MANUAL BLAST

ALPHA Ltd. @ 1992

CUPRINS

B L A S T C O M P I L E R V 3.0

traducere Miodrag Puterity
adaptare Aurel Gontean

COPYRIGHT

DISCLAIMER

Cu toate ca acest produs a fost testat cu atentie, nu se emite nici o pretentie in legatura cu aderarea BLAST-ului la anumite specificatii particulare sau posibilitatea folosirii sale intr-un scop anume.

1. INTRODUCERE

BLAST este primul compilator perfect compatibil BASIC cu posibilitati de optimizare aparut pentru calculatoarele Sinclair. Scopul sau primordial este de a obtine viteza maxima de executie pentru programe scrise in BASIC-ul Spectrum-ului fara a produce coduri obiect de dimensiuni prohibitive. BLAST-ul poate creste viteza BASIC-ului de pina la 40 de ori.

Operarea cu compilatorul este extrem de simpla. Sint foarte putine comenzi noi care trebuie invatate, iar nivelul de compatibilitate cu interpretorul BASIC atit de inalt incit chiar si extensii ale BASIC-ului scrise in cod masina vor fi compilate. Utilizatorii care scriu specific pentru BLAST pot beneficia de o gama larga de extensii prevazute in compilator, precum si de puternicul toolkit livrat impreuna cu compilatorul.

BLAST a fost dezvoltat de catre aceeasi companie care a

produs PETSPEED, compilatorul BASIC pentru Comodore 64 si numeroase alte compilatoare ce ruleaza pe diverse microcomputere. In afara BLAST-ului compania produce si OXFORD PASCAL, o implementare completa a popularului limbaj Pascal pentru diverse microcomputere incluzind Spectrum-ul.

2. DESPRE ACEST MANUAL

Oricine are un program BASIC caruia trebuie sa i se mareasca viteza, poate beneficia de BLAST. Folosirea BLAST-ului insusi, nu necesita nici o cunostinta despre BASIC. Deoarece in principal utilizatorii vor fi programatori in BASIC, acest manual li se adreseaza in mod special.

Manualul BLAST-ului nu incearca sa-i invete pe incepatori programarea in BASIC, ci doar discuta despre programare cind aceasta este necesar pentru a face referiri la compilator.

BLAST-ul este atit de simplu de utilizat incit utilizatorul este tentat sa treaca prea rapid prin manual sau chiar sa-l ignore complet. Noi descurajam in mod hotarit aceasta tendinta. Utilizatorii sint sfatuiti sa citeasca manualul atent inainte de a incepe orice lucru mai serios cu compilatorul.

3. CE ESTE UN COMPILATOR ?

In aceast capitol vom explica notiuni de baza despre compilatoare si citeva elemente de terminologie. O intelegere a conceptelor de baza prezentate aici, altfel neesentiala, va spori mult abilitatea in exploatarea BLAST-ului. Scurta sectiune de terminologie trebuie citita si inteleasa.

Un program BASIC este pur si simplu o portiune de text in care specificam actiunile pe care vrem sa le intreprinda computerul cind programul ruleaza. Microprocesorul Z80 cu care este echipat Spectrum-ul intelége doar un limbaj numit cod masina. Pentru Z80, un program BASIC este o intreaga babilonie. Pentru a rula un program avem nevoie de software care sa inteleaga BASIC-ul si sa-l traduca intr-o forma pe care Z80 sa o poata intelege. Exista doua tipuri de programe de traducere a BASIC-ului, interpretorul BASIC si compilatorul BASIC.

3.1 INTERPRETOARE

Un interpretor BASIC, cum este cel furnizat in ROM ca parte a Spectrum-ului, citeste fiecare instructiune (declaratie) a programului si facind aceasta, intreprinde actiunile specificate. Interpretoarele sint extrem de utile pentru dezvoltarea programelor, deoarece se interpreteaza textul BASIC propriu-zis. Se poate edita un program, rula si apoi reedita rapid si fara multa bataie de cap. Dezavantajul unui interpretor este acela ca ruleaza incet deoarece aproape tot timpul interpretorul incearca sa inteleaga BASIC-ul in loc sa

intreprinda actiunile specificate.

3.2 COMPILATOARE

Spre deosebire de interpretoare, un compilator traduce intreg programul in ceea ce masina poate intelege, intr-o singura operatie, numita compilare. Cind operatia este terminata, avem un bloc de cod masina care este versiunea tradusa a textului BASIC. Compilatoarele sint mult mai putin utile in stadiul de dezvoltare a unui program, fata de interpretoare, deoarece o schimbare cit de mica in textul BASIC necesita o recompilare completa a programului. Cu toate acestea, odata ce un program a fost compilat, va rula la o viteza mult mai mare.

4. TERMINOLOGIE

In continuare se vor adopta urmatorii termeni:

COMPILE TIME (compilare) - perioada de timp in care BLAST-ul compileaza un program

RUN TIME (rulare) - perioda de timp in care programul compilat este executat

SOURCE FILE (fisier sursa) - fisierul de intrare intr-un compilator, in acest caz text BASIC, uneori numit si cod sursa

OBJECT FILE (fisier obiect) - iesirea dintr-un compilator, in acest caz traducerea in cod masina a unui text BASIC, uneori numit si cod obiect

MACHINE CODE (cod masina) - limbajul intern inteles de microprocesorul Z80

P-CODE (cod "p") - o reprezentare intermediara a unui program, intre BASIC si cod masina, fiind o alternativa a codului masina care necesita mult mai putin spatiu dar si un mic interpretor la rulare, si este putin mai lenta decit codul masina dar mult mai rapid decit BASIC-ul interpretat ; BLAST-ul poate compila in p-code, in cod masina sau intr-un amestec al acestora

COMPILER DIRECTIVE (directive ale compilatorului) - un mesaj adresat compilatorului care se adauga textului unui fisier sursa si care afecteaza modul in care se comporta calculatorul; BLAST-ul prezinta o suma de directive compilator foarte utile care se adauga programului sub forma unor instructiuni REM speciale.

5. SA INCEPEM

Acest capitol explica cum se utilizeaza BLAST-ul in modul cel mai simplu. Vom lua un program BASIC deja incarcat in calculator si il vom compila direct in RAM fara acces la banda sau la microdrive. In acest mod (numit mod RAM to RAM) sintem

limitati la programe relativ scurte deoarece atit compilatorul cit si programele sursa si obiect trebuie sa coexiste in memorie.

1) Incarcati BLAST-ul dupa cum urmeaza:

LOAD "BLAST" <ENTER>

BLAST-ul va autorula si va "semna" cu mesajul:

BLAST (c) OCSS 1983 xxxx BYTES FREE

In acest punct BLAST-ul va face o verificare de protectie pentru a stabili daca sinteti un utilizator autorizat al acestui produs software. Speram ca nu veti gasi procedura prea obositoare. Secventa de protectie apare doar cind BLAST-ul se incarca pentru prima data. Odata verificarea facuta, BLAST-ul va va permite sa compilati oricite programe doriti, fara alte complicatii.

Intre copertile acestui manual sau pe o foaie de hirtie separata veti gasi o matrice de patratele colorate. Fiecare patratel poate fi identificat printr-o simpla referinta la grila. De exemplu pentru a gasi patratelul E-13, se identifica coloana E si rindul 13 iar patratelul E-13 este cel in care se intersecteaza coloana E si rindul 13 (vezi figura).

```
. . . . . .
13 . . . . * . .
. . . . . .
. . . . . .
. . . . . .
1 . . . . . .
  A B C D E F .
```

Verificarea de protectie este foarte simpla. Trebuie doar sa identificati corect culorile respective. BLAST-ul va da instructiuni dupa cum urmeaza:

ENTER THE COLOUR IN SQUARE x-xx (W,Y,G,R) ?

unde x-xx este o referinta la grila. Cind ati gasit patratelul introduceti una din literele W,Y,G sau R dupa cum patratelul este alb,galben,verde sau rosu si apasati tasta ENTER. Cind ati raspuns corect la patru astfel de intrebari, verificarea de protectie este terminata.

BLAST-ul este acum complet initializat si gata sa compileze programe. De acum inainte, pina cind se tasteaza NEW sau se debranseaza alimentarea calculatorului, Spectrum-ul va raspunde la un set de comenzi aditionale folosite pentru a comunica cu BLAST-ul. Comenzile BLAST-ului sint precedate de un asterisc (*) pentru a le distinge de cele normale ale Spectrum-ului.

