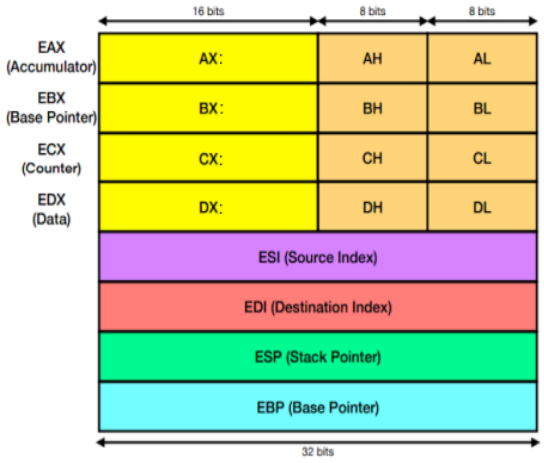
Ce valoare va retine EAX dupa executarea urmatoarei secvente de instructiuni? * Un punct

movl $0, %eax
movb $4, %ah
movb $2, %al



- [x] 1026
- [ ] 258
- [ ] 513
- [ ] 0

Explicație: Considerăm următoarea reprezentare a lui EAX:

		AH	AL

movl $0, %eax :

		AH	AL
00000000	00000000	00000000	00000000

movl $4, %ah :

		AH	AL
00000000	00000000	00000100	00000000

Movl $2, %al :

		AH	AL
00000000	00000000	00000100	00000010

La final, vom avea în EAX valoarea 00000000000000000000010000000010, adică 2^1+2^10, deci 1026.

Ordonati crescator in functie de spatiul ocupat in memorie urmatoarele declaratii * Un punct

```
a: .long 10
b: .byte 50
c: .asciz "Sirul de caractere:\n"
d: .space 20
```

- [ ] a,b,c,d
- [ ] a,b,d,c
- [x] b,a,d,c
- [ ] b,a,c,d

a: variabila a, de tip long, cu valoarea 10: 4 bytes

b: variabila b, de tip byte, cu valoarea 50 : 1 byte

c: câte un byte pe caracter, iar \n se consideră un caracter. Atenție! asciz are un '\0' la final, indifferent de conținutul șirului, deci șirul va avea încă un caracter : 21

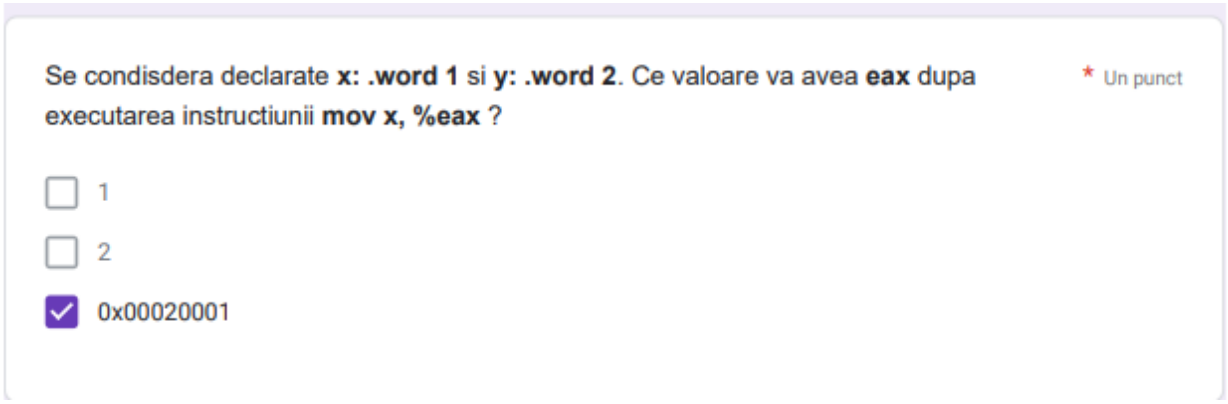
d: spațiu alocat de 20 bytes

ordinea: b, a, d, c

Care este valoarea maxima pe care o poate lua n in urmatoarea declaratie **x: .word n** ? * Un punct

- [ ] 255
- [ ] 256
- [x] 2^16 - 1
- [ ] 2^16

Din moment ce un word ocupă 2 bytes, valoarea maximă posibilă se obține dacă toți biții din reprezentare sunt egali cu 1, adică 2^(2*8)-1=2^16 - 1. (un word ocupă 16 biți în memorie)



		AH	AL
00000000	00000010	00000000	00000001

În hexa: 0x00020001

- Întâi se pune x în EAX (Valoarea se pune de la dreapta la stânga), dar acesta ocupă doar 2 bytes (adică doar al și ah vor fi ocupați cu valoarea lui x - este word), ceilalți 2 bytes ai registrului rămânând neocupați.
- Pentru că instrucțiunea mov (fără sufix) trebuie să pună 4 bytes în EAX, acesta va lua următorii 2 bytes din memorie, corespunzători lui y.
- Astfel, valoarea din EAX este obținută prin plasarea, pe rând, a celor două variabile: primii 2 bytes sunt ocupați de x, ultimii 2 bytes, de y(de la dreapta la stânga).
- Așadar, valoarea finală este 0x00020001.

Fie urmatoarea declarare in sectiunea .data: *

2 puncte

mySpace: .space 100

Acest spatiu poate fi utilizat pentru a retine ulterior:

- [ ] un array de 30 de word-uri
- [x] un array de 25 de long-uri
- [ ] un array de 50 de bytes
- [x] un array de 50 de word-uri

Se alocă 100 de bytes. Aceștia pot fi folosiți pentru 100/4=25 long-uri(un long ocupă 4 bytes în memorie) sau pentru 50 de word-uri(un word ocupă 2 bytes în memorie)

Fie urmatoarea declarare in sectiunea **.data:** *

Un punct

```
str1: .ascii "abc"
str2: .ascii "1"
```

Ce se va afisa in urma apelului WRITE urmator?

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

- [ ] abc
- [x] abc1
- [ ] nimic
- [ ] abc + o valoare reziduala

- În apelul WRITE, valoarea pusă în EDX este lungimea stringului care va fi afișat, a cărui adresă este încărcată în ecx.
- Deoarece str1 și str2 sunt de tip ascii, acestea nu au '\0' la final.
- Deci, în ECX se va pune un string de lungime 3.
- Totuși, imediat după str1 este declarat str2, un string de lungime 1, exact cât este necesar pentru a ocupa lungimea pusă în EDX, așadar și str2 va fi plasat imediat după str1, rezultând abc1.

