Numele si prenumele (cu MAJUSCULE): _____ Grupa: _____

Test: _____ Tema: _____ Colocviu: _____ FINAL: _____

Test de laborator - Arhitectura Sistemelor de Calcul

ianuarie 2025

Seria 15

- Nota maxima pe care o puteti obtine este 10.
- Nota obtinuta trebuie sa fie minim 5 pentru a promova, fara nicio rotunjire superioara.
- Orice tentativa de frauda este considerata o incalcare a Regulamentului de Etica!

1 Partea 0x00: x86 - maxim 6p

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 cod. Analizati-l si raspundeti intrebarilor de mai jos. Pentru fiecare raspuns in parte, veti preciza si instructiunile care v-au ajutat in rezolvare.

```
08049176 <f>:
     17a: 55                    push   %ebp
     17b: 89 e5                 mov    %esp,%ebp
     17d: 83 ec 10              sub    $0x10,%esp
     185: 05 7b 2e 00 00        add    $0x2e7b,%eax
     18a: c7 45 fc 01 00 00 00  movl   $0x1,-0x4(%ebp)
     191: c7 45 f8 00 00 00 00  movl   $0x0,-0x8(%ebp)
     198: eb 40                 jmp    1da <f+0x64>
     19a: 8b 45 f8              mov    -0x8(%ebp),%eax
     19d: 8d 14 85 00 00 00 00  lea    0x0(,%eax,4),%edx
     1a4: 8b 45 08              mov    0x8(%ebp),%eax
     1a7: 01 d0                 add    %edx,%eax
     1a9: 8b 00                 mov    (%eax),%eax
     1ab: 0f af 45 10           imul   0x10(%ebp),%eax
     1af: 89 c2                 mov    %eax,%edx
     1b1: 8b 45 14              mov    0x14(%ebp),%eax
     1b4: 0f af 45 18           imul   0x18(%ebp),%eax
     1b8: 39 c2                 cmp    %eax,%edx
     1ba: 7e 1a                 jle    1d6 <f+0x60>
     1bc: 8b 45 f8              mov    -0x8(%ebp),%eax
     1bf: 8d 14 85 00 00 00 00  lea    0x0(,%eax,4),%edx
     1c6: 8b 45 08              mov    0x8(%ebp),%eax
     1c9: 01 d0                 add    %edx,%eax
     1cb: 8b 00                 mov    (%eax),%eax
     1cd: 8b 55 fc              mov    -0x4(%ebp),%edx
     1d0: 0f af c2              imul   %edx,%eax
     1d3: 89 45 fc              mov    %eax,-0x4(%ebp)
     1d6: 83 45 f8 01           addl   $0x1,-0x8(%ebp)
     1da: 8b 45 f8              mov    -0x8(%ebp),%eax
     1dd: 3b 45 0c              cmp    0xc(%ebp),%eax
     1e0: 7c b8                 jl     19a <f+0x24>
     1e2: 8b 45 fc              mov    -0x4(%ebp),%eax
     1e6: c3                    ret
```

```
g00:    <g>:
g01:        pushl   %ebp
g02:        movl    %esp, %ebp
g03:        subl    $16, %esp
g04:        movl    8(%ebp), %eax
g05:        movss   (%eax), %xmm0
g06:        cvttss2sil      %xmm0, %eax
g07:        pushl   %eax
g08:        pushl   $5
g09:        pushl   $5
g0A:        pushl   12(%ebp)
g0B:        pushl   8(%ebp)
g0C:        call    f
g0D:        addl    $20, %esp
g0E:        movl    %eax, -12(%ebp)
g0F:        fldz
g10:        fstps   -8(%ebp)
g11:        movl    $0, -4(%ebp)
g12:        jmp     .L9
g13:    .L10:
g14:        flds    -8(%ebp)
g15:        movl    -4(%ebp), %eax
g16:        leal    0(,%eax,4), %edx
g17:        movl    8(%ebp), %eax
g18:        addl    %edx, %eax
g19:        flds    (%eax)
g1A:        fldl    .LC2
g1B:        fmulp   %st, %st(1)
g1C:        faddp   %st, %st(1)
g1D:        fstps   -8(%ebp)
g1E:        addl    $1, -4(%ebp)
g1F:    .L9:
g20:        movl    -4(%ebp), %eax
g21:        cmpl    12(%ebp), %eax
g22:        jl      .L10
g23:        flds    -8(%ebp)
g24:        ret
```

a. (0.75p) Cate argumente primeste procedura `f` si cum ati identificat acest numar de argumente?

