EL213: Computer Org. & Assembly Language Lab

# Lab#13: Strings and Arrays

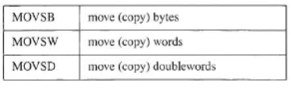
## Agenda

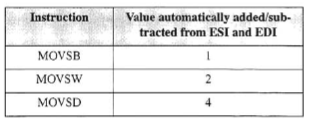
* String Primitive Instructions
  + MOVSB, MOVSW, and MOVSD
  + CMPSB, CMPSW, and CMPSD
  + SCASB, SCASW, and SCASD
  + STOSB, STOSW, and STOSD
  + LODSB, LODSW, and LODSD
* String Procedures
  + Str\_compare Procedure
  + Str\_Iength Procedure
  + Str\_copy Procedure
  + Str\_trim Procedure
  + Str\_ucase Procedure
* Searching and Sorting Integer Arrays
  + Bubble Sort
  + Binary Search

## String Primitive Instructions

### MOVSB, MOVSW, and MOVSD

The MOVSB, 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):





include irvine32.inc

.data

source DWORD 20 DUP(0FFFFFFFFh)

target DWORD 20 DUP(?)

.code

main PROC

cld ;direction = forward

mov ecx,LENGTHOF source ;set REP counter

mov esi,OFFSET source ;ESI points to source

mov edi,OFFSET target ;EDI points to target

rep movsd ;copy doublewords

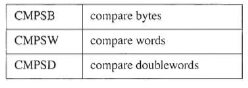
exit

main ENDP

END main

### 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:



Suppose we want to compare a pair of double words using CMPSD. In the following sample data, we see that source is less than target. When JA exec utes, the conditional jump is not taken; the JMP instruction is executed instead:

include irvine32.inc

.data

source DWORD 1234h

target DWORD 5678h

msg\_g byte "Source>Target",0

msg\_l byte "Source<=Target",0

.code

main PROC

mov esi ,OFFSET source

mov edi,OFFSET target

cmpsd ;compare doublewords

ja L1 ;jump if source > target

jmp L2 ;jump, since source <= target

L1:

mov edx, OFFSET msg\_g

call writestring

call crlf

jmp quit

L2:

mov edx, OFFSET msg\_l

call writestring

call crlf

quit:

exit

main ENDP

END main

**Output**



### SCASB, SCASW, and SCASD

The SCASB, SCASW, and SCASD instructions compare a value in AL/AX/EAX to a byte, word , or doubleword , respectively, addressed by EDI.

INCLUDE Irvine32.inc

.data

alpha BYTE "ABCDEFGH " , 0

msg byte "Letter Found", 0

.code

main PROC

mov edi ,OFFSET alpha

mov al, 'F'

mov ecx , LENGTHOF alpha

cld

repne scasb

jnz quit

mov edx, OFFSET msg

call writestring

call crlf

quit:

exit

main ENDP

END main

**Output**



### STOSB, STOSW, and STOSD

The STOSB, STOSW, and STOSD instructions store the contents of ALiAX/EAX, respectively. in memory at the offset pointed to by EDI.

INCLUDE Irvine32.inc

.data

Count = 100

string1 BYTE Count DUP(?)

.code

main PROC

mov al,0FFh

mov edi,OFFSET string1

mov ecx,Count

cld

rep stosb

exit

main ENDP

END main

### LODSB, LODSW, and LODSD

The LODSB, LODSW, and LODSD instructions load a byte or word from memory at ESI into AL/AX/EAX, respectively.

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

multiplier DWORD 10

.code

main PROC

cld ;direction = forward

mov esi,OFFSET array ;source index

mov edi,esi ;destination index

mov ecx,LENGTHOF array ;loop counters

L1 : lodsd ;load [ESI] into EAX

mul multiplier ;multiply by a value

stosd ;store EAX into [EDI]

loop L1

exit

main ENDP

END main

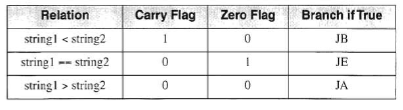
## String Procedures

### Str\_compare Procedure

The Str\_compare procedure compares two strings. The calling format is:

