Barem de evaluare - Test de laborator
Arhitectura Sistemelor de Calcul

Seriile 13 & 14 & 15

7 Ianuarie 2022

Cuprins

1 Informatii generale

1. Nota maxima care poate fi obtinuta este 10.
2. Subiectul e impartit in trei: o parte de implementare si intrebari teoretice asupra implementarii, 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 2h din momentul in care subiectele sunt transmise. Rezolvarile vor fi completate in Google Form-ul asociat.
   - Seriile 13 & 15: `https://forms.gle/Ey4kPA8pe9Vg2YKx6`
   - Seria 14: `https://forms.gle/DT7oUz5QdSDNGp95A`
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 Teams - General, pentru a avea toata lumea acces la intrebari si raspunsuri.

2 Partea 0x00 - maxim 4p

Fie urmatoarea functie `f`, definita astfel:

$$f(x) = \begin{cases} \text{stop} & x = 1 \\ f(\frac{x}{2}) & x \text{ par} \\ f(3x+1) & x \text{ impar} \end{cases}$$

Subiectul 1 (1.5p procedura + 0.5p main) Sa se implementeze functia recursiva `f` in limbajul de asamblare `Intel x86`, sintaxa `AT&T`, care primeste ca argument un **numar natural nenul** si returneaza **numarul de autoapeluri pana la obtinerea rezultatului 1**. **Important! Se accepta variabila care numara autoapelurile sa fie declarata in sectiunea .data.** Sa se scrie un program complet, in care se citeste un numar de la tastatura, se apeleaza procedura `f` si se afiseaza pe ecran cate autoapeluri au fost necesare pentru a obtine `1`.
De exemplu, pentru `x = 5`, apelurile sunt $5 \to 16 \to 8 \to 4 \to 2 \to 1$, insemnand 5 apeluri.

Solutie propusa (sau orice solutie corecta):

```
.data
        x: .space 4
        count: .long 0
        formatPrintf: .asciz "%d\n"
        formatScanf: .asciz "%d"
.text

.global main

f:
        pushl %ebp
        movl %esp, %ebp
        pushl %ebx

        movl 8(%ebp), %eax
        cmp $1, %eax
        je f_exit

        movl $2, %ebx
        pushl %eax
        xorl %edx, %edx
        divl %ebx
        popl %eax

        cmp $0, %edx
        je f_even

        incl count
        movl $3, %ebx
        mull %ebx
        incl %eax
        pushl %eax
        call f
        popl %eax

        jmp f_exit

f_even:
        divl %ebx
        incl count
        pushl %eax
        call f
        popl %eax

f_exit:
        popl %ebx
        popl %ebp
        ret

main:
        pushl $x
        pushl $formatScanf
        call scanf
        popl %ebx
        popl %ebx

        pushl x
        call f
        popl %ebx
```

```
et_exit:
        pushl count
        pushl $formatPrintf
        call printf
        popl %ebx

        popl %ebx

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

- daca programul ruleaza corect, se pleaca de la 2p;
- daca nu sunt utilizate procedurile, punctajul e de 30% din intreg punctajul pe subiect, deci maxim 0.6p;
- se scad 0.25p pentru fiecare registru de restaurat care nu este utilizat corect in procedura;
- se scad 0.25p pentru apelul procedurilor printf / scanf care lipsesc sau care sunt apelate gresit.

Subiectul 2 (0.5p) Daca executam urmatorul `main` vom obtine `segmentation fault`; care este semnificatia acestei erori?