Solution: sunt 5 argumente - identificam un offset de 0x18(ebp) la 1b4; la fel, putem observa in calupul g07-g0B in procedura g ca sunt 5 argumente cand se apeleaza local f

b. (0.75p) Ce tip de date returneaza procedura `f` si cum ati identificat acest tip?

Solution: Urmarim ce se intampla cu registrul eax - ultima lui aparitie este la 1e2 cand se pune -0x4(ebp) in eax. urmarim acum -0x4(ebp) (prima variabila locala)
la 1d3 se pune eax, o valoare pe 32 de biti, si avem de la 1cb ca eax este un continut pe 32b de la o adresa de memorie, de unde conchidem ca se returneaza un intreg pe 32b, deci un .long

c. (0.75p) In timp ce analizati executabilul, observati in sectiunea `.data` valoarea `0x40B00000`. In timp ce cititi codul, va dati seama ca aceasta valoare este, de fapt, o reprezentare pe formatul `single` a unei valori rationale. Despre ce valoare este vorba?

Solution: 0x40B00000 = 0100 0000 1011 0000 0000 0000 0000 0000 de unde semnul este pozitiv, exponentul este 0b10000001 -127 = (2**7 + 1) - 127 = 2 iar mantisa 01100000000000000000000. Numarul este 2 * (1.0110). Analizam separat (1.0110) in baza 2, care este 1 + (.0110) in baza 2, adica 0*2**(-1) + 1 * 2**(-2) + 1 * 2**(-3) + ..., ceea ce inseamna ca este 0.375 in baza 10, deci 1.M este 1.375, iar rezultatul final este 2**2 * 1.375 = 5.5.

d. (0.75p) Va atrage atentia codificarea hexa a instructiunilor, si doriti sa vedeti care este semantica acestora. Specific, analizati instructiunile `mov` si observati ca ele difera destul de mult ca reprezentare, incat chiar si `opcode`-ul difera intre doua instructiuni `mov`. Stiind ca aceste `opcode`-uri sunt `89`, `c7` si `8b`, identificati cand se foloseste fiecare. In cazul codificarii `c7` analizati ultimii 4 Bytes din fiecare reprezentare - ce semnifica acea valoare?

Solution: Analizam unde avem opcode 89: la 17b, 1af, 1d3: in toate aceste cazuri, sursa este un registru, iar destinatia este sau registru, sau adresa de memorie.
Analizam unde avem opcode 8b: 19a, 1a4, 1a9, 1b1, 1bc, 1c6, 1cb, 1cd, 1da, iar in toate aceste situatii avem o adresa de memorie care se copiaza intr-un registru.
In cazul c7, avem mov cu valoare imediata intr-un registru. Ultimii 4Bytes din reprezentare ne dau valoarea imediata (in little endian).

e. (0.5p) Procedura `f` contine o structura repetitiva. Identificati toate elementele acestei structuri: initializarea contorului, conditia de a ramane in structura, respectiv pasul de continuare (operatia asupra contorului).

Solution: Identificam structura repetitiva dupa saltul inapoi, de la 1e0 la 19a. Analizam ce se intampla la fiecare instructiune.
19a: variabila locala -0x8(ebp) se copiaza in eax, ca la 19d sa se ia adresa de memorie din continutul de la 4*eax si sa se mute in edx, care ulterior este utilizat la 1a4, 1a7 si 1a9 pentru a accesa continutul de la 0x8(ebp) (primul argument) + 4*eax, ceea ce indica accesarea unui element dintr-un array. Ne dam seama de aici ca 0x8(ebp) este adresa unui array, deci f este sigur parametrizata de un *v in primul argument).
Pentru ca nu are sens la fiecare pas sa incarcam acelasi lucru, fiind in array, cautam un pas de incrementare, si il gasim la 1d6, se adauga 1 la -0x8(ebp), care intra in calculul adresei de mai sus. Ne uitam in exteriorul structurii (inainte de adresa la care facem salt, 19a), si gasim ca la 191 se face initializarea cu 0 a lui -0x8(ebp). Avem, deci, initializarea unui contor cu 0 si o incrementare a acestui contor. Identificam cand ramane in structura: facem salt prin jl 19a, urmarim compare-ul: cmp 0xc(ebp), eax. Citim instructiunea natural: daca eax lt 0xc(ebp), facem salt, deci ramanem in structura. 0xc(ebp) este al doilea argument. eax aici este facut la 1da -0x8(ebp), adica e contorul, astfel ca avem conditia de continuare ca i lt arg2. In final, arata i = 0; i lt arg2; i++

