# Laborator 0x07

18 Nov 2024

1. Recap : proceduri
2. Manipularea stivei
3. Convenții pt. implementarea procedurilor
4. Funcție pt suma a 2 nr.

   # long sum ( long x, long y )
5. Funcție pt minimul a 2 nr.

   # long min ( long x, long y )

# Recap: Implementarea procedurilor în assembly

Apelul procedurilor printf și scanf

- pentru apel **call**
- pentru a pasa arg.: **push op**
- pentru a restaura stiva: **pop op**

printf ( "(%d, %d)\n", x, y )

```
push y
push x
push $formatPrintf
call printf
pop %ebx
pop %ebx
pop %ebx
```

scanf ( "%d", &x )

```
push $x
push $formatRead
call scanf
pop %ebx
pop %ebx
```

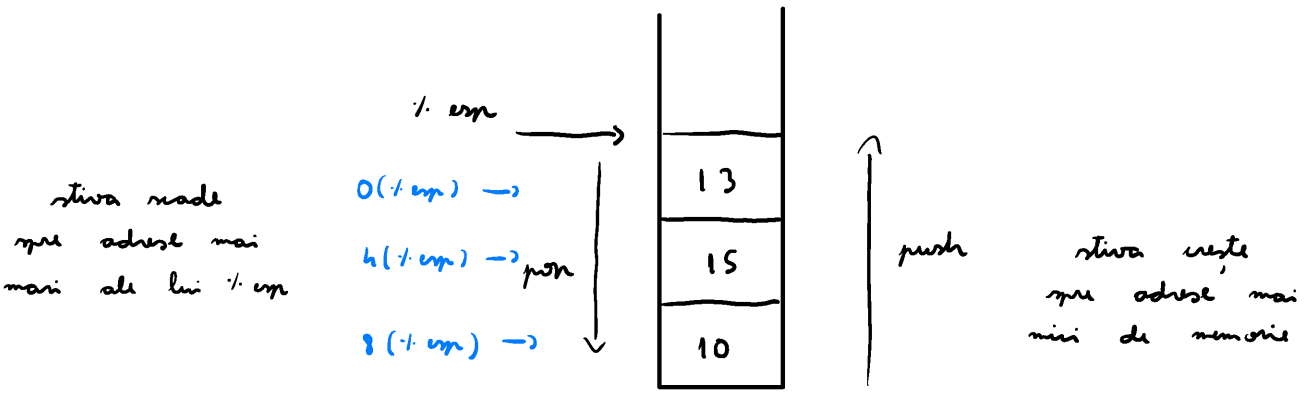
Obs În urma apelului, %eax, %ecx, %edx nu garantează păstrarea valorii

```
push %ecx
[
   <asm>
pop %ebx
```

# Manipularea stivei

- stiva este o zonă de memorie partajată gestionată prin intermediul registrului %esp (stack pointer)

- în %esp avem adresa vârfului stivei




Diferențiere

$a(b, c, d) \rightarrow a + b + c * d$

$b \in Reg$

Pe măsură ce facem
- push-uri stiva crește spre adrese mai mici
- pop-uri stiva scade spre adrese mai mari

Arhitectura pe 32 b (4 B)

→ reg. pe 4 B

→ fiecare locație pe stivă ocupă 4 B

push op

```
sub  $4, %esp
mov  op, 0(%esp)
```

pop op

```
mov  0(%esp), op
add  $4, %esp
```

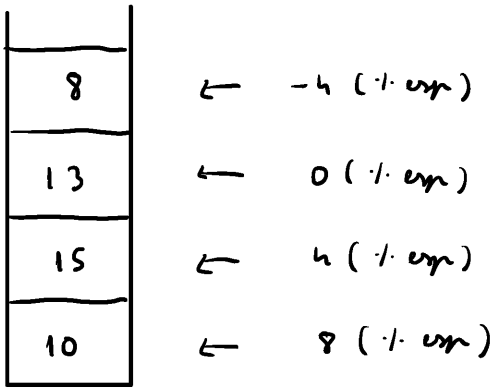
(=>)

```
add  $4, %esp
mov  -4(%esp), op
```

Exemplu :

```
printf ("%d␣%d", x, y)
```

```
push y
push x
push $formatPrintf
call printf
add  $12, %esp        # pop
                      # pop
                      # pop
```