Dorim sa folosim BLAST-ul pentru a compila un program deja in memorie si sa avem codul obiect rezultat tot in memorie. In acest mod BLAST-ul se comporta in mod implicit.

Se incarca (sau se tasteaza) un program BASIC, nu mai mare de 5K si se tasteaza:

*C

pentru a-l compila.

In acest moment compilatorul va putea decide daca are nevoie de memoria ecran a Spectrum-ului ca spatiu de lucru. Nu va alarmati pentru orice babilonie care ar putea sa apara pe ecran; este pur si simplu BLAST-ul folosind optim memoria disponibila. Presupunind ca nu sint probleme, dupa un minut sau doua, controlul va va fi redat cu mesajul:

(0) WARNINGS (0) ERRORS

Pentru a rula versiunea compilata a programului, tastati:

*R

Daca BLAST-ul ramine fara spatiu de lucru la compilare, va va intreba daca poate sa stearga din memorie programul sursa. Daca nu doriti ca acest lucru sa se intimple, tastati N si veti reveni la interpretor. In caz contrar, tastati Y si compilarea va contina. BLAST-ul nu va sterge niciodata un program fara permisiune.

Atita timp cit BLAST-ul este in memorie, veti putea edita codul sursa, rula sub interpretor sau compila si rula programul BLAST-at ori de cite ori doriti. Lucrind cu BLAST-ul veti considera ca uneori trebuie sters programul din memorie fara a sterge BLAST-ul. Pentru aceasta in loc de NEW (care ar sterge toata memoria, inclusiv BLAST-ul), folositi directiva *N. Aceasta sterge orice text BASIC fara a afecta compilatorul.

N.B. Desi BLAST-ul poate face fata la codul masina scris de utilizator si chemat dintr-un program BASIC, acest lucru nu e posibil in modul RAM to RAM (vezi capitolul BLAST SI CODUL MASINA AL UTILIZATORULUI).

6. SALVAREA PROGRAMELOR BLASTATE

Pentru salvarea codului obiect se da directiva *S. BLAST-ul va intreba daca doriti o salvare pe banda sau pe microdrive iar apoi va va intreba numele de fisier sub care doriti sa salvati programul BLAST-at. Bineinteles, se poate utiliza orice nume de fisier legal dar o practica utila ar fi sa folositi numele original al programului BASIC cu un amendament subscris. Fisierul astfel scris pe banda sau pe microdrive contine codul obiect al programului d-voastra impreuna cu sistemul run-time al BLAST-ului. El nu contine nici o parte a BLAST-ului propriu zis.

Tastati numele de fisier si apoi ENTER.

Pentru a verifica daca codul a fost salvat corect, tastati NEW pentru a "goli" calculatorul. Aveti posibilitatea de a incarca codul obiect salvat la fel ca si un program BASIC obisnuit. Pentru a-l executa tastati RUN. BLAST-ul salveaza intotdeauna un cod obiect in asa fel incit se incarca in zona de memorie rezervata fisierului text BASIC. Compilatorul face acest

lucru pentru ca programele BLAST-ate sa poata fi incarcate si rulate la fel ca programele BASIC. Deoarece computerul a fost golit, va fi necesar sa se incarce din nou BLAST-ul pentru a putea continua.

7. BLAST-AREA PROGRAMELOR LUNGI

Pina acum am vazut cum BLAST-ul compileaza din memorie in memorie. Dupa cum am explicat inainte, acest lucru este posibil daca programul compilat este scurt. Pentru a rezolva aceasta problema, BLAST-ul este prevazut cu optiuni de citire a codului sursa de pe banda sau de pe microdrive, si scriere a codului obiect rezultat pe oricare din aceste periferice. In continuare vom explica cum se foloseste BLAST-ul in diverse moduri de intrare/iesire.

Selectarea dispozitivelor de intrare/iesire

Pentru a selecta dispozitivul de la care BLAST-ul va citi fisierul sursa se foloseste optiunea INPUT tastind:

*I

si raspunzind intrebarii:

ACCEPT INPUT FROM: RAM, TAPE, MICRODRIVE

cu R, T sau M.

Pentru a selecta dispozitivul la care BLAST-ul va scrie codul obiect, se tasteaza:

*O

si se procedeaza ca mai sus.

Ori de cite ori e selectata optiunea de compilare cu:

*C

BLAST-ul va cere informatii corespunzatoare optiunilor de intrare/iesire alese. Spre exemplu, daca a fost selectat un microdrive, BLAST-ul va cere numarul microdrive-ului si numele de fisier. Daca s-a selectat banda, BLAST-ul va cere doar numele de fisier.

Diferitele combinatii ale dispozitivelor de intrare/iesire, lasa BLAST-ul cu mai mult sau mai putin spatiu de lucru pe perioada compilarii. Daca programul de compilat depaseste aproximativ 5K, probabil va fi necesar ca citirea sa se faca de pe banda sau microdrive, in loc de memorie. Daca programul sursa e foarte lung, (mai mult de 8K) va fi necesara si selectarea iesirii pe banda sau pe microdrive. Insa oricare ar fi dispozitivele selectate, BLAST-ul va va conduce pas cu pas pe parcursul procesului de compilare.

Daca dispozitivul de iesire este banda sau microdrive-ul, compilarea se va sfirsi cu codul obiect inscris pe dispozitivul respectiv. Evident, programul obiect va trebui incarcat de pe

suportul sau pentru a putea fi rulat. Retineti ca BLAST-ul insusi consuma o mare cantitate din memoria Spectrum-ului, insa acest lucru nu va impiedica sa compilati programe lungi, ci doar va determina sa inlaturati BLAST-ul din memorie atunci cind rulati astfel de programe. Inlaturarea BLAST-ului din memorie se face cu comanda:

*Q

BLAST-area pe microdrive

Este cel mai bun mod de a compila programe lungi. Daca aveti astfel de programe de compilat si nu aveti microdrive va recomandam sa va procurati de urgenta unul. BLAST-ul poate fi copiat pe microdrive cu comanda:

*B

BLAST-area pe banda

Daca nu aveti microdrive si doriti sa compilati programe lungi, veti proceda in felul urmator: Datorita naturii limitate a benzii ca dispozitiv I/O, programul de compilat trebuie intii salvat pe banda intr-un format special. Facilitatile necesare sint incluse in TOOLKIT-ul furnizat pe fata opusa a casetei BLAST-ului. Pentru mai multe detalii va trebui sa consultati capitolul dedicat TOOLKIT-ului. Odata ce programul sursa a fost salvat pe banda intr-un format adecvat, BLAST-ul poate fi incarcat in memorie si se poate incepe compilare. Procesul e continuu in cazul in care programul e suficient de scurt pentru a genera un cod obiect compilat in memorie. Daca programul e mai lung decit 8K, acest lucru nu va fi posibil si va trebui sa folositi banda atit ca dispozitiv de intrare cit si de iesire. Desi BLAST-ul permite acest lucru, calea de urmat e destul de anevoioasa. Daca obisnuiti sa folositi BLAST-ul pentru a compila programe lungi, aveti nevoie de un microdrive.

BLAST-area programelor de pe banda pe banda

In acest mod se folosesc doua benzi: o banda sursa si o banda obiect. Banda sursa contine programul d-voastra in forma speciala necesara compilarii (vezi aliniatul precedent), iar banda obiect e goala.

Sa presupunem ca ati selectat banda atit pentru intrare cit si pentru iesire. Cind tastati *C pentru a incepe compilarea, BLAST-ul va va instrui sa inserati banda sursa si sa apasati pe PLAY. Dupa putin timp calculatorul va fluiera (beep) si va va cere sa schimbati benzile. Banda sursa trebuie oprita in mai putin de 5 secunde de la beep. Daca nu veti proceda astfel, datele urmatoare vor fi pierdute.

Dupa un timp vi se va cere sa schimbati din nou benzile. Timpul de inlocuire a benzii obiect cu cea sursa nu e critic dar va sfatuim sa fiti rapid(a) pe perioada intregului proces pentru a micsora timpul total de incarcare a programului BLAST-at.

