PC magazin

REVISTA INDEPENDENTA DE INFORMATICA

- STRATEGII IN INTELIGENTA ARTIFICIALA
- LIMBAJUL C (IV)
- RETELE LOCALE
- WINDOWS 3.0 IN COMPETITIE CU PRESENTATION MANAGER?
- NOUTATI SOFTWARE

ADISAN

nr. 4 / 1990

Colectivul de redactie al revistei

Tiparul a fost executat la Tipografia Universul
Cda. 2402
1990

CUPRINS

- STRATEGII FUNDAMENTALE IN INTELIGENTA ARTIFICIALA SI PROGRAMAREA LOGICA (IV)..... 3
- LIMBAJE DE PROGRAMARE IN DESIGNUL SISTEMELOR DE OPERARE SI AL APLICATIILOR - LIMBAJUL C (IV)..... 16
- PROGRAMAREA CONSOLEI SISTEM (TASTATURII) IN PROLOG (III).....21
- MICROCALCULATOARELE CU ARHITECTURA DE MULTIPROCESARE RIVALIZEAZA CALCULATOARELE MARI..... 33
- INITIERE IN CONCEPTELE COMUNICATIEI DE DATE (III)..... 36
- 1990 - ANUL RETELELOR LOCALE..... 41
- HOBBY..... 43
- EDITOARE SI FONTURI (III)..... 46
- FISIERE INDEXATE..... 51
- WINDOWS 3.0 IN COMPETITIE CU PRESENTATION MANAGER?.....62
- APLICATII IN LIMBAJUL MASINA Z80 PE CALCULATOARE COMPATIBILE SPECTRUM(I).....64
- A SIMULA ? NIMC MAI SIMPLU..... 66
- NOUTATI SOFTWARE 70
- ACTUALITATI ANTIVIRUS PENTRU MACINTOSH..... 72
- NOUA GENERATIE DE PRODUSE SOFTWARE FOLOSITE IN MATEMATICA..73


ADISAN

STRATEGII FUNDAMENTALE IN INTELIGENTA ARTIFICIALA SI PROGRAMAREA LOGICA (IV)

Adrian Negru,
Alexandru Babin

Algoritmul Backtracking, Taieturi si negatii

IV.1 Algoritmul mersului inapoi sau Backtracking

Am vazut, pina acum, ca scopul unui program logic este incercarea de satisfacere a realizarilor cerute sau raspunsul la intrebari. In incercarea de satisfacere a realizarii, baza de date, formata din fapte si functori, este cercetata pentru realizarea unificarii cu clauza-realizare putind intilni urmatoarele situatii:

i) Un fapt, sau capul unei reguli clauzale, care unifica poate fi gasit (reamintim ca o regula clauzala de forma R:-R1,...,Rn este delimitata semantic prin capul regulei (R) si corpul regulei (R1,...,Rn) fiecare Ri fiind o alta regula clauzala formata din cap si corp etc.).

In urma unificarii vom instanta toate variabilele neinstantate anterioare care apar in corpul realizarii si care au fost unificate cu argumentele faptului **marcat** in baza de date. Daca avem o conjunctie de reguli R1,...,Rn, cerute ca realizare, se incearca satisfacerea regulei cea mai din stinga mai intii (R1, in cazul nostru), unificarea cu fapte din baza, deci realizarea ei, determinind incercarea de realizare a regulei imediat urmatoare din partea dreapta (R2, in cazul nostru) etc.

ii) Se poate intimpla ca, la un moment dat, o realizare clauzala Rk sa nu poata fi unificata cu nici un fapt din baza, deci procesul de unificare a lui Rk cu baza de date raporteaza **esec** (*fail*). Atunci se incearca mersul inapoi prin asa numita **incercare de satisfacere** a realizarii anterioare, Rk-1.

Sa vedem acum in ce consta acest proces de resatisfacere a unei realizari. Intorcindu-ne inapoi la o realizare reusita Rk-1, primul pas pe care-l facem este anularea instantarii prin unificare a variabilelor ce apar in realizarea clauzala Rk-1, pornind sa cautam in baza de date o noua instantare reluind procesul de cercetare dupa faptul marcat care realizase anterior unificarea lui Rk-1 cu el. Aceasta cautare poate gasi un nou fapt cu care sa realizeze unificarea, deci realizarea lui Rk-1 ducind la o noua incercare pentru Rk, sau, poate raporta esec (*fail*), ceea ce determina mersul inapoi la realizarea clauzala anterioara, Rk-2, careia ii anuleaza instantarea obisnuita prin unificarea cu un fapt din baza de date si reincepe cautarea unei noi unificari dincolo de faptul care o instantase anterior etc.

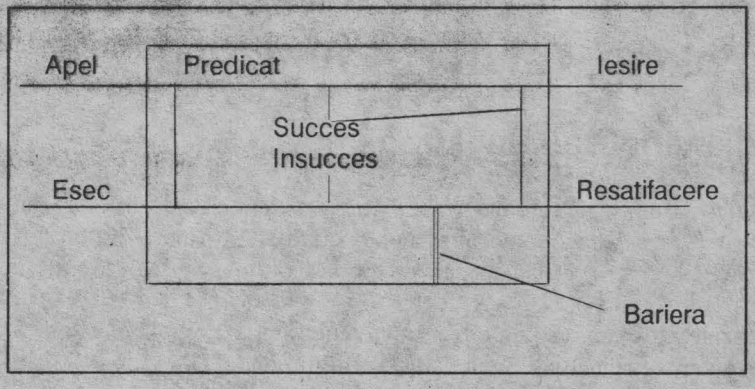
Aceasta descriere este esenta principiului de *backtraking* pe care-l vom discuta pe larg in continuare, sub aspectul teoretic si, mai ales, in situatiile concrete intilnite in programe logice.

Procesul de *backtraking* este mai eficient daca selectia regulei pentru realizare nu se face arbitrar si este ghidata de informatii despre posibilitatea alegerii celei mai bune miscari care, daca exista, determina ordonarea in consecinta a realizarilor, reducind mult **mersul inapoi**.

Vom distinge, in fapt, doua modalitati de mers inapoi si anume: mersul inapoi in adincime sau *backtraking* in adincime care se aplica in momentul in care unificarea unei ultime clauze - fapt din baza de date cu o realizare - esueaza si controlul este redat unei realizari anterioare din sirul realizarilor cerute; cealalta modalitate cunoscuta ca mersul inapoi se produce daca unificarea dintre o realizare si un fapt clauzal esueaza si atunci un alt fapt clauzal este ales.

In sensul acestor definitii este, citeodata, convenabil sa se introduca predicate de test ca prime incercari de unificare, predicate care apar la inceput in corpul unor reguli clauzale din program, *backtraking*-ul initiat de esecul lor putind fi clasificat drept **mers inapoi** .

Sa ne inchipuim un program logic ca o cutie cu patru porti sau fante de intrare si iesire.



Cind un predicat sau realizare este invocata, intra in cutie prin poarta **Apel** iar daca raporteaza succes iese prin poarta **Iesire**. Succesul implica instantarea prin unificare a variabilelor neinstantate ce intra la apel odata cu predicatul din cutie. Variabilele instantate ramin in cutie, in caz de succes, pina la un nou apel. Daca procesul de unificare nu poate avea loc, atunci ce iese prin poarta **Esec** are drept consecinta si anularea instantarii variabilelor care erau rezident instantate in cutie, intrate cu un apel anterior prin poarta **Apel**.

Ce se intimpla cu realizarea care a raportat esec si iese prin poarta **Esec** si reintra la un nou apel prin poarta **Resatisfacere**?

Ce semnificatie are atunci bariera pusa in calea iesirii pe poarta **Esec** a unei realizari ce, odata esuata, reintra prin poarta **Resatisfacere** in incercarea unei noi unificari? Vom vedea ca ea joaca tocmai rolul taieturii in mecanismul de *backtraking* ce impiedica anumite realizari de a reincerca o noua satisfacere in urma unui esec. Notatia clasica pentru taietura in programarea logica este "!". Ca esenta avem definitia "Nu se poate merge inapoi peste o taietura" adica, in momentul in care intr-un sir de realizari clauzale avem secventa R1, ... , Rk, !, Rk + 1, ... , Rn, nerealizarea unei clauze Rj, j > k, permite mersul inapoi pina la realizarea Rk + 1, unde intilnind "!", procesul se opreste, realizarile R1,..., Rk, odata raportate ca succes, nemaifiind reinstantate de *backtraking*, taietura "!" prezenta dupa ele marcind valoarea absoluta a substitutiilor prin care s-au realizat in procesul de unificare.

Sa consideram urmatoarea baza de date data de predicatul:

```
adresa(persoana,strada).
```

si baza:

```
adresa(Adi,1Mai)
adresa(Cornel,Turda)
adresa(Ioana,1Mai)
adresa(Irina,1Mai)
```

ca si urmatorul program logic:

```
vecini(Persoana1,Persoana2,Strada) :- adresa(Persoana1,Strada),
                                      adresa(Persoana2,Strada),
```

Persoana1 < > Persoana2.

deci doua persoane sint vecine daca locuiesc pe aceeasi strada. Sa vedem cum actioneaza mecanismul de *backtraking* asupra satisfacerii realizarii:

(1) vecini(P1,P2,1Mai)

adica gasirea in baza de date, a perechilor (P1,P2) care satisfac 1), adica sint vecini pe strada 1Mai. Facem observatia ca faptele in baza de date sint luate in ordinea aparitiei lor.

PAS 1:

Pasul 1 s-a realizat astfel: prima clauza, adresa (P1,1Mai), a unificat cu baza de date, si anume cu primul fapt pe care l-a si marcat, instantind P1 cu Adi si; raportind succes, a trecut la unificarea celei de a doua clauze, adresa(P2,1Mai) care acceseaza din nou baza de date global, deci unifica din nou cu primul fapt

1		2		3	Comentariu
Adresa (P1, 1Mai) | Adi Adresa (Adi, 1Mai) Adresa (Cornel, Turda) Adresa (Ioana, 1Mai) Adresa (Irina, 1Mai)	si	Adresa (P2,1Mai) | Adi Adresa (Adi, 1Mai) Adresa (Cornel, Turda) Adresa (Ioana, 1Mai) Adresa (Irina, 1Mai)	si	P1 < > P2 | | Adi Adi	aleg din baza Esec (Fail!)

adresa (Adi,1Mai), deci instanteaza P2 cu {P2 = Adi} deci, raportind succes, trece la incercarea de unificare a celei de a treia clauze, P1 < > P2 care, prin unificarile anterioare, a devenit Adi < > Adi, fals. Acum incepe mecanismul de *backtraking*.

Se anuleaza instantarea {P2 = Adi} in clauza anterioara lui P1 < > P2 si anume in adresa(P2,1Mai) care devine acum neinstantata in variabila P2.

In acest moment se incearca unificarea lui adresa(P2,1Mai) cu un fapt in baza de date, dupa faptul adresa(Adi,1Mai) cu care unificase anterior.

PAS 2:

adresa(Adi,1Mai) si adresa(P2,1Mai) si Adi ≠ P2

unifica | | unifica succes

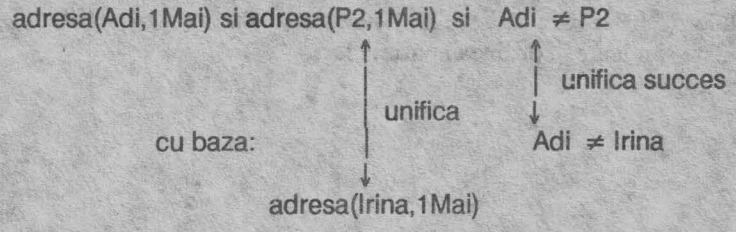
cu baza: Ioana Adi ≠ Ioana

adresa(Cornel,Turda)
adresa(Ioana,1Mai)
adresa(Irina,1Mai)

Deci P2 unifica cu Ioana raportind succes si trecind la ultima clauza P1 < > P2 ea a devenit, in urma instantarilor anterioare, Adi < > Ioana deci se raporteaza succes, sistemul raportind realizarea vecini(Adi,Ioana,1Mai).

Cum s-a ajuns la ultima clauza, dar baza de date nu a fost exhaustata, intervine actiunea *backtraking*-ului in adincime care anuleaza din nou instantarea clauzei P1 < > P2 restaurind variabila P2 si clauza anterioara, adresa(P2,1Mai), pe care cauta sa o unifice cu un fapt din baza de date dincolo de ultimul fapt marcat, adresa(Ioana,1Mai) cu care unificase anterior.

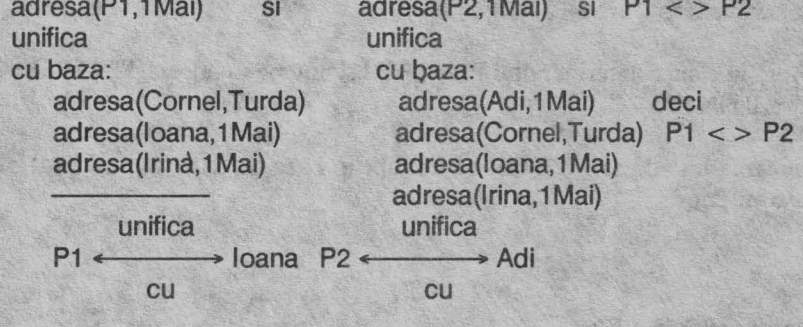
PAS 3:



sistemul raportind o noua realizare vecini(Adi,Irina,1Mai).

Din nou succesul actioneaza datorita epuizarii clauzelor de unificare, mecanismul de *backtraking* care anuleaza ultima instantare a lui P2 si reia incercarea de unificare a lui adresa(P2,1Mai) cu baza de date dincolo de ultimul fapt marcat. Cum insa baza de date a fost epuizata, ultimul fapt marcat, adresa(Irina,1Mai), cu care adresa(P2,1Mai) s-a unificat este si ultimul fapt testat. Deci, raportind esec in incercarea de unificare a lui adresa(P2,Adi) cu baza, mecanismul de *backtraking* actioneaza asupra clauzei imediat din stinga: adresa(Adi,1Mai) obtinuta prin instantarea {P1 = Adi}, asupra careia actioneaza prin restaurarea variabilei P1 anulind instantarea acesteia si incercarea de unificare a clauzei adresa(P1,1Mai) cu baza de date dincolo de faptul marcat adresa(Adi,1Mai) care realizase unificarea anterioara.

PAS 4:



Sa observam ca daca adresa(P1,1Mai) a unificat cu baza de date dincolo de faptul marcat prin unificarea anterioara in schimb adresa(P2,1Mai) isi cauta un fapt cu care sa unifice cercetind baza de date de la inceput (odata ce o clauza exhausteaza prin unificari recursive baza de date, procesul de unificare se reia de la inceput).

Se raporteaza succes cu realizarea vecini(Ioana,Adi,1Mai) si procesul de *backtraking* este reluat. Sa observam ca, asa cum decurge, pot fi generate, prin mersul inapoi, solutii care au mai fost nominalizate in pasi anteriori, deci se intrevede necesitatea punerii unei "bariere" in calea reluarii unor unificari de clauze ce nu conduc cu siguranta la solutii noi ci pot repeta realizari deja raportate de sistem.

Acesta este modul de analiza *backtraking* pe care-l proceseaza in linii generale orice sistem al inteligentei artificiale caruia i s-a implementat structural algoritmul.

Acest mecanism poate genera o multime finita sau infinita de solutii depinzind de modul de alegere al clauzelor. Sint situatii in care dorim generarea unui numar infinit de realizari nu din necesitatea de a le obtine pe toate ci datorita faptului ca nu stim cu exactitate de cite anume solutii avem nevoie pentru a ne satisface cererile. In acest moment ne trebuie sigur o relatie recursiva ce va apela o anumita clauza direct sau indirect din ea insasi. Sa consideram predicatul nr_natural (N) care va raporta succes daca N este instantat cu un numar natural. Sa formulam definitia clauzala:

(1) nr_natural(0).

(2) nr_natural(X) :- nr_natural(Y), X = Y+1.

care la cererea nr_natural(X)? va produce sirul infinit de realizari ce constituie enumerarea elementelor multimii numerelor naturale N, {0,1,2,...,n,...}.

De fapt, sistemul va raporta doar atitea solutii cite numere incap in memoria calculatorului, apelul recursiv al clauzei (2) pastrind fiecare instantare anterioara in memorie din cauza legarii ei de recursivitatea apelului la ea insasi.

O schita a arborelui de descompunere marcat de *backtraking* este:

nr_natural(X)?

apel la faptul (1)
ne da solutia X=0 regula (2)

apel recursiv

apel la faptul (1)
cu solutia X = 1 regula (2)

apel recursiv

apel la faptul (1)
cu solutia X = 2 regula (2)

etc.

Avem, la inceput, a alege intre regula 1 si 2 ca raspuns la intrebare. Alegind (1) avem instantarea {X = 0} si se reia procesul de *backtraking*. Acum alegem regula (2) si facem o alegere de instantare cu realizarea data; astfel, alegind din regula (1) se obtine instantarea {X = 1}; altfel, folosim regula (2) din nou trebuie sa alegem cum satisfacem noua realizare etc. In fiecare faza sistemul alege faptul (1) si, sub *backtraking*, insatisface ultima alegere si, de fiecare data cind opereaza, anularea instantarii merge inapoi acolo unde alesese ultima regula 1 si alege, in locul ei, regula 2. Odata decizia luata asupra ei, o noua realizare este introdusa satisfacuta de faptul (1) si asa mai departe.

Mecanismul acesta este pastrat in majoritatea compilatoarelor PROLOG care adopta o strategie de ordonare tip stiva in sensul ca rezolvenții este pastrat ca o stiva: se reactualizeaza realizarea din virful stivei pentru reducerea ei prin rezolutie cu rezolventul si salveaza pe stiva realizarea derivata. Pe linga politica de stiva, Prologul simuleaza alegerea nedeterminata a clauzelor pentru reducere prin cautare si *backtraking*. Astfel, in incercarea de reducere a unei realizari, este aleasa prima clauza al carei cap unifica cu realizarea, iar in cazul in care nici o clauza nu unifica cu realizarea extrasa din stiva, calculul este depus pina la ultima alegere posibila din baza si urmatoarea clauza unificabila este aleasa.

Vom introduce notiunea de **trasa** a unui program logic ce se poate defini ca un sir de perechi (Ri,ui) unde ui sint elemente ale celui mai general unificator Ui calculat la pasul de reducere de nivel i, restrictionat la variabilele in realizarea Ri (Ri sint realizari).

Notiunea de pas de reducere a aparut ca urmare a procesului de *backtraking* asupra unui program logic a carui realizare progreseaza prin reducerea de realizari. Astfel, in fiecare stadiu al abordarii programului, exista un rezolvent sub forma de conjunctii de realizari ce trebuiesc demonstrate.

Se alege o realizare a rezolventului si o clauza a programului logic astfel ca sa unifice capul clauzei cu realizarea.

In acest moment vom continua cu un nou rezolvent obtinut prin inlocuirea realizarii gasite cu corpul clauzei in rezolvent si aplicind cel mai general unificator dintre capul clauzei si realizare. Procesul se termina cind rezolventul este multimea vida, caz in care se spune ca realizarea a fost satisfacuta de program. Adoptind o notatie algebrica putem formula calculul unei realizari ca un sir de tripleti (Xi,Ri,Cli) unde Xi este o realizare conjunctiva, Ri este o realizare din Xi si Cli este o clauza (A:-B1,...,Bn) din program redenumita astfel ca sa obtina simboluri de variabile ce nu apar in realizarile Xk, $0 \leq k \leq i$. Pentru orice $k > 0$, $Xk + 1$ este rezultatul inlocuirii lui Rk cu corpul lui Clk in Xk si aplicarea substitutiei uk, cel mai general unificator intre Rk si Ak, uk lui Clk.

Se poate intimpla ca, daca Rk este singura realizare din Xk si corpul lui Clk este vid, sa obtinem succes sau insucces (daca Rk si capul lui Clk nu unifica). Astfel, daca formalizam un program logic sub forma multimii tripletilor (Xi,Ri,Cli), notiunea de trasa apare imediat ca multimea perechilor (Ri,ui) cu ui unificator la pasul de reducere i raportat la variabilele din Ri.

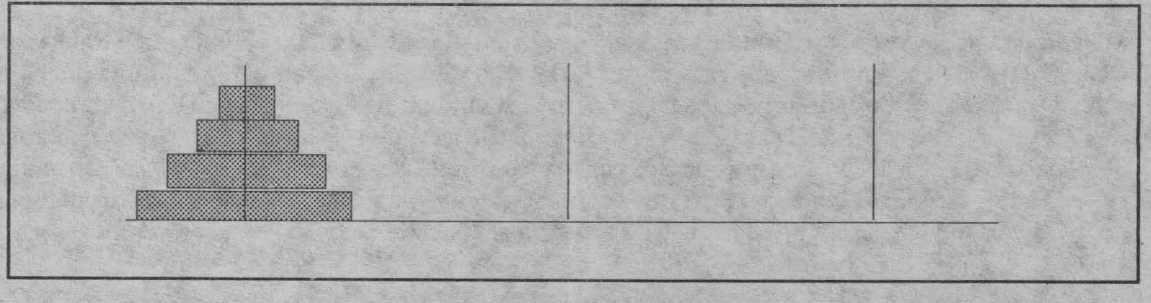
Sa consideram raportul care realizeaza problema turnurilor din Hanoi. Aceasta veche problema chinezeasca se poate formula astfel:

Se dau trei tije alaturate ca in figura:

si un numar de discuri de diametre descrescatoare asezate toate pe tija A, ca in figura. Problema revine la a muta discurile de pe tija A pe tija C in aceeasi ordine, cu restrictia ca de fiecare data se muta un singur disc iar niciodata un disc cu diametrul mai mare nu poate fi pus peste unul cu diametru mai mic, tija B putind fi folosita ca pozitie intermediara in procesul de transfer al discurilor. Una din strategiile de rezolvare consta in pasii:

- un singur disc se muta direct;
- n discuri se muta in trei etape;
- se muta n-1 discuri pe tija B;
- se muta ultimul disc pe tija C;
- se muta cele n-1 discuri de pe B pe C.

Vom prezenta doua variante ale acestui program, una directa si alta rezolvabila cu ajutorul listelor, asupra careia ii vom pune in evidenta trasa. Prima varianta o vom prezenta ca program complet scris in TURBO PROLOG.



```
domains
loc = dreapta;mijloc;stinga
predicates
hanoi(integer)
muta(integer,loc,loc,loc)
informez(loc,loc)
clauses
hanoi(N) :- muta(N,stinga,mijloc,dreapta).
muta(1,A,_,C) :- informez(A,C),!.
muta(N,A,B,C) :-
N1=N-1,muta(N1,A,C,B),informez(A,C),muta(N1,B,A,C).
informez(Loc1,Loc2) :-
write("\nMuta un disc din",Loc1,"in",Loc2).
```

Predicatele folosite sint hanoi (N) in care N specifica numarul de discuri cu care se lucreaza, muta(N,stinga,mijloc,dreapta) care specifica numarul de discuri ce se muta din stinga pe dreapta utilizind ca stadiu temporar tija de mijloc (numerotarea fiind A, B, C, ca in figura). Iar informez foloseste predicatul intern write care afiseaza pe display-ul calculatorului mesajul dintre ghilimele).

Sa definim, inaintea scrierii celei de a doua variante a programului Hanoi, un program logic care defineste un algoritm larg folosit in programarea conventionala si anume concatenarea a doua liste, rezultatul fiind o lista unica obtinuta prin juxtapunerea componentelor. In mediul informaticii procesul este cunoscut ca append-area a doua liste (termenul append inseamna adaugare, in engleza). Putem defini, deci, programul care realizeaza predicatul adaug (l1s,l2s,l1sl2s) l1sl2s fiind concatenarea listelor l1s cu l2s. Avem:

(1) adaug([],Xs,Xs).

(2) adaug([Y | Ys],Xs,[Y | Zs]):-adaug(Ys,Xs,Zs).

Sa punem in evidenta o trasa a programului adaug prin punerea in evidenta a rezolventului. Astfel, intrebarea:

adaug([a1,a2],[a3,a4],lista)?

initializeaza rezolventul la adaug ([a1,a2],[a3,a4],lista) care este, de fapt, si singura realizare supusa reductiei. Este aleasa regula (2) din program. Unificarea dintre realizare si capul regulei se realizeaza prin substitutia {Y = a1,Ys = [a2],Xs = [a3,a4], lista = [a1 | Zs]}.

Noul rezolvent este instantarea lui adaug(Ys,Xs,Zs) sub unificator si anume adaug([a2],[a3,a4],Zs). Aceasta este si realizarea aleasa pentru urmatorul pas al recursiei. Este nevoie, inainte, de o renumerotare a variabilelor pentru a evita ambiguitatile. Alegem, astfel, regula 2 sub forma:

(2) adaug([Y | Ys'],Xs',[Y | Zs']) :- adaug(Ys',Xs',Zs').

Unificarea capului regulei cu realizarea ne da:

{Y' = a2;Ys' = [],Xs' = [a3,a4],Zs = [a2 | Zs']}

cu noul rezolvent:

adaug([],[a3,a4],Zs).

In continuare este aleasa regula (1) care este un fapt si, redenumind in (1) variabilele, avem:

(1) adaug([],Xs',Xs')

care unifica rezolventul sub instantarea:

{Xs' = [a3,a4], Zs' = [a3,a4]}.

Noul rezolvent este clauza vida, deci calculul s-a terminat. Rezultatul final consta in punerea cap la cap a unificatorilor obtinuti in diversele stadii de calcul si anume: lista s-a instantat cu [a1 | Zs]; apoi Zs cu [a2 | Z's] si Z's cu [a3,a4], deci:

```
lista = [a1|[a2],[a3,a4]] sau, echivalent:
lista = [a1,a2,a3,a4].
```

Trasa este imediata prin:

```
adaug([a1,a2],[a3,a4],lista)  lista = [a1|ls1]
adaug([a2],[a3,a4],ls1)       ls1   = [a2|ls2]
adaug([],[a3,a4],ls2)         ls2   = [a3,a4].
true
```

Iesirea: lista = [a1,a2,a3,a4].

Sa formulam acum programul turnurilor din Hanoi folosind un singur predicat hanoi si adaug, definit mai sus. Definim **hanoi(N,A,B,C,mutari)** cu N numarul de discuri, A,B,C tijele ca in figura si mutari = numarul de mutari necesare.

```
hanoi(1,A,C,B,[A pe C]).
hanoi(N+1,A,C,B,mutari) :-
hanoi(N,A,B,C,Ms1),
hanoi(N,B,C,A,Ms2),

adaug(Ms1,[A pe C|Ms2],mutari).
```

Semnificatia lui A pe C consta in mutarea discului superior de pe tija A pe tija C.

IV.2 Taieturi si negatii.

Definitie. Efect si utilizare.

In programarea logica exista un mecanism special folosit in orice implementare a limbajului PROLOG sub forma de predicat intern numit taietura sau *cut*, metoda ce afecteaza comportarea procedurala a oricarui program.

Functiunea sa principala este reducerea spatiului de cautare intr-un program logic spunind mecanismului de *backtraking* ce clauze nu mai trebuiesc luate in considerare din nou, in incercarea de realizare a lantului de cereri, eliminind sau inghetind siruri de cereri satisfacute a caror noua nesatisfacere prin instantare cu alte fapte decit pentru cele care au realizat odata nu duce la solutii noi sau repeta solutii deja considerate.

Folosirea acestui procedeu poate fi interpretata drept procedurala venind in contradictie cu aspectul declarativ al unui program logic, dar tehnica de utilizare poate imbunatati eficienta programului fara a reduce din lizibilitatea lui.

Aparitia acestui procedeu s-a datorat necesitatii de micsorare a timpului de executie a unui program prin eliminarea spatiului folosit in resatisfacerea unor realizari ce nu contribuie la noi solutii precum si din nevoia de a folosi eficient memoria calculatorului eliminind etape ale *backtraking*-ului ce stocheaza elemente care nu mai pot duce la alte realizari decit cele expuse de program.

Sintactic, taietura este un predicat fara argumente scris sub forma "!". In procesul de satisfacere a clauzelor este o realizare ce raporteaza succes odata intilnita dar care nu mai poate fi resatisfacuta prin backtraking.

Neintelegerea efectului de taietura este intotdeauna o sursa de greseli frecvent intilnite in programare, prin aceea ca se poate interpreta gresit atit procesul de inghetare a unor realizari pe care taietura le produce cit si delimitarea gresita a clauzelor absolute ce nu ar mai trebui sa participe la *backtraking*. Consideram atunci binevenita, chiar inainte de exemplificarea prin programe comentate, punctarea implicatiilor generate de taietura "!" intr-un program logic:

1) Intilnirea unei taieturi "!" ca realizare clauzala are loc atunci cind toate clauzele intre realizarea clauzala initiala sau parinte si taietura au fost realizate prin unificare;

2) Orice incercare de resatisfacere a clauzelor dintre parinte si taietura nu mai poate avea loc;

3) Daca o taietura este prezenta in partea dreapta a unor conjunctii clauzale atunci solutia care le instanteaza pina la taietura este unica pentru program;

4) Taietura nu afecteaza realizarile caluzale aflate in dreapta clauzei ce o contine, aceasta putind crea, in procesul de *backtraking*, solutii multiple. Orice realizare din dreapta taieturii esuata starteaza mersul inapoi pina la prima realizare clauzala de dupa ea ("!"), cele precedente lui "!" raminind cu solutii absolute;

5) Ca expresie formala sa consideram clauza:

CL = {P:-Q1,...,Qi,!,Qi+2,...,Qn}

ce realizeaza P. Daca o realizare curenta R unifica cu capul clauzei CL si Q1,...,Qi sint satisfacute cu succes atunci alta instantare care poate unifica cu R, pina aici, este ignorata. Daca mai departe Qt, $t \geq i+2$, esueaza, mersul inapoi se opreste la "!". Daca mersul inapoi atinge taietura, "!", atunci ea esueaza si cautarea este procesata pornind de la ultima alegere facuta inainte de cererea ca R sa se unifice cu CL.

