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

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

Test de laborator - Arhitectura Sistemelor de Calcul

ianuarie 2025

Seria 13

- 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.

```
000011ad <f>:
    11b1: 55                    push   %ebp
    11b2: 89 e5                 mov    %esp,%ebp
    11b4: 83 ec 10              sub    $0x10,%esp
    11c1: c7 45 fc 00 00 00 00  movl   $0x0,-0x4(%ebp)
    11c8: c7 45 f8 00 00 00 00  movl   $0x0,-0x8(%ebp)
    11cf: eb 4b                 jmp    121c <f+0x6f>
    11d1: 8b 45 f8              mov    -0x8(%ebp),%eax
    11d4: 8d 14 85 00 00 00 00  lea    0x0(,%eax,4),%edx
    11db: 8b 45 08              mov    0x8(%ebp),%eax
    11de: 01 d0                 add    %edx,%eax
    11e0: 8b 00                 mov    (%eax),%eax
    11e2: 39 45 10              cmp    %eax,0x10(%ebp)
    11e5: 7f 31                 jg     1218 <f+0x6b>
    11e7: 8b 45 f8              mov    -0x8(%ebp),%eax
    11ea: 8d 14 85 00 00 00 00  lea    0x0(,%eax,4),%edx
    11f1: 8b 45 08              mov    0x8(%ebp),%eax
    11f4: 01 d0                 add    %edx,%eax
    11f6: 8b 00                 mov    (%eax),%eax
    11f8: 39 45 14              cmp    %eax,0x14(%ebp)
    11fb: 7c 1b                 jl     1218 <f+0x6b>
    11fd: 8b 45 f8              mov    -0x8(%ebp),%eax
    1200: 8d 14 85 00 00 00 00  lea    0x0(,%eax,4),%edx
    1207: 8b 45 08              mov    0x8(%ebp),%eax
    120a: 01 d0                 add    %edx,%eax
    120c: 8b 10                 mov    (%eax),%edx
    120e: 89 d0                 mov    %edx,%eax
    1210: c1 e0 02              shl    $0x2,%eax
    1213: 01 d0                 add    %edx,%eax
    1215: 01 45 fc              add    %eax,-0x4(%ebp)
    1218: 83 45 f8 02           addl   $0x2,-0x8(%ebp)
    121c: 8b 45 f8              mov    -0x8(%ebp),%eax
    121f: 3b 45 0c              cmp    0xc(%ebp),%eax
    1222: 7c ad                 jl     11d1 <f+0x24>
    1224: 8b 45 fc              mov    -0x4(%ebp),%eax
    1228: c3                    ret
```

```
00001229 <g>:
    122d: 55                    push   %ebp
    122e: 89 e5                 mov    %esp,%ebp
    1230: 53                    push   %ebx
    1231: 83 ec 14              sub    $0x14,%esp
    123f: 6a 0a                 push   $0xa
    1241: 6a 00                 push   $0x0
    1243: ff 75 0c              pushl  0xc(%ebp)
    1246: ff 75 08              pushl  0x8(%ebp)
    1249: e8 5f ff ff ff        call   11ad <f>
    124e: 83 c4 10              add    $0x10,%esp
    1251: 89 45 f8              mov    %eax,-0x8(%ebp)
    1254: d9 ee                 fldz
    1256: d9 5d f4              fstps  -0xc(%ebp)
    1259: c7 45 f0 00 00 00 00  movl   $0x0,-0x10(%ebp)
    1260: eb 26                 jmp    1288 <g+0x5f>
    1262: 8b 45 f0              mov    -0x10(%ebp),%eax
    1265: 8d 14 85 00 00 00 00  lea    0x0(,%eax,4),%edx
    126c: 8b 45 08              mov    0x8(%ebp),%eax
    126f: 01 d0                 add    %edx,%eax
    1271: 8b 00                 mov    (%eax),%eax
    1273: 89 45 e8              mov    %eax,-0x18(%ebp)
    1276: db 45 e8              fildl  -0x18(%ebp)
    1279: d8 4d 10              fmuls  0x10(%ebp)
    127c: d9 45 f4              flds   -0xc(%ebp)
    127f: de c1                 faddp  %st,%st(1)
    1281: d9 5d f4              fstps  -0xc(%ebp)
    1284: 83 45 f0 01           addl   $0x1,-0x10(%ebp)
    1288: 8b 45 f0              mov    -0x10(%ebp),%eax
    128b: 3b 45 0c              cmp    0xc(%ebp),%eax
    128e: 7c d2                 jl     1262 <g+0x39>
    1290: d9 45 f4              flds   -0xc(%ebp)
    1293: 8b 5d fc              mov    -0x4(%ebp),%ebx
    1297: c3                    ret
```

