目录

第 9 章 串处理程序设计

串操作如比较两个串是否相同、将一个串拷贝到另外一个位置等等都是非常常见的操作。采用前面学习过的分支和循环程序结构、比较指令、数据传送指令、转移指令就可以完成串操作的功能。但是 x86 中提供了专门的串操作指令，使用操作指令不仅编写程序要简单一些，而且它们具有更高的执行效率。在 C 语言程序设计中， memcpy、memcmp、 memset、memchr 等函数的内部实现都采用了串操作指令。本章将介绍串操作指令，并且给出实现相同功能的不同程序的运行时间的对比结果。

9．1 串操作指令简介

在编写程序时，经常遇到对 ASCII 字符串的操作。例如，将字符串从一个存储区移至另一个存储区、计算某一字符串的长度、判断两字符串是否相等、在一字符串中查找某一字符出现的次数或者要在某一字符串中插入另一字符串等。在编写程序时，也会遇到将一个整型数组的各个元素置 0 或者置成某一个初值；在一个整型数组中查找某个元素是否出现；将一个整型数组中的各个元素依次拷贝到另一个数组中等等功能要求。这些操作都可以归类为串操作，可以用串操作指令来编写程序。由此可以，串可以看成是由字节组成的串，也可以是由字或者双字组成的串。为了方便地实现串的操作，简化程序设计并且提高程序的运行速度，X86 提供了串操作指令。正确、灵活地使用串操作指令可以大大方便程序设计的工作。

串操作指令共有五条。

① 传送字节/字/双字串的指令 movs、movsb、movsw，movsd；

② 比较字节/字/双字串的指令 cmps、cmpsb、cmpsw、cmpsd；

③ 搜索字节/字/双字串的指令 scas、scasb、scasw、scasd；

④ 存储字节/字/双字串的指令 stos、stosb、stosw、stosd。

⑤ 取字节/字/双字串的指令 lods、lodsb、lodsw、lodsd；

串操作指令的方便之处体现在，只要按规定置好初始条件，选用正确的串操作指令，就可以完成规定的操作。而且这些指令的前面均可加重复前缀，能在条件满足的情况下重复执行，而不用考虑指针如何移动，循环次数如何控制等问题，从而简化了程序设计，节省了存储空间，加快了运行的速度。可使用的前缀有：

① rep——重复，即无条件重复 ecx 寄存器中指定的次数，即(ecx)≠0 重复。

② repe/repz——相等时重复，即(ecx)≠0 同时 zf＝1(重复次数还未为 0 且比较时相等)时重复，否则，重复终止。

③ repne/repnz——不相等时重复，即(ecx)≠0 同时 zf＝0 时重复，否则，重复终止。

串操作指令在使用格式和使用方法上有许多类似的地方，它们隐含地使用相同的寄存器、标志位和符号，现将它们的使用情况列表如下：

源串指示器	ds：esi
目的串指示器	es：edi
重复次数计数器	ecx
scas 指令的搜索值在	al/ax/eax 中
lods 指令的目的地址为	al/ax/eax
stos 指令的源地址为	al/ax/eax
传送方向	df＝0(用指令 cld 实现)时， esi、edi 自动增量
	df＝1(用指令 std 实现)时， esi、edi 自动减量

系统规定：源串一般要在当前数据段中，目的串在当前附加数据段中。在 32 段扁平模式

下，(ds)与(es)相同，可以认为数据段和附加数据段是一个段。所有的串操作指令均以寄存器间接方式访问源串或目的串中的各元素，并自动修改 esi 和 edi 的内容。若 df＝0，则每次操作后， esi、edi 自动增量(字节操作加 1、字操作加 2、双字加 4)；若 df＝1，则每次操作后， esi、edi 自动减量(字节操作减 1，字操作减 2、双字减 4)，使之指向下一个元素。

当指令带有重复前缀时，指令要被重复执行，每执行一次，就检查一次重复条件是否成立，如成立，则继续重复；否则中止重复，执行后续指令。带重复前缀的串操作指令的执行流程如图 9.1 所示。

