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

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

Test de laborator - Arhitectura Sistemelor de Calcul

Grupele 141, 142, 143, 144, 152
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    $0x14,%esp
 0x03: 8049176:       mov    0xc(%ebp),%eax
 0x04: 8049179:       mov    %al,-0x14(%ebp)
 0x05: 804917c:       movl   $0x0,-0x8(%ebp)
 0x06: 8049183:       movl   $0x0,-0x4(%ebp)
 0x07: 804918a:       jmp    80491a4 <f+0x3e>
 0x08: 804918c:       mov    -0x8(%ebp),%edx
 0x09: 804918f:       mov    0x8(%ebp),%eax
 0x0a: 8049192:       add    %edx,%eax
 0x0b: 8049194:       movzbl (%eax),%eax
 0x0c: 8049197:       cmp    %al,-0x14(%ebp)
 0x0d: 804919a:       jne    80491a0 <f+0x3a>
 0x0e: 804919c:       addl   $0x1,-0x4(%ebp)
 0x0f: 80491a0:       addl   $0x1,-0x8(%ebp)
 0x10: 80491a4:       mov    -0x8(%ebp),%edx
 0x11: 80491a7:       mov    0x8(%ebp),%eax
 0x12: 80491aa:       add    %edx,%eax
 0x13: 80491ac:       movzbl (%eax),%eax
 0x14: 80491af:       cmpb   $0,%al
 0x15: 80491b1:       jne    804918c <f+0x26>
 0x16: 80491b3:       mov    -0x4(%ebp),%eax
 0x17: 80491b7:       ret
```

a. (0.5p) Câte argumente primeşte procedura `f`?

> **Solution:** doua argumente, aflate la 0x8(ebp) si 0xc(ebp).

b. (0.5p) Care este tipul argumentelor identificate şi cum aţi identificat acest lucru? Instrucţiunea `movzbl` reprezintă o instrucţiune care face `mov` cu o conversie de la `byte` la `long`.

> **Solution:** identificam tipul acestora prin instructiunile in care apar. urmarim 0x8(ebp) si observam ca apare la 0x09, valoarea acestuia fiind stocata in eax. la aceasta valoare se adauga edx, care este initial -0x8(ebp), adica o variabila locala initial egala cu 0 si incrementata la fiecare pas (cum vedem la 0x0f), deci la valoarea din eax se adauga indexul curent. apoi se ia valoarea de la adresa eax+edx si se stocheaza in eax, cu o conversie de la byte la long, de unde conchidem ca primul argument reprezinta adresa unui sir de caractere. [0.25p] pentru 0xc(ebp), observam ca apare la 0x04, este stocat in eax, iar la 0x05 se salveaza al in -0x14(ebp) (intr-o variabila locala). acea variabila locala este utilizata iar la 0x0c, cand al (elementul curent al sirului) se compara cu respectiva variabila locala, din care s-a retinut tot cel mai nesemnificativ byte, de unde conchidem ca 0xc(ebp) (arg2) reprezinta un caracter (un byte). [0.25p]

c. (0.5p) Procedura `f` conţine o structură repetitivă. Identificaţi condiţia de ieşire din structură (oferiţi o explicaţie a condiţiei de ieşire, nu este suficient doar să precizaţi instrucţiunile în limbaj de asamblare).

Solution: Pentru a gasi conditia de iesire, cautam saltul inapoi, si il identificam la linia 0x15. Conditia vine din comparatia cu 0 a registrului al - daca al este diferit de 0, se face salt inapoi, deci se iese cand al este 0. Urmarim cine este al in acest context, si din calupul 0x11-0x13 conchidem ca este vorba despre elementul curent al unui sir de caractere, deci se face o parcurgere pana la finalul sirului (pana cand str[i] eq 0).

1.2 Exerciţiul 0x01 (1.5 puncte)

În acest exerciţiu vreţi să calculaţi şi să afişaţi valoarea lui $\Gamma\left(\frac{x}{2}\right)$ (extinderea funcţiei factorial pentru numerele complexe, definită prin $\Gamma(z) := \int_0^\infty t^{z-1}e^{-t}dt$). Consideraţi că aveţi următoarea secţiune `.data`:

```
x: .float 3
y: .space 4
formatPrintf: .asciz "tgammaf(x/2) = %f\n"
```

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

Solution:

```
movl $2, %eax
cvtsi2ss %eax, %xmm1
movss x, %xmm0
divss %xmm1, %xmm0          # 0.5p

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

sub $12, %esp
movl $formatPrintf, 0(%esp)
movss y, %xmm0
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 în RISC-V care să calculeze suma cifrelor pare ale unui număr. În această procedură veţi utiliza cel puţin **un registru** dintre `s1-s11` pentru valorile suplimentare necesare, şi nu doar `t0-t6`. Reprezentaţi stiva în momentul în care are adâncimea maximă (pentru programul vostru).

