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

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

Test de laborator - Arhitectura Sistemelor de Calcul

16 ianuarie 2025

Seria 14, Varianta 2

- 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 28                 jmp    11f9 <f+0x4c>
    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: 8b 55 10              mov    0x10(%ebp),%edx
    11e5: 89 d1                 mov    %edx,%ecx
    11e7: 2b 4d 14              sub    0x14(%ebp),%ecx
    11ea: 8b 55 f8              mov    -0x8(%ebp),%edx
    11ed: 01 d1                 add    %edx,%ecx
    11ef: 99                    cltd
    11f0: f7 f9                 idiv   %ecx
    11f2: 01 45 fc              add    %eax,-0x4(%ebp)
    11f5: c1 65 f8 02           shll   $0x2,-0x8(%ebp)
    11f9: 8b 45 f8              mov    -0x8(%ebp),%eax
    11fc: 8d 14 85 00 00 00 00  lea    0x0(,%eax,4),%edx
    1203: 8b 45 08              mov    0x8(%ebp),%eax
    1206: 01 d0                 add    %edx,%eax
    1208: 8b 00                 mov    (%eax),%eax
    120a: 85 c0                 test   %eax,%eax
    120c: 7f c3                 jg     11d1 <f+0x24>
    120e: 8b 45 fc              mov    -0x4(%ebp),%eax
    1212: c3                    ret
```

```
g00 <g>:
g01:        pushl   %ebp
g02:        movl    %esp, %ebp
g03:        subl    $20, %esp
g04:        movl    $1, -8(%ebp)
g05:        movl    $0, -4(%ebp)
g06:        jmp     .L6
    .L7:
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   $3
g0E:        pushl   $2
g0F:        pushl   %eax
g10:        pushl   %edx
g11:        call    f
g12:        addl    $16, %esp
g13:        movl    %eax, -20(%ebp)
g14:        fildl   -20(%ebp)
g15:        fmuls   16(%ebp)
g16:        fstps   -20(%ebp)
g17:        cvttss2sil      -20(%ebp), %eax
g18:        movl    %eax, -12(%ebp)
g19:        movl    -8(%ebp), %eax
g1A:        imull   -12(%ebp), %eax
g1B:        movl    %eax, -8(%ebp)
g1C:        addl    $1, -4(%ebp)
    .L6:
g1D:        movl    -4(%ebp), %eax
g1E:        cmpl    12(%ebp), %eax
g1F:        jl      .L7
g20:        fildl   -8(%ebp)
g21:        ret
```

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

> **Solution:** Procedura primeste patru argumente, identificam sau prin offset-uri (0x8, 0xc, 0x10, 0x14 - acesta la 11e7), sau ne uitam la apelul din g, sunt patru argumente incarcate pe stiva de la g0D la g10).

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 ca se completeaza -0x4(ebp), iar -0x4(ebp) este folosit doar in instructiuni long.

c. (0.75p) In timp ce analizati executabilul, va ganditi sa testati cu o valoare de tip *float*. Alegeti valoarea `-64.5`. 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 64, cea fractionara este 0.5. Reprezentarea partii intregi 64 = 0b1000000, iar cea fractionara este 1 (0.5 * 2 = 1). Scriem numarul ca 1000000.1. Scriem acum in forma stiintifica, si anume 1.0000001 * 2**6. In acest caz, bitul de semn este 1, exponentul este 6 + 127 = 133 = 128 + 4 + 1 = 2**7 + 2**2 + 2**0 = 10000101, iar mantisa este 00000010000000000000000. Avem, deci, reprezentarea binara 1100 0010 1000 0001 0000 0000 0000 0000 = 0xC2810000.

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 codul obiect `-0x4(%ebp)`. Observati ca aparitia este destul de rara, asa ca va bazati pe alte instructiuni pentru a deduce acest lucru. Analizati liniile `11d1`, `11db`, `120e`. Ce concluzie puteti trage?

Solution: Concluzionam de aici ca reprezentarea este `fc`, ultimul octet ne spune offset-ul relativ la ebp, adica exact ceea ce se cere.

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 120c ca fiind salt inapoi. Operatia asupra contorului este shll 0x2, deci inmultire cu 4, de la 11f5. Conditia de a ramane este ca elementul curent sa fie mai mare decat 0. (in calupul 11fc-1208 se iau instructiunile, in 120c 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 `fmuls 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 este vizibila prin saltul inapoi de la g1F. Identificam contorul ca fiind in -4(ebp) (se face add cu 1 pe el la g1C, respectiv este initializat cu 0 la g05, in afara structurii). Conditia de a ramane in structura este in calupul g1D-g1F: in cazul in care contorul curent este mai mic strict decat argumentul 2, ramanem in structura, altfel mergem la exit. Stim, momentan, ca avem un for i = 0; i lt arg2; i++
Vrem sa determinam ce se calculeaza in structura, astfel ca ne uitam intre g07 si g1C. Observam ca se face un call la f, cu f(edx, eax, 2, 3). Urmarim eax.
g07: primeste arg2
g08: se scade -4(ebp) din arg2, deci devine arg2 - i
nu se mai aplica nicio operatie asupra lui pana la apel, deci este un f(edx, arg2 - i, 2, 3). Urmarim edx
g09: se incarca in edx contorul
g0A: se incarca in ecx adresa de memorie 4*edx = 4i
g0B: se incarca primul argument in edx, deci avem arg1
g0C: se calculeaza arg1 + 4i ca adresa, ceea ce inseamna ca avem o deplasare fata de un tablou cu i long-uri la dreapta, deci un v + i
avem apelul f(v + i, n - i, 2, 3)
rezultatul acestui apel este pus pe stiva FPU
se inmulteste cu arg3

rezultatul este pus intr-o variabila locala, convertit apoi in eax, si cumulat cu eax la un produs, tinut in -8(ebp) (la g1B). observam si ca produsul este initial 1 (la g04)
conchidem ca se calculeaza un prod *= f(v + i, n - i, 2, 3) * arg3

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

Solution:

```
g14. movss -20(%ebp), %xmm0
     movss 16(%ebp), %xmm1
g15: mulss %xmm1, %xmm0
g16: movss %xmm0, -20(%ebp)
```

h. (1p) Observati ca la linia `g11` 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) Presupunem ca avem urmatorul apel C din main `f(g(x) + 1)`. In urma unor optimizari, compilatorul o sa foloseasca urmatoarele instructiuni de salt `call f`, respectiv `jmp g`. Se va produce vreodata revenirea in main in cazul unui procesor RISC-V? Dar al unuia x86 (considerand instructiunea de salt in g corespunzatorare `j g`)?

Solution: Pentru RISC-V se revine in main intrucat adresa de retur in main este retinuta in continuare in `ra`. Totusi trecerea inapoi prin f nu va mai avea loc (rezultatul va fi eronat, dar intoarcearea are loc). Pentru x86, acest lucru nu se va intampla, `ret`-ul din g de la final va produce cel mai probabil un segmentation fault intrucat in varful stivei nu o sa fie o adresa de retur valida.

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 `I` si urmatoarele specificatii (`opcode = 0b0000101 și funct3 = 0b000`) și `instr2` avand formatul `U` (`opcode = 0b0000101`). Este aceasta o decizie corecta? Explicati.

Solution: Nu este o decizie corecta, cele 2 instructiuni pot face overlapping. (de exemplu, instructiunile `instr1 a0, a1, 0` si `instr2 a0, 0x58000` au amandoua codificarea `0x00058505`)

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, 8
j final
label:
srli a0, a1, 8
final:
```

Solution: 1: a0 = 0x12345000 → 2: a1 = 0x12345004 → nu se face salt → 3: a0 = 0x34500400 → salt la final

d. (0.75p) Se da urmatorul schelet de functie. Care este efectul sau?

```
proc:
addi sp, sp, -16
addi s0, sp, 8
sw ra, 4(s0)
sw s0, 0(s0)

// Cod care nu mai modifica valoarea lui sp

lw s0, 0(s0)
lw ra, 4(s0)
addi sp, sp, 16
ret
```

Solution: Incarcarea in ra de la final nu se mai face de pe locatia corespunzatoare salvarii sale (s0 se modifica pe linia anterioara, preluand vechea sa valoare).

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 cicli), operații de citire/sciere date în memorie (2 cicli) și operații de branch/salt (4 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 20% pseudoinstructiuni aritmetice/logice, 60% alte operații de citire/sciere (30% operații citire și 30% operații scriere) și 20% operații de branch/salt. Presupunem că adăugăm o nouă instrucțiune pentru a inlocui pseudinstructiunea aritmetica/logica care include cele două citiri și scrierea rezultatului. Noua instruncțiune are nevoie de 4 cicli. Cât de mult (procentual) este îmbunătățit sistemul de calcul?

Solution:

$$CPI_{initial} = 0.2 * 7 + 0.3 * 2 + 0.3 * 2 + 0.2 * 4 = 1.4 + 0.6 + 0.6 + 0.8 = 3.4$$

$$CPI_{optimizat} = 0.2 * 4 + 0.3 * 2 + 0.3 * 2 + 0.2 * 4 = 0.8 + 0.6 + 0.6 + 0.8 = 2.8$$

Sistemul este deci imbunatatit cu $\frac{3.4-2.8}{3.4} * 100 = \frac{0.6}{3.4} * 100 = 17.64\%$.

b. (0.5p) Un sistem are o memorie principală de 2^{24} bytes iar cache-ul are o capacitate totală de 16 KB, cu o dimensiune a unui bloc de 32 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 20, tag-ul 0b0001001011. Cărei adrese din memoria principală îi corespunde adresa de la offsetul 28 (word-ul cu offsetul 7) de pe această linie?

Solution:

$$\frac{2^{24}}{32} = \frac{2^{24}}{2^5} = 2^{19} \text{ blocuri in memoria principala}$$

$$\frac{2^4 * 2^{10}}{32} = \frac{2^{14}}{2^5} = 2^9 = 512 \text{ linii in cache}$$

Avem așadar:

- Offset - ultimii 5 biți (dimensiunea liniei e 2^5): 28 = 11100
- Index - următorii 9 biți (dimensiunea cache-ului e 2^9): 20 = 000010100
- Tag - 0001001011

Concatenam valorile de mai sus si obtinem adresa 0001 0010 1100 0010 1001 1100 = 0x12B29B.