DI的地址指针。  
 $(DS:SI) - (ES:DI)$   
如果DF=0，则： $SI \leftarrow SI + \text{DELTA}$ ,  $DI \leftarrow DI + \text{DELTA}$   
否则： $SI \leftarrow SI - \text{DELTA}$ ,  $DI \leftarrow DI - \text{DELTA}$
- **受影响的状态标志位**：OF，SF，ZF，AF，PF，CF
- **说明**：
  - 1) 此指令只改变标志，并不回送结果，因而不改变数的原始值。

## 四. 数据串操作指令

### 4. CMPS/CMPSB/CMPSW – 串比较

- 例：比较两个数据块是否相同

```
MOV AX,1000H  
MOV DS,AX  
MOV SI,0000H  
MOV AX,2000H  
MOV ES,AX  
MOV DI,0100H  
MOV CX,0100H
```

```
AG: MOV AL,[SI]  
     CMP AL,ES:[DI]  
     JNE EXIT1  
     INC SI  
     INC DI  
     LOOP AG  
...  
EXIT1: .
```

```
AG: CLD  
     CMPSB  
     JNE EXIT1  
     LOOP AG  
.  
.  
EXIT1: .
```

1000:0000H

01H
02H
03H
04H

2000:0100H

01H
02H

## 四. 数据串操作指令

### ▶ 4. CMPS/CMPSB/CMPSW – 串比较

条件重复前缀 REPZ/REPE 和 REPNZ/REPNE

- 使用方法及格式：这两条条件重复前缀的使用方法和格式与REP相同
- 操作：加有这两条前缀的串操作指令，在循环时不仅检查CX的值是否为0，还检查ZF标志位是否是1，以CX和ZF的情况共同决定是否重复执行后面的基本串操作指令。
  - REPZ/REPE为CX $\neq$ 0且ZF=1时重复执行基本操作，CX=0或ZF=0时停止重复。
  - REPNZ/REPNE为CX $\neq$ 0且ZF=0时重复执行基本操作，CX=0或ZF=1时停止重复。
- 受影响状态标志位：由基本串操作指令的执行情况决定

## 四. 数据串操作指令

|> 例子：比较两个数据块是否相同

```
MOV AX,1000H  
MOV DS,AX  
MOV SI,0000H  
MOV AX,2000H  
MOV ES,AX  
MOV DI,0100H  
MOV CX,0100H  
CLD
```

```
AG:    CMPSB  
        JNE    EXIT1  
        LOOP   AG  
.  
.
```

```
EXIT1: .
```

```
REPE    CMPSB  
JNE     EXIT1  
. .
```

1000:0000H

01H
02H
03H
04H

2000:0100H

01H
02H

## 四. 数据串操作指令

---

### ► 5. SCAS/SCASB/SCASW – 串搜索

- **指令汇编格式**： SCAS/SCASB/SCASW [dstr]
- **操作**：将累加器AL（或AX）的值与ES段DI指出的元素相减比较，然后修改 DI 的地址指针。  
$$AL(\text{或}AX) - (ES:DI)$$

如果DF=0，则： $DI \leftarrow DI + \text{DELTA}$

否则： $DI \leftarrow DI - \text{DELTA}$
- **受影响的状态标志位**：OF，SF，ZF，AF，PF，CF
- **说明**：
  - 1) 此指令隐含指定AX或AL的值作为一个操作数，将其与 ES:DI 指出的数据串元素相比  
较，用以实现在一个串中对给定值的搜索或查找。当给定串为字节串时与AL中的内  
容相比较；当给定串为字串时，AX中的内容相比较。
  - 2) 此指令只改变标志，不回送结果

## 四. 数据串操作指令

### 5. SCAS/SCASB/SCASW – 串搜索

- 例：从一组数据中寻找一个指定的数据。

STR1:	<table border="1"><tr><td>'C'</td></tr><tr><td>'O'</td></tr><tr><td>'M'</td></tr><tr><td>'P'</td></tr><tr><td>'U'</td></tr><tr><td>'T'</td></tr><tr><td>'E'</td></tr><tr><td>'R'</td></tr><tr><td>'M'</td></tr></table>	'C'	'O'	'M'	'P'	'U'	'T'	'E'	'R'	'M'
'C'										
'O'										
'M'										
'P'										
'U'										
'T'										
'E'										
'R'										
'M'										
CHR:	<table border="1"><tr><td>'M'</td></tr></table>	'M'								
'M'										

```
DSEG SEGMENT
STR1 DB      'COMPUTER'
CHR  DB      'M'
DSEG SEGMENT
ASSUME CS:CSEG,DS:DSEG
START: MOV    AX,DSEG
       MOV    DS,AX
       LEA    SI,STR1-1
       MOV    CX,8
       MOV    AL,CHR
AGAIN: INC   SI
       CMP   AL,[SI]
       JE    FOUND
       LOOP  AGAIN
.
.
.
FOUND: .
.
.
CSEG ENDS
END   START
```

## 四. 数据串操作指令

### 5. SCAS/SCASB/SCASW – 串搜索

- 例：从一组数据中寻找一个指定的数据。

STR1:  

'C'
'O'
'M'
'P'
'U'
'T'
'E'
'R'
'M'

CHR:  

'M'
-----

START: MOV AX,DSEG  
MOV ES,AX  
LEA DI,STR1  
MOV CX,8  
MOV AL,CHR  
CLD  
AGAIN: SCASB  
JE FOUND  
LOOP AGAIN

FOUND: .