Pe durata compilarii vi se va cere sa schimbati benzile intre ele de un numar de ori ce depinde de lungimea programului

ce se compileaza. In final, compilatorul va afisa obisnuitul raport despre starea erorilor.

Cind compilati de pe banda pe banda, fisierul obiect este scris intr-un format oarecum nestandard. Cu toate acestea, in afara de faptul ca timpul de incarcare este mai lung decit in mod normal, nu vor exista diferente majore fata de un program obisnuit.

8. P-CODE SI COD MASINA

BLAST-ul poate compila programe atit in cod masina Z80 cit si intr-un pseudo cod masina mai compact, numit p-code. Argumentele pro si contra ale acestor doua tipuri de cod obiect pot fi rezumate in tabelul de mai jos:

	P-CODE	COD MASINA
VITEZA	Mai mare decit a BASIC-ului dar mai mica decit a codului masina.	Cea mai mare posibila.
DIMENSIUNE	Mai mica decit in BASIC mai mica decit in cod masina.	De regula mai mare decit BASIC-ul. Intotdeauna mai mare decit a p-code-ului.

Harta memoriei pentru un program BLAST-at este data in ANEXA 1. Se observa ca in afara de cod obiect si date, un program BLAST-at contine si un bloc de cod denumit Run Time System (RTS). RTS-ul este in principal o biblioteca de subrutine chemate din codul obiect pentru operatii cum ar fi inmultirea, impartirea si manipularea sirurilor. RTS-ul se include intotdeauna intr-un program BLAST-at si necesita in plus 5K din memorie. Din aceasta cauza, un program BLAST-at va fi intotdeauna mai lung decit 5K. Cu toate acestea, avind in vedere ca p-code-ul are cam 2/3 din lungimea echivalentului sau BASIC, programele lungi compilate in p-code pot deveni mai scurte decit originalul. Spre exemplu, un program de 3K dupa ce a fost BLAST-at in p-code va ocupa aproximativ 7K; 2/3*3 pentru p-code si 5K pentru RTS. Similar, un program BASIC de 30K va deveni dupa compilare de 25K. Evident undeva exista un punct de intilnire, la care cele doua marimi sint aproximativ egale. Acest lucru se intimpla la aproximativ 15K.

Evident, cele de mai sus au un caracter foarte general. Unele tipuri de programe genereaza mai putin p-code decit altele iar programele care contin numeroase comentarii vor suferi o reducere in lungime mult mai pronuntata decit cele care nu le contin.

Daca BLAST-ul este determinat sa genereze cod masina (in loc de p-code) programul va avea aproape in mod sigur o crestere in lungime. Un astfel de program va rula mai rapid dar acest lucru

nu e de folos daca nu va incapea in memorie. Din fericire insa, BLAST-ul poate fi determinat sa genereze cod masina pentru acele sectiuni din program in care viteza e critica, si p-code in rest. De multe ori, compilind o sectiune relativ scurta de BASIC in cod masina si restul in p-code vom avea aproape aceeasi viteza ca si cind programul ar fi fost compilat in intregime in cod masina.

Tipul de cod obiect generat de BLAST este specificat prin intermediul directivelor compilatorului (vezi capitolul referitor la acest subiect).

Pentru a instrui compilatorul sa genereze p-code vom scrie:

```
REM! P-CODE
```

iar pentru a-l determina sa genereze cod masina:

```
REM! MACHINE CODE
```

BLAST-ul genereaza implicit p-code.

9. BLAST SI CODUL MASINA AL UTILIZATORULUI

BLAST-area unui program care cheama subrutine in cod masina nu ar trebui sa prezinte probleme. BLAST-ul a fost proiectat pentru comatibilitate maxima cu BASIC-ul Spectrum-ului si aceasta compatibilitate se extinde la variabile si la formatul de inmagazinare a programului. In particular, au fost prevazute urmatoarele practici adoptate uzual de utilizatorii de Spectrum:

1) Un program BLAST-at poate sa rezerve spatiu pentru cod masina coborind RAMTOP-ul in mod obisnuit.

2) BLAST-ul inmagazineaza variabilele in exact acelasi mod ca si BASIC-ul Spectrum-ului. In consecinta codul masina care ar prelua si manipula variabile poate functiona sub BLAST.

3) Subrutinele cod masina care extind BASIC-ul interceptind subrutina de tratare a erorii din sistemul de operare al Spectrum-ului (sau prin alte metode) vor functiona. Explicatia acestui fapt surprinzator e urmatoarea: cind la compilare BLAST-ul intilneste o instructiune care apare incorecta sintactic, compilatorul va copia textul suparator in fisierul obiect precedat de un cod special de ESCAPE. La rulare, cind RTS-ul intilneste acest cod de escape, va chema interpretorul BASIC pentru a-l manipula. Daca textul e o eroare de sintaxa veritabila, interpretorul va raporta acest fapt si va reactiona in mod obisnuit. Daca insa textul e o extensie a BASIC-ului care a fost prevazuta, interpretorul se va comporta intocmai ca si cu programul ne-BLAST-at. (N.T. Desi apetisanta, aceasta facilitate nu functioneaza pentru cea mai raspindita extensie de BASIC, Beta Basic.)

Extensiile de BASIC prevazute de BLAST precum si directivele compilatorului se introduc sub forma unor

instructiuni REM speciale, recunoscute de BLAST la compilare. Este posibil ca in viitor alte programe comerciale sau chiar codul masina al utilizatorului sa foloseasca aceeasi tehnica pentru a introduce noi comenzi in BASIC. Din acest motiv s-a introdus in BLAST facilitatea de a permite instructiunilor REM sa fie trecute interpretorului daca incep cu caracterul escape %. Daca BLAST-ul intilneste o instructiune REM care incepe cu acest caracter, va genera un cod care va face ca instructiunea REM cu caracterul % indepartat sa fie trecuta interpretorului la rulare. BLAST-ul indica acest lucru cu mesajul:

COMMENT TRANSFERRED AT LINE xxxx

Este posibil ca unele practici obscure sa creeze probleme. Spre exemplu, codul masina continut in instructiuni REM, nu va functiona cu siguranta atunci cind programul e compilat deoarece aceasta metoda de inmagazinare a subrutinelor cod masina depinde de modul in care textul BSIC e organizat in memorie.

N.B. Din cauza unor posibile suprapuneri intre BLAST si codul masina al utilizatorului, compilatorul nu va permite programelor sa cheme subrutine Z80 cind compilarea se face in RAM. Astfel de programe trebuie compilate folosind banda sau microdrive-ul ca mediu de iesire.

10. FOLOSIREA VARIABILELOR INTREGI

Adeseori este posibil sa ajutati BLAST-ul la crearea unui cod mai eficient informindu-l despre orice variabila care va lua valori intregi intre -65535 si +65535. Majoritatea programelor contin multe astfel de variabile si merita osteneala sa informati compilatorul. Variabilele intregi se declara printr-o directiva compilator de forma:

REM! INT <lista de variabile>

de exemplu:

REM! INT I,J,K,A(10,5)

declara variabilele I, J, K si tabloul A(10,5) ca intregi. Declaratia care va initializa variabilele declarate la valoarea 0, trebuie sa apara la inceputul programului, inainte ca acestea sa fie utilizate. In exemplul de mai sus, declaratia A(10,5) serveste ca instructiune de tip DIM pentru respectivul tablou si va inlocui orice instructiune DIM existenta.

Retineti ca daca unei variabile care a fost declarata intreaga i se asigneaza o valoare neintreaga sau in afara domeniului de valori, rezultatele vor fi imprevizibile.

11. COMPATIBILITATEA CU BASIC-UL SPECTRUM-ULUI

BLAST-ul a fost proiectat pentru o compatibilitate maxima cu BASIC-ul Spectrum-ului. Aceasta compatibilitate se extinde nu numai la limbajul propriu-zis ci si la mediu de programare.

In BASIC, este posibila oprirea unui program in timp ce el ruleaza, se pot citi variabile, executa instructiuni s.a.m.d. Rularea poate fi apoi continuata sau reinceputa. Aceste actiuni sint posibile si sub BLAST cu o singura diferenta. Instructiunea CONTINUE nu va functiona cu un program BLAST-at.

12. PROTECTIA PROGRAMELOR BLAST-ATE

BLAST-ul prezinta un numar de mijloace care pot fi folosite pentru a preveni amestecul neautorizat in programele compilate.

