Test de laborator - Arhitectura Sistemelor de Calcul

Anul I

Numărul 2

- Nota maxima pe care o puteti obtine este 10.
- Nota obtinuta trebuie sa fie minim 5 pentru a promova, fara nicio rotunjire superioara.
- Aveti voie cu orice material, dar NU aveti voie sa discutati intre voi! Orice tentativa de frauda este considerata o incalcare a Regulamentului de Etica!

1 Partea 0x00 - maxim 4p

Consideram ca a fost implementata, in limbajul de asamblare studiat in cadrul laboratorului, o procedura `reciprocInverse` care primeste ca argumente, in ordine, adresa a doua tablouri bidimensionale de elemente de tip `.long`, dimensiunea comuna, tot ca argument de tip `.long`, si returneaza 1 daca cele doua matrice sunt mutual inverse, respectiv 0 altfel. Signatura este `reciprocInverse(&matrice1, &matrice2, n)`.

Observatie 1 Matricea matrice1 este inversa pentru matrice2 daca produsul lor este $\mathcal{I}_n$.

Subiectul 1 (3p) Sa se scrie o procedura `perecheInversabila` care primeste ca argumente, in ordine, trei tablouri bidimensionale de elemente de tip `.long`, dimensiunea comuna, tot ca argument de tip `.long`, si returneaza in `%eax` si `%ecx` adresele celor doua tablouri bidimensionale care sunt reciproc inverse, respectiv -1 in ambii registri daca nu exista o astfel de pereche. Pentru implementarea procedurii se vor respecta **toate** conventiile de apel din suportul de laborator. Procedura `perecheInversabila` va efectua apeluri interne catre procedura `reciprocInverse`.

Observatie 2 Toate tablourile bidimensionale sunt patratice si de aceeasi dimensiune.

Solution: Se accepta orice implemetarea valida care rezolva problema si respecta conventiile. Se vor acorda punctaje partiale.

Subiectul 2 (1p) Sa se reprezinte continutul stivei in momentul in care ajunge la adancimea maxima, conform scenarului de implementare de mai sus, considerand apelata din `main`, in mod corect, procedura `perecheInversabila`. Pentru reprezentarea stivei in aceasta configuratie, trebuie sa marcati si pointerii existenti in cadrul de apel (`%esp` si `%ebp`).

Solution: Se accepta orice desen al stivei in care sunt marcati cei doi pointeri si sunt reprezentate adresa de retur, vechea valoare a lui %ebp, registrii callee-saved si argumentele procedurii.

2 Partea 0x01 - maxim 3.5p

Subiectul 1 (0.5p) Se condisdera declarate `x: .word 1` si `y: .word 2`. Ce valoare va avea eax dupa executarea instructiunii `mov x, %eax`? Realizati o reprezentare pe octeti.

Solution: Execitiul 4 din TestLaborator2.1, 0x00020001.

Subiectul 2 (0.5p) Fie o procedura recursiva care primeste 4 argumente. In corpul acestei proceduri, pe langa conventiile standard, se salveaza registrul `%ebx` si se defineste un spatiu pentru 6 variabile locale de tip `.long`. Initial, registrul `%esp` se afla la adresa `0xffef1020`, iar spatiul disponibil de adrese este pana la `0xffcf0aa0`. Dupa cate autoapeluri se va obtine **segmentation fault**?

Solution: Calculam diferenta, spatiu = 0xffef1020 - 0xffcf0aa0 = 0x200580
= 2098560 bytes
= 524640 spatii pentru long
Stim ca stiva ocupa 4 argumente + r.a. + ebp + ebx + 6 variabile locale
= 13 long-uri la fiecare autoapel 524640 / 13 = 40356 rest 12
la al 40357-lea autoapel seg fault; exemplu in test ASC 2021

Subiectul 3 (0.5p) Care este semnificatia gruparii `a(b,c,d)`? Dati un exemplu de astfel de scriere pentru accesarea `v[i-3]` stiind ca `i` este depozitat in `%ecx`, adresa de inceput a lui `v` in `%edi` si ca v este un vector de long-uri.

Solution: Se obtine locatia b + c * d + a. v[i-3] poate fi scris ca -12(%edi, %ecx, 4).

Subiectul 4 (0.5p) Care este rolul simbolului `$` in limbajul de asamblare studiat?

Solution: Prefixare de constante numerice, operator de referentiere cand preceda simboluri din .data

Subiectul 5 (0.5p) Fie urmatoarea secventa de cod `mov $8, %eax`, `mov $4, %ebx`, `div %ebx`. Va fi in urma executiei acestei secvente in %edx mereu aceeasi valoare? Argumentati.

Solution: Da, chiar daca edx nu este initializat, orice valoare ar avea,
$2^32 * edx + 8$ va fi mereu divizibil cu 4, deci va avea restul, depozitat in edx = 0.

Subiectul 6 (0.5p) De ce `jmp (label + 4)` ar putea produce **segmentation fault**? Exista cazuri cand nu se intampla asta? Observatie: instructiunea este valida, nu se obtin erori de compilare.