Prima utilizare a taieturii se poate produce in momentul in care vrem sa spunem programului logic ca a gasit regula propice pentru o realizare particulara.

De obicei, putem specifica in program ce reguli trebuiesc folosite pentru o anumita realizare dispunind de anumite tipare in regula care vor unifica numai cu realizarea specificata. Sa reluam programul adaug(XS,YS,XSYS):

(1) adaug([],XS,XS).

(2) adaug([AS|BS],CS,[AS|DS]) :- adaug(BS,CS,DS).

Daca vrem sa punem in evidenta listele obtinute prin alaturare, si anume care este prima lista si care a doua, utilizarea primei reguli este ineficienta iar o cerere adaug([],[a1,a2],X), incercata de a doua regula, esueaza aratind ca, daca prima lista este cea vida, , atunci doar prima regula este cea corecta acest fapt eliminind cautarea in cea de-a doua prin "!". Atunci folosirea programului sub forma:

```
adaug([],X,X) :- !.

adaug(ASBS,CS,ASDS) :- adaug(BS,CS,DS)
```

nu afecteaza utilizarea sa corecta ci reduce spatiul de cautare prin utilizarea numai a regulei 1 pentru liste vide. Sa vedem insa efectul taieturii intr-o cerere de forma:

```
adaug(XS,YS,[a,b])?
```

Unificarea cu prima regula din program ne da:

XS = [],YS = [a,b]

dupa care este intilnita taietura "!" ce va influenta alte alegeri pentru solutii prin *backtraking* si, deci, cererea altei solutii va primi raspuns negativ din partea sistemului desi ele exista dar nu pot fi puse in evidenta.

Sa observam, astfel, ca introducerea taieturii pentru cereri de o anumita forma prestabilita nu garanteaza ce se poate intimpla in cazul cererilor de alt tip, strategia taieturii nemaifunctionind pentru o alta interpretare data programului.

Sa definim, astfel, un predicat ce indica proprietatea:

```
proprietar(persoana,NR_masini)
```

intelegind ca persoana este proprietarul unui numar NR. de masini, si urmatorul program.

```
(1)   proprietar(Adi,1):-!.
(2)   proprietar(Cornel,1):-!.
(3)   proprietar(X,0).
```

Taietura este folosita pentru a evita mersul inapoi asupra clauzei (3) daca cererile ce se formuleaza au ca persoana unul din numele {Adi,Cornel}. Deci intrebari ca: proprietar(Adi,X)? sau proprietar(Vasile,0)? au raspunsuri respectiv favorabile: {X=1}, {True}.

O intrebare, insa, de forma: proprietar(Adi,0)? va primi raspunsul {True} (Adevarat) raportind deci succes iar pentru noi o contradictie! Falsul provine din mecanismul de unificare folosit de orice program logic. Iata deci ca taietura "!" folosita in scopul evitarii clauzelor cu numele de persoana instantat in multimea {Adi,Cornel} nu functioneaza corect. Sa incercam atunci o formulare corecta a lor:

```
(1) proprietar(Adi,Nr):-!,Nr=1.
(2) proprietar(Cornel,Nr):-!,Nr=1.
(3) proprietar(X,0).
```

Aceasta formulare raspunde negativ la cererea de proprietar (Adi,0) mersul inapoi gasind in regula (1) Nr=1. Si aceasta varianta nu este completa nefurnizind toate solutiile la cererea proprietar (X,Y).

Sa studiem in paralel doua programe unul continind taietura "!" si altul nu, amindoua realizind ordonarea crescatoare prin adaugare a doua liste intr-una singura. Astfel, din listele XS = [1,5,7] si YS = [2,3,8] se va realiza lista o_adaug([1,3,5,7],[2,3,8],[1,2,3,3,5,7,8]) a elementelor ordonate ce provin din cele doua liste. Programul are ca predicat principal:

o_adaug(ls1,ls2,lordonata)

definit prin programul fara taieturi:

```
(1) o_adaug([X|Xs],[Y|Ys],[X|Zs]):- X<Y,o_adaug(Xs,[Y|Ys],Zs).
(2) o_adaug([X|Xs],[Y|Ys],[X,Y|Zs]):- X=Y,o_adaug(Xs,Ys,Zs).
(3) o_adaug([X|Xs],[Y|Ys],[Y|Zs]):- X>Y,o_adaug([X|Xs],Ys,Zs).
(4) o_adaug(Xs,[],Xs).
(5) o_adaug([],Ys,Ys).
```

Sa observam ca testele X < Y, X = Y, X > Y exclud reciproc folosirea clauzelor 1-3, numai una singura avind loc o data. Asa cum este scris daca, de exemplu, are loc realizarea testului X < Y oricare din clauzele (2) si (3) esueaza actionind mersul inapoi care va realiza din nou pe (1) producind aceleasi solutii, deci avem nevoie de taietura care sa marcheze caracterul mutual exclusiv al regulilor (1)-(3) reducind, astfel, ciclarea. Introducerea taieturii "!" in program genereaza un caracter determinist al predicatului o_adaug in sensul ca pentru orice cerere de realizare numai o singura clauza poate fi folosita cu succes, taietura oprind calculul la aceasta clauza in momentul in care sistemul isi "da seama" ca este singura posibila in unificare:

Program o_adaug cu taieturi:

```
(1) o_adaug([X|Xs],[Y|Ys],[X|Zs]):- X<Y,!,o_adaug(Xs,[Y|Ys],Zs).
(2) o_adaug([X|Xs],[Y|Ys],[X,Y|Zs]):- X=Y,!,o_adaug(Xs,Ys,Zs).
(3) o_adaug([X|Xs],[Y|Ys],[Y|Zs]):- X>Y,!,o_adaug([X|Xs],Ys,Zs).
(4) o_adaug(Xs,[],Xs):-!.
(5) o_adaug([],Ys,Ys):-!.
```

Sa observam ca taietura elimina solutiile redondante in cererea o_adaug([],[],Xs) care, in programul anterior, se realiza daca se mentiona ca Xs sau Ys au cel putin un element.

In general, pentru claritatea programelor, este utila inlocuirea taieturii "!" cu un predicat care sa exprime negatia unei afirmatii, cunoscut ca predicat intern **not**. Am putea defini acest predicat sub forma unui program logic astfel:

```
(1) not(X) :- X,!,fail.
(2) not(X).
```

Deci, daca in incercarea de satisfacere a primei clauze se realizeaza X atunci predicatul esueaza datorita lui fail (ce apare ca un predicat intern cu misiunea de a raporta esec) iar daca X nu este satisfacut mecanismul de *backtraking* incearca clauza a (2)-a realizind functorul not(X). Utilizarea gruparii "!,fail" este cunoscuta drept combinatia "taietura- esec", tehnica larg folosita in scrierea programelor logice, pe care o vom analiza pe larg pe parcursul acestui capitol. Sa observam insa ca utilizarea negatiei not poate fi ineficienta in momentul in care reanaliza unei clauze prin *backtraking* este greoaie daca aceasta este foarte complexa. Sa analizam doua secvente echivalente, una cu utilizarea lui **not** si cealalta cu taietura "!":

```
(1) R :- R1,R2,R3.
    R :- not(R1),R4,R5.
```

si:

```
(2) R :- R1,!,R2,R3.
    R :- R4,R5.
```

In secventa (1) programul poate incerca satisfacerea lui R1 de doua ori, o data daca R1 este adevarata sau, daca nu, prin *backtraking* incearca regula a (2)-a, incearca resatisfacerea lui R1 pentru a vedea daca not(R1) este adevarata.

Acest lucru nu se intimpla in secventa (2) unde, daca R1 este satisfacut si deci se trece de "!", orice incercare de *backtraking* se opreste in taietura "!" pentru prima regula si trece la resatisfacerea celei de-a doua. Utilizarea cea mai comuna a taieturii, asa cum am vazut si in programul o_adaug este punerea lui la sfirsitul unui test clauzal excluzind analiza unui alt test de acelasi tip.

Sa scriem, in acest sens, programul de analiza al maximului dintre doua numere:

(1) maxim(X,Y,X) :- X≥Y,!.

(2) maxim(X,Y,Y) :- X≤Y,!.

Efectul lui "!" este exclusiv in sensul ca realizarea clauzei (1), dupa intilnirea lui "!", nu mai permite mersul inapoi in testul clauzei (2) determinind deci terminarea programului.

Eficienta taieturii "!" apare in momentul in care intr-un program se pot elimina solutii redondante, cum este cazul programului ce urmeaza, care realizeaza sortarea unei multimi reprezentata sub forma de lista.

```
sortez(ls1,ls2):-/*lista ls1 este sortata crescator*/
sortez(ls1,ls2):-adaug(Xs,[X,Y|Ys],ls1),
XY,!,
adaug(Xs,[Y,X|Ys],ls3),
sortez(ls3,ls2).
sortez(ls1,ls1):-ordonat(ls1),!.
ordonat(ls):-/*este program de test a ordonarii crescatoare a unei liste*/
ordonat([X,Y|ls]):- X<Y,ordonat([Y|ls]).
ordonat([X]).
```

Programul functioneaza pe baza comportarii perechilor (X,Y) de termeni care, odata gasiti, nu satisfac testul X≤Y atunci sint inversati, (X,Y)→ (Y,X), dupa care se apeleaza recursiv predicatul sortez pina la testul final in care multimea este cea ordonata.

Testul XY este primul folosit in compararea termenilor primei liste care, daca reuseste trecind de "!" nu mai venim asupra perechii (X,Y) ce este inversata in clauza imediat urmatoare taieturii in (Y,X).

Ultima taietura incheie programul daca lista data ls1 este ordonata.

Taieturile folosite pina acum nu afecteaza semnificatia unui program logic ci numai elimina acele cautari care nu pot duce la solutii noi, eliminarea lor din program neafactind decit timpul de executie necesar realizarii cererilor din program si nu structura lui declarativa. Taieturile de acest tip sint cunoscute ca taieturi verzi pentru a le distinge de cele rosii a caror utilizare poate schimba sensul si desfasurarea programului logic.

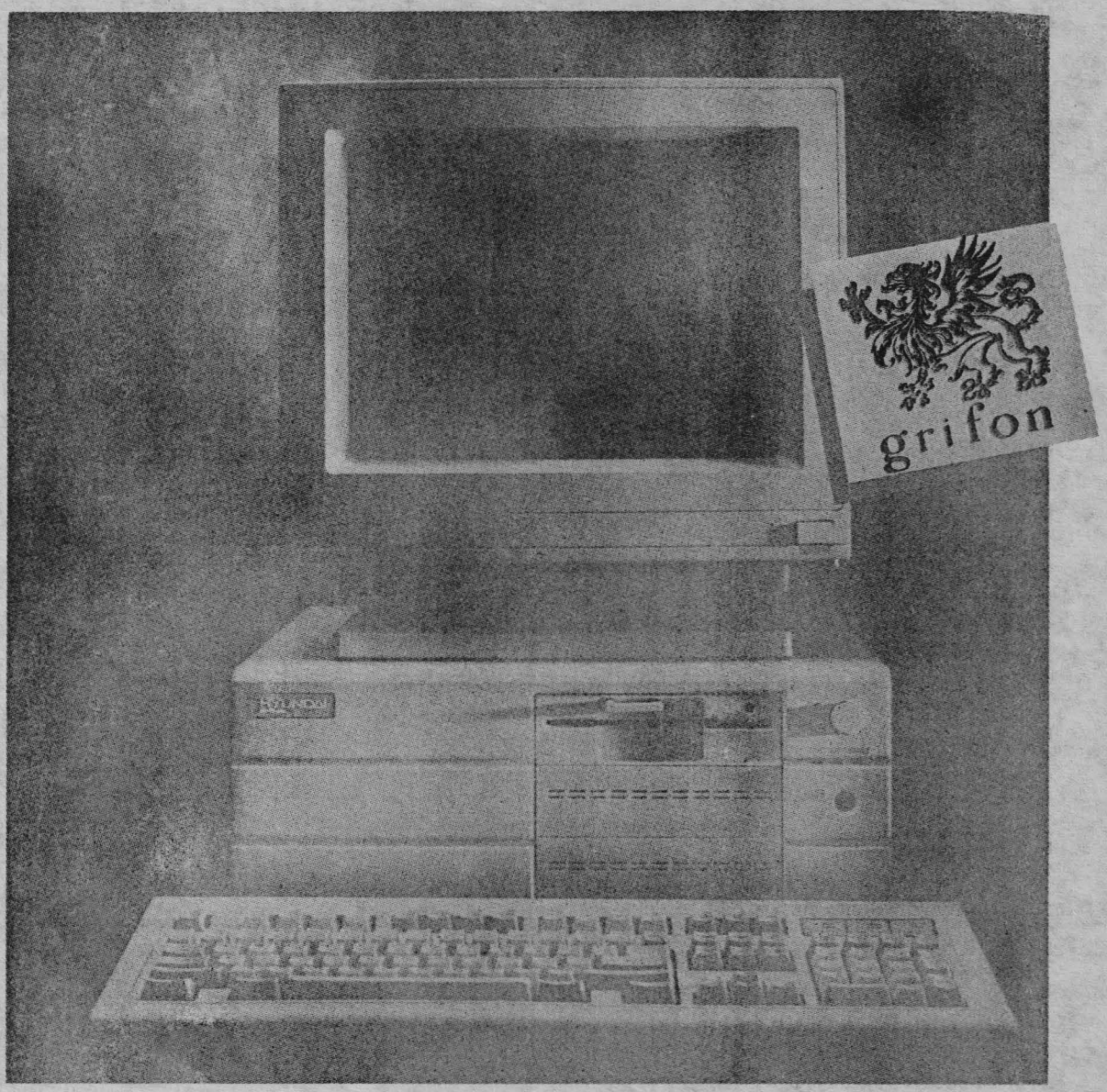
Din punct de vedere al performantei in economisirea spatiului de memorie afectat, un program logic devine ineficient in momentul in care recursivitatea isi gaseste cadru larg si elegant de aplicare in el. Se cere atunci gasirea unei modalitati de transformare a recursivitatii in iteratie, aceasta metoda actionind in spatiu constant indiferent de lungimea ciclurilor de apel formulate.

O astfel de metoda in care taietura intervine esential este cunoscuta ca mecanismul de optimizare finala sau metoda de optimizare a cozilor ce isi propune executia unui program recursiv ca si cum ar fi iterativ.

Sa consdieram o clauza R:-Q1,Q2,...,Qn a carei potentiala optimizare are loc daca actiunea ei se face asupra ultimei realizari, Qn, prin reutilizarea zonei alocate de realizarea parinte lui Qn. Acest lucru se realizeaza adaugind o taietura inaintea ultimei realizari clauzale asupra careia va actiona mecanismul de optimizare:

R:-Q1,Q2,...,Qn-1,!,Qn.

Taietura elimina atit realizarile alternative posibile pentru clauza parinte R cit si alternativele calculatorii ale unor substitutii s pentru (Q1,...,Qn-1). In acest caz este greu de spus daca taietura aplicata este verde sau rosie, functie de partea declarativa a clauzelor.



Limbaje de programare in designul sistemelor de operare si al aplicatiilor

Constantin Mihai

Limbajul C (IV)

Operatori, Expresii conditionale

In acest numar vom continua sa prezentam citiva operatori specifici limbajului C, iar in finalul articolului vom descrie pe scurt forma expresiilor conditionale.

1. Operatorii de incrementare si decrementare

Operatorul de incrementare + + si cel de decrementare – sint operatori unari cu aceeasi prioritate ca si minusul unar, ambii asociindu-se de la dreapta la stinga. Acestia se pot aplica numai variabilelor. Mai mult chiar, ei pot apare in pozitii prefix sau postfix, cu efecte diferite.

Iata citeva exemple de folosire a acestora:

```
++i

sp–

++stack_pointer

–index
```

Este interzisa aplicarea acestor operatori constantelor sau expresiilor:

```
666++      /* constantele  nu pot fi incrementate */

++(a+b/c) /* expresiile nu pot fi incrementate */
```

Instructiunea

`++i;` este echivalenta cu `i = i+1;`

La fel, instructiunea

`i++;` este echivalenta cu `i = i+1`

similar

`–i;` este echivalenta cu `i = i-1;`

`i–;` este echivalenta cu `i = i-1;`

In situatii simple putem considera operatorii + + si – ca fiind notatii simplificate ale incrmentarii si decrementarii variabilelor. Totusi, in unele expresii situatia este mult mai complicata. Ambii operatori + + si – pot fi folositi ca operatori prefix sau postfix, iar efectul poate fi diferit. Expresiile + + i si i + + incrementeaza pe i. Similar, – i si i – decrementeaza pe i. Totusi, cind se foloseste forma + + i intr-o expresie, valoarea este incrementata inainte de evaluarea expresiei. Daca se foloseste i + +, expresia este evaluata folosind valoarea curenta a lui i, dupa care este incrementata valoarea lui i. Similar se comporta – i si i –.

O expresie de forma a + b are o valoare care depinde de valorile lui a si b, dar valorile lui a si b in memorie nu sint afectate in nici un fel de catre expresie. Totusi, expresia + + a are un comportament deosebit. Aceasta are o valoare care depinde de valoarea lui a, si mai mult, expresia schimba valoarea lui a in memorie. Acest fenomen se numeste "efect lateral". In anumite constructii un efect lateral poate produce rezultate nedorite. Sa consideram instructiunea:

```
a = ++b + b; /* cod dependent de masina */
```

Aici + + este prefixat lui b, deci se presupune ca b este incrementat inainte de folosirea lui. Dar b este folosit de doua ori in expresie, iar aceasta schimba valoarea lui b in ambele locuri in instructiune. Calculul valorii expresiei + + b + b este dependent de masina, iar asemenea expresii sint considerate exemple de programare defectuoasa. Este bine ca in asemenea situatii sa scriem un cod care izoleaza efectul operatorului + +, ca de exemplu:

```
++b;
a = b + b;
```

sau

```
a = 2 * (++b);
```

Aceste doua secvente de cod determina strict secventa de evaluare si atribuire.

2. Operatori de lucru la nivel de bit

Limbajul C furnizeaza o serie de operatori pentru manipulari de biti. Acestia nu pot fi aplicati operatorilor de tip float sau double. Iata lista acestora:

& - AND la nivel de bit

| - OR inclusiv la nivel de bit

^ - OR exclusiv la nivel de bit

< < - shift la stinga

> > - shift la dreapta

~ - complement fata de unu (operator unar)

Operatorul & (AND la nivel de bit) este folosit deseori pentru a masca o anumita multime de biti; de exemplu,

```
b = n & 0177;
```

seteaza la zero toti bitii cu exceptia celor mai putin semnificativi 7 biti ai lui n. Operatorul | (OR inclusiv) este folosit pentru a seta la unu o anumita multime de biti:

```
a = a | MASK;
```

seteaza la unu biti din a care sint setati la unu in MASK.

Aveti grija sa distingeti cu mare atentie operatorii la nivel de bit & si | fata de conectorii logici && si ||, care implica evaluarea valorii de adevar de la stinga la dreapta. De exemplu, daca a este 1 si b este 2, atunci x & y este zero in timp ce x && y este unu.

Operatorii de shift-are < < si > > realizeaza deplasarea la stinga, respectiv dreapta, a operandului din stinga lor cu un numar de biti egal cu operandul din dreapta. Astfel x< <2 deplaseaza bitii din componenta lui x cu doua pozitii la stinga, umplind bitii vacanti cu zero; lucru echivalent cu inmultirea lui x cu 4. Shift-area la dreapta a unei cantitati fara semn umple bitii vacanti cu zero. Shift-area la dreapta a unei cantitati unsigned are un comportament dependent de masina, tipul acesteia determinind umplerea cu zero sau cu unu a bitilor ramasi vacanti.

Operatorul unar ~ realizeaza complementarea fata de unu a unei cantitati intregi, adica converteste fiecare bit-1 intr-un bit-0 si vice versa. Acest operator se foloseste in mod uzual in expresii de forma:

```
x & ~077
```

care mascheaza ultimii sase biti ai lui x la zero.

In continuare vom exemplifica lucrul cu acesti operatori printr-o functie C care intoarce ca rezultat mixarea octetilor superiori din doi intregi dati.

```
unsigned int mixer (unsigned int left, unsigned int right) {
    return ((left & ~0x00ff) | (right >> 8));
}
```

3. Operatori de atribuire

O expresie de forma

```
count = count + 2
```

va aduna 2 la vechea valoare a lui count si va atribui rezultatul lui count, iar expresia in ansamblul ei va avea aceasta valoare. Expresia

```
count += 2
```

foloseste operatorul + = pentru a realiza acelasi lucru. Lista de mai jos contine toti operatorii de atribuire:

Operatori de atribuire

```
- - - - - - - - - - - - - - - - - - - - - - - - - - -
= += -= *= /= %= >>= <<= &= ^= |=
```

Toti acesti operatori au aceeasi prioritate si se asociaza de la "dreapta la stinga". Semantic ei pot fi descrisi in felul urmator:

variabila op= expresie

echivalenta cu

variabila = variabila op (expresie)

Folosirea conjugata a acestor operatori da nastere unor constructii complexe. Asemenea constructii confera limbajului nota de concizie care ii este specifica.

Iata citeva exemple:

```
k %= n = 1 + 1/2      echivalenta cu k = (k % (n = (1 + (1/2))))
1 + 4 * (m -= 6) / 5  echivalenta cu 1 + (4 * (m = (m-6)) / 5)
```

4. Expresii conditionale

Secventa de cod

```
if (x > y)
    z = x;
else
    z = y;
```

calculeaza in z valoarea maxima dintre valorile lui x si y. O "expresie conditionala", scrisa cu ajutorul operatorului ternar "?:", furnizeaza un mod mai concis de a rescrie aceasta constructie. In expresia

```
e1 ? e2 : e3
```

expresia e1 este evaluata mai intii. Daca aceasta nu este zero (adevarata) atunci este evaluata expresia e2 si aceasta va deveni valoarea expresiei conditionale. Altfel, va fi evaluata e3 si valoarea ei va fi valoarea intregii expresii. De remarcat, ca doar una dintre expresiile e2 si e3 este evaluata. Acum putem rescrie secventa sub forma:

```
z = (x > y) ? x : y; /* z = max (x, y) */
```

Este de notat faptul ca o expresie conditionala este cu adevarat o expresie si ea poate fi folosita ca oricare alta. Daca e2 si e3 au tipuri diferite, tipul rezultatului este determinat de catre regulile de conversie prezentate in numarul anterior. De exemplu, daca f este de tip float, iar n este de tip int, atunci expresia

```
(n > 0) f : n
```

este de tip double indiferent dâca n este pozitiv sau nu.

Parantezele din jurul primei expresii nu sint necesare, deoarece precedenta operatorului "?:" este foarte scazuta, imediat deasupra atribuirii. Ele sint insa recomandate pentru o mai buna lizibilitate a partii conditionale din expresie.

Folosirea expresiilor conditionale conduce la un cod mai succint. Iata de exemplu felul in care putem scrie o functie care va converti in majuscule elementele unui vector de tip char terminat cu caracterul nul (0x00).

```
to_upper (char vector []) {
    for (i=0; vector [i] != ""; i++)
        vector [i] = (vector[i] > 'a')? vector[i]-'a'+'A':vector[i];
}
```

5. Instructiunea vida

Introducem in continuare notiunea de instructiune vida ";". Dupa cum se va observa, ea este necesara in acele locuri in care este ceruta sintactic prezenta unei instructiuni, fara a fi necesara vre-o actiune din punct de vedere semantic. Vom vedea in articolele viitoare, ca aceasta se dovedeste folositoare in instructiunile care controleaza fluxul executiei, cum sint instructiunile if sau for.

Bibliografie:

- The C Programming Language, Kernighan Brian, Ritchie Dennis;
- A Book on C, Kelley Al, Pohl Ira, 1986;
- Standard C, Plauger P.J., Brodie Jim, 1989.

PROGRAMAREA CONSOLEI SISTEM (TASTATURII) IN PROLOG (III)

si...sfaturi generale pentru utilizatorii de PC

Adrian Negru

III. Utilizarea tastaturii prin predicate interne Prolog

III.1. Functionarea tastaturii

Tastatura unui microsistem PC contine un procesor Intel sensibil la orice apasare a unei taste avind ca efect depozitarea unui cod numeric de scanare (*scan code*) in portul placii de interfata periferica 8255 situata pe placa sistemului. Acest cod este un numar pe un octet ai carui biti formeaza o combinatie numerica specifica fiecarei taste care poseda, in fapt, doua tipuri de coduri de scanare, unul generator, atunci cind tasta este apasata si unul intrerupator, cind tasta este eliberata. Relatia intre cele doua coduri este: cod apasare + 128 = cod intrerupere sau, in cazul microprocesoarelor 80286 si 80386, codul de intrerupere are o lungime de doi octeti din care primul este codul de apasare, cel de-al doilea continind valoarea F0H. La orice actiune a unei taste cipul 8255 trimite spre microprocesorul tastaturii un semnal de intrerupere. Cind acest cod de scanare ajunge la portul A este apelata intreruperea INT 9. Aceasta intrerupere comanda procesorului central executarea unui program intern care transfera codul de scanare in cod de caracter, exceptie facind tastele speciale (cele programabile) sau tastele SHIFT, ALT, CTRL al caror cod este "tinut minte" in memorie si verificat inaintea analizei oricarui cod de caracter; ceea ce permite recunoasterea caracterelor de rind (a, b, c, d,...) sau a majusculelor (A, B, C,...) ori a comenzilor de sistem. Pasul urmator il constituie depozitarea in memorie a caracterelor in ordinea tastarii lor intr-un bufer de tastatura ce poate acomoda pina la cincisprezece caractere.

Codurile de caracter nu sint altceva decit codurile ASCII standard (pina la 128) si ASCII extins (129-255). Codurile 1-32 sint folosite drept comanda de procesor neparticipind drept caractere tiparibile ci drept comenzi pentru periferice in timp ce codurile 33-128 corespund, in fapt, caracterelor alfabetice si numerice situate pe tastatura (ASCII 7 este avertizorul acustic).

Codul ASCII extins, situat in gama 129-255, are fiecare numar pe doi octeti primul octet fiind tot timpul 0, conventie ce permite ca un program sa detecteze daca un anumit caracter apartine setului ASCII standard sau extins. Aceste coduri nu reprezinta taste fizice existente pe tastatura ci caractere speciale afisate in memoria grafica a sistemului (litere speciale, simboluri matematice, simboluri grafice, alfabetul grec etc.).

Exista combinatii de chei care nu produc coduri de scanare precum <CTRL-ALT-DEL> (initializare sistem), <CTRL-Break> (<Ctrl- C>),

<PrtScr> (hard copy al ecranului).

Majoritatea compilatoarelor sau a diverselor pachete software furnizeaza o mare varietate de rutine interne pentru controlul si gestiunea tastaturii ceea ce usureaza utilizatorului folosirea acesteia raminind insa deschise si posibilitati de utilizare a ei mai rafinate cum ar fi reprogramarea setului de caractere, programarea tastelor etc.

III.2. Predicate interne Prolog pentru tastatura

Compilatoarele de Prolog pun la dispozitia utilizatorului predicate pentru utilizarea tastaturii orientate in procesarea citirii de caractere si a tiparirii lor pe ecran care, cu mici eventuale diferente sintactice, se pot defini in continuare.

1) **comline** (Parametri) - tipul argumentului este **string**. Predicatul are ca efect citirea parametrilor unei linii de comanda folosita la invocarea numelui unui program executabil;

2) **inkey** (Caracter) - tipul argumentului este **char**. Predicatul are ca efect citirea unui caracter de la tastatura, daca aceasta exista. Actiunea ei esueaza daca nu este tastat nimic;

3) **keypressed**. Predicatul are ca efect testarea tastaturii in vederea apasarii unei chei. Actiunea lui are succes in momentul in care se apasa orice caracter, altfel esueaza;

4) **readchar** (Caracter) - tipul argumentului este **char**. Predicatul are ca efect citirea unui caracter de la tastatura sau de la un periferic de intrare redirectat prin **readdevice** (periferic);

5) **readdevice** (Nume simbolic) - tipul argumentului este **symbol**. Predicatul are ca efect redirectarea suportului de intrare spre fisierul cu numele simbolic Nume-simbolic. Astfel, **readdevice** (Screen) determina citirea, in continuare, a caracterelor de pe ecran in loc de tastatura, **readdevice** (Keyboard) repune tastatura ca suport de intrare iar o comanda de tipul **readdevice** (Fis) determina citirea din fisierul Fis. Predicatul mai poate avea actiune de a redenumi suportul de intrare curent cu numele Nume-simbolic. Astfel, daca in mod curent se citeste de la tastatura acesta, in urma apelului de predicat, poate purta numele Nume- simbolic pe parcursul desfasurarii programului;

6) **readint** (Nr_intreg) - tipul parametrului este **integer**. Predicatul are ca efect citirea unui intreg de pe suportul de intrare terminat cu caracter retur - ASCII 13 (< CR >);

7) **readln** (Sir) - tipul parametrului este **string**. Predicatul are ca efect citirea unui sir de caractere de pe suportul de intrare pina la intilnirea unui caracter de retur (< CR >);

8) **readreal** (Nr_real) - tipul parametrului este **real**. Predicatul are ca efect citirea unui numar real de pe suportul de intrare terminat cu caracterul de retur (< CR >);

9) **readterm** (Domeniu, Termen) - tipul parametrilor este (< name > , < variabila >). Predicatul are ca efect citirea obiectului scris cu predicatul write. Termen este considerat real pina la o declarare in Domeniu. Permite accesul **faptelor** in fisiere;

10) **write** (arg1, arg2,..., arg n). Predicatul are ca efect scrierea argumentelor argi pe suportul de iesire (ecran) sau orice alt suport redirector cu **writedevice**. Poate avea oricite argumente (maxim 50);

11) **writedevice** (Nume-simbolic) - tipul argumentului este **symbol**. Predicatul are acelasi efect ca cel al lui **readdevice** insa asupra suportului de iesire **writedevice** (printer) redirecteaza orice actiune ulterioara a lui **write** spre imprimanta;

12) **writef** (Format, Arg1, Arg2,...). Predicatul are ca efect tiparirea cu format. Format are forma:

% - m.p.

unde "-" inseamna aliniere la stinga (cea dreapta este implicita), "m" specifica marimea cimpului de tiparire iar "p" precizia unui numar in virgula flotanta sau numarul maxim de caractere de tiparit dintr-un sir. Cimpul "p" mai poate contine literele "f" (numere reale in notatie zecimala), "e" (numere reale in notatie exponentiala), "g" (alege cel mai scurt format).