DSEG SEGMENT  
STR1 DB 'COMPUTER'  
CHR DB 'M'  
DSEG SEGMENT  
ASSUME CS:CSEG,DS:DSEG

START: MOV AX,DSEG  
MOV DS,AX  
LEA SI,STR1-1  
MOV CX,8  
MOV AL,CHR

AGAIN: INC SI  
CMP AL,[SI]  
JE FOUND  
LOOP AGAIN

FOUND: .

CSEG ENDS

END START

## 五. 循环程序的控制方法

---

- |▶ 1. 计数法
- 2. 寄存器终值法
- 3. 条件控制法

## 五. 循环程序的控制方法

---

### ▶ 1. 计数法

- 此方法适用于循环次数已知的循环程序，其特点是简单方便。计数法又分为**正计数法**和**倒计数法**两种。
  - 正计数：将计数器的初值设置为0，每执行一遍工作部分，计数器值增1，然后与规定的已知循环次数比较，若相等，则退出循环，否则，继续执行循环体。

例：计算  $S=\sum_{i=0}^{100} i$

```
MOV AX,0  
MOV BX,0  
AG: ADD AX,BX  
     INC BX  
     CMP BX,100  
     JBE AG
```

## 五. 循环程序的控制方法

---

### ▶ 1. 计数法

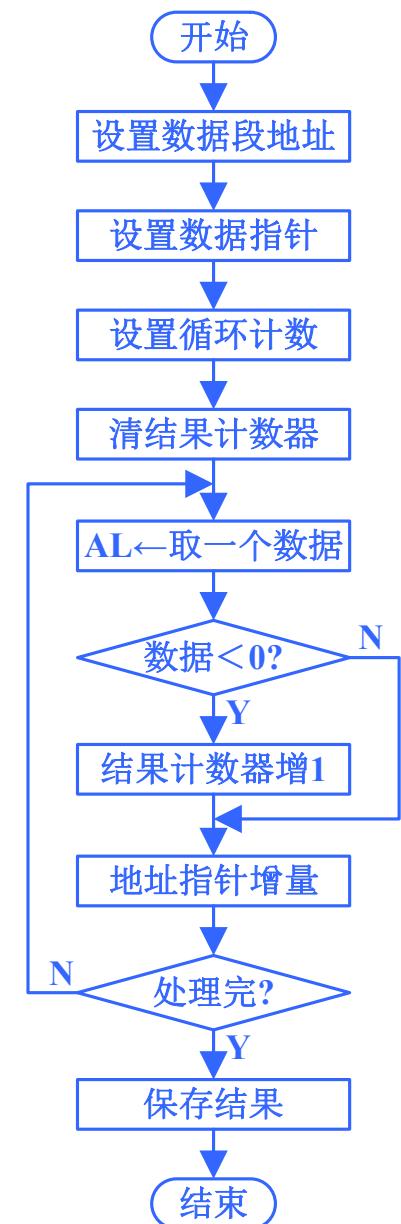
- 此方法适用于循环次数已知的循环程序，其特点是简单方便。计数法又分为**正计数法**和**倒计数法**两种。
  - 正计数：将计数器的初值设置为0，每执行一遍工作部分，计数器值增1，然后与规定的已知循环次数比较，若相等，则退出循环，否则，继续执行循环体。
  - 倒计数：将计数器的初值设置为规定的循环次数，每执行一遍工作部分，计数器值减1，测试是否为0，若为0则退出循环，否则继续执行循环体。
  - **注意**：由于正计数法与倒计数法逻辑上没有任何区别，又由于倒计数法有专门指令，因此用得较多。

# 五. 循环程序的控制方法

## 1. 计数法

- 例：统计由DATA开始的字节数据块中负元素的个数，数据个数在COUNT单元，统计结果存入RLT单元。

DATA:	-1 3 -5 . .	1 ;*****EXAM7.1***** 2 DSEG SEGMENT 3 DATA DB -1,3,-5,.... 4 COUNT DW 50 5 RLT DW 0 6 DSEG ENDS 7 8 CSEG SEGMENT 9 ASSUME CS:CSEG, DS:DSEG 10 START: MOV AX, DSEG 11 MOV DS, AX 12 MOV BX, OFFSET DATA 13 MOV CX, COUNT	14 15 AG1: 16 AND AL, AL 17 JNS PLUS 18 INC DX 19 PLUS: 20 DEC CX 21 JNZ AG1 22 MOV RLT, DX 23 MOV AH, 4CH 24 INT 21H 25 CSEG ENDS 26 END START
COUNT:	32H		
RLT:	00H 00H 00H		



