

Introduction to 8086 Assembly

Lecture 18

String Instructions

String instructions



K. N. Toosi
University of Technology

- Working with sequence of bytes (words, double-words, quad-words)
- Using **Index** registers
 - ESI (**source** index)
 - EDI (**destination** index)

String instructions



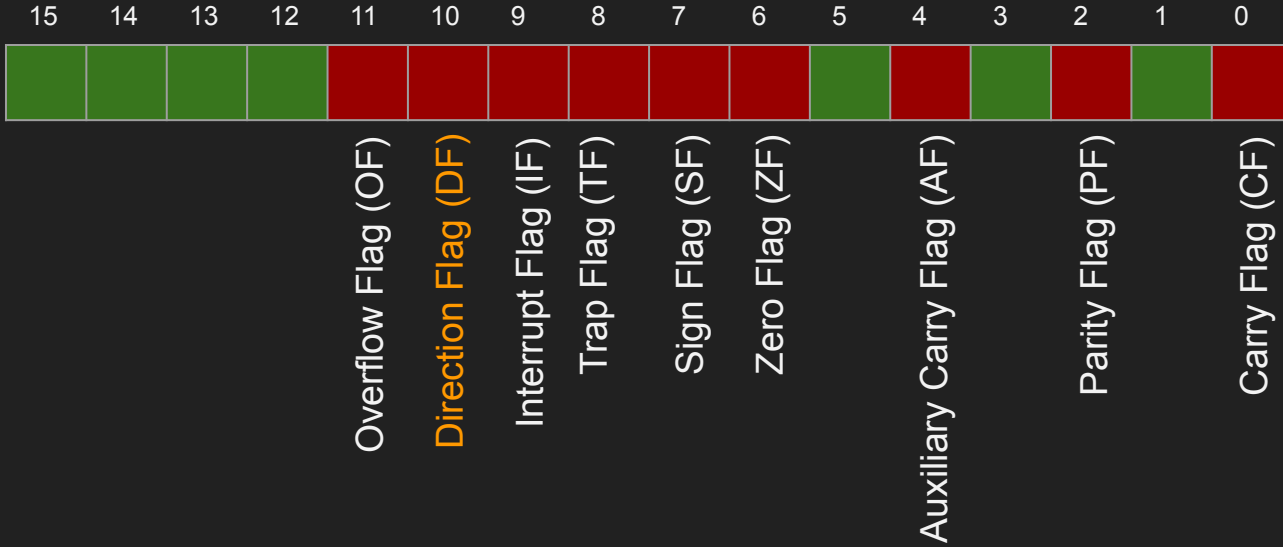
K. N. Toosi
University of Technology

- Working with sequence of bytes (words, double-words, quad-words)
- Using **Index** registers
 - ESI (**source** index)
 - EDI (**destination** index)
- The direction flag
 - CLD (sets DF=0)
 - STD (sets DF=1)

Remember: the FLAGS Register



K. N. Toosi
University of Technology



CF: carry flag

OF: overflow flag

SF: sign flag

ZF: zero flag

PF: parity flag

DF: direction flag

IF: interrupt flag

Storing in a string



K. N. Toosi
University of Technology

	DF = 0	DF = 1
STOSB	<code>mov [EDI], AL</code> <code>inc EDI</code>	<code>mov [EDI], AL</code> <code>dec EDI</code>

Storing in a string



	DF = 0	DF = 1
STOSB	<code>mov [EDI], AL add EDI, 1</code>	<code>mov [EDI], AL sub EDI, 1</code>
STOSW	<code>mov [EDI], AX add EDI, 2</code>	<code>mov [EDI], AX sub EDI, 2</code>
STOSD	<code>mov [EDI], EAX add EDI, 4</code>	<code>mov [EDI], EAX sub EDI, 4</code>