Fie urmatoarea declarare in sectiunea **.data** * Un punct

```
str1: .ascii "abc"
str2: .ascii "123"
```

Ce se va afisa in urma apelului WRITE urmator?

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

- [ ] abc
- [x] abc12
- [ ] nimic
- [ ] abc + o valoare reziduala

Asemănător, de data aceasta lungimea fiind 5, se va lua str1, apoi primele două caractere din str2, afișându-se abc12.

In apelul sistem WRITE, sirul este incarcat in %ecx cu simbolul $. De exemplu, pentru * Un punct

```
str: .asciz "Sir"
```

incarcarea in %ecx se va face cu $str. Care este scopul acestui simbol?

- [ ] semnifica faptul ca sirul este constant
- [ ] respecta conventia WRITE, unde toate argumentele trebuie precedate de $
- [x] semnifica preluarea adresei din memorie pentru str
- [ ] nu are nicio semnificatie, se poate utiliza si scrierea str pentru a obtine exact acelasi rezultat

(vezi suportul de laborator, pag. 17)

Fie urmatorul program: * 2 puncte

```
.data
x: .long 0x04030201
y: .long 0x08070605

.text
.global main
main:

mov x, %eax
mov y, %ah

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

Acesta se compileaza si executabilul se ruleaza folosind gdb. In gdb se vor da urmatoarele comenzi:

```
b main
run
stepi
stepi
```

Ce valoare va afisata in **%eax** in urma rularii comenzi **i r**?

- [ ] 0x04030201
- [ ] 0x08070605
- [ ] 1
- [x] 0x04030501
- [ ] 0x04030205

		AH	AL
00000100	00000011	00000010	00000001

Apoi, se vor pune primii 8 biți din y din ah:

		AH	AL

00000100	00000011	00000101	00000001

Așadar, valoarea obținută este 0x04030501.

Fie urmatorul program: * 2 puncte

```
.data
x: .long 0x04030201
y: .long 0x08070605

.text
.global main
main:

mov x, %eax
mov y, %al

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

Acesta se compileaza si executabilul se ruleaza folosind gdb. In gdb se vor da urmatoarele comenzi:

```
b main
run
stepi
stepi
```

Ce valoare va afisata in **%eax** in urma rularii comenzi **i r**?

- [ ] 0x04030201
- [ ] 0x08070605
- [ ] 1
- [ ] 0x04030501
- [x] 0x04030205

Asemănător, de data aceasta primii 8 biți ai lui y se vor pune în al, obținându-se 0x04030205

Ordonati crescator in functie de spatiul ocupat in memorie urmatoarele declaratii * Un punct

```
a: .long 4
b: .asciz "hi!\n"
c: .byte 50
d: .space 50
```

- [ ] c,d,b,a
- [x] c,a,b,d
- [ ] b,a,d,c
- [ ] b,a,c,d

a: variabila numită a, de tip long, cu valoarea 4: 4 bytes

b: variabila numită b, de tip asciz, având 4 caractere: 5 bytes(cu tot cu \n care se pune la final)

c: variabila numită c, de tip byte, cu valoarea 50: 1 byte

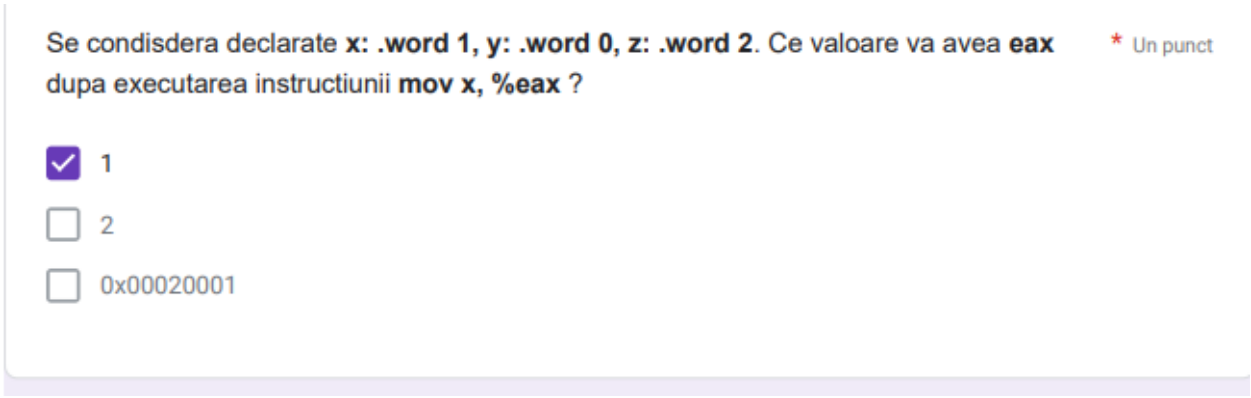
d: variabila numită d, care alocă un spațiu de 50 de bytes

Ordinea este: c, a, b, d

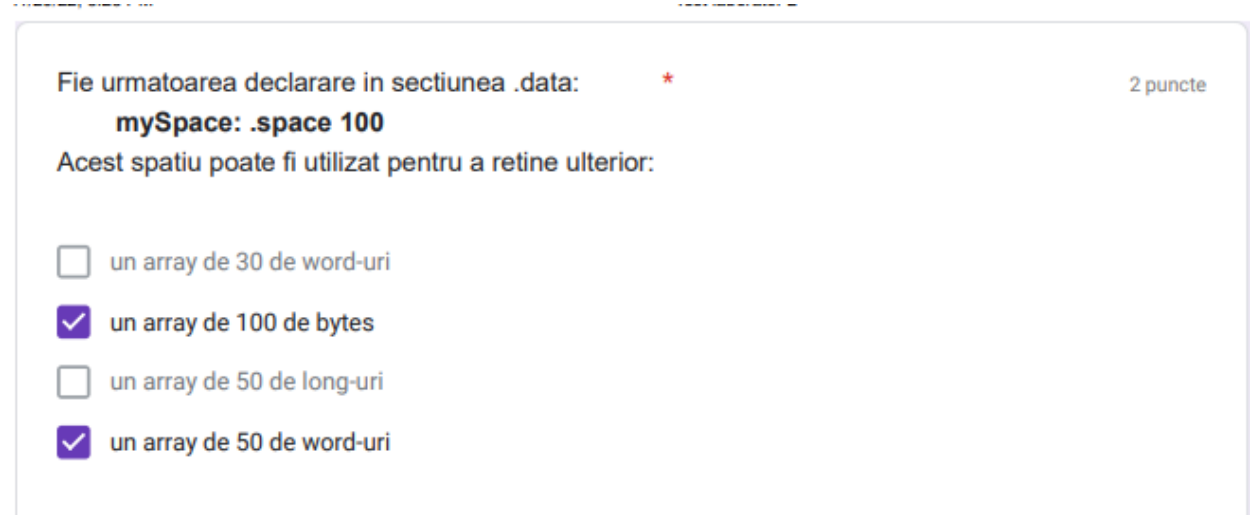
Care este valoarea maxima pe care o poate lua n in urmatoarea declaratie **x: .byte n** ? * Un punct

- [x] 255
- [ ] 256
- [ ] 2^16 - 1
- [ ] 2^16

Un byte ocupă 8 biți în memorie, deci valoarea maximă se obține dacă toți biții sunt egali cu 1, adică 2^8-1=255.



