Referirea datelor în structuri complexe prin proceduri assembler

Prof.dr. Ion IVAN, asist. Marian DÂRDALA,
ec. Gabriel SUTAC, ec. Valentin DRAGOMIR
Catedra de Informatica Economica, A.S.E. Bucuresti

Lucrarea are ca obiectiv evidentierea modului de accesare a bazelor de date de tip FoxPro folosind proceduri scrise în limbaj de asamblare. Aceasta abordare este necesara pentru a putea prelucra datele stocate independent de sistemul de gestiune a bazelor de date de care apartin.
Cuvinte cheie: *header, database file, index, structuri de date, assembler.*

1. Organizarea structurilor complexe

Limbajele de programare descriu tipuri fundamentale de date si tipuri de date derivate.
Limbajul de asamblare contine descrieri corespunzatoare tipurilor fundamentale manipulate de catre programe assembler precum: *db* (define byte), *dw* (define word), *dd* (define double), etc.
În secventa S1 se definesc în limbajul C++ masivele:

```
int x[N];
int y[N][M];                    (S1)
int z[N][M][P];
```

Secventa S2 corespunde definirii în limbaj de asamblare a secventei S1:

```
x dw N DUP (?)
y dw N DUP (M DUP (?))          (S2)
z dw N DUP (M DUP ( P DUP (?)))
```

Secventele S3 si S4 definesc structuri de tip articol în limbajele C++, respectiv în limbaj de asamblare:

```
struct adresa {
        char oras [20];
        char strada[30];
        int numar;}
struct muncitor {
        int marca;
      char num[30];
      adresa adresa_m;
        int vechime;
        float salariu;}
```

Secventa S3

```
adresa      struct
   oras     db      20 DUP (?)
   strada   db      30 DUP (?)
   numar    dw      ?
ends
muncitor    struct
marca       dw      ?
mun         db      30 DUP (?)
adresa_m    adresa  <?>
vechime             dw      ?
salariu             dd      ?
ends
```

Secventa S4

Concatenarea de elemente de acelasi tip sau a membrilor de tipuri diferite reprezinta o modalitate de agregare a datelor.
Secventei de definire si referire de pointeri în limbajul C++:

```
int a, pa=&a,*ppa=&pa;
int b, *pb, **ppb; pb=&b; ppb=&pb;
```

îi corespunde în limbajul de asamblare secventa:

```
a   dw    ?
pa  dw    a
ppa dw    pa
b   dw    ?
pb  dw    ?
ppb dw    ?
mov ax, offset b
mov pb, ax
mov ax, offset pb
mov ppb, ax
```

Agregarea structurilor de date, ca procedeu cadru de obtinere a structurilor de date complexe conduce la obtinerea unor structuri precum vectorii de structura, structura de vectori, pointerul spre structuri, structurile de pointeri. Lor le corespund, în limbajele de programare descrieri si referiri corespun-

zatoare. Astfel, descrierea în limbajul C++ a secventei S5

```
struct vect {
        int     a[10];
        char    b[5];
} c[20];
```

Secventa S5

defineste un vector de structura *vect* având 20 de componente.

Referintele *c[i].a[j]* si *c[j].b[k]* permit extragerea continutului zonelor de memorie a unor parti ale variabilei agregate *c*.

Agregarile cu autoreferire permit definirea de structuri de date complexe precum liste simple, liste duble, arbori binari, arbori B etc. Definirea secventei S5 în limbaj de asamblare este prezentata în secventa S6.

```
vect struct
        a dw 10  DUP(?)
        b db 5 DUP(?)
ends
c vect 20 DUP<?>
```

Secventa S6

Definirile tuturor sabloanelor asociate fisierelor reprezentate în diverse formate constituie modalitati ale agregarii tipurilor de date. Traversarea structurilor de date poate fi omogena sau eterogena. Daca legaturile se realizeaza între elementele aceleiasi structuri, traversarea este omogena (figura 1), în caz contrar (când elementele apartin unor structuri diferite), traversarea este eterogena (figura 2).