Storing in a string - 64-bit mode



	DF = 0	DF = 1
STOSB	<code>mov [RDI], AL</code> <code>add RDI, 1</code>	<code>mov [RDI], AL</code> <code>sub RDI, 1</code>
STOSW	<code>mov [RDI], AX</code> <code>add RDI, 2</code>	<code>mov [RDI], AX</code> <code>sub RDI, 2</code>
STOSD	<code>mov [RDI], EAX</code> <code>add RDI, 4</code>	<code>mov [RDI], EAX</code> <code>sub RDI, 4</code>
STOSQ	<code>mov [RDI], RAX</code> <code>add RDI, 8</code>	<code>mov [RDI], RAX</code> <code>sub RDI, 8</code>

Example



```
segment .bss  
array1: resd 10
```

```
mov eax, 0  
mov ecx, 10  
mov edi, array1  
cld  
lp:  
stosd  
add eax, 2  
loop lp  
  
push 10  
push array1  
call printArray
```


Example



```
segment .bss  
array1: resd 10
```

```
CS@kntu:lecture18$ ./run.sh test_stosd  
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
```

```
lp:
```

```
mov eax, 0  
mov ecx, 10  
mov edi, array1  
cld  
  
stosd  
add eax, 2  
loop lp  
  
push 10  
push array1  
call printArray
```

Reading a string



	DF = 0	DF = 1
LODSB	<code>mov AL, [ESI] add ESI, 1</code>	<code>mov AL, [ESI] sub ESI, 1</code>
LODSW	<code>mov AX, [ESI] add ESI, 2</code>	<code>mov AX, [ESI] sub ESI, 2</code>
LODSD	<code>mov EAX, [ESI] add ESI, 4</code>	<code>mov EAX, [ESI] sub ESI, 4</code>

Reading a string



```
segment .data
array1: dd 1,2,3,4,5,6,7,8,9,10
array2: times 10 dd 0
```

```
mov ecx, 10
mov esi, array1
mov edi, array2
cld

lp:
lodsd
stosd
loop lp

push 10
push array1
call printArray

push 10
push array2
call printArray
```

Reading a string



```
segment .data
array1: dd 1,2,3,4,5,6,7,8,9,10
array2: times 10 dd 0
```

```
nasihatkon@kntu:code$ ./run test_str
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

```
mov ecx, 10
mov esi, array1
mov edi, array2
cld

lp:

lodsd
stosd
loop lp

push 10
push array1
call printArray

push 10
push array2
call printArray
```

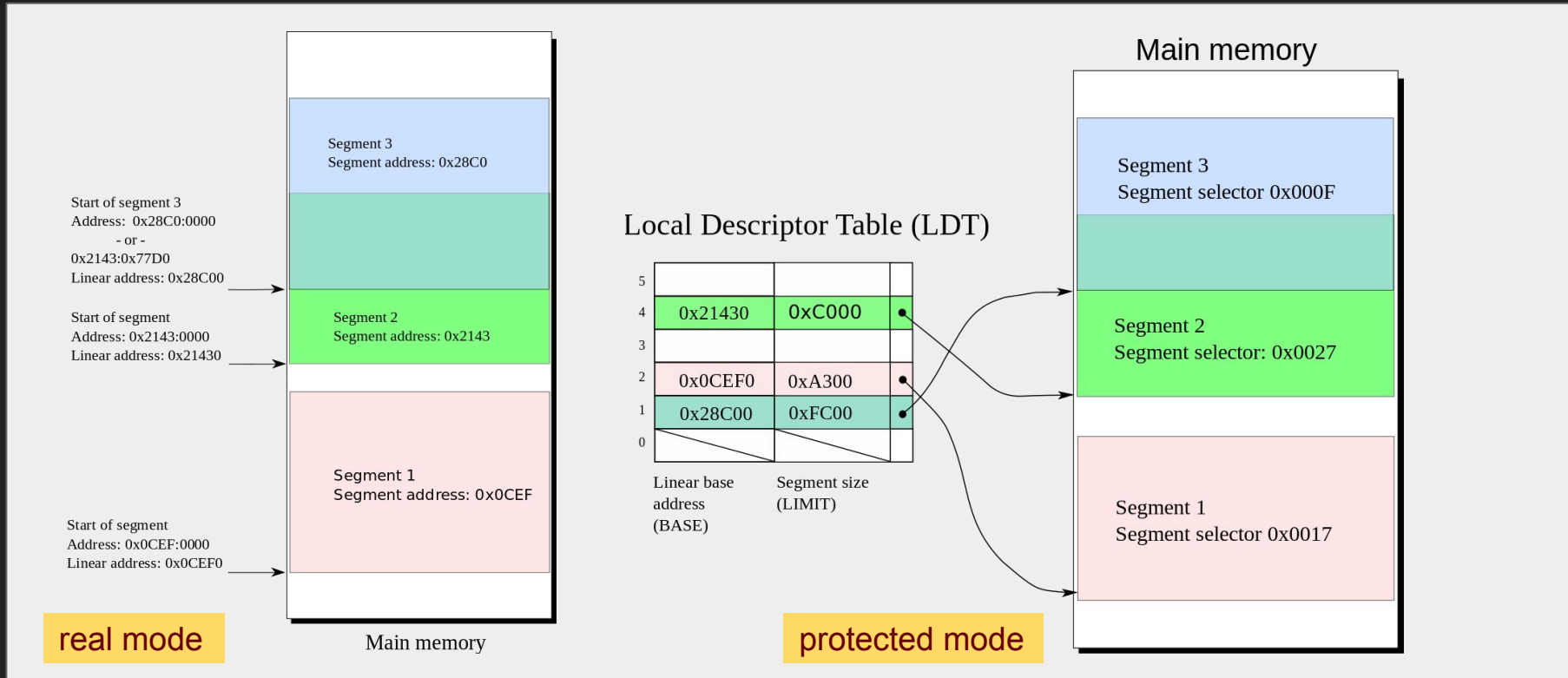
The full story!



	DF = 0	DF = 1
STOSB	<code>mov [ES:EDI], AL add EDI, 1</code>	<code>mov [ES:EDI], AL sub EDI, 1</code>
STOSW	<code>mov [ES:EDI], AX add EDI, 2</code>	<code>mov [ES:EDI], AX sub EDI, 2</code>
STOSD	<code>mov [ES:EDI], EAX add EDI, 4</code>	<code>mov [ES:EDI], EAX sub EDI, 4</code>

	DF = 0	DF = 1
LODSB	<code>mov AL, [DS:ESI] add ESI, 1</code>	<code>mov AL, [DS:ESI] sub ESI, 1</code>
LODSW	<code>mov AX, [DS:ESI] add ESI, 2</code>	<code>mov AX, [DS:ESI] sub ESI, 2</code>
LODSD	<code>mov EAX, [DS:ESI] add ESI, 4</code>	<code>mov EAX, [DS:ESI] sub ESI, 4</code>

Segmentation



The full story!



	DF = 0	DF = 1
STOSB	<code>mov [ES:EDI], AL add EDI, 1</code>	<code>mov [ES:EDI], AL sub EDI, 1</code>
STOSW	<code>mov [ES:EDI], AX add EDI, 2</code>	<code>mov [ES:EDI], AX sub EDI, 2</code>
STOSD	<code>mov [ES:EDI], EAX add EDI, 4</code>	<code>mov [ES:EDI], EAX sub EDI, 4</code>

	DF = 0	DF = 1
LODSB	<code>mov AL, [DS:ESI] add ESI, 1</code>	<code>mov AL, [DS:ESI] sub ESI, 1</code>
LODSW	<code>mov AX, [DS:ESI] add ESI, 2</code>	<code>mov AX, [DS:ESI] sub ESI, 2</code>
LODSD	<code>mov EAX, [DS:ESI] add ESI, 4</code>	<code>mov EAX, [DS:ESI] sub ESI, 4</code>

string copy instructions



	DF = 0	DF = 1
MOVSB	<pre>mov BYTE [EDI], [ESI] add ESI, 1 add EDI, 1</pre>	<pre>mov BYTE [EDI], [ESI] sub ESI, 1 sub EDI, 1</pre>
MOVSW	<pre>mov WORD [EDI], [ESI] add ESI, 2 add EDI, 2</pre>	<pre>mov WORD [EDI], [ESI] sub ESI, 2 sub EDI, 2</pre>
MOVSD	<pre>mov DWORD [EDI], [ESI] add ESI, 4 add EDI, 4</pre>	<pre>mov DWORD [EDI], [ESI] sub ESI, 4 sub EDI, 4</pre>

`mov [EDI], [ESI]` is for illustration
(`mov mem, mem` is invalid)

string copy instructions: full story



	DF = 0	DF = 1
MOVSB	<pre>mov BYTE [ES:EDI], [DS:ESI] add ESI, 1 add EDI, 1</pre>	<pre>mov BYTE [ES:EDI], [DS:ESI] sub ESI, 1 sub EDI, 1</pre>
MOVSW	<pre>mov WORD [ES:EDI], [DS:ESI] add ESI, 2 add EDI, 2</pre>	<pre>mov WORD [ES:EDI], [DS:ESI] sub ESI, 2 sub EDI, 2</pre>
MOVSD	<pre>mov DWORD [ES:EDI], [DS:ESI] add ESI, 4 add EDI, 4</pre>	<pre>mov DWORD [ES:EDI], [DS:ESI] sub ESI, 4 sub EDI, 4</pre>

`mov [ES:EDI], [DS:ESI]` is for illustration
(`mov mem, mem` is invalid)

Reading a string



