www.onlinegdb.com / online_c++_compiler : language: ASSEMBLY
extra comp: -no-pie

LAB 2

add op1, op2 : op2 = op2 + op1

! op2 = registru
op1 = registru / val. din memorie

add $10, %eax → eax = eax + 10
mov $10, %eax → eax = 10

sub op1, op2 : op2 = op2 - op1

mov $50, %eax
sub $5, %eax → eax = 45

mul op1 $(2^{32} \cdot edx + eax)$

! op1 este neapărat un registru
|edx, eax| = eax · op1 ← înmulțește eax cu op1
În cazul în care rezultatul înmulțirii depășește 32 de biți, primii 32 vor merge în **eax** și ultimii 32 în **edx**

mov $5, %eax
mov $6, %ebx
mul %ebx ← cu eax implicit

div op1

! op1 - registru
(edx, eax) = (edx, eax) / op1
în edx vom avea restul împărțirii
în eax vom avea câtul împărțirii

mov $16, %eax
mov $0, %edx ← de obicei vom pune 0 în edx
mov $3, %ebx
div %ebx ← în eax = 5, edx = 1

! ex: înseamnă că se împarte $(2^{32} * edx + eax) / op1$

imul, idiv
sunt pentru nr. cu semn

EXEMPLU DE PROGRAM:

```
.data
a: .long 30
b: .long 7
sum: .space 4
dif: .space 4
prod: .space 4
cat: .space 4
rest: .space 4
.text
.global main
main:

mov a, %eax
add b, %eax
mov %eax, sum          ← suma

mov a, %eax
mov b, %ebx
sub %ebx, %eax
mov %eax, dif          ← diferența

mov a, %eax
mov b, %ebx
mul %ebx
mov %eax, prod         ← produsul

mov a, %eax
mov b, %ebx
div %ebx
mov %eax, cat
mov %edx, rest         ← împărțirea

etexit:   ← label (îl denumim cum vrem)
          sunt puse pt. a ne folosi de jump-uri
mov $1, %eax
mov $0, %ebx           ← pt. a închide programul
int $0x80              ← system call
```

OPERATORI LOGICI:

not op : negația

1 → 0
0 → 1

and op1, op2 : „și"

și logic pe biți

op2 = op2 & op1

&	0	1
0	0	0
1	0	1

or op1, op2

op2 = op2 | op1

|	0	1
1	1	1
0	0	1

xor op1, op2

op2 = op1 ^ op2

^	0	1
0	0	1
1	1	0

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

] un alt mod de a scrie instrucțiunea

shl nr, op

shr nr, op

→ shiftare la dreapta și stânga; dar nu ține cont de bitul de semn

sal nr, op

sar nr, op

→ -"- dar păstrează bitul de semn, nu îl shiftează

! La shiftarea la ~~dreapta~~ stânga dispar biți dacă trec de 32.

jmp nume_label

↳ poate fi condiționat
↳ sare la acel label ignorând instr. până la el

pc - e un număr care știe la ce instr suntem

! Numele label-urilor pot fi alese de noi și trebuie urmate de „:"

SALTURI CONDIȚIONATE:

addl op1, op2

→ ↑ registrul de flags în care avem inf. ref. la ultimul

`cmp op1, op2` : comp op1 cu op2

↳ rez. va fi în flags

`je label` : se duce la label dacă op1 = op2

`jne label` : —"— op1 ≠ op2

`jg label` : se duce la label dacă op2 > op1

`jge label` : —"— op2 >= op1

`jl label` : —"— op2 < op1

`jle label` : —"— op2 <= op1

+ în materialul de la lab

`loop label`

↳ în ecx

+ se duce la eticheta label dacă ecx ≠ 0 și decrementează automat pe ecx