Compilație Probleme Laborator ASC
(x86 AT&T Syntax)

Cuprins

1 Test Lab 2.1 / 2.2 / 2.3

(vezi rezolvări)

Întrebarea 1

Ce valoare va reține EAX după executarea următoarei secvențe de instrucțiuni?

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

a) 513
b) 1026
c) 0
d) 258

Întrebarea 2

Se consideră declarate `x: .word 1` și `y: .word 2` (sau `x: .word 1, y: .word 0, z: .word 2`). Ce valoare va avea eax după executarea instrucțiunii `mov x, %eax`?

a) 1
b) 2
c) 0x00020001

Întrebarea 3

Fie următoarea declarare în secțiunea .data:

```
str1: .ascii "abc"
str2: .ascii "123" # (sau .ascii "1")
```

Ce se va afișa în urma apelului WRITE următor?

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

pentru testul cu "123" și edx=5

a) abc12
b) abc + *o valoare reziduală*
c) nimic
d) abc

pentru testul cu "1" și edx=4

a) nimic
b) abc1
c) abc
d) abc + *o valoare reziduală*

Întrebarea 4

Fie programul următor. Ce valoare va fi afișată în **%EAX** în urma rulării comenzilor `b main`, `run`, `stepi`, `stepi`, `i r`?

```
.data
    x: .long 0x04030201
    y: .long 0x08070605
.text
.global main
main:
    mov x, %eax
    mov y, %al  # (sau %ah)
    mov $1, %eax
    mov $0, %ebx
    int $0x80
```

pentru `mov y, %al` sau `mov y, %ah`

a) 0x08070605
b) 0x04030205
c) 0x04030501
d) 1
e) 0x04030201

Întrebarea 5

Ce valoare va reține registrul CH după executarea următoarei instrucțiuni?

```
movl $553, %ecx
```

a) 0
b) 2
c) 10
d) 512

Întrebarea 6

Fie următoarea declarare în secțiunea .data:

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

Ce se va afișa în urma apelului WRITE următor?

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

a) 1234
b) 1234a
c) 12349
d) 123497

Întrebarea 7

Fie următorul program. Precizați secvența corectă de instrucțiuni în debugger, în urma căreia vom obține valoarea 8.

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

a) `b main; run; i r eax`
b) `b main; run; stepi; stepi; i r cl`
c) `b main; run; stepi; stepi; i r ch`
d) `b main; run; stepi; i r ah`

2 Test Lab 3.1 / 3.2

(vezi rezolvări)

Întrebarea 1

Fie codul de mai jos. Care este valoarea lui s când execuţia ajunge la `et_exit`?

```
.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
```

a) 1
b) 0
c) 0xffffffff
d) 15

Întrebarea 2

Se stochează în registrul eax valoarea `0x40000000`, în ebx `0x8`, în ecx `0x1` și în edx `0x8`. Ce valori vor avea registrii eax si edx dupa executarea instructiunii `mul %edx`?

a) eax=32, edx=0
b) eax=0, edx=2
c) eax=2, edx=0
d) eax=32, edx=2

Întrebarea 3

Se stochează în %edx valoarea 0, în eax 47 și în ebx 15. Ce valori vor avea registrii eax si edx dupa executarea instructiunii `div %ebx`?

a) eax=2, edx=3
b) eax=3, edx=0
c) eax=3, edx=2
d) eax=2, edx=0

Întrebarea 4

Fie codul de mai jos. Care este valoarea depozitată la final în ecx (în dreptul etichetei `exit`)?

```
.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
```

a) 5
b) 6

Întrebarea 5

Fie următorul program. Ce valoare vom obține dacă vom rula cu debuggerul `b et_exit`, `run`, `ir 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
```

a) 8
b) 3
c) 5
d) 2

Întrebarea 6

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
```

a) ety, etz, ety, ett
b) etx, ety, ett
c) ety, etx, etz, ett
d) etx, ety, etz, ety, ett

Întrebarea 7

Fie codul de mai jos. Care sunt valorile lui eax și edx când execuția ajunge la `label`?

```
.data
    x: .long 17
    y: .long 6
.test
.global main
main:
    mov $1, %edx
    mov x, %eax
    jmp et
    mov $0, %edx
et:
    divl y
label:
    mov $1, %eax
    mov $0, %ebx
    int $0x80
```

a) eax=0x2, edx=0x5
b) eax=0x2aaaaaad, edx=0x3
c) eax=0x5, edx=0x2
d) eax=0x3, edx=0x2aaaaaad

Întrebarea 8

Se stochează în EAX valoarea `0x80000000`, în EBX `0x8`, în ECX `0x1` și în EDX `0x4`. Ce valori vor avea registrii EAX, respectiv EDX după executarea instrucțiunii `mul %ebx`?

a) eax=0, edx=4
b) eax=32, edx=0
c) eax=4, edx=0
d) eax=32, edx=4

Întrebarea 9

Se stochează în %edx valoarea 0, în eax 37 și în ebx 15. Ce valori vor avea registrii eax si edx dupa executarea instructiunii `div %ebx`?