- Întâi se vor pune 2 bytes în EAX, corespunzători valorii lui x
- Următoarea variabilă declarată este tot un word, de data aceasta cu valoarea 0, care se va pune după x în registru.
- Așadar, se va obține, în hexa, valoarea 0x00000001, adică 1.

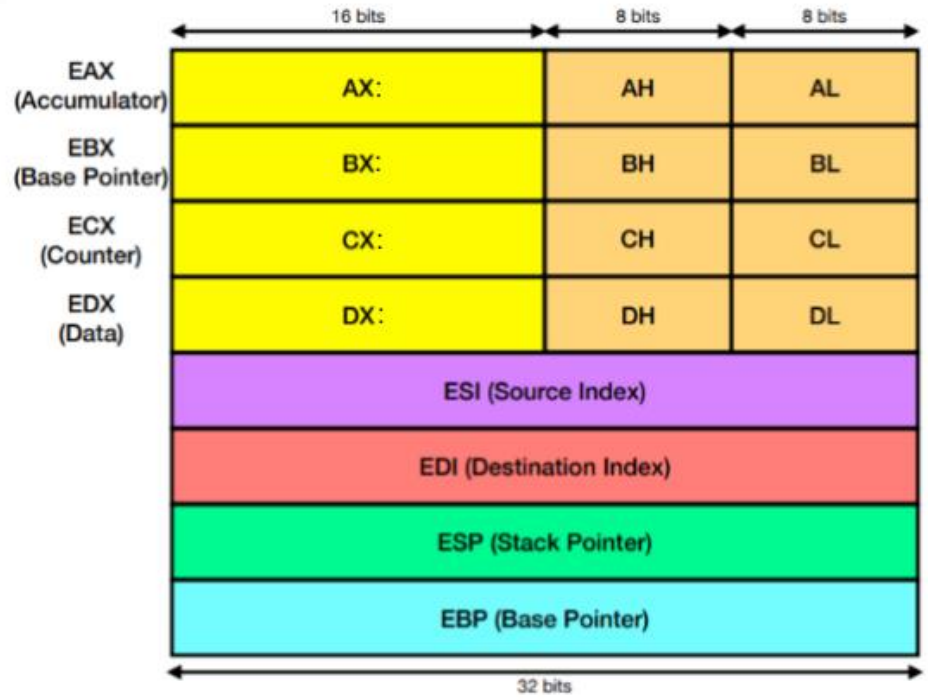


Se alocă 100 de bytes, care pot fi folosiți pentru 100 de variabile de tip byte sau de 50 de variabile de tip word, care ocupă câte 2 bytes în memorie.

Ce valoare va retine registrul CH dupa executarea urmatoarei instructiuni? *

Un punct

movl $553, %ecx



- [ ] 10
- [x] 2
- [ ] 512
- [ ] 0

		CH	CL
00000000	00000000	00000010	00101001

553=0x229

În CH, valoarea este 2.

Ordonati crescator, in functie de spatiul ocupat in memorie, urmatoarele declaratii: * Un punct

ch: .byte 'z'
v: .space 10
str: .asciz "1234567890"
y: .long 2

- [ ] y, v, str, ch
- [ ] ch, y, str, v
- [ ] y, v, ch, str
- [x] ch, y, v, str

ch: variabilă numită ch, de tip byte, cu valoarea ‘z’ (cod ascii)-1 byte

v: spațiu pentru 10 bytes

str: este asciz, deci va avea numărul de caractere+1 bytes, adică 11 bytes

y: variabila numită y, de tip long, cu valoarea 2, adică 4 bytes

ordinea: ch, y, v, str

Care este valoarea maxima pe care o poate stoca registrul DX? * Un punct

- [x] 2^16 - 1
- [ ] 2^16
- [ ] 255
- [ ] 256

Conform desenului anterior cu organizarea regiștrilor, DX are 16 biți(2 octeți), deci valoarea maximă se obține când toți acești biți au valoarea 1: 2^16-1.

In instructiunea pentru intreruperea hardware, **int $0x80**, parametrul 0x80 este prefixat de simbolul $. Care este semnificatia acestui simbol in contextul curent? * Un punct

- [ ] reprezinta o conventie de implementare a intreruperii hardware;
- [x] simbolul $ este utilizat pentru prefixarea unei constante;
- [ ] semnifica preluarea adresei din memorie la care se regaseste codul de intrerupere 0x80;
- [ ] nu are nicio semnificatie, se poate utiliza si scrierea int 0x80, cu acelasi rezultat.

$ este folosit atât pentru preluarea unei adrese din memorie, cât și pentru prefixarea unei constante.

Care dintre urmatoarele siruri de instructiuni efectueaza o interschimbare corecta a valorilor din variabilele de tipul .long, x si y? * Un punct

- [ ] movl %eax, x; movl y, x; movl y, %eax;
- [ ] movl x, %eax; movl y, %ebx; movl %eax, %ebx; movl %ebx, x; movl %eax, y;
- [ ] movl x, %ecx; movl y, %edx; movl %ecx, %eax; movl %edx, %ecx; movl %eax, %edx; movl %edx, x; movl %ecx, y;
- [ ] movl x, %ecx; movl y, %edx; movl %ecx, %eax; movl %edx, %ecx; movl %edx, %eax;

Pentru simplitate, vom da valori: eax=a, ebx=b, ecx=c, edx=d, x=e, y=f.

Prima variantă:

x <- a, x <- f, eax <- f (nicio valoare nu a fost interschimbată)

A doua:

eax <- e, ebx <- f, ebx <- e, x <- e, eax <-y (nicio valoare nu a fost interschimbată)

A treia:

ecx <- e, edx <- f, eax <- e, ecx <- f, edx <- e, x <- e, y <- f(nicio valoare nu a fost interschimbată)

A patra:

ecx <- e, edx <- f, eax <-e, ecx <- f, eax <- f(nicio valoare nu a fost interschimbată)

Așadar, nicio variantă nu efectuează o interschimbare a variabilelor x și y.

