

Lab Session 11

String Handling Instructions

Primitive String Instructions

- The x86 instruction set has five groups of instructions for processing arrays of bytes, words, and double words.
- Each instruction implicitly uses ESI, EDI, or both registers to address memory.
- References to the accumulator imply the use of AL, AX, or EAX, depending on the instruction data size. String primitives execute efficiently because they automatically repeat and increment array indexes.

String Primitive Instructions.

Instruction	Description
MOVS _B , MOVSW, MOVSD	Move string data: Copy data from memory addressed by ESI to memory addressed by EDI.
CMPS _B , CMPSW, CMPSD	Compare strings: Compare the contents of two memory locations addressed by ESI and EDI.
SCAS _B , SCASW, SCASD	Scan string: Compare the accumulator (AL, AX, or EAX) to the contents of memory addressed by EDI.
STOS _B , STOSW, STOSD	Store string data: Store the accumulator contents into memory addressed by EDI.
LDS _B , LODSW, LODSD	Load accumulator from string: Load memory addressed by ESI into the accumulator.

1. MOVS_B, MOVSW, and MOVSD

The MOVS_B, MOVSW, and MOVSD instructions copy data from the memory location pointed to by ESI to the memory location pointed to by EDI. The two registers are either incremented or decremented automatically (based on the value of the Direction flag):

MOVS _B	Move (copy) bytes
MOVSW	Move (copy) words
MOVSD	Move (copy) doublewords

Using a Repeat Prefix

By itself, a string primitive instruction processes only a single memory value or pair of values. If you add a repeat prefix, the instruction repeats, using ECX as a counter. The repeat prefix permits you to process an entire array using a single instruction. The following repeat prefixes are used:

REP	Repeat while ECX > 0
REPZ, REPE	Repeat while the Zero flag is set and ECX > 0
REPNZ, REPNE	Repeat while the Zero flag is clear and ECX > 0

Example #1: Copy a String

In the following example, MOVSB moves 10 bytes from string1 to string2. The repeat prefix first tests ECX > 0 before executing the MOVSB instruction. If ECX = 0, the instruction is ignored and control passes to the next line in the program. If ECX > 0, ECX is decremented and the instruction repeats:

```
INCLUDE Irvine32.inc
.data

string1 BYTE 'this is first string',0
string2 BYTE 'this is second string',0

.code
main PROC
    cld      ; clear direction flag
    mov esi,OFFSET string1    ; ESI points to source
    mov edi,OFFSET string2    ; EDI points to target
    mov ecx,sizeof string1    ; set counter to 10
    rep movsb                 ; move bytes
    mov edx,offset string2    ; changed String
    call writestring
main ENDP
END main
```

ESI and EDI are automatically incremented when MOVSB repeats. This behavior is controlled by the CPU's Direction flag.

Direction Flag

String primitive instructions increment or decrement ESI and EDI based on the state of the Direction flag. The Direction flag can be explicitly modified using the CLD and STD instructions:

```
CLD      ; clear Direction flag (forward direction)
STD      ; set Direction flag (reverse direction)
```

Direction Flag Usage in String Primitive Instructions.

Value of the Direction Flag	Effect on ESI and EDI	Address Sequence
Clear	Incremented	Low-high
Set	Decrement	High-low

2. CMPSB, CMPSW, and CMPSD

The CMPSB, CMPSW, and CMPSD instructions each compare a memory operand pointed to by ESI to a memory operand pointed to by EDI. You can use a repeat prefix with CMPSB, CMPSW, and CMPSD. The Direction flag determines the incrementing or decrementing of ESI and EDI.