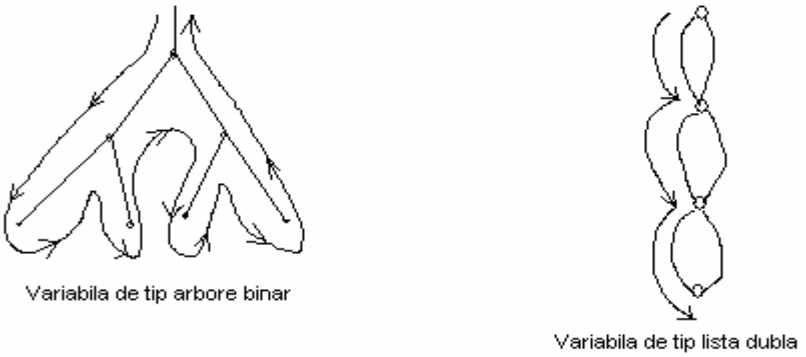

FIGURA 1 - Traversarea omogena

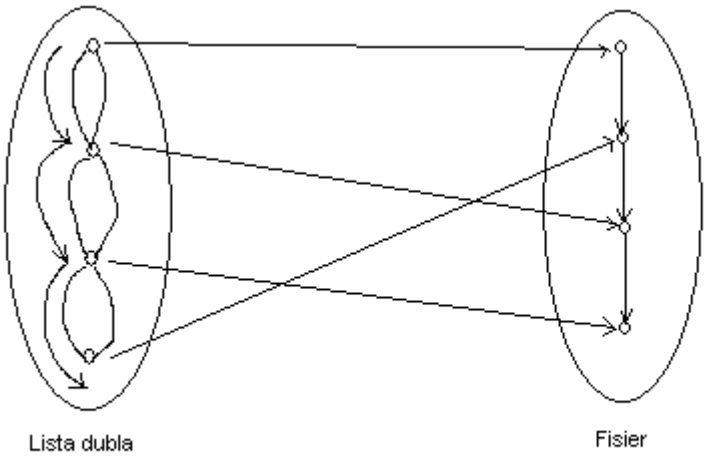

FIGURA 2 - Traversarea neomogena

Traversarea neomogena presupune definirea pointerilor pentru autoreferire si a pointerilor pentru referirea elementelor de alt tip din structura. De asemenea, sunt definite mecanismele care asigura trecerea de la un tip de variabila la altul, în ambele sensuri.

2. Fisiere complexe

Problema care apare în cazul folosirii diverselor tipuri de structuri de date este aceea ca logica programelor care le folosesc cere ca ele sa fie încarcate, de obicei, cu informatie utila. Pe de alta parte, folosirea structurilor de date complexe presupune o activitate de programare laborioasa, care se justifica în cazul în care este necesar sa se foloseasca cantitati de date considerabile. De aceea, a încarca manual aceste structuri de date la fiecare lansare a programului, devine un inconvenient în exploatare. De asemenea, datele care se prelucreaza în programe trebuie sa fie persistente de la o rulare a programului la alta. În aceste conditii, se pune problema stocarii datelor în fisiere, în vederea încarcarii lor o singura data pentru toate rularile ulterioare ale progamului.

Imaginea în memorie a structurilor de date, oricat de complexe ar fi ele, este liniara, memoria fiind accesata la nivelul cel mai de jos ca un vector. Tinând cont de acest lucru, poate fi folosit ca mediu de memorare discul.

Spre exemplificare, putem considera structura fisierelor ce stocheaza tabele în Dbase – FoxPro (cu extensia dbf). În FoxPro, o tabela este memorata într-un fisier, însotit, eventual, de unul în care se memoreaza câmpurile *MEMO* si altul de memorare a indexurilor.

Tabelul 1. Structura tabelelor

Octetul	Descriere	Explicatii
0	Numar de versiune	*1
1-3	Data ultimei operatii de update	
4-7	Numarul de înregistrari din partea de date	*14
8-9	Lungimea structurii de header (16 biti)	
10-11	Lungimea fiecarei inregistrari	*2
12-13	Rezervat	*3
14	Flag pentru tranzactie incompleta	*12
15	Flag de criptare fisier	*13
16-27	Rezervat pentru baze de date multiuser (dBASE III +)	
28	Flag pentru existenta fisier de tip MDX	*4
29	Driverul de limbaj	*5
30-31	Rezervat	*3
32-..n	Zona de descriere a câmpurilor	*7
n+1	Terminator (0Dh)	
m+263	Database container	*15
m+264 - …	Inregistrari	
	EOF	*11

