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_length Procedure
    - Str_copy Procedure
    - Str_trim Procedure
    - Str_ucase Procedure
- Searching and Sorting Integer Arrays
    - Bubble Sort
    - Binary Search

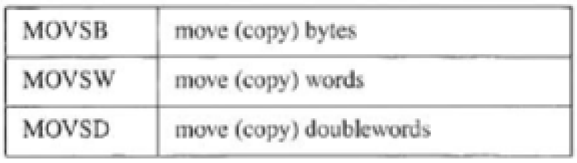
String Primitive Instructions

MOVSB, MOVSW, and MOVSD

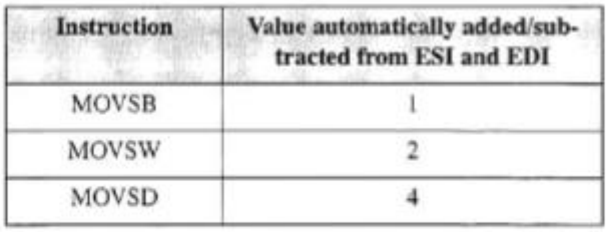
The MOVSB, MOVSW, and MOVSD instructions copy data from the memory location pointed to by ESI to the memory location pointed to by EDI. The two registers are either incremented or decremented automatically (based on the value of the Direction flag):

The **Direction Flag** is used to influence the direction in which some of the instructions work when used with the REP prefix. There is a number of instructions that are influenced by this flag directly, for example MOVS, LODS, SCAS, and many others.

CLD clears the direction flag. No other flags or registers are affected. After **CLD** is executed, string operations will increment the index...



MOVSB	move (copy) bytes
MOVSW	move (copy) words
MOVSD	move (copy) doublewords



Instruction	Value automatically added/subtracted from ESI and EDI
MOVSB	1
MOVSW	2
MOVSD	4

```
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
```

Repeat String Operation (rep, repnz, repz)

```
rep;
repnz;
repz;
```

Operation

repeat string-operation until tested-condition

REP executes the instruction, decreases CX by 1, and checks whether CX is zero. It repeats the instruction processing until CX is zero.

Description

Use the rep (repeat while equal), repnz (repeat while nonzero) or repz (repeat while zero) prefixes in conjunction with string operations. Each prefix causes the associated string instruction to repeat until the count register (CX) or the zero flag (ZF) matches a tested condition.

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:

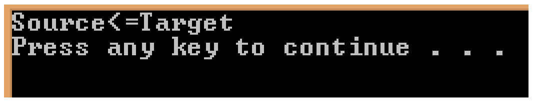
CMPSB	compare bytes
CMPSW	compare words
CMPSD	compare doublewords

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



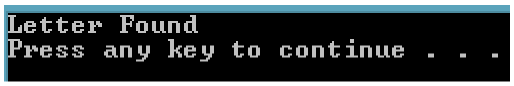
```
Source<=Target
Press any key to continue . . .
```

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



```
Letter Found
Press any key to continue . . .
```

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
```

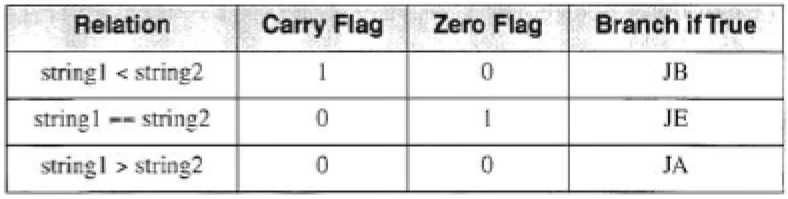
String Procedures

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):



Relation	Carry Flag	Zero Flag	Branch if True
string1 < string2	1	0	JB
string1 == string2	0	1	JE
string1 > string2	0	0	JA