1) AUTORUN (autorulare)
Daca directiva compilator

```
REM! AUTORUN
```

este inclusa la inceputul unui program BASIC, BLAST-ul va face ca fisierul compilat sa ruleze automat la rulare. Posibilitatea de AUTORUN face pirateria mult mai dificila si duce la un produs mai profesional.

2) P-CODE sigur

Majoritatea programelor disponibile comercial contin subrutine, scrise de regula in cod masina, care verifica daca nu "s-a umblat" prin program si ofera in plus si alte mijloace de protectie. Deoarece codul Z80 e bine cunoscut celor care se ocupa cu pirateria, aceste subrutine sint adesea gasite si dezactivate. P-code-ul generat de BLAST e un limbaj nedocumentat si astfel ofera un nivel de securitate mult mai inalt decit codul masina. Astfel, se recomanda ca subrutinele de protectie sa se scrie in BASIC si sa se compileze in p-code.

13. COPIEREA PROGRAMELOR BLAST-ATE

Programele compilate nu pot fi salvate direct utilizind SAVE. Comanda:

```
*S.
```

nu va resalva un program BLAST-at care a fost incarcat de pe banda sau microdrive. Daca doriti copierea unui program care a fost compilat pe unul dintre aceste dispozitive, procedati dupa cum urmeaza:

Salvarea pe banda

1) Incarcati programul BLAST-at in calculator.
2) Inserati urmatoarele linii:

```
15 LOAD "<prog>"
20 RANDOMIZE USR PEEK 23635+256*PEEK 23636+150
```

unde <prog> e noul nume de fisier.
3) Tastati:

```
SAVE "<prog>" LINE 15
```

Puteti verifica (VERIFY) daca codul a fost salvat corect in mod uzual, inlocuind SAVE cu VERIFY in liniile anterioare.

Salvarea pe microdrive

Metoda de salvare pe microdrive este exact aceeasi, cu exceptia ca parametrii de microdrive (cei obisnuiti) trebuie sa fie prezenti. De exemplu pentru a salva programul <prog>, adaugati liniile:

```
15 LOAD *"m";1;"<prog>"
20 RANDOMIZE USR PEEK 23635+256*PEEK 23636+150
```

si tastati:

```
SAVE *"m";1;"<prog>" LINE 15
```

Detalii despre forma exacta in care apar programele BLAST-ate in memorie, sint date in Anexa 1.

14. ERORI

1) Erori la compilare.

Desi prin editorul Spectrum-ului nu se introduce decit BASIC corect din punct de vedere sintactic, exista totusi moduri in care se poate impune BLAST-ului un cod sursa incorect. De exemplu, iesirea dintr-un generator de programe (program generator) poate contine erori; de asemenea programul poate fi oricind deteriorat pe banda sau microdrive. In plus exista posibilitatea introducerii de directive compilator eronate sau instructiuni ale unor extensii de BASIC. Din aceste motive, BLAST-ul verifica in mod riguros sintaxa textului care i se ofera.

Totusi, lucrurile nu sint chiar atit de simple. S-ar putea ca o instructiune care apare incorecta pentru BLAST la compilare, sa fie de fapt o extensie de BASIC perfect normala, posibil de genul celor oferite de anumite programe comerciale. Astfel de extensii sint perfect admise sub BLAST, problema fiind doar aceea ca la compilare BLAST-ul nu are suficiente informatii

pentru a le distinge de adevaratele erori.

Solutia adoptata de BLAST e urmatoarea: Ori de cite ori BLAST-ul intilneste o posibila eroare de sintaxa, afiseaza textul "suparator" impreuna cu o avertizare (WARNING). Apoi compilarea continua. Daca se dovedeste la rulare ca a fost de fapt o eroare, rularea se opreste cu mesajul:

NONSENSE IN BASIC

2) Erori la rulare.

La rulare, cu o singura exceptie, programele BLAST-ate vor raspunde cu erori cum ar fi NUMBER TOO BIG sau RETURN WITHOUT GOSUB in exact aceeasi maniera ca si interpretorul. Exceptia se refera la eroarea SUBSCRIPT WRONG. Pentru a evita verificarea continua a indicilor de tablou, la rulare, RTS-ul va ignora aceste erori. Daca indicii ies din domeniu, rezultatele vor fi imprevizibile.

15. DIRECTIVELE COMPILATORULUI

BLAST-ul prezinta anumite optiuni de compilare care pot fi apelate prin directivele compilatorului. Acestea apar in instructiuni REM speciale de forma:

REM! <directiva compilator>

Toate directivele compilatorului sint precedate de REM!. Semnul exclamari (!) permite o cale usoara de a spune daca BLAST-ul sa ignore sau nu textul care urmeaza REM-ului. Exista inca doua tipuri de instructiuni REM speciale recunoscute de BLAST:

REM%

face ca textul comentariului sa fie trecut interpretorului la rulare (vezi BLAST si codul masina al utilizatorului), si:

REM&

folosit ca predecesor pentru instructiunile BASIC permise in plus de BLAST. Acestea sint explicate in capitolul extensiilor de BASIC.

Optiunile compilatorului disponibile sub BLAST sint:

DIRECTIVA	INTELES
1) REM! PCODE	Face ca BLAST-ul sa genereze p-code pina se specifica altceva. Acest mod e implicit.
2) REM! MACHINE CODE	Face ca BLAST-ul sa genereza cod masina pina se specifica altceva.

3) REM! INT I,J,K — Declara variabilele I, J si K ca intregi (vezi capitolul referitor la variabilele intregi).

4) REM! AUTORUN — Face ca programul obiect sa ruleze automat dupa ce s-a incarcat. Aceasta trebuie sa fie prima linie din program.

SUMARUL COMENZILOR

Urmatoarele comenzi sint recunoscute de BLAST in starea sa initiala. Retineti ca o comanda NEW va sterge BLAST-ul complet din memorie. Daca doriti sa stergeti un program BASIC din memorie, folositi:

*N

COMPILE sintaxa *C

Compileaza un program BASIC folosind tipul precedent de periferic de intrare (vezi *I) pentru fisierul sursa si tipul precedent de periferic de iesire (vezi *O) pentru fisierul obiect. Modul implicit pentru intrari si iesiri este RAM.

RUN sintaxa *R

Ruleaza un program compilat. Comanda *R e folosita doar pentru a rula un program compilat din RAM in RAM. Daca a fost selectata banda sau microdrive-ul, programul obiect trebuie incarcat de pe aceste dispozitive si executat cu RUN.

SAVE sintaxa *S

Salveaza un program BLAST-at care a fost compilat in RAM. BLAST-ul va cere detalii despre dispozitiv, numar de microdrive si nume de fisier.

INPUT sintaxa *I

Stabileste dispozitivul de pe care BLAST-ul va citi codul sursa la compilare. BLAST-ul va afisa mesajul:

ACCEPT INPUT FROM: RAM, TAPE, MICRODRIVE

pentru care raspunsul este R, T sau M. Dispozitivul implicit e RAM-ul.

OUTPUT sintaxa *O

Stabileste dispozitivul pe care BLAST-ul va scrie codul obiect la compilare. BLAST-ul va afisa mesajul:

ACCEPT OUTPUT FROM: RAM, TAPE, MICRODRIVE

pentru care raspunsul este R, T sau M. Dispozitivul implicit e RAM-ul.

BACKUP sintaxa *B

Copiaza compilatorul BLAST pe microdrive.

QUIT sintaxa *Q

Paraseste BLAST-ul si elibereaza memoria utilizata de BLAST in favoarea altui cod.

16. EXTENSII DE BASIC

Prezentam in continuare o lista a extensiilor de BASIC recunoscute de BLAST. Deoarece Spectrum-ul nu va accepta text care apare incorect editorului BASIC, toate extensiile sint introduse ca instructiuni REM speciale care incep cu caracterul de escape &.

1) Dezactivarea tastei de BREAK.

sintaxa: REM& BREAK ON
REM& BREAK OFF

Aceste instructiuni activeaza si dezactiveaza tasta de BREAK. Tasta BREAK e activata implicit.

2) WHILE ... WEND

sintaxa: REM& WHILE <conditie>
REM& WEND

Aceasta face ca blocul de instructiuni terminat cu REM& WEND sa fie executat in mod repetat, pina cind (while) <conditie> e adevarata (diferita de 0). Daca <conditie> e falsa la inceput, instructiunile se ignora (se "depasesc").