Fie urmatoarea declarare in sectiunea **.data**: *

Un punct

```
str: .ascii "1234"
x: .byte 97
```

Ce se va afisa in urma apelului WRITE urmator?

```
movl $4, %eax
movl $1, %ebx
movl $str, %ecx
movl $5, %edx
```

- ○ 1234
- ◉ 1234a
- ○ 12349
- ○ 123497
- ○ 1234a9
- ○ 1234a97

Pentru că un caracter are un byte, după ce se va scrie str, va mai fi nevoie de un caracter, deci se va scrie caracterul corespunzător codului ascii 97, adică a.

Fie urmatorul program. Precizati secventa corecta de instructiuni in debugger, in urma careia vom obtine valoarea 8. * Un punct

```
.data
.text
.global main
main:
  movl $8, %eax
  movl $2048, %ecx
et_exit:
  movl $1, %eax
  movl $0, %ebx
  int $0x80
```

- ○ b main; run; stepi; stepi; i r cl
- ◉ b main; run; stepi; stepi; i r ch
- ○ b main; run; i r eax
- ○ b main; run; stepi; i r ah

Se va pune breakpoint pe main, apoi se va da run. Dacă verificăm regiștrii, vor avea valori reziduale. La primul stepi, eax va avea valoarea 8.

		AH	AL
00000000	00000000	00000000	00001000

Dacă verificăm ah, obținem 0.

La al doilea stepi, ecx va avea valoarea 2048.

		CH	CL
00000000	00000000	00001000	00000000

Verificând cl, obținem 0, iar, verificând ch, obținem 8. Așadar, varianta corectă este a doua.

Fie urmatoarea declaratie in sectiunea **.data**: * 2 puncte

v: .space 120

Acest spatiu se poate utiliza pentru a retine ulterior:

- [x] un array de 60 de word-uri;
- [ ] un array de 60 de long-uri;
- [x] un array de 30 de long-uri;

Spațiul alocat este de 120 bytes, ce poate fi folosit pentru 60 de word-uri(a câte 2 bytes) sau 30 de long-uri(a câte 4 bytes).

Fie codul de mai jos. Care este valorea lui s cand executia ajunge la et_exit? * 2 puncte

```
.data
s: .long 0

.text
.global main
main:
mov $1, %edx
mov $0, %eax
movl $0xffffffff, %ebx
divl %ebx
mov %eax, %ecx
et_loop:
add %ecx, s
loop et_loop

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

- [x] 1
- [ ] 0
- [ ] 0xffffffff
- [ ] 15

EDX	EAX
00000001	00000000

- La împărțire, regiștrii edx și eax sunt alăturați ca în figura de mai sus. În (edx, eax) vom avea valoarea 2^32.
- Dată fiind reprezentarea în complement față de 2, ne-am gândi că 0xffffffff este -1.
- Însă operația div este definită pe numere naturale (pentru numere întregi se folosește operația idiv), deci 0xffffffff va fi văzut ca 2^32 , iar câtul împărțirii lui 2^32 la 2^32-1 este 1.
- Așadar, în eax vom avea 1, valoare pe care o vom pune în ecx.
- Ecx va avea valoarea 1, iar, la fiecare loop, se adaugă valoarea lui ecx la s, apoi ecx se decrementează (loop se execute cât timp ecx e diferit de 0).
- Așadar, s=1 la final.

Se stocheaza in registrul eax valoarea 0x40000000, in ebx 0x8, in ecx 0x1 si in edx 0x8. * Un punct
Ce valori vor avea registrii eax si edx dupa executarea instructiunii **mul %edx**?

- [x] eax = 0, edx = 2
- [ ] eax = 32, edx = 0
- [ ] eax = 32, edx = 2
- [ ] eax = 2, edx = 0

eax va avea valoarea 2^30. Înmulțind cu 8, adică 2^3, se obține 2^33, adică

EDX	EAX
00000002	00000000

Deci, în edx se află 2, iar în eax, 1.

Se stocheaza in %edx valoarea 0, in eax 47 si in ebx 15. Ce valori vor avea registrii eax si edx dupa executarea instructiunii **div %ebx** ? * 2 puncte

- [x] eax = 3, edx = 2
- [ ] eax = 3, edx = 0
- [ ] eax = 2, edx = 0
- [ ] eax = 2, edx = 3

47/15=3 rest 2, câtul se duce în eax, iar restul, în edx.

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

```
.data
x: .long 0x80000000
y: .long 0x70000000
.text
.global 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
```

- [x] 5
- [ ] 6

X: 10000000000000000000000000000000

Y: 01110000000000000000000000000000

Bitul de semn este 1 în cazul lui x, deci numărul este negativ. Așadar, nu inră în eticheta label. Astfel, în ecx se află valoarea 5 la final.

Fie urmatorul program. Ce valoare vom obtine daca vom rula cu debuggerul urmatoarele comenzi? * 2 puncte

b et_exit
run
i r ebx

```
.data
.text
.global main
main:
mov $3, %eax
shl $2, %eax
mov $2, %ebx
mul %ebx
mov $0, %edx
mov $8, %ebx
div %ebx
sub %eax, %ebx

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

- [x] 5
- [ ] 2
- [ ] 3
- [ ] 8

Eax <- 3

Eax <- 3* 2^2=12

Ebx <- 2

Eax <- 12*2=24

Ebx <- 8

Eax <- 24/8=3

Ebx <-8-3=5

Așadar, la final, în ebx va fi valoarea 5.