Sa vedem cum putem realiza aceste predicate in citeva programe uzuale.

Sa imbogatim biblioteca de rutine de scriere Prolog prin citeva programe utilitare. Un prim astfel program ar fi elaborarea unor rutine de citire si scriere a listelor.

```
domains
lista = integer*
predicates
cit_lista (lista)
goal
write("Scrieti o lista de intregi"),cit_lista(L),
wrjte("Lista este" L)
clauses
cit_lista([H|T]) :- readint(H),!,cit_lista(T).
cit_lista().
```

Programul introduce un predicat util pentru citirea unei liste. In cazul nostru, lista este de tip intreg si foloseste predicatul readint dar se poate citi si o lista de caractere sau de numere reale folosind readchar sau readreal.

Procedura este clasica, lista delimitindu-se prin capul ei si coada ei, apelul la citire facindu-se recursiv, pas cu pas. Existenta taieturii dupa citirea capului listei ingheata o eventuala recitire a listei si apeleaza recursiv citirea cozii determinind corectitudinea citirii listei element cu element.

Un alt program ar fi cel ce scrie la terminal o lista.

```
domains
lista = integer*
predicates
cit_lista(lista)
scrie_lista(lista)
goal
write("Scrieti o lista de intregi"),cit_lista(lista),
scrie_lista(lista)
clauses
cit_lista([H|T]) :- definit anterior
scrie_lista([H|T]) :- write (H,";"),!,scrie_lista(T)
scrie_lista([]) :- nl,!.
```

Elementele listei sint scrise ordonat despartite prin simbolul ; urmind ca la sfirsitul scrierii, dupa o comanda de retur de car (nl = <CR> + <NL> = ODH + OAH), sa se termine programul.

Un al treilea program va elabora un predicat de citire a unei structuri oarecare declarate functional de la terminal.

```
domains
persoana = f(nume,virsta,telno,adresa)
virsta = integer
telno,nume,adresa = string
predicates
readpersoana(persoana)
```

```
test
goal
run.
clauses
readpersoana(p(Nume,Virsta,Telno,Adresa)):-
write("Nume?"),readln(Nume),
write("Adresa?"),readint(Adresa),
write("Virsta?"),readint(Virsta),
write("Telefon no?"),readln(Telno).
test:-
readpersoana(P),nl,write(P),nl,nl,
write("Este functia persoana corecta (d/n)"),
readchar(Ch),Ch='d'.
test:-
nl,nl,write("Mai incercati o data"),nl,nl,test.
```

Programul declara o structura functionala *persoana* care este un predicat definit printr-o functie p(„„) cu patru parametri ce se citesc de la terminal.

Prima clauza a testului cere introducerea datelor de la terminal si ne intreaba daca ceea ce s-a obtinut este corect, altfel executa o a doua clauza test si anume reluarea primei clauze test care ne determina sa introducem din nou datele.

Programul urmator executa un test al tastaturii scriind pe ecran tastele apasate, atit cele cu valori tiparibile cit si cele cu caracter de control sau comanda.

```
domains
tasta = cr;esc;break;tab;btab;bdel;ins;end;home;
tasta(integer);sus;jos;stinga;dreapta;char(CAR);alta
predicates
cit_lista(tasta)
tasta_cod(tasta,char,integer)
tasta_cod2(tasta,integer)
goal
cit_tasta("Test de tastatura. Apasati o tasta!"),
cit_tasta(Tasta),
cit_tasta("Tasta a fost apasata").
clauses
cit_tasta(Tasta):-
readchar(T),char_int(T,Val),tasta_cod(Tasta,T,Val).
tasta_cod(Tasta,_,0):-
readchar(T),char_int(T,Val),tasta_cod2(Tasta,Val),!.
tasta_cod(break,_,3):-!.    tasta_cod(bdel,_,8):-!.
tasta_cod(tab,_,10):-!.     tasta_cod(cr,_,13):-!.
tasta_cod(esc,_,27):-!.     tasta_cod(char(T),T,_).
tasta_cod2(btab,15):-!.     tasta_cod2(home,71):-!.
tasta_cod2(sus,72):-!.      tasta_cod2(stinga,75):-!.
tasta_cod2(dreapta,77):-!. tasta_cod2(end,79):-!.
tasta_cod2(jos,80):-!.      tasta_cod2(ins,82):-!.
tasta_cod2(del,83):-!.
tasta_cod2(ftasta(N),V):-V>58, V<70, N=V-58,!.
tasta_cod2(alta,_).
```

Predicatul cit_tasta citeste o cheie de la tastatura apoi realizeaza conversia codului ASCII al caracterului citit in valoare intreaga recunoscuta in tabela ASCII a caracterelor. Urmatorul predicat este tasta_cod ce contine cheia tastata (Tasta), codul sau de caracter (T) si valoarea lui intreaga (Val). Distinctia care

determina definirea predicatului tasta_cod2 este ca acele taste reprezentate prin doua coduri ASCII produc intotdeauna, ca prim octet (caracter), valoarea 0, daca ele nu sint insotite de taste de comanda <ALT>, <CTRL>, <SHIFT>. Astfel, tasta break are configuratia pe doi octeti din care primul octet este 0 ea producind o intrerupere de sistem in combinatia <CRTL> + break (valoare 3 ASCII). Aceasta tasta, deci, nu produce un caracter tiparibil si de aceea este reprezentata printr-un nume simbolic standardizat break corespunzator codului ei ce se obtine prin citirea celui de al doilea octet (caracter) daca primul octet a fost 0 sau tocmai apelul clauzei tasta_cod (Tasta,_,0).

Un alt program acceseaza parametrii unui predicat pentru a regasi intr-o baza de date un nume sau un numar de telefon simulind o agenda.

```
predicates
agenda(symbol,symbol)
goal
write("Tastati numele al carui telefon vreti sa-l aflati"),
readln(Nume),
agenda(Nume,Tel),
write("Telefonul",Tel,"este al lui",Nume).
clauses
agenda("Adrian","660436").
agenda("Ioana","655060").
agenda("Irina","608321").
```

Sa observam ca baza de date este specificata de baza de fapte asertata sub comanda clauses, dupa tastarea numelui Nume predicatul agenda (Nume, Tel) incearca unificarea lui Tel cu parametrul corespunzator din baza care are acelasi Nume.

Mai putem simula si un test, daca o anumita tasta apasata este caracter tiparibil sau nu:

```
predicates
caracter(char)
goal
write("Apasati o tasta"),
readln(T),
caracter(T),
write("Tasta",T,"este caracter").
clauses
caracter(T) :- t≤'Z', 'a'≥T.
caracter(T) :- T≤'Z','A'≥T.
```

Sa incercam, in continuare, elaborarea unui editor primitiv care sa permita utilizatorului controlul de la tastatura al ecranului, scrierea si modificarea unor siruri de caractere de la tastatura.

O prima parte a programului va fi initiata de rutina care sa ne permita controlul cursorului pe ecran si posibilitatea de miscare libera a lui. In corpul acestui program vom face apel la predicatul de sistem cursor (Linie, Coloana) pe care-l vom analiza in capitolul destinat sistemului video si care are ca efect mutarea cursorului ecran la linia Linie si coloana Coloana. Predicatul care va initia miscarea libera a cursorului pe ecran este:

```
mut_cursor(Linie,Coloana,Directie)
```

unde miscarea Directie este sensul de mutare al cursorului aflat in pozitia curenta (Linie, Coloana).

```
domains
linie,coloana,deplasare = integer
directie = sus(deplasare);jos(deplasare);
```

```
            stinga(deplasare);dreapta(deplasare);stationare.
predicates
mut_cursor(linie,coloana,directie)
clauses
mut_cursor(L,C,sus(Deplasare)):-
cursor(L,C),L1 = L-Deplasare,cursor(L1,C).
/*actiunea predicatului produce deplasarea in sus cu un numar de Deplasare linii din pozitia
curenta a cursorului (Linie, Coloana)*/
mut_cursor(L,C,jos(Deplasare)):-
cursor(L,C),L1 = L + Deplasare,cursor(L1,C).
/*Program cursor*/
/*
```

Cererea de realizare se face la promptul Goal prin invocarile posibile:

1) mut_cursor(R,C,sus(10)) - care, dupa mutarea cursorului din pozitia curenta, intoarce in R si C valorile pe care acesta le-a avut initial (astfel, daca cursorul se afla pe linia 10, coloana 5 predicatul intoarce R = 10, C = 5, si-l muta apoi pe linia 0, coloana 5;

2) mut_cursor(3,3,jos(5)) - muta cursorul din linia 3, coloana 3 in linia 8, coloana 3.

```
/*

domains
linie,coloana,pas = integer
mutare = sus(pas);jos(pas);
         stinga(pas);dreapta(pas);nu
predicates
mut_cursor(linie,coloana,mutare)
clauses
mut_cursor(R,C,sus(Pas)):-
cursor(R,C),
R1 = R-Pas,cursor(R1,C).
mut_cursor(R,C,jos(Pas)):-
cursor(R,C),
R1 = R + Pas,cursor(R1,C).
mut_cursor(R,C,stinga(Pas)):-
cursor(R,C),
C1 = C-Pas,cursor(R,C1).
mut_cursor(R,C,dreapta(Pas)):-
cursor(R,C),
C1 = C + Pas,cursor(R,C1).
mut_cursor(R,C,nu):-
cursor(R,C).
```

/*Predicatul cursor (R,C) este un predicat intern al sistemului si apelul lui are ca efect mutarea cursorului in pozitia R (rind), C (coloana) (vezi capitolul destinat graficii in Prolog din numerele viitoare)*/

```
mut_cursor(L,C,stinga(Deplasare)):-
cursor(L,C),C1 = C-Deplasare,cursor(L,C1).
mut_cursor(L,C,dreapta(Deplasare)):-
cursor(L,C),C1 = C + Deplasare,cursor(L,C1).
mut_cursor(L,C,stationare):-
cursor(L,C).
```

Actiunea predicatului mut_cursor, in celelalte cazuri, este usor de dedus.

Sa exemplificam ,in continuare, un program in care vom face apel si la predicatul cit_tasta rezultat dintr-un program anterior, si unde vom putea edita un text cu ajutorul predicatului editez putind umbla prin textul editat cu tastele de miscare ale cursorului stinga si dreapta, putind abandona in orice moment programul tastind cheia functionala F1 si incheia sesiunea de editare cu cheia F2. Orice actiune de la tastatura, nerecunoscuta de editor, se va semnala printr-o avertizare sonora.

```
Include "program.pro"        /*includem predicatul cit_cod*/
domains
linie,coloana,lungime = integer
cimp = f(linie,coloana,lungime)
pozitie = p(linie,coloana)
predicates
editez(cimp,pozitie,tasta)
goal
Linie = 5, Coloana = 5, Lungime = 50,
makewindow(1,20,2, ,0,0,25,80),
write("Editor de text. Folositi tastele cu sageti pentru deplasare"),nl.

/*makewindow este o instructiune grafica pe care o vom analiza in capitolul urmator ea realizind o
fereastra grafica pe ecran*/

/*field_attr este un predicat care determina lungimea si atributele unui cimp de scriere intr-o fereastra
grafica situat pe o anumita linie si coloana*/

write("Tastati F2 pentru sfirsitul editarii si F1 pentru abandon"),
cursor(Linie,Coloana).
field_attr(Linie,Coloana,Lungime,60),
editez(f(Linie,Coloana,Lungime),p(Linie,Coloana),home)),nl,nl.

/*cheia home are valoarea 71 definita in cit_tasta*/

field_str(Linie,Coloana,Lungime,Continut)
write("Textul editat:",Continut).

/*predicatul field_str permite scrierea sirului de caractere Continut pe cimpul din fereastra grafica situat
pe pozitia (Linie, Coloana) si cu lungimea Lungime. Sirul Continut nu trebuie sa depaseasca 50 de
caractere altfel acestea nu mai apar*/

clauses
editez(_,_,ftasta(1)) :- !,fail.   /*abandon pentru F1*/
editez(_,_,ftasta(2)) :- !         /*sfirsit editare pentru F2*/
editez(f(Linie,Coloana,Lungime),p(L,C),char(car):-
editez(L,C,Car),C1 = C + 1,C1 < Coloana + Lungime,
cursor(Linie,C1),cit_tasta(Tasta),
editez(f(Linie,Coloana,Lungime),p(L,C1),Tasta).

/*Corpul principal al predicatului editez permite inserarea caracter cu caracter intr-un cimp a textului
dorit avansind cursorul dupa fiecare tastare*/

editez(f(Linie,Coloana,Lungime),p(L,C),dreapta):-
C1 = C + 1,C < Coloana + Lungime,cursor(L,C1),cit_tasta(Tasta),
editez(f(Linie,Coloana,Lungime),p(L,C1),Tasta).
editez(f(Linie,Coloana,Lungime),p(L,C),stinga):-
C1 = C-1,C > Coloana,cursor(L,C1),cit_tasta(Tasta),
editez(f(Linie,Coloana,Lungime),p(L,C1),Tasta).
```

```
/*Cele doua clauze permit mutarea cursorului in cadrul limitelor textului de editat fara scoaterea
caracterului*/

editez(Cimp,Pozitie,_):-
beep,cit_tasta(Tasta),
editez(Cimp,Pozitie,Tasta).
```

III.3. Programarea tastaturii la nivel de sistem

In aplicatii complexe predicatele interne furnizate de Prolog se pot dovedi insuficiente pentru exploatarea dorita de utilizator. Se impune, astfel, o cunoastere intima a proceselor ce se pot produce la nivelul tastaturii si redactarea unor rutine Prolog, eventual impreuna cu rutine, in limbaj de asamblare care, combinate, sa duca la bun sfirsit cerintele programatorului.

III.3.1. Exploatarea buferului de tastatura si tehnicilor de citire

Este la indemina programatorului sa goleasca buferul tastaturii inaintea unei cereri de citire. Acest bufer, care poate contine pina la cincisprezece caractere, este construit ca o stiva circulara cunoscuta sub numele de bufer FIFO (primul intrat, primul iesit). Ca orice bufer el ocupa o zona continua de memorie, doi pointeri tinind evidenta virfului si capatului sirului de caractere existent curent in bufer orice nou caracter fiind depozitat in locatia de memorie imediat urmatoare adresei indicate de capatul de bufer, acesta, dupa insertie, fiind ajustata corespunzator.

In momentul in care buferul s-a umplut, sistemul trimite un sunet (*beep*) sistemului. Cei doi pointeri ai acestui bufer se afla la adrese imediat urmatoare daca in el se afla un caracter iar valorile lor sint egale in momentul cind acesta este gol.

Pointerii virf si coada se afla la adresele 0040:001A si, respectiv, 0040:001C conditia de golire a buferului fiind egalizarea valorilor existente la aceste adrese.

Pentru orice programator este posibila inserarea unui sir de caractere in acest bufer urmat de un retur de car < CR > care, daca desemneaza o comanda, poate fi executata de sistemul MS-DOS imediat dupa terminarea unui program; permitind executia unei functii fără o interventie a utilizatorului sub actiunea sistemului de operare.

Cu ajutorul functiilor mem_word si ptr_word putem citi si schimba valori ale pointerilor de bufer. Secventa:

```
mem_word($0040,$1A,Continut)
ptr_word(Continut,$0040,$1C)
```

citeste valoarea pointerului virf la adresa 40:1A, in locatia Continut, pe care apoi o depune la adresa pointerului **capat**.

De asemenea, functia C a intreruperii 21H executa aceeasi functie de golire a buferului inaintea tastarii vreunui caracter, apelul Prolog echivalent secventei anterioare fiind:

```
AX = $0C01
bios($21,reg(AX,_,_,_,_,_,_,_),_).
```

Se poate intimpla ca in timpul executiei unui program sa se tasteze un sir de caractere de la tastatura care este automat pastrat in buferul acesteia daca nu este utilizabil de program.

Functia B a intreruperii 21H intoarce valoarea 0 in AL daca acest bufer este gol si FFH daca contine unul sau mai multe caractere.

```
clauses
test:-
AX = $0B00,
bios($21,reg(AX,_,_,_,_,_,_,_),reg(VAL,_,_,_,_,_,_)).
egal(0,VAL),write("Buferul este gol").
test:-
AX = $0B00,
bios($21,reg(AX,_,_,_,_,_,_,_),reg(VAL,_,_,_,_,_,_)).
egal($FF,Val),write("Buferul nu este gol").
```

Locatia Val va contine valoarea lui AL dupa executarea intreruperii 21H.

Functia 1 a intreruperii Bios 16H executa aceeasi functie cu deosebirea ca AX va contine si caracterul tastat.

Daca avem de-a face cu un cod ASCII standard acesta se afla in AL, altfel AL contine 0 pentru codul ASCII extins si AH este registrul cu numarul codului.

```
AX = $0100
bios($21,reg(AX,_,_,_,_,_,_,_),reg(Val,_,_,_,_,_,_)),
Val1 = Val,
bitand(Val1,256,Val2), /*pun pe 0 octetul superior al lui Val declarat intreg*/
egal(Val2,0),write("Caracterul este un cod extins"),
bitright(Val,8,Cod) /*pun codul din AH in primul octet din Cod*/
ascii (Cod)          /*conversie ASCII a codului*/
char_int(Cod,Numar),write("Codul numeric al caracterului tastat este",Numar).
```

Caracterele citite de la tastatura sint, in mod normal, scrise pe ecran pentru a se valida ce a fost tastat. De multe ori, aceasta scriere, cunoscuta si ca ecou, nu este dorita (ca, de exemplu, o comanda intr-un meniu de editare) sau se cere o verificare a lor inaintea afisarii.

O functie utilizata in astfel de situatii este predicatul inkey care nu asteapta o tastare daca nu este pus intr-o bucla ce cicleaza pina la sosirea unui caracter. De exemplu, poate pastra o imagine pe ecran pina la tastarea unui caracter, ea intorcind mereu caracterul NULL desemnat, de multe ori, printr-un sir de lungime zero (""). O astfel de ciclare se poate reprezenta prin:

```
clauza:- program,
inkey(car),egal(car,""),clauza.
```

Secventa are ca efect executia predicatului program atita timp cit nu se tasteaza nici un caracter.

Sistemul MS-DOS pune la dispozitie rutine pentru interceptarea sirurilor de caractere functie pe care o poate pune la dispozitie predicatul readln care considera sirul terminat in momentul intilnirii unui retur <CR> . Functia OAH a intreruperii 21H permite citirea sirurilor cu o lungime de pina la 254 de caractere scriindu-le pe ecran. DS:DX contine adresa de memorie unde este plasat sirul, primul octet al acestei locatii continind numarul de octeti alocati sirului. Dupa tastare, al doilea octet ne da numarul caracterelor real tastate, sirul efectiv incepind cu cel de-al treilea octet. Sfirsitul sirului este semnalat de <CR> care nu este insa numarat drept caracter actual tastat. Cu aceasta pregatire sa elaboram o rutina, in limbaj de asamblare, care poate simula predicatul readln atunci cind acesta nu satisface cererile utilizatorilor sau nu exista. Vom scrie un sir de 253 de caractere efective, alti trei octeti fiind pastrati pentru <CR> si pentru octetii de stare ai sirului.

```
SIR DB 256 DUP (?) ; rezerv loc pentru 253 caractere, doi octeti
                   ; pentru descriptori si unul pentru <CR> .
. CODE
mov AX,0DATA       ; incarc in DS adresa segmentului de date
```

```
mov DS,AX
lea DX,SIR          ; incarc in DX adresa de unde incepe SIR
mov BX,DX    ; pastrez BX ca un contoar al adreselor din sir
mov AL,254   ; incarc si spatiu pentru <CR>
mov [BX],AL  ; in registrul BX pun numarul de caractere
mov AH,0AH   ; functie de citire a sirului
INT 21H
```

Foarte important pentru programator este testarea starilor in care se afla la un moment dat cheile functionale. Astfel, doi octeti de memorie aflati la adresele 0040:0017 si 0040:0018 pastreaza starile tastelor speciale astfel:

Adresa	Bit nr.	Tasta	Semnificatie pentru bit = 1
0040:0017	7	Insert (Ins)	Modul de insertie este activ.
	6	CapsLock	Modul de scriere cu majuscule este activ.
	5	NumLock	Modul NumLock este activ.
	4	ScrollLock	Modul ScrollLock este activ.
	3	Alt-Shift	Tasta este apasata.
	2	Ctrl-Shift	Tasta este apasata.
	1	Shift stinga	Tasta este apasata.
	0	Shift dreapta	Tasta este apasata.
0040:0018			
	7	Insert	Tasta este apasata.
	6	CapsLock	Tasta este apasata.
	5	NumLock	Tasta este apasata.
	4	ScrollLock	Tasta este apasata.
	3	Ctrl-NumLock	Tasta este apasata.

Sa exemplificam printr-o rutina Prolog, care sa determine scrierea cu majuscule pe ecran, inhibind celelalte functii, adica sa puna pe 1 bitul 6 la adresa 0040:0017 sau sa efectueze un AND logic intre valoarea octetului de la aceasta adresa si numarul binar 01000000B a carui valoare zecimala este 64.

```
mem_byte($40,$17,Continut),
bitand(Continut,64,Stare_noua),
ptr_byte($40,$17,Stare_noua).
```

Putem simula si BASIC aceasta rutina prin secventa:

```
10 DEFSEG = &H40
20 Continut = PEEK(&H17)
30 Stare_noua = Continut AND 64
40 POKE(&H17,Stare_noua).
```

III.3.2. Programarea si exploatarea tastelor speciale

Tastele **ENTER, ESC** (Escape), **Backspace, Tab** sint singurele chei functionale care genereaza cod ASCII pe un octet. Exista o echivalenta intre ele si combinatii de tip **CTRL + caracter**, astfel:

ENTER (valoare ASCII 13) = CTRL+M

ESC (valoare ASCII 27) = CTRL+[

Backspace (valoare ASCII 8) = CTRL+H

Tab (valoare ASCII 9) = CTRL+I

Functia readchar poate interpreta aceste chei in timp ce predicatul **inkey** nu le recunoaste dat fiind faptul ca actiunea lor este orientata pe ecran.

Combinatiile cheilor Shift produc, in general, cod extins. Sint doar doua cazuri in care ele produc cod ASCII standard: cind se actioneaza pentru a scrie cu majuscule si in combinatiile Ctrl + A pina la Ctrl + Z unde produc coduri in gama 1-26.

Cu exceptia tastei <INS> celelalte chei speciale (CapsLock, ScrollLock, NumLock) nu produc nici un caracter in buferul de tastatura ci numai schimbari de stare in octetii 0040:0018, stari ce sint verificate de sistem inaintea acceptarii oricarui caracter de la tastatura. Tasta <Ins> produce caracterul 0;82 (doi octeti) in buferul consolei; astfel, programatorul poate simula orice cheie speciala daca cunoaste exact actiunea si recunoasterea ei.

Cheile programabile 1, ..., 10, 12 sau 14, in functie de tipul tastaturii, notate F1, F2,..., Fn (n cu valori in {10,12,14}) produce coduri diferite, functie de combinatiile cu tastele Shift, Ctrl, Alt, pe o lungime de doi octeti, primul fiind ASCII 0 iar al doilea un numar in tabelul:

Taste	Coduri
F1-F10	59-68
Shift + (F1-F10)	84-93
Ctrl + (F1-F10)	94-103
Alt + (F1-F10)	104-113

Aplicatiile uzuale cu aceste chei constau in reprogramarea lor astfel incit sa produca diverse comenzi sau actiuni.

Utilitarele de reprogramare a acestor chei puse la dispozitie de sistemul DOS sint cunoscute ca secvente escape.

Un sir care incepe cu caracterul ESC este trimis spre ecran dar, datorita codului ESC el nu ajunge niciodata pe display ci reasigneaza o tasta cu o valoare noua (ambele valori fiind mentionate in sir). Forma generala a acestor siruri este urmatoarea: incep cu caracterul ESC, apoi , [apoi codul numeric al tastei ce urmeaza a fi schimbata, semnul ";", apoi codul numeric nou ce se asigneaza tastei vechi si, in sfirsit, caracterul "p". Codurile extinse sint scrise impreuna cu cei doi octeti (de exemplu, 0;82 pentru tasta <INS>).

Este important si esential ca in configuratia sistemului DOS sa fie fisierul ANSI.SYS altfel secventele escape sint ignorate. In reasignarea acestor chei putem folosi functia 9 a intreruperii 21H care trimite un sir pe ecran. DS:DX contine adresa primului caracter din sir care trebuie sa se termine cu caracterul $ (24H). Ideea algoritmului este urmatoarea: se scrie sirul dorit dupa care in predicatul ptr_dword se citeste adresa sa de implantare DS:DX apoi se apeleaza functia 9 a lui INT 21H.

```
domains
```

```
Sir = string
clauses
Sir = "27,'[0;59;13p$",
ptr_word(Sir,DS,DX,)    /*intoarce in DS:DX adresa sirului Sir*/
AX = $0900,
bios($21,reg(AX,_,_,DX,_,_,DS,_),_).
```

Rutina asigneaza tasta F1 sa aiba aceeasi actiune cu ENTER (< CR >) astfel ca orice apasare ulterioara a lui F1 va produce un retur < CR >.

Dar se pot reprograma tastele astfel incit ele sa execute o comanda de tip DOS. Astfel, sa scriem o rutina care sa-i atribuie tastei F2 comanda DOS "Dir" iar tastei F3 comanda "TYPE ADI.DAT" care sa listeze fisierul ADI.DAT pe ecran. Ambele comenzi sint urmate de < CR > pentru a intra in executie la actiunea tastelor F2, F3.

```
domains
Sir = string
clauses
SIR = "27,'[0;60;"dir";13p$",
AX = $0900,
ptr_word(Sir,DS,DX),
bios($21,reg(AX,_,_,DX,_,_,DS,_),_).
```

si

```
ptr_word("27,'[0;61;"type ADI.DAT;13p$",DS,DX),
AX = $0900,
bios($21,reg(AX,_,_,DX,_,_,DS,_),_).
```

Exercitii

1) Sa se scrie o rutina care sa simuleze o intrerupere utilizator si care sa fie executabila prin actiunea unei taste.

(Indicatie: Se poate redirecta INT 5 produsa la apasarea tastei < PrtScr > (tipareste pe imprimanta continutul ecranului) printr- o intrerupere utilizator ce se instaleaza cu ajutorul functiei 25H sau a lui INT 21H iar DX contine deplasamentul in segmentul de date de unde incepe rutina utilizator de intrerupere.)

2) Sa se scrie un program care sa reprogrameze intreruperea de tastatura astfel ca sa se poata scrie cu caractere create de utilizator.

Microcalculatoarele cu arhitectura de multiprocesare rivalizeaza calculatoarele mari

Gheorghe Curelet-Balan

Aparitia microcalculatoarelor bazate pe microprocesoarele Intel 80486 si Motorola 68040 situeaza microcalculatorul, din punctul de vedere al puterii de calcul, la acelasi nivel cu minicalculatorul. Cursa in diminuarea raportului pret/performanta (adica a pretului pe care utilizatorul trebuie sa-l plateasca pentru un nivel de performanta) nu se opreste insa aici.

Lansarea, in anul trecut, de catre una dintre cele mai prestigioase firme constructoare de microcalculatoare, Compaq, a sistemului Systempro, a deschis o noua nisa in piata microcalculatoarelor. Este vorba despre microcalculatoarele cu arhitectura de multiprocesare. Aceste sisteme pot oferi practic o putere de pina la 200 MIPS (milioane de instructiuni pe secunda), ceea ce le situeaza, din punctul de vedere al raportului pret/performanta, cu mult peste facilitatile calculatoarelor mari (puterea de calcul fiind apropiata de cea a supercalculatorului Cray).

Multiprocesare inseamna cresterea puterii de calcul prin folosirea mai multor unitati centrale in construirea unui sistem de calcul. Dintre marii producatori de microcalculatoare care au lansat astfel de masini in ultimii 12 luni amintim:

- firma Compaq cu sistemul Systempro cu doua unitati centrale Intel;
- firma Zenith cu sistemul Z1000 care suporta o unitate centrala Intel 80486 si una Intel 860;
- firma Mitac cu sistemul Series 500 bazat pe mai multe unitati centrale Intel 80386.
- firma Aix cu familia de sisteme System 90 (modelele 25, 45, 85 pot avea 2, 4, 8 procesoare tip Motorola 68020 (la 25 MHz) ca unitati centrale, 1 procesor Motorola 68030 pentru procesarea I/E si un procesor Motorola 68020 pentru arbitrarea bus-ului sistem).

