Test de laborator - Arhitectura Sistemelor de Calcul

Anul I

Numărul 1

- 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 `produsScalar` care primeste ca argumente, in ordine, adresa a doua tablouri unidimensionale de elemente de tip `.long`, dimensiunea comuna, tot ca argument de tip `.long`, si returneaza produsul scalar al acestora. Signatura este `produsScalar(&v, &w, n)`.

Subiectul 1 (3p) Sa se scrie o procedura `mutualOrtogonali` care primeste ca argumente, in ordine, trei tablouri unidimensionale de elemente de tip `.long`, dimensiunea comuna, tot ca argument de tip `.long`, si returneaza in `%eax` valoarea 1 daca cei trei vectori descrisi de cele trei tablouri unidimensionale sunt mutual ortogonali, respectiv 0 in sens contrar. Pentru implementarea procedurii se vor respecta **toate** conventiile de apel din suportul de laborator. Procedura `mutualOrtogonali` va efectua apeluri interne catre procedura `produsScalar`.

Observatie 1 Doi vectori sunt ortogonali (sau perpendiculari) daca produsul lor scalar este 0.

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 `mutualOrtogonali`. 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) Fie variabila `var` declarata in sectiunea `.data` astfel: `var: .long 18`. In cadrul programului, se efectueaza o incrementare asupra acestei zone de memorie, prin intermediul instructiunii `inc var`, care cauzeaza o eroare. De ce apare aceasta eroare?

Solution: Trebuie sufixata instructiunea cu dimensiunea tipului de date - decl;
var este doar un nume simbolic pentru o adresa din memorie, nu se poate face inferenta de tip

Subiectul 2 (0.5p) Care este diferenta dintre instructiunile `lea` si `mov`?

Solution: Instructiunea lea va depozita in destinatie adresa sursei, iar mov va depozita valoarea in sine a sursei.

Subiectul 3 (0.5p) Putem obtine adresa de memorie a unei etichete din program? Daca da, precizati un mod prin care putem face salt la respectiva adresa. Daca nu, de ce nu putem obtine adresele de memorie ale etichetelor din program?

Solution: Da - test lab ASC 2020, se poate;
Exista doua variante de raspuns:
1) lea et, reg sau mov $et, reg urmat de jmp *reg (steluta poate fi omisa);
2) push adresei pe stiva, si apoi un ret;

Subiectul 4 (0.5p) Sirul de instructiuni `xorl %eax, %eax; cmp $-1, %eax; jae L1` va conduce la efectuarea saltul la L1? De ce?

Solution: Nu, prezentarea diferentei dintre jae si jge

Subiectul 5 (0.5p) Care va fi rezultatul instructiunii `mul %ebx`, stiind ca in `%eax` avem stocata valoarea `0x40000000` si in `%ebx` avem stocata valoarea `8`? Descrieti ce se intampla la inmultire si explicati rezultatul.

Solution: Execitiul 2 din TestLaborator3.1, %eax = 0, %edx = 2

Subiectul 6 (0.5p) Dorim sa efectuam un apel de sistem, care primeste *flag*-uri de permisiune. Dupa ce citim documentatia, stabilim ca cele doua *flag*-uri pe care dorim sa le punem in acest argument sunt `S_IRUSR` si `S_IWUSR`. Valorile acestor constante sunt `S_IRUSR = 256`, iar `S_IWUSR = 128`. Ce valoare **hexa** veti pune in registrul care primeste valoarea *flag*-ului de permisiune?

Solution: 256 || 128 = 384 = 0x180, similar cerintei 3 de la tema

Subiectul 7 (0.5p) Fie o procedura recursiva care primeste 5 argumente. In corpul acestei proceduri, pe langa conventiile standard, se salveaza registrii `%ebx` si `%esi` si se defineste un spatiu pentru 8 variabile locale de tip `.long`. Initial, registrul `%esp` se afla la adresa `0xffff2024`, iar spatiul disponibil de adrese este pana la `0xffdf0ba0`. Dupa cate autoapeluri se va obtine **segmentation fault**?

Solution: Calculam diferenta, spatiu = 0xffff2024 - 0xffdf0ba0 = 0x201484
= 2102404 bytes
= 525601 spatii pentru long
Stim ca stiva ocupa 5 argumente + r.a. + ebp + ebx + esi + 8 variabile locale
= 17 long-uri la fiecare autoapel 525601 / 17 = 30917 rest 12
la al 30918-lea autoapel seg fault; exemplu in test ASC 2021

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   55d
  5. 4f8:   add    $0x1ae4,%eax
  6. 4fd:   mov    0x14(%ebp),%eax
  7. 500:   mov    %al,-0x14(%ebp)
  8. 503:   mov    0xc(%ebp),%eax
  9. 506:   imul   0x10(%ebp),%eax
 10. 50a:   mov    %eax,-0x4(%ebp)
 11. 50d:   movl   $0x0,-0xc(%ebp)
 12. 514:   movl   $0x0,-0x8(%ebp)
 13. 51b:   movl   $0x0,-0xc(%ebp)
 14. 522:   jmp    53c <func+0x4f>
 15. 524:   mov    -0xc(%ebp),%edx
 16. 527:   mov    0x8(%ebp),%eax
 17. 52a:   add    %edx,%eax
 18. 52c:   movzbl (%eax),%eax
 19. 52f:   cmp    %al,-0x14(%ebp)
 20. 532:   jne    538 <func+0x4b>
 21. 534:   addl   $0x1,-0x8(%ebp)
 22. 538:   addl   $0x1,-0xc(%ebp)
 23. 53c:   mov    -0xc(%ebp),%eax
 24. 53f:   cmp    -0x4(%ebp),%eax
 25. 542:   jl     524 <func+0x37>
 26. 544:   mov    -0x8(%ebp),%eax
 27. 547:   leave
 28. 548:   ret
```

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

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

b. (0.5p) Stiind ca `movzbl` efectueaza un `mov` cu o conversie de tip, de la `.byte` la `.long`, ce tip de date are primul argument al acestei proceduri?

Solution: linia 16, se pune in eax primul arg, se adauga edx la eax
apoi se face movzbl (eax), eax, adica e un byte ptr = char star

c. (0.5p) Ce tip de date are valoarea returnata de aceasta procedura?

Solution: ne uitam la ultima aparitie a lui eax - de la linia 26 obs ca e dependent de -8(ebp)
urmarim -8(ebp), la linia 21 i se adauga long un 1, la linia 12 a fost facut 0
astea sunt singurele 2 aparitii, tipul returnat este long - ceva care se tot incrementeaza, plecand de la 0

d. (1p) Liniile 15 - 25 descriu o structura repetitiva (indicata, in special, de liniile 24 si 25). Descrieti, cat mai detaliat, care este conditia care trebuie indeplinita pentru a se executa aceasta secventa.

Solution: Se executa daca eax lt -0x4(ebp), altfel merge la exit
daca se respecta conditia, se sare la 524 = linia 15
acolo se muta -0xc(ebp) in edx
dar -0xc(ebp) este initial 0 (linia 11)
urmarim -0xc(ebp) si vedem incrementarea la 22
apoi se muta in eax, si de acolo verificarea
deci se executa o structura cu indici de la 0 la -0x4(ebp)
iar -0x4(ebp) este construit la linia 10
dupa o inmultire intre 0xc(ebp) si 0x10(ebp), adica intre argumentele 2 si 3
deci for (int i = 0; i ¡ arg2 * arg3; i++) este solutia