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

Test: _____ Activitate: _____ Colocviu: _____ FINAL: _____

Test de laborator - Arhitectura Sistemelor de Calcul

Grupele 131, 132, 133, 134, 151

17 ianuarie 2026

Varianta B

- Nota maximă pe care o puteţi obţine este 10.
- Nota obţinută trebuie să fie minim 5 pentru a promova, fără nicio rotunjire superioară.
- Orice tentativă de fraudă este considerată o încălcare a Regulamentului de Etică!

1 Partea 0x00: x86 (3 puncte)

1.1 Exerciţiul 0x00 (1.5 puncte)

Presupunem că aveţi acces la un executabil `exec`, pe care îl inspectaţi cu `objdump -d exec`. În momentul în care rulaţi această comandă, vă opriţi asupra următorului cod. Analizaţi-l şi răspundeţi întrebărilor de mai jos. Pentru fiecare răspuns în parte, veţi preciza şi instrucţiunile (indicând numărul liniilor) care v-au ajutat în rezolvare. Formatul de mai jos este *număr linie: adresă: instrucţiune*.

```
08049166 <f>:
 0x00: 8049166:       push   %ebp
 0x01: 8049167:       mov    %esp,%ebp
 0x02: 8049169:       sub    $0x10,%esp
 0x03: 8049176:       movl   $0x0,-0x8(%ebp)
 0x04: 804917d:       movl   $0x0,-0x4(%ebp)
 0x05: 8049184:       jmp    80491ae <f+0x48>
 0x06: 8049186:       mov    -0x4(%ebp),%eax
 0x07: 8049189:       cmp    0x10(%ebp),%eax
 0x08: 804918c:       jle    80491aa <f+0x44>
 0x09: 804918e:       mov    -0x4(%ebp),%eax
 0x0a: 8049191:       cmp    0x14(%ebp),%eax
 0x0b: 8049194:       jg     80491aa <f+0x44>
 0x0c: 8049196:       mov    -0x4(%ebp),%eax
 0x0d: 8049199:       lea    0x0(,%eax,4),%edx
 0x0e: 80491a0:       mov    0xc(%ebp),%eax
 0x0f: 80491a3:       add    %edx,%eax
 0x10: 80491a5:       mov    (%eax),%eax
 0x11: 80491a7:       add    %eax,-0x8(%ebp)
 0x12: 80491aa:       addl   $0x1,-0x4(%ebp)
 0x13: 80491ae:       mov    -0x4(%ebp),%eax
 0x14: 80491b1:       cmp    0x8(%ebp),%eax
 0x15: 80491b4:       jl     8049186 <f+0x20>
 0x16: 80491b6:       mov    -0x8(%ebp),%eax
 0x17: 80491ba:       ret
```

a. (0.5p) Câte argumente primeşte procedura `f` şi cum aţi identificat acest număr de argumente?

Solution: are patru argumente, situate la 0x8, 0xc, 0x10 si 0x14 relativ la ebp, cu indexul maxim identificat la linia 0x0a. se acorda 0.3p pentru raspuns gresit (1, 2 sau 3 argumente, dar daca liniile s-au precizat corect. altfel, 0p sau 0.5p)

b. (0.5p) Ce tip de date returnează procedura `f` şi cum aţi identificat acest tip?

Solution: se urmareste ultima aparitie a lui eax - apare la 0x16 intr-un mov cu sursa in -0x8(ebp). [0.2p] urmarim -0x8(ebp) si observam ca apare in instructiuni pe long. Conchidem, astfel, ca tipul returnat este un .long. [0.3p]

c. (0.5p) Procedura `f` conţine o structură repetitivă. Identificaţi toate elementele acestei structuri: iniţializarea contorului, condiţia de a rămâne în structură, respectiv pasul de continuare (operaţia asupra contorului).

Solution: Identificam structura repetitiva prin salt inapoi: la 0x15 se face salt la 8049186 (linia 0x06) [0.2p], iar saltul este conditionat, cu cmp-ul la 0x14: daca eax lt 0x8(ebp) (=arg1) [0.1p]. urmarim ce scop are eax in cadrul buclei repetitive: de la 0x13 are valoarea lui -0x4(ebp), adica valoarea unei variabile locale. observam ca -0x4(ebp) este incrementat la 0x12, si initial are valoarea 0 pusa la 0x04. conchidem, deci, ca structura repetitiva are forma i = 0; i lt arg1; i++. [0.2p] dupa forma operatiilor, arg1 este dimensiunea unui array.

