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 A

- 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    80491a6 <f+0x40>
0x06: 8049186:       mov    -0x8(%ebp),%eax
0x07: 8049189:       lea    0x0(,%eax,4),%edx
0x08: 8049190:       mov    0x8(%ebp),%eax
0x09: 8049193:       add    %edx,%eax
0x0a: 8049195:       mov    (%eax),%eax
0x0b: 8049197:       cmp    %eax,0x10(%ebp)
0x0c: 804919a:       jge    80491a2 <f+0x3c>
0x0d: 804919c:       mov    -0x8(%ebp),%eax
0x0e: 804919f:       add    %eax,-0x4(%ebp)
0x0f: 80491a2:       addl   $0x1,-0x8(%ebp)
0x10: 80491a6:       mov    -0x8(%ebp),%eax
0x11: 80491a9:       cmp    0xc(%ebp),%eax
0x12: 80491ac:       jl     8049186 <f+0x20>
0x13: 80491ae:       mov    -0x4(%ebp),%eax
0x14: 80491b2:       ret
```

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

> **Solution:** are trei argumente, aflate la 0x8, 0xc, respectiv 0x10 relativ la ebp (indexul maxim la 0x0b). se acorda 0.3p pentru raspuns gresit (1 sau 2 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 0x13 intr-un mov cu sursa in -0x4(ebp). [0.2p] urmarim -0x4(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 0x12 se face salt la 8049186 (linia 0x06) [0.2p], iar saltul este conditionat, cu cmp-ul la 0x11: daca eax lt 0xc(ebp) (=arg2) [0.1p]. urmarim ce scop are eax in cadrul buclei repetitive: de la 0x10 are valoarea lui -0x8(ebp), adica valoarea unei variabile locale. observam ca -0x8(ebp) este incrementat la 0x0f, si initial are valoarea 0 pusa la 0x03. conchidem, deci, ca structura repetitiva are forma i = 0; i lt arg2; i++. [0.2p] dupa forma operatiilor, arg2 este dimensiunea maxima a unui array.

1.2 Exerciţiul 0x01 (1.5 puncte)

În acest exerciţiu vreţi să calculaţi şi să afişaţi pătratul radicalului de ordin 3 al unui număr `float` dat. Consideraţi că aveţi următoarea secţiune `.data`:

```
x: .float 27
y: .space 4
p: .float 0.333333333333
formatPrintf: .asciz "sqrt3(x)*sqrt3(x) := %f\n"
```

Pentru a calcula radicalul de ordin 3, veţi apela procedura `powf(x, p)` din `libmath`, care returnează valoarea conform convenţiilor `FPU`. Pentru ridicarea la pătrat, veţi utiliza instrucţiunea `mulss` din `SIMD`. Scrieţi secvenţa de instrucţiuni din `main` care calculează radicalul de ordin 3, efectuează ridicarea la pătrat, respectiv afişează rezultatul la `stdout`, printr-un apel al procedurii `printf`.

Solution:

```
movss x, %xmm0
movss p, %xmm1

sub $8, %esp
movss %xmm0, 0(%esp)
movss %xmm1, 4(%esp)
call powf
add $8, %esp        # 0.5p

fstps y

movss y, %xmm0
mulss %xmm0, %xmm0  # 0.5p

sub $12, %esp
movl $formatPrintf, 0(%esp)
cvtss2sd %xmm0, %xmm0
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 `productNotFixedPoints` în RISC-V care primeşte ca argumente adresa unui *array* de întregi şi dimensiunea acestuia şi calculează produsul elementelor din *array* care nu 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:

```
productNotFixedPoints:
  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, 1
productNotFixedPointsLoop:
  beq s1, a1, productNotFixedPointsExit
  lw s3, 0(a0)
  bne s3, s1, isNotFixedPoint
```

```
productNotFixedPointsCont:
  addi s1, s1, 1
  addi a0, a0, 4
  j productNotFixedPointsLoop

isNotFixedPoint:
  mul s2, s2, s3
  j productNotFixedPointsCont

productNotFixedPointsExit:
   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)

Determinaţi numărul de cicli de rulare pentru următoarele instrucţiuni, ştiind că toate hazardurile de date care apar sunt rezolvate prin *stall*-uri. Procesorul considerat are un pipline cu 5 stadii (Fetch, Decode, Execute, Memory si Write-Back).

