
String Processing

Chapter 9
S. Dandamudi

Outline

- String representation
 - * Using string length
 - * Using a sentinel character
- String instructions
 - * Repetition prefixes
 - * Direction flag
 - * String move instructions
 - * String compare instructions
 - * String scan instructions
- Illustrative examples
 - * LDS and LES instructions
- Examples
 - * str_len
 - * str_cpy
 - * str_cat
 - * str_cmp
 - * str_chr
 - * str_cnv
 - * str_mov
- Indirect procedure call
- Performance: Advantage of string instructions

String Representation

- Two types
 - * Fixed-length
 - * Variable-length
- Fixed length strings
 - * Each string uses the same length
 - » Shorter strings are padded (e.g. by blank characters)
 - » Longer strings are truncated
 - * Selection of string length is critical
 - » Too large ==> inefficient
 - » Too small ==> truncation of larger strings

String Representation (cont'd)

- Variable-length strings
 - * Avoids the pitfalls associated with fixed-length strings
- Two ways of representation
 - * Explicitly storing string length (used in PASCAL)

```
string    DB    'Error message'
```

```
str_len   DW    $-string
```

 - \$ represents the current value of the location counter
 - \$ points to the byte after the last character of **string**
 - * Using a sentinel character (used in C)
 - » Uses NULL character
 - Such NULL-terminated strings are called *ASCIIZ strings*

String Instructions

- Five string instructions

LODS	LOaD String	source
STOS	STORe String	destination
MOVS	MOVE String	source & destination
CMPS	CoMPare String	source & destination
SCAS	SCAn String	destination

- Specifying operands

- * 32-bit segments:

DS:ESI = source operand

ES:EDI = destination operand

- * 16-bit segments:

DS:SI = source operand

ES:DI = destination operand

1998

© S. Dandamudi

String: Page 5

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

String Instructions (cont'd)

- Each string instruction

- * Can operate on 8-, 16-, or 32-bit operands

- * Updates index register(s) automatically

- » Byte operands: increment/decrement by 1

- » Word operands: increment/decrement by 2

- » Doubleword operands: increment/decrement by 4

- Direction flag

- * DF = 0: Forward direction (increments index registers)

- * DF = 1: Backward direction (decrements index registers)

- Two instructions to manipulate DF

std set direction flag (DF = 1)

cld clear direction flag (DF = 0)

1998

© S. Dandamudi

String: Page 6

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

Repetition Prefixes

- String instructions can be repeated by using a repetition prefix
- Two types
 - * Unconditional repetition
rep REPeat
 - * Conditional repetition
 - repe/repz** REPeat while Equal
REPeat while Zero
 - repne/repnz** REPeat while Not Equal
REPeat while Not Zero

Repetition Prefixes (cont'd)

rep

```
while (CX ≠ 0)
    execute the string instruction
    CX := CX-1
end while
```

- CX register is first checked
 - * If zero, string instruction is not executed at all
 - * More like the **JCXZ** instruction

Repetition Prefixes (cont'd)

repe/repz

```
while (CX ≠ 0)
    execute the string instruction
    CX := CX-1
if (ZF = 0)
then
    exit loop
end if
end while
```

- Useful with **cmps** and **scas** string instructions

1998

© S. Dandamudi

String: Page 9

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

Repetition Prefixes (cont'd)

repne/repnz

```
while (CX ≠ 0)
    execute the string instruction
    CX := CX-1
if (ZF = 1)
then
    exit loop
end if
end while
```

1998

© S. Dandamudi

String: Page 10

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Move Instructions

- Three basic instructions
 - * **movs**, **lods**, and **stos**

Move a string (**movs**)

- Format

```
movs    dest_string,source_string  
movsb           ; operands are bytes  
movsw          ; operands are words  
movsd          ; operands are doublewords
```

- First form is not used frequently
 - * Source and destination are assumed to be pointed by DS:(E)SI and ES:(E)DI, respectively

1998

© S. Dandamudi

String: Page 11

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Move Instructions (cont'd)

movsb --- move a byte string

ES:DI:= (DS:SI) ; copy a byte

if (DF=0) ; forward direction

then

SI := SI+1

DI := DI+1

else ; backward direction

SI := SI-1

DI := DI-1

end if

Flags affected: none

1998

© S. Dandamudi

String: Page 12

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Move Instructions (cont'd)

Example

```
.DATA
string1    DB    'The original string',0
strLen     EQU   $ - string1
string2    DB    80 DUP (?)
.CODE
    .STARTUP
    mov     AX,DS           ; set up ES
    mov     ES,AX          ; to the data segment
    mov     CX,strLen      ; strLen includes NULL
    mov     SI,OFFSET string1
    mov     DI,OFFSET string2
    cld                     ; forward direction
    rep     movsb
```