Solution: Limbajul x86 are instructiuni de lungime variabila. Daca, de exemplu, instructiunea de dupa label este o instructiune de o lungime 4 sau exista doua instructiuni de lungime 2, nu se va produce segmentation fault. Altfel saltul in interiorul encodarii unei instructiuni ar putea produce un segmentation fault pentru ca citirea din dreptul PC-ului nu va mai fi recunoscuta ca o instructiune valida. (Se accepta raspunsuri mai scurte care indica aceasta idee)

Subiectul 7 (0.5p) Fie urmatoarele variabile declarate in memorie `x: .space 4`, `y: .long 6`. Se considera secventa de cod `movl $y, x`, `lea x, %eax`. Sa se scrie un scurt fragment de cod pentru a obtine in registrul `%eax` valoarea 6 fara a accesa memoria.

Solution: movl 0(%eax), %eax; movl 0(%eax), %eax

3 Partea 0x02 - maxim 2.5p

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 fragment de cod. Analizati acest cod si raspundeti la intrebarile de mai jos. Pentru fiecare raspuns in parte, veti preciza si liniile de cod / instructiunile care v-au ajutat in rezolvare.

```
000004ed <func>:
  1. 4ed:   push   %ebp
  2. 4ee:   mov    %esp,%ebp
  3. 4f0:   sub    $0x14,%esp
  4. 4f3:   call   56a
  5. 4f8:   add    $0x1ae4,%eax
  6. 4fd:   mov    0x14(%ebp),%eax
  7. 500:   mov    %al,-0x14(%ebp)
  8. 503:   movl   $0x0,-0x8(%ebp)
  9. 50a:   movl   $0x0,-0xc(%ebp)
 10. 511:   jmp    545 <func+0x58>
 11. 513:   mov    -0xc(%ebp),%edx
 12. 516:   mov    0x8(%ebp),%eax
 13. 519:   add    %edx,%eax
 14. 51b:   movzbl (%eax),%eax
 15. 51e:   cmp    %al,-0x14(%ebp)
 16. 521:   jle    541 <func+0x54>
 17. 523:   mov    -0xc(%ebp),%edx
 18. 526:   mov    0x8(%ebp),%eax
 19. 529:   lea    (%edx,%eax,1),%ecx
 20. 52c:   mov    -0x8(%ebp),%eax
 21. 52f:   lea    0x1(%eax),%edx
 22. 532:   mov    %edx,-0x8(%ebp)
 23. 535:   mov    %eax,%edx
 24. 537:   mov    -0x4(%ebp),%eax
 25. 53a:   add    %eax,%edx
 26. 53c:   movzbl (%ecx),%eax
 27. 53f:   mov    %al,(%edx)
 28. 541:   addl   $0x1,-0xc(%ebp)
 29. 545:   mov    0xc(%ebp),%eax
 30. 548:   imul   0x10(%ebp),%eax
 31. 54c:   cmp    %eax,-0xc(%ebp)
 32. 54f:   jl     513 <func+0x26>
 33. 551:   mov    -0x4(%ebp),%eax
 34. 554:   leave
 35. 555:   ret
```

a. (0.5p) Cate argumente primeste procedura de mai sus?

Solution: 4 argumente - avem 0x8(%ebp), 0xc, 0x10 si 0x14

b. (0.5p) Care este tipul de date al primului argument, stiind ca instructiunea `movzbl` efectueaza un `mov` cu o conversie de tip, de la `.byte` la `.long`?

Solution: urmarim ce se intampla cu 0x8(ebp)
la linia 12, se face un mov in eax, apoi la 13 se adauga edx peste eax
iar la linia 14, se face movzbl (eax), eax, ceea ce inseamna ca se preia locatia de memorie si se face conversie byte to long, deci in eax aveam un byte ptr, deci un char*, deci primul argument este un char*

c. (0.5p) Urmariti liniile 23 - 27 si retineti ca `%edx` retine o adresa de memorie. Pe baza acestor informatii, care este tipul de date returnat de procedura?

Solution: tipul returnat se afla la -0x4(ebp), conform liniei 33
observam ca -0x4(ebp) nu apare niciodata in dreapta intr-o instructiune
si atunci ne asteptam ca modificarea sa fie prin referinta
la linia 23, stim ca eax este o adresa de memorie, deci si edx este
la 24: se salveaza continutul de returnat in eax
la 25: se adauga o valoare peste adresa din edx
la 26: in eax avem un byte / char, pentru ca avem movzbl
la 27: se pune un byte la adresa din edx
dar edx era acel eax de la -0x4(ebp)
de aici conchidem ca -0x4(ebp) retine o adresa de memorie la care se salveaza char-uri, deci tipul returnat este char*

d. (1p) Liniile 11 - 32 descriu o structura repetitiva (indicata, in special, de liniile 31 si 32). Descrieti, cat mai detaliat, care este conditia care trebuie indeplinita pentru a se executa aceasta secventa.

Solution: incepem cu verificarea conditiei, la liniile 31 si 32
daca -0xc(ebp) lt eax, atunci se ramane in structura, altfel se iese
la linia 9 avem ca -0xc(ebp) este 0, si observam la 28 ca se tot incrementeaza
deci e de forma for (i = 0; i lt eax; i++), trebuie sa mai intelegem cine e eax
ne uitam la 29 si 30
se pune 0xc(ebp) in eax si apoi se inmulteste cu 0x10(ebp)
adica al doilea si al treilea argument
deci se executa (for i = 0; i lt arg2 * arg3; i++)