Seminar IV. String instructions. Complex string problems.

The string instructions have all default operands and they work in the following pattern: they do something with the current element of the string(s) and they move to the next element in the string(s). In order to work with string instructions, we must initially:

- set the offset of the source string in ESI (the source string is the one we do not modify)
- set the offset of the destination string in EDI (the destination string is the one we modify)
- set the parsing direction (rom. directia de parcurgere) of strings; if the Direction Flag DF=0 strings are parsed from left to right and if DF=1 strings are parsed from right to left

Some string instructions work only with the source string, some others work only with the destination string and some others work with both.

String instructions for data transfer

(Load String of Bytes)
1. LODSB AL← <DS:ESI>
if DF=0 inc(ESI) else dec(ESI)

(Load String of Words)
2. LODSW AX← <DS:ESI>
if DF=0 ESI←ESI+2 else ESI←ESI-2

(Store String of Bytes)
3. STOSB <ES:EDI>← AL
if DF=0 inc(EDI) else dec(EDI)

(Store String of Words)
4. STOSW <ES:EDI>← AX
if DF=0 EDI←EDI+2 else EDI←EDI-2

(Move String of Bytes)
5. MOVSB <ES:EDI>← <DS:ESI>
if DF=0 {inc(ESI); inc(EDI)} else {dec(ESI); dec(EDI)}

(Move String of Words)
6. MOVSW <ES:EDI>← <DS:ESI>
if DF=0 {ESI←ESI+2; EDI←EDI+2}
else {ESI←ESI-2; EDI←EDI-2}

String instructions for data comparisons

(Scan String of Bytes)
7. SCASB CMP AL, <ES:EDI>
if DF=0 inc(EDI) else dec(EDI)

(Scan String of Words)
8. SCASW CMP AX, <ES:EDI>
if DF=0 EDI←EDI+2 else EDI←EDI-2

(Compare String of Bytes)
9. CMPSB CMP <DS:ESI>, <ES:EDI>
if DF=0 {inc(ESI); inc(EDI)} else {dec(ESI); dec(EDI)}

(Compare String of Words)
10. CMPSW CMP <DS:ESI>, <ES:EDI>
if DF=0 {ESI←ESI+2; EDI←EDI+2}
else {ESI←ESI-2; EDI←EDI-2}

There also exist instructions LODSD, STOSD, MOVSD, SCASD, CMPSD that work with strings of doublewords, use EAX and always increment/decrement ESI and EDI by 4 bytes.

We solve the last problem from the previous seminar using string instructions.

Ex.1. Being given a string of bytes containing lowercase letters, build a new string of bytes containing the corresponding uppercase letters.

```
bits 32
global start
extern exit
import exit msvcrt.dll

segment data use32 class=data
    s1 db 'abcdef'
    lenS1 equ $-s1                 ; defines the length in bytes of string “s1”, i.e. 6
    s2 times lenS1 db 0            ; reserve lenS1 bytes for string “s2”

segment code use32 class=code
start:
    mov esi, s1             ; set the offset of the source string s1 in ESI
    mov edi, s2             ; set the offset of the destination string s2 in EDI
    mov ecx, lenS1          ; we will use a loop/cycle with lenS1 iterations
    cld

    repeat:
        lodsb           ; mov al, [esi]     +    inc esi
        sub al, 'a' -'A'
        stosb           ; mov [edi], al     +    inc edi

        loop repeat     ; is equivalent to these 3 instructions:
```

```
                                  ;      dec ecx
                                  ;      cmp ecx, 0
                                  ;      ja repeat

   push dword 0
   call [exit]
```

Ex.2. Being given a string of bytes, write a program that obtains the mirrored string of bytes.

Example: Being given the string of bytes:
s   db   17, 20, 42h, 1, 10, 2ah
the corresponding mirrored string of bytes will be
t   db   2ah, 10, 1, 42h, 20, 17.

In order to solve the problem, we will parse the initial string “s” in a loop and copy each byte in string “t”. While string “s’ will be parsed from left to right (i.e. DF=0), string “t” will be parsed from right to left (i.e. DF=1). Thus, the first byte of string “s” will be copied in the last byte of string “t”, the second byte of string “s” will be copied in the last but one byte of string “t” and so on..

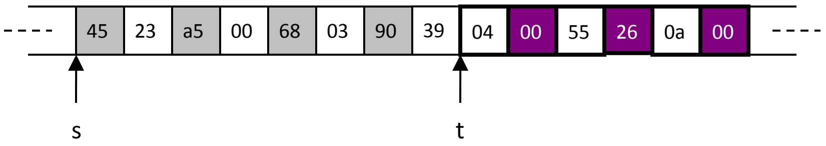
```
bits 32
global start
extern exit
import exit msvcrt.dll

segment data use32 class=data
        s   db   17, 20, 42h, 1, 10, 2ah
        len_s   equ   $-s
        t   times  len_s  db 0

segment code use32 class=code
start:
        mov esi, s          ; set the starting offset of the source string “s” in ESI
                            ; ESI now contains the offset of the first byte in string “s”
        mov edi, t          ; set the starting offset of the destination string “t” in EDI
                            ; but because string “t” needs to be parsed from right to left, the starting offset
                            ; of string “t” should be the offset of the last byte in string “t” (i.e. EDI=t + len_s - 1)
        add edi, len_s-1
        mov ecx, len_s
        jecxz theend        ; if ECX==0 jump to “theend”
repeat:
        cld                 ; DF=0 (parse strings from left to right)
        lodsb               ; mov al, [esi]     +    inc esi

        std                 ; DF=0 (parse strings from right to left)
        stosb               ; mov [edi], al     +    dec edi
        loop repeat     ;

theend:
        push dword 0
        call [exit]
```

Ex.3. Two strings of words are given. Concatenate the string of low bytes of the words from the first string to the string of high bytes of the words from the second string. The resulted string of bytes should be sorted in ascending order in the signed interpretation.