```
main:
    movl $100, %esp
    pushl %eax
    movl $0, %eax
    xorl %ebx, %ebx
    int $0x80
```

Solutie: Registrul `%esp` primeste o adresa de memorie dintr-o zona de adrese mici, in care programul nostru nu are drepturi. Cand se incearca `pushl %eax`, se incearca o completare de valoare la adresa nepermisa de memorie, si este aruncat `segmentation fault`.

Subiectul 3 (0.75p exemplu + 0.75p explicatie) Presupunem ca valoarea curenta a registrului `%esp` la inceperea executarii programului este `0xffff2022`, iar adresa pana la care avem spatiu disponibil pentru programul nostru este `0xffdf8d3a`. Dati exemplu de un input pentru procedura implementata de voi pentru care se obtine `segmentation fault`. (inputul oferit trebuie sa fie un numar cat mai mic pentru care se satisface aruncarea exceptiei) Explicati alegerea respectivului input.

Solutie:

- avem adresa initiala `0xffff2022`;
- adresa minima pana la care avem drepturi: `0xffdf8d3a`;
- calculam diferenta lor, iar rezultatul in baza 10 este `2069224`; acesta reprezinta numarul de Bytes disponibili pentru stiva;
- in functie de implementare, pentru cadrul local de apel din procedura avem cel putin urmatoarea stiva:

*alti registri
%ebp vechi
[adresa de intoarcere]
argument

deci un minim de 12Bytes ocupati. (in procedura de mai sus sunt 16Bytes)

- daca isi face un autoapel, dimensiunea stivei se dubleaza - initial era 12Bytes minim, acum are 24Bytes minim (respectiv 16Bytes pe implementarea de mai sus, 32Bytes dupa inca un autoapel);
- inseamna ca pentru X autoapeluri, avem minim 12(X + 1) Bytes ocupati, respectiv 16(X + 1) Bytes ocupati;
- consideram 12(X + 1) = 2069224, sau 16(X + 1) = 2069224, de unde X + 1 = 172435 rest..., si putem alege X = 172435 (in cazul cu 12Bytes);
- un input bun este $X = 2^{172435}$. Se poate reduce acest input, considerand ca provine dintr-un Y peste care s-a aplicat un $3Y + 1$ etc.
- se acorda punctaj doar pentru raspuns in concordanta cu implementarea de la Subiectul 1;
- se acorda punctaj maxim pentru finalizarea rationamentului pentru un input foarte mare, de forma 2^X, unde `X` este calculat conform Subiectului 1, **nu** este nevoie de o reducere a dimensiunii lui astfel incat sa incapa pe 32 de biti, de exemplu.

Important

1. Procedura va fi implementata respectand conventiile prezentate in cadrul laboratorului, referitoare la constructia cadrului de apel si la restaurarea registrilor.
2. Pentru implementarea corecta a problemei, dar fara utilizarea procedurilor, se acorda maxim 30% din punctajul acelui subiect.

3 Partea 0x01 - maxim 3p

Fie urmatorul program, dezvoltat in limbajul de asamblare x86:

```
.data
    x: .long 3
    lindex: .long L0, L1, L2, L3
    n: .long 7
    v: .long 15, 3, 2, 10, 1, 20, 0
    formatPrintf: .asciz "%d\n"
.text
.global main

f:
    pushl %ebp
    movl %esp, %ebp

    movl $0, %eax
    movl 8(%ebp), %ecx
    cmp $4, %ecx
    jge final

    cmp $-1, %ecx
    jae final

    movl $lindex, %edi
    movl (%edi, %ecx, 4), %eax

    jmp *%eax

L0:
    movl $1, %eax
    jmp final

L1:
    movl $2, %eax
    jmp final

L2:
    movl $3, %eax
    jmp final

L3:
    movl $4, %eax
    jmp final

final:
    popl %ebp
    ret

main:
    movl $v, %edi
    movl $0, %ecx

for_main:
    cmp n, %ecx
    jge final_main
    movl 0(%edi), %eax
    pushl %eax
    call f
    popl %ebx

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

    incl %ecx
    incl %edi

    jmp for_main

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

Atenție! Instrucțiunea **jmp *%eax** produce saltul la adresa reținută în registrul %eax.

Subiectul 4 (1.2p) Care sunt modificarile ce trebuie efectuate pentru a elimina erorile? Explicati fiecare corectura in parte.

Solutie (0.3p fiecare raspuns):

- se adauga `pushl %edi` si `popl %edi`;
- se adauga `pushl %ecx` si `popl %ecx`;
- se schimba `jae` in `jge`;
- se schimba `incl %edi` in `addl $4, %edi`.

Subiectul 5 (0.5p) Ce va afisa, dupa corectarea erorilor, codul de mai sus?

