Laborator 3

Info:

- Teorie

Exercitii rezolvate:

1. Scrieti o functie care face calculul: 2 + 2*4 + 2 * 4 * 6 + ... + 2*4*...*2n, cu n parametrul functiei. Apelati functia din ASM.

```
// Lab3.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include<iostream>
using namespace std;
// 2 + 2*4 + ... + 2*4*...*2n
int f(int n) {
 /*
 push ebp
 mov ebp, esp
 (astea le face deja compilatorul inainte sa intram in blocul asm)
 */
 _asm {
  push 0 //Declaram o variabila locala cu valoarea 0.
  mov ecx, [ebp + 8] //ebp+8 e adresa primului parametru.
   //Copiem valoarea de acolo in ecx, pentru loop.
  mov eax, 1
  mov ebx, 2
 start_loop:
  xor edx, edx
  mul ebx
  add [esp], eax //In varful stivei, adica in variabila locala, adunam eax.
  add ebx, 2
  loop start_loop
  mov eax, [esp]
  add esp, 4 //Dealocam variabila locala. Puteam inlocui astea 2 linii cu pop eax
 }
 /*
 mov esp, ebp
 pop ebp
 ret
 (astea le face deja compilatorul inainte sa intram in blocul asm)
 */
}

int main()
{
 int n;
 cin >> n;
 int value;
 /*
 standard cdecl: eu sunt call => Eu curat stiva (aka sub esp, 4).
  => Parametrii ii dau un sens invers (de la dreapta la stanga)
  => Am eu grija de aici sa nu fie stricati registrii.
 */
 _asm {
  push n
  call f
  add esp, 4
```

```
  mov value, eax
 }
 cout << value << '\n';
    return 0;
}
```

2. Implementati o functie care calculeaza al n-lea termen fibbonacci, recursiv.

```
#include "stdafx.h"
#include<iostream>
using namespace std;

int fibo(int n) {
 _asm {
  cmp [ebp + 8], 1 //Parametrul.
  jle simple_case
  //Nu suntem pe cazul simplu => trb apelat fact(n-1) si fact(n-2) si adunat.
  mov ebx, [ebp + 8]
  dec ebx
  push 0 //Vom face suma in variabila asta.
  push ebx // Parametrul functiei apelate
  call fibo
  add esp, 4 // Fac curat pe stiva. Adica scot ebx-ul vechi.
  add [esp], eax //Adun la suma valoarea de return a lui fact.
  mov ebx, [ebp+8] //ebx e foarte probabil sa nu mai fie n-1. Trebbuie recalculat.
  sub ebx, 2
  push ebx
  call fibo //Apel fact(n-2)
  add esp, 4 //Curatenie pe stiva
  add [esp], eax //In varful stivei (variabilea mea locala) adun eax (ret val)
  pop eax //Variabila locala o pun in eax ca sa pot da return.
  jmp done
 simple_case:
  mov eax, 1 //Returnez 1.
 done:

 }
}


nt main()
{
 int n;
 cin >> n;
 int value;
 /*
 standard cdecl: eu sunt call => Eu curat stiva (aka sub esp, 4).
  => Parametrii ii dau un sens invers (de la dreapta la stanga)
  => Am eu grija de aici sa nu fie stricati registrii.
 */
 _asm {
  push n
  call fibo
  add esp, 4
  mov value, eax
 }
 cout << value << '\n';
    return 0;
}
```

Exercitii nerezolvate:

1. Implementati o functie care primeste 4 parametrii de tip int si il returneaza pe cel mai mare dintre ei. Apelati-o din asm.
2. Traduceti functia de mai jos in ASM (Fara a face optimizari, cum ar fi omiterea lui while). Folositi doua variabile locale pentru c si d:

```
int adunare(int a,int b)
```

```
{
  int c=3;
  int d=6;

  while(c>10) c++;
  if(c>15) return a+b+c+d;
  else return a+b+c-d;
}
```

3. Implementati functia factorial, recursiv. Apelati functia tot dintr-un bloc asm.
4. Rezolvati problemele de la laboratorul precedent, fara a declara alte variabile din afara blocului asm.
5. Implementati calculul cmmdc a doua numere

Comments

You do not have permission to add comments.