a) eax=7, edx=0
b) eax=2, edx=7
c) eax=2, edx=0
d) eax=7, edx=2

Întrebarea 10

Fie codul de mai jos. Care este valoarea depozitată în z când ajungem la eticheta `final`?

```
.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
```

```
et:
    mov x1, %eax
    mov y1, %ebx
    cmp %eax, %ebx
    jge et2
    add %eax, %ebx
    jmp final
et1:
    add %eax, %ebx
    jmp final
et2:
    sub %eax, %ebx
final:
    mov %ebx, z
    mov $1, %eax
    mov $0, %ebx
    int $0x80
```

a) 12
b) 4
c) 1
d) 23

Întrebarea 11

Care este ordinea de trecere prin etichete?

```
.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
```

a) etb, eto, eth, etd
b) etb, eth, eto, etd
c) eto, eth, etb,etd
d) eth, etb, etd, eto

Întrebarea 12

De câte ori se va executa instrucțiunea `loop`?

```
.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
```

a) 0x5
b) 0x1
c) este un ciclu infinit
d) 0x0
e) 0xffffffff

Întrebarea 13

Fie următorul program. Ce valoare vom obține dacă vom rula cu debuggerul `b et_exit`, `run`, `ir 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
```

a) 0x4
b) 0x3
c) 0x1
d) 0x0

3 Test Lab 4.1 / 4.2 / 4.3

(vezi rezolvări)

Întrebarea 1

Ce valori vor fi depozitate în v când execuția va ajunge în dreptul etichetei `et_exit`?

```
.data
    v: .space 20  # (sau .space 24)
    n: .long 5
.text
.global main
main:
    lea v, %edi  # (sau movl $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
```

a) 0, 1, 2, 3, 4, 5
b) 11, 12, 13, 14, 15, 16
c) 11, 12, 13, 14, 15
d) 11, 12, ..., 27
e) Execuția nu ajunge la et_exit

Întrebarea 2

Ce se va afișa pe ecran?

```
.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
```

a) ababaabbab
b) aaaaaaaaaa
c) bbbbbaaaaa
d) 61, 61, 61, 61, 61, 61, 61, 61, 61, 61

Întrebarea 3

Ce se va afișa pe ecran?

```
.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
```

a) Abc
b) ABC
c) abc
d) Abc + o valoarea reziduala

Întrebarea 4

Ce se va afișa pe ecran?

```
.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
```

a) abc
b) cba
c) def
d) dcb

Întrebarea 5

Fie următorul program. Ce valoare vom obține dacă vom rula `b et_exit`, `run`, `ir 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
```

a) 0x0506070c
b) 0x090a0b08
c) 0x05060704
d) 0x090a0b0c

Întrebarea 6

Fie următorul program. Ce valoare vom obține dacă vom rula `b et_exit`, `run`, `ir 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
```

a) 0x05060708
b) 0x01020304
c) Execuție cu eroare
d) 0x090a0b0c

Întrebarea 7

Ce se va afișa pe ecran?

```
.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
```

a) aCdBx
b) 1C95dB3
c) 1953
d) a1C95dBx3

Întrebarea 8

Ce valori vor fi stocate în x când execuția va ajunge la eticheta `et_exit`?

```
.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
```

a) 16, 4, 1, 25, 36
b) 1, 2, 3, 4, 5
c) 16, 4, 2, 32, 64
d) 0, 0, 0, 0, 0

Întrebarea 9

Fie următorul program. Ce valori se vor regăsi în registrul %ebx la trecerea prin eticheta `et`?

```
.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
```

a) 4, 18, 16, 30
b) 15, 21, 30, 16
c) 18, 16, 30, 21
d) 15, 21, 30, 16, 18

Întrebarea 10

Fie următorul program. Ce valoare va avea elementul din mijloc din vector dacă vom rula `b et_exit`, `run`, `x/3xw 8v`?

```
.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
```

a) 0x0506070c
b) 0x090a0b0c
c) 0x01020708
d) 0x05060708

Întrebarea 11

Ce valoare va fi stocată în s când execuția va ajunge la eticheta `et_exit`?

