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

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

Test de laborator - Arhitectura Sistemelor de Calcul

16 ianuarie 2025

Seria 14, Varianta 1

- 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
    11bc: 05 20 2e 00 00        add    $0x2e20,%eax
    11c1: c7 45 f8 00 00 00 00  movl   $0x0,-0x8(%ebp)
    11c8: c7 45 fc 00 00 00 00  movl   $0x0,-0x4(%ebp)
    11cf: eb 2e                 jmp    11ff <f+0x52>
    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 10                 mov    (%eax),%edx
    11e2: 8b 45 f8              mov    -0x8(%ebp),%eax
    11e5: 8d 0c 85 00 00 00 00  lea    0x0(,%eax,4),%ecx
    11ec: 8b 45 08              mov    0x8(%ebp),%eax
    11ef: 01 c8                 add    %ecx,%eax
    11f1: 8b 00                 mov    (%eax),%eax
    11f3: 0f af c2              imul   %edx,%eax
    11f6: 2b 45 10              sub    0x10(%ebp),%eax
    11f9: 01 45 fc              add    %eax,-0x4(%ebp)
    11fc: d1 65 f8              shll   -0x8(%ebp)
    11ff: 8b 45 f8              mov    -0x8(%ebp),%eax
    1202: 8d 14 85 00 00 00 00  lea    0x0(,%eax,4),%edx
    1209: 8b 45 08              mov    0x8(%ebp),%eax
    120c: 01 d0                 add    %edx,%eax
    120e: 8b 00                 mov    (%eax),%eax
    1210: 85 c0                 test   %eax,%eax
    1212: 75 bd                 jne    11d1 <f+0x24>
    1214: 8b 45 fc              mov    -0x4(%ebp),%eax
    1218: c3                    ret
```

```
g00 <g>:
g01:        pushl   %ebp
g02:        movl    %esp, %ebp
g03:        subl    $20, %esp
g04:        movl    $0, -8(%ebp)
g05:        movl    $0, -4(%ebp)
g06:        jmp     .L9
    .L10:
g07:        movl    12(%ebp), %eax
g08:        subl    -4(%ebp), %eax
g09:        movl    -4(%ebp), %edx
g0A:        leal    0(,%edx,4), %ecx
g0B:        movl    8(%ebp), %edx
g0C:        addl    %ecx, %edx
g0D:        pushl   $2
g0E:        pushl   %eax
g0F:        pushl   %edx
g10:        call    f
g11:        addl    $12, %esp
g12:        movl    %eax, -20(%ebp)
g13:        fildl   -20(%ebp)
g14:        fdivs   16(%ebp)
g15:        fstps   -20(%ebp)
g16:        cvttss2sil      -20(%ebp), %eax
g17:        movl    %eax, -12(%ebp)
g18:        movl    -12(%ebp), %eax
g19:        addl    %eax, -8(%ebp)
g1A:        addl    $1, -4(%ebp)
    .L9:
g1B:        movl    -4(%ebp), %eax
g1C:        cmpl    12(%ebp), %eax
g1D:        jl      .L10
g1E:        fildl   -8(%ebp)
g1F:        ret
```

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

Solution: Procedura primeste trei argumente, observate ca offset pozitiv relativ la ebp, si anume 0x8(ebp), 0x0c(ebp) si 0x10(ebp). (11f6) In acelasi timp, putem observa ca sunt trei argumente din modul in care apelam f din g (linia g10 cu cele trei push-uri dinainte).

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

Solution: Pentru a determina valoarea de retur, urmarim ce se completeaza in registrul eax. Observam la 1214 ca avem un -0x4(ebp) care isi copiaza adresa in eax, iar -0x4(ebp) apare la 11f9 in contextul unei instructiuni pe tipul long. In concluzie, procedura returneaza un .long.

c. (0.75p) In timp ce analizati executabilul, va ganditi sa testati cu o valoare de tip *float*. Alegeti valoarea `-32.5`. Care este reprezentarea acestei valori pe formatul `single` (32b)? Scrieti valoarea in hexa.

Solution: Numarul este negativ, deci avem bitul de semn 1. Partea intreaga este 32, cea fractionara este 0.5. Reprezentarea partii intregi 32 = 0b100000, iar cea fractionara este 1 (0.5 * 2 = 1). Scriem numarul ca 100000.1. Scriem acum in forma stiintifica, si anume 1.000001 * 2**5. In acest caz, bitul de semn este 1, exponentul este 5 + 127 = 132 = 128 + 4 = 2**7 + 2**2 = 10000100, iar mantisa este 00000100000000000000000. Avem, deci, reprezentarea binara 1100 0010 0000 0010 0000 0000 0000 0000 = 0xC2020000.

d. (0.75p) Va atrage atentia codificarea hexa a programului si vreti sa vedeti care este semantica reprezentarilor. In acest caz, vreti sa vedeti cum se reprezinta in hexa `-0x8(%ebp)`. Analizati functia `f` si explicati cum deduceti aceasta reprezentare.

Solution: Analizam 11d1, 11e2, 11fc, 11ff si conchidem ca este octetul f8.

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: Contorul este in -0x8(ebp), facut 0 la 11c1. Structura incepe de la 11d1, identificata la 1212 ca fiind salt inapoi. Operatia asupra contorului este shll, deci inmultire cu 2, de la 11fc. Conditia de a ramane este ca elementul curent sa fie diferit de 0. (in calupul 11ff-120e se iau instructiunile, in 1210 se face verificarea cu zero).

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

Solution: Structura repetitiva: saltul inapoi, vizibil la g1D cu salt la .L10. Cautam contorul dupa initializare si incrementare, si il gasim in -4(ebp), initializat la g05, incrementat la g1A, iar conditia de a ramane este contor lt 12(ebp) = arg2, deci un i lt n. Ramane sa vedem ce calculam in structura repetitiva. Observam ca se apeleaza f cu trei argumente, f(edx, eax, 2), unde eax este calculat ca 12(ebp) din care se scade -4(ebp), deci un n - i, avem deci f(edx, eax, 2), iar edx este raportat la zona de memorie indicata in primul argument, deci un v, care este incrementat mereu cu indexul curent. avem un f(v + i, n - i, 2). Acest rezultat este stocat in -20(ebp), iar apoi se pune pe stiva FPU rezultatul si apoi se imparte la 16(ebp), deci la argumentul 3, ca mai apoi sa fie restocat in eax de unde este cumulat la o suma. Avem, deci o suma care devine vechea suma, la care se adauga f(v + i, n - i, 2) / arg3.

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

Solution:

```
g13. movss -20(%ebp), %xmm0
     movss 16(%ebp), %xmm1
g14: divss %xmm1, %xmm0
g15: movss %xmm0, -20(%ebp)
```

h. (1p) Observati ca la linia `g10` din procedura `g` se face un *call* imbricat in procedura `f`. Reprezentati configuratia stivei de apel, in momentul in care se obtine adancimea maxima.

Solution: Pentru adancimea maxima, avem cadrul lui g cu trei argumente, return address, ebp, spatiu pentru 5 (20 / 4) variabile locale, argumentele lui f, return address si ebp, spatiu pentru 4 (0x10/4) variabile locale.

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

a. (0.75p) Functioneaza o instructiune `ret` pusa arbitrar intr-o procedura simpla RISC-V (nerecursiva si care nu contine apeluri la alte proceduri)? Dar intr-o procedura simpla x86? Explicati.

Solution: Functioneaza pentru RISC-V pentru ca adresa de retur este mentinuta pe tot timpul procedurii in registrul ra. Pentru x86, saltul se face la valoarea din varful stivei (care poate varia in functie de procedura).

b. (0.75p) Sa presupunem ca lucrati la designul unui nou procesor RISC-V si doriti sa adaugati o extensie proprie. Aceasta extensie va contine printre alte 2 instructiuni noi `instr1` avand formatul `R` si urmatoarele specificatii (`opcode = 0b0000111, funct3 = 0b000, funct7 = 0b1111111`) și `instr2` avand formatul `U` (`opcode = 0b0000111`). Este aceasta o decizie corecta? Explicati.

Solution: Nu este o decizie corecta, cele 2 instructiuni pot face overlapping. (de exemplu, instructiunile `instr1 a0, a1, a2` si `instr2 a0, 0xFEC58000` au amandoua codificarea `0xFEC58507`)

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
auipc a1, 0x12345
beq a0, a1, label
slli a0, a1, 4
j final
label:
srli a0, a1, 4
final:
```

Solution: 1: a0 = 0x12345000 → 2: a1 = 0x12345004 → nu se face salt → 3: a0 = 0x23450040 → salt la final

d. (0.75p) Sa presupunem ca in `a2` avem stocata adresa functiei `func` la care vrem sa facem un salt folosind folosind instructiunea `jalr a7, a2, 0`. 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) Considerăm un sistem de calcul de 32 de biți. Sistemul poate realiza operațiile următoare: operații aritmetice/logice (1 ciclu), operații de citire/sciere date în memorie (2 cicli) și operații de branch/salt (3 cicli). Pentru ca operațiile artimetice/logice să fie executate programul realizează o pseudoinstructiune compusa din instructiunea aritmetica/logica propriu-zisă, două instrucțiuni de citire (citirea operanzilor) și apoi o operație de scriere (scrierea rezultatului). Avem un program care are în componență 10% pseudoinstructiuni aritmetice/logice, 60% alte operații de citire/scriere (40% operații citire și 20% operații scriere) și 30% operații de branch/salt.
Presupunem că adăugăm o nouă instrucțiune pentru inlocuirea pseudoinstructii aritmetice/logice care include cele două citiri și scrierea rezultatului. Noua instruncțiune are nevoie de 3 cicli. Cât de mult (procentual) este îmbunătățit sistemul de calcul?

Solution:

$$CPI_{initial} = 0.1 * 7 + 0.4 * 2 + 0.2 * 2 + 0.3 * 3 = 0.5 + 0.8 + 0.4 + 0.9 = 2.8$$

$$CPI_{optimizat} = 0.1 * 3 + 0.4 * 2 + 0.2 * 2 + 0.3 * 3 = 0.5 + 0.8 + 0.4 + 0.9 = 2.4$$

Sistemul este deci imbunatatit cu $\frac{2.8-2.4}{2.8} * 100 = \frac{0.4}{2.8} * 100 = 14.28\%$.

b. (0.5p) Un sistem are o memorie principală de 2^{20} bytes iar cache-ul are o capacitate totală de 16 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, presupunem că avem la linia 42, tag-ul 0b001101. Cărei adrese din memoria principală îi corespunde adresa de la offsetul 20 (word-ul cu offsetul 6) de pe această linie?

Solution:

$$\frac{2^{20}}{64} = \frac{2^{20}}{2^6} = 2^{14} \text{ blocuri in memoria principala}$$

$$\frac{2^4 * 2^{10}}{64} = \frac{2^{14}}{2^6} = 2^8 = 256 \text{ linii in cache}$$

Avem așadar:

- Offset - ultimii 6 biți (dimensiunea liniei e 2^6): 20 = 010100
- Index - următorii 8 biți (dimensiunea cache-ului e 2^8): 42 = 00101010
- Tag - 001101

Concatenam valorile de mai sus si obtinem adresa 0011 0100 1010 1001 0100 = 0x34A94.