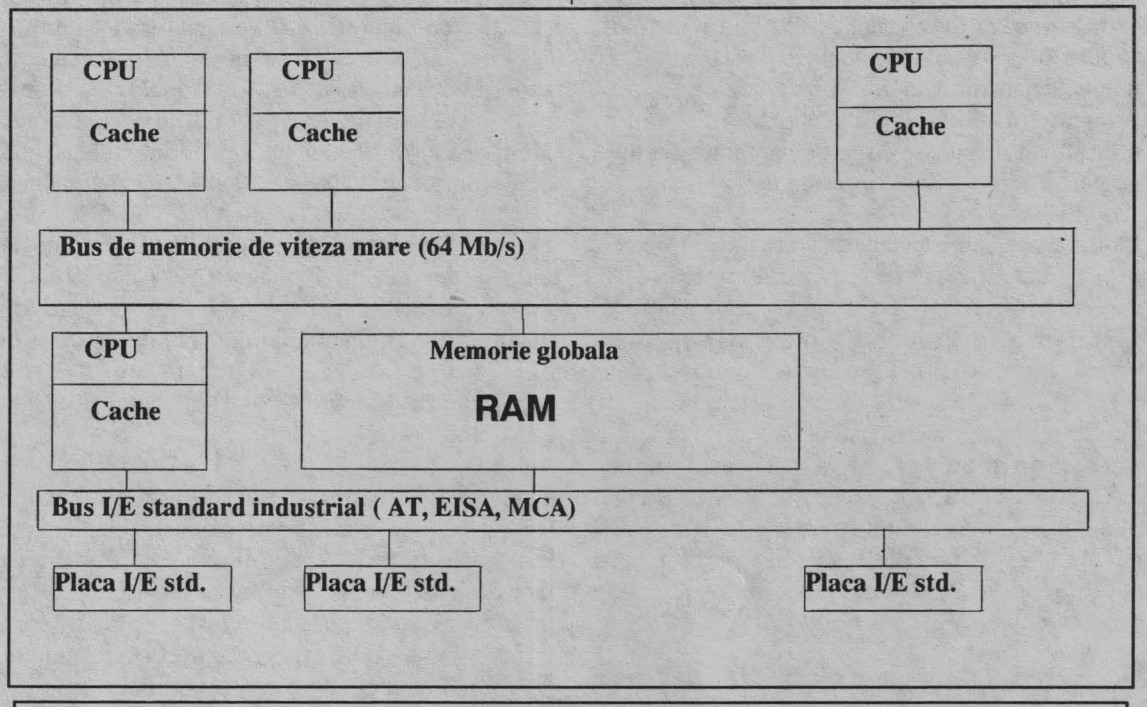
Un exemplu elocvent relativ la performantele unor astfel de sisteme il reprezinta declaratia lui Rod Canion, presedintele firmei Compaq, care in noiembrie 1989 la lansarea lui Systempro afirma ca intr-o configuratie cu 60 utilizatori, Systempro a fost de 6 ori mai rapid decit superminicalculatorul VAX al firmei DEC si de 3 ori mai rapid decit HP9000 al firmei Hewllet-Packard chiar daca e vindut cu 135000 $ mai putin decit VAX si 68000 $ mai putin decit HP9000. Deasemeni, modelul 25 al sistemului System 90 al firmei AIX poate avea o memorie interna de pina la 160 megaocteti suportind 3 unitati de disk sau banda, si 4 procesoare de comunicatie. In final amintim ca noile sisteme de calcul concureaza ultimele modele de superminicalculatoare AS/400 si System 3X ale IBM-ului.

Este justificata deci, temerea constructorilor traditionali de minicalculatoare si calculatoare mari relativ la accentuarea revolutiei microcalculatoarelor prin lansarea sistemelor cu multiprocesare. Se precipita astfel, pe zi ce trece o ruptura in evolutia raportului pret/performanta. "Atmosfera incinsa" generata de aparitia noilor sisteme este simtita probabil mai mult decit oricine de numarul 1 mondial, IBM. Totusi, IBM nu este insensibil, prezentind in public la inceputul anului 1989 in laboratoarele sale din Yorktown Heights un sistem cu arhitectura de multiprocesare ce foloseste 8 procesoare RISC (Reduced Instruction Set Computers). Sistemul foloseste procesoarele RISC, ROMP (cele folosite in sistemele RISC, IBM RT) rulind sub sistemul de operare de tip UNIX, Mach.

Impunerea microcalculatoarelor cu multiprocesare este conditionata de doua elemente, si anume: micsorarea regiei sistemului in operatiile de intrare-iesire (I/E) si existenta platformei software care sa exploateze facilitatile specifice arhitecturii de multiprogramare. Evolutiile curente in domeniile hard/soft ne fac sa credem ca noile masini se vor impune cit de curind. Solutii privind micsorarea regiei

operatiilor I/E au fost date deja in lumea microcalculatoarelor. Un exemplu elocvent il reprezinta masina NEXT a firmei Next Inc., in care arhitecturile de acces direct la memorie (DMA) au fost proiectate prin preluarea metodelor de procesare pe canal de banda larga din lumea calculatoarelor mari. In plus, aparitia bus-urilor pe 32 biti (EISA, MCA, VME, si noul bus al firmei SUN, S-bus) va duce la inlaturarea diferentelor intre regiile operatiilor I/E dintre calculatoarele mari si microcalculatoarele cu multiprocesare.

Microcalculatoarele proiectate in arhitectura de multiprocesare pot fi clasificate dupa doua criterii. Primul criteriu este gradul de cuplare al procesoarelor. In schema de cuplare slaba fiecare procesor isi are propria memorie, comunicarea facindu-se pe bus-ul folosit de periferice. Deoarece o astfel de schema nu asigura o viteza de procesare comparabila cu a doua, ea nu este foarte raspindita. A doua schema de cuplare este cea tare. In aceasta schema, toate procesoarele partajeaza o singura memorie RAM, conectarea acestora la ea realizindu-se printr-un bus de memorie de viteza mare (64 Mb/s). Toate



Schema de multiprocesare a firmei Corollary

In ceea ce priveste software-ul, se observa o aliniere a sa la tendintele de evolutie a noilor tehnologii hard. Majoritatea producatorilor de software de baza si-au ajustat software-ul pentru a suporta arhitecturile cu multiprocesare. Este cazul software-ului de retele locale LAN Manager 2.0 care pe un sistem Systempro cu solicitari de I/E puternice sub OS/2 ar putea fi configurat astfel: pe un procesor ruleaza aplicatiile iar pe celalalt soft-ul de retea si cel de gestiune fisiere HPFS (High Performance File System). Nucleul produsului de retea Netware 386 al firmei Novell a fost proiectat pentru a suporta multiprocesarea. Deasemeni, IBM-ul declara ca a prevazut posibilitatea implementarii multiprocesarii in noua versiune (a 3-a) a sistemului de operare AIX (sistem de tip UNIX).

sistemele exemplificate mai sus respecta o astfel de schema.

Al doilea criteriu de clasificare este relativ la relatia dintre procesoare. Deosebim astfel sisteme proiectate dupa schema simetrica (in care toate procesoarele sint egale intre ele) si sisteme asimetrice (in care un procesor este 'master' asigurind procese celorlaltor procesoare, arbitrind bus-ul de memorie si controlind activitatea de I/E).

Solutia de proiectare hard/soft cea mai raspindita care a fost acceptata de majoritatea producatorilor, este cea a firmei Corollary (vezi figura - schema de multiprocesare aproape simetrica a firmei Corollary).

Deoarece un procesor este folosit pentru gestiunea transferurilor de I/E intre bus-ul de memorie si periferice, schema Corollary nu este perfect simetrica, ci aproape simetrica. Dupa cum se observa fiecare procesor are propria sa memorie cache, care reflecta o parte din datele din memoria principala.

Schema Corollary s-a impus prin eleganta cu care a rezolvat cele doua probleme hard/soft esentiale in multiprocesare: coerenta memoriilor cache si echilibrarea incarcarii procesoarelor. Primul aspect se refera la pastrarea coerentei datelor din memoriile cache in momentul in care un procesor prin activitatea sa modifica datele din memoria principala. Adica, o modificare a unei date din memoria principala trebuie actualizata in toate memoriile cache care reflecta acea data. Al doilea aspect, relativ la echilibrarea incarcarii procesoarelor se refera la asigurarea distribuirii corespunzatoare a proceselor la procesoare.

Una din problemele ramasa inca nerezolvata in domeniul multiprocesarii este toleranta la erori, in sensul ca daca unul din procesoare cade, sa nu cada tot sistemul.

Totusi, performantele deosebite ale noilor sisteme au sensibilizat in primul rind producatorii de SGBD-uri (Oracle Corp., Informix, Ingres Corp.). Ei afirma ca performantele soft- ului lor va creste cu o rata proportionala cu numarul de procesoare folosite intr-un sistem cu multiprocesare.

In concluzie putem afirma ca, desi volumul imens de soft DOS existent poate fi compatibilizat pe sistemele cu multiprocesare (prin asignarea fiecarui program la cite un procesor distinct) aceasta nu justifica inca optiunea pentru un sistem cu multiprocesare atit timp cit nu va fi conceput volumul corespunzator de software care sa exploateze la maxim facilitatile de multiprocesare.

INITIERE IN CONCEPTELE COMUNICATIEI DE DATE (III)

Octavian Paiu

1. Standardul de retele ISO/OSI

Organizatia internationala pentru standardizare (ISO), cu sediul la Geneva, este o agentie care realizeaza recomandari (standarde) pentru intreaga lume stiintifica si tehnica. Deoarece, la ora actuala, exista o multime de tipuri diferite de calculatoare care pot fi implicate in comunicatii de date, ISO a realizat un model denumit OSI (Open Systems Interconnection) ce poate fi utilizat pentru simplificarea schimbului de date catre aceste calculatoare.

Proiectarea hardware-ului si software-ului in concordanta cu recomandarile standardului OSI permite utilizatorilor realizarea si adaptarea la cerintele proprii ale unui sistem de comunicatii de date prin selectarea produselor care le ofera facilitatile cerute. ISO a divizat cerintele generale ale unei comunicatii intre calculatoare, in modul OSI, in sapte subprocese denumite straturi. De asemenea, s-au descris specificatiile functionale pentru fiecare strat. Actiunile care trebuiesc realizate in fiecare strat sint precis definite dar metodele folosite pentru implementarea acestor actiuni sint lasate la dispozitia fiecarui proiectant care se incadreaza in standardul OSI. Evident, orice metoda de realizare tehnica a unui strat este acceptabila daca este utilizata la ambele capete ale legaturii de date.

In continuare vom prezenta o descriere sumara a specificatorilor straturilor modelului ISO/OSI incepind cu stratul cel mai apropiat de utilizator, si anume stratul aplicatie.

a) Stratul 7 - Stratul aplicatie

Acest strat (nivel) este "raspunzator" de realizarea unor functiuni apropiate de utilizatorul unui calculator, ca de exemplu: introducere de la distanta a lucrarilor (RJE - Remote Job Entry), transfer de fisiere, activitatile pentru baze de date distribuite. Acest nivel OSI manipuleaza schimbul de informatii. Continutul si realizarea exacta, practica a acestui strat este, bineinteles, lasata la latitudinea fiecarui realizator. Pentru ca programele utilizatorilor din diferite calculatoare sa poata comunica acestea ar trebui sa includa definitia precisa a mesajelor (si raspunsurilor la mesaje) vehiculate intre aceste programe. De asemenea, la acest nivel trebuiesc luate decizii privind "transparenta" retelei, decizii care trebuie sa raspunda, cel putin, la urmatoarele intrebari:

- Care ar fi efortul de programare pe care trebuie sa-l depuna un utilizator pentru a folosi reteaua?
- Care ar fi cele mai bune metode de utilizare cu eficienta maxima a retelei?

Daca in retea sint incluse si baze de date atunci trebuiesc incluse, la acest nivel, si protocoale standardizate existente. Pe scurt, stratul aplicatiei este "responsabil" cu stabilirea unei interfete de utilizator.

b) Stratul 6 - Stratul de prezentare

Acest nivel trebuie sa realizeze "prezentarea" datelor catre sistemul receptor in sintaxa si cu formatul acceptabil de acel sistem. Aici se implementeaza servicii cum ar fi editarea, formatarea si afisarea datelor. Daca reteaua permite comprimari de date acestea se realizeaza la acest nivel, eventual impreuna cu cifrarea/descifrarea mesajelor. De asemenea, nivelul 6 rezolva discrepantele dintre terminale incompatibile. Aceste compatibilitati includ, printe altele: lungimea liniei, dimensiunile ecranului, tehnicile de adresare a cuvintului, marcarea "liniilor" in fisiere, selectia modului "derulare" sau "pagina", selectia setului (fontului) de caractere etc.

c) Stratul 5 - Stratul sesiune

Stratul 5 reprezinta interfata efectiva a utilizatorului cu reteaua. Este responsabil cu actiunile de initiere, gestiune si terminare a sesiunii de comunicatie. Aici se realizeaza, de asemenea, sarcini de control ale retelei, conform

cu definitiile stabilite in stratul aplicatie; stratul de prezentare preia informatiile de la acest nivel si le transforma corespunzator cerintelor stratului de aplicatie.

d) Stratul 4 - Stratul transport

Stratul de transport controleaza transmisia cap-la-cap adica procesul de "miscare" a datelor de la un punct la altul al retelei. De regula, datele sint segmentate, la acest nivel, in "pachete" de dimensiuni convenabile (tinind cont de o serie de factori: dimensiunea memoriei, capacitatea discurilor etc.). Acest strat reprezinta, de fapt, un coordonator al fluxului de date. De fapt, la acest nivel nu se transmit efectiv date in retea ci se coordoneaza pregatirea datelor pentru transmisie: "ambalarea" in pachete, decizia de folosire a circuitelor fizice sau virtuale, controlul integritatii si unicitatii circulatiei de mesaje.

e) Stratul 3 - Stratul retea

Acest strat gestioneaza dirijarea datelor in retea. Aici se determina adresa procesului receptor din retea. Daca se folosesc circuite virtuale atunci acestea se creeaza la acest nivel. Fiecare circuit virtual reprezinta o legatura logica permanenta intre doua puncte (noduri) ale retelei.

f) Stratul 2 - Stratul legaturii de date

La acest nivel se realizeaza, in principal, urmatoarele actiuni: detectia (si, eventual, recuperarea) erorilor, transmisia si receptia blocurilor sau/si a "semnalelor" de confirmare/ informare. Pe scurt, acest strat implementeaza protocolul de comunicatii la nivelul legaturii de date.

g) Stratul 1 - Stratul fizic

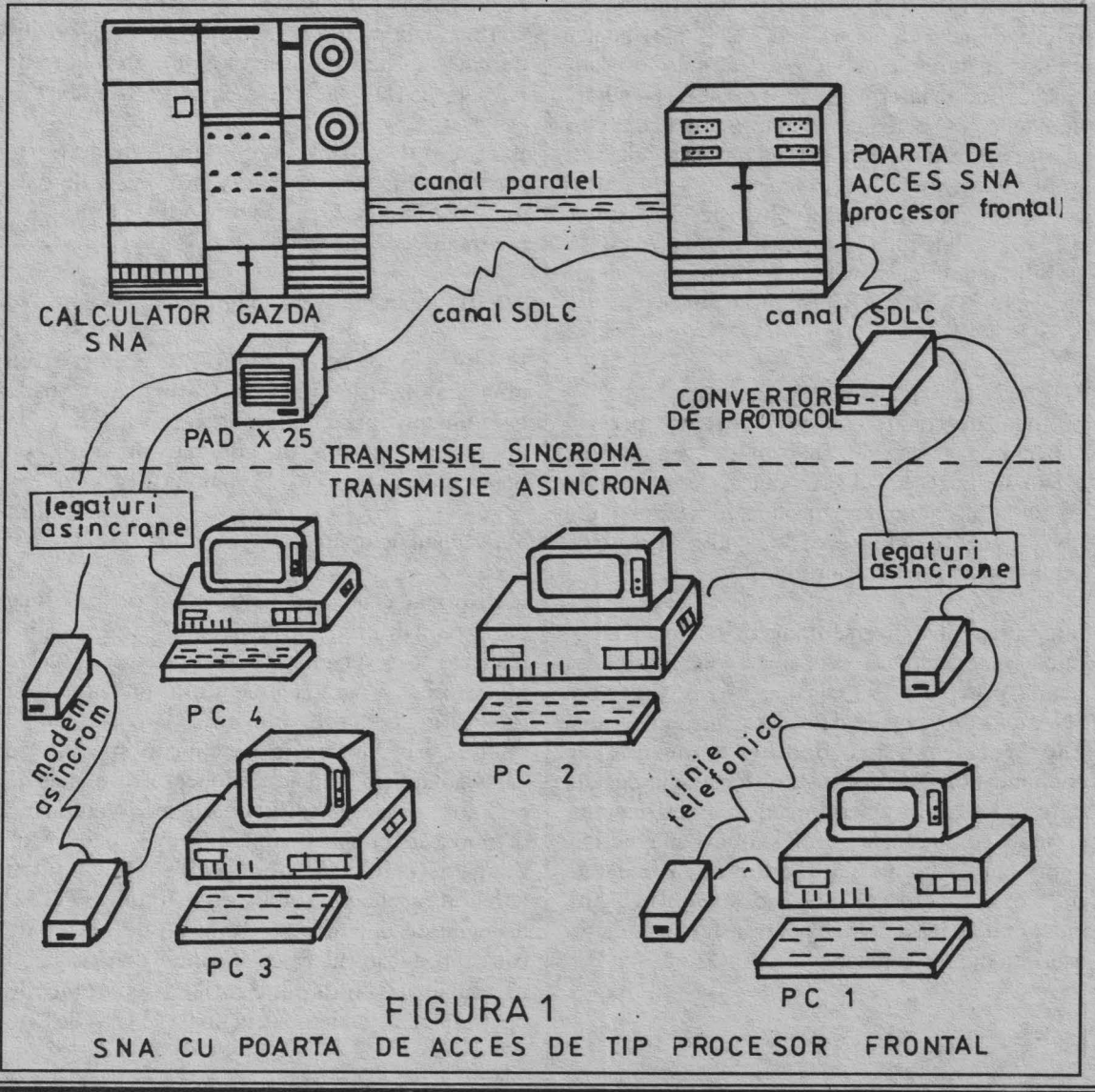


FIGURA 1
SNA CU POARTA DE ACCES DE TIP PROCESOR FRONTAL

Stratul fizic, cel mai de jos strat al modelului ISO/OSI, mentine, activeaza si dezactiveaza conexiunile fizice (electrice!) in retea. Datele binare (1 si 0) sint transformate astfel incit sa fie acceptabile pentru canalul de comunicatie (modulare/demodulare, transmisie in mediul de comunicatii etc.).

2. SNA si alte retele

Majoritatea marilor producatori de calculatoare si-au dezvoltat arhitecturi (modele) de retea care sa "suporte" propriile echipamente. Dar, pentru a extinde posibilitatile retelei, multe din aceste arhitecturi ale marilor producatori prevad mijloacele necesare pentru includerea unor calculatoare ale altor producatori. De exemplu, firma DEC (Digital Equipment Corporation) ofera solutii de arhitectura de retea denumita DECNET. Un alt protocol (X.25) a fost propus de catre CCITT (Consultative Commitee on International Telephone and Telegraph) pentru utilizarea, in retele publice, de transmisii de date. X.25 defineste interfata pentru retele cu comutare de pachete. O retea cu comutare este acea retea in care conexiunea intre doua puncte se realizeaza prin "apelul" (formarea numarului corespondentului) (exemplul clasic il constituie reteaua telefonica publica). Odata legatura stabilita, datele se vehiculeaza sub forma de "pachete". De regula pachetele au dimensiuni fixe (256, 512, 1024 bytes).

SNA (System Network Architecture) este protocol dezvoltat de firma IBM care permite conectarea in retea a mai multor generatii de calculatoare IBM. Ca si modelele de arhitectura de retea ale altor mari producatori, arhitectura SNA permite integrarea si a altor tipuri de echipamente (ale altor producatori).

Integrarea intre diverse tipuri de echipamente si retele asemanatoare (sau nu) se realizeaza, de regula, prin porti de acces (gateway). Arhitectura SNA foloseste portile de acces pentru integrarea unor retele cu protocoluri asincrone de comunicatie. SNA asigura transferul sigur de date intre utilizatorii retelei precum si utilizarea unor resurse comune ale retelei: benzi magnetice, imprimante, discuri s.a. Fiecare retea integrata intr-o arhitectura SNA poate functiona independent iar datele se pot transfera cu relativa usurinta dintr-o retea in alta.

In figura 1 este prezentata utilizarea portilor de acces ale retelei SNA pentru integrarea unor comunicatii asincrone.

Microcalculatorul PC comunica, pe linie asincrona, de exemplu telefonica, cu un "convertor" de protocol care este conectat prin linie sincrona cu un procesor frontal. Protocolul de comunicatie pe linia sincrona de SDLC (Synchronins Data Link Control) - protocolul arhitecturii SNA. "Poarta de acces" catre SNA este reprezentata de procesorul frontal. Transferul de date catre poarta de acces si calculatoarele din SNA se face pe canale (de regula, paralele) de foarte mare viteza.

Microcalculatorul PC 2 foloseste (spre deosebire de PC 1) o legatura asincrona delicata care nu necesita modemuri.

Microcalculatoarele PC 3 si PC 4 folosesc, pentru "intrarea" in poarta de acces SNA, un dispozitiv definit de X.25 ca asamblor/ dezasamblor de pachete (PAD - Packet Assembler Dissambler).

Protocolul SDLC "negociaza" prioritatile cererilor utilizatorilor pentru transmisia de date dintre PAD-ul X.25 sau convertorul de protocol si poarta de acces in arhitectura SNA.

3. Software implicat in comunicatiile de date

In principiu, pentru comunicatia de date, la un microcalculator sint implicate urmatoarele componente principale software: sistemul de operare, programul de comunicatie, driver-ul pentru interfata seriala de comunicatie.

a) Sistemul de operare

Este format, in general, dintr-o serie de programe care coordoneaza folosirea, pe cit posibil eficienta, a resurselor unui calculator de catre utilizatori si programele acestora. Functiunile si facilitatile concrete oferite de un sistem de operare sint variate in functie, in special, de calculatoarele pe care le gestioneaza. Sistemele de operare folosite pe calculatoarle medii/mari sint, in mod evident, mai complexe si mai "puternice" decit sistemele de operare folosite pentru calculatoarele personal/profesionale (PC'S). Sistemul de operare este furnizat, de obicei, de catre producatorul calculatorului. Dar exista si producatori "independenti" de sisteme de operare. Pentru microcalculatoare (PC'S) de 8/16

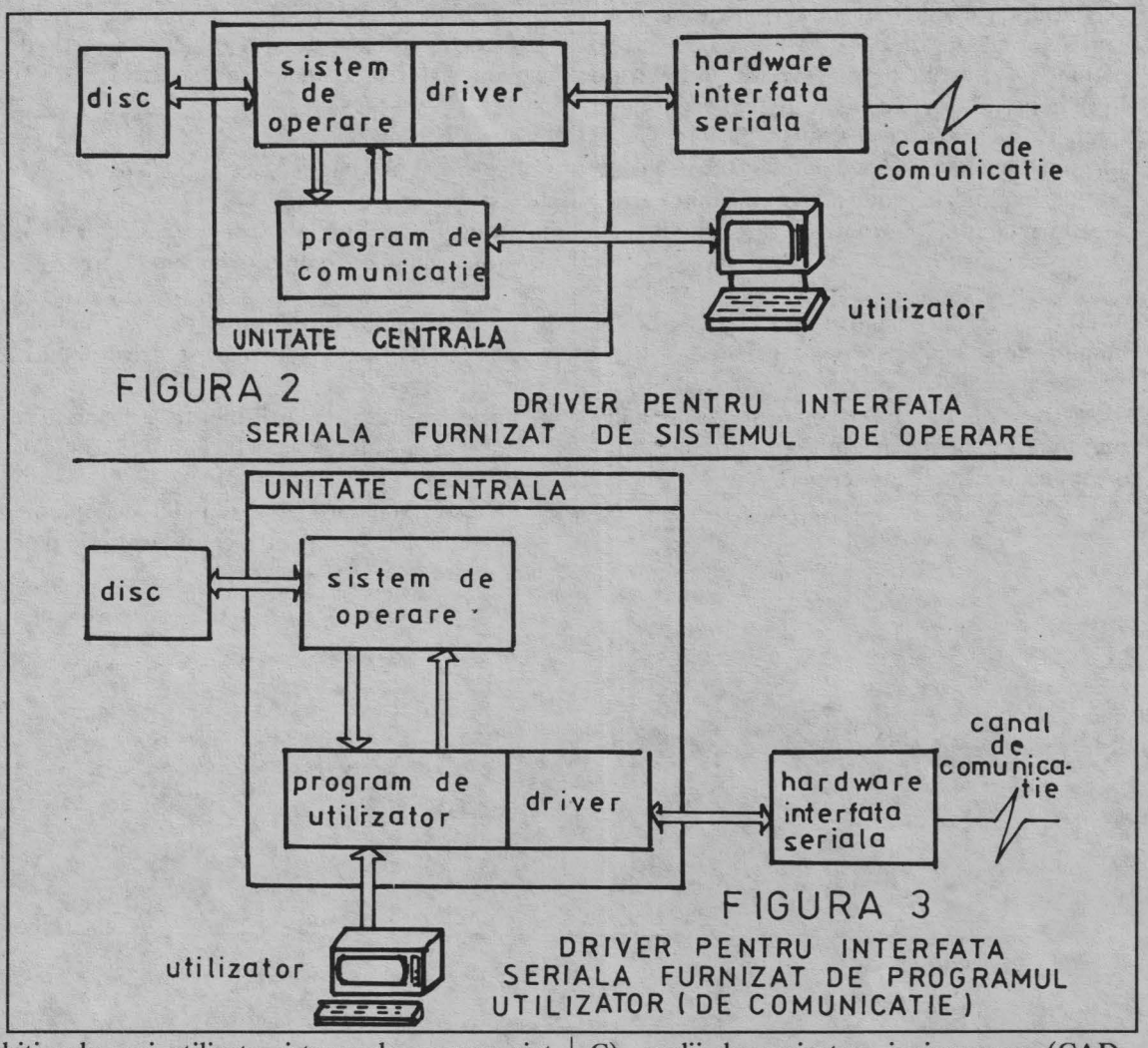


FIGURA 2
DRIVER PENTRU INTERFATA SERIALA FURNIZAT DE SISTEMUL DE OPERARE

FIGURA 3
DRIVER PENTRU INTERFATA SERIALA FURNIZAT DE PROGRAMUL UTILIZATOR (DE COMUNICATIE)

biti cele mai utilizate sisteme de operare sint CP/M, MS-DOS si UNIX.

Una dintre functiile cele mai importante ale unui sistem de operare pe microcalculatoare personal/profesionale o reprezinta gestiunea memoriei externe pe discuri magnetice. De exemplu, programul utilizatorului are acces la fisiere numai prin folosirea numelui de fisier, localizarea efectiva a acestuia pe discuri fiind realizata de catre sistemul de operare.

b) Programul de comunicatie

La ora actuala, pe microcalculatoare exista o multitudine de programe de "utilizator" specializate pe domenii: "foi" de calcul (spreadsheets) (ex.: LOTUS), editoare de texte la nivel de linie sau "cuvinte" (ex.: WordStar, WordPerfect. Word etc.), gestiune de baze de date (ex.: dBase, Fox, SQL etc), medii "integrate" pentru dezvoltarea de alte programe (ex.: Turbo C), medii de proiectare inginereasca (CAD Computer Aided Design) etc., etc.

In aceasta gama, un loc oarecum aparte il ocupa programele suport de comunicatie care devin din ce in ce mai "populare" pe masura ce microcalculatoarele se conecteaza intre ele sau cu calculatoare mini sau mainframe. Aceste programe ofera facilitati importante pentru utilizator: modul de lucru terminal, suport pentru controlul modemurilor "inteligente", transfer de fisiere cu controlul erorilor, limbaje de control (ex.: BLAST, Kermitt, CommPlus, 3+Comm etc.). Unele programe de comunicatie pot fi implementate nu numai pe microcalculatoare ci si pe minicalculatoare sau calculatoare medii/mari (mainframe) realizind, astfel, un transfer "complet" de date cap-la-cap.

Printre facilitatile uzuale oferite de astfel de programe se pot enumera, de exemplu: automatizarea comenzilor prin intermediul

"scenariilor", emulatoare de terminale ale unor mini sau mainframe (VT100, VT220 etc), gestiunea automata a fisierelor de inregistrare (log) a actiunilor desfasurate pe canalul de comunicatie. Toate aceste tipuri de facilitati fac ca utilizarea "pachetelor" de comunicatie actuale sa fie mult simplificate reducindu-se, in multe cazuri, la apasarea pe citeva taste are claviaturii microcalculatorului.

c) Driver-ul pentru interfata seriala de comunicatie

Acest software de "baza" controleaza detaliile comunicatiei prin "gestiunea" hardware-ului corespunzator (interfata seriala).

Driver-ul pentru interfata seriala poate fi furnizat fie de catre sistemul de operare (figura 2), fie de catre pachetul de comunicatie respectiv (figura 3).