1998

© S. Dandamudi

String: Page 13

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Move Instructions (cont'd)

Load a String (LODS)

- Copies the value from the source string at DS:(E)SI to
 - * AL (**lods**)
 - * AX (**lodsw**)
 - * EAX (**lods**)
- Repetition prefix does not make sense
 - * It leaves only the last value in AL, AX, or EAX register

1998

© S. Dandamudi

String: Page 14

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Move Instructions (cont'd)

lodsb --- load a byte string

AL := (DS:SI) ; copy a byte

if (DF=0) ; forward direction

then

SI := SI+1

else ; backward direction

SI := SI-1

end if

Flags affected: none

1998

© S. Dandamudi

String: Page 15

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Move Instructions (cont'd)

Store a String (STOS)

- Performs the complementary operation

- Copies the value in

- » AL (**lodsb**)

- » AX (**lodsw**)

- » EAX (**lodsd**)

to the destination string at ES:(E)DI

- Repetition prefix can be used if you want to initialize a block of memory

1998

© S. Dandamudi

String: Page 16

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Move Instructions (cont'd)

stosb --- store a byte string

ES:DI := AL ; copy a byte

if (DF=0) ; forward direction

then

DI := DI+1

else ; backward direction

DI := DI-1

end if

Flags affected: none

1998

© S. Dandamudi

String: Page 17

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Move Instructions (cont'd)

Example: Initializes **array1** with -1

```
.DATA
array1    DW    100 DUP (?)
.CODE
.STARTUP
mov     AX,DS           ; set up ES
mov     ES,AX           ; to the data segment
mov     CX,100
mov     DI,OFFSET array1
mov     AX,-1
cld                       ; forward direction
rep     stosw
```

1998

© S. Dandamudi

String: Page 18

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Move Instructions (cont'd)

- In general, repeat prefixes are not useful with **lods** and **stos**
- Used in a loop to do conversions while copying

```
mov    CX,strLen
mov    SI,OFFSET string1
mov    DI,OFFSET string2
cld                                ; forward direction
loop1:
  lodsb
  or    AL,20H
  stosb
  loop  loop1
done:
```

1998

© S. Dandamudi

String: Page 19

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Compare Instruction

cmpsb --- compare two byte strings

Compare two bytes at DS:SI and ES:DI and

set flags

if (DF=0) ; forward direction

then

SI := SI+1

DI := DI+1

else ; backward direction

SI := SI-1

DI := DI-1

end if

Flags affected: As per **cmp** instruction (DS:SI)–(ES:DI)

1998

© S. Dandamudi

String: Page 20

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Compare Instruction (cont'd)

```
.DATA
string1    DB    'abcdefghi',0
strLen     EQU   $ - string1
string2    DB    'abcdefgh',0
.CODE
    .STARTUP
    mov     AX,DS            ; set up ES
    mov     ES,AX           ; to the data segment
    mov     CX,strLen
    mov     SI,OFFSET string1
    mov     DI,OFFSET string2
    cld                     ; forward direction
    repe   cmpsb
    dec     SI
    dec     DI ; leaves SI & DI pointing to the last character that differs
```

1998

© S. Dandamudi

String: Page 21

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Compare Instruction (cont'd)

```
.DATA
string1    DB    'abcdefghi',0
strLen     EQU   $ - string1 - 1
string2    DB    'abcdefgh',0
.CODE
    .STARTUP
    mov     AX,DS            ; set up ES
    mov     ES,AX           ; to the data segment
    mov     CX,strLen
    mov     SI,OFFSET string1 + strLen - 1
    mov     DI,OFFSET string2 + strLen - 1
    std                     ; backward direction
    repne  cmpsb
    inc     SI ; Leaves SI & DI pointing to the first character that matches
    inc     DI ; in the backward direction
```

1998

© S. Dandamudi

String: Page 22

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Scan Instruction

scasb --- Scan a byte string

Compare AL to the byte at ES:DI and set flags

if (DF=0) ; forward direction

then

DI := DI+1

else ; backward direction

DI := DI-1

end if

Flags affected: As per **cmp** instruction (DS:SI)-(ES:DI)

- **scasw** uses AX and **scasd** uses EAX registers instead of AL

1998

© S. Dandamudi

String: Page 23

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Scan Instruction (cont'd)

Example 1

```
.DATA
string1    DB    'abcdefgh',0
strLen     EQU   $ - string1

.CODE
.STARTUP
mov    AX,DS    ; set up ES
mov    ES,AX    ; to the data segment
mov    CX,strLen
mov    DI,OFFSET string1
mov    AL,'e'   ; character to be searched
cld     ; forward direction
repne scasb
dec    DI ; leaves DI pointing to e in string1
```

