串操作指令

8.1 串操作指令

- 串，是数据结构中，由线性表转化而来的一种基本的线性结构。

- 和串具有相同结构特征的结构还有堆栈和队列，但是串的操作特征更为自由。

机器底层的数据结构

- （1）队列：指令队列、中断请求队列
- （2）堆栈：堆栈段
- （3）串：字符串

8.1 串操作指令

- 由于实际应用中大量涉及到串操作，所以8086 / 8088 CPU 提供了一组专门的串操作指令用于提高串操作的效率。

串操作指令所使用的寻址方式都属于串操作寻址方式。

（1）取串指令 LODS（Load from String）

- 指令格式：LODS 源串

- 功能：
- 字节操作：AL<=（（SI）），SI<=（SI）∓1
- 字操作：AX<=（（SI）），SI<=（SI）∓2

（1）取串指令 LODS（Load from String）

- 其中源串为变量名称，用于指明源串类型是字还是字节。
- SI的增减由DF标志控制。
- 标志位影响：无
- 串操作指令对源串操作时，使用DS给出段基值

另两种格式

- 格式2：LODSB
- 指明使用字节操作。

- 格式3：LODSW
- 指明使用字操作。

- 这两种格式除了指明隐含操作数的类型以外，其余特征和第一种格式完全相同。

（2）存串指令STOS（Store in String）

- 指令格式：STOS 目的串
- 功能：
- 字节操作：（DI）<=（AL），DI<=（DI）∓1
- 字操作：（DI）<=（AX），DI<=（DI）∓2

- 其中目的串为变量名称，指明目的串类型是字还是字节。

- DI的增减由DF标志控制
- 标志位影响：无

（2）存串指令STOS（Store in String）

- 串操作指令中对目的串进行操作时，使用ES段寄存器给出的段寄存器。

- 如果源串、目的串都在数据段中，那么DS、ES都应指向数据段。

另两种格式

- 格式2：STOSB
- 指明使用字节操作。

- 格式3：STOSW
- 指明使用字操作。

（3）串传送指令 MOVS（Move String）

- 指令格式：MOVS 目的串，源串

- 功能：
- 字节操作：（DI）<=（（SI）），SI<=（SI）∓1，DI<=（DI）∓1
- 字操作：（DI）<=（（SI）），SI<=（SI）∓2，DI<=（DI）∓2

（3）串传送指令 MOVS（Move String）

- 目的串和源串用于指明操作数类型，二者类型必须一致，否则会出现语法错误。

- 标志位影响：无

另两种格式

- 格式2：MOVSB
- 指明使用字节操作

- 格式3：MOVSW
- 指明使用字操作

（4）串比较指令CMPS（Compare String）

- 指令格式：CMPS 源串，目的串

- 功能：
- 字节操作：（（SI））-（（DI）），SI<=（SI）∓1，DI<=（DI）∓1
- 字操作：（（SI））-（（DI）），SI<=（SI）∓2，DI<=（DI）∓2

（4）串比较指令CMPS（Compare String）

- 其中源串、目的串用于指明操作数类型，二者类型必须保持一致。

- 标志位影响：AF、CF、OF、PF、SF、ZF，标志位解释和算术运算指令相同。

（4）串比较指令CMPS（Compare String）

- 实际上，CMPS指令执行的操作和CMP指令非常相似。

- 通常在循环结构中使用CMPS指令比较两个串是否相等。

（5）串搜索指令SCAS（Scan String）

- 指令格式：SCAS 目的串

- 功能：
- 字节操作：（AL）-（（DI）），DI<=（DI）∓1
- 字操作：（AX）-（（DI）），DI<=（DI）∓2

- 其中目的串用于指定操作数的类型

- 标志位影响：AF、CF、OF、PF、SF、ZF，标志位解释和算术运算指令相同。

（5）串搜索指令SCAS（Scan String）

- 通常在循环结构中使用SCAS指令在字符串中查找指定的字符。

（6）重复前缀操作指令 REP（Repeat）

- 指令格式：REP

- REP是指令前缀，不能单独使用，必须在后面跟上串操作指令，配合使用。

- 功能：把CX寄存器作为计数器，循环执行前缀后面的串操作指令，每执行一次，把CX中内容减1，直到（CX）=0时退出循环。

（6）重复前缀操作指令 REP（Repeat）

- REP指令前缀产生的循环是指令执行过程内部实现的循环，不是程序中的循环结构。

- 因为它的实现级别更低，所以比循环结构具有更高的效率。

另外两种重复前缀

- REPZ（REPE）：循环条件为（CX）≠0，ZF=1
- REPNZ（REPNE）：循环条件为（CX）≠0，ZF=0
- 它们和REP前缀的对应关系和LOOPZ、LOOPNZ指令与LOOP指令的对应关系很相似。