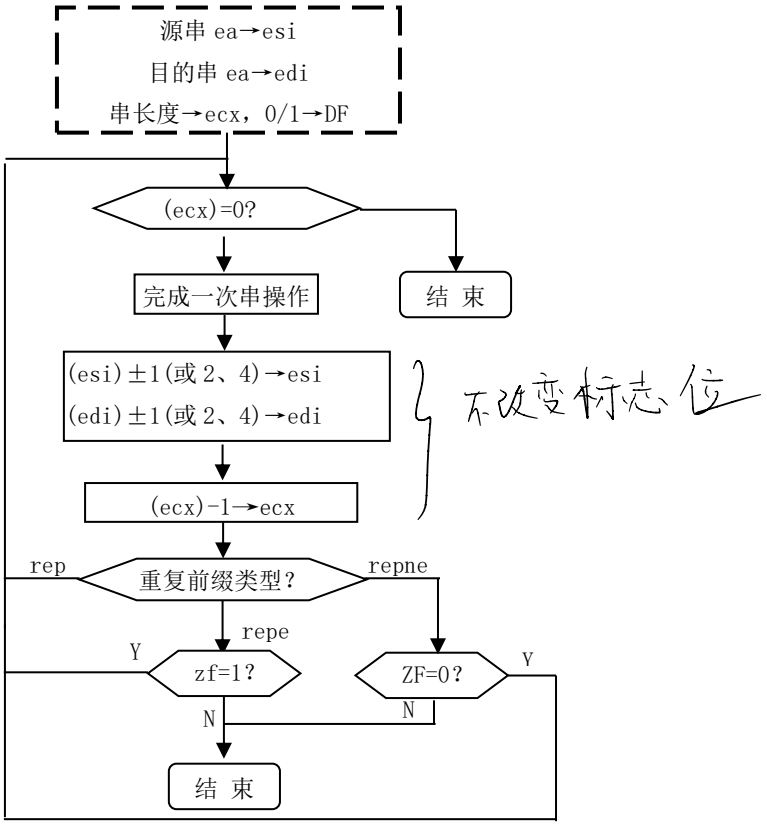

图 9.1　带重复前缀的串操作指令的执行过程示意图

从图 9.1 中可知：

(1) 在有重复前缀时，先判断(ecx)是否为 0，若(ecx)初值为 0，则不会引起串操作。

(2) 串操作中止后，(esi)、(edi)均为下一个存储单元的偏移地址，其方向由 df 确定。

(3) (ecx)-1->ecx 的操作并不影响标志位。

需要说明的是，在 8086 中，串操作语句在 16 位段中执行时使用的是 16 位寄存器 cx、si、di。

9．2 串传送指令

语句格式：　movs　opd，ops

　　movsb —— 字节串传送

　　movsw —— 字串传送

　　movsd —— 双字串传送

功　　能：(ds:[esi])→es:[edi]

　　当 df＝0 时，(esi)和(edi)增量可以为 1(字节操作)、2(字操作) 或 4(双字操作)。

当 df＝1 时，(esi)和(edi)减量可以为 1(字节操作)、2(字操作) 或 4(双字操作)。

注意：

(1) opd、ops 分别为目的串和源串的符号首址，movs 根据该首地址定义的类型确定串操作的类型(字节、字或双字)。

例如 “movs buf2，buf1”，当 buf1、buf2 是字节类型的变量时，显示的反汇编语句是：“movs byte ptr es:[edi],byte ptr [esi] ”，当这两个变量是字类型时，显示的反汇编语句是：“movs word ptr es:[edi],word ptr [esi] ”。这也就表明不是将 buf1 缓冲区的内容拷贝到 buf2 所在的缓冲区，而是将以[esi]为地址的单元中的内容拷贝到以[edi]为地址的单元中。

(2) movsb/movsw/movsd 指出了串操作的类型，不带操作数。

(3) 该指令前，可带重复前缀 rep，用于数据串的成块传送。

【例 9.1】 设有两个长度皆为 10000 字节的缓冲区 buf1 和 buf2，编写程序将 buf1 中 10000 个字节内容拷贝到 buf2 中。

为了比较用串传送指令和以用学过的指令编写程序的执行效率的差异，我们在程序中使用了 2 个 C 语言函数，clock()和 getchar()。clock()是获得以毫秒为单位的当前时间，在程序段执行前后两次调用 clock(),获得 2 个时间,它们的差即是程序消耗的时间。getchar()用于在显示时间后等待用户击键后退出程序。此外，我们将程序放在了一个子程序中，这样就像 C 语言程序那样，一个程序是由很多子程序（函数）组成。因为目前的 CPU 的速度很快，我们将缓冲区的拷贝重复执行了 20000 次。

