01101001
01101111
01100011
01101100
01100001

Nume și grupă:

Introducerea în Organizarea Calculatoarelor și Limbaje de Asamblare

3 iunie 2024

Timp de lucru: 120 de minute

1. Punându-și la punct lucrul cu manipularea biților, într-un final glorios, studenții de an superior vă provoacă la o competiție demnă de intrat în legendă.

a. Fără a vă folosi de împărțire, verificați dacă un întreg este par sau impar. Pentru testare, folosiți variabila `n`, deja definită în program. În cazul în care aceasta este pară, afișați `even`, altfel `odd`. (**4 puncte**)

Atenție! Este interzisă folosirea instrucțiunii `div`.

b. Salvați elementele aflate pe poziții pare și care au ultima cifră 8 din vectorul `arr`, deja definit în program, într-un alt vector, `res`, definit de voi în orice secțiune doriți. Afișați vectorul rezultat. (**5 puncte**)

c. Afișați perechile de numere aflate pe poziții **consecutive** din `arr` care au ultimul bit egal. (**6 puncte**)

Spre exemplu, pentru șirul de numere `10, 4, 2, 3, 5, 1024`, se va afișa

```
10 4
4 2
3 5
```

Explicația este următoarea:

`10 = 1010; 4 = 100`. Cele două numere au ultimul bit identic, și anume, 0.

`4 = 100, 2 = 10`. Cele două numere au ultimul bit identic, și anume, 0.

`3 = 11, 5 = 101`. Cele două numere au ultimul bit identic, și anume, 1.

2. Știindu-vă maeștri în ale algoritmicii **și** ale assembly-ului, sunteți provocați la o serie de încercări pentru a vă dovedi valoarea. Fiți curajoși!

a. Creați funcția `pow4`, cu semnătura `void pow4(int *n)` care salvează în n rezultatul operației 4^n. Cu alte cuvinte, n va conține 4 la puterea valorii sale inițiale. Este garantat că rezultatul ridicării la putere este o valoare pe 32 de biți. (**3 puncte**)

b. Creați funcția `read_array`, cu semnătura `void read_array(int *n, int *v)`, care citește de la tastatură un număr întreg n și apoi, n elemente ale vectorului `v`, folosind funcția `scanf`. Apelați funcția `read_array(int *n, int *v)` cu n, o variabilă salvată pe stivă, iar v un vector declarat de voi în ce secțiune doriți. Puteți considera că numărul maxim de elemente citite este 100. (**4 puncte**)

c. Creați funcția `print_array` cu semnătura `void print_array(int n, int *v)`, care printează cele n elemente întregi ale vectorului v. Apelați funcția `void print_array(int n, int *v)` folosindu-vă de vectorul citit la punctul anterior. (**3 puncte**)

d. Creați funcția `map_array`, cu semnătura `void map_array(int n, int *v, void (*f)(int *))`, care primește un vector v cu n elemente și apelează funcția f pentru fiecare element. Testați implementarea folosindu-vă de vectorul citit la punctul **b.** și funcția `pow4` implementată la punctul **a.**, împreună cu funcția de printare de la punctul **c**. Astfel, dacă vectorul inițial conține valorile 1, 2, 3, 4, în urma aplicării funcției `pow4` pe elementele sale, v va conține valorile 4, 16, 64, 256. (**5 puncte**)

Atenție! Este interzisă apelarea directă a funcției `pow4` pe elementele vectorului. Folosiți-vă de funcția trimisă ca parametru, pentru a păstra genericitatea!

3. Marcel are nevoie de ajutorul pentru a deslușii misterele a trei funcții scrise în asamblare a căror implementare se găsește în fișierul `file.asm` cu prototipurile descrise în fișierul `file.h`. Pentru ca ajutorul să fie complet urmăriți conținutul fișierului `todo.c` și implementați următoarele cerințe:

a. Urmăriți conținutul funcției `todo_a`. Apelați funcția `compute` astfel încât aceasta să întoarcă rezultatul 0. Implementarea funcției `compute` se găsește în fișierul `file.asm`. Pentru verificare rulați `./main a` (**4 puncte**)

b. Urmăriți conținutul funcției `todo_b`. Inițializați corespunzător vectorul `vec` astfel încât apelul funcției `check_array` să întoarcă rezultatul 0. Pentru verificare rulați `./main b` (**5 puncte**)

c. Urmăriți conținutul funcției `todo_c`. Apelați funcția `read_string` și identificați o vulnerabilitate astfel încât să se apeleze funcția `secret_function`. Pentru verificare rulați `./main c` (**6 puncte**)