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 `distantaCuvinte` care primeste ca argumente, in ordine, adresa a doua siruri de caractere si returneaza o distanta, numita distanta Levenshtein. Aceasta distanta este o valoare de tipul `.long`, mai mare sau egala cu 0. Signatura procedurii este `distantaCuvinte(&cuv1, &cuv2)`.

Subiectul 1 (3p) Sa se scrie o procedura `celeMaiApropiate` care primeste ca argumente, in ordine, trei siruri de caractere si returneaza in `%eax` si `%ecx` adresa sirurilor intre care distanta Levenshtein este cea mai mica. Daca intre siruri exista aceeasi distanta Levenshtein, se vor intoarce oricare doua dintre ele. Pentru implementarea procedurii se vor respecta **toate** conventiile de apel din suportul de laborator. Procedura `celeMaiApropiate` va efectua apeluri interne catre procedura `distantaCuvinte`.

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 `celeMaiApropiate`. Pentru reprezentarea stivei, 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) Care este rolul simbolului `$` in limbajul de asamblare studiat?

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

Subiectul 2 (0.5p) Care este semnificatia gruparii `a(b,c,d)`? Dati un exemplu de astfel de scriere pentru accesarea `v[i-2]` 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-2] poate fi scris ca -8(%edi, %ecx, 4).

Subiectul 3 (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 4 (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 5 (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

Subiectul 6 (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 7 (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)

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:   push   %ebx
  4. 4f1:   sub    $0x10,%esp
  5. 4f4:   call   59a
  6. 4f9:   add    $0x1ae3,%eax
  7. 4fe:   movl   $0x0,-0xc(%ebp)
  8. 505:   movl   $0x0,-0x8(%ebp)
  9. 50c:   jmp    571 <func+0x84>
 10. 50e:   mov    -0x8(%ebp),%eax
 11. 511:   lea    0x0(,%eax,4),%edx
 12. 518:   mov    0x8(%ebp),%eax
 13. 51b:   add    %edx,%eax
 14. 51d:   mov    (%eax),%edx
 15. 51f:   mov    -0x8(%ebp),%ecx
 16. 522:   mov    0x10(%ebp),%eax
 17. 525:   add    %ecx,%eax
 18. 527:   movzbl (%eax),%eax
 19. 52a:   movzbl %al,%eax
 20. 52d:   lea    (%edx,%eax,1),%ecx
 21. 530:   mov    -0x8(%ebp),%eax
 22. 533:   lea    0x0(,%eax,4),%edx
 23. 53a:   mov    0x8(%ebp),%eax
 24. 53d:   add    %edx,%eax
 25. 53f:   mov    (%eax),%edx
 26. 541:   mov    -0x8(%ebp),%eax
 27. 544:   lea    0x0(,%eax,4),%ebx
 28. 54b:   mov    0x8(%ebp),%eax
 29. 54e:   add    %ebx,%eax
 30. 550:   mov    (%eax),%eax
 31. 552:   imul   %edx,%eax
 32. 555:   cmp    %eax,%ecx
 33. 557:   jle    56d <func+0x80>
 34. 559:   mov    -0x8(%ebp),%eax
 35. 55c:   lea    0x0(,%eax,4),%edx
 36. 563:   mov    0x8(%ebp),%eax
 37. 566:   add    %edx,%eax
 38. 568:   mov    (%eax),%eax
 39. 56a:   add    %eax,-0xc(%ebp)
 40. 56d:   addl   $0x1,-0x8(%ebp)
 41. 571:   mov    -0x8(%ebp),%eax
 42. 574:   imul   -0x8(%ebp),%eax
 43. 578:   cmp    %eax,0xc(%ebp)
 44. 57b:   jg     50e <func+0x21>
 45. 57d:   mov    -0xc(%ebp),%eax
 46. 580:   add    $0x10,%esp
 47. 583:   pop    %ebx
 48. 584:   pop    %ebp
 49. 585:   ret
```

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

Solution: 3 argumente - avem 0x8(%ebp), 0xc si 0x10

b. (0.5p) Care este tipul de date al valorii de retur?

Solution: se urmareste ultima aparitie a lui eax inainte de retur: linia 45, se pune -0xc(ebp), iar apoi urmarim ce tip are -0xc(ebp): vedem de la linia 39 ca se pune eax in -0xc(ebp), iar eax este un long: se face la 38 un mov (eax), eax, ceea ce inseamna mov de tip long - se ia continutul de memorie de la adresa retinuta de eax, si se pune in eax. Raspuns: intoarce un long.

c. (0.5p) Ce tip de date are al treilea argument, stiind ca instructiunea `movzbl` efectueaza un `mov` cu o conversie de tip, de la `.byte` la `.long`?

Solution: al treiela argument este un 0x10(ebp), il gasim la linia 16
se adauga o valoare la eax, iar apoi se face movzbl de la (eax) la eax, deci se schimba in tip long de la o adresa de memorie care era un byte, deci al treilea argument este un byte ptr = char*

d. (1p) Liniile 10 - 44 descriu o structura repetitiva (indicata, in special, de liniile 43 si 44). Descrieti, cat mai detaliat, care este conditia care trebuie indeplinita pentru a se executa aceasta secventa.

Solution: liniile 43 + 44 ne spun ca daca eax lte 0xc(ebp) ramane in structura, altfel exit
la linia 10, initial, eax este -0x8(ebp), care si el initial este 0
urmarim ce se intampla cu -0x8(ebp) si eax
observam la linia 40 ca este incrementat
iar apoi observam la 41 si 42 ca este ridicat indicele la patrat
deci are o structura (for i = 0; i * i lte 0xc(ebp); i++)
iar 0xc(ebp) este al doilea argument
deci se executa pana la parte intreaga din radical din al doilea argument