

Test de laborator

Arhitectura Sistemelor de Calcul

Seria 15

8 Ianuarie 2021

Cuprins

1	Informatii generale	1
2	Subiectul I - Implementare (3p)	2
2.1	Problema 1 (0.75p / 3p)	2
2.2	Problema 2 (0.75p / 3p)	2
2.3	Problema 3 (1.5p / 3p)	2
3	Subiectul II - Analiza programelor (4p)	3
4	Subiectul III - Intrebari generale (2p)	4

1 Informatii generale

1. Se acorda 1p din oficiu, iar nota maxima ce poate fi obtinuta este 10.
2. Subiectul e impartit in trei: o parte de implementare, o parte de analiza de cod si o parte de intrebari generale. Nu exista un punctaj minim pe fiecare parte, dar nota finala trebuie sa fie minim 5, fara nicio rotunjire superioara, pentru a promova.
3. Timpul efectiv de lucru este de 1h si 30 de minute din momentul in care subiectele sunt transmise. Rezolvarile vor fi completate in Google Form-ul asociat.
4. Este permis accesul la orice fel de materiale, insa orice incercare de fraudare atrage, dupa sine, notarea examenului cu nota finala 1 si sesizarea *Comisiei de etica a Universitatii din Bucuresti*.
5. In cazul suspiciunilor de frauda, studentii vizati vor participa si la o examinare orala.
6. In timpul testului de laborator, toate intrebarile privind subiectele vor fi puse pe Classroom, pentru a avea toata lumea acces la intrebari si raspunsuri.

2 Subiectul I - Implementare (3p)

2.1 Problema 1 (0.75p / 3p)

Sa se implementeze procedura `customEven` care primeste ca argumente doua numere intregi, x si y astfel incat $x \cdot y \geq 0$ si care sa returneze prin intermediul registrului `%eax` valoarea 1 daca suma cifrelor numarului $x \cdot y$ este para, respectiv 0 daca suma cifrelor numarului $x \cdot y$ este impara.

2.2 Problema 2 (0.75p / 3p)

Sa se implementeze procedura `divisors` care, primind ca argument un numar natural x , afiseaza toti divizorii acestuia.

2.3 Problema 3 (1.5p / 3p)

Fie dat un *array* v in memorie, continand elemente intregi. Utilizand procedurile de la problemele de mai sus, sa se afiseze pe ecran toate elementele $v[i]$ din *array* impreuna cu divizorii lor, pentru care `customEven(v[i], v[i+1])` intoarce valoarea 1. Pentru ultimul element din *array*, $v[i+1]$ va fi considerat ca fiind $v[0]$.

Important

1. Procedurile vor fi implementare respectand conventiile prezentate in cadrul laboratorului, referitoare la constructia cadrului de apel si la restaurarea registrilor.
2. Pentru implementarea corecta a problemei, fara utilizarea procedurilor, se acorda cel mult 1p. Daca se implementeaza doar una dintre proceduri, se acorda cel mult 1.5p.

3 Subiectul II - Analiza programelor (4p)

Fie urmatorul program, dezvoltat in limbajul de asamblare x86.

```
.data
    n: .long 5
    m: .long 100
    formatPrintf: .asciz "%d "
.text

.global main

f:
    pushl %ebp
    movl %esp, %ebp

    movl %ecx, 8(%ebp)
    movl %eax, 12(%ebp)

    divl %ecx
    pushl %eax

f_for:
    pushl %ecx
    pushl $formatPrintf
    call printf
    popl %ebx
    popl %ebx

    pushl $0
    call fflush
    popl %ebx

    loop f_for

f_exit:
    popl %eax
    popl %ebp
    ret

main:
    movl n, %edx
    decl %edx

    pushl m
    pushl %edx
    call f
    popl %ebx
    popl %ebx

    pushl %eax
    pushl $formatPrintf
    call printf
    popl %ebx
    popl %ebx

    pushl $0
    call fflush
    popl %ebx

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

Programul de mai sus nu functioneaza corect.

1. Care sunt modificarile minimale ce trebuie efectuate pentru a elimina erorile? Explicati fiecare corectura in parte. Nu se accepta, pentru corectarea erorilor, adaugarea de variabile suplimentare in sectiunea `.data`.
2. Ce alte modificari sunt necesare, dar care in forma actuala nu produc nicio eroare? De ce sunt necesare si de ce nu produc nicio eroare?
3. Ce va afisa, dupa corectarea erorilor minimale, codul de mai sus?
4. Explicati necesitatea utilizarii lui `fflush` si necesitatea instructiunii `pushl %eax`, insotita de un `popl %eax` in cadrul procedurii `f`, stiind ca procedura `f` returneaza prin intermediul lui `%eax`.

4 Subiectul III - Intrebari generale (2p)

1. Sa se justifice echivalenta in limbajul C a scrierilor `v[i]` si `i[v]`, unde `v` este un *array* de elemente intregi, iar `i` este un index valid in acest context. Se va utiliza, pentru demonstrarea echivalentei, scrierea in limbajul x86, sintaxa AT&T.
2. Cum se poate explicita instructiunea `popl %ebx`? Care va fi valoarea din registrul `%esp` dupa executarea ei, stiind ca, initial in `%esp` aveam valoarea `0xffffd05c`? Cum ar arata o secventa foarte scurta de cod, scrisa in x86, care sa genereze un *stack overflow*?
3. Este posibil sa aflam adresa din memorie a etichetelor din program? Daca da, cum? Daca nu, de ce? Dar sa le modificam? Daca da, cum, si daca nu, de ce?
4. Presupunem ca exista in x86 un tip de date `unsigned` pentru declararea numerelor naturale pe 4 Bytes, astfel incat, in codificarea interna, sa nu fie necesar un bit de semn. De exemplu, numarul 10 ar fi codificat `00 00 00 0A` in hexa, respectiv `00000000 00000000 00000000 00001010` in binar. In aceste conditii, fie urmatorul program:

```
.data
    v: .unsigned 10, 20, 30
.text

.global main

main:
    lea v, %edi
    movl $2, %ecx
    movl (%edi, %ecx, 1), %ebx

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

Daca in momentul in care ajungem la `et_exit` inspectam valoarea din registrul `%ebx`, ce valoare vom gasi? **Atentie!** Programul nu poate fi copiat si rulat pentru a gasi raspunsul, intrucat tipul `unsigned` nu exista, iar pentru tipul `long`, rezultatul este diferit de cel asteptat.