```
.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  # (sau movl (%esi), %ebx)
    add %ebx, s        # (sau addl %ebx, s)
    add $8, %esi       # (sau addl $8, %esi)
    inc %ecx
    jmp et_loop

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

a) 63
b) 66
c) 129
d) 23

Întrebarea 12

Fie următorul program. Este acesta scris corect? Daca da, de ce, daca nu, de ce?

```
.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
```

a) este scris corect, deoarece x1 e doar o adresa, ca numele unui vector.
b) nu este scris corect, din cauza ca intram in zone de memorie;
c) nu este scris corect, deoarece n trebuia sa fie egal cu 1;
d) nu este scris corect, deoarece foloseste x1 pe post de array;
e) este scris corect, dar nu va functiona conform asteptarilor;

Întrebarea 13

Știind 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?

```
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
```

a) 15720
b) 27510
c) 83412
d) 1024
e) 1572

4 Rezultate

Test Lab 2.1 / 2.2 / 2.3

1. b)
2. a)
3. a) / b)
4. b) / c)
5. b)
6. b)
7. c)

Test Lab 3.1 / 3.2

1. a)
2. b)
3. c)
4. a)
5. c)
6. a)
7. b)
8. a)
9. b)
10. b)
11. b)
12. c)
13. a)

Test Lab 4.1 / 4.2 / 4.3

1. b)
2. a)
3. b)
4. d)
5. a)
6. b)
7. c)
8. c)
9. a)
10. c)
11. a)
12. a)
13. b)

5 Explicații

5.1 Lab 2.x

Întrebarea 1

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

- `movl $0, %eax`: EAX este `0x00000000`.
- `movb $4, %ah`: Partea `AH` (biții 8-15) devine 4. EAX este `0x00000400`.
- `movb $2, %al`: Partea `AL` (biții 0-7) devine 2. EAX este `0x00000402`.
- `0x402` în zecimal este $4 \times 16^2 + 0 \times 16^1 + 2 \times 16^0 = 4 \times 256 + 2 = 1024 + 2 = 1026$.

Întrebarea 2

```
.data
x: .word 1
y: .word 2
.text
mov x, %eax
```

- `mov x, %eax` este ambiguu. Compilatorul (GCC/as) îl tratează adesea ca `movl` (mută 4 bytes).
- Citește 2 bytes de la `x` (valoarea 1) în `%ax`.
- Citește următorii 2 bytes din memorie (valoarea lui `y`, adică 2) în biții superiori ai `%eax`.
- Rezultat EAX: `0x00020001`.

Întrebarea 3

```
.data
str1: .ascii "abc"
str2: .ascii "123"
.text
movl $4, %eax
...
movl $str1, %ecx
movl $5, %edx
int $0x80
```

- `.ascii` nu adaugă terminator null. Datele în memorie sunt contigue: `'a','b','c','1','2','3'`.
- `movl $str1, %ecx` încarcă adresa lui `'a'` în `%ecx`.
- `movl $5, %edx` specifică sistemului să citească 5 bytes de la acea adresă.
- Sistemul citește: `'a','b','c','1','2'`.
- Rezultatul afișat este `abc12`.

Întrebarea 4

```
.data
x: .long 0x04030201
y: .long 0x08070605
.text
main:
    mov x, %eax
    mov y, %ah # varianta 2
    # mov y, %al # varianta 1
    ...
```

varianta 1: `mov y, %al`

- `stepi 1 (mov x, %eax)`: EAX = `0x04030201`.
- `stepi 2 (mov y, %al)`: `mov` citește valoarea `y` (0x08070605). Deoarece destinația e `%al` (8 biți), doar byte-ul cel mai puțin semnificativ (LSB) al lui `y` este folosit. LSB-ul lui `y` este `0x05`.
- `%al` (care era `0x01`) este suprascris cu `0x05`.
- Rezultat EAX: `0x04030205`.

varianta 2: `mov y, %ah`

- `stepi 1 (mov x, %eax)`: EAX = `0x04030201`.
- `stepi 2 (mov y, %ah)`: LSB-ul lui `y` (`0x05`) este copiat în `%ah`.
- `%ah` (care era `0x02`) este suprascris cu `0x05`.
- Rezultat EAX: `0x04030501`.

Întrebarea 5

```
movl $553, %ecx
```

- Se încarcă valoarea 553 în `%ecx`.
- 553 în zecimal este `0x229` în hexazecimal (pe 16 biți, `0x0229`).
- Registrul `%ecx` (32 biți) arată astfel: `0x00000229`.
- `%cx` (16 biți) este `0x0229`.
- `%ch` (partea high a `%cx`) este `0x02`.
- `%cl` (partea low a `%cx`) este `0x29`.
- Valoarea din `%ch` este 2.

Întrebarea 6

```
.data
str: .ascii "1234"
x: .byte 97
.text
movl $4, %eax
...
movl $str, %ecx
movl $5, %edx
int $0x80
```

- La fel ca la Întrebarea 3, `.ascii` nu pune terminator null.
- În memorie, datele sunt: `'1','2','3','4'` (de la `str`) urmate imediat de `97` (de la `x`).
- `movl $str, %ecx` încarcă adresa lui `'1'`.
- `movl $5, %edx` cere sistemului să citească 5 bytes.
- Sistemul citește: `'1','2','3','4'` (4 bytes) și următorul byte, care este `97`.
- Byte-ul 97 este interpretat ca și caracterul ASCII `'a'`.
- Rezultat afișat: `1234a`.