## 五. 循环程序的控制方法

---

### ▶ 2. 寄存器终值控制法

- 此方法类似于计数法，用一个寄存器存放初始值，每执行一次循环体该寄存器的值都按某种规律而有所变化，直到该寄存器值达到某一终值退出循环。与计数法不同之处是：
  - 寄存器的值本质上不一定是循环的次数，可能是一个操作数的地址，也可能是时间的一种表示或程序中要使用的其他数据。显然，这种控制方法比计数法适用问题更为广泛。
  - 控制寄存器的初值或终值也不一定是0或循环计数值；所使用的寄存器也不一定限于CX。

# 五. 循环程序的控制方法

## 2. 寄存器终值控制法

- 例：内存DATA开始存放若干个单字节无符号数，数据末地址为DTEND，编制程序将其中最大数送入MAX单元

```
1 ;*****EXAM7.2*****  
2 DSEG SEGMENT  
3 DATA DB 15H,26H,03H,8AH,11H,...  
4 DTEND DB 62H  
5 MAX DB 0  
6 DSEG ENDS  
7  
8 SSEG SEGMENT STACK  
9 STACK DB 50 DUP(0)  
10 SSEG ENDS  
11  
12 CSEG SEGMENT  
13 ASSUME CS:CSEG, DS:DSEG  
14 ASSUME SS:SSEG  
15 SMAX: MOV AX, DSEG  
16 MOV DS, AX  
17 MOV AX, SSEG  
18  
19  
20  
21  
22 AG:  
23  
24  
25  
26  
27  
28  
29 LOAD:  
30  
31  
32  
33
```

MOV SS, AX  
MOV SP, SIZE STACK  
LEA SI, DATA  
MOV AL, [SI] ;取第一个数据  
INC SI ;指向下一个数据  
CMP SI, OFFSET DTEND  
JA LOAD ;指针大于末址  
CMP AL, [SI] ;两数比较  
JA AG ;AL中数大, 转  
MOV AL, [SI] ;大数→AL  
JMP AG ;转, 继续执行  
MOV MAX, AL ;保存最大值  
MOV AH, 4CH  
INT 21H  
CSEG ENDS  
END SMAX

DATA:	15H
	26H
	03H
	8AH
	11H
	.
	.
	.
	.
DTEND:	62H

## 五. 循环程序的控制方法

---

### 3. 条件控制法

- 在应用问题中常有循环次数未知的情况。在循环次数未知的程序中，虽然有时循环次数可以通过某种方法求得，但比较麻烦；而有时循环程序的循环次数根本无法求得。如有些用有限次计算代替无限计算过程的问题，用循环程序求解时就属于循环次数未知的程序，其控制方法大多是以所给的精度允许误差来控制，把允许误差作为控制条件，将前后两次计算结果的误差与允许误差相比较，如果满足要求则退出循环，否则继续循环。
- 这种利用问题本身的结束条件来控制循环结束的方法叫条件控制法。此方法适用于循环次数未知的情形，比前两种方法更具一般性，因而也稍复杂。LOOP指令对完成此控制方法无能为力，需要用条件转移指令来实现此控制方法。

## 五. 循环程序的控制方法

---

### ▶ 3. 条件控制法

- 例：内存DATA字单元存放一个完全平方数，编制程序求其平方根并存入ROOT字单元。

设计：现在已知N，求N的平方根i，则可以从N中依次减去从1开始的连续奇数，直到N为0时，减去奇数的个数i即为N的平方根。

算法：

$$1=1=1^2$$

$$1+3=4=2^2$$

$$1+3+5=9=3^2$$

$$1+3+5+7=16=4^2$$

$$1+3+5+7+9=25=5^2$$

...

## 五. 循环程序的控制方法

### 3. 条件控制法

```
1 ;*****EXAM7.3*****
2 DSEG SEGMENT
3 DATA DW 13924
4 ROOT DW 0
5 DSEG ENDS
6
7 SSEG SEGMENT PARA STACK 'SSEG'
8 STACK DB 50 DUP(0)
9 SSEG ENDS
10
11 CSEG SEGMENT
12 ASSUME CS:CSEG, DS:DSEG
13 ASSUME SS:SSEG
14 SQRT: MOV AX, DSEG      ; 设置数据段地址
15     MOV DS, AX
16     MOV AX, SSEG      ; 设置堆栈段地址
17     MOV SS, AX
18     MOV SP, SIZE STACK
```