INVOKE Str\_compare, ADDR stringl, ADDR string2

The strings are compared byte by byte, using their 8-bit integer ASCII codes. The comparison is case-sensitive because ASCII codes are different for uppercase and lowercase letters. The procedure does not return a value, but the Carry and Zero flags can be interpreted as follows (using the string1 and string2 arguments):



TITLE Comparing Strings (Compare.asm)

INCLUDE Irvine32.inc

.data

string\_1 BYTE "ABCDEFG",0

string\_2 BYTE "ABCDEFG",0

string\_3 BYTE 0

string\_4 BYTE 0

.code

main PROC

call Clrscr

INVOKE Str\_compare, ADDR string\_4, ADDR string\_3

Call DumpRegs

exit

main ENDP

END main

### Str\_Iength Procedure

The Str\_Iength procedure returns the length of a string in the EAX register. When you call it, pass the offset of a string. For example:

INVOKE Str\_length, ADDR myString

TITLE String Length (Length.asm)

; Testing the Str\_length procedure.

INCLUDE Irvine32.inc

.data

string\_1 BYTE "Hello",0

string\_2 BYTE "#",0

string\_3 BYTE 0

.code

main PROC

call Clrscr

INVOKE Str\_length,ADDR string\_1

call DumpRegs

INVOKE Str\_length,ADDR string\_2

call DumpRegs

INVOKE Str\_length,ADDR string\_3

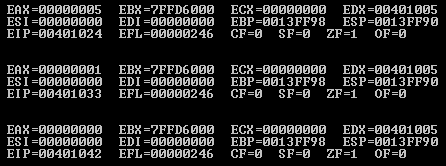
call DumpRegs

exit

main ENDP

END main

**Output**



### Str\_copy Procedure

Tne Str\_copy procedure copies a null-terminated string from a source location to a target location. Before calling this procedure, you must make sure the target operand is large enough to hold the copied string. The syntax for calling Str\_copy is:

INVOKE Str\_copy , ADDR source, ADDR target

TITLE Copying Strings (CopyStr.asm)

; Testing the Str\_copy procedure

INCLUDE Irvine32.inc

.data

string\_1 BYTE "ABCDEFG",0

string\_2 BYTE 100 DUP(?)

.code

main PROC

call Clrscr

INVOKE Str\_copy, ; copy string\_1 to string\_2

ADDR string\_1,

ADDR string\_2

mov edx,OFFSET string\_2

call WriteString

call Crlf

exit

main ENDP

END main

**Output**



### Str\_trim Procedure

The Str\_trim procedure removes all occurrences of a selected trailing character from a null terminated string. You might use it, for example, to remove all spaces from the end of a string.

TITLE Trim Trailing Characters (Trim.asm)

; Test the Trim procedure. Trim removes trailing all

; occurences of a selected character from the end of

; a string.

INCLUDE Irvine32.inc

Str\_trim PROTO,

pString:PTR BYTE, ; points to string

char:BYTE ; character to remove

Str\_length PROTO,

pString:PTR BYTE ; pointer to string

ShowString PROTO,

pString:PTR BYTE

.data

; Test data:

string\_1 BYTE 0 ; case 1

string\_2 BYTE "#",0 ; case 2

string\_3 BYTE "Hello###",0 ; case 3

string\_4 BYTE "Hello",0 ; case 4

string\_5 BYTE "H#",0 ; case 5

string\_6 BYTE "#H",0 ; case 6

.code

main PROC

call Clrscr

INVOKE Str\_trim,ADDR string\_1,'#'

INVOKE ShowString,ADDR string\_1

INVOKE Str\_trim,ADDR string\_2,'#'

INVOKE ShowString,ADDR string\_2

INVOKE Str\_trim,ADDR string\_3,'#'

INVOKE ShowString,ADDR string\_3

INVOKE Str\_trim,ADDR string\_4,'#'

INVOKE ShowString,ADDR string\_4

INVOKE Str\_trim,ADDR string\_5,'#'

INVOKE ShowString,ADDR string\_5

INVOKE Str\_trim,ADDR string\_6,'#'

INVOKE ShowString,ADDR string\_6

exit

main ENDP

;-----------------------------------------------------------

ShowString PROC USES edx, pString:PTR BYTE

; Display a string surrounded by brackets.