Example # 2: Comparing Doublewords

Suppose you want to compare a pair of double words using CMPSD. In the following example, source has a smaller value than target, so the JA instruction will not jump to label L1.

```
INCLUDE Irvine32.inc
.data

greater BYTE 'source > target',0
lessOrEqual BYTE 'source <target',0
source BYTE 'abcd',0
target BYTE 'abc',0

.code
main PROC
    mov esi,OFFSET source
    mov edi,OFFSET target
    cmpsd                ; compare doublewords
    ja L1                ; jump if source > target
    mov edx,offset lessOrEqual ;else print source <= target
    call writestring
    jmp endd
L1:
    mov edx,offset greater
    call writestring

endd:
exit
main ENDP
END main
```

3. SCASB, SCASW, and SCASD

The SCASB, SCASW, and SCASD instructions compare a value in AL/AX/EAX to a byte, word, or double word, respectively, addressed by EDI. The instructions are useful when looking for a single value in a string or array. Combined with the REPE (or REPZ) prefix, the string or array is scanned while ECX > 0 and the value in AL/ AX/ EAX match each subsequent value in memory. The REPNE prefix scans until either AL/AX/EAX matches a value in memory or ECX = 0.

Example #3: Scan for a Matching Character

```
INCLUDE Irvine32.inc
.data

alpha BYTE "ABCDEFGH",0

.code
main PROC
    mov edi,OFFSET alpha      ; EDI points to the string
    mov al,'F'                ; search for the letter F
    mov ecx,LENGTHOF alpha   ; set the search count
    cld                       ; direction = forward
    repne scasb               ; repeat while not equal
    jnz quit                  ; quit if letter not found
    dec edi                   ; found: back up EDI

quit:
    exit
main ENDP
END main
```

JNZ was added after the loop to test for the possibility that the loop stopped because ECX = 0 and the character in AL was not found.

4. STOSB, STOSW, and STOSD

The STOSB, STOSW, and STOSD instructions store the contents of AL/AX/EAX, respectively, in memory at the offset pointed to by EDI. EDI is incremented or decremented based on the state of the Direction flag. When used with the REP prefix, these instructions are useful for filling all elements of a string or array with a single value. For example, the following code initializes each byte in string1 to 0FFh:

```
.data
Count = 100
string1 BYTE Count DUP(?)
.code
    mov al,0FFh                ; value to be stored
    mov edi,OFFSET string1     ; EDI points to target
    mov ecx,Count              ; character count
    cld                        ; direction = forward
    rep stosb                  ; fill with contents of AL
```

5. LODSB, LODSW, and LODSD

The LODSB, LODSW, and LODSD instructions load a byte or word from memory at ESI into AL/AX/EAX, respectively. ESI is incremented or decremented based on the state of the Direction flag. The REP prefix is rarely used with LODS because each new value loaded into the accumulator overwrites its previous contents. Instead, LODS is used to load a single value. In the next example, LODSB substitutes for the following two instructions (assuming the Direction flag is clear):

```
mov al,[esi]      ; move byte into
inc esi           ; point to next byte
```

Example#4: Array Multiplication:

The following program multiplies each element of a doubleword array by a constant value. LODSD and STOSD work together:

```
TITLE Multiply an Array (Mult.asm)
;      This program multiplies each element of an array
;      of 32-bit integers by a constant value.
INCLUDE Irvine32.inc

.data
array DWORD 1,2,3,4,5,6,7,8,9,10      ; test data
multiplier DWORD 10                    ; test data

.code
main PROC
Cld                                     ; direction = forward
mov esi,OFFSET array                   ; source index
mov edi,esi                             ; destination index
mov ecx,LENGTHOF array                 ; loop counter
L1:
Lodsd                                  ; load [ESI] into EAX
mul multiplier                          ; multiply by a value
Stosd                                   ; store EAX into [EDI]
loop L1
mov esi , OFFSET array
mov ecx, LENGTHOF array
mov ebx, TYPE array
call dumpmem                            ; updated array Display
Exit
main ENDP
END main
```

String Procedures

1. STR_COPY

The Str_copy procedure copies a null-terminated string from a source location to a target location.

Syntax: `INVOKE Str_copy, ADDR source, ADDR target`

2. STR_LENGTH

The Str_length procedure returns the length of a string in the EAX register. When you call it, pass the string's offset.

Syntax: `INVOKE Str_length, ADDR myString`

3. STR_COMPARE

The Str_compare procedure compares two strings. It affects the CF and ZF as shown in the following table.