1.2 Exerciţiul 0x01 (1.5 puncte)

În acest exerciţiu vreţi să calculaţi şi să afişaţi valoarea lui $\sin\left(\frac{\pi}{2}\right)$. Consideraţi că aveţi următoarea secţiune `.data`:

```
pi: .float 3.1415926535
y: .space 4
formatPrintf: .asciz "sin(pi/2) = %f\n"
```

Pentru a calcula valoarea $\sin\left(\frac{\pi}{2}\right)$, veţi apela procedura `sinf(x)` din `libmath`, care returnează valoarea conform convenţiilor `FPU`. Pentru calculul $\frac{\pi}{2}$ veţi utiliza instrucţiunea `divss` din `SIMD`. Scrieţi secvenţa de instrucţiuni din `main` care calculează $\frac{\pi}{2}$, aplică funcţia sin, respectiv afişează rezultatul la `stdout`, printr-un apel al procedurii `printf`.

Solution:

```
movss pi, %xmm0
movl $2, %ecx
cvtsi2ss %ecx, %xmm1
divss %xmm1, %xmm0      # 0.5p

sub $4, %esp
movss %xmm0, 0(%esp)
call sinf
add $4, %esp
fstps y                 # 0.5p

movss y, %xmm0
cvtss2sd %xmm0, %xmm0
sub $12, %esp
movl $formatPrintf, 0(%esp)
movsd %xmm0, 4(%esp)
call printf
add $12, %esp           # 0.5p
```

2 Partea 0x01: RISC-V (3 puncte)

2.1 Exerciţiul 0x00 (1 punct)

Scrieţi o procedură `sumFixedPoints` în RISC-V care primeşte ca argumente adresa unui *array* de întregi şi dimensiunea acestuia şi calculează suma elementelor din *array* care sunt egale cu indexul pe care se afla (elementele `v[i]` cu proprietatea că `v[i] == i`, cu indexarea elementelor de la 0). În această procedură veţi utiliza exclusiv regiştrii `s0-s11` pentru valorile suplimentare necesare, şi nu veţi utiliza niciun registru dintre `t0-t6`. Reprezentaţi stiva în momentul în care are adâncimea maximă.

Solution:

```
sumFixedPoints:
  addi sp, sp, -8
  sw ra, 4(sp)
  sw s0, 0(sp)
  add s0, sp, x0

  addi sp, sp, -12
  sw s1, 8(sp)
  sw s2, 4(sp)
  sw s3, 0(sp)

  li s1, 0
  li s2, 0
sumFixedPointsLoop:
  beq s1, a1, sumFixedPointsExit
  lw s3, 0(a0)
  beq s3, s1, isFixedPoint
sumFixedPointsCont:
  addi s1, s1, 1
  addi a0, a0, 4
  j sumFixedPointsLoop
```

```
isFixedPoint:
  add s2, s2, s3
  j sumFixedPointsCont

sumFixedPointsExit:
  add a0, s2, x0
  lw s3, -12(s0)
  lw s2, -8(s0)
  lw s1, -4(s0)
  lw ra, 4(s0)
  lw s0, 0(s0)
  addi sp, sp, 20
  jr ra
```

2.2 Exerciţiul 0x01 (1 punct)

Un sistem de calcul pe 32b are o memorie principală de 2^{17}`B` şi o memorie cache de `8 KiB`, împărţite pe blocuri de dimensiune `256 B`. Determinaţi numărul de blocuri din memoria principală, respectiv numărul de blocuri din memoria cache. Determinaţi unde se mapează, în cache, variabila de la adresa de memorie `0xB0F1`.

Solution: Mem. principala: 2**(17) : 2**8 = 2**9 blocuri; Cache: 2**10 * 2**3 : 2**8 = 2**5 blocuri in cache. Adresa 0xB0F1 = 1011 0000 1111 0001, de unde extragem tag, index, offset. Avem offset = ultimii 8b, adica 11110001 (0xF1), index urmatorii 5, adica 10000 (0x10), iar tag = 101 (0x5).