Driver-ele furnizate de catre sistemul de operare sint cele mai utilizate in special pentru sistemele de calcul mainframe. Dar "pachetul" de comunicatie trebuie sa furnizeze driver-ul in situatii de genul celor aratate in continuare:

- unele sisteme de operare (ca, de exemplu, Apple-DOS) nu includ acest driver pentru interfata seriala;
- unele drivere "standard" oferite de sistemul de operare nu ofera toate facilitatile cerute de programul de comunicatie;
- driver-ul standard al sistemului de operare poate fi ineficient pentru comunicatii de date de mare viteza.

1990 - anul retelelor locale

Anul 1989 a fost marcat de citeva evenimente semnificative in domeniul retelelor locale de calculatoare personale (Local Area Networks (LAN)), ceea ce a dus la o penetrare mai puternica a acestora la utilizatori. Sondajele de opinie facute de marile publicatii de informatica precum si organizatiile de studiu al pietei releva faptul ca numarul utilizatorilor de calculatoare personale care opteaza pentru achizitionarea de retele locale este in continua crestere. Pentru exemplificare prezentam mai jos tabelul cu dinamica vinzarilor de echipamente LAN, intocmit de International Data Corporation.

82	83	84	85	86	87	88	89
58	103	209.6	310.4	1100	1800	4800	6300

Sumele sint specificate in milioane $.

Iata care sint evenimentele majore ale anului 1989 care au influentat pozitiv domeniul retelelor locale.

Probabil cel mai mare eveniment a venit din partea IBM-ului, prin recunoasterea standardului Ethernet. Dupa mai multe incercari de a impune propriul standard, Token Ring si de a descuraja tehnologiile competitoare, IBM hotaraste sa sprijine Ethernet-ul. Astfel IBM a incercat sa impuna reteaua sa locala Token Ring de 2 megabiti/s apoi de 4 si de 16, insa restrictiile hardware la care erau supusi utilizatorii n-au asigurat succesul acestei alternative. Decizia de a adopta standardul Ethernet a fost luata ca un compromis necesar in a asigura succesul masinilor PS/2, IBM-ul sustinind ca important este ca masinile sale sa fie conectate intre ele si-nu prin ce se face aceasta. Prin licentierea in 1989 a NDIS (Network Driver Interface Specification) al firmei 3Com si adaugarea suportului Ethernet soft-ului sau de retea Lan Server, IBM a recunoscut standardul Ethernet. In acest fel el a spulberat starea de neliniste a uriasei sale clientele, preocupata de prelungita sa indecizie in a da o solutie eleganta in problema retelelor locale. Aceasta atitudine dovedeste schimbarea politicii arogante de lider a IBM- ului si recunoasterea tacita a faptului ca piata trebuie vazuta asa cum este si ca trebuie sa puna accentul pe mediile multivinzator. Ca un corolar al acestei atitudini mentionam lansarea versiunii DOS a LAN Server-ului. In acest fel clientii retelelor IBM nu mai sint fortati sa foloseasca sistemul de operare OS/2. In plus, tot in 1989, IBM a decis sa renunte la concurenta sa cu Microsoft, hotarind sa uneasca LAN Server-ul sau cu soft-ul de retea locala al firmei Microsoft, LAN Manager.

Un alt eveniment semnificativ il reprezinta aparitia in 1989 a produselor care implementeaza standardul Ethernet pentru linii telefonice. Desi initial solutia nu parea promitatoare, performantele deosebite de 10 megabiti/s precum si avantajele date de usurinta exploatarii si administrarii unei astfel de retele duc la o continua sensibilizare a utilizatorilor. Initial solutia a fost propusa pentru a fi folosita in zonele aglomerate unde costul instalarii cablurilor coaxiale este destul de ridicat. Avind in vedere insa ca o astfel de retea poate fi configurata, depanata si controlata mai usor (toate legaturile retelei pot fi centralizate intr-un singur loc, unde poate fi instalat si server-ul de retea) ne putem explica de ce acest tip de retele este din ce in ce mai raspindit. Totusi succesul acestei alternative va depinde de modalitatea in care producatorii vor rezolva cele doua mari probleme ale domeniului, si anume : a) IEEE n-a luat inca o decizie finala asupra standardului acestui tip de retele locale (numit 802.3 10 Base-T) si b) facilitarea intretinerii unei multimi de astfel de retele care ar fi dispersate intr-o organizatie mare.

Lansarea firmei Compaq in domeniul retelelor locale reprezinta un alt eveniment semnificativ. De fapt Compaq a facut acest pas oarecum involuntar, deoarece ea a cautat mai mult sa gaseasca o utilitate (ca server de retea) a puternicului sau sistem mutiprocesor Systempro. S-ar parea ca aceasta ar fi singura justificare pentru un utilizator care intentioneaza sa achizitioneze un sistem atit de puternic si de scump. Ceea ce ramine de vazut este daca solutia propusa de Compaq va fi acceptata de majoritatea

utilizatorilor. Totusi, s-ar parea ca prestigiul firmei da aceasta garantie.

Desi anul 1989 n-a rezolvat problema standardizarii produselor de control retea locala, variante functionale ale acestora au inceput sa apara deja si sa fie intens folosite. Aparitia lor marcheaza un alt eveniment in domeniul retelelor locale. Produsele firmelor Cabletron Systems Inc., Hewlett- Packard, Synoptics Communications Inc., culeg statistici privind activitatea retelei, identifica defectele si ofera reprezentari grafice ale topologiei retelei locale. Scopul unor astfel de produse este de a minimiza timpul de cadere al retelei si de a reduce munca necesara pentru a pastra activa reteaua. Ultimele produse din aceasta categorie, numite monitoare de retea, ajuta la identificarea erorilor si la imbunatatirea performantelor retelei. Daca la acestea mai adaugam tendintele de simplificare a interfetei cu utilizatorul, rezulta ca-n viitor reteaua locala va fi si mai usor de folosit si gestionat.

Aparitia superserver-elor reprezinta un alt elemert marcant in domeniul retelelor locale. Acestea permit cresterea capacitatii retelei la ordinul sutelor de utilizatori fara insa a-i creste complexitatea. Superserver-ele rivalizeaza minicalculatoarele suportind tot atitia utilizatori ca si majoritatea acestora insa cu o performanta ridicata si la o zecime din cost. Aparitia lor rezolva problema gestiunii multimii de servere de retea dintr-o organizatie mare (personalul de intretinere nu trebuie sa serveasca o multime de servere raspindite ci doar superserver-ul). Este de mentionat ca ascensiunea retelelor a diminuat piata minicalculatoarelor ceea ce se observa si din faptul ca firme renumite ca Data General, NCR, Prime au adoptat soft-ul de retea locala Portable Netware al renumitei firme Novell pe mediile lor de calcul.

Un alt element marcant il constituie aparitia server-elor baze de date de tip SQL (SQL Server al firmei Microsoft, Netware SQL al firmei Novell etc.) ivindu-se posibilitatea de a crea baze de date distribuite. Pentru elaboratorii de aplicatii acest eveniment reprezinta de fapt argumentul hotaritor in decizia lor de a opta pentru achizitionarea unei retele locale, daca avem in vedere faptul ca pina-n prezent singurele tipuri de aplicatii existente pentru acestea erau posta electronica si serviciile de transfer de fisiere si listare. De-abia acum reteaua de PC-uri poate rivaliza calculatoarele mari, avind insa in plus avantajul unei multitudini de aplicatii front-end specifice, foarte performante (procesoare de text, spreadsheet-uri, etc.).

In domeniul integrarii platformelor multiple, pasul hotaritor l-a facut firma 3Com prin introducerea protocolului Netbios de salvare a memoriei si a schemei "swapping" de protocol (care incarca un anumit protocol dupa cum acesta este cerut de aplicatia in curs). In acest fel statiile de lucru vorbesc protocolul gazdei.

Performantele deosebite ale produselor firmei Novell (leader-ul de necontestat in domeniul retelelor locale) reprezinta un alt element care va duce la larga acceptare a LAN- urilor. Sistemul de operare de retea Netware 386 permite elaboratorilor aplicatiilor de retea sa treaca in citeva zile o aplicatie in C intr-un modul incarcabil in retea (Network Loadable Module). Exista deci premize ca in 1990 numarul aplicatiilor de retea sa creasca simtitor. In plus Novell a luat decizia de a apropia si mai mult software-ul de retea de utilizator prin compatibilizarea produsului Netware 386 cu front- end-ul specific utilizatorului. Astfel un utilizator al produselor Appleshare sau Tops poate stoca fisiere pe un Netware Server.

Mark Stephens editor la Infoworld considera ca argumentele prezentate mai sus il indreptatesc sa afirme ca 1990 va fi anul retelelor locale. Iata ca previziunea din 1981 a lui Bob Metcalfe (inventatorul protocolului Ethernet si unul din fondatorii firmei 3Com) precum ca 1982 va fi anul retelelor locale a fost nerealista. Aceasta deoarece hardware-ul de retea era destul de scump pe vremea aceea iar software-ul de retea prea primitiv. Idealul sau insa, de a lega toata lumea calculatoarelor personale de pe vremea aceea (PC, XT, Apple, CP/M) este in curs de a se implinii, dar la alte proportii.

HOBBY

IMBUNATATIREA PERFORMANTELOR MICROCALCULATOARELOR COMPATIBILE ZX-SPECTRUM

Adrian Popa

Unul dintre cele mai raspindite Home-Computere este produsul firmei Sinclair: **ZX-SPECTRUM**. Flexibil, fiabil si usor de manevrat, el se preteaza la utilizari "amatoricesti" iar gama de jocuri propuse de diferite firme este foarte mare si satisface cele mai variabile preferinte.

Schema foarte simpla a acestui microcalculator a imbiat pe multi constructori amatori sa-si realizeze singuri acest aparat. In prezent circula mai multe scheme impreuna cu cablajele imprimate aferente, unele dintre ele fiind produsul unor constructori amatori, altele produse de fabrica (HC85, Cobra, TIM-S).

Una dintre principalele probleme de care se lovesc constructorii amatori este "punerea in functiune". Chiar si montajele realizate foarte ingrijit (cu piese testate, cu toate scurtcircuitele eliminate) necesita unele reglaje "de finete". Un capitol important il constituie "potrivirea" retelelor RC de intirziere, folosite de majoritatea proiectantilor, in special in zona de formare a semnalelor de acces la memoriile dinamice.

Alegerea valorilor rezistentelor si condensatorilor din aceste retele se face, de obicei, prin incercari. Dar se constata ca, si dupa aceste reglaje, unele dintre montaje refuza sa functioneze determinist: nu incarca corect toate programele, depind critic de tensiunea de alimentare, dau rateuri cind se incalzesc putin etc.

O solutie de remediere a acestor anomalii v-o propunem in continuare. Este vorba de montajele care folosesc un "multiplexor" cu rezistente pentru legarea iesirilor memoriei video la magistrala de date a microprocesorului.

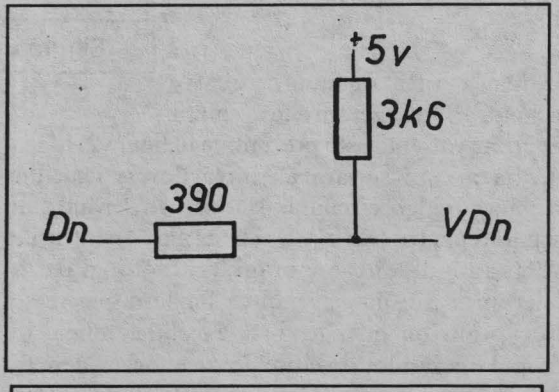


Figura 1

Memoria video este un "punct cald" al microcalculatoarelor compatibile **SPECTRUM** deoarece in aceasta zona se fac cele mai multe accesuri la informatii. Aici se pastreaza informatia afisata pe ecran si pe care controlerul video o baleiaza cu 875 KHz (pe grupuri de 8 puncte) dar tot aici este rezidenta si zona variabilelor de sistem BASIC. Aceste variabile sint foarte importante pentru sistem iar modificarea accidentala a uneia dintre ele poate conduce programul care ruleaza pe un drum incontrolabil (de cele mai multe ori in HALT sau RESET). Din aceasta cauza este gresit sa credem ca o functionare defectuoasa a acestei zone de memorie intre 16384 (4000H) 32767 (7FFFH) cauzeaza numai "paraziti" pe imagine.

Schema originala se prezinta ca in figura 1. Liniile Dn sint liniile de date ale microprocesorului iar VDn sint liniile de date de iesire ale memoriilor dinamice. Teoretic, aceasta schema ar trebui sa functioneze corect, practic insa mai multi factori concura la o functionare nesigura:

- pragurile de trigger nesimetrice si variabile (in limite restrinse) cu temperatura ale circuitelor TTL;
- incarcarea capacitiva a magistralei de date a microprocesorului care, impreuna cu rezistenta de 390 W, formeaza o retea de intirziere;
- fan-out-ul mic al memoriilor dinamice care, in afara de aceasta retea de cuplare, mai conduc si intrarile TTL ale circuitelor de serializare (495).

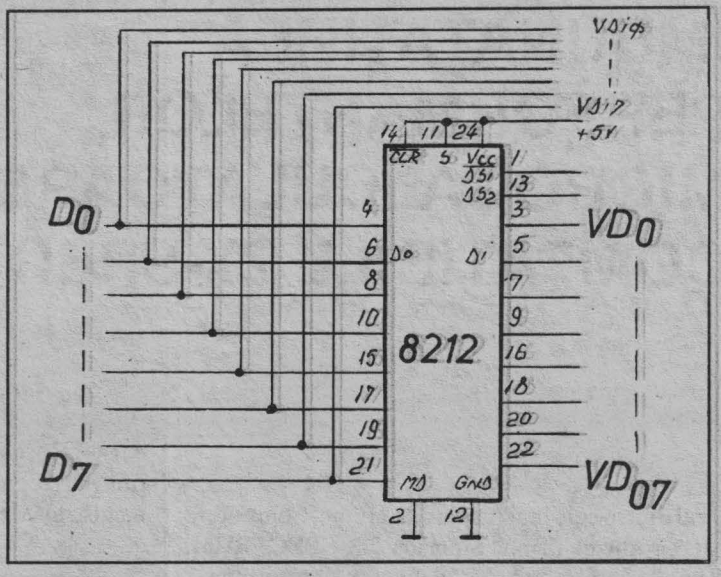


Figura 2

Pentru inlaturarea acestor neajunsuri va sugeram citeva solutii.

O prima varianta este prezentata in figura 2. Ideea de baza este separarea intrarilor si iesirilor memoriei video, circuitele 4116 sau 4164 avind pini separati pentru DI (pin 2) si DO (pin 14). Intrarile se leaga direct pe magistrala de date a microprocesorului in timp ce liniile de iesire se trec printr-un buffer tri-state validat numai in timpul operatiilor de citire din memoria video. In aceasta varianta am folosit ca buffer tri- state un circuit 8212 cu punctul strobe (11) legat la (+). In aceasta situatie, latch-ul intern este in mod "transparent" si circuitul se poate folosi ca un simplu amplificator de magistrala. La pinii DS1 si DS2 se formeaza semnalele de selectie care, pentru starea "deschis", trebuie sa fie DS1 = - si DS2 = 1. Pentru ca amplificatorul sa fie deschis in starea "citire din memoria video" vom aduce la DS1 semnalul de la iesirea decodificatorului de adresa (442), semnalul VRAM activ "low" care identifica accesul la adresele 4000H 7FFFH (A15 = -, A14 = 1, MREQ = -, RFSH = 1). La DS2 vom aduce semnalul de citire RD dupa o inversare, deci activ "high". Acest semnal gata inversat este disponibil pe placa. Circuitul se

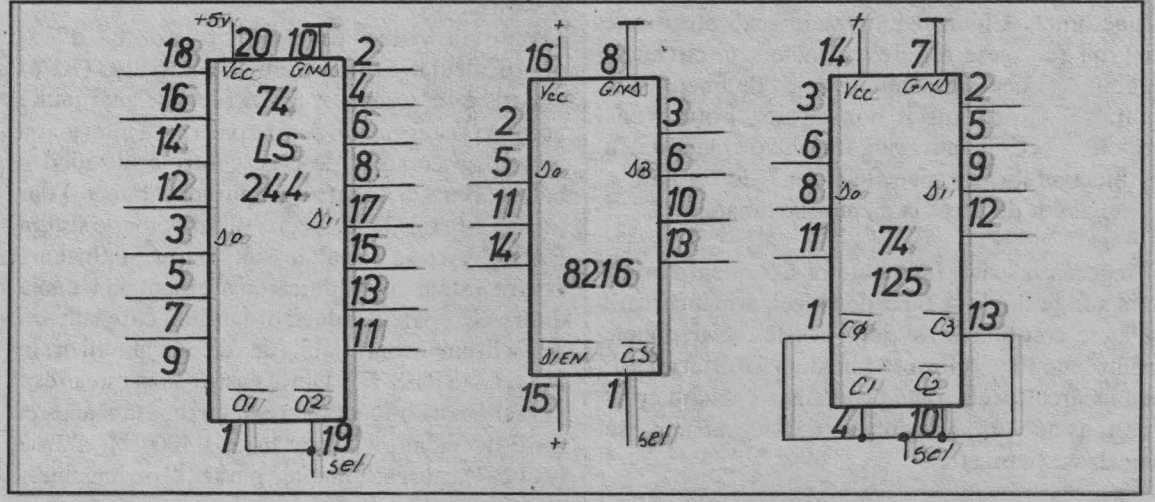


Figura 3

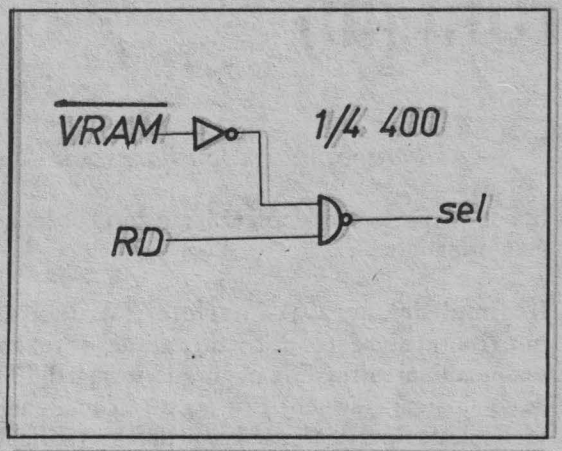


Figura 4

monteaza pe placa microcalculatorului, lipit cu prenandez, intr-o zona libera (sau chiar peste un alt circuit) cu terminalele in sus. Legaturile la placa se fac cu fir de wrap.

Pe aceasta solutie de baza se pot broda multe variante care se refera la folosirea unor diferite circuite pe post de buffer tri- state. In figura 3 sint prezentate citeva asemenea circuite. In cazul circuitelor 8216 si 74125 care nu contin decit patru celule sint necesare cite doua bucati pentru magistrale de opt biti. Dezavantajul acestor variabile este acela ca au un singur pin de validare si deci semnalul de selectie RD.VRAM trebuie fabricat extern cu porti libere (daca exista!!) de la alte circuite de pe placa.

In figura 4 este prezentata schema de selectie a acestor amplificatoare de magistrala, activa "low", folosind acelasi semnal RD inversat, ca si in primul caz, impreuna cu semnalul VRAM de data aceasta inversat (multe variante de SPECTRUM ofera acest semnal gata inversat).

Va recomandam sa incercati aceste solutii numai dupa o consultare atenta a schemei dupa care este realizat microcalculatorul dumneavoastra si dupa identificarea pe placa a tuturor traseelor implicate in aceasta operatiune. Nu uitati! Este mai usor sa tai decit sa faci la loc!! Deci, multa atentie si rabdare. Succes!

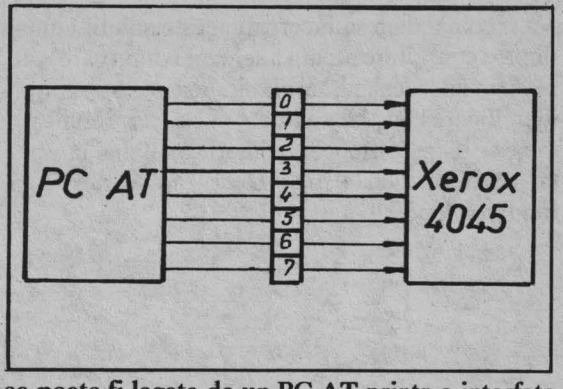
EDITOARE SI FONTURI (III)

Tiberiu Spircu si Ion Paraschiv

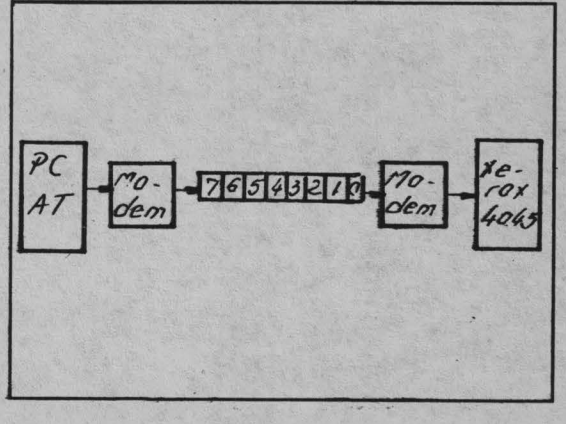
IMPRIMANTE LASER

Imprimantele laser sint instrumente de redare pe hirtie a informatiei (de regula grafice) provenite de la un calculator (host).

Sa exemplificam cu imprimanta laser Xerox 4045;



ea poate fi legata de un PC AT printr-o interfata paralela, informatiile scurgindu-se byte cu byte, sau printr-o interfata seriala.

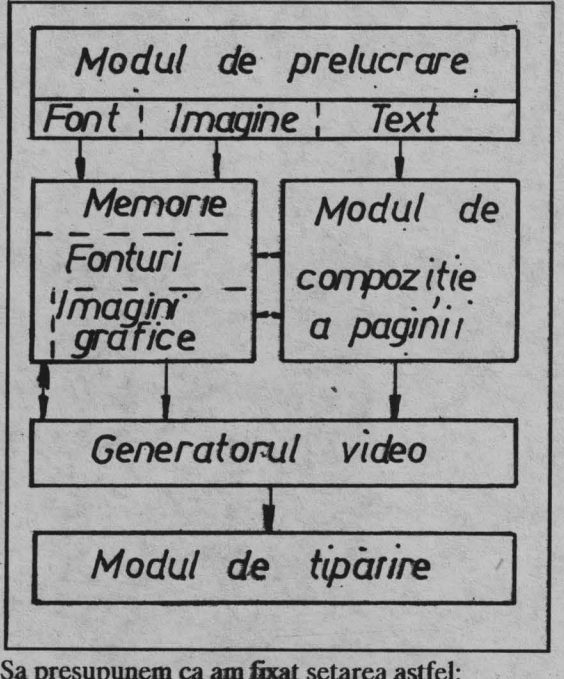


Orice interfata implica doua aspecte:

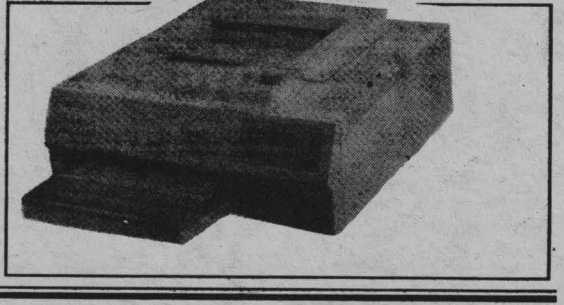
a) mediul fizic de transmisie (cablurile, conectoarele, eventual modemurile);

b) conventiile logice ce asigura transferul ordonat al informatiilor.

Regimul de lucru al interfetei (si deci al imprimantei) este determinat de setarea asa-numitului cartus de configuratie, format din 32 de comutatoare (switch). Prin aceste comutatoare se stabileste tipul de transmisie al datelor (paralel sau serial) ce se interpreteaza ca End Of Line (CR + LF, doar CR, doar LF), citi biti se preiau din fiecare byte (7 sau 8), care este codificarea de baza (ISO, ASCII, EBCDIC sau IBM PC), care este protocolul folosit in cazul transmisiei seriale (XON sau XOFF, ETX sau ACK, viteza de transmisie), daca se controleaza sau nu paritatea, etc.



Sa presupunem ca am fixat setarea astfel:



00001000 11111100 00000000 00000000

ceea ce asigura, printre altele, faptul ca interpretarea byte- ilor receptionati (paralel) de catre imprimanta se face conform codului IBM PC.

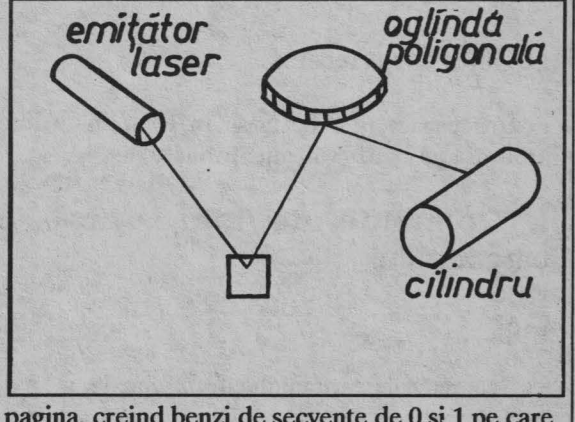
Imprimanta propriu-zisa este formata din cinci componente: modulul de prelucrare, memoria, modulul de compozitie a paginii, generatorul video, modulul de tiparire.

Modelul de prelucrare determina ce fel de date se preiau: fonturi, imagini grafice sau text obisnuit. Fonturile sint desixelizate apoi stocate in memorie. Si imaginile grafice sint stocate in memorie pina la imprimarea lor. Textele sint expediate spre modulul de compozitie al paginii unde:

a) fiecare byte este sau interpretat drept comanda, sau asociat cu caracterul corespunzator al fontului;

b) se calculeaza pozitia pe pagina;

c) atunci cind s-a terminat o pagina ea este transferata spre generatorul video. Acesta preia din memorie BITMAP-ul fiecarui caracter, impreuna cu informatiile privind pozitia pe



pagina, creind benzi de secvente de 0 si 1 pe care le transmite spre scanner. (Fiecare banda este formata din 32 de linii, fiecare linie continind maxim 2550 biti.)

Modulul de tiparire preia foaia de hirtie si o trece in jurul unul cilindru (drum) ce a fost expus anterior prin fata unei raze laser. Fiecare bit din banda transmisa spre scanner face ca raza de laser sa lumineze sau nu cilindrul care este maturat cu praf avind proprietati electrostatice (toner). Tonerul se transfera de pe cilindru pe foaia de hirtie apoi, prin incalzire, este fixat pe hirtie.

In numarul urmator vom prezenta felul in care editam texte folosind sistemul de editare original MatTex.

Anexa I. Descrierea unui fisier-font-imprimanta tip HP

Fisierele-font-imprimanta tip Hewlett-Packard sint formate din trei zone indicate, in cele de mai jos, prin A, B si C.

A. Comanda de incarcare a fontului

Esc) s ... W

numarul l de bytes in descrierea generala, exprimat in cifre zecimale

B. Descrierea generala

0-1 Lungimea partii de descriere a fontului (numar intreg, anume l);

2 Byte neutilizat (si egal cu 0);

3 Tipul fontului (byte avind valoarea 0, 1 sau 2). In caz ca acest tip este 0 avem LOWCHR = 32 si HIGCHR = 127; in caz ca tipul este 1 avem LOWCHR = 32 si HIGCHR = 255; iar in caz ca tipul este 2 avem LOWCHR = 1, HIGCHR = 255 iar caracterele 7, 8,...,15 si 27 nu sint descrise (semne vide). Vom folosi, de regula, tipul 0 intrucit, in acest caz, caracterele descrise corespund tastarii normale. Sistemul de editare VENTURA foloseste fonturi de tipul 1;

4-5 Bytes neutilizati (egali cu 0);

6-7 ASCMAX (numar intreg intre 0 si 65535), exprimat in pixeli;

8-9 Latimea maxima permisa a BITMAP-urilor caracterelor (numar intreg intre 1 si 4200), exprimata in pixeli;

10-11 Inaltimea maxima permisa a BITMAP-urilor caracterelor (numar intreg intre 1 si 4200), exprimata in pixeli. Este egala cu ASCMAX + DESMAX;

12 Orientarea (byte avind valoarea 0 sau 1). In caz ca valoarea este 0 semnele sint imprimate pe foaia de hirtie in "portret"; in caz ca valoarea este 1 imprimarea se face in "peisaj". De regula, foaia de hirtie format A4 este considerata "portret" iar cea format A3 este considerata "peisaj";

13 Spatierea (byte avind valoarea 0 sau 1). In caz ca valoarea este 0 toate semnele au aceeasi latime (adica avem de-a face cu spatiere fixa); in caz ca valoarea este 1 semnele pot avea latimi diferite (spatierea este "proportionala"). De regula, vom folosi spatierea "proportionala" cu exceptia fonturilor ce simuleaza masini de scris clasice;

14-15 Simbolul (numar-intreg). Este legat de standardul tipografic folosit. Putem considera ca are valoarea 5. In sistemul VENTURA, fonturile folosite au simbolul 277 = 115 h;

16-17 Lungimea blocului (numar intreg intre 0 si 16800), exprimata in sferturi de pixel;

18-19 Inaltimea fontului (numar intreg intre 0 si 10922), exprimata in sferturi de pixeli. Este legata de inaltimea literelor descrise, exprimata in puncte tipografice. Vom folosi, de regula, fonturi cu inaltimea literelor intre 6 si 24 de puncte tipografice (deci cu inaltimea intre 120 si 480); pentru scrisul obisnuit se folosesc fonturi cu inaltimea literelor de 10 sau 12 p.t.;

20-22 Bytes ignorati;

23 Inclinare (byte avind valoarea 0 sau 1). Indica faptul ca semnele sint drepte (valoarea 0), respectiv inclinate (= "italice", valoarea 1);

24 Ingrosarea (byte avind valori intre 0 si 7 sau intre 249 si 255). Indica faptul ca semnele sint normale (valoarea 0), ingrosate (= "bold", valori intre 1 si 7) sau subtiate (valori peste 249). Vom folosi, de regula, fonturi avind ingrosarea 0 (normale), 3 ("bold") sau 253 ("light");

25 Standardul tipografic (byte). Are valoarea 3 pentru semne "Courier", 4 pentru semne "Helvetica", 5 pentru semne "Times Roman" etc;

26-29 Bytes ignorati;

30 Distanta de subliniere (byte). Reprezinta distanta, exprimata in pixeli, intre linia de baza si marginea de sus a liniei de subliniere, pentru semnele subliniate. Valorile mai mari decit 127 sint interpretate prin complementarizare corespunzind cazului in care linia de subliniere este peste linia de baza;

31 Grosimea liniei de subliniere (byte), exprimata in pixeli. Unele aparate ca, de exemplu HP Laser Jet II, ignora aceasta informatie considerind ca linia de subliniere are intotdeauna grosimea de 3 pixeli;

32-33 INTLIN (numar intreg intre 0 si 65535). Unele aparate ignora aceasta informatie;

34-39 Bytes ignorati;

40 Completare la lungimea blocului, exprimata in miimi (mai precis 1/1024) de pixel;

41 Completare la inaltimea fontului, exprimata in 1/1024 de pixel;

42-47 Bytes ignorati;

48-(l-1) Denumirea standard a fontului (16 caractere) urmata de comentarii.

