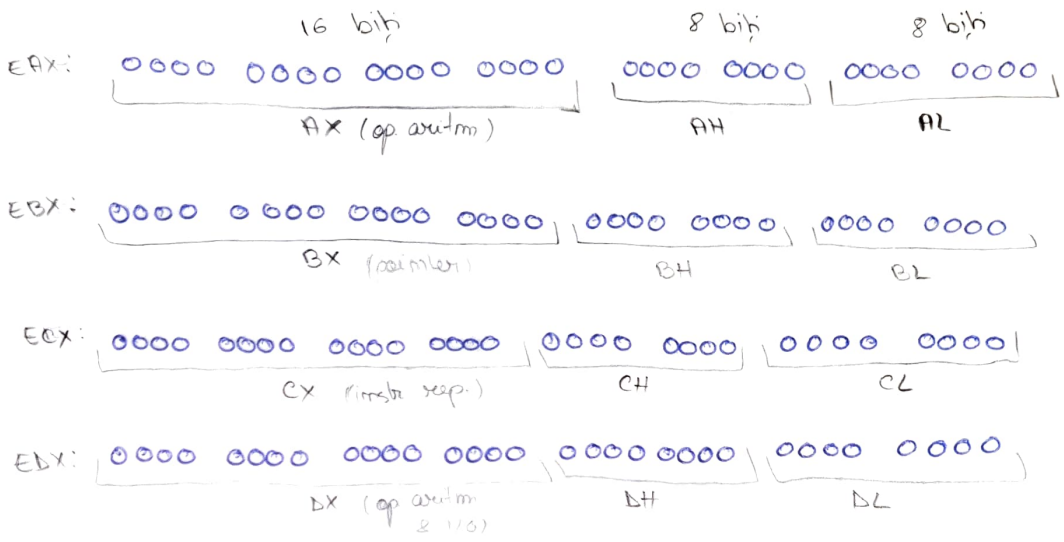
PT TEST LABORATOR

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$

Regiștii



- Întotdeauna se citesc invers: dr -> stg
- Regiștii sunt pe 32 biți
- $ -> semnifică preluarea adresei din memorie
  -> semnifică faptul că e o constantă / prefixarea unei constante

---

1 byte = 8 biți
1 single = 4 bytes = 32 biți
1 word = 2 bytes = 16 biți
1 long = 4 bytes = 32 biți
1 quad = 8 bytes = 64 biți
1 asciiz = (nr. de caractere) bytes = nr * 8 biți
1 asciz = [(nr. caractere) + 1] bytes = nr * 8 + 8 biți
space x = x bytes = x · 8 biți

- Îm baza 2 / binar => 1 cifră = 1 bit
- Îm baza 16 / hexa => 1 cifră = 4 biți

```
mov $4, %eax   => WRITE (citire, specială)
mov $1, %ebx   => ARGUMENTUL PE CARE O SĂ-L AFIȘEM
```

se fol. doar pt. sfârșituri
Dacă sunt bytes, long etc => se vor transforma în ascii

```
mov $1, %eax
mov $0, %ebx   => return 0 ; întreruperea funcției
int $0x80
```

În terminal:

- b main (break point la adr. intrare în program)
- run (rulăm programul)
- ir (afișează ce vrei; ex: ir %eax etc.)
- stepi (debug step by step; îți și arată schimbarea valorilor)

```
mov $4, %eax    => funcția write
mov $1, %ebx    => locul unde af. textul
mov str, %ecx   => stocăm mesajul
mov x, %edx     => lungimea mesajului/șirului alocată
```
=> DOAR STRING-URI

Operații aritmetice:

mul x => $eax = eax \cdot x$ (baza 10) & $edx = 0$

$eax = 2^{32} \cdot edx + eax$ (hexa)

imul x => $eax = eax \cdot x$ (baza 10)

$eax = 2^{32} \cdot edx + eax$ (hexa cu semn)

div x => $D = 2^{32} \cdot edx + eax$

$eax = D / x = C$ și R | => doar pt hexa edx = 0

$eax = eax / x = C$ și R pt baza 10

C = mod EAX

R = mod EDX

idiv x analog imul

Dacă nu se specifică la început val %edx => %edx = 0

Operații de shiftare logice:

shr nr, x => x = x >> nr

ex: 1101 >> 1 = 110

$2^3 + 2^5 >> 2 = 2 + 2^3 =$

se împarte

| $m >> k => \frac{m}{2^k}$

shl nr, x => x = x << nr

shl $2, %eax

unde eax = 0011 = 3

se înmulțește

0011 << 2 = 1100

| => $eax = 3 \cdot 2^2 = 12$

$ex = eax \cdot 2^{nr}$

$nr >> k => nr \cdot 2^k$

sar nr, x
sal nr, x
| => păstrează semnul

---

cmp x, y
j(cond) label
| => se iau împreună y (cond) x = Adv => jump label

---

loop label => for (i=%ecx, i>0, i--) ; cât timp %ecx != 0 se execută label-ul

Exerciții

①

movl $0, %eax => 0000 0000 0000 0000 | 0000 0000 | 0000 0000
AX | AH | AL

movb $4, %ah => 0000 0000 0000 0000 | 0000 0100 | 0000 0000
AX | AH | AL

movb $2, %al => 0000 0000 0000 0000 | 0000 0100 | 0000 0010
AX | AH | AL

eax = ? = $2^1 + 2^{10}$ = 2 + 1024 = 1026

②

x = 0x04030201
y = 0x08040605

Pt. că 1 cifră hexa = 4 biți

mov x, %eax => 0403 | 02 | 01
AX | AH | AL

mov y, %ah => 0403 | 05 | 01
AX | AH | AL

- se va suprascrie
- y mai are loc tot în ah, deci se vor scrie primii 8 biți din y de la dr. la stg