```
lw t0, 4(gp)
lw s1, 12(gp)
add t0, t0, s1
sw t0, 0(gp)
```

Solution: 12 cicli

2.3 Exerciţiul 0x02 (1 punct)

Un sistem de calcul pe 32b execută în 20% dintre cazuri instrucţiuni aritmetice, în 30% dintre cazuri efectuează doar accesări de memorie, iar în rest execută instrucţiuni cu *floating point*. Ştim că o instrucţiune aritmetică necesită 3 cicli, o accesare de memorie 5 cicli, iar o instrucţiune pe *floating point* necesita 4 cicli şi o accesare suplimentară de memorie. Un alt sistem de calcul pe 32b execută aceleaşi instrucţiuni şi cu aceeaşi frecvenţă: o instrucţiune aritmetică necesită 3 cicli, o accesare de memorie 3 cicli, iar o instrucţiune cu *floating point* 10 cicli, dar fără altă accesare suplimentară de memorie. Determinaţi, pentru fiecare sistem în parte, măsurile CPI şi IPC. Care dintre cele două este mai performant?

Solution: $CPI_1 := 0.2 \cdot 3 + 0.3 \cdot 5 + 0.5 \cdot (4 + 5) = 0.6 + 1.5 + 4.5 = 6.6; IPC_1 := \frac{1}{CPI_1} = \frac{1}{6.6} \approx 0.1515;$ $CPI_2 := 0.2 \cdot 3 + 0.3 \cdot 3 + 0.5 \cdot 10 = 0.6 + 0.9 + 5 = 6.5; IPC_2 := \frac{1}{CPI_2} = \frac{1}{6.5} \approx 0.1538$. Sistemul de calcul 2 este mai performant.

3 Partea 0x02: Întrebări generale (3 puncte)

Alegeţi varianta corectă sau răspundeţi pe scurt.

0x00 (0.25p) În `RISC-V`, două codificări diferite ale instrucţiunilor garantează şi un comportament diferit al acestora.

a. Adevărat b. Fals

0x01 (0.50p) Precizaţi diferenţa între o instrucţiune `b label` din clasa B şi o instrucţiune `j label` din clasa J, ştiind că atât `b` cât şi `j` efectuează un salt la eticheta `label` indicată.

Solution: Salturi scurte in B, salturi lungi in J.

0x02 (0.50p) Precizaţi prin ce diferă instrucţiunena `call` din `x86` de cea din `RISC-V`.

Solution: In x86 se pune return address pe stiva, in RISC-V se pune in registrul ra.

0x03 (0.75p) În analiza unui executabil, identificaţi următoarea valoare pe formatul `single`, `x = 0xC47A2500`. Ce număr îi corespunde în baza 10?

Solution: 0xC47A2500 are codificarea binara 1 10001000 11110100010010100000000. 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 1.11110100010010100000000 * 2**9, deci forma initiala era 1111101000.10010100000000. Transformam partile intreaga/fractionara: 1111101000 este 1000, iar 100101 este 2**(-1) + 2**(-4) + 2**(-6) = 1/2 + 1/16 + 1/64 = (32 + 4 + 1)/64 = 37/64 = 0.578125. Astfel, numarul este -1000.578125.

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

Solution: d

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 `0xAF0AB024`, 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 (%eax), %eax
et:
    add $2, %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 0xAF0AB028. Pentru x86 trebuie sa identificam cat ocupa instructiunea. O identificam in executabilul de la partea 0x00, la linia 0x0a, si observam ca ocupa spatiul de la 0x8049195 pana la 0x8049197, deci 0x2, astfel ca adresa pentru et este 0xAF0AB026.

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 ...*.