Întrebarea 7

```
main:
    movl $8, %eax
    movl $2048, %ecx
et_exit:
...
```

- `b main; run;`: Programul se oprește la `main:`.
- `stepi 1`: Execută `movl $8, %eax`. Acum `%eax` = 8.
- `stepi 2`: Execută `movl $2048, %ecx`.
- 2048 în zecimal este `0x800` în hexazecimal.
- `%ecx` = `0x00000800`.
- `%cx` = `0x0800`.
- `%ch` (partea high) = `0x08` (adică 8).
- `%cl` (partea low) = `0x00` (adică 0).
- Comanda `i r ch` (info register ch) va afișa 8.

5.2 Lab 3.x

Întrebarea 1

```
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, %edx` și `mov $0, %eax`. Deîmpărțitul (EDX:EAX) este 2^{32}.
- `movl $0xffffffff, %ebx`. Împărțitorul `%ebx` este $2^{32} - 1$.
- `divl %ebx` este o împărțire *fără semn*.
- Se calculează $2^{32}/(2^{32} - 1)$.
- Câtul este 1, Restul este 1.
- Câtul (1) se stochează în `%eax`.
- `mov %eax, %ecx`: `%ecx` devine 1.
- `et_loop:`:
- `add %ecx, s`: `s` devine 1.
- `loop et_loop`: Decrementează `%ecx` (devine 0) și NU sare, deoarece `%ecx` este 0.
- Valoarea finală a lui `s` este 1.

Întrebarea 2

```
# eax = 0x40000000 (2^30)
# edx = 0x8 (8)
# ebx = 0x8 (8)
mul %edx
```

- `mul %edx` este o înmulțire *fără semn*.
- Calculează `EAX * EDX` și stochează rezultatul pe 64 de biți în `EDX:EAX`.

- `EAX` = 2^{30}. `EDX` = $8 = 2^3$.
- Calculul este $2^{30} \times 2^3 = 2^{33}$.
- 2^{33} pe 64 de biți este `0x00000002_00000000`.
- Partea superioară (32 biți) merge în `%edx`.
- Partea inferioară (32 biți) merge în `%eax`.
- Rezultat: `%edx` = `0x2`, `%eax` = `0x0`.

Întrebarea 3

```
# edx = 0
# eax = 47
# ebx = 15
div %ebx
```

- `div %ebx` este o împărțire *fără semn*.
- Împarte valoarea pe 64 de biți din `EDX:EAX` la `%ebx`.

- Deîmpărțitul `EDX:EAX` este `0x00...0 : 47` (adică 47).
- Împărțitorul `%ebx` este 15.
- 47/15 = 3 (câtul) și 2 (restul).
- Câtul se stochează în `%eax`.
- Restul se stochează în `%edx`.
- Rezultat: `%eax` = 3, `%edx` = 2.

Întrebarea 4

```
.data
    x: .long 0x80000000
    y: .long 0x70000000
.text
main:
    mov x, %eax
    cmp y, %eax
    jge label
    mov $5, %ecx
    jmp exit
label:
    mov $6, %ecx
exit:
...
```

- `mov x, %eax`: `%eax` = `0x80000000`.
- `cmp y, %eax`: Compară `%eax` (op2) cu `y` (op1, `0x70000000`).
- `jge` (jump if greater or equal) este un salt *cu semn*.
- `0x80000000` este cel mai mic număr negativ (în complement de 2).
- `0x70000000` este un număr pozitiv mare.
- Deoarece un număr negativ este *mai mic* decât un număr pozitiv, condiția "greater or equal" este falsă.
- Saltul `jge label` nu se execută.
- Se execută `mov $5, %ecx`.
- Se execută `jmp exit`.
- Rezultat: `%ecx` = 5.

Întrebarea 5

```
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 $3, %eax`: `%eax` = 3.
- `shl $2, %eax`: `%eax` = $3 \ll 2 = 3 \times 4 = 12$.
- `mov $2, %ebx`: `%ebx` = 2.
- `mul %ebx`: `EDX:EAX` = `EAX * EBX` = $12 \times 2 = 24$. (`%eax`=24, `%edx`=0).
- `mov $0, %edx`: `%edx` = 0 (era deja 0).
- `mov $8, %ebx`: `%ebx` = 8.
- `div %ebx`: Împarte `EDX:EAX` (24) la `%ebx` (8).
- `%eax` (câtul) = 3. `%edx` (restul) = 0.
- `sub %eax, %ebx`: `%ebx` = `%ebx - %eax` = $8 - 3 = 5$.
- Rezultat: `%ebx` = 5.