3) REPEAT ... UNTIL

sintaxa: REM& REPEAT
REM& UNTIL <conditie>

Blocul de instructiuni dintre REM& REPEAT si REM& UNTIL se repeta pina cind (until) <conditie> ce urmeaza lui REM& UNTIL devine falsa (zero). Indiferent de valoarea <conditie>, instructiunile se executa cel putin o data.

4) DOKE

sintaxa: REM& DOKE <ne>,<ne>

Unde <ne> e o expresie numerica. Acesta este un POKE pe 16 biti.

Rezultatul celei de a doua expresii este depus in doua locatii de memorie la adresa data de prima expresie. Datele sint inmagazinate in format LO-HI. Ambele expresii trebuie sa fie in intervalul 0 la 65535.

5) DEEK

sintaxa: REM& DEEK <nv>,<ne>

Unde <nv> e o variabila numerica. Acesta este un PEEK pe 16 biti. Continutul celor doua locatii de memorie de la adresa data de cei de-al doilea parametru se asigneaza variabilei numerice din primul parametru. Deci <nv> devine egal cu PEEK (<ne>)+256*PEEK (<ne>+1).

6) CALL

sintaxa: REM& CALL <ne> |<lista de parametrii>|

Cheama subrutina cod masina de la adresa data de expresia numerica <ne>. Parametrii (optionali), separati prin virgula pot fi variabile numerice, in intervalul 0 la 65535, sau adresa unei variabile numerice exprimate &<nume_de_variabila>. Acesti parametrii sint inmagazinati, in ordine, primul fiind in adresa specificata de IX. De exemplu:

```
REM& CALL 50000,X,&Y
```

va avea ca rezultat o chemare a subrutinei cod masina de la adresa 50000. La intrarea in subrutina, intregul X va fi inmagazinat in (IX+0) si (IX+1) si adresa variabilei numerice in (IX+2) si (IX+3).

7) ELSE

sintaxa: REM& ELSE:<lista de instructiuni>

O extensie optionala la IF ... THEN, frecventa in numeroase BASIC-uri. De exemplu:

```
IF x=0 THEN GOSUB 100: REM& ELSE: GOSUB 200
```

va avea ca rezultat o chemare la linia 100 daca x este 0 si o chemare la linia 200 daca x este diferit de 0. Instructiunile IF ... THEN ... ELSE nu pot fi imbricate (nested) si orice ELSE trebuie sa apara pe aceeasi linie cu IF-ul aferent. RETINETI CA ELSE TREBUIE URMAT DE "DOUA PUNCTE".

8) Functii "multi line"

In BASIC-ul Spectrum-ului exista posibilitatea definirii si apelarii de functii cu parametrii. Principala limitare a functiilor definite de utilizator e faptul ca ele pot contine doar o singura instructiune care trebuie sa fie o expresie. BLAST extinde aceasta facilitate, permitind functii pe mai multe

linii. Acestea pot fi cel mai bine explicate printr-un exemplu. Sa presupunem ca dorim sa scriem o astfel de functie care sa aiba ca rezultat pe cel mai mare dintre cei doi parametrii de intrare. Vom proceda in felul urmator:

```
1000 REM& DEF M(A,B)
1010 IF A>B THEN LET M=A: REM& ELSE: LET M=B
1020 REM& END PROC
```

Functia poate fi chemata cu instructiunea:

```
100 REM& M(X,Y)
```

Linia 1000 defineste functia M. In linia 1010, M, numele de functie e tratat ca o variabila si e egalat cu cel mai mare dintre A si B. Linia 1020 termina procedura, si reda controlul instructiunii de dupa apelare. Parametrii din definitia de procedura, in acest caz X si Y sint locali pentru procedura. Aceasta inseamna ca parametrii sint necunoscuti in afara procedurii. In plus, daca X si Y sint definiti in afara procedurii sau in alta procedura, ei vor fi tratati ca variabile diferite. O procedura poate avea oricite linii ii sint necesare, dar trebuie terminata cu o instructiune REM& END PROC.

Numele si parametrii procedurilor pot fi formati dintr-o singura litera, optional urmata de semnul $. Functiile multi-line pot fi utilizate recursiv.

17. OPTIMIZARI

BLAST-ul nu traduce pur si simplu instructiunile BASIC in echivalentul lor in cod masina, ci aplica si o gama larga de tehnici de marire a vitezei si compactitatii programului obiect. Autorii BLAST-ului au aderat riguros la vechea maxima a producatorilor de compilatoare:

" Nu lasa pina la rulare ce poti face la compilare ! "

Aceasta se aplica in special la calculul indicilor de tablou. Daca un tablou, sa zicem A(10,10), a fost DIM-ensionat cu constante, BLAST-ul va sti adresa unui element dat A(1,2) (referit cu indici constanti) la compilare. In plus, chiar daca un indice e constant, sa zicem A(I,2), BLAST-ul poate sa perfectioneze codul, facind calcule de indici la compilare. In programe care contin numeroase accese la tablouri, se va observa o crestere de viteza semnificativa.

In evaluarea unei expresii, BLAST-ul va lucra in modul cel mai economic pentru calculul valorii expresiei, fara a retine si manipula valori intermediare inutile.

BLAST-ul poate recunoaste aparitia aceleiasi subexpresii si daca apare de mai multe ori intr-o expresie sau instructiune. In acest caz el va evalua expresia o singura data si apoi va folosi rezultatul calculat.

Daca cantitatea de memorie permite, BLAST-ul va crea spatiu

pentru variabile la compilare, in loc sa lase acest lucru pina la rulare. Spre deosebire de BASIC-ul Spectrum-ului, el va folosi toata memoria disponibila pentru a inmagazina variabile inainte de a fi fortat sa consume timp " colectind deseurile ".

In multe cazuri BLAST-ul poate mari viteza buclelor FOR-NEXT calculind numarul de ciclari inainte de a fi facute si folosind drept contor un registru al masinii.

BLAST-ul se foloseste pe larg de aritmetica intreaga. Intregii se pot manipula mult mai rapid decit numerele in virgula flotanta si se va obtine o crestere de viteza semnificativa folosindu-i ori de cite ori este posibil. Exista optiunea de declarare a unei variabile numerice ca intreg; in acest caz, la rulare, fiecare valoare e inmagazinata in format intreg.

18. SA OBTINEM CIT MAI MULT DE LA BLAST

Spre deosebire de interpretorul BASIC, BLAST-ul nu trebuie sa piarda timpul cautind prin program dupa numere de linie, instructiuni DATA si definitii de functii. El cunoaste adresa tuturor acestor obiecte si le poate referi direct.

Puteti ajuta BLAST-ul foarte mult urmind citeva principii simple care-i vor permite sa faca o cit mai mare parte din munca la compilare si nu la rulare. Veti gasi ca facilitatile oferite de toolkit va vor fi de mare folos.

In particular, evitati instructiunile de tipul:

GOTO <expresie>.

Acestea forteaza BLAST-ul sa intirzie calculul adresei de salt pina la rulare si sa rezerve memorie (pretioasa) pentru lista tuturor numerelor de linie si adresele lor la rulare. Incercati sa inlocuiti o astfel de <expresie> cu un numar de linie propriu zis a carui adresa se poate determina la compilare. Acelasi lucru se aplica la toate celelalte instructiuni ce folosesc numere de linie.

Desi e perfect legal, incercati sa nu intrati sau sa iesiti din bucle FOR-NEXT. Daca faceti aceasta, BLAST-ul nu va putea sa prevada consecintele si nu va aplica una dintre cele mai puternice optimizari de care dispune.

Folositi extensiile de BASIC prevazute. Acestea duc la un cod mult mai rapid decit echivalentul lor BASIC.

Incercati sa nu utilizati acelasi tablou de mai multe ori si FOLOSITI CONSTANTE pentru a defini dimensiunile tabloului.

Incercati sa folositi variabile dintr-o singura litera ori de cite ori este posibil, deoarece BLAST-ul trateaza aceste variabile in mod special.

19. TOOLKIT-UL BLAST-ULUI

BLAST-ul este livrat impreuna cu un toolkit conceput sa ajute la dezvoltarea programelor.