```
mov ecx, 10  
mov esi, array1  
mov edi, array2  
cld
```

lp:

```
lodsd  
stosd  
loop lp
```

```
push 10  
push array1  
call printArray
```

```
push 10  
push array2  
call printArray
```

```
mov ecx, 10  
mov esi, array1  
mov edi, array2  
cld
```

lp:

```
movsd  
loop lp
```

```
push 10  
push array1  
call printArray
```

```
push 10  
push array2  
call printArray
```

The `rep` instruction prefix



```
mov ecx, 10
mov esi, array1
mov edi, array2
cld
```

lp:

```
lodsd
stosd
loop lp
```

```
push 10
push array1
call printArray
```

```
push 10
push array2
call printArray
```

```
mov ecx, 10
mov esi, array1
mov edi, array2
cld
```

lp:

```
movsd
loop lp
```

```
push 10
push array1
call printArray
```

```
push 10
push array2
call printArray
```

```
mov ecx, 10
mov esi, array1
mov edi, array2
cld
```

```
rep movsd
```

```
push 10
push array1
call printArray
```

```
push 10
push array2
call printArray
```

REPx instruction prefixes



K. N. Toosi
University of Technology

REPE, REPZ (repeat while equal/zero)

REPNE, REPNZ (repeat while not equal/not zero)

Searching strings



	DF = 0	DF = 1
SCASB	<code>cmp AL, [EDI] (sets FLAGS)</code> <code>add EDI, 1 (FLAGS unchanged)</code>	<code>cmp AL, [EDI] (sets FLAGS)</code> <code>sub EDI, 1 (FLAGS unchanged)</code>
SCASW	<code>cmp AX, [EDI] (sets FLAGS)</code> <code>add EDI, 2 (FLAGS unchanged)</code>	<code>cmp AX, [EDI] (sets FLAGS)</code> <code>sub EDI, 2 (FLAGS unchanged)</code>
SCASD	<code>cmp EAX, [EDI] (sets FLAGS)</code> <code>add EDI, 4 (FLAGS unchanged)</code>	<code>cmp EAX, [EDI] (sets FLAGS)</code> <code>sub EDI, 4 (FLAGS unchanged)</code>

`[EDI] => [ES:EDI]`

Searching for an element in array



```
segment .data
array1: dd    10,11,12,13,14,15,16,17,18,19

        LEN equ ($-array1)/4

segment .text
global asm_main

asm_main:
    pusha

    push LEN
    push array1
    call printArray
```

Searching for an element in array




```
segment .data
array1: dd 10,11,12,13,14,15,16,17,18,19

        LEN equ ($-array1)/4

segment .text
global asm_main

asm_main:
    pusha

    push LEN
    push array1
    call printArray
```

 current address

Searching for an element in array