<u>Example:</u> Having the strings of words:

```
s   dw   2345h, 0a5h, 368h, 3990h
t   dw   4h, 2655h, 10
```

these strings will be represented in the memory in little-endian format as (the colored bytes are the ones required by the text of the problem):



The result string should be:

u: 90h, a5h, 0h, 0h, 26h, 45h, 68h

```
bits 32
global start
extern exit
import exit msvcrt.dll

segment data use32 class=data
        s   dw   2345h, 0a5h, 368h, 3990h
        len_s   equ   ($-s)/2              ; the length (in words) of string “s”
        t   dw   4h, 2655h, 10
        len_t   equ   ($-t)/2              ; the length (in words) of string “t”
        len   equ   len_s+len_t            ; the length of the result string

        u   times len  db  0               ; the result string

segment code use32 class=code
start:
        ; first we copy the low bytes of the words from string “s” into the resulted string “u”

        mov esi, s      ; set the offset of the source string (i.e. the offset of the 1st byte from string “s”)
        mov edi, u      ; set the offset of the dest string (i.e. the offset of the 1st byte from string “u”)

        cld             ; DF=0

        mov ecx, len_s          ; use a loop with len_s iterations
        jecxz theend
        repeat:
                lodsw           ; mov ax, [esi]     +    esi:=esi+2
                                ; AL will store the low byte of the current word from string “s”
                                ; AH will store the high byte of the current word from string “s”

                stosb           ; mov [edi], al     +    edi:=edi+1
                                ; we only need to copy the low byte (i.e. AL) into the “u” string
                loop repeat

        ; next, we need to copy the high bytes of the words from string “s” into the string “u”
        mov esi, t              ; set the offset of the source string “t”
        mov ecx, len_t          ; use a loop with len_t iterations
        jecxz theend
                        .
        repeta1:
                lodsw           ; mov ax, [esi]     +    esi:=esi+2
                                ; AL will store the low byte of the current word from string “s”
                                ; AH will store the high byte of the current word from string “s”

                xchg al, ah     ; interchange AL with AH
                                ; we need to put the high byte in AL in order to use stosb below
                stosb           ; mov [edi], al     +    edi:=edi+1

        loop repeta1    ;the loop block could have also been written like this:
                        ; repeta1:
                                ; lodsb
                                ; lodsb
                                ; stosb
                                ; loop repeta1

        ; We now begin the second part of the program, that is sorting the string “u” in ascending order (in
        ; the signed interpretation). In order to perform the sorting, we use a variant of bubble sort
        ; algorithm which is depicted below :
        ;       // u is a vector of length “len”
        ;       changed = 1;
        ;       while (changed = =1) {
        ;               changed = 0;
        ;               for (i=1; i<=len-1; i++) {
        ;                       if (u[i+1]<u[i]) {
        ;                               aux = u[i];
        ;                               u[i] = u[i+1];
```

```
        ;                               u[i+1] = aux;
        ;                               changed = 1;
        ;                       }
        ;               }
        ;       }

        mov dx, 1               ; the equivalent of “changed=1” from the algorithm.

repeat2:
        cmp dx, 0
        je theend               ; if DX=0 then it means that there was no change is the last parse of the
                                ; string, so we exit the loop because the string is sorted ascending
        mov esi, u              ; prepare the parsing of string “u”; set the starting offset in ESI
        mov dx, 0               ; initialize DX
        mov ecx, len-1          ; parse string “u” in a loop with len-1 iterations (the equivalent of the “for”-
                                ; loop from the above algorithm).

        repeat3:
                mov al, byte [esi]      ; al = u[i].
                cmp al, byte [esi+1]    ; compare al=u[i] cu u[i+1]
                jle next                ; if u[i]<=u[i+1] move to the next iteration (i++). Otherwise
                                         ; interchange u[i]  (byte [esi]) with u[i+1] (byte [esi+1]) in the
                                         ; following 3 instructions. We used the “jle” instruction because
                                         ; we want to do signed comparison.
                mov ah, byte [esi+1]
                mov byte [esi], ah
                mov byte [esi+1], al
                mov dx, 1                ; set DX to 1 in order to signalize that an interchange happened

                next:
                        inc esi          ;we move to the next byte in the “u” string (equivalent to “ i++”).
                        loop repeat3     ; resume repeat3 if we did not reach the end of string “u”
                        jmp repeat2      ; otherwise resume the repeat2 cycle.

theend:
        push dword 0
        call [exit]
```