① 使用带有重复前缀的串传送指令

```
.686P
.model flat, c
  ExitProcess  proto stdcall :dword
  printf       proto :ptr sbyte, :vararg
  clock        proto
  getchar      proto
  includelib  libcmt.lib
  includelib  legacy_stdio_definitions.lib
.data
  lpFmt        db  "%d (ms)", 0dh,0ah,0
  buf1         db   10000 dup(0)
  buf2         db   10000 dup(1)
  begin_time   dd   0
  end_time     dd   0
  spend_time   dd   0          ; 运行时间(毫秒)
.stack 200
.code
main  proc
   invoke clock
   mov    begin_time, eax
   mov    ebx, 20000
lp1:
   lea    esi, buf1
```

```
    lea    edi, buf2
    mov    ecx, 10000
    cld
    rep    movsb
    dec    ebx
    jnz    lp1
    invoke clock
    mov    end_time, eax
    sub    eax, begin_time
    mov    spend_time, eax
    invoke printf,offset lpFmt, spend_time
    invoke getchar
    invoke ExitPRocess, 0
main  endp
end
```

在本人的机器上执行该程序约需要 10 毫秒。不同配置的机器的运行速度是不同的。

② 使用一般的数据传送和循环控制

对于上面的程序，将 “rep movsb” 用如下的程序段代替：

```
lp2:
    mov    al, [esi]
    mov    [edi],al
    add    esi, 1
    add    edi, 1
    dec    ecx
    jnz    lp2
```

修改后的程序用时约为 350 毫秒。这比使用带有重复前缀的串传送指令的程序慢很多。

上述程序一次执行只传送了一个字节，可以改为一次传送 4 个字节，用时约 80 毫秒。程序段修改的部分如下。

```
……
 mov    ecx, 10000/4
lp2:
    mov    eax, [esi]
    mov    [edi],eax
    add    esi,4
    add    edi,4
    dec    ecx
    jnz    lp2
    ……
```

③ 使用不带重复前缀的串传送指令

rep movsb 与如下程序段的功能也是等价的。

```
lp2:
  movsb
  dec  ecx
  jnz  lp2
```

但是，使用该方法的程序运行很慢，用时约 1600 毫秒。

此外，我们也比较了程序段“mov ecx, 10000”、“rep movsb”与程序段“mov ecx, 10000/4”、“rep movsd”的执行效率，两者运行时间并没有什么差别。

在使用 C 语言编写程序时，假设要将一个数组中的元素拷贝到另一个数组中，可以调用 memcpy 函数来实现，它的实现中采用了串操作指令，这比用循环来拷贝元素的执行速度要快。

9．3 串比较指令

语句格式：　cmps　opd，ops
　　cmpsb ——　字节串比较
　　cmpsw ——　字串比较
　　cmpsd ——　双字串比较

功　　能：(ds:[esi])—(es:[edi])，即将[esi]所指的源串中的一个字节(或字、双字)存储单元中的数据与[edi]所指的目的串中的一个字节(或字、双字)存储单元中的数据相减，并根据相减的结果设置标志位，但结果并不保存。
修改串指针，使之指向串中的下一个元素。
当 df＝0 时，每做完一次比较操作，(esi)和(edi)增加 1(字节操作)、2(字操作) 或 4(双字操作)。
当 df＝1 时，每做完一次比较操作，(esi)和(edi)减 1(字节操作)、2(字操作) 或 4(双字操作)。

注意：opd、ops 分别为目的串和源串的符号首址，cmps 根据该首地址定义的类型确定串操作的类型(字节、字或双字)，但不会自动给 esi、edi 赋值。这与“movs opd,ops”的规则是相同的。cmpsb、cmpsw、cmpsd 指出了串操作的类型，不带操作数。

与 cmp 一样，串比较指令并不保存两个单元内容的差值，而只设置标志位，因此比较结束后，源串与目的串的内容并不改变。通常，此语句的后面常常跟着条件转移指令，用来根据比较的结果确定转移方向。