;-----------------------------------------------------------

.data

lbracket BYTE "[",0

rbracket BYTE "]",0

.code

mov edx,OFFSET lbracket

call WriteString

mov edx,pString

call WriteString

mov edx,OFFSET rbracket

call WriteString

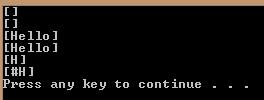
call Crlf

ret

ShowString ENDP

END main

**Output**



### 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:

INVOKE Str\_ucase, ADDR myString

TITLE Upper Case Conversion (Ucase.asm)

; Testing the Str\_ucase procedure.

INCLUDE Irvine32.inc

.data

string\_1 BYTE "abcdef",0

string\_2 BYTE "aB234cdEfg",0

.code

main PROC

call Clrscr

INVOKE Str\_ucase,ADDR string\_1

INVOKE Str\_ucase,ADDR string\_2

mov edx, OFFSET string\_1

call writestring

call crlf

mov edx, OFFSET string\_2

call writestring

call crlf

exit

main ENDP

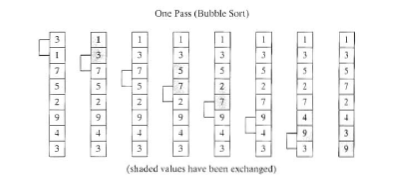
END main

**Output**



## Searching and Sorting Integer Arrays

### Bubble Sort



;----------------------------------------------------------

BubbleSort PROC USES eax ecx esi,

pArray:PTR DWORD, ; pointer to array

Count:DWORD ; array size

;

; Sort an array of 32-bit signed integers in ascending order

; using the bubble sort algorithm.

; Receives: pointer to array, array size

; Returns: nothing

;-----------------------------------------------------------

mov ecx,Count

dec ecx ; decrement count by 1

L1: push ecx ; save outer loop count

mov esi,pArray ; point to first value

L2: mov eax,[esi] ; get array value

cmp [esi+4],eax ; compare a pair of values

jge L3 ; if [esi] >= [edi], don't exch

xchg eax,[esi+4] ; exchange the pair

mov [esi],eax

L3: add esi,4 ; move both pointers forward

loop L2 ; inner loop

pop ecx ; retrieve outer loop count

loop L1 ; else repeat outer loop

L4: ret

BubbleSort ENDP

### Binary Search

;-------------------------------------------------------------

BinarySearch PROC uses ebx edx esi edi,

pArray:PTR DWORD, ; pointer to array

Count:DWORD, ; array size

searchVal:DWORD ; search value

LOCAL first:DWORD, ; first position

last:DWORD, ; last position

mid:DWORD ; midpoint

;

; Search an array of signed integers for single value.

; Receives: Pointer to array, array size, search value.

; Returns: If a match is found, EAX = the array position of the

; matching element; otherwise, EAX = -1.

;-------------------------------------------------------------

mov first,0 ; first = 0

mov eax,Count ; last = (count - 1)

dec eax

mov last,eax

mov edi,searchVal ; EDI = searchVal

mov ebx,pArray ; EBX points to the array

L1: ; while first <= last

mov eax,first

cmp eax,last

jg L5 ; exit search

; mid = (last + first) / 2

mov eax,last

add eax,first

shr eax,1 ;divide by 21

mov mid,eax

; EDX = values[mid]

mov esi,mid

shl esi,2 ; scale mid value by 4

mov edx,[ebx+esi] ; EDX = values[mid]

; if ( EDX < searchval(EDI) )

; first = mid + 1;

cmp edx,edi

jge L2

mov eax,mid ; first = mid + 1

inc eax

mov first,eax

jmp L4

; else if( EDX > searchVal(EDI) )

; last = mid - 1;

L2: cmp edx,edi ; (could be removed)

jle L3

mov eax,mid ; last = mid - 1

dec eax

mov last,eax

jmp L4

; else return mid

L3: mov eax,mid ; value found

jmp L9 ; return (mid)

L4: jmp L1 ; continue the loop

L5: mov eax,-1 ; search failed

L9: ret

BinarySearch ENDP