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

Solution: 4 argumente - identificam un 0x14(%ebp) ca offset maxim, de unde conchidem ca exista argumentele aflate la offset-urile 0x8, 0xc, 0x10, 0x14 relativ la %ebp. Alta varianta: in procedura g este apelata procedura f, si vedem exact constructia cadrului de apel

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

Solution: Urmarim ce se stocheaza in %eax: ultima aparitie este mutarea valorii din -0x4(%ebp), analizam ce se intampla cu -0x4(%ebp): avem un add %eax, -0x4(%ebp), de unde conchidem ca intoarce un intreg pe 32b, deci un .long.

c. (0.75p) In timp ce analizati executabilul, observati in sectiunea `.data` valoarea `0x40200000`. 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: 0x40200000 = 0100 0000 0010 0000 0000 0000 0000 0000 de unde semnul este pozitiv, exponentul este 0b10000000 -127 = 2**7 - 127 = 1 iar mantisa 01000000000000000000000. Numarul este 2 * (1.010). Analizam separat (1.010) in baza 2, care este 1 + (.010) in baza 2, adica 0*2**(-1) + 1 * 2**(-2) + 0 * 2**(-3) + ..., ceea ce inseamna ca este 0.25 in baza 10, deci 1.M este 1.25, iar rezultatul final este 2 * 1.25 = 2.5.

d. (0.75p) Va atrage atentia codificarea hexa a instructiunilor, si doriti sa vedeti care este semantica acestora. Analizati instructiunea `jl`. Stiind ca `0x1b = 27`, respectiv `0xad = -83`, determinati regula reprezentarii acestei instructiuni. Numim aceasta instructiune ca fiind de salt scurt, de ce?

Solution: Observam ca jl este de forma 7c zz, cautam semantica lui zz. Fiind vorba de salt, ne gandim la salt pe adrese. Daca ne uitam la 11fb, 7c 1b face salt la 1218, care este la 1b = 27B distanta de instructiunea curenta. Daca analizam adresa 1222, 7c ad face salt la 11d1, care este la 0xad = -83B distanta. Conchidem ca jl ¡label¿ se codifica 7c zz, unde zz reprezinta offsetul la care ne deplasam relativ la instructiunea curenta. Conchidem de aici si ca maximul de salt este la offset 0x7F = 127 (respectiv -128), de unde numele de short jump.

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: Am identificat salturile inapoi la pasul anterior, astfel ca ne gandim ca structura repetitiva este data de salturile inapoi. Avem la adresa 1222 acel jl 11d1, astfel ce urmarim ce se intampla de la 11d1: se pune o variabila locala in %eax, variabila care este initial 0, conform 11c8. Observam aici si ca variabila locala de la -0x4(%ebp) este tot cu 0 initializata.
Luam pe rand ce se intampla:
11d1: se pune variabila locala in %eax, initial 0
11d4: se incarca adresa de la %eax + 4 in %edx
11db: se incarca primul argument in %eax
11de: se adauga %edx la %eax
11e0: se ia elementul de la adresa lui %eax: de aici tragem concluzia ca am incarcat elementul curent dintr-un array in %eax.
11e0: se compara elementul curent din array cu al treilea argument (0x10(%ebp))
11e2: daca argumentul 3 este mai mare decat %eax, se merge la 1218, unde se incrementeaza -0x8(%ebp) cu 2, de unde conchidem ca joaca un rol de index care creste din 2 in 2, pentru ca imediat dupa se verifica daca indexul este mai mic strict (lt) decat argumentul 2, daca da se ramane in structura, altfel se iese.
De aici putem conchide: initial, un contor este egal cu 0. Se executa cat timp contorul este lt arg2. incrementarea la fiecare pas este contor += 2.

