**Curs 10**

**Recapitulare**

1. <https://nasm.us/doc/>
2. Macrouri v.s. Proceduri

|  |  |
| --- | --- |
| **Macrouri** | **Proceduri** |
| - la fiecare apel se copiază secvența de instrucțiuni  - nu sunt necesare instrucţiuni de apel (CALL) şi de revenire din rutină (RET)  - nu se foloseşte stiva  - transferul de parametri se face prin copierea numelui | - o singură copie pentru mai multe apeluri  - se folosesc instrucţiuni de apel (CALL) şi de revenire (RET)  - se utilizează stiva la apel şi la revenire  - transferul de parametri se face prin registre sau stivă |

1. Problema etichetelor in macrouri

La copierea repetată, etichetele se duplică. Soluţia: folosirea etichetelor „locale”

Directiva **LOCAL**

<nume\_macro> **macro** [<par1> [<par2> ...]]

local <et1> [<et2> ..]]

..................

<et1>:

....................

<et2>:

**endm**

1. Avantajele şi dezavantajele utilizării macrourilor

|  |  |
| --- | --- |
| **Avantaje** | **Dezavantaje** |
| 1. pot fi create „instrucțiuni” noi 2. poate duce la o programare mai eficientă 3. execuţie mai eficientă în comparaţie cu apelurile de proceduri | 1. pot ascunde operații care afectează conținutul registrelor 2. utilizarea extensivă a macrourilor îngreunează înțelegerea şi mentenanţa programului |

5) Problema optimizării

Când şi ce se optimizează

* regula 90/10 - 90% din timp se execută 10% din cod
  + consecinţă – dacă se elimina 90% din codul rar folosit îmbunătăţirea este de 10%
  + ce se doreşte?
* timp redus sau resurse ocupate (în special memorie) redusă
  + Cum se măsoară timpul ocupat al procesorului pentru fiecare modul – cu programe de tip „profiler”
  + când este bine să se optimizeze:
    - de la început:
      * se optimizează şi partea nesemnificativă
      * programul se scrie greu şi se înţelege şi mai greu
    - la sfârşit:
      * prea târziu

Optimizarea este necesară?

Contra-argumente:

* viteza mare a procesoarelor, a memoriilor şi a magistralelor
* spaţii de memorie foarte mari

Când este necesară optimizarea:

* prelucrări de informaţii multimedia
* prelucrări de semnale
* sisteme de control în timp real, sisteme reactive

Trei tipuri de optimizare

* optimizare de nivel înalt

Implică găsirea unui alt algoritm, mai bun, de pildă cu grad mai mic de complexitate (de exemplu O(*n*2) =>O(*n*log(*n*)) )

* optimizare de nivel mediu

Algoritm implementat mai bine

* optimizare de nivel scăzut

Minimizarea numărului de cicluri

Tehnici de optimizare

* reducerea numărului de bucle imbricate
* reducerea timpului de execuţie al buclei interioare
* utilizarea pointerilor în adresarea elementelor unor structuri de date
* parcurgerea structurilor de date prin incrementarea şi decrementarea poantorilor în locul calculării adresei elementului

–ex: tab[*i*][*j*]

adr\_element*ij* = adresa(tab) + *i* \* lung\_rand + *j*

* utilizarea registrelor interne ale procesorului în operaţiile curente

Exemplu de optimizare

Problema: filtrarea unei imagini de 100 de ori, rezoluție 256 \* 256 pixeli

Variante:

* program Pascal - 45s
* program C - 29s
* asamblare (V1) - 6s
  + s-au folosit deplasamente precalculate î locul calculului de adresă prin indecşi
* asamblare (V2) - 4s
* s-au folosit registre interne în locul variabilelor de memorie
* asamblare (V3) - 2,5s
* s-a evitat recopierea imaginii intermediare în matricea iniţială
* asamblare (V4) 2,4s
* s-a redus dimensiunea variabilelor de la întreg la caracter pentru a beneficia de memoria cache
* asamblare (V5) 2,2s
* s-a schimbat algoritmul de filtrare, s-au luat în calcul numai 4 vecini ai pixelului în loc de 8 (rezultatul este diferit)

6) Apelul procedurilor scrise în asamblare din limbaje de nivel înalt

Când se justifică:

* la accesul direct al unor resurse fizice (de exemplu: interfeţe de I/E, memoria video etc.)
* pentru creşterea eficienţei unor secvenţe critice
* atunci când funcţiile de acces la resurse au fost scrise (deja) în asamblare

Dificultăţi:

* altă filozofie de scriere a programelor (ex: registre în loc de variabile, date simple în locul celor structurate)
* transferul parametrilor de apel şi a rezultatelor

Reguli:

* procedura scrisă în limbaj de asamblare se va declara “far” şi “public”
* numele şi tipul procedurii se va declara în limbajul de nivel înalt cu menţiunea EXTERN
* parametrii de apel se transmit prin stivă:
  + - ordinea de scriere pe stivă coincide cu ordinea din lista de parametri
    - primul parametru este cel mai adânc în stivă (se afla la o adresă mai mare)
    - ultima dată înregistrată pe stivă este un dublucuvânt care reprezintă adresa de revenire din rutină