TOOLKIT-ul se gaseste pe fata a doua a casetei. Pentru a-l incarca, tastati:

LOAD "TOOLKIT" <ENTER>

TOOLKIT-ul va porni automat si va semna cu mesajul:

BLAST TOOLKIT (C) OCSS 1985

La fel ca si compilatorul, TOOLKIT-ul se incarca in partea superioara a RAM-ului si pozitioneaza RAMTOP-ul sub zona pe care o ocupa. TOOLKIT-ul reduce memoria disponibila cu aproximativ 2K.

Nota: TOOLKIT-ul nu poate sa coexiste in RAM cu compilatorul.

Facilitatile disponibile sint listate mai jos. Fiecare functie este executata introducind un asterisc (*) urmat de o comanda formata dintr-o singura litera si un numar de parametrii.

In continuare n, n1 si n2 sint intregi.

Portiunea de program asupra careia va avea efect o anumita comanda e specificata de un interval de numere de linie dupa cum urmeaza:

n1-n2 inseamna liniile de la n1 la n2 inclusiv
n1- inseamna de la linia n1 la sfirsitul programului
-n2 inseamna de la inceputul programului la linia n2 inclusiv

Daca un interval de numere de linii e omis, toolkit-ul va considera ca acest interval cuprinde tot programul

Un punct (.) poate fi utilizat pentru a indica linia curenta.

20. COMENZI PENTRU LINII

1) EDIT sintaxa: *En1

Linia n1 e afisata pentru editare.

2) COPY sintaxa: *Cn1,n2

Copiaza linia n1 peste linia n2 care se pierde.

3) DELETE sintaxa: *Dn1

Sterge linia n1.

4) MOVE sintaxa: *Mn1,n2

Muta linia n1 la linia n2, stergind linia n1.

21. COMENZI PENTRU BLOCURI

1) COPY sintaxa: *C<interval>,n

Copiaza liniile din intervalul specificat la linia n, scriind peste orice linie existenta. Liniile vor fi numerotate consecutiv, incepind de la linia n.

2) DELETE sintaxa: *D<interval>

Sterge intervalul specificat.

3) MOVE sintaxa: *m<interval>,n

Muta intervalul specificat la linia n, stergind liniile originale.

4) RENUMBER sintaxa: *R<interval>,n1,n2

Renumeroteaza intervalul specificat incepind cu n1, cu pasul n2. Valoarea implicita pentru n2 este 10.

22. FUNCTII PENTRU SIRURI

1) FIND sintaxa: *F<interval>, sir

Cauta in intervalul specificat prima aparitie a sirului. Daca sirul este omis, functia FIND va utiliza ultimul sir introdus.

2) SEARCH & REPLACE sintaxa: *S<interval>,sir1,sir2

Cauta in intervalul specificat dupa sir1 si il inlocuieste cu sir2. Noua linie va fi verificata sintactic. Daca apare o eroare, prima linie care contine eroarea se afiseaza. Toate liniile precedente in care s-a facut modificarea ramin modificate. Delimitatorii intre intervalul specificat si sir1 si intre sir1 si sir2 nu sint in mod necesar virgule; se poate utiliza orice caracter nenumeric.

23. ALTE COMENZI

1) TRACE sintaxa: *Tn

Ruleaza programul incepind de la linia n afisind numarul de linie a instructiunii in executie. Tasta de SPATIU poate fi utilizata pentru a incetinii executia si cea de ENTER pentru a o opri.

2) KILL sintaxa: *K

Sterge toate instructiunile REM din program care nu incep cu &, ! sau %.

3) WRITE sintaxa: *W<interval>,<nume de fisier>

Salveaza intervalul specificat pe caseta sub <nume de fisier> (max. 10 caractere).

4) BLAST SAVE sintaxa: *B<nume de fisier>

Salveaza programul intr-o forma care se preteaza compilarii de pe banda. Programul va fi salvat in blocuri, impreuna cu informatiile pe care BLAST-ul le necesita la compilare.

5) QUIT sintaxa: *Q

Paraseste toolkit-ul.

24. CUM S-A NASCUT BLAST-UL

OCSS e o fabrica de compilatoare. Compania construieste numeroase compilatoare pentru diferite limbaje si masini. Sintem adesea intrebati cum procedam pentru a produce un compilator ca BLAST-ul; cit timp lucram la el, ce limbaje folosim s.a.m.d. In acest capitol, incercam sa raspundem citorva din aceste intrebari.

In ce limbaj a fost scris BLAST-ul ?

Raspunsul este ca BLAST-ul nu a fost scris intr-un limbaj anume. Toate compilatoarele OCSS sint generate automat, folosind "instrumente" de generare automata. Un limbaj cum este BASIC-ul, e un limbaj ca oricare altul si astfel poate fi descris prin mijloace gramaticale. De exemplu, putem incepe prin a defini o propozitie in limba engleza scriind:

<propozitie>::=<subiect><verb><obiect>;

unde

::= insemna "se defineste ca" iar numele dintre parantezele unghiulare sint obiecte numite "non terminals" care vor fi definite ulterior. De exemplu, <verb> poate fi definit dupa cum urmeaza:

<verb>::="maninca"|"doarme"|"munceste"|etc.

unde bara verticala (|) inseamna "sau". In exact aceeasi maniera putem defini o instructiune BASIC:

```
<instructiune>::="LET"<variabila>=<expresie>
|"GOTO"<numar de linie>
|"PRINT"<lista de expersii> s.a.m.d.
```

Bineinteles, toti "non terminalii" trbuie definiti iar definirea completa a BASIC-ului de Spectrum e un fisier de sute de linii.

Pina acum am vazut cu se poate scrie SINTAXA unui limbaj. Pentru a face insa un compilator, trebuie sa gasim o cale pentru a exprima SEMANTICA, adica intelesul instructiunilor si actiunile pe care trebuie sa le ia computerul atunci cind recunoaste o instructiune. Notatia folosita nu o vom explica deoarece e mai complicata decit definirea sintaxei. Totusi ceea ce se intimpla in esenta e ca specificatiile referitoare la sintaxa impreuna cu cele privitoare la semantica sint oferite unui program numit METAPOD care le foloseste pentru a scrie codul compilatorului. METAPOD-ul, dezvoltat de OCSS, ruleaza pe un sistem UNIX multi-user si genereaza fisiere sursa scrise in limbajul C. Bineinteles, METAPOD-ul insusi este un fel de compilator; accepta un fisier sursa (definirea limbajului) si produce un fisier obiect (compilatorul), astfel putind fi utilizat pentru a se genera pe sine insusi. Asa a fost creat METAPOD-ul.

De ce folosim un generator de compilatoare ?

In special deoarece e mai ieftin. Scrierea "de mina" a unui compilator ia mult timp, iar produsul final e mult mai susceptibil sa contina erori. Un compilator generat, reflecta exact specificatiile cu care a fost alimentat generatorul de compilatoare si nu apare tentatia de a "taia coltul" in scrierea codului. Bineinteles, exista parti dintr-un compilator cum e BLAST-ul care trebuie scrise "de mina"; RTS-ul BLAST-ului a fost scris de mina pentru a creste viteza de rulare a programului. Intr-un compilator precum BLAST-ul aproximativ 70% din cod e generat automat.

Putem genera compilatoare pentru orice limbaj ?

Pentru majoritatea limbajelor, raspunsul e da. Bineinteles ca limbaje ca PASCAL, MODULA, C si BASIC fac parte din planurile noastre de viitor, desi putem produce si compilatoare dedicate pentru aplicatii cum ar fi controlul masinilor si robotilor, rapid si competitiv.