```
    call read_int

    mov edi, array1
    mov ecx, LEN
    cld

loop1:
    scasd

    je    endloop1
    loop loop1

endloop1:
```


Searching for an element in array



```
    call read_int

    mov edi, array1
    mov ecx, LEN
    cld

loop1:
    scasd
    je  endloop1
    loop loop1

endloop1:
    je found
    mov eax, -1
    jmp print_eax

found:
    mov eax, edi
    sub eax, array1+4 ← why?
    shr eax, 2      ; eax /= 4:

print_eax:
    call print_int
    call print_nl
```

REPx instructions



```
call read_int

mov edi, array1
mov ecx, LEN
cld

loop1:
    scasd
    je endloop1
    loop loop1

endloop1:
    je found
    mov eax, -1
    jmp print_eax

found:
    mov eax, edi
    sub eax, array1+4
    shr eax, 2 ; eax /= 4:

print_eax:
    call print_int
    call print_nl
```

```
call read_int

mov edi, array1
mov ecx, LEN
cld

repne scasd

    je found
    mov eax, -1
    jmp print_eax

found:
    mov eax, edi
    sub eax, array1+4
    shr eax, 2 ; eax /= 4:

print_eax:
    call print_int
    call print_nl
```

Comparing strings



	DF = 0	DF = 1
CMPSB	<code>cmp BYTE [EDI], [ESI] (sets FLAGS)</code> <code>add ESI, 1 (FLAGS unchanged)</code> <code>add EDI, 1 (FLAGS unchanged)</code>	<code>cmp BYTE [EDI], [ESI] (sets FLAGS)</code> <code>sub ESI, 1 (FLAGS unchanged)</code> <code>sub EDI, 1 (FLAGS unchanged)</code>
CMPSW	<code>cmp WORD [EDI], [ESI] (sets FLAGS)</code> <code>add ESI, 2 (FLAGS unchanged)</code> <code>add EDI, 2 (FLAGS unchanged)</code>	<code>cmp WORD [EDI], [ESI] (sets FLAGS)</code> <code>sub ESI, 2 (FLAGS unchanged)</code> <code>sub EDI, 2 (FLAGS unchanged)</code>
CMPSD	<code>cmp DWORD [EDI], [ESI] (sets FLAGS)</code> <code>add ESI, 4 (FLAGS unchanged)</code> <code>add EDI, 4 (FLAGS unchanged)</code>	<code>cmp DWORD [EDI], [ESI] (sets FLAGS)</code> <code>sub ESI, 4 (FLAGS unchanged)</code> <code>sub EDI, 4 (FLAGS unchanged)</code>

[ESI] => [DS:ESI]

[EDI] => [ES:EDI]

Comparing strings, strcmp



```
segment .data
s1:      db  "Behnam", 0
s2:      db  "Behrooz", 0
```

```
mov edi, s2

; compute length of s2
cld
mov ecx, 0xFFFFFFFF ; large number (or zero)
mov al, 0
repne scasb

sub edi, s2+1
mov ecx, edi ; ecx = strlen(s2)

mov esi, s1
mov edi, s2

repe cmpsb

mov al, [esi-1]
sub al, [edi-1]

movsx eax, al
call print_int
call print_nl
```

Inline Example



str_inline.c

```
char s1[] = "Only from the heart can you touch the sky!";
char s2[100];

int n = strlen(s1);

asm volatile ("cld;"
              "rep movsb"
              :
              : "S" (s1), "D" (s2), "c" (n+1)
              : "cc", "memory"
              );

puts(s1);
puts(s2);
```

Inline Example



str_inline.c

```
char s1[] = "Only from the heart can you touch the sky!";
char s2[100];

int n = strlen(s1);

asm volatile ("cld;"
             "rep movsb"
             :
             : "S" (s1), "D" (s2), "c" (n+1)
             : "cc", "memory"
             );
```

```
b.nasihatkon@kntu:lecture18$ gcc -m32 -masm=intel str_inline.c && ./a.out
Only from the heart can you touch the sky!
Only from the heart can you touch the sky!
```