Care este ordinea de trecere prin etichete? *

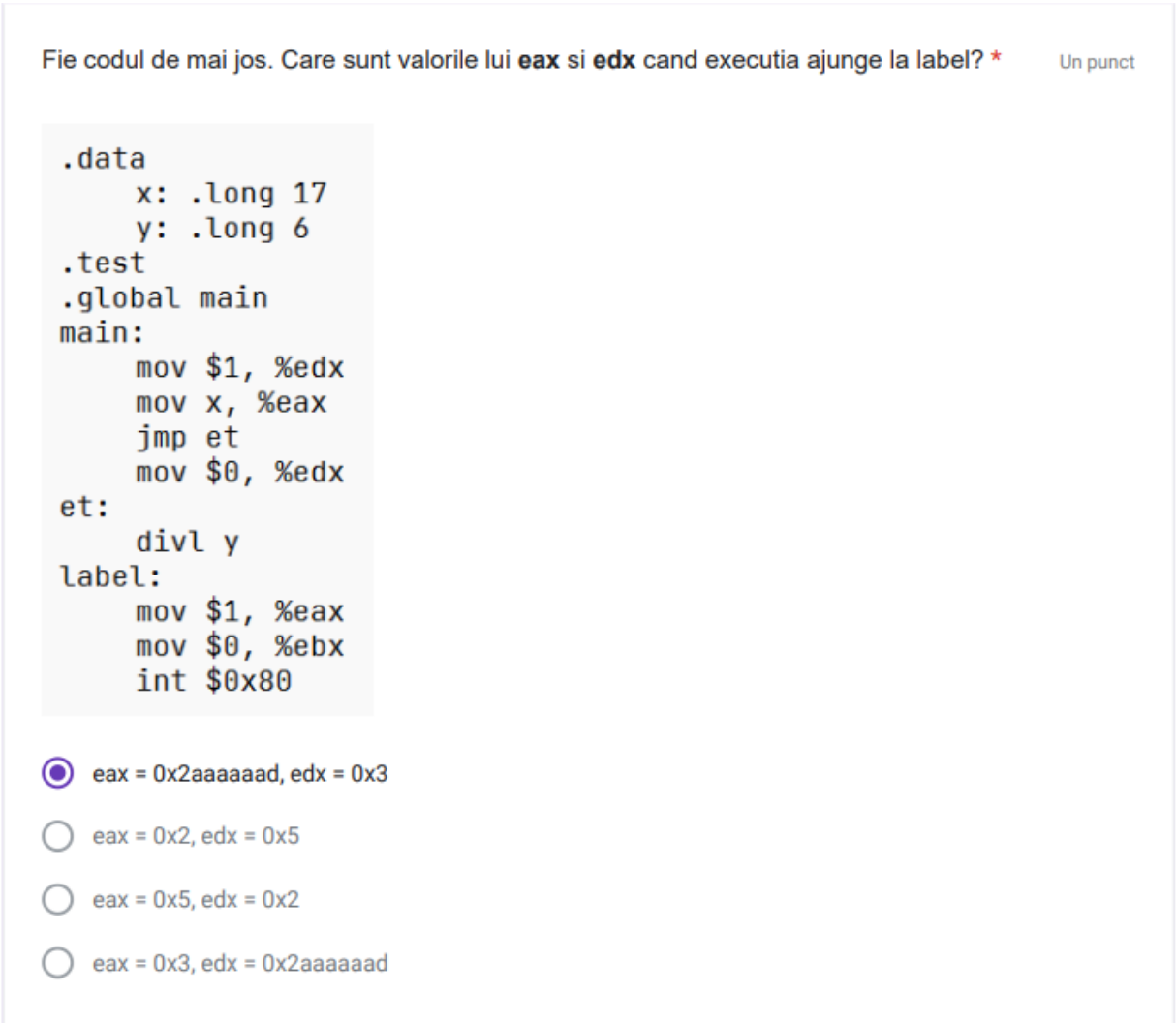
2 puncte

```
.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
```

- [x] ety, etz, ety, ett
- [ ] etx, ety, etz, ety, ett
- [ ] etx, ety, ett
- [ ] ety, etx, etz, ett

- Deoarece ebx și eax nu au valori egale, nu se va intra în etx, ci în ce urmează imediat după, adică ety.
- Mai departe, se va compara ecx cu edx, iar, cel din urmă fiind mai mare, se va efectua saltul către etz.

- Apoi, următorul salt efectuat va fi spre ty, iar, cum de data aceasta edx nu mai este mai mare decât ecx(edx=2, ecx=3), se va efectua saltul spre ett.



Edx <- 1

Eax <- 17

(edx, eax)=2^32+17

eax<-(2^32+17)/6=0x2aaaaaad

edx<-0x3

Se stocheaza in EAX valoarea 0x80000000, in EBX 0x8, in ECX 0x1 si in EDX 0x4. Ce valori vor avea registrii EAX, respectiv EDX dupa executarea instructiunii **mul %ebx**? * Un punct

- (●) eax = 0, edx = 4
- ( ) eax = 32, edx = 0
- ( ) eax = 32, edx = 4
- ( ) eax = 4, edx = 0

2^3*2^31=2^34

EDX	EAX
00000000000000000000000000000100	00000000000000000000000000000000

Edx=4, eax=0

Se stocheaza in %edx valoarea 0, in eax 37 si in ebx 15. Ce valori vor avea registrii eax si edx dupa executarea instructiunii **div %ebx** ? * 2 puncte

- ( ) eax = 7, edx = 2
- ( ) eax = 7, edx = 0
- ( ) eax = 2, edx = 0
- (●) eax = 2, edx = 7

37/15=2 rest 7, deci eax=2, edx=7.

Fie codul de mai jos. Care este valoarea depozitata in **z** cand ajungem la eticheta **final**? * 2 puncte

```
.data
x: .long 17
y: .long 6
x1: .long 5
y1: .long 9
z: .space 4
.text
.global main
main:
    mov x, %eax
    mov y, %ebx

    cmp %eax, %ebx
    jge et1

    mov x1, %eax
    cmp %eax, %ebx
    jle et

    mov x, %ebx
    sub %eax, %ebx
    jmp final
et:
    add %eax, %ebx
    jmp final

et1:
    mov x1, %eax
    mov y1, %ebx
    cmp %eax, %ebx
    jge et2
    add %eax, %ebx
    jmp final

et2:
    sub %eax, %ebx

final:
    mov %ebx, z

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

- ○ 4
- ○ 1
- ◉ 12
- ○ 23

Eax<-17

Ebx<-6

Eax<-5

Ebx<-17

Ebx<-12

Z<-ebx=12, dei răspunsul este 12

Care este ordinea de trecere prin etichete? * 2 puncte

```
.data
.text
.global main
main:
    jmp etb
eto:
    jmp etd
eth:
    jmp eto
etb:
    jmp eth
etd:
    mov $1, %eax
    mov $0, %ebx
    int $0x80
```

- (•) etb, eth, eto, etd
- ( ) eth, etb, etd, eto
- ( ) eto, eth, etb,etd
- ( ) etb, eto, eth, etd

Se efectuează toate salturile: începând de la main, se ajunge la etb, apoi la eth, apoi la eto, urmată de etd.

De cate ori se va executa instructiunea **loop**? *

Un punct