ANEXA 1. HARTA MEMORIEI PENTRU BLAST

```
PRAMT ----------------------------------------------------------
      USER DEFINED GRAPHICS (grafice definite de utilizator)
UDG ------------------------------------------------------------
      BLAST
RAMTOP ---------------------------------------------------------
                  GOSUB STACK (stiva de GOSUB)
                  MACHINE STACK (stiva masinii)
                    SPARE RAM (RAM liber)
STKEND ---------------------------------------------------------
             CALCULATOR STACK (stiva calculatorului)
STKBOT ---------------------------------------------------------
                  WORKSPACE (spatiu de lucru)
WORKSP ---------------------------------------------------------
                 EDITING AREA (zona de editare)
E-LINE ---------------------------------------------------------
               BASIC VARIABLES (variabile BASIC)
VARS -----------------------------------------------------------
                 BASIC PROGRAM (program BASIC)
PROG -----------------------------------------------------------
             CHANNEL INFORMATION (informatii de canal)
CHANS ----------------------------------------------------------
            MICRODRIVE MAPS (harti pentru microdrive)
     INTERFACE 1 SYSTEM VARIABLES (var. de sist. pt. interf. 1)
              SYSTEM VARIABLES (variabile de sistem)
             PRINTER BUFFER (tampon pt. imprimanta)
               ATTRIBUTES (atribute de culoare)
                DISPLAY FILE (memorie video)
       ROM
-----------------------------------------------------------------
```

HARTA MEMORIEI LA RULARE

```
PRAMT ----------------------------------------------------------
      USER DEFINED GRAPHICS (grafice definite de utilizator)
UDG, RAMTOP ----------------------------------------------------
                  GOSUB STACK (stiva de GOSUB)
                  MACHINE STACK (stiva masinii)
                    SPARE RAM (RAM liber)
STKEND ---------------------------------------------------------
             CALCULATOR STACK (stiva calculatorului)
STKBOT ---------------------------------------------------------
                  WORKSPACE (spatiu de lucru)
WORKSP ---------------------------------------------------------
                 EDITING AREA (zona de editare)
E-LINE ---------------------------------------------------------
              RUN TIME VARIABLES (variabile de rulare)
VARS -----------------------------------------------------------
```

```
                BLASTED PROGRAM (program BLAST-at)
PROG ------------------------------------------------------------
                CHANNEL INFORMATION (informatii de canal)
CHANS -----------------------------------------------------------
                MICRODRIVE MAPS (harti pt. microdrive)
              RUN TIME SYSTEM (sistem de rulare (RTS-ul))
-----------------------------------------------------------------
      INTERFACE 1 SYSTEM VARIABLES (var. de sist. pt. interf. 1)
                SYSTEM VARIABLES (variabile de sistem)
              PRINTER BUFFER (tampon pt. imprimanta)
                 ATTRIBUTES (atribute de culoare)
                  DISPLAY FILE (memorie video)
      ROM
-----------------------------------------------------------------
```

ERATA

Cantitatea de memorie disponibila pentru codul sursa este de 2K si nu 5K. Bineinteles, un program de orice marime poate fi compilat de pe banda pe banda sau de pe microdrive pe microdrive.

Retineti ca directiva *N sterge din memorie atit programul sursa cit si cel obiect.

Comanda *S poate fi folosita doar pentru a salva un program compilat pe caseta. Utilizatorii posesori de microdrive vor gasi ca e mai usor sa se compileze de pe microdrive pe microdrive.

Cind BLAST-ul compileaza pe microdrive se scriu doua sau trei fisiere, necesare pentru rularea programului. Numele lor este:

- numele de fisier al codului obiect (cel specificat).
- acelasi nume cu .P anexat
- un fisier optional .V (vezi capitolul de variabile salvate la sfirsitul manualului).

Programul compilat se executa incarcind primul dintre acestea (cu numele specificat de utilizator) si tastind RUN.

Capitolul referitor la copierea programelor BLAST-ate este incorect. In schimb, s-a furnizat un program pentru copierea programelor BLAST-ate pe fata a doua a casetei, imediat dupa toolkit. Pentru a incarca acest program tastati:

```
LOAD "COPIER"
```

si urmati instructiunile programului.

Retineti ca toate comenzile (*C, *N, etc.) pot fi tastate atit cu majuscule cit si cu minuscule.

Comanda *B (BACKUP) nu a fost implementata. In schimb s-a prevazut optiunea de copiere a BLAST-ului pe microdrive imediat dupa incarcarea programului.

Datorita penuriei de memorie, extensiile de BASIC nu au fost implementate. Drept compensatie, toolkit-ul a fost prevazut cu

cinci noi comenzi (vezi capitolul de informatii aditionale la sfirsitul manualului).

TOOLKIT

Comanda:

RANDOMIZE USR 60500

va reactiva toolkit ul dupa comanda *Q.

Comanda *D fara parametrii va sterge intreg programul lasind variabilele intacte.

In comenzile *F si *S este imposibila introducerea directa a cuvintelor cheie BASIC. Exista totusi un artificiu pentru rezolvarea problemei. Cind e nevoie de un cuvint cheie, se tasteaza intii THEN si apoi cuvintul cheie. Apoi se muta cursorul inapoi si se sterge THEN.

In comenzile SEARCH SI REPLACE:

i. Separatorul trbuie sa fie o virgula.
ii. Sirul care e inlocuit nu poate fi vid.
iii. Liniile modificate nu sint verificate sintactic.

Daca intervalul de numere de linii e omis dintr-o comanda a toolkit-ului,rebuie inclusa orice virgula care ar urma intervalului respectiv.

Comanda *B (BLAST SAVE) se foloseste cu caseta. BLAST-ul poate compila un program care a fost salvat pe microdrive in mod normal.

Comanda TRACE (*T) (de urmarire) nu are ca parametru un numar de linie. *T va activa optiunea TRACE care va ramine operativa pina la dezactivarea ei cu comanda *U.

Functia de renumerotare nu va functiona la mai mult de 643 de linii odata. Programele lungi pot fi renumerotate in doua sau mai multe etape.

PROBLEME

Foarte rar, BLAST-ul poate obiecta unei linii de BASIC care apare corecta la inspectare. Daca se intimpla asa ceva, BLAST-ul se va opri cu mesajul:

WARNING - HIT ANY KEY

De indata ce s-a apasat o tasta, BLAST-ul va face ca acea instructiune sa fie trecuta interpretorului la rulare si va continua compilarea. Desi acest eveniment va fi etichetat de BLAST ca un avertisment, el nu va afecta rularea programului final. De asemenea, BLAST-ul poate fi fortat sa treaca o linie interpretorului inserindu-i REM% la inceput (vezi manualul). Acest lucru poate fi util daca o anumita linie da probleme la compilare. Aceasta facilitate nu poate fi folosita pentru functiile definite de utilizator.

EXTENSII DE BASIC

Din pacate, din cauza lipsei de memorie, acestea nu au fost implementate. In schimb, ca o compensatie, s-a extins toolkit-ul cu inca cinci comenzi.

*V listeaza diferite variabile de sistem utile, inclusiv cantitatea de memorie ramasa libera.

*L listeaza toate variabilele BASIC, definite in mod curent, impreuna cu valoarea lor.

(*J <nr. de linie> se alatura liniei indicate ulterior.

*G si *A. In mod normal operatiile de cautare si gasire se vor opri si vor astepta ca utilizatorul sa apese o tasta dupa fiecare gasire sau substitutie. Pentru a dezactiva aceasta modalitate, folositi *G iar pentru a reveni la vechiul mod, folositi *A.

VARIABILE SALVATE (DOAR PENTRU MICRODRIVE)

Pentru a economisi spatiu, multe programe BASIC sint salvate impreuna cu o parte din variabile. BLAST-ul trateaza o astfel de eventualitate folosind directiva REM! AUTORUN dupa cum urmeaza:

Daca directiva AUTORUN apare la inceputul programului, BLAST-ul va crea un fisier separat (numit .V) care contine variabile salvate. Cind se incarca programul BLAST-at, acest fisier va fi adus in memorie automat. Daca directiva AUTORUN nu este inclusa, BLAST-ul va considera ca nu are de salvat variabile.

O alta metoda, aplicabila si pentru banda, si pentru microdrive, e urmatoarea:

Se incarca toolkit-ul si programul BASIC de compilat. Se foloseste comanda *D pentru a sterge programul si apoi se salveaza variabilele sub un nume adecvat, ca un program normal. Se insereaza o linie la inceputul programului de compilat pentru a face MERGE cu aceste variabile (N.T. vazute ca un program BASIC). Cind programul BLAST-at va rula, el va face merge cu aceste variabile.

