

Ministerul Educației, Culturii și Cercetării Republicii Moldova  
Universitatea Tehnică a Moldovei  
Facultatea Calculatoare Informatica și Microelectronică

# RAPORT

Laboratorul nr.5

la disciplina Arhitectura Calculatoarelor

**Tema:** Prelucrarea șirurilor. Utilizarea tabelelor pentru conversii de coduri

A realizat:

st.gr Popescu Victoria TI-173

A verificat:

Colesnic V.

Chișinău 2019

## Scopul lucrării:

Se prezinta probleme legate de prelucrarea in limbaj de asamblare a şirurilor. Pentru aceasta se recomanda utilizarea instrucţiunilor speciale pentru tratarea şirurilor. Se prezinta de asemenea tehnici specifice limbajului de asamblare pentru realizarea conversiilor de coduri, bazate pe utilizarea tabelelor de conversie.

## Indicaţii teoretice:

### Tratarea şirurilor

În afară de tipurile de bază, există şi posibilitatea efectuării unor operaţii de transfer, sau operaţii aritmetice şi logice cu şiruri de date (cu informaţii aflate în zone continue de memorie). Operaţiile pe şiruri pot fi efectuate individual, pentru fiecare cuvânt din şir, sau automat - cu repetare, numărul de repetări al instrucţiunii fiind dictat de conţinutul unui registru contor.

Operaţiile tipic efectuate sunt:

- transferul unui şir din zonă sursă în zonă destinaţie;
- comparare între două şiruri;
- căutarea unei valori într-un şir;
- încărcarea acumulatorului cu elementele unui şir;
- citirea unui şir de la un port de intrare;
- scrierea unui şir la un port de ieşire.

Exemple :

### Instrucţiunile MOVSB (Move (copy) bytes)

#### MOVSW (Move (copy) words)

#### MOVSD (Move (copy) doublewords)

Transfer pe 8 (16,32) biţi, din zona de memorie indicată de ESI, în zona de memorie indicată de registrul EDI. După transferul primului byte (word, doubleword), dacă flag-ul DF=0, se petrece autoincrementarea  $ESI \leftarrow ESI + 1$ ;  $EDI \leftarrow EDI + 1$  (decrementare pentru DF=1).

În operaţii cu şiruri sunt utilizate prefixe de repetare:

REP	Repetare până $ecx > 0$
REPZ, REPE	Repetare până $ZF = 1$ şi $ecx > 0$
REPNZ, REPNE	Repetare până $ZF = 0$ şi $ecx > 0$

Exemplu. Fie dat să copiem 20 de cuvinte duble din şirul sursă **source** în şirul destinaţie **target**:

```
.data
source DWORD 20 DUP(0FFFFFFFFh)
target DWORD 20 DUP(?)
.code
cld ; direction = forward
mov ecx,LENGTHOF source ; setam contorul REP
```

```

mov esi,OFFSET source ; incarcam ESI cu adresa sourcei
mov edi,OFFSET target ; incarcam EDI cu adresa destinației
rep movsd ;copiem cuvinte duble

```

### **Instrucțiunile CMPSB (Compare bytes)**

### **CMPSW (Compare words)**

### **CMPSD (Compare doublewords)**

Comparare pe 8 (16,32) biți, din zona de memorie indicată de ESI, cu zona de memorie indicată de registrul EDI. După compararea primului byte (word, doubleword), dacă flag-ul DF=0, se petrece autoincrementarea  $ESI \leftarrow ESI+1$ ;  $EDI \leftarrow EDI+1$  (decrementare pentru DF=1).

Exemple:

```

.data
source DWORD 1234h
target DWORD 5678h
.code
mov esi,OFFSET source
mov edi,OFFSET target
cmpsd ; compare doublewords
ja L1 ; jump if source > target

```

Dacă comparăm cuvinte multiple:

```

mov esi,OFFSET source
mov edi,OFFSET target
cld ; direction = forward
mov ecx,LENGTHOF source ; repetition counter
repe cmpsd ; repeat while equal

```

Prefixul REPE repetă compararea, incrementând ESI și EDI în mod automat, până când ECX=0 sau o pereche de cuvinte duble nu va fi egală.