Tabelul 2. Structura zonei de descriere a câmpurilor

Octetul	Descriere	Explicatii
0-10	Numele câmpului (sir ASCII terminat cu 00h)	
11	Tipul câmpului (sir ASCII)	
12-15	Adresa datelor câmpului (în memorie !!!) (dBASE III)	*6

16	Lungimea câmpului (binar)	
17	Numarul de zecimale (binar)	
18-19	Rezervat pentru baze de date dBASE multiuser	
20	Identificatorul ariei de lucru	
21-22	Rezervat pentru baze de date dBASE multiuser	
23	Flag pentru comanda SET FIELDS	
24-30	Rezervat	
31	Flag de câmp indexat	*8

Tabelul 3. Structura unei înregistrari

Octetii	Descriere	Explicatii
0-1	Flag de câmp sters	*9
1-…	Date	*10

*1: **dBase III**:
- 03h: fisier fara DBT;
- 83h: fisier cu DBT (de exemplu bitii 0-2 numarul de versiune, bitii 3-5 SQL, bitul 7 DBT flag).

dBase IV:
- 0-2: Numar de versiune;
- 3: Prezenta câmpului memo;
- 4-6: Prezenta tabelei SQL;
- 8: flag DBT.

Exemple:
03h: fisier dBase III cu câmp memo;
04h: fisier dBase IV cu câmp memo;
05h: fisier dBase V cu câmp memo;
F5h: fisier dBase III+ cu câmp memo
30h: fisier Visual FOXPro de tip DBC

*2: Suma lungimii tuturor câmpurilor + 1;
*3: (dBase IV) initializat cu 00h;
*4:**dBase IV**, tabela indexata/multiindexata:
- 01h: Fisier MDX prezent;
- 00h: Nu exista fisier MDX, indexat la cerere.

FoxBase:
- 01h: Fisier de tip CDX prezent;
- 00h: Nu exista fisier CDX.

Visual FoxPro:
- 02h: Contine MEMO;
- 04h: Exista definit fisier DBC (database container);
- 07h: Exista DBX incluzând câmpuri MEMO si indexuri.

*5: **FoxPro**, coduri de pagina (de tastatura)
- 01h: DOS USA code page 437;
- 02h: DOS Multilingual code page 850;
- 03h: Windows ANSI code page 1251;
- C8h: Windows EE code page 1250.

*6: **FoxPro** 12-13: deplasarea câmpului fata de începutul înregistrarii.
*7: **FoxPro**, **Clipper**: lungimea câmpului pentru câmpuri nenumerice.
*8: **dBase IV**: flag pentru câmp indexat
- 00h: Nu exista cheie definita pentru câmp;
- 01h: Exista cheie definita.

*9: 2Ah înregistrare stearsa;
20h înregistrare valida.
*10: Nu exista separatori de câmp pentru final de înregistrare
*11: **dBase II**, trateaza orice caracter de sfârsit de fisier (valoare 1Ah) ca sfârsit de fisier. DBase III nu trateaza caracterul 1Ah ca sfârsit de fisier, dar adauga la sfârsitul fisierului un caracter 1Ah.
*12: **dBase IV**: flag de tranzactie
- 00h: tranzactie încheiata cu succes;
- 01h: tranzactie începuta.

*13: **dBase IV**: flag de criptare
- 00h: fisier necriptat
- 01h: fisier criptat

*14: Stocare în binar; de exemplu, valoarea este generata ca byte#1 + (byte# *256) + (byte#3*256)*256 + …
*15: **Visual FoxPro**: câmp DBC (Database Container) de 263 bytes inclus în structura de header.

Tabelul 4. Structura fisierului care stocheaza câmpurile MEMO

Pozitia	Descriere	Explicatii
0-3	Adresa urmatorului bloc disponibil pentru adaugari de date	
4-7	(Rezervat) Dimensiunea blocurilor	*1
8-511	(Rezervat) Valori aleatoare	
512-…	Bitii primului bloc	
	Terminator de camp (1Ah)	
	Terminator de camp (1Ah)	
	Nefolositi	
1024	Urmatorul bloc	

*1: **FOXBase**, **Dbase IV**: dimensiunea blocurilor din fisierul MEMO (utilizat de comanda SET BLOCKSIZE), implicit 512 bytes.