# Convenții pt. implementarea procedurilor

1. Argumentele unei proceduri se încarcă în ordine inversă pe stivă

2. Apelul funcțiilor se face prin instrucțiunea call iar revenirea din codul de apel se face prin instrucțiunea ret.

3. Registrii %eax, %ecx, %edx NU își restaurează val. în codul de apel și sunt utilizați drept registrii de return

4. Registrii %ebx, %esi, %edi, %esp TREBUIE să își restaureze valoarea în urma codului de apel.

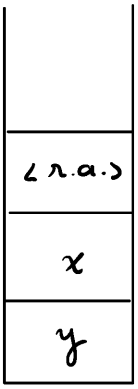
5. Se fixează %ebp în codul de apel ca pointer iar toate accesările pe stivă se fac în raport cu acesta; NU uităm să restaurăm %ebp

## Exemplu 2

```
# func ( x, y )

main :
    ...
    push  y
    push  x
    call  func

func
    ...
    ret
```




< r. a. > = return address

% eip = instruction pointer

**call** - pune adresa urm. instrucțiuni de după revenire și efectuează salt

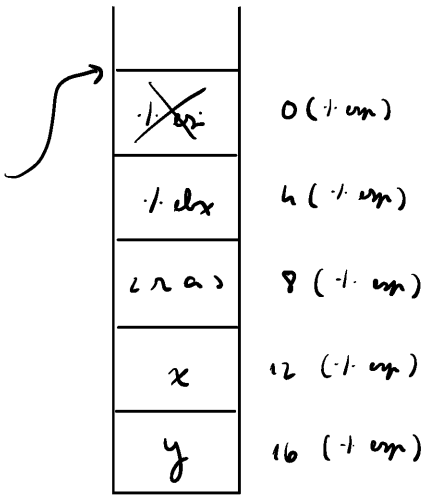
**ret** - sare la adresa din vârful stivei și îi dă pop

## Exemplu 3

```
# func ( x, y )

main :
    :
    push  y
    push  x
    call  func

func
    :
    push  %ebx
    :
    push  %esi
    :
    pop   %esi
    ret
```

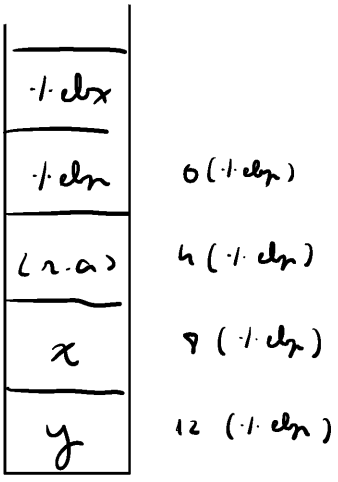


Exercițiu : suma a două nr.

```
.data
    x: .long 5
    y: .long 3
.text
.global main

sum:
obligatoriu [ pushl %ebp
              movl %esp, %ebp
    movl 8(%ebp), %eax
    pushl %ebx
    movl 12(%ebp), %ebx
    addl %ebx, %eax
    popl %ebx
    popl %ebp
    ret

main:
    pushl y
    pushl x
    call sum
    addl $8, %esp
    rezultatul este în %eax
    ...
    EXIT
```



Probleme :

ex 1

Pt procedura sum, scrieți un main în care se citesc (de la STDIN) 2 întregi și afișează rez. printr-un apel

ex 2

Scrieți o procedură pt. a calcula minimul dintre două valori

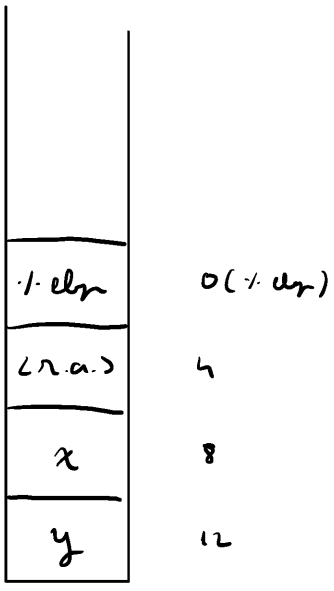
# long min ( long x, long y )

ex 2

```
min :
    push %ebp
    mov %esp, %ebp
    mov 8(%ebp), %eax      // eax ← x
    mov 12(%ebp), %ecx     // ecx ← y
    cmp %eax, %ecx
    jl change_min          // if ecx < eax
    jmp min_exit

change_min :
    mov %ecx, %eax

min_exit :
    pop %ebp
    ret
```



ex 1

```
.data
    x: .space 4
    y: .space 4
    formatScanf: .asciz "%d"
    formatPrint: .asciz "Min = %d \n"
.text
.global main
#min

main:
    push $x
    push $formatScanf
    call scanf
    add $8, %esp

    push $y
    push $formatScanf
    call scanf
    add $8, %esp

    push y
    push x
    call min
    add $8, %esp
```

```
push %eax
push $formatPrint
call printf
add $8, %esp
```

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