Întrebarea 6

```
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:
...
```

- `cmp %ebx, %eax`: Compară `%eax` (1) cu `%ebx` (2). 1 == 2 este fals.
- Saltul `je etx` nu se execută.
- Se ajunge la `ety:`.
- `cmp %ecx, %edx`: Compară `%edx` (4) cu `%ecx` (3). $4 > 3$ este adevărat.
- `jg etz` (salt cu semn) se execută.
- Se ajunge la `etz:`.
- `mov %ebx, %edx`: `%edx` devine 2 (valoarea din `%ebx`).
- `jmp ety`: Se sare înapoi la `ety:`.
- `cmp %ecx, %edx`: Compară `%edx` (acum 2) cu `%ecx` (3). $2 > 3$ este fals.
- Saltul `jg etz` nu se execută.
- `jmp ett`: Se sare la `ett:`.
- Ordine: **ety, etz, ety, ett**.

Întrebarea 7

```
.data
x: .long 17
y: .long 6
.test
main:
    mov $1, %edx
    mov x, %eax
    jmp et
    mov $0, %edx
et:
    divl y
label:
...
```

- `mov $1, %edx`: `%edx` = 1.
- `mov x, %eax`: `%eax` = 17.
- `jmp et`: Sare peste `mov $0, %edx`. `%edx` rămâne 1.
- `et:`:
- `divl y`: Împarte `EDX:EAX` la `y` (6).
- Deîmpărțitul este $1 \times 2^{32} + 17 = 4294967296 + 17 = 4294967313$.
- $4294967313 / 6 = 715827885$ (cât) și 3 (rest).
- În hexazecimal, 715827885 este `0x2AAAAAAD`.
- Câtul se stochează în `%eax`, restul în `%edx`.
- Rezultat: `%eax` = `0x2aaaaaad`, `%edx` = 3.

Întrebarea 8

```
# eax = 0x80000000 (2^31)
# ebx = 0x8 (8)
# edx = 0x4 (4)
mul %ebx
```

- `mul %ebx` este *fără semn*.
- Calculează `EAX * EBX` = $(2^{31}) \times 8 = 2^{31} \times 2^3 = 2^{34}$.
- Valoarea din `%edx` (4) este ignorată.
- 2^{34} pe 64 de biți este `0x00000004_00000000`.
- Partea superioară → `%edx`.
- Partea inferioară → `%eax`.
- Rezultat: `%edx` = 4, `%eax` = 0.

Întrebarea 9

```
# edx = 0
# eax = 37
# ebx = 15
div %ebx
```

- `div %ebx` (fără semn).
- Împarte `EDX:EAX` (valoare 37) la `%ebx` (valoare 15).
- 37/15 = 2 (câtul) și 7 (restul).
- Câtul → `%eax`.
- Restul → `%edx`.
- Rezultat: `%eax` = 2, `%edx` = 7.

Întrebarea 10

```
.data
x: .long 17, y: .long 6
x1: .long 5, y1: .long 9
.text
main:
    mov x, %eax  # eax=17
    mov y, %ebx  # ebx=6
    cmp %eax, %ebx # comp 6 cu 17
    jge et1      # 6 >= 17? Fals.
    mov x1, %eax # eax=5
    cmp %eax, %ebx # comp 6 cu 5
    jle et       # 6 <= 5? Fals.
    mov x, %ebx  # ebx=17
et:
    mov x1, %eax # eax=5
    mov y1, %ebx # ebx=9
    cmp %eax, %ebx # comp 9 cu 5
    jge et2      # 9 >= 5? Adevarat.
    jmp et2      # Sare la et2
et2:
    sub %eax, %ebx # ebx = ebx-eax =
   9-5=4
final:
    mov %ebx, z  # z = 4
...
```

- `eax`=17, `ebx`=6. `cmp` (6 cu 17). `jge` (6>=17) e fals.
- `eax`=5. `cmp` (6 cu 5). `jle` (6<=5) e fals.
- `ebx`=17.
- `et:`: `eax`=5, `ebx`=9.
- `cmp` (9 cu 5). `jge` (9>=5) e adevărat. Se sare la `et2`.
- `et2:`: `ebx` = `ebx - eax` = $9 - 5 = 4$.
- `final:`: `mov %ebx, z`. Variabila `z` primește valoarea 4.

Întrebarea 11

```
main:
    jmp etb
eto:
    jmp etd
eth:
    jmp eto
etb:
    jmp eth
etd:
```

```
...
```

- `main:` sare la `etb`.
- `etb:` sare la `eth`.
- `eth:` sare la `eto`.
- `eto:` sare la `etd`.
- `etd:` oprește programul.
- Ordine: **etb, eth, eto, etd**.

Întrebarea 12

```
main:
    mov x, %ecx
    sub y, %ecx
et:
    add %ecx, s
    loop et
exit:
...
```

- `mov x, %ecx`: `%ecx` = 5.
- `sub y, %ecx`: `%ecx` = `%ecx - y` = $5 - 5 = 0$.
- `et:`:
- `add %ecx, s`: `s` = $0 + 0 = 0$.
- `loop et`: Instrucțiunea `loop` mai întâi decrementează `%ecx`. `%ecx` devine -1 (`0xFFFFFFFF`).
- Apoi verifică dacă `%ecx` != 0. Deoarece -1 != 0, sare la `et`.
- Bucla va rula de $2^{32} - 1$ (sau `0xFFFFFFFF`) ori până când `%ecx` va ajunge din nou la 0.