```
.data
x: .long 5
y: .long 5
s: .long 0
.text
.global main
main:
    mov x, %ecx
    sub y, %ecx

et:
    add %ecx, s
    loop et
exit:
    mov $1, %eax
    mov $0, %ebx
    int $0x80
```

- ○ 0x1
- ○ 0x0
- ○ 0x5
- ◉ 0xffffffff
- ○ este un ciclu infinit

- Întâi se decrementează ecx, apoi se execută instrucțiunea loop, iar la final se verifică dacă ecx e 0.
- Întrucât verificarea este efectuată după decrementare, iar valoarea lui ecx scade mereu cu 1, la un moment dat se va ajunge la underflow, motiv pentru care ecx va avea din nou valori pozitive.
- Astfel, după mulți pași (0xffffffff pași, mai exact), vom ajunge înapoi la 0, moment în care se va opri eticheta de loop.

Fie urmatorul program. Ce valoare vom obtine daca vom rula cu debuggerul urmatoarele comenzi? * Un punct

b et_exit
run
i r edx

```
.data
.text
.global main
main:
     mov $2, %eax
     mov $3, %ebx
     add %eax, %ebx
     mul %ebx
     mov $0, %edx
     divl $3
     add %eax, %edx
et_exit:
     mov $1, %eax
     mov $0, %ebx
     int $0x80
```

- ○ 0x1
- ◉ 0x4
- ○ 0x0
- ○ 0x3

Eax <- 2

Ebx <- 3

Ebx <- 3+2=5

Eax <- 2*5=10

Eax <- 10/3=3

Edx <- 10 mod 3=1

Edx <- 3+1=4

Așadar, edx are valoarea 4, sau 0x4 în hexa.

Precizare: Memoria este continuă, deci datele sunt așezate unele după altele în memorie, în ordinea declarării (de la dreapta la stânga). Este important în special pentru rezolvarea problemelor cu vectori!

Ce valori vor fi depozitate in v cand executia va ajunge in dreptul etichetei **et_exit** ? * Un punct

```
.data
  v: .space 20
  n: .long 5
.text
.global main
main:
  lea v, %edi
  mov $11, %edx
  mov $0, %ecx
et_loop:
  cmp n, %ecx
  jg et_exit
  mov %edx, (%edi, %ecx, 4)
  inc %ecx
  inc %edx
  jmp et_loop
et_exit:
  mov $1, %eax
  xor %ebx, %ebx
  int $0x80
```

- ○ 11, 12, 13, 14, 15, 16
- ○ 11, 12, 13, 14, 15
- ◉ 11, 12, ... , 27
- ○ 0, 1, 2, 3, 4, 5
- ○ Executia nu ajunge la et_exit, loop-ul este infinit.

V se pune în edi, iar în edx se pune 11, ecx este indexul elementului curent în v. Cât timp indexul curent este mai mic decât sau egal cu n, se pune valoarea conținută în edx în vector, la indexul curent, apoi se incrementează atât ecx, cât și edx. În vector se pun elemente de tip long, deci avem spațiu pentru 20/4=5 elemente. Se completează vectorul până la v[4], epuizându-se spațiul alocat pentru v, dar condiția de oprire este ca indexul curent să fie mai mare strict decât ecx, deci se pune 16 (corespunzătoare lui v[5]) la adresa imediat următoare (în ordinea declarărilor, aceea a lui n). Astfel, este suprascrisă valoarea lui n cu valoarea lui v[5], adică 16, iar bucla se încheie atunci când ecx ajunge egal cu 16.

Ce se va afisa pe ecran? *

2 puncte

```
.data
  x:  .long 1, 3, 6, 7, 9
  n1: .long 5
  n2: .long 10
  c:  .long 0x64636261
  s:  .space 11
.text
.global main
main:
  mov $s, %edi
  mov $x, %esi
  movb c, %al
  mov $0, %ecx

et_loop1:
  cmp n2, %ecx
  je et_exit_loop1
  mov %al, (%edi, %ecx, 1)
  inc %ecx
  jmp et_loop1

et_exit_loop1:
  mov $c, %eax
  movb 1(%eax), %al
  mov $0, %ecx

et_loop2:
  cmp n1, %ecx
  je et_exit
  mov (%esi, %ecx, 4), %ebx
  mov %al, (%edi, %ebx, 1)
  inc %ecx
  jmp et_loop2

et_exit:
  mov $10, %ecx
  movb $0, (%edi, %ecx,1)
  mov $4, %eax
  mov $1, %ebx
  mov $s, %ecx
  mov $11, %edx
  int $0x80

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

- (●) ababaabbab
- ( ) 61, 61, 61, 61, 61, 61, 61, 61, 61, 61
- ( ) aaaaaaaaaa
- ( ) bbbbbaaaaa

Vectorul s se pune în edi, iar vectorul x, în esi. Se mută ultimii 16 biți din c în al(eax), adică în al va fi valoarea 61(în hexa). Indexul curent se află în ecx. Cât timp indexul curent este strict mai mic decât n2, se mută conținutul din al, adică un byte, în vector, la indexul curent. 61 în hexa este 97 în zecimal, valoare corespunzătoare codului ascii al lui ‘a’, așadar, la finalul execuției etichetei et_loop1, vectorul conține 10 de ‘a’. În et_exit_loop, se pune 0x64636261 în eax. 1(%eax) înseamnă %eax+1, deci în al se vor pune ultimii 16 biți din eax (62 în hexa, deci 98 în zecimal, codul ascii al lui ‘b’). ecx redevine 0. În et_loop2, cât timp ecx este strict mai mic decât n1, se mută valoarea de la indexul curent din ecx în ebx. Această valoare devine, la rândul ei, index curent în s, iar valoarea din al, adică ‘b’ se pune la poziția ebx în vectorul s. practic, et_loop_2 execută următoarele instrucțiuni:

Ebx <- x[ecx]

S[ebx] <- ‘b’

Ecx <- ecx+1

Așadar, valorile de pe pozițiile 1, 3, 6, 7, 9 vor fi schimbate în ‘b’, în rest vor rămâne ‘a’. În et_exit, pe poziția 10 din vectorul s se va pune 0(dar, dată fiind indexarea de la 0, nu afectează cu nimic conținutul vectorului) Se afișează prin apelul WRITE conținutul vectorului. Răspunsul este ababaabbab.