```
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
```

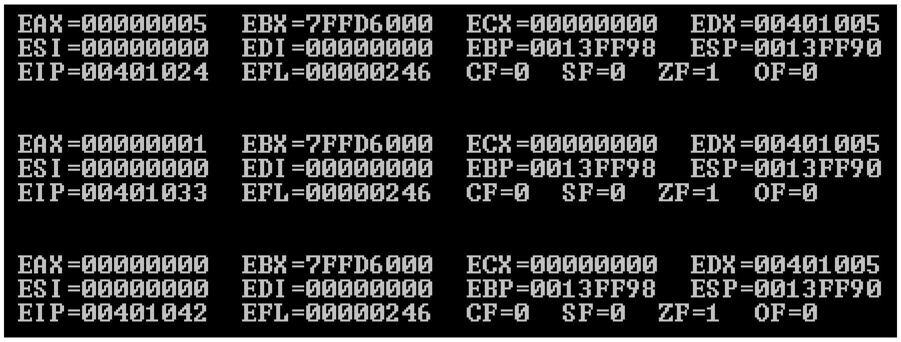
Str_length Procedure

The Str_length 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



```
EAX=00000005  EBX=7FFD6000  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0013FF98  ESP=0013FF90
EIP=00401024  EFL=00000246  CF=0  SF=0  ZF=1  OF=0


EAX=00000001  EBX=7FFD6000  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0013FF98  ESP=0013FF90
EIP=00401033  EFL=00000246  CF=0  SF=0  ZF=1  OF=0


EAX=00000000  EBX=7FFD6000  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0013FF98  ESP=0013FF90
EIP=00401042  EFL=00000246  CF=0  SF=0  ZF=1  OF=0
```

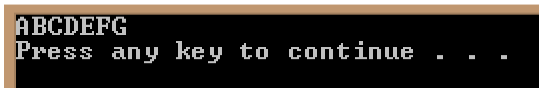
Str_copy Procedure

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



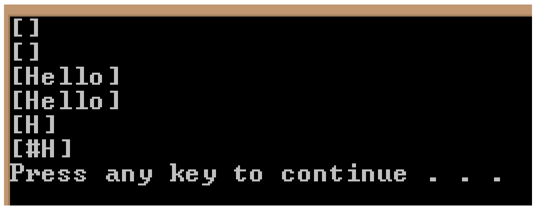
```
ABCDEFG
Press any key to continue . . .
```

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



```
[]
[]
[Hello]
[Hello]
[H]
[#H]
Press any key to continue . . .
```

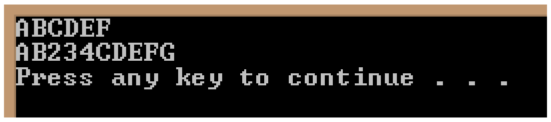
Str_ucase Procedure

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

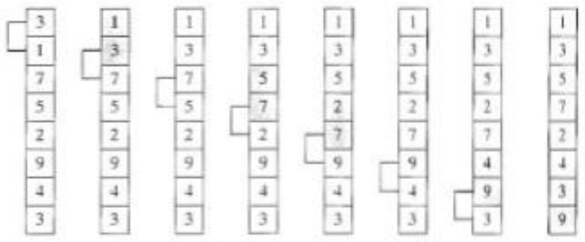


```
ABCDEF
AB234CDEFG
Press any key to continue . . .
```

Searching and Sorting Integer Arrays

Bubble Sort

One Pass (Bubble Sort)



(shaded values have been exchanged)

```
;---------------------------------------------------------
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
```

Lab Tasks

1. Write an assembly program to Swap the values of two string variables without using third variable?
2. Write an assembly program to combine the two string variables by removing the un-necessary characters from them.

Str1 byte	“**aabbccdd** ”,0
Str2 byte	“**eeffgghh$$$$@@@@”,**0
Output:	“**aabbccddeeffgghh**”

3. Using question#2 to do the following;
   - Conver the both strings into uppercase
   - Concatenate both of them into one string i.e. “**AABBCCDDEEFFGGHH**”
4. Ask the user to enter an integer number from console, your task is to find the following;
   - Count the total number of digits in an integer variable
   - Calculate the sum of all the digits in an integer variable

Good luck ☺