Întrebarea 13

```
main:
    mov $2, %eax
    mov $3, %ebx
    add %eax, %ebx
    mul %ebx
    mov $0, %edx
    divl $3
    add %eax, %edx
et_exit:
...
```

- `mov $2, %eax`: `%eax` = 2.
- `mov $3, %ebx`: `%ebx` = 3.
- `add %eax, %ebx`: `%ebx` = `%ebx` + `%eax` = 3 + 2 = 5.
- `mul %ebx`: `EDX:EAX` = `EAX * EBX` = 2 × 5 = 10. (`%eax`=10, `%edx`=0).
- `mov $0, %edx`: `%edx` = 0.
- `divl $3`: Împarte `EDX:EAX` (10) la 3.
- `%eax` (câtul) = 3. `%edx` (restul) = 1.
- `add %eax, %edx`: `%edx` = `%edx` + `%eax` = 1 + 3 = 4.
- Rezultat: `%edx` = 4.

5.3 Lab 4.x

Întrebarea 1

```
.data
    v: .space 20
    n: .long 5
.text
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:
...
```

- `v` are 20 bytes (spațiu pentru 5 `.long`-uri, la indecși 0-4). `n` este 5.
- Bucla rulează cât timp `%ecx <= n` (adică pentru `%ecx` = 0, 1, 2, 3, 4, 5).
- `ecx=0`: v[0] = 11. edx=12.
- `ecx=1`: v[1] = 12. edx=13.
- `ecx=2`: v[2] = 13. edx=14.
- `ecx=3`: v[3] = 14. edx=15.
- `ecx=4`: v[4] = 15. edx=16.
- `ecx=5`: `cmp` (5 cu 5). `jg` e fals.
- `mov %edx, (%edi, %ecx, 4)`: Scrie v[5] (care e în afara spațiului alocat, la adresa lui `n`) cu valoarea 16.
- Variabila `n` este suprascrisă cu 16.
- `inc %ecx`: `%ecx` = 6. `inc %edx`: `%edx` = 17.
- `et_loop:`: `cmp n, %ecx` (compară 6 cu 16). `jg` e fals.
- Bucla continuă până când `%ecx` devine 17 (când `%ecx`=16, `jg` e fals, `%ecx` devine 17, apoi la următorul `cmp`, 17 > 16 și iese).
- Valorile scrise în `v` (primii 5 longi) sunt: `11, 12, 13, 14, 15`. Valoarea de la `v[5]` (adica `n`) este 16. Răspunsul `11..16` este corect.

Întrebarea 2

```
.data
x: .long 1, 3, 6, 7, 9
n1: .long 5, n2: .long 10
c: .long 0x64636261
s: .space 11
.text
...
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_loop1` umple `s` cu 10 caractere `'a'` (din `movb c, %al`, `0x61`='a'). `s` = "aaaaaaaaaa".
- `et_exit_loop1`: `mov $c, %eax` (`eax`=adresa lui c). `movb 1(%eax), %al` (`al` = al doilea byte din c, adică `0x62` sau `'b'`).
- `et_loop2`: Parcurge vectorul `x` (cu `%esi` și `%ecx`).
- `ecx=0`: `ebx` = x[0] = 1. Scrie `'b'` la `s[1]`.
- `ecx=1`: `ebx` = x[1] = 3. Scrie `'b'` la `s[3]`.
- `ecx=2`: `ebx` = x[2] = 6. Scrie `'b'` la `s[6]`.
- `ecx=3`: `ebx` = x[3] = 7. Scrie `'b'` la `s[7]`.
- `ecx=4`: `ebx` = x[4] = 9. Scrie `'b'` la `s[9]`.
- `s` inițial: "aaaaaaaaaa"
- `s` final: "a**b**a**b**aa**bb**a**b**"
- Rezultat: `ababaabbab`.

Întrebarea 3

```
.data
n: .long 3, s: .asciz "abc"
.text
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, %edx
    int $0x80
...
```

- Bucla rulează pentru `%ecx` = 0, 1, 2.
- Pentru fiecare caracter `c` din `s`:
- `sub $'a', %al`: Calculează offset-ul față de 'a'. ('a'-'a'=0, 'b'-'a'=1, 'c'-'a'=2).
- `add $'A', %al`: Adaugă offset-ul la 'A'. (0+'A'='A', 1+'A'='B', 2+'A'='C').
- `mov %al, ...`: Suprascrie caracterul din `s`.
- Șirul `s` devine "ABC".
- `et_exit`: Afișează `s` cu lungimea 4 (include și terminatorul null, dar nu e vizibil).
- Rezultat: `ABC`.

Întrebarea 4