2.3 Exerciţiul 0x02 (1 punct)

Scrieţi, în **hexa**, codificarea instrucţiunii `auipc t0, 0x5`, ştiind că `opcode` este `0x17`, iar instrucţiunea este reprezentată în clasa `U`.

Solution: Formatul clasei U este urmatorul: imm [31:12], rd, opcode. Stim deja opcode = 0x17, dar pe 7 biti, deci 0010111. Registrul destinatie, rd, este t0, care are pozitia 5 in tabel, adica este 101 pe 5 biti, deci 00101. Ramane reprezentarea valorii immediate, care este 0x5, deci 5 pe bitii ramasi, deci 0000 0000 0000 0000 0101. Codificarea este, astfel, 0000 0000 0000 0000 0101 00101 0010111, iar dupa gruparea cate 4 se obtine in hexa 0x00005297.

3 Partea 0x02: Întrebări generale (3 puncte)

Alegeţi varianta corectă sau răspundeţi pe scurt.

0x00 (0.25p) În `x86` putem reveni dintr-o procedură în orice punct al acesteia, independent de configuraţia stivei.

a. Adevărat b. Fals

0x01 (0.50p) Este necesară salvarea `ra` pe stivă (în `RISC-V`) în scrierea procedurilor, ştiind că acest mecanism nu este implicit, aşa cum este în arhitectura `x86`?

Solution: Da, pentru ca altfel nu am putea implementa apelurile imbricate/recursive.

0x02 (0.50p) Precizaţi prin ce diferă revenirea dintr-o procedură în `x86` faţă de cea din `RISC-V`.

Solution: In x86 ret revine la adresa din varful curent al stivei, in timp ce in RISC-V se revine la valoarea din ra.

0x03 (0.75p) În analiza unui executabil, identificaţi următoarea valoare pe formatul `single`, `x = 0xC47FB500`. Ce număr îi corespunde în baza 10?

Solution: 0xC47A2500 are codificarea binara 1 10001000 11111111011010100000000. Este un numar negativ. Identificam exponentul, 10001000 = 2**7 + 2**3 = 136. 136 - 127 = 9, deci exponentul este 9. Avem ca numarul in forma stiintifica este 1.11111111011010100000000 * 2**9, deci forma initiala era 1111111110.11010100000000. Transformam partile intreaga/fractionara: 1111111110 este 1022, iar 110101 este 2**(-1) + 2**(-2) + 2**(-4) + 2**(-6) = 1/2 + 1/4 + 1/16 + 1/64 = (32 + 16 + 4 + 1)/64 = 53/64 = 0.828125. Astfel, numarul este -1022.828125.

0x04 (0.25p) Consideraţi instrucţiunea `addi` din `RISC-V` (instrucţiune a clasei `I`). Care este intervalul de valori `immediate` pe care le poate primi ca argument această instrucţiune?

a. [-2047, 2048] b. [0, 4095] c. [-4094, 4095] d. [0, 0xFFFFFF] e. [-2048, 2047]

Solution: e

0x05 (0.75p) Fie următoarele secvenţe de cod, în stânga în `x86`, iar în dreapta în `RISC-V`. Ştiind că, în ambele cazuri, adresa etichetei `main` este `0xAF0AB014`, precizaţi care este adresa instrucţiunii indicată de eticheta `et`. Cum aţi identificat? **Hint:** toate informaţiile necesare pentru rezolvarea exerciţiului se află în subiectele acestui colocviu.

```
        (x86)
main:
    mov -0x4(%ebp), %eax
et:
    sub $1, %ebx
```

```
        (RISC-V)
main:
    add s0, sp, x0
et:
    addi sp, sp, -4
```

Solution: Pentru RISC-V, stim ca adresele cresc din 4 in 4, deci raspunsul este 0xAF0AB018. Pentru x86 trebuie sa identificam cat ocupa instructiunea. O identificam in executabilul de la partea 0x00, la linia 0x0c, si observam ca ocupa spatiul de la 0x8049196 pana la 0x8049199, deci 0x3, astfel ca adresa pentru et este 0xAF0AB017.

Important! Puteţi folosi spaţiul rămas mai jos pentru a scrie rezolvările altor subiecte, pentru care nu v-a ajuns spaţiul alocat anterior. Marcaţi fiecare completare prin textul *Continuarea părţii . . . , exerciţiul . . . .*