Solutie: se afiseaza

```
0
4
3
0
2
0
1
```

Subiectul 6 (1.3p) Scrieți o secvență într-un limbaj de nivel înalt (C, C++ etc) sau în pseudocod care să reflecte implemenarea din funcția **f**.

```
long f(long x)
{
    switch(x)
    {
        case 0: return 1;
        case 1: return 2;
        case 2: return 3;
        case 3: return 4;
        default: return 0;
    }
}
```

4 Partea 0x02 - maxim 3p

Subiectul 7 (0.25p raspuns + 0.75p argumentare) Analizati urmatorul cod, scris in `assembly x86`: contine o bucla infinita? Daca da, cum putem evita bucla infinita? Daca nu, de ce?

```
main:
        movl $0xae2b, %eax
        movl %eax, %ecx
        decl %ecx

et_loop:
        cmp $0, %ecx
        jl et_exit
        incl %ecx
        jmp et_loop

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

Solutie: nu este o bucla infinita. Echivalentul in C este urmatorul,

```
int i, n = 0xae2b;
for (i = n - 1; i >= 0; i++)
{

}
```

se va executa for-ul pentru `0xae2a`, `0xae2b`, `0xae2c`, ..., pana la `0x7fffffff` = `0b01...1` = `2147483647` (cel mai mare intreg cu semn reprezentabil pe 32 de biti), iar apoi va ajunge la `-2147483648` (`0b10...0`), unde nu se va mai respecta conditia de a ramane in cadrul structurii repetitive.

Subiectul 8 (2p) Presupunem ca aveti acces la un executabil `exec`, pe care il inspectati cu `objdump -d exec`. In momentul in care rulati aceasta comanda, va opriti asupra urmatorului fragment de cod. Analizati acest cod si raspundeti la intrebarile de mai jos. Pentru fiecare raspuns in parte, veti preciza si liniile de cod / instructiunile care v-au ajutat in rezolvare.

```
0000057d <func>:
 1. 57d: 55                    push   %ebp
 2. 57e: 89 e5                 mov    %esp,%ebp
 3. 580: e8 1b 01 00 00        call   6a0 <__x86.get_pc_thunk.ax>
 4. 585: 05 4f 1a 00 00        add    $0x1a4f,%eax
 5. 58a: 8b 45 08              mov    0x8(%ebp),%eax
 6. 58d: 83 e0 01              and    $0x1,%eax
 7. 590: 85 c0                 cmp    $0,%eax
 8. 592: 75 18                 jne    5ac <func+0x2f>
 9. 594: 8b 45 08              mov    0x8(%ebp),%eax
10. 597: 89 c2                 mov    %eax,%edx
11. 599: c1 ea 1f              shr    $0x1f,%edx
12. 59c: 01 d0                 add    %edx,%eax
13. 59e: d1 f8                 sar    %eax
14. 5a0: 89 c2                 mov    %eax,%edx
15. 5a2: 8b 45 0c              mov    0xc(%ebp),%eax
16. 5a5: 01 d0                 add    %edx,%eax
17. 5a7: 0f b6 00              movzbl (%eax),%eax
```

```
18. 5aa: eb 0b                  jmp    5b7 <func+0x3a>
19. 5ac: 8b 55 08               mov    0x8(%ebp),%edx
20. 5af: 8b 45 0c               mov    0xc(%ebp),%eax
21. 5b2: 01 d0                  add    %edx,%eax
22. 5b4: 0f b6 00               movzbl (%eax),%eax
23. 5b7: 5d                     pop    %ebp
24. 5b8: c3                     ret
```

a. (0.4p) Cate argumente primeste procedura de mai sus?
Raspuns: 2 argumente, avem `0x8(%ebp)` la linia 5, respectiv `0xc(%ebp)` la linia 15.

b. (0.4p) Care este conditia de salt din instructiunea 8?
Raspuns:

- se incarca `0x8(%ebp)` in `%eax`, primul argument;
- se face and cu `0x1` si se compara rezultatul cu `0`;
- daca este diferit de `0`, se efectueaza saltul, deci conditia de salt este daca numarul este impar.

c. (0.4p) Este macar unul dintre argumente un pointer?
Raspuns: Da, al doilea argument este un pointer. Observam ca avem `mov 0xc(%ebp), %eax` la linia 15, iar la linia 17 se preia continutul de memorie indicat de `%eax`. Analog si la liniile 20 si 22.

d. (0.4p) Este tipul returnat un pointer?
Raspuns: Nu, inainte de `ret` se pune `(%eax)` in `%eax`, deci un continut de la o anumita adresa din memorie, si la linia 17, si la linia 22.

e. (0.4p) Ce tip de date au argumentele, stiind ca `movzbl` efectueaza un mov de la `byte` la `long`?
Raspuns: primul argument este un `long`, nu avem operatii sufixate specific pentru el, dar pentru al doilea utilizam `movzbl`, astfel ca ocupa un `byte`, iar de la raspunsul anterior stim ca este un pointer, astfel ca tipul lui este un pointer la byte, adica un `char*`.