Syntax: `INVOKE Str_compare, ADDR string1, ADDR string2`

4. Str_trim Procedure

The Str_trim procedure removes all occurrences of a selected trailing character from a null terminated string.

Syntax: `INVOKE Str_trim, ADDR string, char_to_trim`

5. Str_ucase Procedure

The Str_ucase procedure converts a string to all uppercase characters. It returns no value. When you call it, pass the offset of a string:

Syntax: `INVOKE Str_ucase, ADDR myString`

String Library Demo Program

```
INCLUDE Irvine32.inc
.data
string_1 BYTE "abcde////",0
string_2 BYTE "ABCDE",0
msg0 BYTE "string_1 in upper case: ",0
msg1 BYTE "string_1 and string_2 are equal",0
msg2 BYTE "string_1 is less than string_2",0
msg3 BYTE "string_2 is less than string_1",0
msg4 BYTE "Length of string_2 is ",0
msg5 BYTE "string_1 after trimming: ",0
.code
main PROC
    call trim_string
```

```
call upper_case
call compare_strings
call print_length
exit
main ENDP
```

```
trim_string PROC
```

```
                ; Remove trailing characters from string_1.
```

```
INVOKE Str_trim, ADDR string_1, '/'
```

```
mov edx,OFFSET msg5
```

```
call WriteString
```

```
mov edx,OFFSET string_1
```

```
call WriteString
```

```
call CrLf
```

```
ret
```

```
trim_string ENDP
```

```
upper_case PROC
```

```
                ; Convert string_1 to upper case.
```

```
mov edx,OFFSET msg0
```

```
call WriteString
```

```
INVOKE Str_ucase, ADDR string_1
```

```
mov edx,OFFSET string_1
```

```
call WriteString
```

```
call CrLf
```

```
ret
```

```
upper_case ENDP
```

```
compare_strings PROC
```

```
                ; Compare string_1 to string_2.
```

```
INVOKE Str_compare, ADDR string_1, ADDR string_2
```

```
.IF ZERO?
```

```
mov edx,OFFSET msg1
```

```
.ELSEIF CARRY?
```

```
mov edx,OFFSET msg2                ; string 1 is less than...
```

```
.ELSE
```

```
mov edx,OFFSET msg3                ; string 2 is less than...
```

```
.ENDIF
```

```
call WriteString
```

```
call CrLf
```

```
ret
```

```
compare_strings ENDP
```

```
print_length PROC
```

```
                ; Display the length of string_2.
```

```
mov edx,OFFSET msg4
```

```
call WriteString
INVOKE Str_length, ADDR string_2
call WriteDec
call Crlf
ret
print_length ENDP
END main
```

Exercise

1. Write a program to find the index(location) of the first occurrence of the character '#' in the given string.

```
Str1 BYTE '127&j~3#^&***#45^',0
```

2. Repeat the task 1 by creating a procedure named scan_char. Call the procedure to find the index(location) of the first occurrence of the character '#' in the given string.

3. Create IsCompare procedure to compare two strings.

4. Create *Move* procedure to perform move operation on two strings.

5. Create a Str_Reverse procedure to reverse strings.

6. Create a procedure that Loads an array of integer by multiplying it with 3. Display updated Array.

Lab 11

Task1: Code+Output

```
1 Include Irvine32.inc
2 .data
3 str1 byte '127&j~3#^&*#*#45^',0
4
5 .code
6 main PROC
7 mov edi,offset str1
8 mov al,'#'
9 mov ecx,lengthof str1
10 cld
11 repne scasb
12 jnz quit
13 dec edi
14 sub edi,offset str1
15 call dumpregs
16
17 quit:
18 exit
19 main ENDP
20 end main
```

Microsoft Visual Studio Debug Console

EAX=28BBB623 EBX=00778000 ECX=0000000A EDX=0116100A
ESI=0116100A EDI=00000007 EBP=008FFDA4 ESP=008FFD94
EIP=0116367D EFL=00000202 CF=0 SF=0 ZF=0 OF=0 AF=0 PF=0

C:\Users\student\source\repos\Project7\Debug\Project7.exe (process 38512) exited with code 0.
Press any key to close this window . . .