串比较指令与一般比较指令 cmp 有一点很重要的区别是：串比较指令在比较时是源操作数减目的操作数，而一般比较指令是目的操作数减源操作数。在调试程序时，用反汇编观察 cmpsb 看到的指令是：“cmps byte ptr [esi],byte ptr es:[edi] ”。在比较串的大小时需要注意这些细节。

cmps 指令通常可带重复前缀 repe/repz 或 repne/repnz。若带有 repe，表示两个比较的内容相等则重复执行，即当源串与目的串未比较完((ecx)≠0)且两串元素相等(即 zf＝1)时继续比较；若带有 repne，比较操作的执行流程是：当源串与目的串未比较完((ecx)≠0)且两串元素不等(即 zf＝0)时继续比较。

【例 9.2】 设有两个长度皆为 10000 字节的缓冲区 buf1 和 buf2，编写程序比较 buf1、buf2 两个缓冲区中的内容是否相同。若相同，显示 equal，否则显示 not equal。

```
.686P
.model flat, c
  ExitProcess proto stdcall :dword
  printf      proto  :ptr sbyte, :vararg
  getchar     proto
  includelib  libcmt.lib
  includelib  legacy_stdio_definitions.lib
.data
```

```
    msg1        db   'not '
    msg2        db   'equal', 0dh,0ah,0
    buf1        db   'AB', 9998 dup(0)
    buf2        db   'CD', 9998 dup (0)
.stack 200
.code
main  proc
    lea    esi, buf1
    lea    edi, buf2
    mov    ecx, 10000
    cld
    repz   cmpsb
    jz     Display_equ
    invoke printf,offset msg1
    jmp    exit
Display_equ:
     invoke printf,offset msg2
exit:
    invoke getchar
    invoke Exitprocess, 0
main  endp
  end
```

注意，在上面的程序中，虽然 msg 的定义为“msg1 db 'not '”，但是，以 msg1 为起始的地址的串为：'not equal',0dh,0ah,最后串以 0 结束。串不等时，显示 not equal。在使用 C 语言编写程序时，比较两个缓冲区中的内容是否相同，可以调用 memcmp 函数来实现。

9.4 串搜索指令

语句格式：　scas　opd

scasb —— 字节串搜索

scasw —— 字串搜索

scasd —— 双字串搜索

功　能：　字节操作 (al) - (es:[edi])

字操作　(ax) - (es:[edi])

双字操作 (eax) -(es:[edi])

根据相减的结果设置标志位，但结果并不保存。修改串指针 edi，使之指向串中的下一个元素。操作结束后，al/ax/eax 和目的串的内容并不改变。

当 df＝0 时，　(edi)增 1(字节操作)或 2(字操作) 或 4(双字操作)。

当 df＝1 时，　(edi)减 1(字节操作)或 2(字操作) 或 4(双字操作)。

scas 可带重复前缀 repe/repz 或 repne/repnz。如果带 repe/repz，搜索操作流程是：当目的串未搜索完且串元素等于搜索值时，即　(ecx)≠0 且 zf＝1，继续搜索；若带 repne/repnz，搜索操作流程：当目的串未搜索完且串元素不等于搜索值时，即(ecx)≠0 且 zf＝0，继续搜索。

在 C 语言程序设计中，可以使用 memchr 在一个串中搜索指定的字符。不过，在调试跟踪

进入 memchr 函数时，发现它并没有使用串搜索指令，也没有一个字符一个字符的比较，而是用了一种“怪”方法。该方法一次读取 4 个字节，然后判断该双字数据中有无要找的字符，若有，则再从 4 个字节中找出要搜索的字符；若 4 个字节中无要找的字符，则继续取下 4 个字符，直到缓冲区全部扫描完。

下面给出了判断（ecx）中是否有字符变量 x 的核心代码。

首先产生(ebx)，使其 4 个字节的每一个字节的值都是 x 的值。

```
xor   ebx, ebx
mov   bl,  x
mov   edi, ebx
shl   ebx, 8
add   ebx, edi
mov   edi, ebx
shl   ebx, 10H
add   ebx, edi
```