3. Regruparea si regasirea informatiei în fisiere complexe

Header-ul fisierelor cu extensia *dbf* contin si informatii în scopul accesarii corecte a tuplurilor din relatie dupa cum s-a precizat anterior. Cunoasterea structurii antetului este foarte importanta în cazul în care fisierul cu extensia *dbf* se utilizeaza independent de FoxPro. Regasirea informatiei în acest caz poate avea doua sensuri. *Sensul consacrat*, presupune cautarea unei înregistrari sau a unui grup de înregistrari dupa un anumit criteriu. Acest lucru se realizeaza prin folosirea unei cantitati de memorie suplimentara necesara stocarii unor structuri de date ajutatoare si construirea de algoritmi corespunzatori care sa exploateze structurile. Cea mai utilizata structura de date este *B* tree. Aceasta a fost adaptata pentru a putea fi utilizata si în cazul unor noi tipuri de date cum ar fi *hcC* tree pentru regasirea obiectelor, *R* tree si *X* tree pentru regasirea datelor spatiale etc. O alta modalitate de regasire a informatiilor este cea care *se bazeaza pe calculul de adresa*, prin intermediul functiilor *hash*.

Având în vedere contextul problemei propuse, regasirea informatiilor din tabela presupune separarea si interpretarea header-ului fisierului, apoi accesarea înregistrarilor propriu-zise. Ca si la structurile de date dinamice care au informatie utila si de legatura, fisierele cu extensia *dbf* contin, la rândul lor, înregistrarile propriu-zise (datele utile) si informatii structurale.

