**Exercitii tutoriat 3 Rezolvate**

**Mini exercitiu:** Ce valoare va avea eax dupa a doua operatie de mai jos?

.data

.text

.globl main

main:

    movl $1, %eax

    not %eax

    movl $1, %eax

    movl $0, %ebx

    int $0x80

Raspuns: -2

Not %eax intoarce complementul unui numar, deci din 0x00000001 va ajunge 0xFFFFFFFE. Acest numar este reprezentat in complement fata de 2.

Daca folosim formula de la Cursul 1, facem complementul, adunam 1 si o sa ajungem la %eax = -2.

**Mini exercitii 2:** Ce valoare va avea eax dupa a doua operatie de mai jos?

.data

.text

.globl main

main:

    movl $0x7fffffff, %eax

    addl $1, %eax

    movl $1, %eax

    movl $0, %ebx

    int $0x80

Raspuns: -2^31

Eax la inceput:

|  |
| --- |
| 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |

Daca adunam 1:

|  |
| --- |
| 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |

0x80000000

Fiind o reprezentare in complement fata de 2, most significant bit este acum 1, deci numarul va fi negativ. Acum nu mai putem folosi formula de la cursul 1, deoarece complementul + 1 ne va da acelas lucru. E ciudat ca adaugand la un numar pozitiv sa ajungem la unul negativ, dar prin acel + 1 am dat overflow la numerele pozitive si am ajuns la cel mai mic numar posibil in 32 biti (cu aceasta reprezentare).

**Ex1:**

Fie codul de mai jos. Care este valorea lui s cand executia ajunge la et\_exit?

.data

s: .long 0

.text

.globl main

main:

    mov $1, %edx

    mov $0, %eax

    movl $0xffffffff, %ebx

    divl %ebx

    mov %eax, %ecx

    movl %eax, s

    movl $1, %eax

    movl $0, %ebx

    int $0x80

Rezultat: 1

Deimpartitor: (mov $1, %edx ; mov $0, %eax) => 2^32(1) + 0

|  |  |
| --- | --- |
| EDX (32 biti) | EAX (32 biti) |
| 000…………………………………1 | 000………………………………….0 |

Impartitor: (movl $0xffffffff, %ebx)

|  |
| --- |
| EBX (32 biti) |
| 1111…………………….…………1 |

Dupa facem divl %ebx. Poate primul instinct ar fi ca ebx = -1 deci numarul nostru se imparte la -1, DAR div este pentru numere naturale => il vede pe ebx ca un numar natural.  
Deci ebx va fi 2^32 – 1.

O sa avem 2^32 / (2^32 – 1) = 1 rest 1

**Ex 2:**

Se stocheaza in registrul eax valoarea 0x40000000, in ebx 0x8, in ecx 0x1 si in edx 0x8.

Ce valori vor avea registrii eax si edx dupa executarea instructiunii mul %edx?

Raspuns: eax = 0, edx = 2

Eax la inceput arata asa:

|  |
| --- |
| EAX (32 bits) |
| 0100……………..0 |

Cand inmultim eax \* edx o sa avem 2^30 \* 2^3 = 2^33 unde se va reprezenta astfel:

|  |  |
| --- | --- |
| EDX (32 bits) | EAX (32 bits) |
| 00000……………………………..10 | 00000000000000………………..0 |

Rezultatul este 2^32(edx) + eax = 2^33 => edx = 2 si eax = 0

Sau puteti sa va ganditi ca incap numai 32 de biti in eax si restul in edx.

**Ex 3:**

Fie codul de mai jos. Care este valoarea depozitata la final in ecx (in dreptul etichetei exit) ?

.data

x: .long 0x80000000

y: .long 0x70000000

.text

.globl main

main:

mov x, %eax

cmp y, %eax

jge label

mov $5, %ecx

jmp exit

label:

mov $6, %ecx

exit:

mov $1, %eax

mov $0, %ebx

int $0x80

Rezultat: 5

La prima impresie, x pare mai mare de cat y, unul e cu 8 altul cu 7, deci de ce nu se duce pe label?

Hai sa ii reprezentam pe x si pe y.

|  |
| --- |
| X: 100000000000000000000000000000000 |
| Y: 01110000000000000000000000000000 |

Observam ceva, la x most significant bit este 1, deci numarul este negativ, iar y este pozitiv => y > x.

Continuand in problema ajungem la cmp y, %eax; jge label. Ce in pseudocod s-ar transforma de genul: if %eax >= y, ceea ce este fals, deci nu va da jump la label. Il va face pe ecx 5 si dupa se va duce la exit unde se va opri.