```
.data
n: .long 3
s: .byte 'a', 'b', 'c'
t: .byte 'd', 'e', 'f'
u: .space 4
.text
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
...
```

- Bucla rulează pentru `%ecx` = 0, 1, 2.
- `ecx=0`: `edx` = 0 - 0 = 0. `al` = t[0] = 'd'. `u[0]` = 'd'.
- `ecx=1`: `edx` = 0 - 1 = -1. `al` = t[-1]. În memorie, înainte de `t` este șirul `s` ('a','b','c'). `t[-1]` este 'c'. `u[1]` = 'c'.
- `ecx=2`: `edx` = 0 - 2 = -2. `al` = t[-2] = 'b'. `u[2]` = 'b'.
- `et_exit`: `u[3]` = 0 (terminator).
- Se afișează `u`, care conține "dcb".

Întrebarea 5

```
main:
    mov $v, %esi
    mov $1, %ecx
    mov (%esi, %ecx, 4), %eax
    add $4, %esi
    movb (%esi, %ecx, 4), %al
et_exit:
...
```

- `mov $v, %esi`: `%esi` = adresa lui `v`.
- `mov $1, %ecx`: `%ecx` = 1.
- `mov (%esi, %ecx, 4), %eax`: `%eax` = *(v + 1 * 4) = v[1].
- `%eax` = `0x05060708`.
- `add $4, %esi`: `%esi` = adresa lui `v` + 4 (adresa lui v[1]).
- `movb (%esi, %ecx, 4), %al`: Citește 1 byte de la adresa *(`esi` + `ecx` * 4) = *((v+4) + 1 * 4) = *(v+8) = v[2].
- Se citește doar byte-ul cel mai puțin semnificativ (LSB) de la v[2] (0x090a0b0c), adică `0x0c`.
- Acest `0x0c` suprascrie `%al` (care era `0x08`).
- Rezultat `%eax`: `0x0506070c`.

Întrebarea 6

```
main:
    mov $v, %esi
    mov $2, %ecx
    mov -8(%esi, %ecx, 4), %eax
et_exit:
...
```

- `mov $v, %esi`: `%esi` = adresa lui `v`.
- `mov $2, %ecx`: `%ecx` = 2.
- `mov -8(%esi, %ecx, 4), %eax`: Calculează adresa:
- Adresă = `%esi` + `%ecx` * 4 - 8
- Adresă = (adresa `v`) + 2 * 4 - 8
- Adresă = (adresa `v`) + 8 - 8
- Adresă = adresa `v`.
- Se citește valoarea de la adresa `v`, adică v[0].
- Rezultat: `%eax` = `0x01020304`.

Întrebarea 7

```
.data
s: .asciz "a1C95dBx3"
t: .space 10
.text
et_loop:
...
    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:
...
```

- Bucla `et_loop` parcurge șirul `s`.
- `%ecx` este contorul pentru `s`, `%edx` este contorul pentru `t`.
- Pentru fiecare caracter `%al` din `s`:
- Se verifică dacă `%al < '0'` sau `%al > '9'`.
- Dacă este adevărat (nu e cifră), sare la `et2`.
- Dacă ambele sunt false (este cifră), este copiată în șirul `t` la indexul `%edx`, iar `%edx` este incrementat.
- Șirul `t` va conține doar cifrele din `s`.
- Rezultat: `1953`.

Întrebarea 8

```
.data
x: .long 4, 2, 1, 5, 6
n: .long 5
.text
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
...
```

- Bucla parcurge `x` cu indexul `%eax` (de la 0 la 4).
- `mov ... %ecx`: `%ecx` = x[`%eax`].
- `mov $1, %ebx`: `%ebx` = 1.
- `sal %cl, %ebx`: Shift stânga aritmetic. `%ebx` = `%ebx` « `%cl`.
- Calculează `1 « x[eax]`, adică $2^{x[eax]}$.
- `mov %ebx, ...`: Suprascrie x[`%eax`] cu noul rezultat.
- x[0] = $2^4 = 16$
- x[1] = $2^2 = 4$
- x[2] = $2^1 = 2$
- x[3] = $2^5 = 32$
- x[4] = $2^6 = 64$
- Rezultat: `16, 4, 2, 32, 64`.

Întrebarea 9

```
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
...
```

- `mov $n, %esi`: `%esi` = adresa lui `n`.
- `sub n, %eax`: `%eax` = 0 - n = 0 - 4 = -4.
- `mov $0, %ecx`: `%ecx` = 0.
- Bucla rulează cât timp `%ecx != %eax` (-4).
- `ecx=0`: `cmp -4, 0`. `ebx` = *(`esi` + 0*4) = *(&n) = 4. `ecx`=-1.
- `ecx=-1`: `cmp -4, -1`. `ebx` = *(`esi` - 4) = *(&n - 4) = `v[4]` (18). `ebx`=18. `ecx`=-2.
- `ecx=-2`: `cmp -4, -2`. `ebx` = *(`esi` - 8) = `v[3]` (16). `ebx`=16. `ecx`=-3.
- `ecx=-3`: `cmp -4, -3`. `ebx` = *(`esi` - 12) = `v[2]` (30). `ebx`=30. `ecx`=-4.
- `ecx=-4`: `cmp -4, -4`. Sare la `et_exit`.
- Valorile succesive din `%ebx` sunt: **4, 18, 16, 30**.

Întrebarea 10

