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 `factorProportionalitate` 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 factorul de proportionalitate dintre cei doi vectori daca el exista (si este intreg) si 0 altfel. Signatura este `factorProportionalitate(&v, &w, n)`.

Subiectul 1 (3p) Sa se scrie o procedura `mutualProportionali` 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 proportionali, respectiv 0 in sens contrar. Pentru implementarea procedurii se vor respecta **toate** conventiile de apel din suportul de laborator. Procedura `mutualProportionali` va efectua apeluri interne catre procedura `factorProportionalitate`.

Observatie 1 Doi vectori sunt proportionali daca factorul lor de proportionalitate este diferit de 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 `mutualProportionali`. 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) 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 2 (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 3 (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 4 (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 5 (0.5p) Fie variabila `var` declarata in sectiunea `.data` astfel: `var: .long 20`. In cadrul programului, se efectueaza o decrementare asupra acestei zone de memorie, prin intermediul instructiunii `dec 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 6 (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 (similar test ASC 2021)

Subiectul 7 (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 cu subiectul 3 din tema

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 <proc>:
  1. 4ed:   push   %ebp
  2. 4ee:   mov    %esp,%ebp
  3. 4f0:   sub    $0x14,%esp
  4. 4f3:   call   573
  5. 4f8:   add    $0x1ae4,%eax
  6. 4fd:   mov    0x10(%ebp),%eax
  7. 500    :mov    %al,-0x14(%ebp)
  8. 503:   movl   $0x0,-0x8(%ebp)
  9. 50a:   movl   $0x0,-0xc(%ebp)
 10. 511:   jmp    552 <proc+0x65>
 11. 513:   mov    -0xc(%ebp),%eax
 12. 516:   lea    0x0(,%eax,4),%edx
 13. 51d:   mov    0x8(%ebp),%eax
 14. 520:   add    %edx,%eax
 15. 522:   mov    (%eax),%eax
 16. 524:   cmp    %eax,0x18(%ebp)
 17. 527:   jge    54e <proc+0x61>
 18. 529:   mov    -0xc(%ebp),%edx
 19. 52c:   mov    0x14(%ebp),%eax
 20. 52f:   add    %edx,%eax
 21. 531:   movzbl (%eax),%eax
 22. 534:   cmp    %al,-0x14(%ebp)
 23. 537:   jne    54e <proc+0x61>
 24. 539:   mov    -0xc(%ebp),%edx
 25. 53c:   mov    0x14(%ebp),%eax
 26. 53f:   add    %edx,%eax
 27. 541:   mov    -0xc(%ebp),%ecx
 28. 544:   mov    -0x4(%ebp),%edx
 29. 547:   add    %ecx,%edx
 30. 549:   movzbl (%eax),%eax
 31. 54c:   mov    %al,(%edx)
 32. 54e:   addl   $0x1,-0xc(%ebp)
 33. 552:   mov    -0xc(%ebp),%eax
 34. 555:   cmp    0xc(%ebp),%eax
 35. 558:   jl     513 <proc+0x26>
 36. 55a:   mov    -0x4(%ebp),%eax
 37. 55d:   leave
 38. 55e:   ret
```

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

Solution: 5 argumente, ne uitam la adresarile pozitive relativ la %ebx incepand cu 0x8, pana la 0x18

b. (0.5p) Analizand instructiunile in care apare primul argument al acestei proceduri, care este tipul de date pe care il are?

Solution: linia 13, se pune arg1 in eax, se adauga edx la eax, se face mov (eax), eax, e un pointer la long

c. (0.5p) Ce tip de date are valoarea returnata de aceasta procedura, stiind ca `movzbl` efectueaza un `mov` cu o conversie de tip, de la `.byte` la `.long`?

Solution: urmarim ultima aparitie a lui eax
observam linia 36, se pune -0x4(ebp) in eax
urmarim -0x4(ebp)
la linia 28 se foloseste mov -0x4(ebp) in edx
se face un add peste edx, iar apoi un movzbl (eax), eax
deci o conversie byte to long, deci in eax avem reprezentat un byte
iar eax era un byte ptr
se pune partea cea mai nesemnificativa - al, in acel edx
care a fost stocat in -0x4(ebp)
observam la 34 - 35 ca totul e un loop
deci conchidem ca -0x4(ebp) stocheaza de fapt un byte ptr - char *
deci se intoarce un char*

d. (1p) Intre liniile 11 - 35 este descrisa o structura repetitiva (indicata, in special, de liniile 34 si 35). Descrieti, cat mai detaliat, care este conditia care trebuie indeplinita pentru a se executa aceasta secventa.

Solution: E analiza directa pe cod. Urmarim mai intai conditia, data de liniile 34 si 35:
- daca eax lt 0xc(ebp), ramanem in structura
- altfel mergem spre retur
a ramane in structura = salt la adresa 513 = linia 11
la linia 11, se pune -0xc(ebp) in eax
dar -0xc(ebp) este initial 0, deci poate fi un index
urmarim ce se intampla
-0xc(ebp) se pune in eax
eax este folosit cu alte scopuri in structura
dar la linia 32 observam ca se face addl cu 1 peste -0xc(ebp)
deci o incrementare
iar apoi valoarea este iar copiata in eax, pentru conditie
deci conditia de indeplinit este ca un index curent, initial 0 si care se tot incrementeaza, sa fie mai mic strict decat 0xc(ebp) = al doilea argument
deci un
for (int i = 0; i lt arg2; i++) ...