Laborator 0x02

1. Operații aritmetice 2 :
   - MUL
   - DIV

2. Salturi necondiționate și salturi condiționate

3. Simularea structurilor repetitive

4. Instrucțiunea LOOP

5. Tablouri unidimensionale

Operații aritmetice

mul op

div op

Obs mul și div au o destinație și o sursă implicită

mul op

(edx , eax) = eax * op

unde edx este completat când se depășește 2^{32}

$2^{32} \cdot$ edx + eax

eax = eax * op

Exemplu

```
mov $9, %eax
mul %ebx
```

} %eax = 9 * %ebx

mul %eax => eax^2

// %eax = %eax + %eax

div op

(%edx , %eax) = %eax / op

restul împărțirii

câtul împărțirii

Exemplu de program

Sol:

```
.data
    x:. long 30
    y:. long 7
    prod :. space 4
    cat :. space 4
    rest :. space 4
.text
.global main
main :
        mov x, %eax
        mov y, %ebx
        mul %ebx        // %eax := %eax * %ebx = x * y
        mov %eax, produs

        mov x, %eax
        div %ebx        // (%edx, %eax) := %eax / %ebx = x/y
        mov %edx, rest
        mov %eax, cat

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

□

Salturi necondiționate și salturi condiționate

Salturi necondiționate

jmp et - sare la eticheta et

Salturi condiționate

- evaluate prin cmp op2, op1

<	jl	et	>	jg	et
≤	jle	et	≥	jge	et
==	je	et			
!=	jne	et			

```
cmp %eax, %ebx
jle et
    // %ebx ≤ %eax
```

Exemplu de program

```
.data
    x: .long 5
    y: .long 8
    str1: .asciz "x < y \n"
    str2: .asciz "x >= y \n"
.text
.global main
main:
    movl x, %eax
    movl y, %ebx
    cmpl %eax, %ebx
    jg et_afis1
    jmp et_afis2
et_afis1:
    movl $4, %eax
    movl $1, %ebx
    movl $str1, %ecx
    movl $7, %edx
    int $0x80
    jmp et_exit
et_afis2:
    movl $4, %eax
    movl $1, %ebx
    movl $str2, %ecx
    movl $8, %edx
    int $0x80
et_exit:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

if (ebx > eax)
jmp et_afis

Simularea structurilor repetitive

```
int sum = 0                    exit cond i = n
for ( int i = 0; i < n; i++)
    {
        sum += i
    }
```

Sol:

```
mov $0, %ecx          # pe post de "i"
mov $0, %eax

et_for :
        cmp %ecx, n
        je et_exit

        add %ecx, %eax

        add $1, %ecx          // inc %ecx
        jmp et_for

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

□

Instrucțiunea LOOP

- lucrează cu reg %ecx prin decrementarea lui

```
mov $10, %ecx

et_for :
    ...
    loop et_for
```

verifică dacă %ecx == 0, și dacă da,
merge la linia următoare
altfel
decrementează %ecx
sare la et_for

Tablouri unidimensionale de date

- pentru întregi -

15	27	3	11	17	5

↑
v

- elementele sunt liniar așezate în memorie
- v reprezintă un nume pt. adresa de început

long v[6]

I v + 4 * 0 II v + 4 * 1 III v + 4 * 2

a (b, c, d) <=> b + c * d + a

→ offset de deplasare

↑ lipsește în general (mai des la vectori)

(b, c, d) → dimensiunea tipului de date

b → adresa de început a vectorului

c → indexul curent

lea = load effective address

lea v, %edi

(%edi, %ecx, 4)

%edi → adresa vectorului

%ecx → se incrementează la fiecare pas

4 → dim. tipului long

Declarare în memorie

v : . long 15, 27, 3, 11, 17, 5

ex

Suma elementelor dintr-un array

```
for ( %ecx = 0 ; %ecx < n ; %ecx ++ )
{
        %eax + = v [ %ecx ]
}
```

```
.data
        v: .long 10, 15, 30
        n: .long 3
.text
.global main
main:
        lea v, %edi
            // %edi reține adresa de început a array-ului
        mov $0, %ecx
        mov $0, %eax

et_for:
        cmp %ecx, n
        je et_exit

        mov (%edi, %ecx, 4), %ebx
        add %ebx, %eax

        inc %ecx
        jmp et_for
et_exit:
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

□