```
main:
    mov $v, %esi
    mov $2, %ecx
    mov (%esi, %ecx, 1), %eax
    mov %eax, 4(%esi, %ecx, 1)
et_exit:
...
```

- `mov $v, %esi`: `%esi` = adresa `v`. `ecx` = 2.
- `mov (%esi, %ecx, 1), %eax`: Citește 4 bytes (pentru `%eax`) de la adresa `v + 2*1 = v+2`.
- Memorie (Little Endian):
- v[0]: `04 03 02 01` (la `v`)
- v[1]: `08 07 06 05` (la `v+4`)
- `v+2` este adresa byte-ului `02`.
- `%eax` citește 4 bytes de la `v+2`: `02, 01` (din v[0]), `08, 07` (din v[1]).
- `%eax` = `0x07080102`.
- `mov %eax, 4(%esi, %ecx, 1)`: Scrie `%eax` la adresa `v + 2 + 4 = v+6`.
- Adresa `v+6` este la al 3-lea byte din `v[1]`.
- `v[1]` (inițial `0x05060708`) devine suprascris parțial. Byte-ul `06` e suprascris cu `01`, byte-ul `05` e suprascris cu `02`.
- `v[1]` devine `0x01020708`.
- Aceasta este valoarea elementului din mijloc.

Întrebarea 11

```
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
...
```

- `mov n, %eax`: `%eax` = 6.
- `shr $1, %eax`: `%eax` = 6 / 2 = 3. Bucla va rula pentru `%ecx` = 0, 1, 2.
- `%esi` începe la `v`.
- `ecx=0`: `ebx` = *(v) = v[0] = 15. `s` = 15. `%esi` = v + 8 (adresa lui v[2]).
- `ecx=1`: `ebx` = *(v+8) = v[2] = 30. `s` = 15 + 30 = 45. `%esi` = v + 16 (adresa lui v[4]).
- `ecx=2`: `ebx` = *(v+16) = v[4] = 18. `s` = 45 + 18 = 63. `%esi` = v + 24 (dincolo de vector).
- `ecx=3`: `cmp 3, 3`. `jge` sare la `et_exit`.
- Rezultat: `s` = 63.

Întrebarea 12

```
.data
    x1: .long 5
    x2: .long 12
    x3: .long 27
.text
main:
    movl $x1, %edi
    movl $0, %ecx
et_loop:
    cmp n, %ecx
    je et_exit
    movl (%edi, %ecx, 4), %eax
...
```

- `x1`, `x2`, `x3` sunt declarate `.long` consecutiv. În memorie, ele formează un tablou.
- `movl $x1, %edi`: `%edi` = adresa lui `x1`.
- Bucla rulează pentru `%ecx` = 0, 1, 2.
- `ecx=0`: `eax` = *(edi + 0*4) = *(&x1) = 5.
- `ecx=1`: `eax` = *(edi + 1*4) = *(&x1 + 4). Deoarece `x1` e `.long` (4 bytes), `&x1 + 4` este adresa lui `x2`. `eax` = 12.
- `ecx=2`: `eax` = *(edi + 2*4) = *(&x1 + 8). Aceasta este adresa lui `x3`. `eax` = 27.
- Programul este scris corect. El tratează `x1` ca pe adresa de început a unui tablou.

Întrebarea 13

```
main:
    movl $0, %ecx
    movl $10, %edi
et_while:
    cmp x, %ecx
    je et_exit
    movl x, %eax
    movl $0, %edx
    divl %edi
    movl %eax, x
    movl %edx, %esi
    movl y, %eax
    mul %edi
    add %esi, %eax
    movl %eax, y
    jmp et_while
...
```

- Acest cod inversează cifrele unui număr.
- `divl %edi` (împărțire la 10): `%eax` = câtul (noul `x`), `%edx` = restul (cifra).
- `movl y, %eax; mul %edi`: `%eax` = `y` * 10.
- `add %esi, %eax; movl %eax, y`: `y` = (`y` * 10) + cifra.
- Bucla se oprește când `x` devine 0 (comparat cu `%ecx` care e 0).
- Dacă `y` la final este 1572, înseamnă că `x` la început a fost 2751.
- Verificăm opțiunile: 27510.
- Dacă `x` = 27510:
- 1. x=2751, y=0+0=0
- 2. x=275, y=0*10+1=1
- 3. x=27, y=1*10+5=15
- 4. x=2, y=15*10+7=157
- 5. x=0, y=157*10+2=1572
- 6. `cmp x, %ecx` (0==0) → `je et_exit`.
- Valoarea inițială corectă este `27510`.