4. Proceduri assembler pentru accesarea înregistrarilor din fisiere *dbf*

```
model small
dosseg
.data
   lung_header dw ?
   lung_inreg dw ?
   lung_bloc dw ?
   lung_camp_memo dw ?
   err_desch_d db "Fisierul DBF nu poate fi
deschis!$"
   err_cit_d db "Eroare la citirea din fisierul DBF!$"
   err_desch_m db "Fisierul FPT (memo) nu poate fi
deschis!$"
   err_cit_m db "Eroare la citirea din fisierul FPT
(memo)!$"
;macrodefinitie pentru afisare sir de caractere
afis_sir MACRO off_sir
   push ax
   push dx
   mov dx,off_sir
   mov ah,9
   int 21h
   pop dx
   pop ax
ENDM
;**************************************
cit_directa1 MACRO pointerh, pointerl, nr_octeti,
buffer
;macrodefinitie pt. pozitionare si citire din fisier ;(pt.
proc extrag_memo)
   mov cx,pointerh
   mov dx,pointerl
   mov al,0
   mov ah,42h    ;pozitionare in fisier la CX:DX
```

```
   int 21h
   mov cx,nr_octeti
   mov dx,buffer
   mov ah,3Fh  ;citeste din fisier
   int 21h
   jc em_err_citire
   cmp ax,nr_octeti
   jne em_err_citire
ENDM
;**************************************
cit_directa2 MACRO nr_octeti, buffer
;macrodefinitie pentru pozitionare si citire din fisier
;(CX:DX deja pregatit) cu salvarea pointerului din
;CX:DX (pt. proc extrag_memo)
   mov al,0
   mov ah,42h     ;pozitionare in fisier la CX:DX
   int 21h
   push dx
   push ax ;salveaza pointerul rezultat in DX:AX
   mov cx,nr_octeti
   mov dx,buffer
   mov ah,3Fh                    ;citeste din fisier
   int 21h
   pop dx
   pop cx       ;restaureaza pointerul din CX:DX
   jc em_err_citire
   cmp ax,nr_octeti
   jne em_err_citire
ENDM
;**************************************
.code
PUBLIC
extrag_inreg,extrag_memo,conv_BCD_binar
PROC extrag_inreg
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  Extragere inregistrare din DBF  ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;parametrii primiti pe stiva sunt:
;SP+10 : segment adresa nume fisier DBF
;SP+ 8 : offset  adresa nume fisier DBF
;SP+ 6 : numarul inregistrarii ce se va extrage
;SP+ 4 : segment adresa bufer in care se va ;pune
inregistrarea
;SP+ 2 : offset  adresa bufer in care se va pune
;inregistrarea
   push bp
   mov bp,sp
   push ax
   push bx
   push cx
   push dx
   pushf
   push ds
   mov ds,[bp+12]
   mov dx,[bp+10]
;DS:DX <- adresa numelui fisierului
   mov al,0          ;deschidere in mod read-only
   mov ah,3Dh
   int 21h           ;deschide fisierul DBF
   pop ds
   jc ei_err_deschidere
   mov bx,ax             ;BX<-handler fisier DBF
   mov cx,0
    mov dx,8       ;CX:DX<-deplasament in header
;pentru  lungime header
   mov al,0
   mov ah,42h    ;pozitionare in fisier la CX:DX
   int 21h
   mov cx,4
   mov dx,offset lung_header
    mov ah,3Fh         ;citeste 4 octeti la adresa DS:
lung_header
   int 21h   ;(lungime header si lungime inregistrare)
   jc ei_err_citire
   cmp ax,4
   jne ei_err_citire
    mov ax,[bp+8]   ;AX <- numar inregistrare care se
;extrage
   dec ax      ;AX <- numar inregistrari predecesoare
   mul lung_inreg
   add ax,lung_header       ;deplasament pana la
;inregistrare
   jnc ei_1
   inc dx                        ;DX<-DX+CF
ei_1:
    inc ax         ;sare peste deleted flag (1 octet din
;inregistrare)
   jnc ei_2
   inc dx                        ;DX<-DX+CF
ei_2:
   mov cx,dx
   mov dx,ax                     ;CX:DX<-DX:AX
   mov al,0
    mov ah,42h     ;pozitionare in fisier la inceput ;de
inregistrare
   int 21h
   mov cx,lung_inreg
   dec cx  ;CX<-lungime inregistrare fara deleted flag
   push ds
   mov ds,[bp+6]
   mov dx,[bp+4]          ;DS:DX<-adresa buffer
   mov ah,3Fh             ;citeste inregistrarea
   int 21h
   pop ds
   jc ei_err_citire
   inc ax
   cmp ax,lung_inreg
   jne ei_err_citire
   mov ah,3Eh             ;inchide fisier
   int 21h
   jmp ei_sf
ei_err_deschidere:
   afis_sir <offset err_desch_d>
   jmp ei_sf
ei_err_citire:
   afis_sir <offset err_cit_d>
   mov ah,3Eh             ;inchide fisier
   int 21h
```

```
ei_sf:
   popf
   pop dx
   pop cx
   pop bx
   pop ax
   pop bp
   ret 10
ENDP
;***************************************
PROC extrag_memo
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  Extragere camp memo din fisier FPT
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;parametrii primiti pe stiva sunt:
;SP+10 : segment adresa nume fisier FPT
;SP+ 8 : offset  adresa nume fisier FPT
;SP+ 6 : numarul blocului de la care incepe campul
;memo care se extrage
;SP+ 4 : segment adresa bufer in care se va pune
;campul memo
;SP+ 2 : offset  adresa bufer in care se va pune
;campul memo
;returneaza in AX lungimea campului memo
   push bp
   mov bp,sp
   push bx
   push cx
   push dx
   pushf
   push ds
   mov ds,[bp+12]
    mov dx,[bp+10]        ;DS:DX <- adresa numelui
;fisierului
   mov al,0           ;deschidere in mod read-only
   mov ah,3Dh
   int 21h            ;deschide fisierul FPT
   pop ds
   jc em_err_deschidere
   mov bx,ax
   cit_directa1 0,7,1,<offset lung_bloc>
    cit_directa1 0,6,1,<offset [lung_bloc+1]>  ;citeste
lung. unui bloc
    mov ax,[bp+8]      ;AX<-nr. blocului de la care
;incepe campul
   mul lung_bloc
   add ax,7        ;deplasament valoare lungime camp
   jnc em_1
   inc dx
   jmp em_1
em_err_deschidere:
   afis_sir <offset err_desch_m>
   jmp em_sf
em_err_citire:
   afis_sir <offset err_cit_m>
   mov ah,3Eh             ;inchide fisier
   int 21h
   jmp em_sf
em_1:
   mov cx,dx
   mov dx,ax
    cit_directa2 1,<offset lung_camp_memo>  ;citeste
;octetul mai putin semnificativ
   dec dx             ;din valoarea lungimii campului
   jnc em_2
   dec cx
em_2:
   cit_directa2  1,<offset  [lung_camp_memo+1]>
;citeste octetul mai semnificativ
   add dx,2           ;avansez pointerul cu 2 octeti
   jnc em_3
   inc cx
em_3:
   mov al,0
    mov ah,42h            ;aduce pointerul la inceputul
;campului memo
   int 21h
   mov cx,lung_camp_memo
   push ds
   mov ds,[bp+6]
    mov dx,[bp+4]      ;DS:DX<-adresa buffer pentru
;campul memo
   mov ah,3Fh         ;citeste campul memo
   int 21h
   pop ds
   jc em_err_citire
   cmp ax,lung_camp_memo
   jne em_err_citire
    push ax              ;salveaza lungimea campului
;(continuta in AX)
   mov ah,3Eh         ;inchide fisier
   int 21h
   pop ax
em_sf:
   popf
   pop dx
   pop cx
   pop bx
   pop bp
   ret 10
ENDP
;***************************************
PROC conv_BCD_binar
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Conversie din BCD despachetat in binar
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;parametrii primiti pe stiva sunt:
;SP+6 : segment adresa sir BCD
;SP+4 : offset  adresa sir BCD
;SP+2 : lungime sir BCD
;returneaza in AX valoarea numerica in binar
   push bp
   mov bp,sp
   push bx
   push cx
   push dx
   push di
   push si
```

```
   pushf
   push ds
   mov ax,0
   mov ds,[bp+8]
   mov di,[bp+6]     ;DS:DI<-adresa sir BCD
   mov si,10         ;SI<-multiplicatorul zecimal
   mov cx,[bp+4]     ;CX<-lungime sir BCD
conv_cit_cifra:
   mov bl,[di]       ;BL<-caracterul curent din sir
   cmp bl,20h        ;compara cu spatiu
     je conv_urmat_cifra        ;daca e spatiu, atunci
;trece la urmatorul caracter
   mul si            ;AX<-AX*10
   sub bl,30h        ;conversie ASCII->binar
   mov bh,0
   add ax,bx         ;AX<-AX+cifra curenta
conv_urmat_cifra:
   inc di            ;trece la caracterul urmator
   loop conv_cit_cifra
   pop ds
   popf
   pop si
   pop di
   pop dx
   pop cx
   pop bx
   pop bp
   ret 6
ENDP
```