### **Instrucțiunile SCASB (SCAS- Scans a string)**

### **SCASW**

### **SCASD**

Instrucțiunile compară valoarea din AL/AX/EAX cu byte, word sau doubleword din zona de memorie indicată de EDI. Instrucțiunile sunt utile la căutarea unui singur element într-un șir.

Exemple:

```

.data
alpha BYTE "ABCDEFGH",0
.code
mov edi,OFFSET alpha ; incarcam EDI cu adresa
; sirului de scanat
mov al,'F' ; cautam litera F
mov ecx,LENGTHOF alpha ; setam registrul contor

```

```

cld ; direction = forward
repne scasb ; repetam pana nu este egal
jnz quit ; iesire daca litera nu a fost gasita

```

### **Instrucțiunile STOSB (STOS- Store string data)**

**STOSW**

**STOSD**

Instrucțiunile încarcă valoarea din AL/AX/EAX , în memorie cu offset-ul indicat de EDI. Incrementarea se petrece conform flag-ului DF (DF=0- incrementarea, DF=1- decrementarea).

Exemplu. Șirul string1 este completat cu valoarea 0FFh.

```

.data
Count = 100
string1 BYTE Count DUP(?)
.code
mov al,0FFh ; valoarea de de incarcat
mov edi,OFFSET string1 ; EDI cu adresa sirului
mov ecx,Count ; numarul de elemente ale sirului
cld ; direction = forward
rep stosb ; copierea AL in string1

```

### **Instrucțiunile LODSB (LODS- Load Accumulator from String)**

**LODSW**

**LODSD**

Instrucțiunile încarcă valoarea din byte, word sau doubleword din memorie idicat de ESI, în AL/AX/EAX respectiv. Instrucțiunile sunt utile la căutarea unui singur element într-un șir.

Exemplu: Multiplicarea fiecărui element a unui șir cu o constantă.

```

INCLUDE Irvine32.inc
.data
array DWORD 1,2,3,4,5,6,7,8,9,10 ; test data
multiplier DWORD 10
.code
main PROC
cld ; direction = forward
mov esi,OFFSET array ; sirul sursa
mov edi,esi ; sirul destinatie
mov ecx,LENGTHOF array ; setarea contorului
L1: lodsd ; incarcarea [ESI] in EAX
mul multiplier ; multiplicarea cu constanta
stosd ; copie din EAX in [EDI]
loop L1
exit
main ENDP
END main

```

## Mersul lucrării:

Exemplul 1. {Mutarea unui bloc de memorie de la o adresa sursa la o adresa destinație} Vom prezenta o prima variantă a acestei probleme care nu utilizează instrucțiuni pentru șiruri.

```
.DATA
sir1    DB  100 DUP(7)
sir2    DB  100 DUP(?)

.CODE
    mov esi,OFFSET sir1
    mov edi,OFFSET sir2
    mov ecx,LENGTHOF sir1
muta:   mov al,[esi]
        mov [edi],al
        inc esi
        inc edi
        loop muta
```

Varianta care utilizează instrucțiunea **movsb** este urmatoarea:

```
.DATA
sir1    byte  100 DUP(7)
sir2    byte  100 DUP(?)

.CODE
    mov esi,OFFSET sir1
    mov edi,OFFSET sir2
    mov cx,LENGTHOF sir1
    cld
muta:   movsb sir1,sir2
        loop muta
```

Exemplul 2. {Compararea a doua siruri de octeți} Varianta fără prefix **rep** este:

```
.DATA
sir1    byte    'AAAABC'
sir2    byte    'AAAACB'

.CODE
    mov     esi,OFFSET sir1
    mov     edi,OFFSET sir2
    mov     ecx,LENGTHOF sir1
comp:    cmpsb    sir1,sir2
        jne     exit
        loop    comp
exit:    nop
```

