Laborator 0x02 - Probleme

Maria Preda

October 2025

Exercitiu 0x00

Consideram urmatoarele declaratii in sectiunea .data:

```
str1: .ascii "ASC"
str2: .ascii "FMI"
```

Si urmatoarea afisare:

```
mov $4, %eax
mov $1, %ebx
mov $str1, %ecx
mov $5, %edx
int $0x80
```

Care este textul afisat?

Solutie:

str1 are doar 3 caractere, iar noi incercam sa afisam 5, asadar, se va trece si la sirul declarat dupa in memorie, adica str2.
Se va afisa: ASCFM.

Exercitiu 0x01

Consideram urmatoarele declaratii in sectiunea .data:

```
str1: .ascii "ASC"
str2: .long 0X31323334
```

Si urmatoarea afisare:

```
mov $4, %eax
mov $1, %ebx
mov $str1, %ecx
mov $5, %edx
int $0x80
```

Ce se va afisa? Dar in cazul in care avem str2: .long 0X31323348?

Observatie: Arhitectura x86 foloseste sistemul little-endian, ceea ce inseamna ca octetii din numere sunt stocati in ordine inversa in memorie. De exemplu, pentru numarul $0X31323334$, am avea octetii (in hexazecimal): 31 32 33 34. De fapt, in memorie o sa ii avem 34 33 32 31. Pentru sirurile de caractere, octetii sunt stocati in ordinea in care apar.

Solutie:

str1 are doar 3 caractere, iar noi incercam sa afisam 5, asadar, se va trece si la sirul declarat dupa in memorie, adica str2. Va interpreta octetii luati din str2 ca fiind coduri ASCII si va afisa: ASC43, deoarece:

$$0X31 \rightarrow 49_{(10)} \rightarrow 1\ (in\ ASCII)$$
$$0X32 \rightarrow 50_{(10)} \rightarrow 2\ (in\ ASCII)$$
$$0X33 \rightarrow 51_{(10)} \rightarrow 3\ (in\ ASCII)$$
$$0X34 \rightarrow 52_{(10)} \rightarrow 4\ (in\ ASCII)$$

Asadar, se va trece la primi doi octeti din str2, care, asa cum am mentionat mai devreme sunt reprezentati ca fiind:

34 33 32 31

Deci se iau octetii 34 si 33, motiv pentru care, dupa str1, se mai afiseaza 4 si 3.

In cazul in care avem str2: .long 0X31323348, atunci se va afisa "ASCH3".

Exercitiul 0x02

```
.data
.text
.global main

main:
    movl $0, %ecx
    movl $3, %eax
    movl $5, %ebx
    cmp %ebx, %eax
    jge greater_or_equal

    movl $1, %ecx
    jmp end

greater_or_equal:
    movl $2, %ecx

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

Analizati si rulati urmatorul program. Care este valoarea din %ecx cand se ajunge la eticheta end?

Solutie:

```
.data
.text
.global main

main:
    movl $0, %ecx
    movl $3, %eax        ; incarca 3 in EAX
    movl $5, %ebx        ; incarca 5 in EBX
    cmp %ebx, %eax       ; compara EAX cu EBX
    jge greater_or_equal  ; sare la 'greater_or_equal' daca EAX >= EBX

    # Cod pentru cazul in care EAX < EBX
    movl $1, %ecx        ; setam ECX la 1
    jmp end              ; sare la sfarsit

greater_or_equal:
```

```
    # Cod pentru cazul in care EAX >= EBX
    movl $2, %ecx        ; setam ECX la 2

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

In acest exemplu, la finalul programului, %ecx va avea inca valoarea 1.

Exercitiul 0x03

Avem urmatorul program:

```
.data
    x1: .ascii "maria"
    x2: .ascii "maria1"
    x3: .ascii "maria2"
.text
.global main

main:

    mov $4, %eax
    mov $1, %ebx
    mov $x1, %ecx
    mov $14, %edx
    int $0x80

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

Dupa executarea programului, ce se va afisa?

Solutie:

Programul va afisa: mariamaria1mar. De ce?

Trebuie sa afisam 14 bytes, fiecare caracter ascii ocupa exact un byte. Sirul x1 are doar 5 bytes, deci se va continua cu ceea ce avem in memorie. Aceasta

fiind liniara, inseamna ca sirurile x2 si x3 sunt la rand dupa x1, deci vor fi luati bytes din ele.

Exercitiul 0x04

Avem urmatorul program:

```
.data
    x1: .ascii "maria"
    x2: .long  0x61676362
    x3: .ascii "maria2"
.text
.global main

main:

    mov $4, %eax
    mov $1, %ebx
    mov $x1, %ecx
    mov $14, %edx
    int $0x80

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

Dupa executarea programului, ce se va afisa?