；假设变量 x=‘a’,指令上面的程序段后，(ebx)=61616161H。下面判断(ecx)的 4 个字节数据中，有无 x 中的值出现。

```
xor   ecx, ebx
mov   edi, 7EFEFEFFH
add   edi, ecx
xor   ecx, 0FFFFFFFFH
xor   ecx, edi
and   ecx, 81010100H
jz    not_occur
```

；当 zf=1 时，说明 (ecx)中的 4 个字节都不是要找的字符 x；当 zf=0 时，说明 4 个字节中至少有一个字节是要找的字符。

表面上看，上面的程序绕得挺复杂，不那么直观，但其中技巧也值得琢磨。在分析程序段时，我们先看最“关键”的语句（直接决定最后结果的语句），最后是通过“and ecx,81010100H”来判断是否有要搜素的字节，它只看了 4 个二进制位上的值是否全为 0，只有(ecx)在这 4 个二进制位上的值全为 0，才说明 4 个字节中都没有要搜索的字符。“and ecx,81010100H”表明并不关心其他的二进制位上的值，因此可分析这 4 个二进制位的信息在程序段执行中是如何变化的。下面以 81010100H 中的最后一个 1 出现的位置（32 位二进制数的最右一位为第 0 位，该位是第 8 位）上的信息变迁为例来为分析信息变换过程。首先(edi)=7EFEFEFFH，第 8 位上的二进制值为 0。执行“add edi, ecx”，(edi)第 8 位的值就是 0 加上(ecx)的第 8 位以及从第 7 位向第 8 位产生的进位；若进位为 0，则(edi)第 8 位的值就是(ecx)第 8 位的值。“xor ecx，0FFFFFFFFH”，等价于“not ecx”,之后“xor ecx，edi”，第 8 位的值就一定是 1；反之，若第 7 位向第 8 位产生的进位为 1，则运行“xor ecx，edi”的第 8 位一定为 0。假设最开始(ecx)最后一个字节的内容与待找的字符相同，则“xor ecx,ebx”后，最后一个字节的内容就一定是 0，它与 7EFEFEFFH 的最后一个字节(0FFH)相加，就不会向前产生进位，这就使得执行“and ecx,81010100H”前(ecx)的第 8 位为 1，从而使 and 的结果非 0，即表明出现了要找的字节；反之，最开始(ecx)最后一个字节的内容与待找的字符不相同，则“xor ecx,ebx”后，最后一个字节的内容一定不是 0，它与 7EFEFEFFH 的最后一个字节(0FFH)相加，就会向前产生进位，这就使得执行“and ecx,81010100H”前(ecx)的第 8 位为 0，从而使 and 的结果在第 8 位为 0，即表明最后一个字节不是要找的字符。同理，可分析其他字节的信息变化。

9．5 向目的串中存数指令

语句格式：　stos　opd

stosb ——— 向字节串中存数

stosw ——— 向字串中存数

stosd ——— 向双字串中存数

功　　能：　字节操作 (al)-> es:[edi]

字操作　(ax) -> es:[edi]

双字操作 (eax) -> es:[edi]

将 al/ax/eax 中的数据送入 edi 所指的目的串中的字节/字/双字存储单元中。修改指针 edi，使之指向串中的下一个元素。

当 df＝0 时，　(edi)增 1(字节操作)、(字操作) 或 4(双字操作)。

当 df＝1 时，　(edi)减 1(字节操作)、2(字操作) 或 4(双字操作)。

该指令执行后并不影响标志位,因而它只带 rep 重复前缀,用来将一片连续的存储字节、或双字单元置相同的值。

在 C 语言程序设计中，可以使用 memset 来完成相似的功能。下面给出了反汇编中看到的核心代码。

```
    #define M 1000
    #define N 10
    int a[M][N];
memset(a,0,sizeof(int)*M*N);
010040C8  push        9C40h
010040CD  push        0
010040CF  push        18D818C0h   ; a 定义为全局变量时
                      ;若 a 定义为局部变量，则其翻译结果为：lea eax,[a]
                      ;                                      push eax
010040D4  call        _memset (0100107Dh)
010040D9  add         esp,0Ch
```