## Sarcina individuală:

6. Sa se scrie un *macro* care primeste doua adrese de memorie **A<sub>1</sub>** si **A<sub>2</sub>** si un caracter **CAR** si elimina caracterul **CAR** din sirul de caractere incepind de la adresa **A<sub>1</sub>**, depunind rezultatul incepind de la adresa **A<sub>2</sub>**. Afişaţi pe ecran ambele şiruri.

```
INCLUDE Irvine32.inc
```

```
cautare MACRO s1,s2
```

```
mov ebx,OFFSET s2
```

```
mov edi,0
```

```
mov esi,OFFSET s1
```

```
gaseste:
```

```
mov al,len1
```

```
cmp al,0
```

```
je iesire;
```

```
mov al,[esi]
```

```
cmp al,car
```

```
jne inscrie;
```

```
inc esi
```

```
dec len1
```

```
jmp gaseste;
```

```
inscrie:
```

```
mov al,[esi]
```

```
mov byte ptr [ebx][edi],al
```

```
inc edi
```

```
inc esi
```

```
jmp gaseste;
```

```
iesire:
```

```
ENDM
```

```
.data
```

```
sir1    byte 100 DUP(?)
```

```
sir2    DB  100 DUP(?)
```

```
mes2 byte "Introduceti S1 : ",0
```

```
mes1 byte "Introduceti CAR : ",0
```

```
mes3 byte "Introduceti lungimea S1: ",0
```

```
mes4 byte "Sir fara caracterul CAR : ",0
```

```
len1 db ?
```

```
len2 db ?
len3 db ?
max=100
nr dw ?
contor db 1
c1 db 0
pinala db 0
car db ?
l db 0
.code
main proc
mov edx,OFFSET mes3
call WriteString ;
call ReadDec ;
mov len1,al ;

mov edx,OFFSET mes2
call WriteString ;
mov edx,OFFSET sir1
mov ecx,max
call ReadString ;

mov edx,OFFSET mes1
call WriteString

call ReadChar
mov car,al
call WriteChar

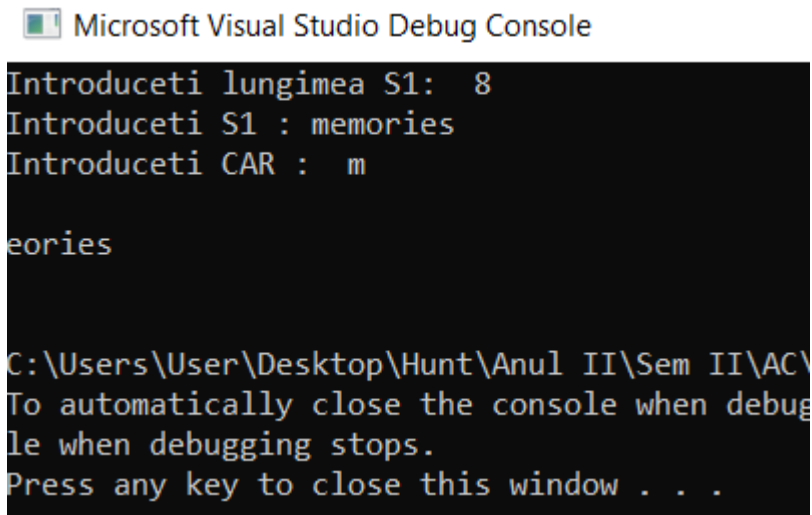
cautare sir1,sir2

call Crlf
call Crlf
mov edx,OFFSET mes4
call WriteString

mov edx,OFFSET sir2
call WriteString
call Crlf
call Crlf

exit

esire:
main ENDP
END main
```



```
Microsoft Visual Studio Debug Console

Introduceti lungimea S1: 8
Introduceti S1 : memories
Introduceti CAR : m

eories

C:\Users\User\Desktop\Hunt\Anul II\Sem II\AC\
To automatically close the console when debug
le when debugging stops.
Press any key to close this window . . .
```

Figura1 – Extragerea din cuvântul ”memories” caracterul ‘m’

## Concluzie:

Scopul propus de această lucrare de laborator este implimentarea șirurilor în limbajul Assembler. Pe lângă aceasta am folosit și tabeluri pentru conversii de coduri.

In cadrul sarcinei principale a servit rulara programelor scrise pe limbajul Assembler în care se utilizează tabelele pentru conversii de coduri. Pentru aceasta am reprezentat tehnici specifice limbajului de asamblare pentru realizarea conversiilor de coduri, bazate pe utilizarea tabelelor de conversie

Datorita acestei lucrari de laborator am înțeles cum se implementează șirurile, astfel și manipularea cu acestea .