Solutie:

Programul va afisa: mariabcgamaria. De ce?

Trebuie sa afisam 14 bytes, fiecare caracter ascii ocupa exact un byte. Sirul x1 are doar 5 bytes, deci se va continua cu ceea ce avem in memorie. Aceasta fiind liniara, inseamna ca sirurile x2 si x3 sunt la rand dupa x1, deci vor fi luati bytes din ele. Pentru ca, atunci cand lucram cu numere, octetii sunt reprezentati invers in memorie, dupa ce sunt luati octetii, transformati in coduri ascii, vor aparea invers.

Exercitiul 0x05

Avem urmatoarea secventa de cod:

```
        mov $0x2, %eax
        mov $0x00000001, %edx
        mov $2, %ebx
        div %ebx

    et_exit:
```

Daca executam programul cu gdb si scriem urmatoarele linii:

```
        b et_exit
        run
        i r eax
        i r edx
```

Ce valori vor fi afisate?

Solutie:

Vom avea registrii %edx si %eax concatenati, deci, in total, vom avea valoarea: $2^{32} + 2$ (deimpartitul). valoarea la care impartim este 2, deci vom avea catul (in %eax) $2^{31} + 1$, iar restul (in %edx) 0. Acestea sunt valorile pe care le vom avea in cei doi registri atunci cand ajungem la eticheta et_exit.

Exercitiul 0x06

Fie urmatorul program:

```
        .data

        .text

        .global main

        main:
            mov $0xff67ffff, %ecx
```

```
    mov $0x5263dfaa, %eax

    cmp %ecx, %eax
    jge et_exit

et_1:
    mov $4, %ecx

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

Ce valoare va fi stocata in registrul %ecx atunci cand programul ajunge la eticheta et_exit.

Valoarea stocata in registrul %ecx va fi $0xff67ffff$.

Solutie:

"jge" este folosit pentru numere cu semn. Cum f este, in baza 2 reprezentat ca 1111, inseamna ca prima cifra a numarului din registrul %ecx este 1, deci numarul este negativ, in timp ce 5 este 0101, ceea ce inseamna ca numarul din registrul %eax este pozitiv, deci, automat, mai mare decat cel din %ecx.

jge $\%ecx$, $\%eax$ se interpreteaza ca fiind $\%eax \geq \%ecx$, ceea ce este adevarat, deci se face direct saltul la eticheta et_exit, asa ca valoarea din registrul %ecx ramane cea initiala.

Exercitiul 0x07

Fie urmatoarele declaratii in sectiunea .data:

```
a: .ascii "Assembly"
b: .word 0x25
c: .asciz "x86"
d: .asciz ";;;"
e: .long 0x15
```

Ce se intampla dupa aceasta secventa de cod?

```
mov $4, %eax
```

```
mov $1, %ebx
mov $a, %ecx
mov $b, %edi
sub %ecx, %edi
or %edi, e
mov e, %edx
int $0x80
```

Solutie:

Se va afisa mesajul: "Assembly%x86;;". La apelul de sistem, adica in dreptul randului "int $0x80", se va verifica codul functiei care trebuie apelata, adica numarul aflat in %eax. Fiind 4, inseamna ca se va apela functia WRITE. In registrul %ebx avem valoarea 1, ceea ce inseamna ca se va afisa in consola. In registrul %ecx am pus adresa lui a, ceea ce inseamna ca afisarea incepe de la primul caracter din a.

In continuare, trebuie sa calculam valoarea aflata in registrul %edx la momentul apelului de sistem.

In %ecx avem adresa lui a, iar in %edi adresa lui b. Diferenta dintre adrese este 8 (lungimea in bytes a sirului a).

Aplicam operatia logica or intre %edi (1000_2) si e (10101_2) si obtinem $0x1D$, ceea ce inseamna ca trebuie sa afisam 29 de bytes: 8 de la a, 2 de la b, 4 de la c, 4 de la d, 4 de la e si am mai avea inca 7 bytes pe langa. Valoarea din b va fi convertita in ascii si obtinem caracterul %. Valoarea din e reprezinta NAK, care nu poate fi afisat. In continuare nu mai avem caractere afisabile, deci pare ca afisarea se opreste mai devreme decat ar trebui, cu toate ca incearca, de fapt, sa afiseze 29 de bytes.

Exercitiul 0x08

Se da un numar natural n, pe care il veti delcara cu valoarea 10. Calculati:

- n!.
- Al n-lea termen Fibonacci.
- Al n-lea termen al unui sir, stiind ca $x_i = x_{i-1} + x_{i-2} + x_{i-3}$, unde $i >= 3$, iar primele 3 valori ale sirului sunt 0, 5, respectiv 2.

Solutie:

Pentru calcularea lui n!, avem codul:

```
.data
 n: .long 10
.text
.global main
main:
 mov n, %ecx
 mov $1, %eax

et_loop:
 mul %ecx
 loop et_loop


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

Pentru a calcula termenul n din sirul Fibonacci, avem codul:

```
.data
 n: .long 10
 t1: .long 0
 t2: .long 1
.text
.global main
main:
 mov n, %ecx
 subl $2, %ecx


et_loop:
 movl t2, %eax
 movl t1, %ebx
 addl t2, %ebx
 movl %ebx, t2
 movl %eax, t1

 loop et_loop

et_exit:
 mov $1, %eax
 xor %ebx, %ebx
```

```
int $0x80
```

Pentru a calcula al n-lea termen din sirul de la ultimul subpunct, avem codul:

```
.data
    n:  .long 10
    t0: .long 0
    t1: .long 5
    t2: .long 2

.text
.global main
main:
    mov n, %ecx
    subl $3, %ecx

et_loop:
    cmpl $0, %ecx
    je et_exit

    mov t0, %eax
    add t1, %eax
    add t2, %eax

    mov t1, %edx
    mov %edx, t0
    mov t2, %edx
    mov %edx, t1
    mov %eax, t2

    loop et_loop

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

Exercitiul 0x09

Se da un numar natural in registrul EAX, calculati dimensiunea celui mai lung sir de biti "1" consecutivi in reprezentarea binara a numarului.

Solutie:

```
.text
.global main
main:
    movl $0b11011111111110111111,  ; 0b este folosit pentru baza 2
    xorl %ebx, %ebx
    xorl %ecx, %ecx           ; daca aplicam xor intre o valoare si ea insasi, obtinem 0
         ; asadar, aceasta linie este echivalenta cu mov $0, %ecx
    movl $32, %edx

et_loop:
    testl $1, %eax
    jz et_zero
    incl %ecx
    cmpl %ebx, %ecx
    jle et_next
    movl %ecx, %ebx

et_next:
    shrl $1, %eax
    decl %edx
    jnz et_loop
    jmp et_exit

et_zero:
    xorl %ecx, %ecx
    jmp et_next

et_exit:

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

Observatie: Aceasta problema poate fi rezolvata si folosind faptul ca suma patratelor numerelor naturale pana la n este $\frac{n(n+1)(2n+1)}{6}$. Am folosit loop-ul pentru exemplificarea structurilor repetitive.

(Problema preluata din suportul de laborator)

Exercitiul 0x0A

In registrul EAX aveti un numar natural. Gasiti cea mai apropiata putere de 2 mai mare decat acest numar. (verificati daca x86/x87 are deja o instructiune care realizeaza aceasta operatie, daca da atunci ganditi-va de ce este importanta aceasta operatie)

(Problema preluata din suportul de laborator)

Solutie:

```
.data
 x: .long 77
.text
.global main
main:
    movl x, %eax
    movl $1, %ebx

et_loop:
    cmpl %eax, %ebx
    ja et_exit
    shll $1, %ebx
    jmp et_loop

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

Exercitiul 0x0B

Scrieti un program care calculeaza suma pătratelor numerelor naturale de la 1 la n, unde $n = 6$.

Solutie:

```
.data
n: .long 6
s: .long 0
```

```
.text
.global main
main:
    movl n, %ecx
    xorl %eax, %eax
    xorl %ebx, %ebx

et_loop:
    cmpl $0, %ecx
    je et_exit

    movl %ecx, %eax
    mull %eax
    addl %eax, %ebx
    decl %ecx
    jmp et_loop

et_exit:
    movl %ebx, s
    movl $1, %eax
    xorl %ebx, %ebx
    int $0x80
```