Task2: Code+Output

```
2 .data
3 str1 byte '127&j~3#^&*#*#45^',0
4 myArrLen dword ?
5
6 .code
7
8 scan_char PROTO copyi: ptr byte, Len:dword
9
10 main PROC
11 mov myArrLen, lengthof str1
12
13 invoke scan_char, ADDR str1, myArrLen
14
15 exit
16 main ENDP
17
18 scan_char PROC copyi:ptr byte, Len:dword
19 mov edi,copyi
20 mov al,'#'
21 mov ecx,Len
22 cld
```

EAX=8DB56923 EBX=00A70000 ECX=0000000A EDX=002E100A
ESI=002E100A EDI=00000007 EBP=00CFFE04 ESP=00CFFE04
EIP=002E369A EFL=00000202 CF=0 SF=0 ZF=0 OF=0 AF=0 PF=0

C:\Users\student\source\repos\Project7\Debug\Project7.exe (process 37280) exited with code 0.
Press any key to close this window . . .

Task3: Code+Output

```
1 Include Irvine32.inc
2
3 .data
4 str1 byte "racecar",0
5 str2 byte "raceca",0
6 str1Len dword ?
7 str2Len dword ?
8 mess1 byte "Sadly It is Not Equal",0
9 mess2 byte "Sadly It is Equal",0
10
11
12 .code
13
14 isCompare PROTO copy1:ptr byte, copy2:ptr byte, len1:dword, len2:dword
15
16 main PROC
17
18 mov str1Len, lengthof str1
19 mov str2Len, lengthof str2
20
21
22 invoke isCompare, ADDR str1, ADDR str2, str1Len, str2Len
```

Microsoft Visual Studio Debug Console

Sadly It is Not Equal
C:\Users\student\source\repos\Project7\Debug\Project7.exe (process 61068) exited with code 0.
Press any key to close this window . . .

2% No issues found

Output
Show output from: Build

Lab 11

Task4: Code+Output

```
18 exit
19 main ENDP
20
21 Moving PROC source:ptr byte, des:ptr byte, len1:dword
22 mov ecx, len1
23 cld
24 mov esi,source
25 mov edi,des
26 l1:
27 lodsb
28 ;call dumpregs
29 stosb
30 loop l1
31
32 mov edx, des
33 call writestring
34
35
36
37 ret
38 Moving ENDP
39 end main
```

Microsoft Visual Studio Debug Console

Why Do I Exist Only To Suffer
C:\Users\acer\source\repos\Practice\Debug\Practice.exe (process 17600) exited with
Press any key to close this window . . .

Task5: Code+Output

```
14 mov edx,offset str3
15 call writestring
16 mov ecx,32
17 mov edx,offset str1
18 call readstring
19 mov var1,eax
20
21 invoke Str_Reverse, addr str1,addr str2,var1
22 mov edx,offset str2
23 call writestring
24
25 exit
26 main ENDP
27
28 Str_Reverse PROC string1:ptr byte, string2:ptr byte, len:dword
29 mov esi, string1
30 add esi,len
31 sub esi,1
32 mov edi, string2
33 std
34 mov ecx,len
35
```

Microsoft Visual Studio Debug Console

Enter Your String:Hello
olleH
C:\Users\acer\source\repos\Practice\Debug\Practice.exe (process 8240) exited with code 0.
Press any key to close this window . . .

Task6: Code+Output

```
17 mov ecx,array
18 call dumpmem
19
20 exit
21 main ENDP
22
23 arrMul PROC array:ptr dword, mux:dword
24 cld
25
26 mov esi,array
27 mov edi,esi
28 mov ecx,len
29 l1:
30 lodsd
31 mul mux
32 stosd
33 loop l1
34 ret
35 arrMul ENDP
36 end main
```

Microsoft Visual Studio Debug Console

Dump of offset 00536000

00000003 00000006 00000009 0000000C 0000000F 00000012 00000015 00000018 0000001B 0000001E
C:\Users\acer\source\repos\Practice\Debug\Practice.exe (process 9984) exited with code 0.
Press any key to close this window . . .