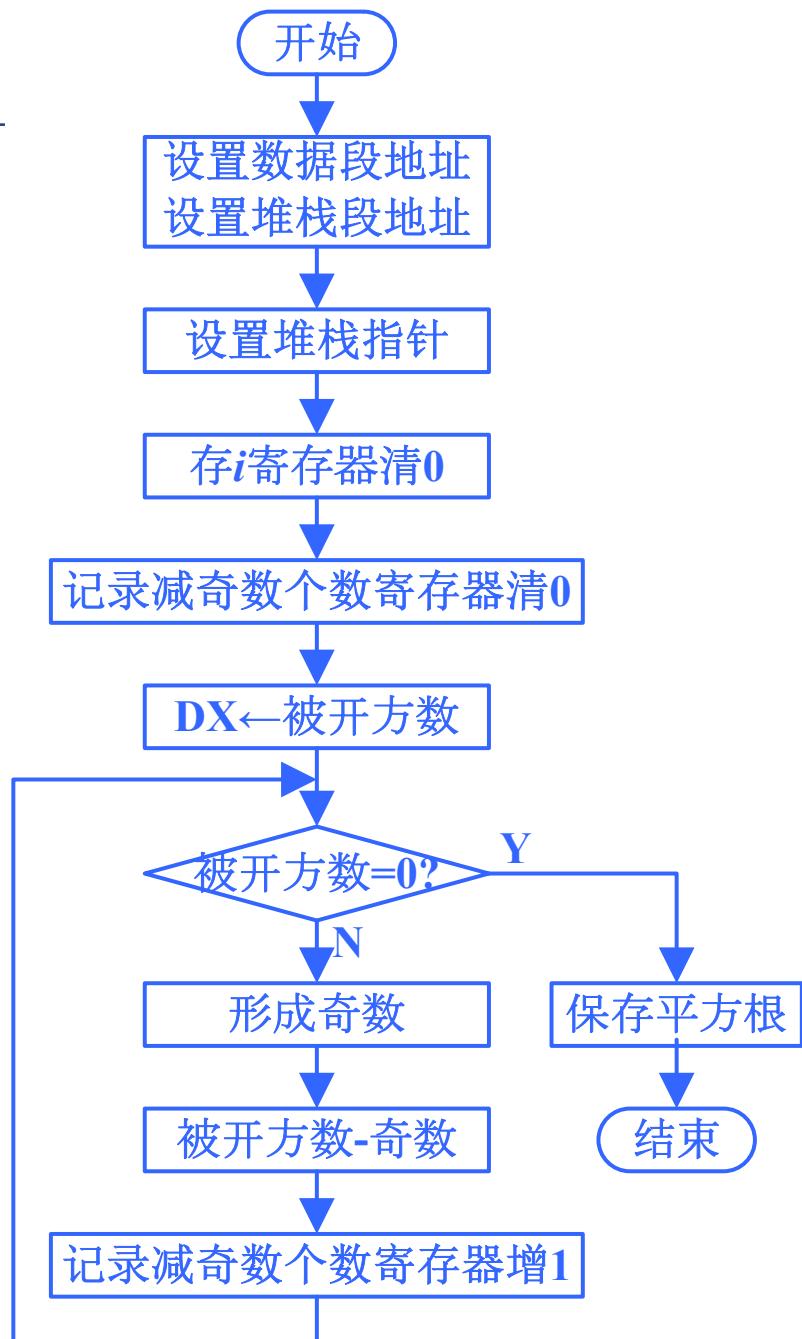


图7.4 求平方根程序流程图

## 五. 循环程序的控制方法

### 3. 条件控制法

```
19      XOR    CX, CX      ;计数器清零
20      XOR    AX, AX      ;设i的初值为0
21      MOV    DX, DATA    ;被开方数送DX
22 AG:   AND    DX, DX      ;被开方数为零吗
23      JZ     LRT        ;被开方数为零,转
24      MOV    BX, AX      ;i值送BX
25      SHL    BX, 1       ;乘2
26      INC    BX          ;形成奇数
27      SUB    DX, BX      ;被开方数减去奇数
28      INC    CX          ;计数器值增1
29      INC    AX          ;i值增1
30      JMP    AG          ;继续工作
31 LRT:   MOV    ROOT, CX    ;保存结果
32      MOV    AH, 4CH
33      INT    21H
34 CSEG   ENDS
35 END SQRT
```

