Seminar IV ASC

Instrucțiuni pentru șiruri. Probleme „complexe” pe șiruri

Instrucțiunile pentru șiruri au operanzi impliciți și sunt folosite in felul următor: fac ceva cu elementul curent al șirului (șirurilor) și se mută la următorul element din șir (șiruri). Pentru a folosi aceste instrucțiuni, inițial trebuie:

- copiat offset-ul șirului sursă în registrul ESI (șirul sursă nu va fi modificat)
- copiat offset-ul șirului destinație în registrul EDI (șirul destinație este cel modificat)
- configurată direcția de parcurgere a șirului: dacă Direction Flag (DF) are valoarea 0 șirurile sunt parcurse de la stânga la dreapta, dacă DF = 1 șirurile sunt parcurse de la dreapta la stânga

Un set de instrucțiuni pentru șiruri folosește doar șirul sursă, alt set folosește doar șirul destinație iar alt set folosește atât șirul sursă cât și șirul destinație.

4.1 Instrucțiuni pentru transferul datelor

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

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

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

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

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

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

4.2 Instrucțiuni pentru compararea datelor

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

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

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

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

Există și instrucțiunile LODSD, STOSD, MOVSD, SCASD, CMPSD care consideră elementele șirului ca fiind dublucuvinte, folosesc registrul EAX și incrementează/decrementează ESI și EDI cu patru octeți.

4.3 Exemple

Se rezolvă problema din seminarul 3 folosind instrucțiuni care manipulează șiruri.
Ex. 1. Se dă un șir de octeți care conține litere mici, să se construiască un șir nou de octeți care să conțină literele din șirul inițial transformate în majuscule.

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

segment data use32 class=data
    s1 db 'abcdef'
    lenS1 equ $-s1       ; lungimea in octeti a sirului s1, 6 octeti
    s2 times lenS1 db 0 ; se rezerva lenS1 octeti pentru sirul s2

segment code use32 class=code
start:
    mov esi, s1     ; se copiaza offset-ul sirului s1 in ESI
    mov edi, s2     ; se copiaza offset-ul sirului ss in EDI
    mov ecx, lenS1  ; se repeta urmatoarele instructiuni de lenS1 ori
    cld

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

        loop repeat
    ; instructiunea loop este echivalenta cu urmatoarele 3 instructiuni:
        ; dec ecx
        ; cmp ecx, 0
        ; ja repeat

    push dword 0
    call [exit]
```

Ex. 2. Se dă un șir de octeți. Sa se obțină șirul oglindit.

Exemplu: Fie următorul șir de octeți

```
s    db    17, 20, 42h, 1, 10, 2ah
```

șirul oglindit este
```
t    db    2ah, 10, 1, 42h, 20, 17.
```

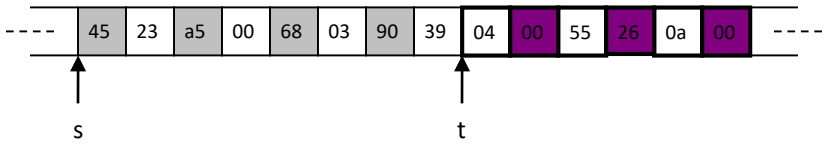
Pentru a rezolva problema se parcurge șirul inițial „s” într-o buclă și se copiază fiecare octet în șirul „t”. Șirul „s” este parcurs de la stânga la dreapta (DF = 0) iar șirul „t” de la dreapta la stânga (DF = 1). Astfel primul octet din șirul „s” se va copia în ultimul octet din șirul „t”, etc.

```
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  ; se copiaza offset-ul s in ESI
            ; ESI contine offset-ul primului octet din sirul s
mov edi, t  ; se copiaza offsetul sirului t in EDI
            ; deoarece sirul t trebuie parcurs de la dreapta la stanga
            ; offset-ul din EDI trebuie sa fie cel al ultimului octet
            ; din sirul t (EDI = t + len_s - 1)
add edi, len_s-1
mov ecx, len_s
jecxz theend    ; daca ECX==0 salt la “theend”
repeat:
    cld         ; DF=0 (se parcurge sirul de la stanga la dreapta)
    lodsb       ; mov al, [esi] + inc esi

    std         ; DF=0 (se parcurge sirul de la dreapta la stanga)
    stosb       ; mov [edi], al + dec edi

    loop repeat ;

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

Ex.3. Se dau două șiruri de cuvinte, să se concateneze șirul octeților inferiori cuvintelor primului șir cu șirul octeților superiori cuvintelor din cel de-al doilea șir. Șirul rezultat trebuie sortat crescător în interpretare cu semn.

Exemplu: fie cele două șiruri de cuvinte:
```
s    dw    2345h, 0a5h, 368h, 3990h
t    dw    4h, 2655h, 10
```

aceste șiruri vor fi reprezentate în memorie în format little-endian în felul următor (octeții colorați sunt cei din cerința problemei)



Șirul rezultat este:

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    ($-s)/2   ; lungimea (în cuvinte) a șirului "s"
    t    dw    4h, 2655h, 10
    len_t    equ    ($-t)/2   ; lungimea (în cuvinte) a șirului "t"
    len    equ    len_s+len_t ; lungimea șirului rezultat
    u    times len   db   0    ; șirul rezultat

segment code use32 class=code
start:
    ; se copiază octeții inferiori cuvintelor din s in șirul u
    mov esi, s  ; offset pentru șirul sursă (offset-ul primului
                ; octet din șirul s)
    mov edi, u  ; offset pentru șirul destinație (offset-ul primului
                ; octet din șirul u)
    cld          ; DF=0

    mov ecx, len_s  ; o buclă cu len_s iterații
    jecxz theend
repeat:
    lodsw   ; mov ax, [esi] + esi:=esi+2
    ; AL va stoca octetul inferior al cuvântul curent din s
    ; AH va stoca octetul superior al cuvântului curent din s

    stosb   ; mov [edi], al + edi:=edi+1
    ; trebuie copiat doar octetul inferior (AL) în șirul destinație
    loop repeat