Solution:

```
sumEvenDigits:
  addi sp, sp, -8
  sw ra, 4(sp)
  sw s0, 0(sp)
  add s0, sp, x0
  addi sp, sp, -4
  sw s1, 0(sp)

  li s1, 0
  li t0, 10
  li t1, 2
sumEvenDigitsLoop:
  beqz a0, sumEvenDigitsLoopExit
```

```
  rem t2, a0, t0
  rem t3, t2, t1
  beqz t3, isEvenDigit

sumEvenDigitsLoopCont:
  div a0, a0, t0
  j sumEvenDigitsLoop

isEvenDigit:
  add s1, s1, t2
  j sumEvenDigitsLoopCont

sumEvenDigitsLoopExit:
  add a0, s1, x0
  lw ra, 4(s0)
  lw s1, -4(s0)
  lw s0, 0(s0)
  addi sp, sp, 12
  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 a0, 4(gp)
lw a2, 8(gp)
xor a1, a0, a2
sw a1, 0(gp)
```

Solution: 12 cicli

2.3 Exerciţiul 0x02 (1 punct)

Fie codul de mai jos:

```
main:
  addi t0, t0, 0
et:
  addi t0, t0, 0
  beq t0, x0, et
```

Scrieţi, în **hexa**, codificarea instrucţiunii `beq t0, x0, et`, ştiind că `opcode` este `0x63`, `func3` este 0x0, iar instrucţiunea este reprezentată în clasa `B`. **Important!** Când se face un salt înapoi (sărim la `et`, care este instrucţiunea fix de deasupra), saltul se face relativ la *offset*-ul curent: aveţi, deci, că valoarea imediată este -4. (PC curent este `0x8`, iar `et` este la 0x4, deci faţă de PC curent, mergem cu 4 înapoi) Formatul clasei B este `imm[12|10:5] rs2 rs1 f3 imm[4:1|11] opcode`.

Solution: Se urmareste formatul clasei B, stim imediat ca rs2 = x0 = 0, rs1 = t0 = 5, deci rs2 = 00000 si rs1 = 00101. Adaugam f3 = 000, opcode = 1100011. Ramane sa reprezentam valoarea imediata, si anume -4. Avem ca 4 este 100, si avem nevoie de 4 pe 13 biti (de la imm[12] la imm[0]), adica avem 0000000000100. Vrem ca numarul sa fie negativ, il facem in complement fata de 2, adica 1111111111011, si adaugam 1, adica 1111111111100. Identificam ca imm[12—10:5] = 1111111 iar imm[4:1—11] = 11101. Avem astfel: 1111111 00000 00101 000 11101 1100011. Le grupam cate 4: 1111 1110 0000 0010 1000 1110 1110 0011 si scriem in hexa 0xFE028EE3.

3 Partea 0x02: Întrebări generale (3 puncte)

Alegeţi varianta corectă sau răspundeţi pe scurt.

0x00 Regiştrii `s0-s11` sunt o alegere bună pentru stocarea unor valori care ne aşteptăm să persiste chiar şi după apelul unor proceduri.

a. Adevărat b. Fals

0x01 (0.50p) De ce, în codificarea unei instrucţiuni din clasa **B**, nu scriem şi valoarea pentru `imm[0]`?

> **Solution:** Mereu este 0 valoarea respectiva, pentru ca aliniem mereu la multiplu de 2, obtinem codificari de forma ...0, sau ...1 + 1 = ...0 (complementul fata de 2).

0x02 (0.50p) Fie următoarea secţiune `.data`:

```
.data:
    .ascii "sir"
    .word 5
```

Scrieţi o instrucţiune prin intermediul căreia să încărcaţi valoarea 5 din `.data` în registrul `s1`.

> **Solution:** `lw s1, 3(gp)`

0x03 (0.75p) În analiza unui executabil, identificaţi următoarea valoare pe formatul `single`, `x = 0xC4801D00`. Ce număr îi corespunde în baza 10?

> **Solution:** 0xC4801D00 are codificarea binara (si grupata dupa bit de semn, exponent, mantisa) 1 10001001 00000000001110100000000. Observam ca numarul este negativ. Exponentul este 10001001 = 2**(7) + 2**(3) + 2**(0) = 128 + 8 + 1 = 137, deci exponentul real este 137-127=10. Numarul este, deci 1.00000000001110100000000 * 2**(10) = 10000000000.1110100000000. Avem ca 10000000000 = 2**(10) = 1024. Din mantisa, partea fractionara este 2**(-1) + 2**(-2) + 2**(-3) + 2**(-5) = 1/2 + 1/4 + 1/8 + 1/32 = (16 + 8 + 4 + 1) / 32 = 29/32 = 0.90625. Astfel, numarul este -1024.90625.

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

a. [-2047, 2048] b. [-4096, 4095] c. [-4094, 4095] d. $[-2^{11}, 2^{12}]$ e. $[-2^{11}, 2^{12} - 1]$

> **Solution:** b

0x05 (0.75p) Precizaţi de ce decodificarea instrucţiunilor în `RISC-V` poate fi paralelizată, spre deosebire de decodificarea în `x86`.

> **Solution:** In RISC-V, instructiunile au codificare pe lungime fixa, astfel ca stim mereu unde incepe si unde se termina o instructiune, incat putem paraleliza.

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 . . . .*