③

x: .word 1
y: .word 2
mov x, %eax => 0002 | 00 | 01 în hexa
AX | AH | AL

Explicație: word are 16 biți
eax e pe 32 biți
când stocăm pe x
=> rămân liberi 16 biți unde va fi pus automat y, deoarece ele în memorie sunt stocate unul după altul.

④

x: .word 1
y: .word 0
z: .word 2
mov x, %eax => y x
0000 | 00 | 01 în hexa => eax = 1
AX | AH | AL

2 mai are loc în eax

⑤

str1 = "abc"
str2 = "123"
=> în memorie: ... abc123 ...
movl $str1, %ecx
movl $5, %edx
=> abc__
1 2 3 4 5
=> se va afișa primii 5 = abc12

⑥ str = "1234"
x = .byte 97 |=> în memorie "1234" 97

movl $str, %eax
movl $5, %edx |=> 1 2 3 4 a (ascii)

Pt că suntem cu fct. de string, orice nr/val care nu e string va fi convertit în ascii

⑦ Crescător?

Pt long, byte, single, word, quad nu contează val. de după
Pt. space cont doar val
Pt. ascii cont nr. de carac
asciz cont nr de carac +1

.long 10 = 4 bytes (2)
.byte 50 = 1 byte (1)
.asciz "Șirul de caractere: Dm." = 21 bytes (4)
.space 20 = 20 bytes (3)

Similar pt: .byte 'z' = 1 byte (1)
.space 10 = 10 bytes (3)
.asciz "1234567890" = 11 bytes (4)
.long 2 = 4 bytes (2)

⑧ val max pt DX = ?
DX are maxim lungimea 16 biți
Deci cel mai mare nr pe 16 biți în baza 2 este 11...1111 1111 = $2^0+2^1 \dots 2^7 \dots 2^{15} = 2^{16}-1$

Similar pt: val. max pt x: .word m
word pe 16 biți |=> $\underbrace{11\dots111}_{16} = 2^{16}-1$

x: .byte m
byte = 8 biți |=> $2^8-1=255$

⑨ Poate ce? reține ulterior
x: .space 100
Pt a vedea cât și ce poate încăpea, se împarte nr la nr. de bytes al altui var
în x se poate reține: 25 long-uri (1 long = 4 bytes); 50 word-uri (1 word = 2 bytes)

⑩ ecx = 553 = 1000 101001

0000 0000 0000 0000 | 0000 0010 | 0010 1001
CX — CH — CL

ch = 0010 = $2^1 = 2$

553 : 2 = 276 r 1
276 : 2 = 138 r 0
138 : 2 = 69 r 0
69 : 2 = 34 r 1
34 : 2 = 17 r 0
17 : 2 = 8 r 1
8 : 2 = 4 r 0
4 : 2 = 2 r 0
2 : 1 = 1 r 0
1 : 2 = 0 r 1

(11)

```
mov $1, %edx
mov $0, %eax
movl $0xffffffff, %ebx
divl %ebx
mov %eax, %ecx
et_loop:
add %ecx, s
loop et_loop
```

s = 0
edx = 1
eax = 0
ebx = 0xffffffff = 1111 ..... 1111 = $2^{32}-1$ max. neg.
eax = $[2^{32} \cdot edx + eax] / ebx = [2^{32} + 2^{32} - 1] / 2^{32} - 1 = 1 \text{ rest } 1$
eax = 1
edx = 1
ecx = 1
s = 0 + 1 = 1
ecx = ecx - 1 = 0 => gata
deci s = 1

(12)

eax = 0x4000 0000 = 0100 0000 .... 0000 = 2^{30}
ebx = 0x8 = 8 = 2^3
ecx = 0x1 = 1 = 2^0
edx = 0x8 = 8 = 2^3

mul %edx => eax = eax · edx = $2^{30} \cdot 2^3 = 2^{33} = 2 \cdot 2^{32}$ => $\begin{cases} EAX = 0 \\ EDX = 2 \end{cases}$

(13)

eax = 0x80000000 = 1000 0000 .... 0000 = 2^{31} sau -2^{31}
ebx = 0x8 = 8 = 2^3
edx = 0x4 = 4 = 2^2

mul %ebx => eax = ~~eax · ebx = $2^{32} \cdot 2^3 = 2^{35}$~~ = $2^{32} \cdot edx + eax$
imul %ebx => (daca era axa) ~~eax = eax · ebx = $2^{32} \cdot 2^3 = -2^{35}$~~ Se fol. -2^{32}
eax = $2^{32} \cdot 2^2 + 2^{31} = 2^{34} + 2^{31} = 2^{31}(2^3 + 1) =$