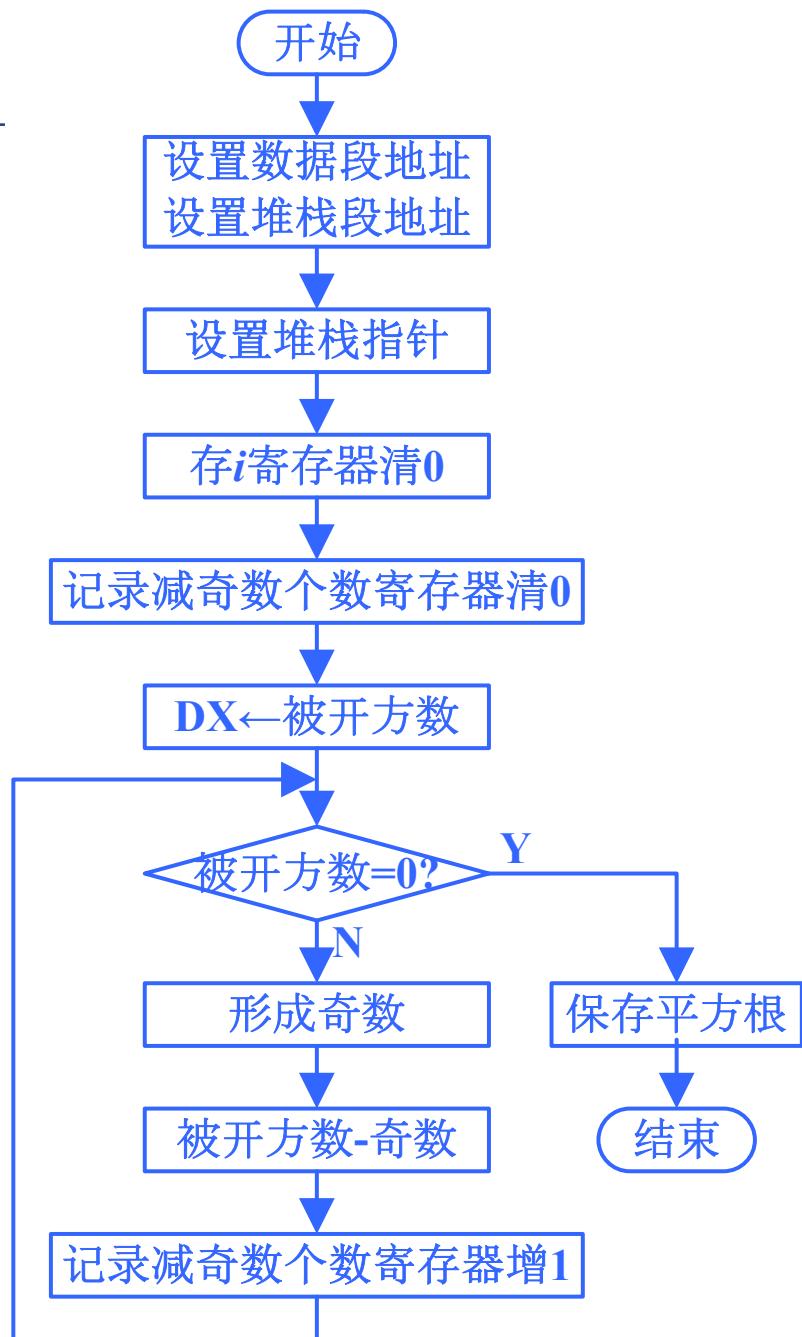


图7.4 求平方根程序流程图

## 六. 单重循环程序举例

---

► 例1：自内存DATA单元开始存放若干个无符号字节数据，数据个数在COUNT单元存放。编制程序分别计算其中奇数、偶数及被4整除的数的和，并分别存入ODDSUM，EVENSUM和FORSUM单元。设各类和不超过16位二进制数，可用一个字表示或存放。

算法：

- 奇数：数据的最低位为1
- 偶数：数据的最低位为0
- 被4整除的数：数据的最低位和次低位均0

## 六. 单重循环程序举例

```
1 ;*****EXAM7.4*****
2 DSEG SEGMENT
3 DATA DB 15H,26H,03H,64H
4 | DB 8AH,0AAH,24H,48H
5 COUNT DW 08
6 ODDSUM DW 0
7 EVENSM DW 0
8 FORSUM DW 0
9 DSEG ENDS
10
11 SSEG SEGMENT STACK
12 STACK DB 20 DUP(0)
13 SSEG ENDS
14
15 CSEG SEGMENT
16 | ASSUME CS:CSEG, DS:DSEG
17 | ASSUME SS:SSEG
18 FSUM: MOV AX, DSEG      ;设置数据段地址
19 | MOV DS, AX
20 | MOV AX, SSEG      ;设置堆栈段地址
21 | MOV SS, AX
22 | MOV SP, SIZE STACK
```

DATA:	15H
	26H
	03H
	64H
	8AH
	0AAH
	24H
	48H
COUNT:	08H
	00H
ODDSUM:	00H
	00H
EVENSM:	00H
	00H
FORSUM:	00H
	00H

## 六. 单重循环程序举例

```
1 ;*****EXAM7.4*****
2 DSEG      SEGMENT
3 DATA      DB      15H, 26H, 03H, 64H
4           DB      8AH, 0AAH, 24H, 48H
5 COUNT     DW      08
6 ODDSUM    DW      0
7 EVENSM   DW      0
8 FORSUM    DW      0
9 DSEG      ENDS
10
11 SSEG      SEGMENT STACK
12 STACK    DB      20 DUP(0)
13 SSEG      ENDS
14
15 CSEG      SEGMENT
16           ASSUME CS:CSEG, DS:DSEG
17           ASSUME SS:SSEG
18 FSUM:    MOV     AX, DSEG
19           MOV     DS, AX
20           MOV     AX, SSEG
21           MOV     SS, AX
22           MOV     SP, SIZE STACK
```

```
23          LEA      SI, DATA      ;设置地址指针
24          MOV      CX, COUNT    ;计数值送CX
25          XOR      AX, AX      ;AX清0
26          XOR      BX, BX      ;清存和寄存器
27          XOR      DX, DX
28          XOR      DI, DI
29 AG:       MOV      AL, [SI]    ;取数据
30          TEST     AL, 01      ;测试最低位
31          JZ       EVNS        ;偶数,转
32          ADD      BX, AX      ;奇数,累计和
33          JMP      CHNT
34 EVNS:     ADD      DX, AX      ;偶数,累计和
35          TEST     AL, 03      ;能否被4整除
36          JNZ      CHNT        ;不能被4整除
37          ADD      DI, AX      ;能,累计和
38 CHNT:     INC      SI         ;指向下一个数据
39          LOOP     AG         ;计算完?未完继续
40          MOV      ODDSUM, BX    ;保存结果
41          MOV      EVENSM, DX
42          MOV      FORSUM, DI
43          MOV      AH, 4CH
44          INT      21H
45 CSEG      ENDS
46 END FSUM
```