Ex 4: (original in lab3.1 penultima poza, aici este schimbat)

Fie urmatorul program. Ce valoare vom obtine daca vom rula cu debuggerul

urmatoarele comenzi?

b et\_exit

run

i r ebx i r edx

.data

.text

.globl main

main:

mov $11, %eax

shr $1, %eax

mov $3, %ebx

mul %ebx

mov $0, %edx

xor $7, %eax

mov $8, %ebx

div %ebx

sub %eax, %ebx

et\_exit:

mov $1, %eax

mov $0, %ebx

int $0x80

Rezultat: ebx = 7, edx = 0

Mergem pas cu pas:

1: mov $11, %eax => %eax = 11 => 1011 (in biti)

2: shr $1, %eax => %eax = 5 => 101 (in biti)

3: mov $3, %ebx => %ebx = 3

4: mul %ebx => %eax = 15

5: mov $0, %edx

6: xor $7 , %eax =>

1111 XOR

0111

1000 = %eax = 8

7: mov $8, %ebx

8: div %ebx => %eax = 1, %edx = 0 (rezultatul la 8/8)(%eax/%ebx)

9: sub %eax, %ebx => %ebx = 8 – 1 = 7

**Ex 5 :** Care este ordinea de trecere prin etichete?

.data

.text

.global main

main:

mov $1, %eax

mov $2, %ebx

mov $3, %ecx

mov $4, %edx

cmp %ebx, %eax

je etx

ety:

    cmp %ecx, %edx

    jg etz

    jmp ett

etx:

    jmp ety

etz:

    mov %ebx, %edx

    jmp ety

ett:

    mov $1, %eax

    mov $0, %ebx

    int $0x80

Rezultat: ety, etz,ety,ett

Cmp %ebx, %eax; je etx inseamna if %eax == %ebx : jump la etx

Urmatorul calcul este cmp %ecx, %edx; jg etz ce inseamna if %edx > %ecx : jump etz ceea ce se intampla.

La urmatorul calcul pune ebx in edx si face jump la ety. Acum ecx este mai mare de cat edx deci nu mai face primul jump, ci se duce pe al doilea. In ett se face exit.

**Optional:** (se va duce pe ramura ceva sau continua? De ce?)   
[aici gasiti toate jumpurile](https://faydoc.tripod.com/cpu/jnp.htm)

.data

.text

.globl main

main:

    movl $0x2, %eax

    test $1, %eax

    jp ceva

    movl $1, %eax

    movl $0, %ebx

    int $0x80

ceva:

    movl $1, %eax

    movl $0, %ebx

    int $0x80

Rezolvare: se duce pe “ceva”

**JP este un jump** care nu este predat, dar foarte important. Acesta face jump atunci cand Parity flag este activat**. Parity flag** se uita la primul bit (cel din dreapta de tot) si vede daca este 1 sau 0. Daca numarul este par face jump, daca este impar, nu face jump. (JNP este jump daca numarul este impar).

“test” este un fel de AND care nu schimba nimic, ci doar activeaza flagurile.

Deci facem:

10 AND

01

00 -> parity flag este activat deoarece primul bit este 0.

La ce va ajuta asta? (pai in loc sa faceti o impartire, sa schimbati edx, eax, alt registru, sa verificati edx daca este 0 si dupa sa faceti jump, puteti scrie ce e mai sus, si nu distrugeti nici un registru/valoare)

**Optional 2:** (se va duce pe ramura ceva sau continua? De ce?)

.data

.text

.globl main

main:

    movl $0xfffffff2, %eax

    test %eax, %eax

    js ceva

    movl $1, %eax

    movl $0, %ebx

    int $0x80

ceva:

    movl $1, %eax

    movl $0, %ebx

    int $0x80

Rezultat: se va duce pe altceva.

Flagul JS este folosit pentru a verifica daca most significant bit este activat. Adica daca numarul este negativ, sau pentru tratarea erorilor care le-am vazut mai sus (cand de la cel mai mare numar pozitiv ajunge la cel mai mic numar pe numere intregi).

Numarul este negativ deci va merge pe ramura “altceva”.

Eu **va recomand** sa testati si alte jumpuri cu flaguri ca sa vedeti ce artificii de calcule puteti face.

**Exercitiu greut:** Aflati daca un numar este prim.

.data

n: .long 17

gre: .asciz "Numarul nu este prim\n"

cor: .asciz "Numarul este prim\n"

.text

.global main

main:

movl $2, %ecx

for: # for(ecx = 2, ecx < n, ecx ++)

    cmp n, %ecx

    je corect # daca nu iese din loop e corect

    movl n, %eax

    xorl %edx,%edx

    div %ecx

    cmp $0, %edx

    je gresit

    # daca %eax/%ecx are restul 0 atunci NU e prim

    incl %ecx

    jmp for

corect:

movl $4, %eax

movl $1, %ebx

movl $cor, %ecx # "Numarul este prim\n"

movl $19, %edx

int $0x80

jmp exit

gresit:

movl $4, %eax

movl $1, %ebx

movl $gre, %ecx # "Numarul nu este prim\n"

movl $22, %edx

int $0x80

exit:

movl $1, %eax

xorl %ebx, %ebx

int $0x80

O sa invatam mai incolo printf si scanf (asemenatoare cu cin cout). Atunci va fi mult mai usor de rezolvat o problema de genul.