在函数体的实现中，可以看到如下信息：

```
51043FB0  mov     edx,dword ptr [esp+0Ch]    ; (edx)为串操作字节个数
51043FB4  mov     ecx,dword ptr [esp+4]      ; (ecx)串首地址
51043FB8  test    edx,edx                    ; 操作字节数为 0，要返回
51043FBA  je      5104403B
51043FBC  movzx   eax,byte ptr [esp+8]
……
51043FCB  mov     ecx,dword ptr [esp+0Ch]
51043FCF  push    edi
51043FD0  mov     edi,dword ptr [esp+8]
51043FD4  rep stos  byte ptr es:[edi]
51043FD6  jmp     51044035
……
51044035  mov     eax,dword ptr [esp+8]
51044039  pop     edi
```

```
5104403A  ret
……
```

注意，在不同运行时间看到的机器指令的地址是不同的。在指令的机器码中，也没有使用指令的绝对地址，而是相对地址，详见转移指令的机器码的介绍。

9．6 从源串中取数指令

语句格式：　lods　ops
　　lodsb ——— 从字节串中取数
　　lodsw ———从字串中取数
　　lodsd ———从双字串中取数

功　　能：　字节操作　(ds:[esi]) ->al
　　字操作　　(ds:[esi]) ->ax
　　双字操作　(ds:[esi]) ->eax

将 esi 所指的源串中的字节/字/双字内容送给 al/ax/eax。修改指针 esi，使之指向串中的下一个元素。

当 df＝0 时，　(esi)自增 1(字节操作)、2(字操作) 或 4(双字操作)。
当 df＝1 时，　(esi)自减 1(字节操作)、2(字操作) 或 4(双字操作)。

由于该指令的目的地址为一固定的寄存器，如果带上重复前缀，源串的内容将连续地送入 al、ax 或 eax 中，操作结束后，al、ax 或 eax 中只保存了串中最后一个元素的值，这是没有多大意义的，因此，该指令一般不带重复前缀。

习题 9

9.1 在 C 语言程序中，哪一个库函数封装了串传送指令？

9.2 C 语言的库函数 memcmp、memset、memchr 分别实现什么功能？它们封装了什么样的串操作指令？

9.3 在实现将一个缓冲器区中的内容拷贝到另一个缓冲区时，采用 “rep movsb”的方式实现，有哪些优点？

9.4 编写一个程序，求一个字符串的长度，字符串是以 0 字节结束。要求使用串搜索指令。

9.5 试编写一程序，将以变量 buf 为首址的 1000 个字节存储单元清零，要求使用串操作指令。

9.6 设有以 buf 首地址的字存储区中存放了一个稀疏数组，现要求将数组加以压缩，使其中的非 0 元素仍按序存放在 buf 存储区中，而 0 元素不再出现，试用串操作指令编写实现上述功能的程序。

9.7 删除 STR 串中出现的所有字符‘A’，剩下的字符仍以原有顺序紧凑的存放在 STR 中，分两行显示删除前和删除后的 STR 串。要求使用串操作指令完成核心功能。

9.8 某程序设计语言的关键字存放在以 KEYWORDS 为首址的字节存储区中，每个关键字均以 0 结尾，试利用串操作指令编制一程序，查关键字表，判断存放在以 STR 为首址的字节存储区中的串是否为该语言的关键字。

上机实践 9

9.1　设有两个二维数组 int A[M][N],B[M][N]。用 C 语言编写程序，用不同方法实现将数

组 A 拷贝到数组 B 中。通过对拷贝数据程序段执行计时，比较不同方法的效率差异。

（1）按行序优先的顺序逐个元素拷贝；

（2）按列序优先的顺序逐个元素拷贝；

（3）用 memcpy 函数使用。

试通过观察反汇编代码以及所学计算机工作的原理，解释所看到的运行效率的差异。为了让时间差异较明显，M、N 可取较大的值。另外，比较在二维数组元素个数固定的情况下，不同的 M、N 对结果的影响，试分析解释所看到的结果。

9.2 编写一个程序，求一个字符串的长度。要求不使用串搜索指令，也不能使用逐字符比较的方法。可一次比较 4 个字节，判断这 4 个字节中是否出现 0 字节，若出现 0，则再判断是哪个字节为 0。

提示：参考 9.4 节介绍的方法。C 语言中的函数 strlen 就是采用 9.4 中介绍的方法。