<b>DATA:</b>	15H
	26H
	03H
	64H
	8AH
	0AAH
	24H
	48H
<b>COUNT:</b>	08H
	00H
<b>ODDSUM:</b>	00H
	00H
<b>EVENSUM:</b>	00H
	00H
<b>FORSUM:</b>	00H
	00H

## 六. 单重循环程序举例

---

例2：已知某件密码由英文字母A,B,C,D,...,Z组成，最后以美元符号\$结束，且以ASCII码形式存在内存**CIPHER**开始的单元。统计各字母在此件密码中出现的次数，并依次存入**CHRFQ**开始的内存单元。

分析：为了统计各字母出现的次数，先把**CHRFQ**开始的26个单元清0，每当出现某一字母，则在相应的内存单元加1，所有密码全部测试后，**CHRFQ**开始的单元依次存放了A,B,C,...,Z在密码文件中出现的次数。

如何判断一个密码文字是哪个字母，首先想到的一定是从密码中取出一个字母与字母A~Z逐个比较，那么每个字母平均比较次数为 $26/2$ 次，显然是比较麻烦，程序冗长，执行时间长。

## 六. 单重循环程序举例

由于字母相邻的差都是1，因此可以通过简单的计算找到该字母对应的内存单元的地址，对该单元的内容加1即可。例如取得字母‘B’，其ASCII码为42H，将42H-’A’，结果为1，与CHRFQ的地址相加，即得到CHRFQ+1，就是对该单元操作。

CIPHER+0	'B'	CHRFQ+0	00H
+1	'V'	+1	00H
+2	'D'	+2	00H
+3	'C'	+3	00H
+4	'J'	+4	00H
+5	'K'	+5	00H
+6	'J'	+6	00H
+7	'K'	+7	00H
+8	'L'	+8	00H
+9	'H'	+9	00H
+10	'T'	+10	00H
+11	'V'	+11	00H
+12	'U'	+12	00H
.	.	.	.
.	.	.	.
.	\$	+25	.

## 六. 单重循环程序举例

```
1 ;*****EXAM7.5*****
2 SSEG    SEGMENT STACK
3 STACK   DB      50 DUP(0)
4 SSEG    ENDS
5
6 DSEG    SEGMENT
7 CIPHER  DB      'BVDCJKJKLHTVUIPRERTZEQ'
8      DB      'HKAXUERTJKLHFSDSAPAWBEQ'
9      DB      'DHFSBNMHMVRTUDPOIHFXJMO$'
10 CHRFQ   DB      26 DUP(?)
11 DSEG    ENDS
12
13 CSEG    SEGMENT
14      ASSUME CS:CSEG, DS:DSEG
15      ASSUME SS:SSEG, ES:DSEG
16 DECPHR: MOV AX, DSEG          ;设置数据段
17      MOV DS, AX
18      MOV ES, AX
19      MOV AX, SSEG          ;设置堆栈段地址
20      MOV SS, AX
21      MOV SP, SIZE STACK ;设堆栈指针
```

CIPHER+0	'B'	CHRFQ+0	00H
+1	'V'	+1	00H
+2	'D'	+2	00H
+3	'C'	+3	00H
+4	'J'	+4	00H
+5	'K'	+5	00H
+6	'J'	+6	00H
+7	'K'	+7	00H
+8	'L'	+8	00H
+9	'H'	+9	00H
+10	'T'	+10	00H
+11	'V'	+11	00H
+12	'U'	+12	00H
.	.	.	.
.	.	.	.
.	\$	+25	.

## 六. 单重循环程序举例

```
22      LEA DI, CHRFQ      ;将结果单元清0
23      MOV CX, 26
24      CLD
25      XOR AL, AL
26      REP STOSB
27      LEA SI, CIPHER      ;密码首址送SI
28 AG:   LEA DI, CHRFQ      ;结果单元首址送DI
29      MOV AL, [SI]          ;取一密码字符
30      CMP AL, '$'          ;是结束符吗?
31      JZ STOP              ;是,转停机
32      SUB AL, 41H           ;代码减41H
33      XOR AH, AH           ;高位部分清零.
34      ADD DI, AX           ;形成结果单元地址
35      INC BYTE PTR[DI]      ;出现次数增1
36      INC SI                ;SI指向下一个代码
37      JMP AG                ;继续工作
38 STOP:  MOV AH, 4CH
39      INT 21H
40 CSEG  ENDS
41 END DECPHR
```

CIPHER+0	CHRFQ+0
'B'	00H
'V'	00H
'D'	00H
'C'	00H
'J'	00H
'K'	00H
'J'	00H
'K'	00H
'L'	00H
'H'	00H
'T'	00H
'V'	00H
'U'	00H
.	.
.	.
\$	.
	+25