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)), `fmuls op` care efectueaza pe formatul `float` operatia `%st(0) := %st(0) * op`, `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: Ca in cazul exercitiului anterior din procedura f, cautam un salt inapoi, care sa sugereze existenta unei structuri repetitive. Identificam la 128e ca se face salt inapoi la 1262, astfel ca analizam codul din acel punct.
1262: se incarca o variabila locala in %eax, variabila care este initial 0, apoi gasim aceeasi structura care incarca, de fapt, in %eax, elementul curent dintr-un array, la adresa 1271. De aici urmeaza partea de float.
se muta elementul curent in variabila locala -0x18(%ebp), care este convertita la float si pusa pe stiva (fildl -0x18(%ebp) la 1276)
se inmulteste valoarea cu argumentul 3 (fmuls 0x10(%ebp)) la 1279
se incarca o variabila locala pe stiva FPU la 127c, si gasim la 1256 ca -0xc(%ebp) este tot un float, initial 0 (fldz + fstps -0xc(%ebp))
se adauga rezultatul inmultirii anterioare v[i] * arg3 la aceasta valoare intermediara, iar apoi se descarca in -0xc(%ebp), deci tot in valoarea initial 0
inseamna ca avem un calcul de forma val := val + v[i] * arg3

g. (0.5p) Considerati rescrierea instructiunilor pe stiva FPU din procedura `g` in SIMD. Care este echivalentul lor? (pentru liniile dintre adresele de memorie 1276 - 1281).

Solution:

```
mov $0, %eax
cvtsi2ss %eax, %xmm0
cvtsi2ss -0x18(%ebp), %xmm1
mov 0x10(%ebp), %eax
cvtsi2ss %eax, %xmm2
mulss %xmm2, %xmm1
addss %xmm1, %xmm0
```

Orice alta solutie corecta este acceptata.

h. (1p) Observati ca la adresa 1249, 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: Se presupune constructia din g, plecam cu cele trei argumente din g, se pun ebp, ebx, se face spatiu pentru $20/4 = 5$ variabile locale, apoi cele 4 argumente pentru f; se continua cu f: spatiu pentru ebp, spatiu pentru $16/4 = 4$ variabile locale, de unde reiese fix configuratia de adancime maxima.

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

a. (0.75p) De ce RISC-V are un numar mult mai mare de registrii decat x86? Oferiti macar 2 motivatii.

Solution: Instructiunile pot opera doar pe registrii (arhitectura load-store) si este nevoie de mai multi registri intermediari pentru a realiza un calcul simplu (nu exista instructiuni complexe ca pe x86). Nu exista de asemenea mai multe variante ale aceluiasi registru (al, ah, ax, eax) etc (Orice alte 2 raspunsuri corecte se puncteaza).

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 0x00f28293

.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 15 in t0 (decodarea instructiunii addi t0, t0, 15) 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
srli a0, a0, 4
auipc a1, 0x10000
add a0, a0, a1
```

Solution: 1: a0 = 0x12345000 → 2: a0 = 0x01234500 → 3: a1 = 0x10000008 → 4: a0 = 0x11234508

d. (0.75p) Vrem sa facem salt in functia `func` folosind instructiunea `jal a7, func`. Ce modificari ar trebui aduse functiei astfel incat revenirea din functie sa se realizeze cu succes?

Solution: a7 contine acum pc + 4, deci va trebui sa se salveze a7 pe stiva in loc de ra (orice alt raspuns similar e acceptat, de exemplu sa se faca la final jr a7 si sa nu se modifice a7 pe parcurs).

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
sw a1, 4(gp)
```

Identificati hazardurile de date si tipul lor. Presupunem ca folosim stall-uri pentru a le rezolva (intructiunea dependinta trebuie sa finalize stadiul de write-back pentru ca instructiunea curenta sa poata intra in stadiul de execute). 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 11 cicluri pentru terminarea secventei.

Ciclul	1	2	3	4	5	6	7	8	9	10	11
lw	F	D	E	M	WB						
add		F	D	S	S	E	M	WB			
sw			F	D	S	S	S	S	E	M	WB

b. (0.5p) Un sistem are o memorie principală de 2^{16} bytes iar cache-ul are o capacitate totală de 4 KB, cu o dimensiune a unui bloc de 64 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 0x2F3C în cache?

Solution:

$$\frac{2^{16}}{64} = \frac{2^{16}}{2^6} = 2^{10} = 1024 \text{ blocuri in memoria principala}$$

$$\frac{2^2 * 2^{10}}{64} = \frac{2^{12}}{2^6} = 2^6 = 64 \text{ linii in cache}$$

Pentru adresa 0x2F3C = 0010 1111 0011 1100:

- Offset - ultimii 6 biți (dimensiunea liniei e 2^6): 111100 = 0x3c
- Index - următorii 6 biți (dimensiunea cache-ului e 2^6): 111100 = 60
- Tag = cei 4 biți rămași: 0010

Așadar 0x2F3C va fi mapată în cache la linia 60, având offset-ul 0x3c (word-ul 15).