Observatie:

Fonturile din sistemul VENTURA au o alta structura, incepind cu bitul 26; pentru acesta, de regula, l = 75 iar denumirea fontului apare incepind cu bitul 44.

C. Zona caracterelor.

Pentru caracterele descrise in font, in ordine arbitrara, urmeaza subzone formate din:

I. Comanda de identificare a caracterului

Esc * c ... E

caracterul exprimat in cifre zecimale

II. Comanda de incarcare a descrierii caracterului

Esc (s ... W

numarul de bytes in descriere

III. Descrierea caracterului

0 Formatul datelor (bytes). Pentru aparatele HP Laser Jet II acesta are valoarea 4;

1 Indicator de continuare (byte luind valoarea 0 sau 1). In cazul in care acesta are valoarea 1, descrierea este doar o continuare iar urmatorii bytes constituie continuarea BITMAP-ului.) Mai precis, in comanda de incarcare a descrierii caracterului, numarul de bytes este limitat la 32767. In caz ca o descriere are volumul mai mare, ea trebuie separata in - cel putin - doua blocuri. Primul bloc va fi incarcat normal, deci ca indicatorul de continuare 0, iar urmatorii vor avea indicatorul de continuare 1.) Aceasta este o situatie rara corespunzind unor semne ce ocupa peste 16 centimetrii patrati. De regula, indicatorul de continuare va avea valoarea 0 iar informatiile ce urmeaza vor avea urmatoarele semnificatii;

2 Lungimea in bytes a capului descrierii (byte). Evident, valoarea este 14;

3 Clasa datelor (byte). Pentru aparatele HP Laser Jet II are valoarea 1;

4 Orientarea (byte luind valoarea 0 sau 1). Trebuie sa coincida cu orientarea fontului in ansamblu (vezi B 12 mai sus);

5 Byte neutilizat (avind valoarea 0);

6-7 OFFSET (numar intreg avind valori intre -4200 si 4200). Reprezinta distanta, in pixeli, de la originea semnului pina la marginea din stinga a BITMAP-ului;

8-9 ASC (numar intreg avind valori intre -4200 si 4200). Reprezinta distanta, in pixeli, de la linia de baza pina la marginea de sus a BITMAP-ului;

10-11 WIDTH (numar intreg avind valori intre 1 si 4200). Reprezinta latimea utila BITMAP-ului, in pixeli. Trebuie sa nu depaseasca latimea maxima permisa (inregistrata in B 8-9);

12-13 HEIGHT (numar intreg avind valori intre 1 si 4200). Reprezinta inaltimea maxima permisa (inregistrata in B 10-11);

14-15 ACTWID (numar intreg avind valori intre 0 si 16400). Este exprimat in sferturi de pixel;

16-(a-1) BITMAP (string). Imaginea este descrisa linie dupa linie (de sus in jos), fiecare linie de la stinga spre dreapta necesitind un numar intreg de bytes, anume (WIDTH + 7)/8. Fiecare byte reprezinta 8 pixeli consecutivi, de la stinga spre dreapta, bitii egali cu 1 corespunzind pixelilor negri.

Anexa II. Descrierea unui fisier-font-imprimanta tip Xerox

Fisierele-font-imprimanta tip Xerox (utilizate de aparatele X 3700 si X 4045) sint formate din sase zone. Dupa desixelizare, descrierea este urmatoarea:

A. Zona de recunoastere a fontului,

formata din 2 bytes fiecare avind valoarea 170 = AAh.

B. Descrierea generala

1 Versiunea (byte);

2 Tipul fontului (byte). Daca valoarea acestui byte este un multiplu de 4 atunci fontul este de orientare "portret" si de spatiere fixa. Daca valoarea este un multiplu de 4+1 atunci fontul este de orientare "peisaj" si de spatiere fixa. Daca valoarea este un multiplu de 4+2 atunci fontul este de orientare "portret" si de spatiere "proportionala" in celelalte cazuri fiind de orientare "peisaj" si de spatiere "proportionala";

3-4 Lungimea fisierului-font (numar intreg intre 0 si 65535) desixelizat;

5-24 DENFON (string de 20 caractere);

25 Distanta de subliniere (byte). Reprezinta distanta intre linia de baza si marginea de sus a liniei de subliniere (pentru semnele subliniate), exprimata in pixeli;

26 Grosimea liniei de subliniere (byte), exprimata in pixeli. Aceste ultime doua caracteristici sint utilizate in orientarea "portret";

27 Lungimea cratimei (byte);

28 Grosimea cratimei (byte). Aceste doua caracteristici sint utilizate in orientarea "peisaj";

29 Inaltimea exponentilor (byte). Reprezinta, numarul de pixeli cu care este urcata linia de baza in cazul considerarii semnului ca exponent;

30 Adincimea indicilor (byte). Reprezinta numarul de pixeli cu care este coborita linia de baza in cazul considerarii semnului ca indice;

31-32 DESMAX (numar intreg);

33-34 ASCMAX (numar intreg);

35-36 INTLIN (numar intreg);

37 LOWCHR (byte);

38 HIGCHR (byte);

39-46 Clasificare (string de 8 caractere). Ignorat.

C. Zona de offset.

Este formata din 256 bytes, cite unul pentru fiecare caracter, in ordine. In byte-ul corespunzator unui caracter este inregistrata valoarea OFFSET. Valorile mai mari decit 127 sint interpretate ca negative, prin complementarizare.

D. Zona caracteristicilor.

Contine, pentru fiecare caracter cuprins intre LOWCHR si HIGCHR, in ordine, cite 8 bytes avind urmatoarea interpretare:

0-1 Volumul (numar intreg). Reprezinta numarul de bytes al BITMAP-ului;

2-3 Adresa (pointer) incepind de la care vom intilni BITMAP-ul semnului;

4 Byte neutilizat avind valoarea 255;

5 Inaltimea h (byte). Este valabila relatia:

ASC + DESC = (63 - h) * 8;

6 Abaterea b fata de linia de baza (byte). Este valabila relatia:

DESC = -b * 2;

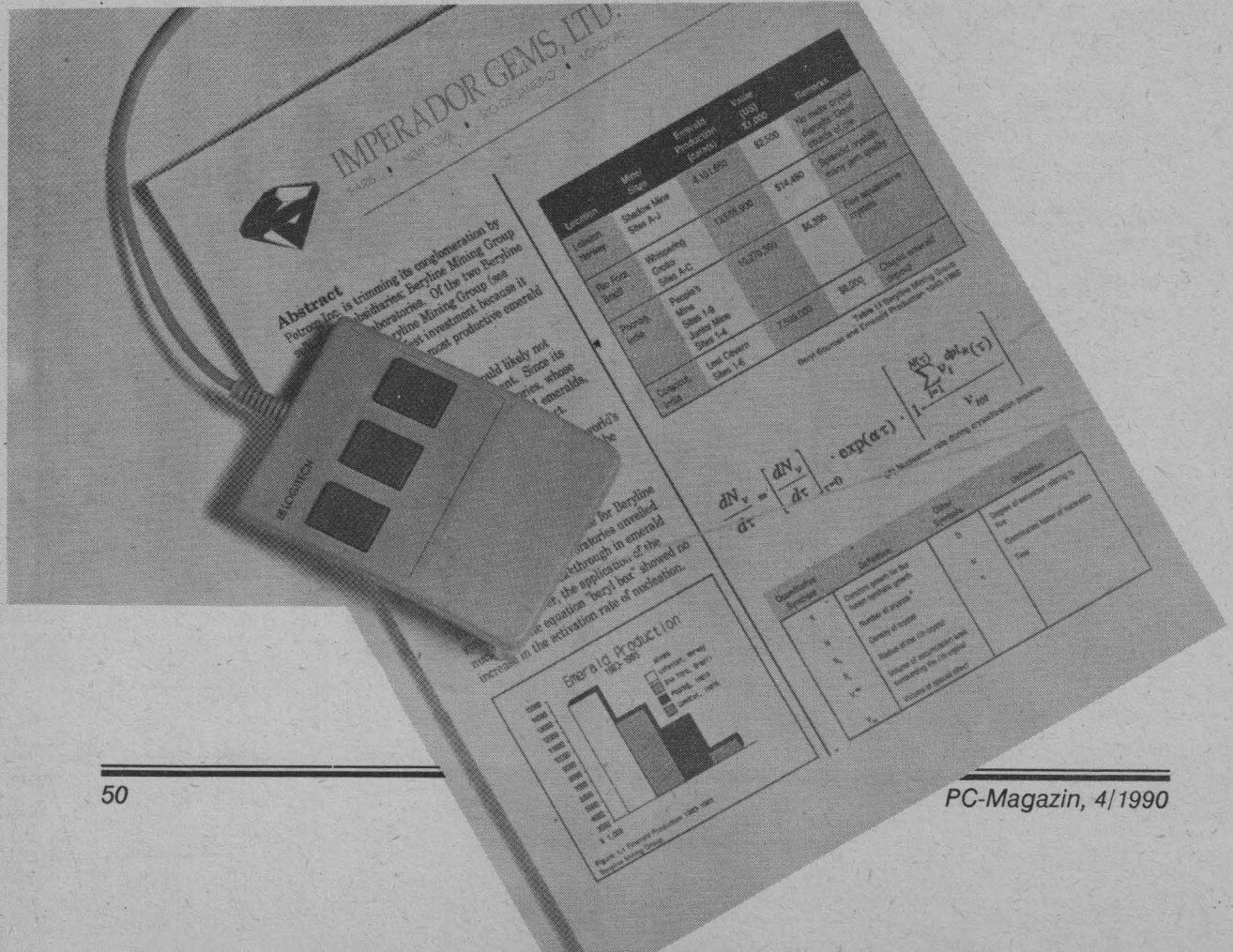
7 ACTWID (byte).

E. Zona descrierilor grafice

Contine, pentru fiecare caracter cuprins intre LOWCHR si HIGCHR, BITMAP-ul. Imaginea este descrisa coloana dupa coloana (de la stinga la dreapta) fiecare coloana de jos in sus necesitind un numar intreg de bytes. Ca de obicei, bitii egali cu 1 corespund pixelilor negri.

F. Zona de incheiere.

Este formata din 8 bytes fiecare avind valoarea 85 = 55 h.



FISIERE INDEXATE

Fisierele folosite pentru stocarea informatiei in bazele de date sint cel mai adesea organizate ca fisiere indexate in acces aleator.

Atunci cind informatiile sint stocate intr-un fisier de date indexat, corpul inregistrarii este memorat intr-un fisier de date principal, iar o forma abreviata a acestuia care consta in general dintr-un cimp cheie si un numar de inregistrare (sau un pointer), este stocata intr-un fisier index separat. Cimpul cheie este un element din inregistrare pe baza caruia se va face cautarea in fisier.

De obicei, indexarea se realizeaza pe mai multe cimpuri cheie, deci vor fi folosite mai multe fisiere index asociate cu acelasi fisier de date.

Ideea de baza pentru construirea unui fisier de date indexat este foarte simpla: deoarece indexul este o submultime a datelor continute in fisierul principal, poate fi inspectat mult mai rapid. De exemplu: chiar daca fisierul de date are marimea de citiva megabytes, in general este posibil ca fisierul index sa fie tinut intreg in memorie la un moment dat.

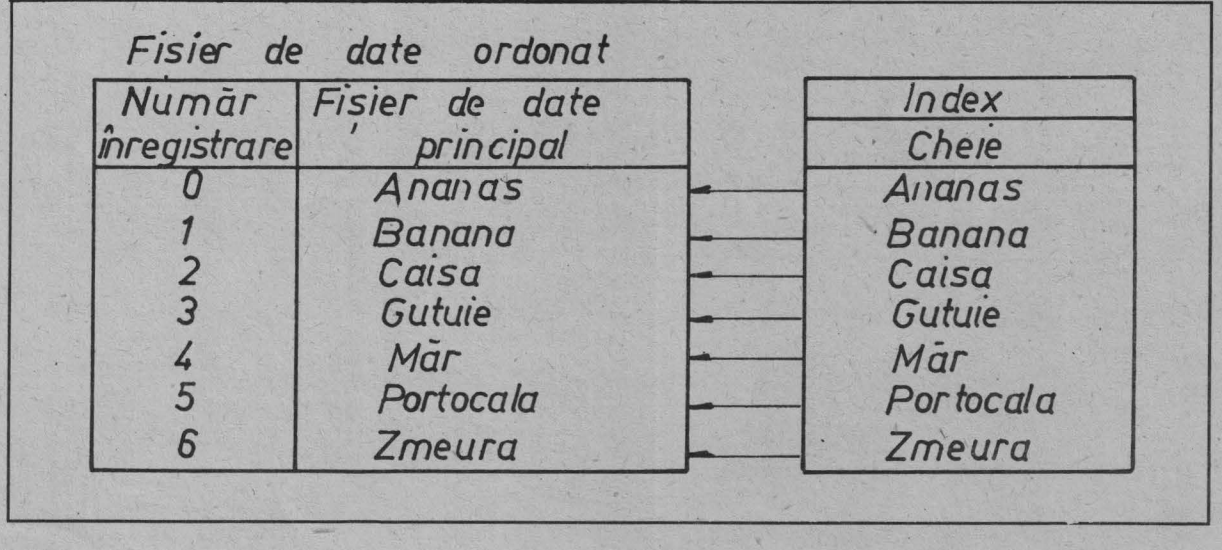
Pentru maximizarea vitezei de cautare, in interiorul fisierului index se pot folosi structuri si tehnici speciale, lucru practic imposibil de realizat in fisierul de date.

ORGANIZAREA FISIERULUI PRINCIPAL

[1] Fisierul principal si fisierul index se mentin sortate in aceeasi ordine

Avantaje:

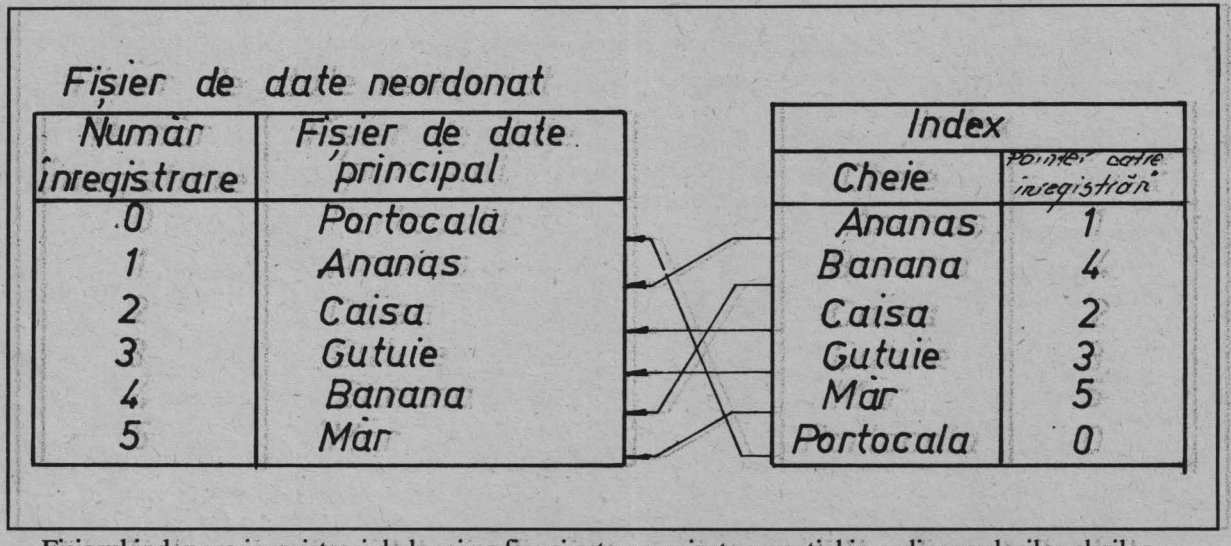
- fisierul index este compact (deoarece nu contine numere de inregistrari),pozitia unei chei date indicind direct pozitia inregistrarii corespondente;
- fisierul index poate fi folosit pentru a gasi o inregistrare data, inregistrarile urmatoare putind fi prelucrate fara viitoare referiri la index.



Dezavantaje:

- in cazul folosirii mai multor fisiere index este necesar a se stabili indexul primar, deoarece fisierul principal nu poate fi sortat decit dupa un anumit criteriu; de aceea fisierele index secundare trebuie sa contina in structura lor si numerele de inregistrari (deci o structura mai putin compacta);
- sint necesare prelucrari elaborate asupra fisierului principal si asupra indexului in cazul adaugarii sau stergerii unei inregistrari sau a actualizarii cimpului cheie - acestea vor consuma timp si spatiu pe disc.

[2] Fisierul de date principal neordonat



Fisierul index are inregistrari de lungime fixa si este organizat secvential in ordinea valorilor cheilor.

Aceasta organizare este mai eficienta pentru prelucrari asupra fisierului principal, actualizarea fisierului index realizindu-se la fiecare adaugare sau modificare de inregistrari.

ORGANIZAREA FISIERULUI INDEX

[1] Fisierul index organizat secvential

Cea mai simpla forma de organizare a fisierului index este cea prezentata in figura 2, constind dintr-o lista de chei asociate cu numerele de inregistrari. In acest caz sint posibile:

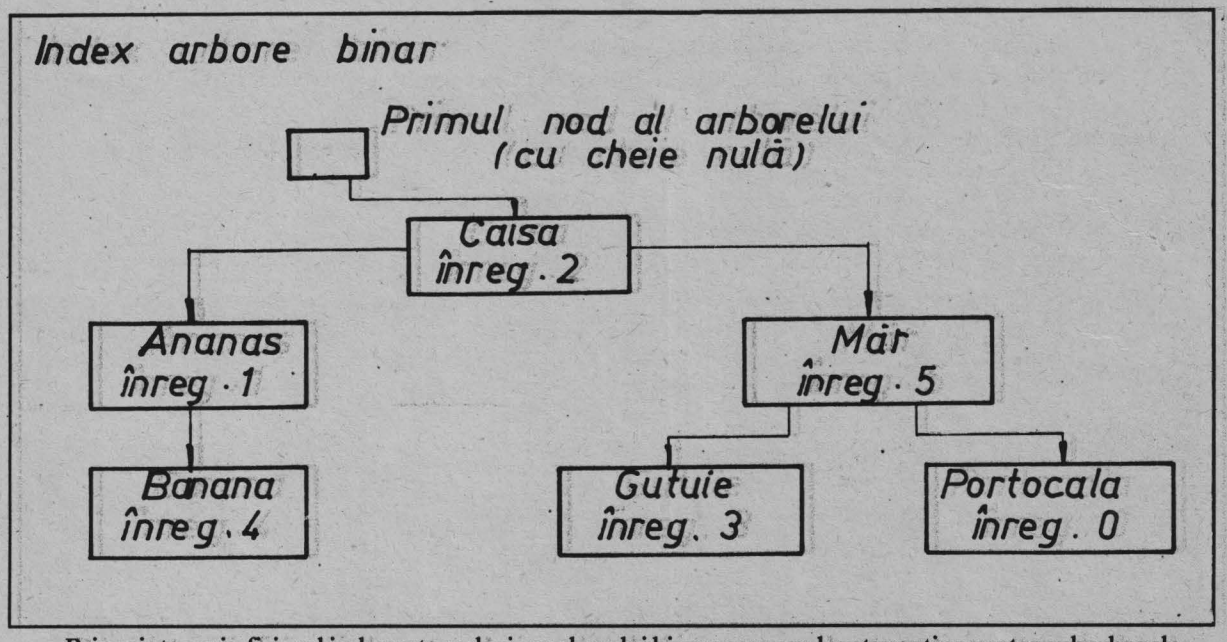
- cautarea secventiala, pina la gasirea cheii dorite;
- cautare binara (daca fisierul index are un numar fix de inregistrari).

Timpul de cautare depinde de continutul fisierului index si de valoarea cheii de cautare, variind pentru valori diferite ale cheilor.

[2] Fisierul index organizat ca arbore binar

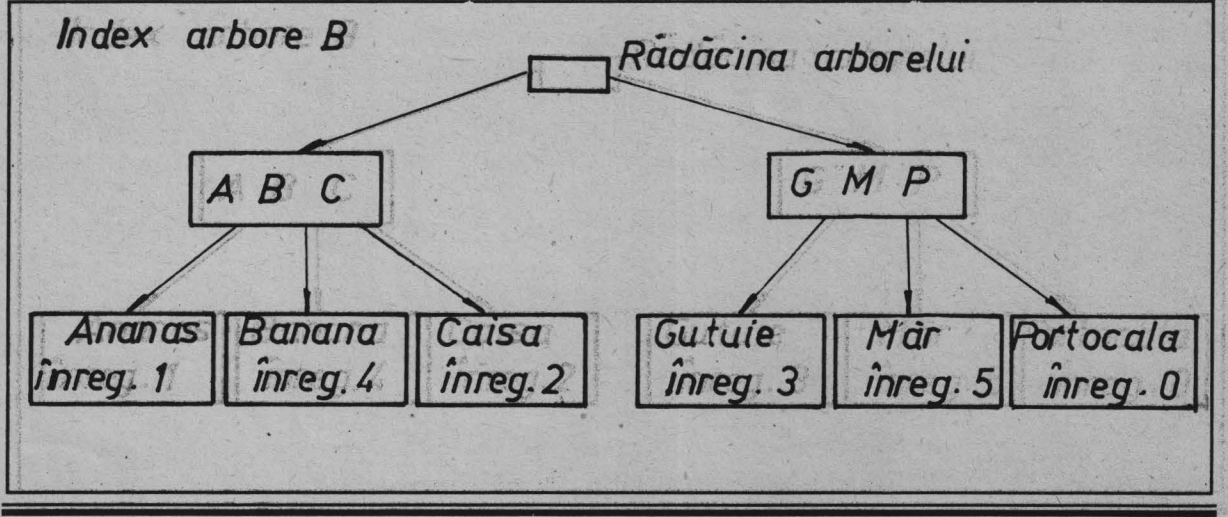
Aceasta este o forma de organizare superioara, constind in implementarea unui arbore binar in care fiecare intrare in fisierul index este un nod al arborelui. Un nod contine:

- valoarea cheii;
- pointer catre inregistrarea corespunzatoare din fisierul principal;
- pointeri catre subarborii sting si drept.



Prima intrare in fisierul index este radacina arborelui binar, care are legatura stinga catre subarborele care contine chei "mai mici", iar legatura dreapta spre subarborele care contine noduri cu chei "mai mari" decit cea din radacina. Pentru gasirea unei chei date, arborele binar este parcurs pornind de la radacina. In fiecare nod cheia este comparata si in functie de rezultatul comparatiei este parcurs apoi subarborele corespunzator. Procedeul este continuat pina la gasirea cheii dorite. Aceasta metoda de cautare se poate implementa cu succes in limbajele care admit recursivitatea. Adaugarea si stergerea de inregistrari in fisierul de date principal implica operatii relativ simple in fisierul index. Organizarea ca arbore binar a fisierului index este neeficienta in cazul anumitor distributii ale cheilor care determina asimetria arborelui si deci variatia marimii timpului de cautare.

[3] Fisier index organizat ca arbore B



Organizarea indexului in arbore binar perfect echilibrat (numit arbore B) presupune ca toti subarborii nodurilor de pe un anumit nivel sa aiba aceesi lungime.

Acest arbore contine doua tipuri de noduri:

- noduri interne care contin pointeri catre alte noduri;
- noduri externe care contin pointeri spre inregistrari din fisierul principal.

Avantajul acestei organizari consta in faptul ca numarul nodurilor care trebuie sa fie examinate pentru gasirea oricarei chei este constant.

EXEMPLE

In continuare sint prezentate doua programe:

- **MAKEIXF.C** - constructie fisiere indexate;
- **SRCHIXF.C** - metode de cautare in fisiere indexate.

Primul program, **MAKEIXF.C**, citeste pina la 100 siruri de caractere.

Cind raspunsul solicitat de prompter este tasta Enter, el creaza un fisier de date principal numit **TESTFILE.DAT**. Programul ordoneaza apoi sirurile de caractere si creaza doua fisiere index: **TESTFILE.IX1** (lista simpla de chei sortate si numere de inregistrari) si **TESTFILE.IX2** (arbore binar).

Al doilea program, **SRCHIXF.C**, deschide fisierele create de programul anterior si solicita o cheie dupa care va face cautarea. Apoi cauta in ambele fisiere index si afiseaza rezultatul cautarii (in cazul in care inregistrarea exista, va afisa numarul cheilor inspectate pentru toate metodele de cautare folosite: secventiala, binara, in arbore binar). Ambele programe pot fi folosite sub **MS-DOS** sau **OS/2** (fara nici o modificare).

Iesirea din cel de al doilea program se poate face apasind tasta Enter sau *Ctrl-C* ori *Ctrl-Break*.