f. (1p) Analizati acum procedura `g`. Primul lucru pe care il observati sunt instructiunile specifice pentru a lucra cu `floating point`. Identificati `fldz` care incarca 0 peste stiva FPU (in %st(0)), `fildl op` care incarca intregul op ca `float` pe stiva FPU (in %st(0)), `fmulp op1, op2` care efectueaza pe formatul `float` operatia `op2 := op2 * op1` si apoi efectueaza `pop`, `faddp op1, op2` care efectueaza pe formatul `float op2 := op2 + op1` (in cazul `faddp %st, %st(1)` efectueaza `%st(1) += %st(0)` si efectueaza `pop`). Avand aceste informatii, determinati care este structura repetitiva si ce se calculeaza in acea structura.

Solution: In primul rand, identificam structura repetitiva, cautand salturi inapoi. Gasim respectivul salt inapoi la g22, unde se face salt la .L10. Vrem sa identificam contorul, conditia de a ramane in structura, pasul de continuare, respectiv ce vrem sa calculam. Conditia de continuare e cea de la g21-g22, care ne spune ca ramanem in structura cat timp eax lt 12(ebp), deci cat timp eax lt arg2, unde eax este -4(ebp), iar -4(ebp) este o variabila locala incrementata la g1E. Urmarm -4(ebp) in program: mai este utilizat intr-un calcul la g15, si este facut 0 la inceput, la g11, in exteriorul structurii. Putem conchide in acest punct ca -4(ebp) este un contor i, iar structura are forma unui for i = 0; i lt arg2, i++.
Ne intereseaza acum ce calculam: structura incepe la g14 cu incarcarea pe stiva FPU a lui -8(ebp).
Urmarim cine este -8(ebp). g0F si f10 ne spun ca -8(ebp) este initial 0. (se incarca 0 pe stiva FPU la g0F, iar la g10 este stocat in -8(ebp) si i se pop 0-ului).
g14: se incarca o variabila pe stiva FPU, initial 0 g15: se incarca contorul in eax
pana la linia g18 aflam adresa elementului curent din array
g19: elementul curent din array este incarcat pe stiva FPU
g1A: incarcam pe stiva FPU o valoare din memorie aflata la .LC2
avem pe stiva FPU .LC2, element curent, o variabila initial 0
g1B: se inmultesc varful stivei cu elementul curent din array, deci .LC2 * v[i], se face pop
g1C: se adauga rezultatul la variabila pe stiva corespunzatoare lui -0x8(ebp), se face pop, si apoi la g1D se stocheaza totul in -0x8(ebp), de unde conchidem ca in -0x8(ebp) avem o suma de element curent inmultit cu valoarea de la g1A, deci -0x8(ebp) += v[i] * .LC2

g. (0.5p) Considerati rescrierea instructiunilor pe stiva FPU din procedura `g` in SIMD. Care este echivalentul lor? (pentru liniile g14-g1E).

Solution:

```
g14: movss -8(%ebp), %xmm0
g15: movl -4(%ebp), %eax
g16: leal 0(,%eax, 4), %edx
g17: movl 8(%ebp), %eax
g18: addl %edx, %eax
g19: movss (%eax), %xmm1
g1A: movss .LC2, %xmm2
g1B: mulss %xmm2, %xmm1
g1C: addss %xmm1, %xmm0
g1D: movss %xmm0, -8(%ebp)
g1E: add $1, -4(%ebp)
```

h. (1p) Observati ca la linia g0C, din procedura `g` se face un *call* imbricat in procedura `f`. Reprezentati configuratia stivei de apel, in momentul in care se obtine adancimea maxima, considerand reprezentarea incepand cu argumentele primite de `g`.

Solution: Avem constructia din g. Avem 2 argumente, ra, salvarea lui ebp, un spatiu de 16B alocati pe stiva, corespunzator pentru 4 variabile locale. Se pregateste cadrul de apel pentru f, avem 5 argumente, ra, apoi in f: salvarea lui ebp, spatiu pentru 0x10=16B, corespunzator pentru 4 variabile locale. In acest punct se obtine adancimea maxima.

2 Partea 0x01: RISC-V - maxim 3p

a. (0.75p) Au totate instructiunile RISC-V o varianta comprimata? Daca da, explicati de ce. Daca nu, prezentati 2 situatii diferite in care instructiunile nu sunt comprimabile si motivati. De ce x86 nu are o astfel de extensie?

