Laborator 0x03

1. Verificare număr prim
2. Suma a două array-uri
3. Elementul singular dintr-un array folosind XOR

Recapitulare

div op
mul op
> au o destinație implicită și un operand implicit

- mul %ecx

→ (%edx, %eax) = %eax * %ecx

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

- div %ecx

→ (%edx, %eax) = %eax / %ecx

restul ↙ câtul ↘

↓ modif

(%edx, %eax)

($2^{32} \cdot$ %edx + %eax) / %ecx

↓

inițial pun 0 în %edx

mov $0, %edx → pt a ne asigura de împărțirea corectă
div %ecx

Salturi necondiționate și condiționate

- Saltul necondiționat

```
jmp label
```

- Salturi condiționate (bazate pe comparații aritmetice)

```
cmp %eax, %ebx
```

$\leq$	`jle label`	$\Leftrightarrow$	dacă %ebx $\leq$ %eax $\rightarrow$ label
$<$	`jl label`	$\Leftrightarrow$	dacă %ebx $<$ %eax
==	`je label`		
!=	`jne label`		
$>$	`jg label`		
$\geq$	`jge`		

Simulare structură repetitivă

```
for ( i = 0; i < n; i++ )
  { ... }
```

```
i   mov $0, %ecx        # %ecx = "i"
et_for :
        cmp %ecx, n
        je et_exit
          # prelucrare
          inc %ecx      # add $1, %ecx
        jmp et_for
et_exit :
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

□

II

```
mov n, %ecx
et_for :
    # prelucrare
    loop et_for
et_exit :
    mov $1, %eax
    mov $0, %ebx
    int $0x80
```

□

loop et_for :

se citește val. din reg %ecx

- dacă %ecx != 0 atunci
  decrementează %ecx
  sare la et_for
- dacă %ecx = 0 atunci
  sare la linia următoare,
  (în cazul nostru, et_exit)

Array-uri

8	13	25	6	18	27	1

n = 7

↑
v

lea v, %edi

load effective adress → încarcă în reg %edi adresa de început al lui v

daca %ecx este contorul curent, atunci

elementul curent este (%edi, %ecx, 4)

I v → v + 0 * 4

II v + 4 → v + 1 * 4

III v + 8 → v + 2 * 4

(v, index, dimensiune)

↓ ↓ ↓

edi ecx pentru long

4

Verificare număr prim

n: .long 13

n < 2 → compus

n == 2 → prim

n %. 2 == 0 → compus

```
for (int i = 3; i <= n/2; i++)
        if (n %. i == 0)
                compus
```

PRIM

Alt pentru împărțire

```
mov $0, %ecx
div %ecx
```

```
.data
    n: .long 17
    prim: .asciz "nr prim \n"
    compus: .asciz "nr compus \n"
.text
.global main
    main:
        mov n, %eax
        mov $2, %ebx
    et_verif:
        cmp %ebx, %eax
        jl et_compus          ] if n < 2 compus

        cmp %ebx, %eax
        je et_prim            ] if n == 2 prim

        mov $0, %edx
        div %ebx
        cmp $0, %edx
        je et_compus          ] if n % 2 == 0 compus

        mov n, %eax
        mov $0, %edx
        mov $2, %ebx
        div %ebx
        mov %eax, %ebx        ] mi halipare n/2
```

```
mov $3, %ecx        // i = 3

et_loop:
    cmp %ebx, %ecx      ]
    jg et_prim          ]  if i > n/2 prim

    mov n, %eax         ]
    mov $0, %edx        ]
    div %ecx            ]
    mov $0, %eax        ]  if n%i == 0 compus
    cmp %eax, %edx      ]
    je et_compus        ]

    add $2, %ecx        // i = i + 2
    jmp et_loop
```

```
et_prim:
    mov $4, %eax
    mov $1, %ebx
    mov $prim, %ecx
    mov $9, %edx
    int $0x80
    jmp et_exit
```

```
et_compus:
    mov $4, %eax
    mov $1, %ebx
    mov $compus, %ecx
    mov $11, %edx
    int $0x80
```

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

Suma a două array-uri

```
v : . long  5, 6, 7
w : . long  10, 20, 30, 40
r : . space 100
```

=> r = [15, 26, 37, 40]

```
lea v, %edi
lea w, %esi

mov %edx, (%ebx, %ecx, 4)
```

```
. data
    v: . long  5, 6, 7
    n : . long 3
    w : . long 10, 20, 30, 40, 50
    m :. long 5
    r :. space 100
. text
. global main
    main :
        lea v, %edi
        lea w, %esi
        lea r, %ebx

        mov $0, %ecx
```

```
        mov  n, %ah
        mov  m, %al
        cmp  %al, %ah
        jg   et_initializare_n
        jmp  et_initializare_m

et_initializare_n:
        cmp  n, %ecx
        je   et_restaurare_n

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

        inc  %ecx
        jmp  et_initializare_n

et_restaurare_n:
        mov  $0, %ecx
et_adaugare_n
        cmp  m, %ecx
        je   et_exit

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

        inc  %ecx
        jmp  et_adaugare_n
```

```
et_initializare_m :
    cmp m, %ecx
    je et_restaurare_m

    mov (%esi, %ecx, 4), %eax
    mov %eax, (%ebx, %ecx, 4)

    inc %ecx
    jmp et_initializare_m

et_restaurare_m :
    mov $0, %ecx
et_adaugare_m
    cmp n, %ecx
    je et_exit

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

    inc %ecx
    jmp et_adaugare_m

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

Elementul singular dintr-un array folosind XOR

Utilizand XOR, identificati elementul singular in $n = 2k+1$ elemente $\begin{cases} 2k \text{ elemente perechi} \\ 1 \text{ element singular} \end{cases}$

[10, 15, 3, 10, (8), 15, 3] → 8

XOR

$0 \oplus 0 = 1 \oplus 1 = 0$

$1 \oplus 0 = 0 \oplus 1 = 1$

$\begin{cases} a \oplus a = 0 \\ a \oplus 0 = a \end{cases}$

$\oplus$ asociativ

XOR op 1, op 2 $\Rightarrow$ op 2 = op 2 $\oplus$ op 1