    ; se copiază octeții superiori cuvintelor din t în șirul u
    mov esi, t   ; offset-ul șirului sursă t
    mov ecx, len_t  ; o buclă cu len_t iterații
    jecxz theend
```

```
repeta1:
    lodsw   ; mov ax, [esi] + esi:=esi+2
    ; AL va stoca octetul inferior al cuvântul curent din t
    ; AH va stoca octetul superior al cuvântului curent din t

    xchg al, ah  ; se interschimbă valorile din AL și AH
                 ; interesează octetul superior cuvintelor din t
    stosb        ; mov [edi], al + edi:=edi+1
    loop repeta1; această buclă se mai poate scrie astfel:
            ; repeta1:
                ; lodsb
                ; lodsb
                ; stosb
                ; loop repeta1

    ; se sortează șirul u crescător în interpretare cu semn, se va
    ; implementa o variantă a bubble sort descrisă mai jos
;   // u este un vector de lungime „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    ; changed=1
repeat2:
    cmp dx, 0
    je theend
    mov esi, u  ; offset u in ESI
    mov dx, 0   ; inițializare DX
    mov ecx, len-1     ; șirul u este prelucrat într-o bucla în
                       ; len-1 iterații
    repeat3:
        mov al, byte [esi]        ; al = u[i].
        cmp al, byte [esi+1]      ; comparare al=u[i] cu u[i+1]
        jle next       ; dacă u[i]<=u[i+1] -> i++, altfel
                       ; interschimba u[i] (byte [esi]) cu u[i+1]
                       ; (byte [esi+1])
        mov ah, byte [esi+1]
        mov byte [esi], ah
        mov byte [esi+1], al
        mov dx, 1      ; changed = 1
```

```
next:
inc esi      ; i++
loop repeat3
jmp repeat2

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

Ex. 4: Se da un sir de numere intregi reprezentate pe dublucuvinte. Sa se construiasca sirul corespunzator octetilor strict negativi din reprezentarea in memorie a dublucuvintelor

Data segment	Code segment

S1 dd 127, 3500, -125, -670
L equ $-S1
D times L db 0
	CLD
	MOV ESI, S1
	MOV EDI, D
	MOV ECX, L
	
	JECXZ final_loop
	start_loop:
	LODSB
	CMP AL, 0
	JGE skip_store
	stosb
	skip_store:
	LOOP start_loop
	final_loop:

Ex. 5 : Se da un sir de dublucuvinte. Sa se calculeze suma octetilor high ai cuvintelor low multiplii de 10.

Data segment	Code segment

S1 dd 127, 3500, -125, -670
L equ ($-S1)/4
Suma db 0
	Mov ECX,L
	Mov ESI,S1
	CLD
	JECXZ Final
	Repeta:
	lodsb
	Lodsb
	CBW
	MOV BX,AX
	Mov DL,10
	IDIV DL
	CMP AH,0

	JNE nu_multiplu Add [Suma],BL nu_multiplu: lodsw LOOP Repeta Final:

Ex. 6

```
a dd 11223344h, 55667788h

MOV ESI, a
MOV EDI, a+2
LODSB
MOV AH, AL
LODSB
STOSW
INC ESI;
LODSW
STOSB
```

Cum arata memory layout-ul si care sunt registrii implicati la fiecare comanda si ce valoare au?

Rezolvare:

a dd 11223344h, 55667788h

44h|33h|22h|11h|88h|77h|66h|55h

MOV ESI, a

MOV EDI, a+2

LODSB; AL=44h; ESI=a+1

MOV AH, AL; AH=44h

LODSB; AL=33h; ESI=a+2

STOSW; se pune in memorie la adresa a+2 valoarea 4433h si EDI=a+4

44h|33h|33h|44h|88h|77h|66h|55h

INC ESI; ESI=a+3

LODSW; AX=8844h; ESI=a+5

STOSB; se pune in memorie la adresa a+4 valoarea 44h si EDI=a+5

44h|33h|33h|44h|44h|77h|66h|55h

Ex. 7

```
a dd 12345678h,87654321h,11223344h, 44332211h
MOV ESI,a
MOV EDI,a

LODSB
xchg AL,AH
LODSB
MOVSD
STOSW
LODSB
MOVSW
```

Rezolvare:

a dd 12345678h,87654321h,11223344h, 44332211h

MOV ESI,a

MOV EDI,a

LODSB; AL=78h; ESI=a+1

xchg AL,AH; AH=78h

LODSB; AL=56h; ESI=a+2 => AX=7856h

MOVSD; 34h|12h|21h|43h|21h|43h|65h|87h|...; ESI=a+6, EDI=a+4

STOSW; 34h|12h|21h|43h|56h|78h|65h|87h|...; EDI=a+6

LODSB; AL=65h; ESI=a+7

MOVSW; 34h|12h|21h|43h|56h|78h|87h|44h|44h|33h|22h|11h|11h|22h|33h|44h

init: 78h|56h|34h|12h|21h|43h|65h|87h|44h|33h|22h|11h|11h|22h|33h|44h