Laborator 0x06

11 Nov 2024

1. Recap: Array-uri întregi
2. Șiruri de caractere
   - Parcurgerea unui șir de caractere
3. Implementarea procedurilor în assembly
   - PRINTF
   - SCANF
4. Probleme:
   - Afișarea unui array de întregi
   - Citirea unui array de întregi
   - Media aritmetică a el. dintr-un array citit de la tastatură

Recap Array-uri de întregi

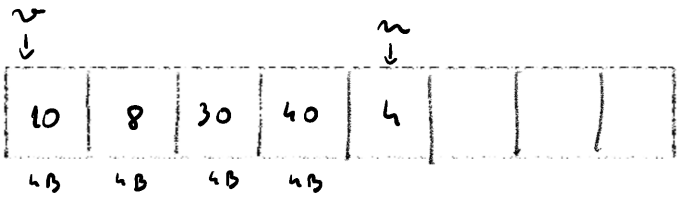
Declarare & inițializare

```
.data
    v: .long 10, 8, 30, 11
    n: .long 4
```

Declarare

```
.data
    v: .space 400    (100 elemente * 4 B = 400 B)
```

În memorie



$10	$v + 0	$v + 0 · 4
$8	$v + 4	$v + 1 · 4
$30	$v + 8	$v + 2 · 4
		...
	v[i]	$v + i · 4

$v → %edi

index → %ecx

⇒ (%edi, %ecx, 4) conține el. de la %edi + %ecx * 4

$ → & referențiere

(·) → * dereferențiere de memorie

Parcurgerea Array-urilor

In C / C++

```
for ( int i = 0;  i < n;  i++ )
  {
        ebx = v[i];
        // prelucrează ebx
  }
```

Assembly

Pp. ca v și n sunt deja declarate și inițializate în secțiunea .data

```
main :
    lea  v, %edi        # mov $v, %edi
    mov  $0, %ecx       # xor %ecx, %ecx
et_loop :
    cmp  n, %ecx
    je   et_exit
    mov  (%edi, %ecx, 4), %ebx
    # proces %ebx
    inc  %ecx           # add  $1, %ecx
    jmp  et_loop
et_exit
    mov  $1, %eax
    mov  $0, %ebx
    int  $0x80
```

□

Șiruri de caractere

- sunt tablouri de date care stochează elem. care ocupă 1 B

Declarare & inițializare

str 1 : .ascii "Șir de caractere"

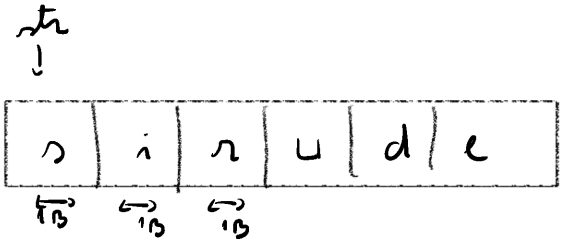
str 2 : .asciz "Șir de caractere"

str 3 : .byte 'ș', 'i', 85, 255, 'ț', 'i', ..., '\0'

Declarare

str : .space 101 (100 citite de la STDIN + '\0')

Accesarea unui element

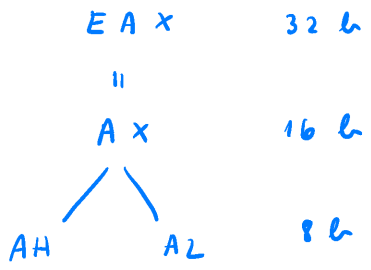


$'ș' = $ str

$'i' = $ str + 1B

str ← %edi

index ← %ecx



movb (%edi, %ecx, 1), %al

1B 1B

Parcurgerea unui șir de caractere

- În C/C++

```
i = 0;
while ( s[i] != 0)
  {
     prelucrez  s[i]
     i++;
  }
```

Considerăm str declarat ca .asciz în secțiunea .data

- Assembly

```
main:
    lea str, %edi      # sau    mov $str, %edi
    xor %ecx, %ecx
et_loop
    movb (%edi, %ecx, 1), %al
    cmpb $0, %al
    je et_exit
    # prelucrare %al
    inc %ecx           # sau    add $1, %ecx
    jmp et_loop
et_exit:
    ...
```

Dacă folosim debuggerul și rulăm

```
b et_exit
run
i r ecx
```

=> lungimea șirului

Implementarea procedurilor în assembly

- apeluri de proceduri din C

```
gcc -m32 file.s -o file -no-pie
```

↳ am automat acces la
- stdlib.h
- string.h

printf scanf fflush

sunt stdlib strlen strcmp strstr strtok

Apelul procedurii PRINTF

%d → întregi

%ld → long

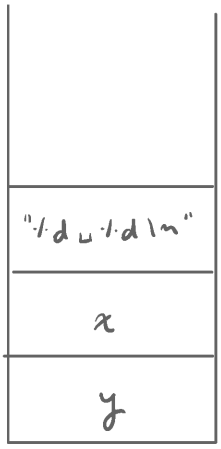
%s → șiruri

Apelare

- apel: instrucțiunea call
- argumentele sunt transmise prin intermediul stivei
- stiva este controlată prin două instrucțiuni
  - push op: pune val op în vf. stivei
  - pop op: elimină vf. stivei dar nu înainte de a salva val. în op