1998

© S. Dandamudi

String: Page 24

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

String Scan Instruction (cont'd)

Example 2

```
.DATA
string1  DB  '   abc',0
strlen  EQU  $ - string1
.CODE
.STARTUP
mov  AX,DS           ; set up ES
mov  ES,AX           ; to the data segment
mov  CX,strlen
mov  DI,OFFSET string1
mov  AL,' '         ; character to be searched
cld                   ; forward direction
repe scasb
dec  DI ; leaves DI pointing to the first non-blank character a
```

1998

© S. Dandamudi

String: Page 25

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

Illustrative Examples

LDS and LES instructions

- String pointer can be loaded into DS/SI or ES/DI register pair by using **lds** or **les** instructions

- Syntax

```
lds  register,source
```

```
les  register,source
```

* **register** should be a 16-bit register

* **source** is a pointer to a 32-bit memory operand

- **register** is typically SI in **lds** and DI in **les**

1998

© S. Dandamudi

String: Page 26

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

Illustrative Examples (cont'd)

- Actions of **lds** and **les**

lds

```
register := (source)
DS := (source+2)
```

les

```
register := (source)
ES := (source+2)
```

- Pentium also supports **lfs**, **lgs**, and **lss** to load the other segment registers

1998

© S. Dandamudi

String: Page 27

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

Illustrative Examples (cont'd)

- Seven popular string processing routines are given as examples

* **str_len**

* **str_cpy**

* **str_cat**

* **str_cmp**

* **str_chr**

* **str_cnv**

* **str_mov**

1998

© S. Dandamudi

String: Page 28

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

Indirect Procedure Call

- Direct procedure calls specify the offset of the first instruction of the called procedure
- In indirect procedure call, the offset is specified through memory or a register
 - * If BX contains pointer to the procedure, we can use
`call BX`
 - * If the word in memory at `target_proc_ptr` contains the offset of the called procedure, we can use
`call target_proc_ptr`
- These are similar to direct and indirect jumps

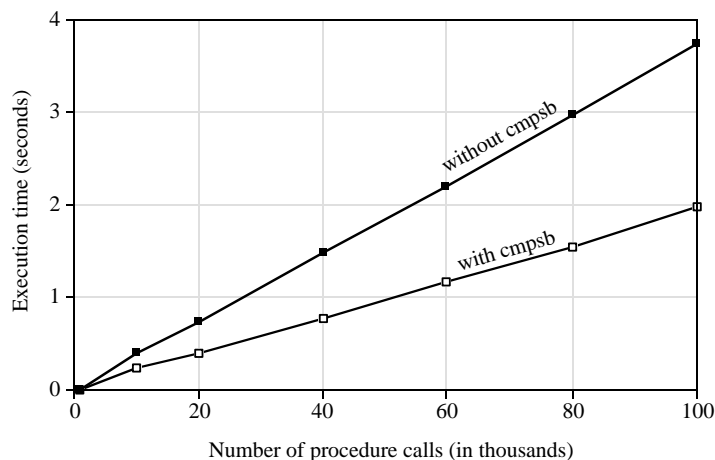
1998

© S. Dandamudi

String: Page 29

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

Performance: Advantage of String Instructions



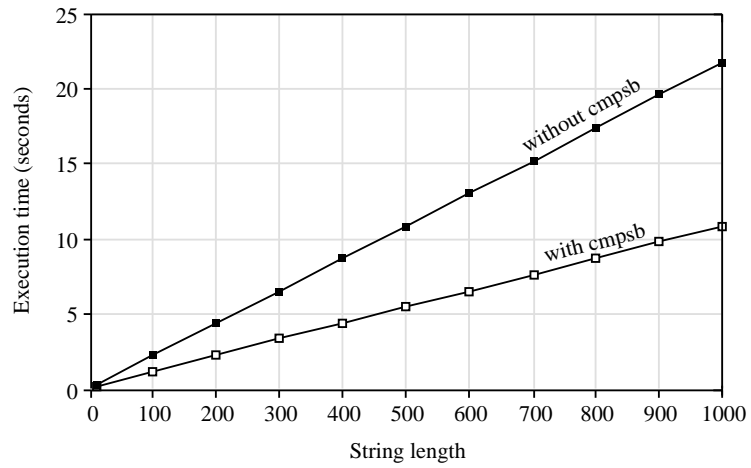
1998

© S. Dandamudi

String: Page 30

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.

Performance: Advantage of String Instructions (cont'd)



1998

© S. Dandamudi

String: Page 31

To be used with S. Dandamudi, "Introduction to Assembly Language Programming." Springer-Verlag, 1998.