Ce se va afisa pe ecran? *

2 puncte

```
.data
  n: .long 3
  s: .asciz "abc"
.text
.global main
main:
  mov $s, %edi
  mov $0, %ecx
et_loop:
  cmp n, %ecx
  je et_exit
  mov (%edi, %ecx, 1), %al
  sub $'a', %al
  add $'A', %al
  mov %al, (%edi, %ecx, 1)
  inc %ecx
  jmp et_loop

et_exit:
  mov $4, %eax
  mov $1, %ebx
  mov $s, %ecx
  mov $4, %edx
  int $0x80

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

- ○ abc
- ○ Abc
- ◉ ABC
- ○ Abc + o valoarea reziduala

Vom itera prin fiecare element al lui s. Fiecare caracter este mutat în al, se scade din valoarea sa, codul ascii al lui ‘a’ (se obțin, astfel, pe rând, 0, 1, și 2), apoi se adaugă codul ascii al lui ‘A’ (se obține A, B, rescpectiv C). Practic, s-a realizat transformarea literelor mici în litere mari. În et_exit se efectuează afișarea lui s. Astfel, răspunsul este ABC.

Ce se va afisa pe ecran? *

Un punct

```
.data
  n: .long 3
  s: .byte 'a', 'b', 'c'
  t: .byte 'd', 'e', 'f'
  u: .space 4
.text
.global main
main:
  mov $0, %ecx
et_loop:
  cmp n, %ecx
  je et_exit
  mov $0, %edx
  sub %ecx, %edx
  mov t(, %edx, 1), %al
  mov %al, u(, %ecx, 1)
  inc %ecx
  jmp et_loop

et_exit:
  movb $0, u(, %ecx, 1)

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

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

- ○ def
- ○ abc
- ○ cba
- ◉ dcb

În memorie, datele sunt declarate astfel: n, s= ’a’, ’b’, ’c’, t= ’d’, ’e’, ’f’, u. Edx rămâne 0 pe tot parcursul execuției et_loop. Cât timp indexul curent (ecx) este mai mic decât n, se scade din valoarea sa edx. Elementul t[edx] se pune în al, iar al este pus în u[ecx]. Doar la primul pas, edx este pozitiv, adică u[0]=t[0], deci u[0]= ’d’. La următorul pas, ecx devine -1, dar indexul -1 nu există, așa că în u[1] se va

pune ultima valoare de dinaintea declarării lui t[0], adică s[2] ș.a.m.d. (vezi precizarea de la pagina 26). În et_exit se afișează conținutul lui u. Prin urmare, răspunsul este dcb.

Fie urmatorul program. Ce valoare vom obtine daca vom rula cu debuggerul urmatoarele comenzi? * 2 puncte

b et_exit
run
i r eax

```
.data
  n: .long 4
  v: .long 0x01020304, 0x05060708, 0x090a0b0c, 0x0d0e0f10
.text
.global main
main:
  mov $v, %esi
  mov $1, %ecx
  mov (%esi, %ecx, 4), %eax
  add $4, %esi
  movb (%esi, %ecx, 4), %al
et_exit:
  mov $1, %eax
  xor %ebx, %ebx
  int $0x80
```

- (●) 0x0506070c
- ( ) 0x090a0b0c
- ( ) 0x05060704
- ( ) 0x090a0b08

Adresa este a lui v[1]. Se mută această valoare în eax. Eax<-0x05060708

Prin adăugarea lui 4 la esi, se trece la următoarea poziție din vector, aflată 4 bytes mai departe (v[2]). Apoi, trebuie mutați ultimii 2 bytes din v[2] în al (se schimbă primele două cifre de la dreapta din reprezentarea hexazecimală a lui eax). Așadar, 0x0506070c este răspunsul.

Fie urmatorul program. Ce valoare vom obtine daca vom rula cu debuggerul urmatoarele comenzi? * 2 puncte

b et_exit
run
i r eax

```
.data
  n: .long 4
  v: .long 0x01020304, 0x05060708, 0x090a0b0c, 0x0d0e0f10
.text
.global main
main:
  mov $v, %esi
  mov $2, %ecx
  mov -8(%esi, %ecx, 4), %eax
et_exit:
  mov $1, %eax
  xor %ebx, %ebx
  int $0x80
```

- (●) 0x01020304
- ( ) 0x090a0b0c
- ( ) 0x05060708
- ( ) Executia nu va ajunge la et_exit, o sa apara o eroare de accesare

Pentru ecx=0, (%esi, %ecx, 4)=v[0]. Dar, când ecx=2, -8(%esi, %ecx, 4)=%esi+%ecx*4-8, practic ne întoarcem la indexul 0. Așadar, răspunsul este v[0]=0x01020304 .

Ce se va afisa pe ecran? *

2 puncte

```
.data
  n: .long 9
  s: .asciz "a1C95dBx3"
  t: .space 10
.text
.global main
main:
  mov $s, %esi
  mov $t, %edi
  mov $0, %ecx
  mov $0, %edx
et_loop:
  cmp n, %ecx
  je et_exit
  mov (%esi, %ecx, 1), %al
  cmp $'0', %al
  jl et2
  cmp $'9', %al
  jg et2
  mov %al, (%edi, %edx, 1)
  inc %edx
  et2:
  inc %ecx
  jmp et_loop

et_exit:
  movb $0, (%edi, %edx, 1)
  inc %edx

  mov $4, %eax
  mov $1, %ebx
  mov $t, %ecx
  int $0x80

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

- ○ aCdBx
- ○ a1C95dBx3
- ◉ 1953

○ 1C95dB3

Se parcurge s. La fiecare pas, se ia câte un caracter din s, se pune în al și se verifică dacă este cifră (dacă are codul ascii corespunzător între codul lui ‘0’ și cel al lui ‘9’). Dacă nu este, se trece mai departe. Altfel, al se pune în t. Astfel, la final, t va conține caracterele din s care sunt cifre, adică 1953.

Ce valori vor fi stocate in **x** cand executia va ajunge la eticheta **et_exit**? *

2 puncte

```
.data
  x: .long 4, 2, 1, 5, 6
  n: .long 5
.text
.global main
main:
  mov $x, %esi
  mov $0, %eax
et_loop:
  cmp n, %eax
  je et_exit
  mov (%esi, %eax, 4), %ecx
  mov $1, %ebx
  sal %cl, %ebx
  mov %ebx, (%esi, %eax, 4)
  inc %eax
  jmp et_loop
et_exit:
  mov $1, %eax
  xor %ebx, %ebx
  int $0x80
```

- (●) 16, 4, 2, 32, 64
- ( ) 0, 0, 0, 0, 0
- ( ) 16, 4, 1, 25, 36
- ( ) 1,2,3,4,5

Se parcurge vectorul x(indexul curent este eax). Valoarea x[eax] este modificată astfel:

X[eax] <- 1<<x[eax] (adică 2^x[eax])

La final, vectorul va conține valorile 2^4, 2^2, 2^1, 2^5, 2^6, adică 16, 4, 2, 32, 64.

Fie urmatorul program. Ce valori se vor regasi in registrul **%ebx** la trecerea prin eticheta **et**? * Un punct