MAKEIXF.C

```
/*
    MAKEIXF.C Creaza un fisier indexat care va fi folosit de SRCHIXF.C

    Utilizatorului ii este solicitata introducerea de 100 siruri
ASCII,
    care vor fi folosite pentru construirea inregistrarilor din
fisierul
    de date principal TESTFILE.DAT (cu lungimea inregistrarii RSIZE
)si
    a doua fisiere index:
    - TESTFILE.IX1 (fisier index simplu, sortat, contine chei si
numere
    de inregistrari);
    - TESTFILE.IX2 (fisier index arbore binar).
    Formatul fisierului index este definit prin constanta KSIZE si
prin
    structurile 'index1' si 'index2'.
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

#define ISIZE  80     /* lg. max. a intrarii */
#define N_ITEMS 100        /* nr. max. de siruri */
#define RSIZE  64     /* lg. inregistrarii */
#define KSIZE   8     /* lg. cheii */

char items[N_ITEMS * ISIZE];

struct _index1 {
    char key[KSIZE];
    int recno;
} index1[N_ITEMS];

struct _index2 {
    char key[KSIZE];
    int recno;
    int left;
    int right;
} index2[N_ITEMS];

int getkeys (void);        /* prototipuri functii */
void writedata (int);
void writeindex1 (int);
void writeindex2 (int);
```

MAKEIXF.C

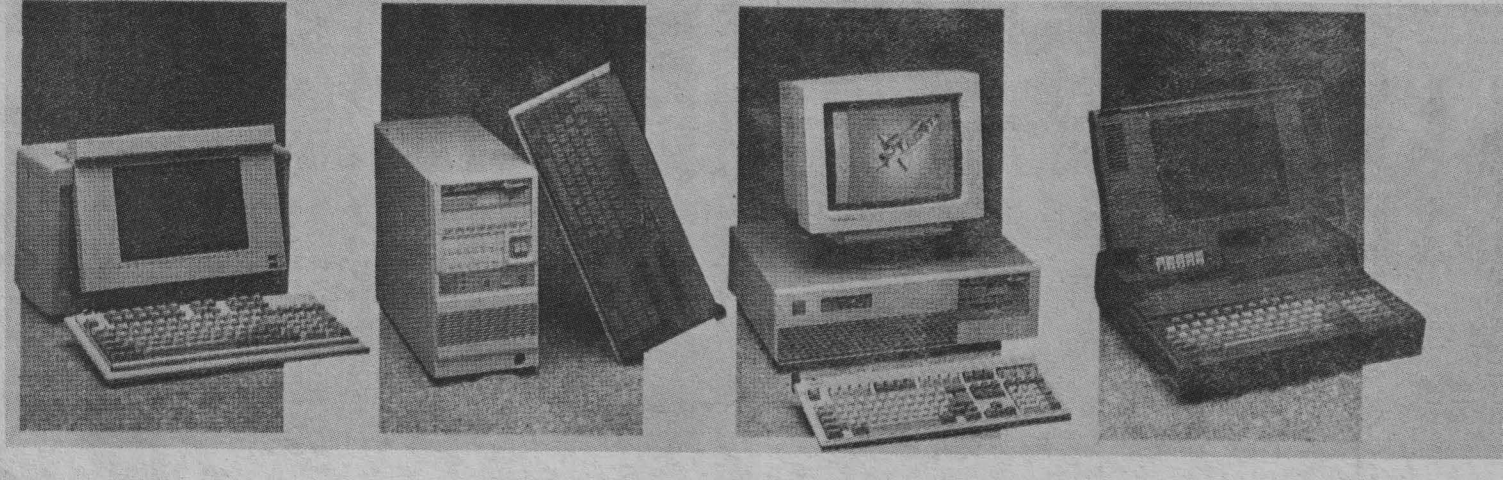
```
void treeinsert (int, int);
void dumptree (int);

main() {
    int i;

    i = getkeys ();          /* citeste cheile inregistrarilor */
    writedata (i);      /* scrie fisierul de date principal */
    writeindex1 (i);         /* scrie indexul secvential */
    writeindex2 (i);         /* scrie indexul arbore binar */
    dumptree (0);       /* afiseaza arborele binar */
}

int getkeys (void) {
    int i, j;

    puts ("\nIntroducere chei inregistrari...");
    i = 0;
    while (i < N_ITEMS) {
     printf ("%2d: ", i+1);
     gets (&items[ISIZE * i]);
     if (items[ISIZE * i] == 0) break;
     for (j = 0; j < i; j++)
         if (strncmp (&items[ISIZE * i], &items[ISIZE * j], KSIZE) == 0)
          break;
     if (i == j) i++;
     else puts ("Cheie dubla. Incercati din nou...");
    }
    return (i);
}

void writedata (recs) {
    int i = 0;
    int fh;
    char recbuf[RSIZE];

    fh = open ("TESTFILE.DAT", O_WRONLY|O_CREAT|O_TRUNC|O_BINARY,
S_IWRITE);
    if (fh == -1) {
     puts ("\nTESTFILE.DAT nu poate fi creat");
     exit (1);
    }
    puts ("\nSe scrie TESTFILE.DAT...");
    while (i < recs) {
     memset (recbuf, 0, RSIZE);
     strncpy (recbuf, &items[ISIZE * i], RSIZE);
     write (fh, recbuf, RSIZE);
     i++;
    }
    close (fh);
```

MAKEIXF.C

```
}

void writeindex1 (recs) {
    int i = 0;
    int fh;

    while (i < recs) {
     strncpy (index1[i].key, &items[ISIZE * i], KSIZE);
     index1[i].recno = i;
     i++;
    }
    if (recs != 0)
     qsort (&index1[0], recs, sizeof(index1[0]), strcmp);
    fh = open ("TESTFILE:IX1", O_WRONLY|O_CREAT|O_TRUNC|O_BINARY,
S_IWRITE);
    if (fh == -1) {
     puts ("\nTESTFILE.IX1 nu poate fi creat...");
     exit (1);
    }
    puts ("\nSe scrie TESTFILE.IX1");
    for (i = 0; i < recs; i++)
     write (fh, (char *)&index1[i], sizeof(index1[0]));
    close (fh);
}

void writeindex2 (recs) {
    int i = 0;
    int fh;

    memset (&index2[0], 0, sizeof(index2[0]));
    index2[0].left = -1;
    index2[0].right = -1;
    while (i < recs) {
     treeinsert (i, i+1);
     i++;
    }
    fh = open ("TESTFILE.IX2", O_WRONLY|O_CREAT|O_TRUNC|O_BINARY,
S_IWRITE);
    if (fh == -1) {
     puts ("\nTESTFILE.IX2 nu poate fi creat");
     exit (1);
    }
    puts ("\nSe scrie TESTFILE.IX2...");
    for (i = 0; i < recs+1; i++)
     write (fh, (char *)&index2[i], sizeof(index2[0]));
    close (fh);
}

void treeinsert (itemno, new) {  /*insereaza un nod nou i arbore*/
    int i, j, node = 0;
```

MAKEIXF.C

```
    do {
     i = node;
     if ((j = strncmp (&items[itemno * ISIZE],
                 index2[node].key, KSIZE)) < 0)
          node = index2[node].left;
     else node = index2[node].right;
    } while (node != -1);
    index2[new].left = -1;
    index2[new].right = -1;
    index2[new].recno = itemno;
    strncpy (index2[new].key, &items[itemno * ISIZE], KSIZE);
    if (j < 0) index2[i].left = new;
    else index2[i].right = new;
}

void dumptree (node) {
    if (node != -1) {
     dumptree (index2[node].left);
     if (node == 0) printf ("\nContinutul arborelui binar este:");
     else printf ("\nNodul = %2d, Nr. inreg. = %2d, Cheie = %8s",
               node, index2[node].recno, index2[node].key);
     dumptree (index2[node].right);
    }
}
```



SRCHIXF.C

```
/*
    SRCHIXF.C Cautare in fisiere indexate pe baza unei chei introduse
    de utilizator.
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

#define RSIZE 64
#define KSIZE  8

static int inspected;

struct _index1 {
    char key[KSIZE];
    int recno;
};

struct _index2 {
    char key[KSIZE];
    int recno;
    int left;
    int right;
};

/* Prototipurile functiilor */
int binsearch (int, char *, int, int);        /* cautare binara */
int seqsearch (int, char *);          /* cautare secventiala */
int treesearch (int, char *);              /* cautare in arbore binar */

main () {
    int i;
    int fhdf, fhix1, fhix2;
    long fsize;
    int frecs;
    char key[80];
    char rec[RSIZE];

    fhdf =  open ("TESTFILE.DAT", O_RDONLY|O_BINARY);
    fhix1 = open ("TESTFILE.IX1", O_RDONLY|O_BINARY);
    fhix2 = open ("TESTFILE.IX2", O_RDONLY|O_BINARY);
    if ((fhdf == -1) || (fhix1 == -1) || (fhix2 == -1)) {
     puts ("\n\nLipsesc fisierul de date sau fisierul index.");
     exit (1);
    }
    fsize = lseek (fhdf, 0L, SEEK_END);
```

SRCHIXF.C

```
    frecs = fsize / RSIZE;
    printf ("\n\nTESTFILE.DAT contine %ld bytes, %d inregistrari.",
         fsize, frecs);
    while (1) {
     printf ("\n\nIntroduceti cheia de cautare: ");
     gets (key);
     if (key[0] == 0) break;
     if (strlen (key) > KSIZE)
         printf ("\nAtentie! Cheia va fi trunchiata la %d caractere.",
KSIZE);

     inspected = 0;
     printf ("\nRezultatul cautarii secventiale:  ");
     if ((i = seqsearch (fhix1, key)) == -1)
         printf ("inregistrarea nu a fost gasita");
     else printf ("nr. inreg. este %d", i);
     printf ("\nNr. intrari inspectate: %d", inspected);

     inspected = 0;
     printf ("\nRezultatul cautarii binare:  ");
     if ((i = binsearch (fhix1, key, 0, frecs-1)) == -1)
         printf ("inregistrarea nu a fost gasita");
     else printf ("nr. inreg. este %d", i);
     printf ("\nNr. intrari inspectate: %d", inspected);

     inspected = 0;
     printf ("\nRezultatul cautarii in arbore binar:");
     if ((i = treesearch (fhix2, key)) == -1)
         printf ("inrgistrarea nu a fost gasita");
     else printf ("nr. inreg. este %d", i);
     printf ("\nNr. intrari inspectate: %d", inspected);

     if (i != -1) {
         lseek (fhdf, (long)(i * RSIZE), SEEK_SET);
         read (fhdf, rec, RSIZE);
         printf ("\nContinutul inregistrarii %2d:   %s", i, rec);
     }
    }
    close (fhdf);
    close (fhix1);
    close (fhix2);
}

/* CAUTARE SECVENTIALA */

int seqsearch (int fh, char *kptr) {
    int i;
    struct _index1 index1;

    lseek (fh, 0L, SEEK_SET);
```

SRCHIXF.C

```
    while (read (fh, (char *)&index1, sizeof(index1)) != 0) {
     inspected++;
     i = strncmp (kptr, index1.key, KSIZE);
     if (i == 0)
         return (index1.recno);
     else
         if (i < 0) break;
    }
    return (-1);
}

/* CAUTARE BINARA */

int binsearch (int fh, char *kptr, int left, int right) {
    int i, j;
    struct _index1 index1;

    if (left > right) return (-1);
    i = (left + right) / 2;
    inspected++;
    lseek (fh, (long)(i * sizeof(index1)), SEEK_SET);
    read (fh, (char *)&index1, sizeof(index1));
    j = strncmp (kptr, index1.key, KSIZE);
    if (j == 0) return (index1.recno);
    if (j < 0) binsearch (fh, kptr, left, i-1);
    else       binsearch (fh, kptr, i+1, right);
}

/* CAUTARE IN ARBORE BINAR */

int treesearch (int fh, char *kptr) {
    int i;
    int node = 0;
    struct _index2 index2;

    while (node != -1) {
     inspected++;
     lseek (fh, (long)(node * sizeof(index2)), SEEK_SET);
     read (fh, (char *)&index2, sizeof(index2));
     i = strncmp (kptr, index2.key, KSIZE);
     if (i == 0)
         node = index2.left;
     else
         node = index2.right;
    }
    return (-1);
}
```

Windows 3.0 in competitie cu Presentation Manager ?

Versiunea a 3-a a produsului **Windows**, lansata de curind de **Microsoft**, vine cu imbunatatiri semnificative ale vechilor versiuni. Cea mai evidenta schimbare este interfata utilizator, care a fost in intregime reproiectata astfel incit ea arata si lucreaza aproape la fel ca interfata **Presentation Manager** (PM) V1.2 a sistemului de operare **OS/2**. Aceasta va duce la cresterea usurintei in folosire a produsului, iar noua schema de gestiune a memoriei va permite rularea de aplicatii mai puternice. Vechiul executiv **MS-DOS** a fost inlocuit de Program Manager. In fereastra acestuia pot coexista mai multe grupuri de aplicatii, fiecare aparind in propria sa fereastra cu icoanele ce reprezinta programele apartinind grupului.

Exista facilitati de setare a grupurilor proprii si de instalare atit a programelor Windows cit si non-Windows. Programele Windows au propriile lor icoane iar cele non-Windows au o icoana generica DOS.

Alte facilitati ale interfetei Windows constau in :

- tabloul de control (control panel) a fost reproiectat pentru a include icoane care permit setarea culorii pentru orice parte a ecranului, schimbarea imprimantei, adaugare sau schimbare de fonte, etc.
- capacitatea de a selecta serverul de retea.
- ecranele de help si mesajele sint mai clare decit la versiunile vechi.

Utilitarul **Paint** prezent la celelalte versiuni a fost inlocuit cu o versiune mai sofisticata Paintbrush. Alaturi de utilitarele Windows obisnuite (*Write, Terminal, Clock, Cardfile, Notepad*) noua versiune vine cu o versiune imbunatatita a utilitarului **Calculator** precum si cu utilitarul **Recorder** care ofera posibilitatea de creare macrouri ce pot fi folosite in **Program Manager** sau in aplicatiile Windows.

A doua mare schimbare introdusa de versiunea a 3-a o constituie noua schema de gestiune a memoriei. Modul normal de lucru este cel protejat fiind suportat de chip-urile **Intel 80286, 80386, 80486**, permitind adresarea memoriei extinse (peste 1 M). Aceasta va permite elaborarea unor aplicatii mai sofisticate si mai rapide. In plus pentru **80386** noua versiune suporta memoria virtuala astfel incit exista facilitatea de memorare transparenta pe disc a partii (dintr-un fisier sau aplicatie) mai mari decit memoria interna disponibila.

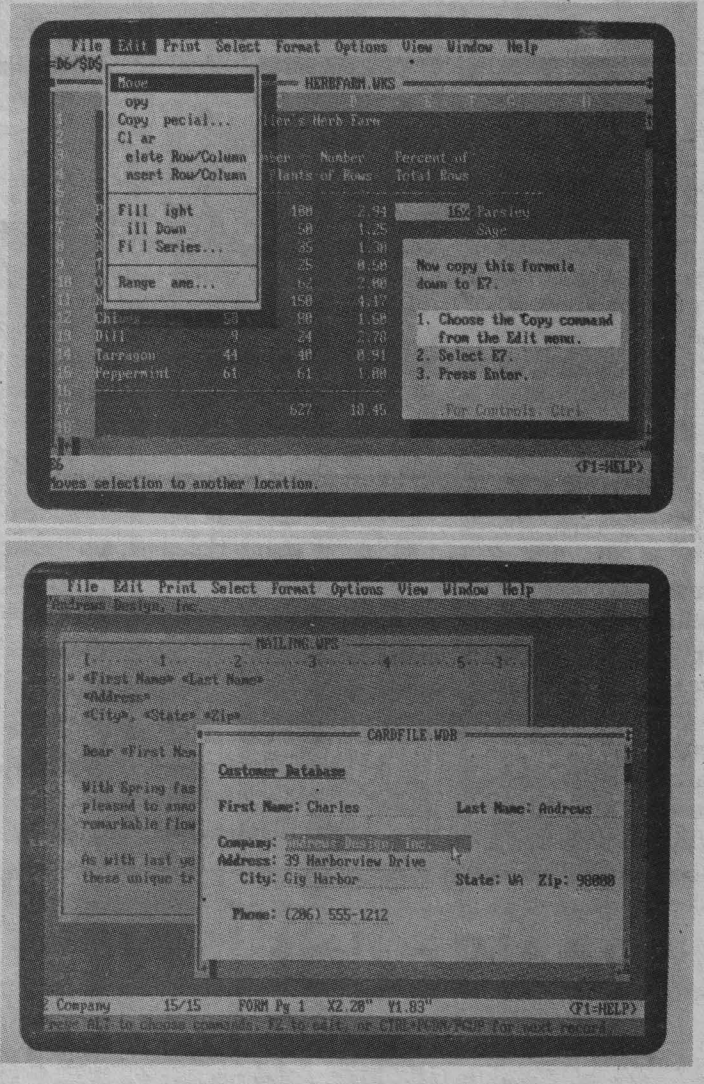
O alta facilitate a produsului **Windows 3.0** este posibilitatea de a lucra in trei moduri diferite : modul real care poate lucra pe masinile cu memoria de minimum 640K, modul standard care necesita un 80286 si cel putin 1M memorie si modul "enhanced" care lucreaza doar pe un 80386 cu cel putin 2M de memorie. Primul mod ofera compatibilitatea cu aplicatiile scise deja sub Windows, al doilea mod implementeaza modul protejat de gestiune a memoriei iar al treilea mod beneficiaza de avantajele modului pretejat, de memoria virtuala permitind multitasking-ul pe aplicatiile non- Windows. In concluzie **Windows 3.0** ofera aproape toate facilitatile promise initial de interfata PM a sistemului de operare **OS/2**. In plus firma **Microsoft** a anuntat ca in viitor aplicatiile Windows vor putea rula sub PM-ul noii versiuni a sistemului OS/2 pe care o va lansa. Semnul de intrebare pe care si-l pun toti utilizatorii Windows este modalitatea in care se va face aceasta tranzitie. Pe moment Microsoft propune urmatorul scenariu relativ la relatia viitoare Windows - PM : Windows va fi folosit de utilizatorii cu resurse de calcul limitate (low-end) pe cind PM de ceilalti (high-end).

O alta problema pe care Microsoft-ul trebuie s-o rezolve este cea a compatibitatii Windows 3.0 cu aplicatiile dezvoltate sub **Windows 2.0**. Introducerea schemei noi de gestiune a memoriei face ca cele mai multe dintre aplicatiile proiectate pentru **Windows 2.0** sa nu fie compatibile ca noua versiune decit daca lucreaza in modul real. Este cazul produselor **Excel 2.1** al Microsoft-ului, **Current** al IBM-ului, Ami Professional, Legend, **Crosstalk**. De asemenea shell-ul **IPX** al retelei **Novell Netware** nu ruleaza sub **Windows 3.0** iar **LAN Manager** 1.0 nu ruleaza decit cu un soft auxiliar.

Cresterea popularitatii mediului de operare **Windows** a dus la o reactie pe masura a numarului unu mondial (IBM), si anume anuntarea elaborarii unei versiuni a PM sub DOS. Aceasta versiune va oferii un multitasking limitat, **API**-ul (*Application Program Interface*) sau fiind o submultime a PM-ului sub OS/2. Aceasta atitudine se explica si prin reorientarea politicii **IBM** in ceea ce priveste impunerea sistemului **OS/2** acceptind momentan coexistenta mediilor **DOS** si **OS/2**, lasind la latitudinea utilizatorilor sa opteze.

Observam deci o oarecare concurenta intre **Microsoft** si **IBM** care nu este deloc benefica pentru utilizatori. Solutia ideala pentru acestia ar fi o unica versiune a **OS/2** elaborata de cei doi. Momentan insa este greu de presupus ca in mediile DOS Windows-ul sa fie concurat, chiar daca concurentul in discutie este **IBM**. Ca strategie pe termen lung insa, solutia **IBM** pare a fi mai viabilă.

Solutia definitiva insa relativa la interfata utilizator a vitoarelor medii de operare va fi data de soarta "razboiului interfetelor" ai carui principali combatanti sint : **Windows, PM, X-Windows** (conceputa la MIT), **OSF/Motif** a asociatiei **OSF (Open Software Foundation)**, **New Wave** a **Hewlett-Packard, Open Look** a **AT&T** (interfata standard pentru ultima versiune (a 4-a) a popularului sistem **UNIX SYSTEM V**) si nu in ultimul rind a noului sosit : interfata "revolutionara" **Nextstep** a firmei **NEXT** condusa de nu mai putin celebrul "copil minune al informaticii calculatoarelor personale" **Steve Jobs** (principalul realizator al primului calculator personal de succes, **Apple**).



Aplicatii in limbajul masina Z80 pe calculatoare compatibile Spectrum (I)

Georgescu Bogdan,
Iordanescu Bogdan

Argument:

Microprocesorul **Zilog Z80** este unul dintre cele mai raspindite microprocesoare de 8 biti, fiind insa depasit in momentul de fata de microprocesoarele pe 16/32 de biti. La noi in tara, varianta romaneasca MMN 80 CPU (3,5/6 MHz) beneficiaza de un foarte mare numar de instalari, mai ales in cadrul calculatoarelor personale compatibile **Sinclair Spectrum.**

S-a scris destul de mult despre utilizarea acestor calculatoare in limbajul **BASIC,** dar mai putin despre programarea in cod masina, respectiv limbaj de asamblare. Datorita performantelor destul de reduse ale programelor **BASIC** scrise sub interpretorul incorpoat, s-a creat impresia ca nu se poate obtine mare lucru de la aceste computere. In realitate, desi exista microprocesoare mult mai puternice, si cu **Z80**-ul se pot obtine rezultate notabile, cu putina perseverenta si lucrind la "nivelul 0" de programare, nivelul codului masina.

In serialul de fata, nu intentionam sa prezentam in amanuntime instructiunile si modul de lucru al microprocesorului, acestea necesitind o descriere laborioasa. Vom incerca sa prezentam ceea ce, in general, nu se scrie in carti si manuale, si anume cum se poate programa acest microprocesor in cadrul microcalculatoarelor **HC85, TIM-S, COBRA.** Datorita structurii hardware a acestor computere, exista restrictii in folosirea anumitor facilitati ale microprocesorului, aparind astfel instructiuni cu "probleme", adevarate capcane pentru incepatori.

Iata citeva dintre subiectele propuse:

- sistemul de intrari/iesiri (I/O);
- sistemul de intreruperi;
- folosirea setului de registri secundar;
- lucrul cu stiva, etc.

Toate aceste descrieri vor fi insotite de programe aplicative.

Sa incepem, deci, cu un subiect foarte interesant si "periculos" in acest timp daca nu este bine cunoscut:

Sistemul de intreruperi Z80

In legatura cu inreruperile, se pun la dispozitia programatorului urmatoarele instructiuni:

- EI si DI - valideaza, respectiv invalideaza intreruperile mascabile. Atit timp cit se lucreaza in interpretorul BASIC, intreruperile trebuiesc validate, altfel sistemul nu poate citi tastatura.IM0, IM1, IM2 -specifica modul de tratare a intreruperilor, Modul 0 nu poate fi folosit, deoarece instructura hardware a Spectrum-ului si a compatibilelor, nu exista un circuit periferic care sa genereze un cod de instructiune CALL sau RST. Modul 1 este folosit de catre interpretorul BASIC pentru citirea tastaturii: la fiecare 1/50 secunde, se lanseaza o cerere de intrerupere in urma careia se executa un RST #38, adresa din ROM unde se afla subrutina de deservire a intreruperii mascabile. Modul 2 este interesant din punctul de vedere al programatorului si va fi explicat pe larg in cele ce urmeaza.
- HALT - cauzeaza oprirea microprocesorului. In asteptarea unei intreruperi, Z80 executa

intern NOP-uri, pentru a asigura reimprospatarea memoriilor. Daca aceasta instructiune este executata si sistemul de intreruperi dezactivat, cum in calculatoarele studiate nu exista intreruperi nemascabile (NMI), iar cele mascabile (INT) nu vor fi acceptate, calculatorul se va bloca definitiv si singura cale de a-l reduce la o comportare normala este operatiunea de "reset".

- LDA,I si LDI,A - folosesc la accesarea registrului de intreruperi de 8 biti, I, folosit in cadrul modului 2 de tratare a intreruperilor.
- RETI si RETN - revenirea din subrutinele de tratare a intreruperilor mascabile si, respectiv, nemascabile. Nu are rost sa fie folosite pe Sinclair, fiind destinate sistemelor cu intreruperi complexe. Secventa normala de revenire pe aceste calculatoare este:

```
EI
RET
```

Din scurta enumerare facuta mai sus a rezultat ca programatorul are access la intreruperi pentru propriile sale programe, prin intermediul modului 2.

In acest mod, in momentul aparitiei unei intreruperi, microprocesorul formeaza o adresa in felul urmator: registrul I contine octetul mai semnificativ, iar octetul cel mai putin semnificativ este generat de un periferic. De la aceasta adresa din memorie, se citesc in ordine 2 octeti, cu care se formeaza o noua adresa , apoi se executa un CALL la aceasta adresa.

Complicat, nu-i asa? Cu toata aceasta incurcatura de octeti si adrese (numita tehnic identificare vectorizata) s-a realizat insa un lucru foarte important: interventia perifericului nu mai este asa de importanta si implicit "pretentiile" pe care le are microprocesorul scad.

Astfel sa notam cu "a" valoarea continuta in registrul I, care poate fi controlat prin software. Pentru o asemenea valoare "a", adresa care se formeaza prima data va lua valori intre a*256 si a*256+255, deci 256 de valori distincte, indiferent de octetul care i se furnizeaza microprocesorului.

Daca toti octetii din memorie intre aceste doua adrese limita vor contine o aceeasi valoare, sa zicem "b", microprocesorul va citi din aceasta tabela doi octeti identici, aflati unul dupa celalalt, si va crea cu ei o a doua adresa, cu valoarea: 256*b+b, deci aceeasi adresa , indiferent de unde se citesc octetii.

In sfirsit, la aceasta adresa se poate plasa o subrutina care sa deserveasca o intrerupere a programatorului.

O asemenea subrutina trebuie sa inceapa cu DI si sa se termine cu EI si RET, pentru ca in timpul executiei sa nu poata fi intrerupta de o a doua intrerupere (daca executia dureaza mai mult de 1/50 sec.). Ea trebuie sa contina, deasemenea, o secventa de salvare in stiva a tuturor registrilor care vor fi alterati, si de restabilire a lor la revenirea in programul principal. Daca se doreste ca intreruperea sa functioneze in paralel cu un program BASIC sub interpretor, nu trebuie citata instructiunea CALL #02BF, la aceasta adresa din ROM fiind subrutina propriu- zisa de citire a tastaturii, deoarece, la trecerea in modul 2 nu se va mai realiza automat RST #38 la fiecare intrerupere, ca in modul 1 in care lucreaza normal BASIC-ul.

Toate aceste probleme explicate teoretic aici, le vom rezolva practic printr-unul sau mai multe programe.

Dar aceasta, in numarul urmator.

A simula? Nimic mai simplu!

Inainte de a incepe prezentarea unui nou scenariu dorim sa multumim tuturor acelora care ne-au scris pe adresa acestei rubrici si sa-i asiguram de faptul ca observatiile precum si solutiile lor isi vor gasi locul in paginile revistei noastre.

In acest numar ne propunem dezvoltarea unei strategii de joc BASIC pornind de la un scenariu pe care sa-l intitulam:

COMOARA PIERDUTA

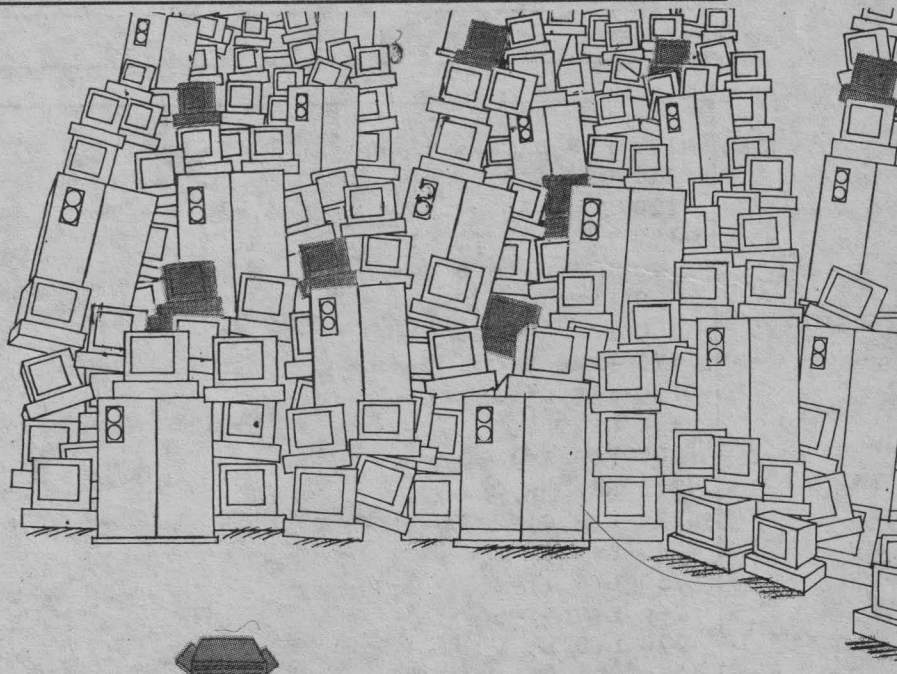
Sa ne inchipuim ca am ajuns pe o insula care ascunde o comoara. Dar, a gasi-o nu este nicidecum o problema chiar asa de simpla, pentru ca insula este inconjurata de rechini, iar aspectul ei nu este deloc prietenos; la fiecare pas putind intilni pesteri necunoscute, munti greu de escaladat, paduri dificil de strabatut, sau prapastii. Daca presupunem ca insula are forma unui caroiaj patratic, atunci obstacolele se pot configura in patratele determinate in timp ce deplasarea in interiorul insulei o putem simula in directiile nord (N), est (E), vest (V), sud (S), cite un patratel odata. Totusi, compasul tau nu este foarte exact. Sint doar 80 % sanse ca tu sa avansezi in directia propusa. Restul de 20 % reprezinta probabilitatea unei deplasari nedorite pe diagonala la stinga sau la dreapta. La fiecare deplasare vei fi informat despre tipul terenului in care te vei gasi.

Daca vei cadea in mare, vei fi plasat inapoi in patratul din care ai facut mutarea gresita. Acest lucru nu se va mai intimpla in situatia in care ai tulburat rechinii. Probabilitatea de a fi mincat de catre rechini la prima cadere in mare este de 20 %. A doua oara ea creste la 70 %. A treia oara va fi ...ultima oara.

Daca vei avea o harta a insulei, vei putea sa-ti determini aproximativ pozitia. De exemplu, daca te gasesti in padure si te deplasezi catre est cu doua patratele si descoperi ca te afli in munti, atunci mai mult ca sigur ca te afli in coltul de nord-est al insulei. Motivul pentru care nu-ti poti localiza cu exactitate propria pozitie este virarea la stinga sau la dreapta pe care deja am amintit-o. Pe masura cistigarii experientei, veti putea gasi comoara in mai putin de 15 mutari.

Exemplu de rulare:

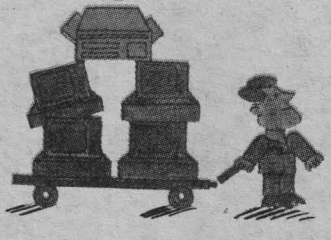
```
RUN

TE AFLI IN LOC DESCHIS.
DEPLASARE (NESV)? S
AI CAZUT IN OCEAN.
ESTI MINCAT DE CATRE RECHINI.
MAI JOCI D SAU N? D

TE AFLI IN LOC DESCHIS.
DEPLASARE (NESV)? S

TE AFLI IN PADURE.
DEPLASARE (NESV)? N

TE AFLI IN MUNTI.
DEPLASARE (NESV)? E
```



TE AFLI IN PADURE.
DEPLASARE (NESV)? S

TE AFLI IN LOC DESCHIS.
DEPLASARE (NESV)? E
AI GASIT COMOARA IN 9 MUTARI.
MAI JOCI D SAU N?

Programul COMOARA PIERDUTA



Variabile:

L (R, C) - Locatii;
S - Probabilitatea de a fi mincat de rechini;
R - Rindul;
C - Coloana in care va gasiti;
RT, CT - Variabile auxiliare;
T - Numarul de schimbari ale directiei;

Listingul programului BASIC este dat incepind cu pagina urmatoare.