Solution: Nu, nu au toate. Un spatiu de 32 de biti nu poate fi redus complet la 16. Exista anumite reguli asadar. Orice 2 exemple de instructiuni care incalca regulile sunt acceptate (instructiuni cu imediat prea mare, care nu au acceasi sursa si destinatie si nici nu folosesc registrii comprimabili). x86 are codificari variabile ca size, o intructiune ocupa atat de putin cat se poate (nu exista biti nevalorificati), deci nu se poate optimiza si mai mult size-ul.

b. (0.75p) Se considera un procesor RISC-V minimal (RV32I) care nu implementeaza niciun mecanism de securitate. Care este efectul codului de mai jos daca se ruleaza pe un astfel de procesor? Descrieti fiecare pas de la inceputul pana la finalul executiei (Mentionati la fiecare pas cum se modifica registrii).

```
.data
x: .long 0x00ee0333

.text
.global main
main:
la a0, x
jr a0

li a7, 93
li a0, 0
ecall
```

Solution: Se face salt in zona .data, se pune suma dintre t3 si a4 in t1 (decodarea instructiunii add t1, t3, a4) si apoi segfault.

c. (0.75p) Ce valoare va fi depozitata in `a0` in urma executiei urmatoarelor instructiuni, stiind ca `pc` este intial 0? Prezentati efectul fiecarei instructiuni.

```
auipc a0, 0x12345
slli a0, a0, 4
auipc a1, 0x1
add a0, a0, a1
```

Solution: 1: a0 = 0x12345000 → 2: a0 = 0x23450000 → 3: a1 = 0x00001008 → 4: a0 = 0x23451008

d. (0.75p) Se da urmatorul schelet de functie. Care este efectul sau?

```
proc:
addi sp, sp, -8
sw ra, 4(sp)
sw s0, 0(sp)
addi sp, sp, -8

// Cod care nu mai modifica valoarea lui sp

lw s0, 0(sp)
lw ra, 4(sp)
addi sp, sp, 16
ret
```

Solution: Incarcarea in ra (respectiv s0) de la final nu se mai face de pe locatiile corespunzatoare salvarilor lor (ar fi trebuit folositi offsetii 12, respectiv 8).

3 Partea 0x02: Performanta si cache - maxim 1p

a. (0.5p) Un procesor are un pipeline cu 5 stadii (Fetch, Decode, Execute, Memory, Write-back). Se considera urmatoarea secventa de instructiuni:

```
lw a0, 0(gp)
add a1, a0, a2
sub a3, a1, a0
```

Identificati hazardurile de date si tipul lor. Presupunem ca folosim mecanismul de forwarding pentru a le rezolva. Astfel pentru ca o instructiune dintr-un hazard sa poata intra in stadul de execute, intructiunea dependinta trebuie sa finalize stadiul de memory daca este o instructiune de accesare a memoriei, respectiv de execute daca este un alt tip de instructiune. In acest context, care este numarul total de cicluri in care se termina de executat intreaga secventa? (Pentru usurinta puteti realiza un tabel cu ciclurile.)

Solution: Exista 2 hazarduri (RAW) - intre instructiunile 1 si 2, respectiv 2 si 3 prin folosirea registrilor a0, respectiv a1. Este nevoie de 8 cicluri pentru terminarea secventei.

Ciclul	1	2	3	4	5	6	7	8
lw	F	D	E	M	WB			
add		F	D	S	E	M	WB	
sub			F	D	S	E	M	WB

b. (0.5p) Un sistem are o memorie principală de 2^{18} bytes iar cache-ul are o capacitate totală de 8 KB, cu o dimensiune a unui bloc de 128 bytes (atât pentru memoria principală, cât și pentru cache). Calculați numărul total de blocuri din memoria principală. Determinați numărul de linii (blocuri) din cache. În cazul unei scheme de mapare directă, unde va fi mapată adresa 0xA1F0 în cache?

Solution:

$$\frac{2^{18}}{128} = \frac{2^{18}}{2^7} = 2^{11} = 2048 \text{ blocuri in memoria principala}$$

$$\frac{2^3 * 2^{10}}{128} = \frac{2^{13}}{2^7} = 2^6 = 64 \text{ linii in cache}$$

Pentru adresa 0xA1F0 = 1010 0001 1111 0000:

- Offset - ultimii 7 biți (dimensiunea liniei e 2^7): 1110000 = 0x70
- Index - următorii 6 biți (dimensiunea cache-ului e 2^6): 000011 = 3
- Tag = cei 3 biți rămași: 101

Așadar 0xA1F0 va fi mapată în cache la linia 3, având offset-ul 0x70 (word-ul 28).