```
.data
  v: .long 15, 21, 30, 16, 18
  n: .long 4
.text
.global main
main:
  mov $n, %esi
  mov $0, %eax
  sub n, %eax
  mov $0, %ecx

et_loop:
  cmp %eax, %ecx
  je et_exit
  mov (%esi, %ecx, 4), %ebx
et:
  dec %ecx
  jmp et_loop

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

- (●) 4, 18, 16, 30
- ( ) 18, 16, 30, 21
- ( ) 15, 21, 30, 16
- ( ) 15, 21, 30, 16, 18

Fiindcă memoria este continuă, întâi se va reține valoarea lui n, deoarece aceasta este pusă în esi la indexul 0, apoi se iau, în ordine descrescătoare, ultimele n-1 valori memorate înainte de n, adică ultimele 3 elemente din v. Așadar, se obține 4, 18, 16, 30

Ce valori vor fi depozitate in v cand executia va ajunge in dreptul etichetei **et_exit** ? * 2 puncte

```
.data
  v: .space 24
  n: .long 5
.text
.global main
main:
  lea v, %edi
  mov $11, %edx
  mov $0, %ecx
et_loop:
  cmp n, %ecx
  jg et_exit
  mov %edx, (%edi, %ecx, 4)
  inc %ecx
  inc %edx
  jmp et_loop
et_exit:
  mov $1, %eax
  xor %ebx, %ebx
  int $0x80
```

- (●) 11, 12, 13, 14, 15, 16
- ( ) 11, 12, 13, 14, 15
- ( ) 0, 1, 2, 3, 4, 5
- ( ) 0, 1, 2, 3, 4, 5, 6
- ( ) Executia nu ajunge la et_exit, loop-ul este infinit.

La fiecare pas, în v, se pun valori consecutive începând de la 11, cât timp ecx este mai mic sau egal cu 5. La final, v va avea 6 astfel de valori, adică 11, 12, 13, 14, 15, 16.

Fie urmatorul program. Ce valoare va avea elementul din mijloc din vector daca vom rula cu debuggerul urmatoarele comenzi? * Un punct

b et_exit
run
x/3x &v

```
.data
  v: .long 0x01020304, 0x05060708, 0x090a0b0c
.text
.global main
main:
  mov $v, %esi
  mov $2, %ecx
  mov (%esi, %ecx, 1), %eax
  mov %eax, 4(%esi, %ecx, 1)
et_exit:
  mov $1, %eax
  xor %ebx, %ebx
  int $0x80
```

- (●) 0x01020708
- ( ) 0x05060708
- ( ) 0x090a0b0c
- ( ) 0x0506070c

Comanda din debugger ne afișează conținutul vectorului.

mov (%esi, %ecx, 1), %eax – se iau 4 bytes începând de la adresa esi+ecx*1=esi+2 (acolo arată pointerul, astfel, practic, arată spre ultimii 2 bytes ai lui v[0]: 0x0102 + cei doi bytes de după)

mov %eax, 4(%esi, %ecx, 1) - se pune conținutul lui eax la adresa esi+ecx*1+4=esi+6 (corespunzătoare ultimilor doi bytes ai lui v[1] : 0x0605 + cei doi bytes de după)

Astfel, v[1] este modificat, devenind 0x01020708

Ce valoare va fi stocata in **s** cand executia va ajunge la eticheta **et_exit**? *

2 puncte

```
.data
  v: .long 15, 21, 30, 16, 18, 12
  n: .long 6
  s: .long 0
.text
.global main
main:
  mov $v, %esi
  mov n, %eax
  shr $1, %eax
  mov $0, %ecx

et_loop:
  cmp %eax, %ecx
  jge et_exit
  mov 0(%esi), %ebx
  add %ebx, s
  add $8, %esi
  inc %ecx
  jmp et_loop

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

- (●) 63
- ( ) 23
- ( ) 129
- ( ) 66

Main: eax = 6/2^1, deci eax= 3.

Et_loop: Cât timp indexul curent este mai mic decât 3, se mută valoarea esi+0 în ebx, adică v[0]. Se adună valoarea la s, apoi se adaugă 8 la esi. Cum adunând 4 la esi putem trece la poziția următoare, instrucțiunea add $8, %esi semnifică avansarea cu două poziții în vector, iar procedeul se repetă de 3 ori în total. Astfel, la final, în s se va afla suma elementelor din v aflate pe poziții pare, deci 15+30+18=63.

Fie urmatorul program. Este acesta scris corect? Daca da, de ce, daca nu, de ce? 2 points

```
.data
    x1: .long 5
    x2: .long 12
    x3: .long 27
    n: .long 3
    s: .long 0
    formatPrintf: .asciz "%d\n"
.text

.global main

main:
    movl $x1, %edi
    movl $0, %ecx
et_loop:
    cmp n, %ecx
    je et_exit

    movl (%edi, %ecx, 4), %eax
    addl %eax, s

    incl %ecx
    jmp et_loop
et_exit:
    push s
    push $formatPrintf
    call printf
    pop %ebx
    pop %ebx

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

- [ ] nu este scris corect, deoarece foloseste x1 pe post de array, in timp ce x1 este doar o variabila
- [ ] nu este scris corect, deoarece n trebuia sa fie egal cu 1, intrucat array-ul x1 este un array doar cu un element
- [x] este scris corect, deoarece x1 e doar o adresa, ca numele unui vector
- [ ] este scris corect, dar nu va functiona conform asteptarilor - ruleaza, dar instructiunile nu au o logica specifica
- [ ] nu este scris corect, din cauza ca intram in zone de memorie in care nu avem drepturi

Fie codul urmator. Stiind ca **y** este un **.long 0** declarat in sectiunea **.data**, care dintre urmatoarele poate fi valoarea initiala din **x**, stiind ca in urma apelului **printf** se va afisa la STDOUT valoarea 1572?

2 points

```
main:
    movl $0, %ecx
    movl $10, %edi
et_while:
    cmp x, %ecx
    je et_exit

    movl x, %eax
    movl $0, %edx
    div %edi

    movl %eax, x
    movl %edx, %esi

    movl y, %eax
    mul %edi

    add %esi, %eax
    movl %eax, y
    jmp et_while

et_exit:
    push y
    push $formatStr
    call printf
    popl %ebx
    popl %ebx
```

- [ ] 1572
- [ ] 15720
- [x] 275100
- [ ] -1572

- [ ] 1024
- [ ] 216
- [ ] 83412
- [ ] 972250
- [x] 27510

Vectorul v se pune în %esi.

–8(%esi, %ecx, 4)= %esi + 4* %ecx + (-8)

Dar %ecx==2

Deci adresa este %esi + 4 * 2 –8 = %esi => v[0] se pune în %eax. Răspunsul este 0x01020304.