## 六. 单重循环程序举例

```
22      LEA DI, CHRFQ  
23      MOV CX, 26  
24      CLD  
25      XOR AL, AL  
26      REP STOSB  
27      LEA SI, CIPHER  
28 AG:   LEA DI, CHRFQ  
29      MOV AL, [SI]  
30      CMP AL, '$'  
31      JZ STOP  
32      SUB AL, 41H  
33      XOR AH, AH  
34      ADD DI, AX  
35      INC BYTE PTR[DI]  
36      INC SI  
37      JMP AG  
38 STOP: MOV AH, 4CH  
39      INT 21H  
40 CSEG ENDS  
41 END DECPHR
```

```
AG:   MOV BL, [SI]  
      CMP BL, '$'  
      JZ STOP  
      SUB BL, 'A'  
      XOR BH, BH  
      INC CHRFQ[BX]  
      INC SI
```

CIPHER+0	'B'	CHRFQ+0	00H
+1	'V'	+1	00H
+2	'D'	+2	00H
+3	'C'	+3	00H
+4	'J'	+4	00H
+5	'K'	+5	00H
+6	'J'	+6	00H
+7	'K'	+7	00H
+8	'L'	+8	00H
+9	'H'	+9	00H
+10	'T'	+10	00H
+11	'V'	+11	00H
+12	'U'	+12	00H
.	.	.	.
.	.	.	.
.	'\$'	+25	.

## 七. 多重循环程序设计

---

|> 如果一个循环程序的循环体内还包含有一个或多个循环结构的程序，那么这个程序称为双重或多重循环结构程序。我们先用软件延时程序来说明多重循环程序的结构。

<b>SOFTDLY:</b>	<b>MOV</b>	<b>BL,10</b>
<b>DELAY:</b>	<b>MOV</b>	<b>CX,2801</b>
<b>WAIT0:</b>	<b>LOOP</b>	<b>WAIT0 ;内循环延时10ms</b>
	<b>DEC</b>	<b>BL</b>
	<b>JNZ</b>	<b>DELAY</b>

此程序每次内循环时，CX由2801减至零，BL维持不变。外循环进行10遍。此程序虽然简单，但是其结构是双重循环。

## 七. 多重循环程序设计

例：内存DATA开始存放100个单字节数据。编写程序统计这些数据内“0”和“1”个数相等的数据有多少，将结果存入NUMB单元。完成此例需要一个数据一个数据地检查0和1是否相等，相等时则计数加1，直到100个数据检查完毕。

DATA+0	15H
+1	27H
+2	94H
+3	11H
+4	3BH
+5	48H
+6	.
+7	.
+8	.
+9	.
+10	.
+11	.
+12	.
.	.
.	.
.	.
+99	.

## 七. 多重循环程序设计

1:;\*\*\*\*\*EXAM7.7\*\*\*\*\*  
2:SSEG SEGMENT STACK  
3:STACK DW 50 DUP(0)  
4:SSEG ENDS  
5:DSEG SEGMENT  
6:DATA DB 15H,27H,94H ;共100个数  
7: DB 11H,3B,48H,...  
8:NUMB DB 0  
9:DSEG ENDS  
10:CSEG SEGMENT  
11: ASSUME CS:CSEG,DS:DSEG  
12: ASSUME SS:SSEG  
13:START: MOV AX,DSEG  
14: MOV DS,AX  
15: MOV AX,SSEG  
16: MOV SS,AX  
17: MOV SP,SIZE STACK

DATA+0	15H
+1	27H
+2	94H
+3	11H
+4	3BH
+5	48H
+6	.
+7	.
+8	.
+9	.
+10	.
+11	.
+12	.
.	.
.	.
.	.
+99	.

## 七. 多重循环程序设计



```
18:    MOV    SI,OFFSET DATA
19:    MOV    CL,100
20:    XOR    AL,AL
21:RPT2:MOV    CH,08H
22:    XOR    BH,BH
23:    MOV    AH, [SI]
24:RPT1:ROR    AH,1
25:    JNC    CHT1
26:    INC    BH
27:CHT1:DEC    CH
28:    JNZ    RPT1
29:    CMP    BH,04
30:    JNZ    CHT2
31:    INC    AL
32:CHT2:INC    SI
33:    DEC    CL
34:    JNZ    RPT2
35:    MOV    DNUMB,AL
36:    MOV    AH,4CH
37:    INT    21H
38:CSEG ENDS
39:    END    START
```

DATA+0	15H
+1	27H
+2	94H
+3	11H
+4	3BH
+5	48H
+6	.
+7	.
+8	.
+9	.
+10	.
+11	.
+12	.
.	.
.	.
.	.
+99	.