Exemplu

```
.data
    formatPrint: .asciz "%d %d\n"
    x: .long 5
    y: .long 6
.text
.global main:
main:
    push y
    push x
    push $formatPrint
    call printf
    pop %ebx
    pop %ebx
    pop %ebx
et_exit:
    mov $1, %eax
    mov $0, %ebx
    int $0x80
```



Obs - push arg. în stivă în ordine inversă

- nr. push - uri = nr. pop - uri

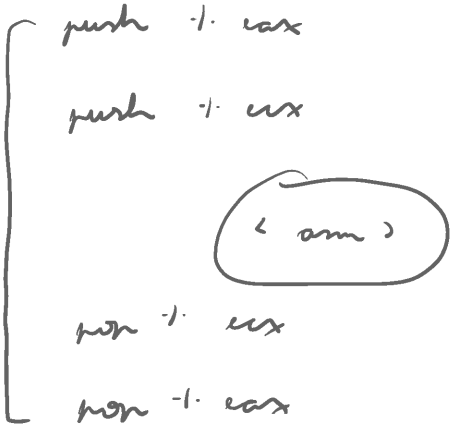
Obs În urma apelului unei proceduri, regiștii %eax, %ecx, %edx NU garantează păstrarea valorii dar %ebp, %esi, %edi, %ebx, %esp garantează păstrarea valorii

Aplicație : afișarea unui array de întregi

```
.data
    v: .long 10, 20, 30, 40
    n: .long 4
    formatPrint: .asciz "%d\n"
.text
.global main
main:
    lea v, %edi
    xor %ecx, %ecx
et_loop:
    cmp n, %ecx
    je et_exit

    movl (%edi, %ecx, 4), %ebx

    pushl %ecx
    pushl %ebx
    push $formatPrint
    call printf
    pop %ebx
    pop %ebx
    pop %ecx

    inc %ecx
    jmp et_loop
et_exit:
...
```

Își pierde valoarea în urma procedurii

! obs : Pt a ne asigura ca nu pierdem valori folosite



Apelul procedurii SCANF

In C : scanf ("%d" , &x)

In Asm : push $x adresă
push $ format Read
call scanf
pop %ebx
pop %ebx

Aplicație : citirea unui array de întregi

```
.data
    v: .space 80    # maxim 20 el
    n: .space 4
    x: .space 4
```

format Scanf : .asciz "%d"

.text
.global main

main :

push $n

push $ format Scanf

call scanf

pop %edx

pop %edx

lea v, %edi

xor %ecx, %ecx

et_loop :

```
for( i=0; i<n; i++)
{
    scanf ("%d", &x)
    v[i] = x
}
```

cmp n, %ecx

je et_exit

push %ecx

push $ x

push $ format Scanf

call scanf

pop %edx

pop %edx

pop %ecx

mov x, %esi

mov %esi, (%edi, %ecx, 4)

inc %ecx

jmp et_loop

et_exit : ...

Problemă: Citiți $n > 0$ și un array de n elemente (max 25 el)

Calculați media aritmetică și afișați-o sub forma

" Media este %d rest %d "

Sol:

```
. data
    v : . space 100
    n : . space 4
    x : . space 4
    medie : . space 4
    rest : . space 4
     format Scanf : . asciz " %d "
     format Print : . asciz " Media este %d rest %d "
. text
. global main
    main :
        push $n
        push $ format Scanf
        call scanf
        pop %ebx
        pop %ebx
                                    } citire n

        lea v, %edi
        xor %ecx, %ecx
```

```
et_loop_citire:
        cmpl %ecx, n
        je et_restaurare

        pushl %ecx
        pushl $x
        push $formatScanf
        call scanf
        pop %ebx
        pop %ebx
        pop %ecx

        movl x, %ebx
        movl %ebx, (%edi, %ecx, 4)
        inc %ecx        # i++
        jmp et_loop_citire
```

citire x

v[i] = x

```
et_restaurare:
        xorl %ecx, %ecx
        xorl %eax, %eax
et_loop_suma:
        cmp n, %ecx
        je et_exit

        movl (%edi, %ecx, 4), %ebx
        addl %ebx, %eax
        inc %ecx
        jmp et_loop_suma
```

%eax += v[i]

```
et_exit :
        xorl %edx, %edx
        divl n
        movl %eax, media
        movl %edx, rest

        push rest
        push media
        push $formatPrintf
        call printf
        pop %ebx
        pop %ebx
        pop %ebx

        push $0
        call fflush
        pop %ebx

        movl $1, %eax
        xorl %ebx, %ebx
        int $0x80
```