Recap:
- -registri
- -variabile in RAM
- -tipuri de date
- -apeluri de sistem: exit, write
- -operatii aritmetice: add, sub

Astazi:
- - operatii aritmetice
- - salturi neconditionate si conditionate

Operatia MUL (inmultire)
-> mul op (op poate fi registru SAU adresa de memorie)
-> Functioneaza conform unor conventii referitoare la sursa, respectiv destinatia implicite
mul op == (eax, edx) [destinatie implicita] = %eax [sursa implicita] * op
-> daca o inmultire depaseste 32 biti, restul ajunge in edx
-> mul op <=> %eax *= op

Inmultirea a doua numere

```
.data
    x: .long 5
    y: .long 3
.text
.globl main
main:
    movl x, %eax
    movl y, %ebx
    mul %ebx
    #pt a inmulti eax cu o variabila pot folosi mull (mul long): mull y, mull $2
    mull y
```

Operatia DIV
-> div op
-> op = registru/adresa
div op <=> (%edx, %eax) [destinatie implicita]= (%edx, %eax) [sursa implicita] / op
-> %eax = catul, %edx = restul

```
.data
    x: .long 30
    y: .lonng 7

.text

.globl main

main:
    movl $0, %edx
    movl x, %eax
    divl y
```

Salturi neconditionate
-> jmp et
-> sare din punctul curent al programului la eticheta precizata

```
et1:
 ......
  ......
  jmp exit --
  et2:      |
exit:    <---
```

Salturi conditionate
-> sunt salturi care depind de o relatie de ordine
j<cond> et
cmp op1, op2
!!!! op2 obligatoriu registru
< jl
<= jle
> jg
>= jge
== je
!= jne

-> cmp op1, op2 # op2 < op1
-> jl et

Implementarea structurilor repetitive

```
//var 1
sum = 0;
for (int i = 0; i < n; i++) {
    sum += i
}

    %ecx = 0
for:
    cmp i, n
    je exit
    //instr
    %ecx++
    jmp for


exit:
```

// var 2
-> loop: este implementata prin decrementarea, la fiecare iteratie, a reg %ecx
-> se executa cel putin o data instructiunile,
se decrementeaza %ecx
if (%ecx == 0) se iese din str rep
altfel se face salt la et
-> nu o vom utiliza asa mult, nu e buna la array-uri

```
loop et
et:
    //instr
```

loop et