Modificari posibile

Minore

- Probabilitatea primului atac al rechinilor – linia 120
- Marimea grilei – liniile 130, 280, 290, 560
- Numarul padurilor – liniile 160, 680
- Numarul muntiilor – liniile 200, 690
- Locatia comorii – linia 270
- Eroarea de deplasare – 480, 520

Majore

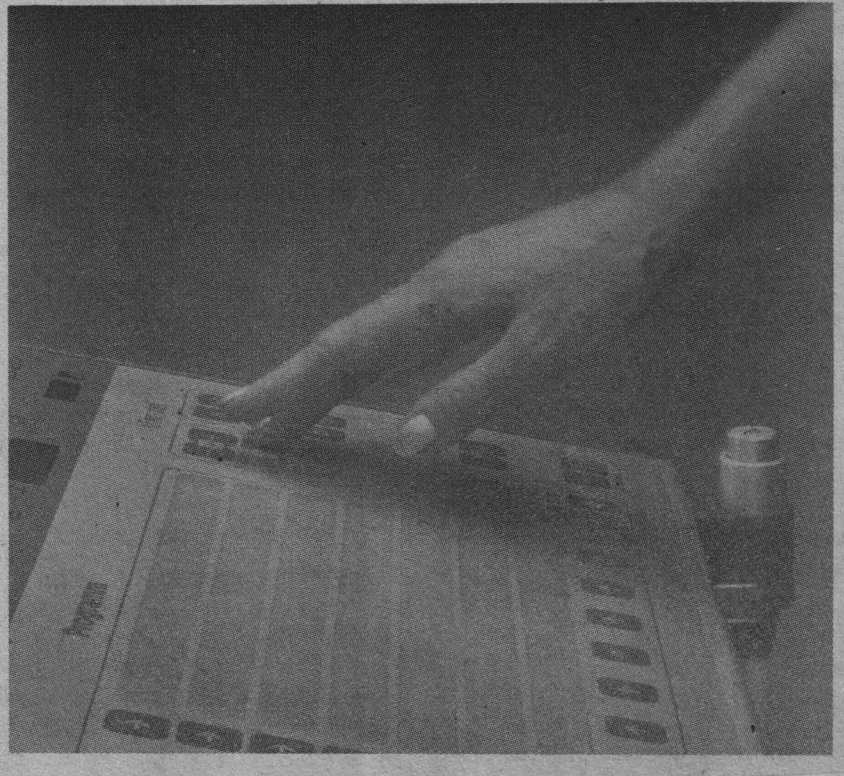
- Variati numarul si cantitatea comorii.
- Adaugati ca parametrii apa si hrana care va va mentine nivelul energiei.
- Vinati o comoara in miscare.
- Adaugati nisipuri miscatoare.
- Plasati aleator comoara inainte de fiecare vinatoare.

Listing

```
105 REM SETARE TEREN
110 DIM L(9,9)
120 S = .2
130 FOR I=1 TO 9: FOR J=1 TO 9
140 L(I,J) = 0
150 NEXT J,I
160 FOR I=1 TO 6
170 READ R,C
180 L(R,C)=1
190 NEXT I
200 FOR I=1 TO 6
210 READ R,C
220 L(R,C)=2 230 NEXT I
240 L(1,8)=3
250 L(6,1)=4
260 L(9,6)=5
270 L(5,5)=6
275 REM LOCATIA TA
280 R=INT(9*RND(1)+1)
290 C=INT(9*RND(1)+1)
300 IF SQR((R-5)^2+(C-5)^2)<2 THEN 280
305 REM START BUCLA PRINCIPALA
310 FOR T=1 TO 100
320 PRINT "TE AFLI ";
330 J=L(R,C)+1
340 ON J GO SUB 350,360,370,380,390,400: GOTO 410
350 PRINT "IN LOC DESCHIS.": RETURN
360 PRINT "IN PADURE.": RETURN
370 PRINT "IN MUNTI.": RETURN
380 PRINT "IN APROPIEREA UNEI PESTERI.": RETURN
390 PRINT "INTR-O BALTA.": RETURN
400 PRINT "LINGA UN STEJAR.": RETURN
410 INPUT "DEPLASARE (NESV)"; M$
420 RT=R: CT=C
430 IF M$="N" THEN R=R-1: GO SUB 480
440 IF M$="E" THEN C=C+1: GO SUB 520
450 IF M$="V" THEN C=C-1: GO SUB 520
460 IF M$="S" THEN R=R+1: GO SUB 480
470 GO TO 560
475 REM SUBRUTINA DE DEPLASARE
480 J=INT(10*RND(1)+1)
490 IF J>2 THEN RETURN
500 IF J=1 THEN C=C+1: RETURN
510 C=C-1: RETURN
520 J=INT(10*RND(1)+1)
530 IF J>2 THEN RETURN
540 IF J=1 THEN R=R+1: RETURN
550 R=R-1: RETURN
555 REM IN OCEAN
560 IF R<1 OR R>9 OR C<1 OR C>9 THEN 590
```

Listing

```
570 IF L(R,C)=6 THEN PRINT "AI GASIT COMOARA IN "; T; "MUTARI":
GO TO 650 580 NEXT T
590 PRINT "AI CAZUT IN OCEAN."
600 IF RND(1)<S THEN PRINT "ESTI MINCAT DE CATRE RECHINI.": GO TO
650 610 S=S+.5: R=RT: C=CT: IF S>1 THEN S=1
620 PRINT "PROBABILITATEA DE A FI MINCAT"
630 PRINT "DE CATRE RECHINI DATA VIITOARE ESTE"; S; "."
640 GO TO 580
650 INPUT "MAI JOCI D SAU N"; D$
660 IF D$="D" THEN RUN
670 END
680 DATA 2,3,3,5,3,9,4,1,7,2,8,8
690 DATA 1,2,3,7,5,2,6,8,8,3,8,6
```



NOUTATI SOFTWARE ...

New Face II este un generator de seturi de caractere, proiectat de firma MicroPress pentru a imbunatati mediul WordPerfect.

Folosind New Face II utilizatorul poate crea seturi de caractere prin intermediul unui editor incorporat.

Tipurile de caractere se pot genera (5 - 100 puncte) si transforma prin expandare, comprimare, inclinare, umbrire.

Programul ofera zece seturi de caractere predefinite. New face II include drivere pentru imprimantele: PostScript, HP LaserJet si DeskJet, Epson, Panasonic, NEC, Toshiba, Star, IBM si altele.

Cerinte: 640K RAM, WordPerfect 5.0 sau ulterior, DOS 3.0 sau ulterior.

Axum - produs al firmei TriMetrix, este destinat realizarii graficelor, in special in domeniul stiintei si ingineriei.

Programul foloseste un mediu pe baza de ferestre si meniuri si ofera diferite tipuri de grafice: 2D, 3D (incluzind axe logaritmice, coordonate polare), contururi regulate si neregulate si grafice de tip bara.

Programul include facilitati avansate de text, cum ar fi: inclinatie, suprascriere si subscriere, seturi de caractere diferite (stiintific, rus). Utilizatorii au acces la noua tipuri diferite de linii, pot modifica marimea graficului, pot suprapune mai multe grafice pe o pagina, insera si roti obiecte si cotrola marimea axelor.

Un editor de date permite importul, introducerea, editarea, transformarea, generarea si analiza datelor.

Seturile de date nu sint limitate ca marime. Exista posibilitatea analizarii mai multor seturi de date, constructiei graficelor corespunzatoare si vizualizarea acestora simultana.

Programul dispune si de un editor grafic interactiv care permite modificarea comentariilor, sagetilor si axelor. Pentru manipularea si transformarea datelor, programul ofera peste 100 functii si operatori care sint accesibile prin sistemul de meniuri sau prin limbajul de progamare incorporat.

Datele pot fi cuprinse in fisiere ASCII sau pot fi importate din Lotus 1-2-3, dBase.

Graficele pot fi exportate in formate: Hewlett-Packard HPGL, Lotus, PostScript si GEM.

Cerinte: 512K RAM, Hard Disk, adaptor grafic, DOS 2.0 sau ulterior (este recomandat coprocesor matematic).

QuickGraf este un editor pentru realizarea de grafice, produs de firma SumakEnterprises.

Programul ofera 32 tipuri de grafice care sint realizate pe baza datelor din foile proprii de lucru, din fisiere ASCII sau importatedin Lotus 1-2-3 sau dBase.

Daca dupa desenarea unui grafic, datele corespunzatoare se modifica, graficul va fi actualizat la urmatoarea incarcare (daca utilizatorul doreste acest lucru).

Programul permite utilizarea concomitenta a sase marimi de litere si a 15 culori pe ecran. QuickGraf are incorporate drivere pentru circa 100 tipuri de imprimante si ofera facilitati de tiparire: pe lungimea sau pe latimea foii de hirtie, stabilirea marimii graficului, controlul marginilor, setarea rezolutiei imprimantei.

Cerinte: 512K RAM, Hard Disk, DOS 2.0 sau ulterior.

Applause II - produs Ashton-Tate - este un editor grafic usor de utilizat, avind o interfata asemanatoare celei folosite de Windows (meniuri pop-up, pull-down).

Programul cuprinde trei elemente distincte: desenare, creare grafice, prezentare (fiecare cu setul proriu de instrumente).

Componentele graficelor nu pot fi manipulate direct, dar li se pot asigna atribute (care sint diferite in functie de tipul de grafic). Programul ofera tipuri noi de grafice si permite importul de fisiere de date (.GIF, .PCX, .TIF, .CGM).

Avantajul principal al programului este dat de efectele speciale de culoare. Paletele de culori incorporate sint stralucitoare si oricare paleta primara are o lista de nuante corespondenta. Pe baza acestei facilitati pot fi create imagini cu multe nuante, prin parcurgerea unui pas care construieste fisiere batch (.GX2) in care vor fi definite efectele de tranzitie ale culorilor.

Cerinte: 512K RAM, DOS 2.1 sau ulterior.

Alpha Four - produs al firmei AlphaSoftware Corp. - este destinat nespecialistilor care isi pot proiecta propriile baze de date, fara a fi necesare cunostinte de programare.

Produsul cuprinde un editor de rapoarte. Realizeaza validarea datelor. Accepta fisiere dBase, fara a fi necesare modificari.

Pizzas Plus este un utilitar produs de firma Applications Techniques, Inc., care realizeaza capturi de imagini care apoi pot fi transferate in editoare de publicatii (Ventura Publisher, AldusPageMaker, Microsoft Word).

Programul face conversia din mod text in mood grafic si din imagini color in imagini alb-negru (256 tonuri de gri), pastrind claritatea detaliilor. Este compatibil cu aprox. 200 tipuri de imprimante. Permite alegerea unitatii de masura dorite de utilizator (inch, milimetru, puncte), selectarea marimii de tiparire si pozitionarea cu precizie 0.01".

Type Director, produs al firmei AGFA Compugraphic, este un editor de texte usor de utilizat si rapid.

Ofera o gama larga de tipuri de caractere si permite marirea sau micsorarea fiecarei litere (gama 4 - 200 puncte). Forma caracterelor afisate pe ecran este aceeasi cu care ele vor fi tiparite. Produsul accepta si seturi de caractere create de utilizator, cu conditia ca acestea sa fie compatibile cu imprimanta laser Hewlett-Packard.

Type Director este usor de utilizat 8prin intermediul unui sistem de meniuri si avind un HELP incorporat).

The Perfect Addition - produs al firmei Applause Software, ofera un sistem de meniuri (tip pop-up si pull-down) pentru utilizatorii editorului WordPerfect (4.1 - 5.0).

Accepta adaugarea unor macro-uri create de utilizator, care vor fi apoi executate folosind meniurile sau cheile functionale.

Software Carousel 3.0 este un produs SoftLogic Solutions, Inc., care permite trecerea de la un program la altul, fara a fi necesare: salvarea fisierelor, iesirea dintr-un program, incarcarea altui program.

La pornirea sistemului se incarca toate programele cu care se va lucra, iar Software Carousel realizeaza activarea programului dorit la un moment dat (specificat prin apasarea unei taste initial asignate programului respectiv).

Software Carousel permite lucrul cu: Lotus 1-2-3, WordPerfect, dBase, SideKick si orice alt program.

Actualitati Antivirus pentru Macintosh

Aparitia in ultima vreme a virusului **WDEF** (considerat a fi cel mai inteligent) a dus la intensificarea eforturilor de elaborare a pachetelor soft de protectie impotriva acestuia. Cele doua firme leader in domeniul produselor antivirus pentru calculatoarele personale de tip **Macintosh** (*Symantec* si *Microcom*) si-au actualizat deja propriile produse pentru a asigura protectia impotriva virusului **WDEF**. Este vorba de produsele: **SAM** (*Symantec Antivirus for Macintosh*) al firmei **Symantec** si **Virex** al firmei **Microcom**. Particularitatea virusului **WDEF** in comparatie cu virusul standard de tip **nVir** (care se replica (multiplica) doar prin lansarea unui program de pe floppy) consta in aceea ca simplul fapt de deschidere a unei ferestre sau de mutare a ei cauzeaza replicarea (multiplicarea) virusului. De asemenea acest virus reuseste sa ocoleasca toate schemele de protectie antivirus cunoscute pina in prezent. El se ataseaza de resursele de definire fereastra din fisierul desktop.

Firma **Microcom** a declarat ca este capabila sa-si actualizeze produsul **Virex** pentru orice clasa noua de virus, oferind aceste actualizari contra doar a 75$.

Unii producatori de soft au inceput sa incorporeze in produsele lor, scheme de protectie antivirus prin care se verifica continuu rularea corecta a programului. In prezent doar **Mac Write 2** implementeaza o astfel de schema. Totusi o astfel de metoda nu este viabila pentru protectia antivirus a programelor ce n-au implementat schema sus-mentionata si nici pentru softul de sistem.

Virusafe

Acesta este numele celui nai recent soft antivirus pus la dipozitie pentru sistemul de operare **MS-DOS**. **Virusafe** este un program rezident care verifica daca un virus este deja prezent si activ, intercepteaza manipularile suspecte, si verifica la cerere programele (marime, structura, *checksums*). Dintre virusii care pot fi detectati si anihilati de catre **Virusafe** amintim: **Pingpong, Alabama, Brain** ... .

Noua generatie de produse software folosite in matematica

Noile facilitati oferite de produsele software **Derive**, **Macsyma**, **Maple** si **Mathematica** se refera la operarea cu simboluri, adica la folosirea variabilelor abstracte x si y familiare matematicienilor.

Aceste programe sint capabile sa efectueze calcule algebrice complexe (factorizari, simplificarea formulelor prin folosirea de identitati trigonometrice).

Calculul simbolic ofera posibilitatea calcularii simbolice a derivatelor si integralelor (in forma functionala). Programele prezentate realizeaza calcule cu numere reale cu o precizie foarte buna.

Programele mai vechi (**MathCAD**, **Gauss**) alocau un spatiu fix de memorie pentru fiecare obiect matematic. Daca un intreg devine prea mare, aceste programe afiseaza un mesaj de eroare si modifica intregul intr-un real cu precizia necesara. Noile pachete software maresc memoria alocata intregului atit cit este nevoie.

In mod asemanator, alte programe aloca 4, 8 sau 10 bytes pentru numere reale, limitind precizia la 16 zecimale. Aceste noi programe pot lucra cu orice nivel de precizie dorit de utilizator. In mod normal, pentru rezultate finale nu sint necesare mai mult de 16 zecimale, dar in cazul unor calcule numerice complexe erorile de rotunjire facute in pasii intermediari pot modifica substantial rezultatul.

Alte facilitati oferite sint calculul numeric, realizarea de grafice, posibilitatea programarii (numai **Derive** nu are un limbaj de programare asociat).

Toate programele, in afara de **Derive**, au un set extern de rutine care sint incarcate automat atunci cind este cazul si pot realiza ca iesire functii C, Fortran (dar necesita PC-uri puternice : 80386, cîtiva megabytes de RAM, spatiu de lucru pe disc 5 - 20MB).

Derive - produs al firmei Soft Warehouse Inc. ofera posibilitatea de lucru cu polinoame, functii trigonometrice si hiperbolice, factoriale, integrale si derivate simbolice (nu include insa un limbaj de programare, functii Bessel, zeta).

Interfata cu utilizatorul este usor de folosit, iar cerintele hardware sint minime : PC - XT, 512 K RAM, nu necesita coprocesor matematic.

Macsyma - produs al firmei Macsyma Division of Symbolics Inc. este foarte puternic in realizarea de calcule simbolice, dar are posibilitati grafice limitate, procesare numerica greoaie si o interfata "neprietenoasa" (lucru in linie de comanda). Programul ofera cea mai extinsa posibilitate de HELP (sintaxa si exemple).

Cerinte : PC/80386, 4 MB RAM, spatiu de lucru pe disc 40 MB,DOS 3.0 sau ulterior, recomandat mouse.

Maple - produs Waterloo Maple Software, foloseste editarea in linie de comanda, dar are posibilitatea reapelului celor mai recente 100 de comenzi. Programul cuprinde un HELP incorporat (pentru sintaxa).

Produsul ofera cel mai bogat set de rutine si este rapid in procesarea numerica.

Cerinte : PC/80386, 2 MB RAM, 10 MB disc, coprocesor matematic 80387, DOS 3.3 sau ulterior.

Mathematica - produs Wolfram Research Inc. poseda un foarte puternic limbaj de programare si ofera posibilitati extinse de grafica. Interfata cu utilizatorul este usor de utilizat. Programul cuprinde un HELP incorporat.

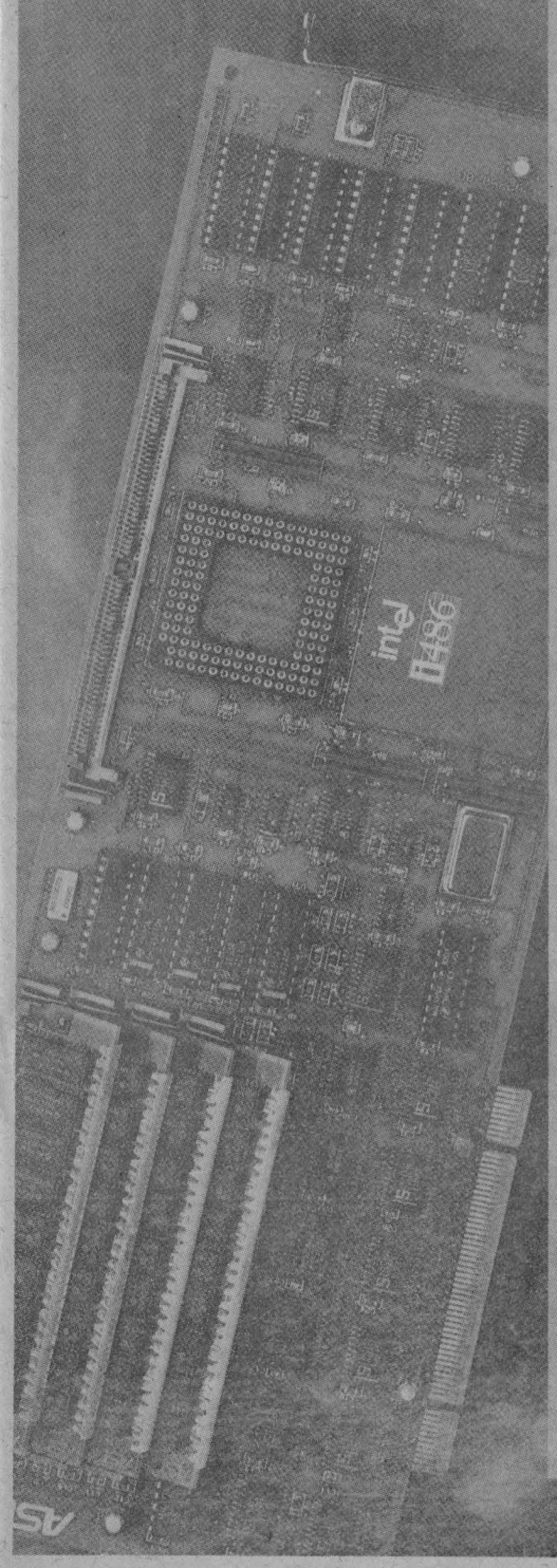
Cerinte : 2 MB RAM, spatiu liber pe disc 8.5 MB, DOS 3.0 sau ulterior.

"Shouldn't you be talking to Rayma?"

Finally, IBM Compatible Computers At Low Prices!

We at Rayma Technologies believe that a computer should not cost an arm or leg but should comfortably fit into anyone's budget, especially a student's. And there's good reason - we at Rayma were once students too. Statistics prove that a student's greatest tool is a personal computer. The question is no longer "Will You Buy A Computer, But Who Will You Buy It From?"

We Left Out The Obsolescence The Others Built In.



Computers That Grow With You.

Until now, advances in PC technology have left you with a single, all important decision. Move up and benefit from increased performance. Or, stay put and try to capitalize on the investments you've already made.

AST is announcing the end to this difficult decision. Our Cupid-32™ architecture allows you to upgrade your computer as your needs grow. All the way to i486™ technology.

That's right: Our Premium 386SX/16 and Premium 386 (33 and 25 MHz) computers incorporate a natural upgrade path. A path that doesn't force you to sacrifice your investment in already-owned systems and peripherals.

With Cupid-32, you can add the performance you need to add more users to a multi-user system. Add more nodes to a network file server. Or, add more power to a standalone situation, like CAD/CAM. All it takes is a simple, single board swap with one of our FASTboards,™ when you are ready for more power.

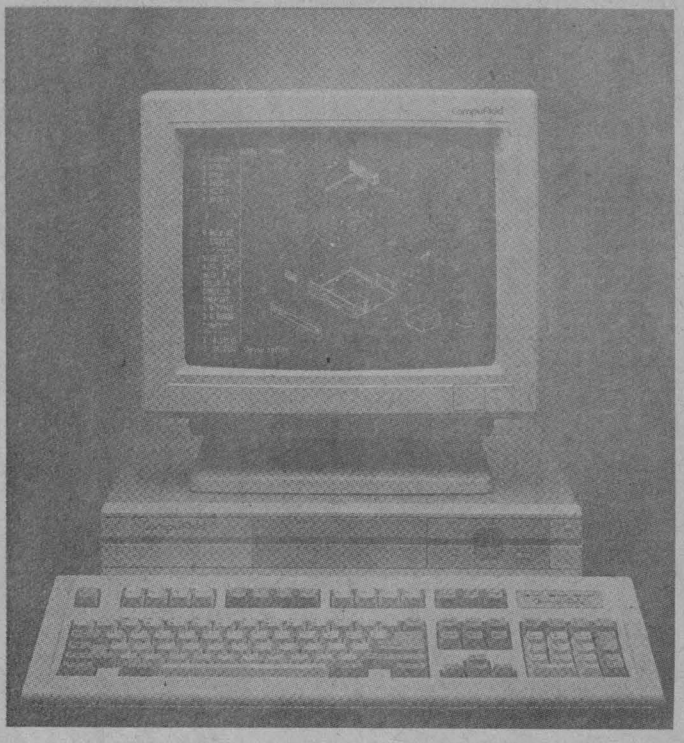
So, in place of the difficult decision, we have a simple one for you. Why buy a computer from anyone else, when you can own an AST Premium Computer? After all, they're the only computers that grow with you.

For more information call **1-800-876-4AST**. Or, use our Bulletin Board Service Number (714) 852-1872.

AST®
RESEARCH INC.

The Power Of Choice.

AST is a leading supplier to government agencies. General Service Contract no. GSOOK89AGS6418. AST markets products worldwide — in Europe and the Middle East call: 44 1 568 4350; in Japan call: 81 3 818 0710; in the Far East call our Hong Kong office at: 852 5 806 4333; in Canada call: 416-826-7514; in Australia call: 02 906 2200. AST, AST logo and AST Premium registered and Cupid-32 and FASTboard trademarks AST Research, Inc. All other brand or product names are trademarks or registered trademarks of their respective companies.



ANUNT IMPORTANT

Pentru Centre de Calcul, Centre de perfectionare a specialistilor, reprezentante ale firmelor straine, firme particulare.

Incepind cu nr.5 al revistei noastre, initiem o rubrica de publicitate in domeniul informaticii la care va asteptam sa participati cu reclame pentru produsele dumneavoastra.

Va rugam sa ne trimiteti text cuprinzind descrierea produselor program, serviciile pe care le oferiti, precum si descrierea dispozitivelor pe care le puneti la dispozitie. De asemenea reallizam reclame privind cursuri de initiere sau pentru avansati, meditatii sau consulting.

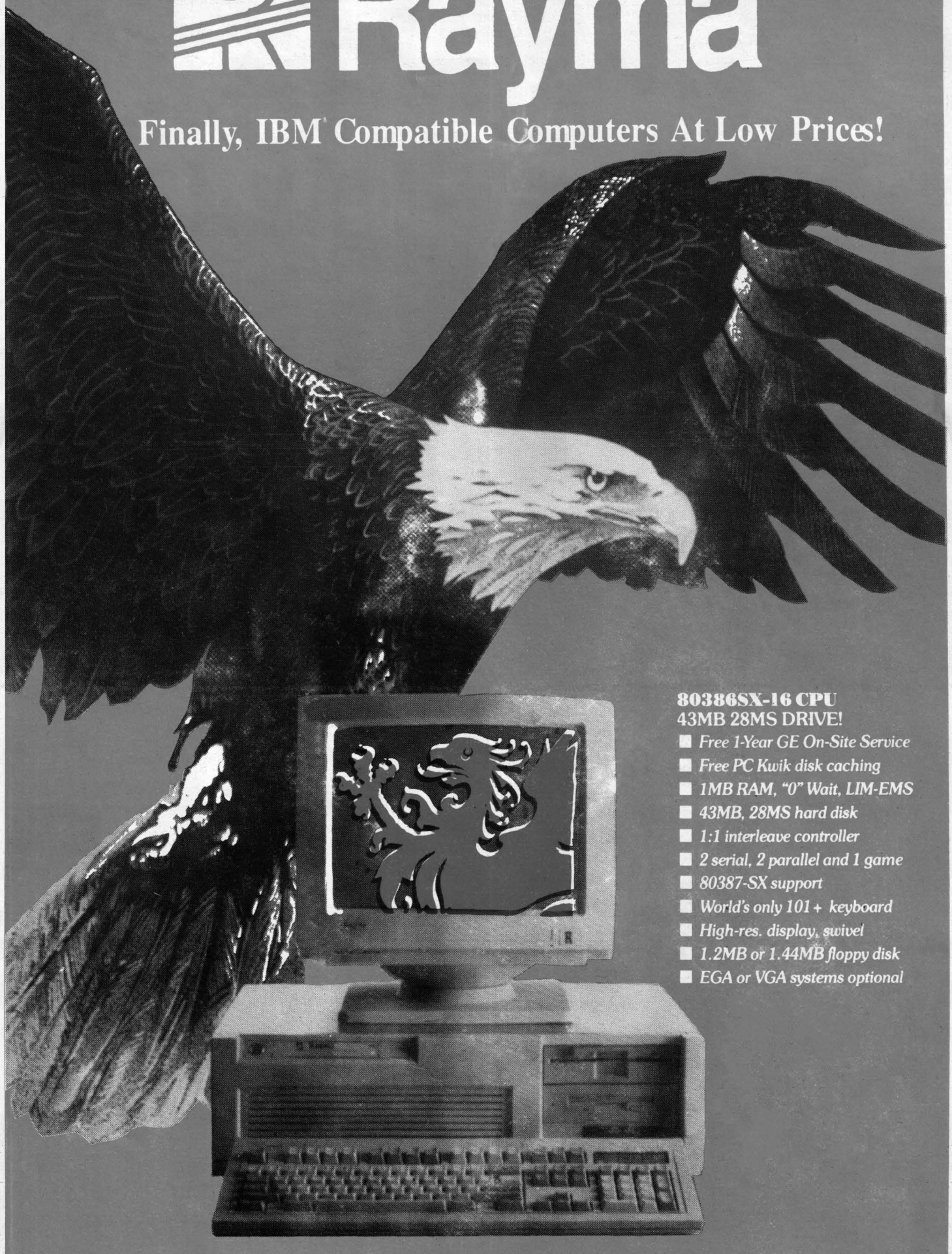
Dispunem de o echipa de graficieni care vor satisface cele mai exigente gusturi!

Materialele vor fi trimise pe adresa Societatea ADISAN, Str. Ion Mincu nr.11, Sect.1, Bucuresti.

Telefon 15.81.65

Rayma

Finally, IBM® Compatible Computers At Low Prices!



80386SX-16 CPU
43MB 28MS DRIVE!

- *Free 1-Year GE On-Site Service*
- *Free PC Kwik disk caching*
- *1MB RAM, "0" Wait, LIM-EMS*
- *43MB, 28MS hard disk*
- *1:1 interleave controller*
- *2 serial, 2 parallel and 1 game*
- *80387-SX support*
- *World's only 101+ keyboard*
- *High-res. display, swivel*
- *1.2MB or 1.44MB floppy disk*
- *EGA or VGA systems optional*

Preturi avantajoase in dolari sau lei

GRIFON

Va ofera

Cea mai inalta calitate la cel mai scazut pret

Computerele Rayma

Special Offer : Rayma Turbo - XT

FEATURES :

640Kb RAM
360 Kb Floppy Drive
20 Mb Hard Drive
Floppy/Hard Disk Controller Card
Clock/Calender
Serial & Parallel Ports
Monochrome Monitor
Monographics Card
24 Hour Burn-In
18 Month Warranty

Special Offer : Rayma 286 - 12 MHz

FEATURES :

1 Mb RAM
1.2 Mb Floppy Drive
20 Mb Hard Drive
Floppy/Hard Disk Controller Card
Clock/Calender
Serial & Parallel Ports
Monochrome Monitor
Monographics Card
24 Hour Burn-In
18 Month Warranty

tel:
79 97 67
11 19 20

Lei 30