ATENTIUNE !!!
NU PUTETI FOLOSI BLAST-UL FARA ACEASTA TABELA DE DECODIFICARE
NU O PIERDETI !!!

```
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

40 R Y G R Y G G R Y G W G G Y R R Y G Y G Y G W Y G G 40
39 Y G R Y W G Y G G R G Y R G Y G G R G R G Y G R G R 39
38 G Y G G Y R R G Y G Y R G Y W Y R Y R G R G R G Y G 38
37 R G Y G R Y G G R Y W Y R G Y G W G Y G Y G Y G G R 37
36 Y R G G Y G R Y G G G R G Y G R G R G W G Y G R G Y 36
35 G G Y Y G W Y W Y G R Y R G R G Y G Y R Y G W Y R Y 35
34 R Y G R R Y G Y R Y G Y Y G Y Y G Y R G G R Y G Y G 34
33 Y G R Y G R Y G G R G G G R G R Y G Y G Y G W Y R G 33
32 R G Y G Y G R Y R Y W G G Y R G W Y G Y R G Y R Y G 32
31 G Y W G R Y G R G G R Y G R Y R G Y G G R Y R G G R G 31

   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

30 Y R G Y G R Y R Y R G G R Y G Y R Y R Y W Y G Y G W 30
29 R Y R G W G G G Y R Y W Y G R G G R Y R G G W G R Y 29
28 G R G Y G G Y R G Y G Y G Y G R R Y G Y R R Y R G G 28
27 R Y G R Y R R Y R G G G W G R Y G R W G G Y R G R Y 27
26 Y R G G Y G Y R G Y R Y G R Y R G R G G Y W Y G Y R 26
25 G Y R Y R W G Y R G G R Y R G Y G G Y R G Y G R G Y 25
24 R G G R G Y G R Y W G R G G R G W Y G G R G Y G Y R 24
23 G G G Y R G R Y G G Y Y R Y G R G G W Y G R R Y G Y 23
22 Y W Y R Y G Y G Y G G Y G R Y R G Y R Y G R R G Y W Y G 22
21 G G R G R Y G Y W G R Y G Y G G Y R R Y G R R G R Y 21

   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

20 R G Y Y G Y R Y G R Y R Y W Y R R G Y G R Y G W Y R 20
19 Y G R G G W Y G Y G R G W G W G G W G W G G Y Y R R G 19
18 R Y G R G Y R G R R Y W Y G Y R G R Y G Y G R G R Y 18
17 Y W G Y R G R G Y G G Y G W G Y R Y G Y G Y G G Y R 17
16 G G G R G R Y R G R G R G Y G G G G G G G R G R Y R G 16
15 Y R G Y R Y W Y G R R Y R R Y R G R Y R Y R G W Y G 15
14 G Y R R G G Y G G R G W G W R G Y G W G Y G Y G G Y 14
13 G G Y G Y G Y G Y W Y G Y G W Y G R G Y G Y G Y G Y R G 13
12 R Y R G G R G G R Y R G W G W Y G R G G Y R G W G Y 12
11 Y R Y W G Y G R Y W Y G G Y G R Y G Y G R Y R Y G R 11

   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

10 R G G Y Y G R G R Y G R Y W Y G R G G W G W G R G Y 10
 9 G Y G R G R Y G G G Y R G Y G G Y R Y G W Y G Y Y G 9
 8 R G W Y R Y G R Y R R G R G R Y G Y G G Y G R G R W 8
 7 Y R G G Y W G Y G G Y G Y R Y Y W G Y R G R Y R Y G 7
 6 G Y R Y R G G G R G G R R G R G G Y R Y W Y G Y R G 6
 5 Y W G G G Y Y W G Y W G G R G Y R G Y G Y G Y G Y R 5
 4 G G Y R G R G G Y R G Y R G Y W Y G R Y R G W Y R G 4
 3 R Y R G Y G R Y R Y G R G Y G G R R G R G Y G R G Y 3
 2 Y G G G R G Y G G R Y G R G R Y R Y R Y R G Y R Y G 2
 1 W Y R Y G Y W Y G Y G R Y R G W Y G Y G G R Y G G W 1

   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Va multumim cà ati cumpárat manualul firmei noastre. Acest manual a fost editat si corectat cu toatà atentia si presupunem cà este corect (dar desigur perfectibil).
ALPHA Ltd. isi imbunátáteste permanent manualele editate si de aceea vá sintem recunoscátori pentru orice sesizare. Vá asteptám cu orice problemà la sediul firmei si la tel.961/12936

IMPORTANT !

Edituta "TM" pune la dispozitia tuturor celor interesati întreaga gamă de manuale în limba română pentru calculatoare compatibile ZX Spectrum (TIM S, TIM S Plus, COBRA, HC 85, CIP, Jet) editate de firma "ALPHA Ltd" S.R.L. :

1.01 Limbajul **BASIC** pe înțelesul tuturor în 12 lectii
1.02 Documentatie **GENS** și **MONS** (Asamblor-dezasamblor)
1.03 Documentatie limbaj **FORTH**
1.04 Documentatie **BETA BASIC 3.1** (Extensie BASIC)
1.05 Documentatie **BETA BASIC 3.1** (Rezumat)
1.06 Documentatie compilator **FORTRAN 77-S**
1.07 Documentatie editor de texte **TASWORD**
1.08 Documentatie compilator **BLAST**
1.09 Documentatie compilator **PASCAL HP4TM** (Rezumat)
1.10 Documentatie limbaj **C**
1.11 Memento timing cod mașină **Z80**
1.12 Documentatie **MEGA BASIC** (Extensie BASIC)
1.13 Documentatie **VU-CALC**
1.14 Manual **BASIC** avansati - conținînd și referiri la **COBRA**
1.15 Documentatie compilator **COLT**
1.16 Documentatie **MASTER - FILE** (sistem gestiune afaceri)
1.17 Documentatie limbaj **microPROLOG**
1.18 Documentatie limbaj **PASCAL HP4TM**
1.19 Documentatie sistem operare **CP/M** cu referire la calculatorul **COBRA**
1.20 Manual **ROM SPECTRUM** complet dezasamblat
1.21 Documentatie **LASER GENIUS** (pachet programe pentru lucrul în cod masină)
1.22 Cum să obtinem cît mai mult de la calculatorul nostru o carte cu programe si trucuri atît pentru începători cît si pentru avansati, în două variante:
a) Numai cartea, cu o parte din figuri în text
b) Cartea si o casetă demonstrativă, cu toate programele si figurile introduse
1.23 Construiți singuri 20 de montaje electronice interfatabile cu microcalculatorul Dvs

Societatea Comercială "TM" S.R.L.

- editează și tipărește
  - revista de "kit"-uri si informatii în electronică "RET"
  - suplimente, cataloage, cărti în domeniul tehnicii de calcul si electronicii

- produce "kit"-uri în electronică

- execută comenzi de producător pe bază de contract cu orice beneficiar

- comercializează - prin magazine proprii, retea proprie de distributie în țară, coletărie, mesagerie sau livrare directă cu mijloace auto:
  - toate publicatiile periodice sau neperiodice din domeniul de activitate, produse în țară;
  - componente active ale S.C. "MICROELECTRONICA" S.A. din Bucuresti: integrate MOS, integrate speciale, componente optoelectronice;
  - conectică produsă de "CONECT" S.A. Bucuresti: întrerupătoare, conectoare, mufe, cabluri, etc;
  - componente pasive realizate de "IPEE" Curtea de Arges: rezistente cu peliculă de carbon, peliculă metalică sau bobinate, coondensatoare ceramice, multistrat sau de trecere, potentiometre si semireglabile, trimeri, sonerii, relee de semnalizare, etc;
  - relee, temporizatoare si transformatoare de putere mică produse de "RELEE" Medias;
  - ferite diverse realizate de "Aferro" Bucuresti;
  - borne, izolatori plastic, sonde osciloscop, aparatură diversă produse de "ICE" Bucuresti;
  - Generatoare de mirá color, convertoare PAL, aparatură complexă antifurt realizate de "ROEL" Bucuresti;
  - casete cu jocuri si programe, diverse cărti de informatică realizate de "ALPHA Ltd" Timisoara;
  - piese de schimb radio-Tv;
  - componente diverse aflate în consignatie sau aduse din import;
  - diskette si consumabile pentru calculatoare;

Vă rugăm să ne contactati pe adresa 1900 Timisoara, Str. Miron Costin Nr. 2, Tel. 961/18576.

Tipografia "MIRTON"
1900 Timișoara Strada Samuil Micu nr.7
Telefon 96 - 18.35.25.