另外两种重复前缀

- REP、REPZ、REPNZ是指令前缀，不能单独使用，必须和串操作指令配合。

（7）串操作指令提高效率的原理

- 1）串操作指令把数据操作和指针修改两个功能结合到一条指令中。
- 2）使用重复前缀把循环控制也结合到同一条指令中。
- 把数据操作、指针修改、循环控制集中到一条指令中完成，比用多条指令实现的同样功能效率高出很多。

串操作示例1

- 例1 将指定字符串中的数字字符全部删除

- data segment
- string1 db ‘He is 35 years old$’
- strlen equ $-string1
- strtmp db strlen dup(0)
- data ends
- ; 堆栈段省略

串操作示例1

- code segment
- assume ss:stack1,ds:data,es:data,cs:code
- main: mov ax, data
- mov ds, ax
- mov es, ax
- lea si, string
- lea di, strtemp
- mov cx, strlen

串操作示例1

```
        cld
lop:    lodsb
        cmp   al, '9'
        ja    l1
        cmp   al, '0'
        jb    l1
        jmp   next
l1:     stosb
next:   loop  lop
```

串操作示例1

- lea si, strtmp
- lea di, string
- mov cx, strlen
- cld
- rep movsb
- mov dx, offset string
- mov ah, 09h
- int 21h

串操作示例1

```
        mov   ah,  4ch
        int     21h
code   ends
        end   main
```

串操作示例2

- 例2. 编制程序确定指定字符是否在指定字符串中，若在，则记录该字符在串中第一次出现的位置（0~n-1）；若不在，则设置相应的标志（0FFH）。

串操作示例2

```
DATA  SEGMENT
STRNG    DB    'SEARCH THIS STRING$'
COUNT    EQU  $-STRING
KEY        DB    'T'
RESULT  DB    ?
DATA  ENDS

STACK1  SEGMENT  STACK
            DW   40H  DUP(0)
STACK1  ENDS
```

串操作示例2

```
CODE  SEGMENT
ASSUME  CS:CODE,SS:STACK1,DS:DATA,ES:DATA
MAIN:  MOV    AX,  DATA
       MOV    DS,  AX
       MOV    ES,  AX
       MOV    DI,  OFFSET  STRING
       MOV    CX,  COUNT
       MOV    AL,  KEY
```

串操作示例2

```
        CLD
        REPNZ  SCASB
        JZ    FOUND
        MOV    RESULT,  0FFH
        JMP    EXIT
FOUND:  DEC    DI
        MOV    AX,  DI
        MOV    RESULT,  AL
EXIT:   MOV    AH,  4CH
        INT    21H
CODE    ENDS
END    MAIN
```

串操作示例3

- 例3. 编制程序确定指定子字符串是否在指定字符串中，若在，则记录该子串第一次出现的位置，若不在，则设置相应的标志（0FFH）。

串操作示例3

- data segment
- string db ‘This is an example.$’
- strlen equ $-string
- substr db ‘exam’
- sublen equ $-substr
- result db ?
- data ends
- ; 堆栈段省略

串操作示例3

- code segment
- assume cs:code,ss:stack1,ds:data,es:data
- main: mov ax, data
- mov ds, ax
- mov es, ax
- mov bx, strlen
- sub bx, sublen-1
- lea si, string

串操作示例3

```
        mov   ax,  si
lop:    lea    di,  substr
        mov    cx,  sublen
        cld
        repz   cmpsb
        jz     mat
        inc    ax
        mov    si,  ax
        dec    bx
        jnz    lop
```

串操作示例3

```
        mov   result,  0ffh
        jmp    exit
mat:     sub    ax,  offset  string
        mov   result,  al
exit:    mov   ah,  4ch
        int     21h
code    ends
end    main
```