Laborator 0x04

1. Apelarea funcțiilor
   - PRINTF (funcția FFLUSH)
   - SCANF
2. Suma a două nr. citite de la tastatura
3. Matrice
   - Afișare
   - Suma el. pare
   - Citire

Apelarea funcțiilor printf și scanf dintr-un program scris în limbaj de asamblare

PRINTF

formatul

printf (" %d : %d " , x , y)

argumentele

=> 3 : 3

4 : 5

push y

push x

push $ format

call printf

- argumentele pe stivă în ordine inversă

pop %ebx // pop elimină din stivă și completează în registrul ce urmează în vârful stivei

pop %ebx

pop %ebx

Exemplu program

```
.data
    format Print : .asciz " %d : %d "
.text
.global main
```

```
main :
    push   $ 1
    push   $ 0
    push   $ formatPrint
    call   printf
    pop    %ebx
    pop    %ebx
    pop    %ebx

    push   $ 0
    call   fflush   // goleste buffer-ul curent in care scrie printf
    pop    %ebx

    mov    $ 1, %eax
    mov    $ 0, %ebx
    int    $ 0x80
```

□

SCANF

```
scanf ("%d", &x)

push   $x        // scanf vrea sa puna valoarea citita la adresa de memorie a lui x
push   $ format
call   scanf
pop    %ebx
pop    %ebx
```

Example program

```
.data
    x: .space 4
    formatInt: .asciz "%d"
.text
.global main
    main:
        // scanf("%d", &x)
        push $x
        push $formatInt
        call scanf
        pop %ebx
        pop %ebx

    et_exit
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

Suma a două numere citite de la tastatură

```
.data
    x: .space 4
    y: .space 4
    formatRead: .asciz "%d %d"
    formatPrint: .asciz "Suma nr. este %d \n"
.text
.global main
main:
    // scanf("%d %d", &x, &y)
    push $y
    push $x
    push $formatRead
    call scanf
    pop %ebx
    pop %ebx
    pop %ebx

    // eax = x + y
    mov x, %eax
    add y, %eax
```

```
// print ("
    push %eax
    push $formatPrint
    call printf
    pop %ebx
    pop %ebx

    push $0
    call fflush
    pop %ebx

    mov $1, %eax
    mov $0, %ebx
    int $0x80
```

□

Matrice

- vor fi stocate în memorie tot ca o zonă continuă
- elementele vor fi accesate prin intermediul a doi indecși

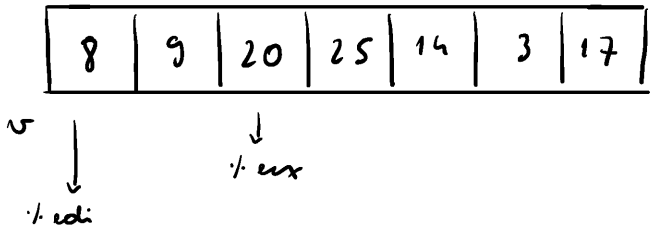
```
lines:    .long 3
columns:  .long 4
lineIndex:  .space 4
columnIndex:  .space
```

matrix : . long 10 , 20 , 30 , 40

. long 50 , 60 , 70 , 80

. long 90 , 100 , 110 , 120

La vectori



(%edi , %ecx , 4)

⇓

elementul situat la 4 * %ecx relativ

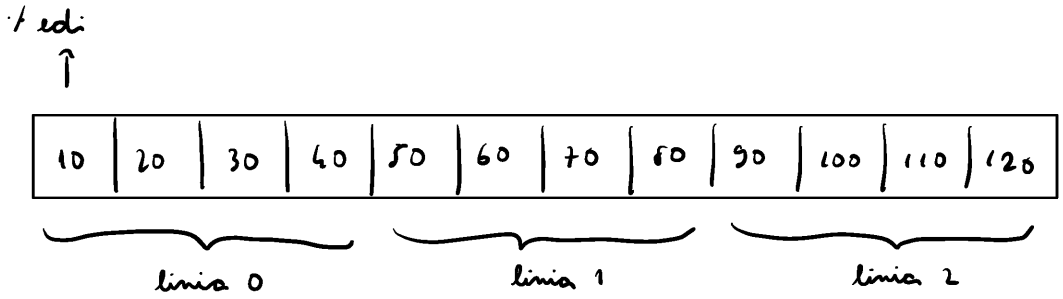
la %edi

⇓

elementul de la adresa

%edi + 4 * %ecx

La matrici



lineIndex

columnIndex

> cine este %ecx în funcție

- lineIndex
- columnIndex
- columns

(%edi, %ecx, 4)

%ecx = lineIndex * columns + columnIndex

linia 0 → lineIndex * columns + columnIndex

0

% ecx = columnIndex

0 1 2 3

linia 1 → 1 * 4 + columnIndex

4 5 6 7

linia 2 → 2 * 4 + columnIndex

8 9 10 11 12

matrix → el. curent. este dat de

(% edi , % ecx , 4)

" " "

matrix lineIndex * columns + columnIndex dim. tipului de date

Exemplu program: Afisarea unei matrici

```
.data
    lines: .long 3
    columns: .long 4
    lineIndex: .space 4
    columnIndex: .space 4
    matrix: .long 100, 20, 30, 40
            .long 50, 60, 70, 80
            .long 90, 15, 25, 35
    formatInt: .asciz "%d "
    newLine: .asciz "\n"
.text
.global main
```

main :

lea matrix , %edi

movl $0 , lineIndex

// for (int lineIndex = 0 ; lineIndex < lines ; lineIndex ++)

// for (int columnIndex = 0 ; columnIndex < columns ; columnIndex ++)

for_lines :

movl lineIndex , %ecx

cmp %ecx , lines

je et_exit

movl $0 , columnIndex

for_columns :

movl columnIndex , %ecx

cmp %ecx , columns

je cont_for_lines

// prelucrarea efectivă

// elementul curent este la

lineIndex * columns + columnIndex relativ la

adresa de început a matricii , i.e. relativ la %edi

// toate elementele sunt .long => au dim 4 B

movl lineIndex , %eax

mull columns

addl columnIndex , %eax

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

// în %ebx se află elementul curent din matrice

```
    pushl   %ebx
    push    $formatInt
    call    printf
    pop     %ebx
    pop     %ebx

    pushl   $0
    call    fflush
    pop     %ebx

    addl    $1, columnIndex
    jmp     for_column

cont_for_lines:

    mov     $4, %eax
    mov     $1, %ebx
    mov     $newline, %ecx
    mov     $2, %edx
    int     $0x80

    addl    $1, lineIndex
    jmp     for_lines

et_exit:

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