## 七. 多重循环程序设计



```
18:    MOV    SI,OFFSET DATA
19:    MOV    CL,100
20:    XOR    AL,AL
21:RPT2:MOV    CH,08H
22:    XOR    BH,BH
23:    MOV    AH, [SI]
24:RPT1:ROR    AH,1
25:    JNC    CHT1
26:    INC    BH
27:CHT1:DEC    CH
28:    JNZ    RPT1
29:    CMP    BH,04
30:    JNZ    CHT2
31:    INC    AL
32:CHT2:INC    SI
33:    DEC    CL
34:    JNZ    RPT2
35:    MOV    DNUMB,AL
36:    MOV    AH,4CH
37:    INT    21H
38:CSEG ENDS
39:    END    START
```

ADC BH , 0

DATA+0	15H
+1	27H
+2	94H
+3	11H
+4	3BH
+5	48H
+6	.
+7	.
+8	.
+9	.
+10	.
+11	.
+12	.
.	.
.	.
.	.
+99	.

## 七. 多重循环程序设计



```
18:    MOV    SI,OFFSET DATA
19:    MOV    CL,100
20:    XOR    AL,AL
21:RPT2:MOV    CH,08H
22:    XOR    BH,BH
23:    MOV    AH, [SI]
24:RPT1:ROR    AH,1
25:    JC     CHT1
26:    INC    BH
27:CHT1:DEC    CH
28:    JNZ    RPT1
29:    CMP    BH,04
30:    JNZ    CHT2
31:    INC    AL
32:CHT2:INC    SI
33:    DEC    CL
34:    JNZ    RPT2
35:    MOV    DNUMB,AL
36:    MOV    AH,4CH
37:    INT    21H
38:CSEG ENDS
39:    END    START
```

DATA+0	15H
+1	27H
+2	94H
+3	11H
+4	3BH
+5	48H
+6	.
+7	.
+8	.
+9	.
+10	.
+11	.
+12	.
.	.
.	.
+99	.

## 八. 循环程序的效率

---

► 循环程序的工作部分、修改部分和控制部分，一般都要反复执行多次，因此，采用循环程序处理问题时，若能使这几部分的指令条数减少；采用长度短、执行时间短的指令，就会大大缩短整个程序的执行时间，提高程序的效率。

所谓使用的**指令条数尽可能少**，一方面是选择好的算法，另一方面是算法确定后，实现算法时少用指令。

所谓**使用指令的长短**，就是实现某种功能时，可以用多种不同的指令实现，而这些不同指令的长度也不同，指令长度越短，控制器和内存打交道的次数越少，就可以减少时间。

## 八. 循环程序的效率

---

► 例：将寄存器AX清0，可以采用下述两种不同的方法：

- ① XOR AX,AX;2个时钟周期，2个字节
- ② MOV AX,0 ;4个时钟周期，3个字节

例：将AL，BL，CL赋同样的数值，可用下述两个不同程序段实现：

- ① MOV AL,DATA ;DATA为字节单元,3个字节,10个周期  
MOV BL,DATA ;4个字节,14个时钟周期  
MOV CL,DATA ;4个字节,14个时钟周期
- ② MOV AL,DATA ;3个字节,10个时钟周期  
MOV BL,AL ;2个字节,2个时钟周期  
MOV CL,AL ;2个字节,2个时钟周期

## 八. 循环程序的效率

---

► 例：计算  $S = \sum_{i=1}^{100} (X_i + Y) * Z$ ，已知  $X_i, Y, Z$  为单字节无符号数， $X_i$  存放在 **ARGX** 开始的单元， $Y, Z$  分别存于 **ARGY**, **ARGZ** 单元。

① 直接计算，200次加，100次乘

② 变形为：  $S = Z * \sum_{i=1}^{100} X_i + 100 * Y * Z$  ; 101次加，3次乘

③ 变形为：  $S = Z * (\sum_{i=1}^{100} X_i + 100 * Y)$  ; 101次加，2次乘

## 八. 循环程序的效率

```
1:;*****EXAM7.8*****  
2:SSEG SEGMENT STACK  
3:STACK DB 50 DUP(0)  
4:SSEG ENDS  
5:DSEG SEGMENT  
6:ARGX DB 15,26,03,64,80,11,24,48...  
7:ARGY DB 34 ;Y  
8:ARGZ DB 21 ;Z  
9:SUM DW 0,0 ;保存结果单元  
10:DSEG ENDS  
11:CSEG SEGMENT  
12: ASSUME CS:CSEG,DS:DSEG  
13: ASSUME SS:SSEG  
14:CALT:  
15: MOV AX,DSEG  
16: MOV DS,AX  
17: MOV AX,SSEG  
18: MOV SS,AX  
19: MOV SP,LENGTH STACK  
20: MOV SI,OFFSET ARGX  
21: XOR BH,BH ;高字节清零  
22: MOV BL,[SI] ;取第一个数据  
23: AGAIN: INC SI ;修改地址指针  
24: ADD BL,[SI] ;与下一数值相加  
25: ADC BH,0 ;加进位  
26: LOOP AGAIN ;未完,继续  
27: MOV AL,100 ;取100  
28: MUL ARGY ;计算100*Y  
29: ADD AX,BX ;与累加和相加  
30: MOV BL,ARGZ ;取Z  
31: XOR BH,BH ;扩展Z为双字长  
32: MUL BX ;计算最终结果  
33: MOV SUM,AX ;保存结果  
34: MOV SUM+2,DX  
35: MOV AH,4CH  
36: INT 21H  
37: CSEG ENDS  
38: END CALT
```