5. Concluzii

Implementarea oricarui algoritm de regasire si regrupare a informatiilor presupune cunoasterea structurii fiecarui articol, a fiecarui fisier, sau a fiecarei zone de memorie si reconstituirea legaturilor stocate. Performanta algoritmilor este influientata de volumul informatiilor pastrate în memoria interna si de numarul de operatii de intrare/iesire, de initializare sau modificare a continutului lor.
Programele *asm* exemplifica, într-un mod transparent, mecanisme de referire a pointerilor si a variabilelor de pozitie pe o structura de header de fisier.

Dezvoltarea oricarei alte aplicatii presupune preluarea structurii unui alt header si adaptarea acestor mecanisme de referire a câmpurilor si de accesare a articolelor.

Bibliografie

1. **Bachmann, E.**, *File format description*, Erik Bachmann. – Roskilde, Denmark: Clickety Click Software, 1996 – 1997. ASCII text file URL: http://www.geocities.com/SiliconValley/Pines/2563/Xbase.htm.
2. **Belkin, J.N., Craft, W.B.**, *Information filtering and information retrieval: Two sides of the same coin?*, Commun. A.C. M., vol 35, Nr. 12, (decembre 1992) pag 29-39.
3. **Ivan, L.**, *Programare în limbaj de asamblare – Culegere de probleme*, Ed. Inforec, Bucuresti, 1997.
4. **Musca, Ghe.**, *Programare în limbaj de asamblare*, Ed., Teora, Bucuresti, 1996.
5. **Sanchez, J., Canton, M.**, *Numerical Programming The 387, 486 and Pentium*, McGraw Hill Inc., New York, 1995.
6. **Somnea, D., Vladut, T.**, *Programarea în Assembler*, Ed., Tehnica, Bucuresti, 1992.
7. **Swan, T.**, *Mastering Turbo Assembler*, SAMS Publishing, Indianapolis, 1995.
8. **Wang, T.**, *What FoxPro Checks When Opening a .DBF File*. – (Microsoft Knowledge Base.; Article Q119763) URL:http://www.microsoft.com/kb/developr/ fox/q119763.htm.
9. **Wyatt, A.**, *Using Assembly Language*, QUE Corporation Carmel – Indiana, 1987.

.